お客様各位

カタログ等資料中の旧社名の扱いについて

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

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

2010年4月1日
ルネサスエレクトロニクス株式会社

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

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

ご注意書き

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

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

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

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

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

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

RENESAS

ユーザーズ・マニュアル

RA78K0R Ver.1.20

アセンブラ・パッケージ

言語編

対象デバイス
78K0Rマイクロコントローラ

資料番号　U18546JJ1V0UM00（第1版）
発行年月　August 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。**

- 本資料に記載されている内容は2007年8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，RA78K0Rアセンブラ・パッケージ（以降，RA78K0Rとします）の各プログラムの基本機能と，ソース・プログラムの記述方法を正しく理解していただくことを目的として書かれています。

このマニュアルでは，RA78K0Rの各プログラムの操作方法に関する説明はいたしません。したがって，このマニュアルをご理解後，各プログラムの操作をされる際に必ずRA78K0R**アセンブラ・パッケージ　ユーザーズ・マニュアル　操作編**（U18547J）（以降，操作編とします）をお読みください。

このマニュアルでのRA78K0Rに関する記述は，Ver.1.20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（78K0R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されているユーザを対象としています。

【構　　成】

このマニュアルの構成は次のとおりです。

第1章　概　　説

RA78K0R全体の基本的な機能概要を説明します。

第2章　ソースの記述方法

ソースの記述方法，式と演算子などについて説明します。

第3章　疑似命令

アセンブラの疑似命令について，その書き方，使い方を使用例をまじえて説明します。

第4章　制御命令

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

第5章　マクロ

マクロの定義，参照，展開など，マクロ機能全体について説明します。マクロ疑似命令については，第3章**疑似命令**でも説明しています。

第6章　製品活用法

ソース・プログラム記述時のノウハウなどを紹介します。

付　録

予約語一覧表，疑似命令一覧表を掲載しています。

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（命令編）をご覧ください。また，アーキテクチャについては，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（ハードウエア編）をご覧ください。

【マ ク ロ】

アセンブラをはじめて使われる方は，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，第1章　概　　説を読み飛ばされても結構です。ただし，1.2　プログラム開発をはじめる前に，第2章　ソースの記述方法については必ずご一読ください。

アセンブラの疑似命令，制御命令について知りたい方は，第3章　疑似命令，第4章　制御命令をお読みください。

各命令の書式，機能，用途を使用例を用いて説明しています。

【凡　　例】

このマニュアル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。**
{ }	**；かっこの中の一つを選択します。**
「 」	**；「 」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。**
（ ）	**；（ ）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（おもにタイトル）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個以上の空白またはタブを表します。**
／	**；文字の区切りを表します。**
～	**；連続性を表します。**
太文字	**；重要箇所または参照箇所を表します。**

【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
CC78K0R Ver.2.00　Cコンパイラ	操作編	U18549J	U18549E
	言語編	U18548J	U18548E
RA78K0R Ver.1.20　アセンブラ・パッケージ	操作編	U18547J	U18547E
	言語編	このマニュアル	U18546E
SM+　システム・シミュレータ	操作編	U18601J	U18601E
PM+ Ver.6.30		U18416J	U18416E
ID78K0R-QB Ver.3.20　統合デバッガ	操作編	U17839J	U17839E

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

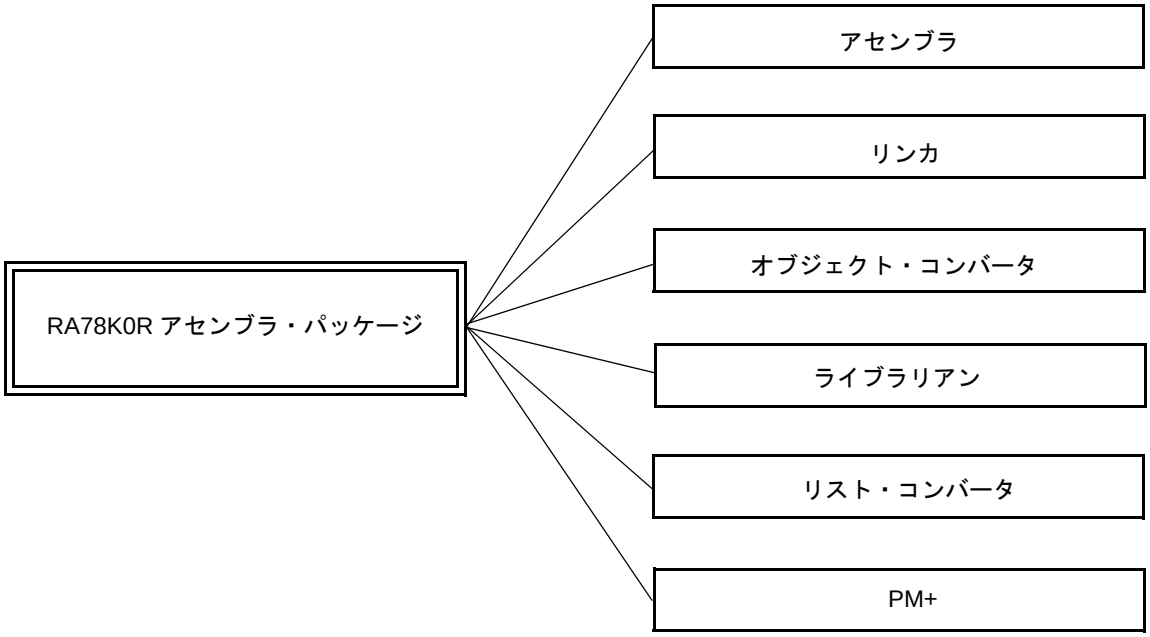
この章では，マイクロコンピュータの開発における RA78K0R の役割など，RA78K0R の特徴について説明します。

1.1 概要

RA78K0R アセンブラ・パッケージ（以下，RA78K0R）は，78K0R のアセンブリ言語で記述されたソースを機械語に変換する言語処理プログラムの総称です。

RA78K0R には，アセンブラ，リンカ，オブジェクト・コンバータ，ライブラリアン，リスト・コンバータといった 5 個のプログラムのほかに，エディット，コンパイル／アセンブル，リンクからデバッグまでの一連の操作を Windows® 上で簡単に行うことを可能にする PM+ が標準添付されています。

図 1-1 RA78K0R アセンブラ・パッケージ



1.1.1 アセンブラ

(1) アセンブリ言語と機械語

アセンブリ言語は，マイクロコンピュータ用の最も基本的なプログラミング言語です。

マイクロコンピュータに仕事を行わせる際には，プログラムやデータが必要となります。これらの情報をユーザがプログラミングし，マイクロコンピュータのメモリに書き込みます。

なお，マイクロコンピュータが扱うことのできるプログラムやデータは 2 進数の集まりであり，これを機械語と呼びます。

機械語でプログラムを作ることは，ユーザにとって覚えにくく，また誤りを起こしやすいものです。そこで，機械語の意味をユーザにとって理解しやすい英語の略記号で表し，この記号を使ってプログラムを作成する方法があります。この記号によるプログラムの基本的な言語体系をアセンブリ言語と呼びます。

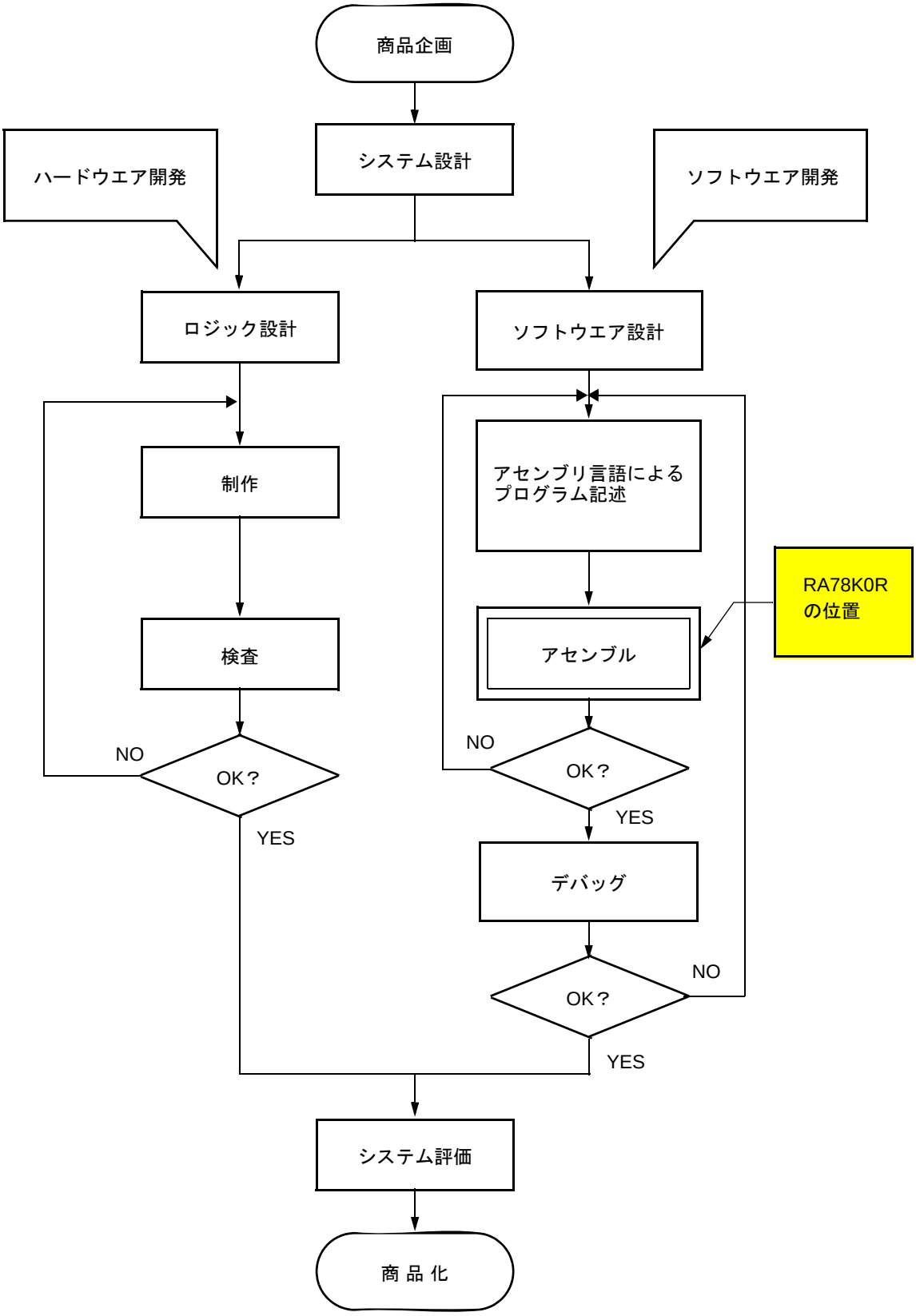
ただし，マイクロコンピュータが扱うことのできるプログラミング言語は機械語に限定されるため，アセンブリ言語で作成したプログラムを機械語に翻訳するプログラムが必要となります。これをアセンブラと呼びます。

図 1-2 アセンブラの流れ

1.1.2 マイクロコンピュータ応用製品の開発と RA78K0R の役割

製品開発における“アセンブル”の位置づけを，次に示します。

図 1-3 マイクロコンピュータ応用製品の開発工程



1.1.3 リロケータブル・アセンブラ

アセンブラが変換した機械語は，マイクロコンピュータのメモリに書き込まれて使用されます。このとき，変換された機械語がメモリのどこに書き込まれるかが，決定されていなければなりません。

したがって，アセンブラの変換する機械語には，「各機械語がメモリのどのアドレスに配置されるべきか」という情報が付加されています。

機械語を配置するアドレスの決定方法により，アセンブラは，“アブソリュート・アセンブラ”と“リロケータブル・アセンブラ”に大別されます。

- アブソリュート・アセンブラ

  アセンブリ言語から変換した機械語は，絶対的なアドレスに配置されます。

- リロケータブル・アセンブラ

  アセンブリ言語から変換した機械語には，一時的なアドレスが与えられます。

  なお，リンカにより，絶対的なアドレスが配置されます。

アブソリュート・アセンブラで 1 つのプログラムを作成する際には，原則として一度にプログラミングしなければなりませんでした。しかし，大きなプログラムを 1 つのまとまりとして作成した場合，プログラムが複雑になり，また保守する際にもプログラムの解析が困難になります。

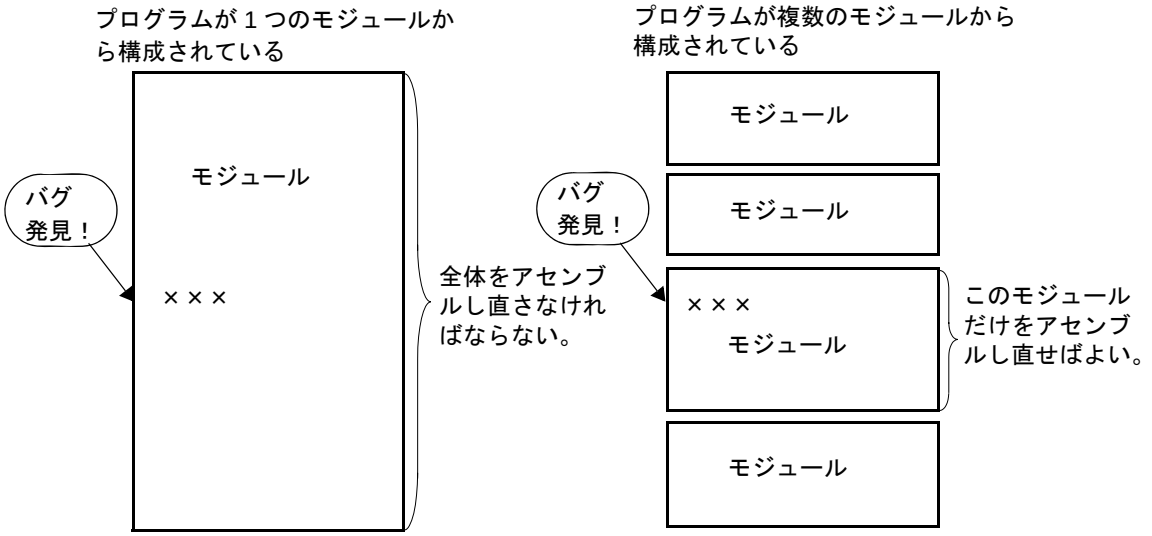
そこで，プログラムを 1 つ 1 つの機能単位ごとにいくつかのサブ・プログラム（モジュール）に分割して，プログラム開発を行います。これをモジュラ・プログラミングと呼びます。

リロケータブル・アセンブラは，モジュラ・プログラミングに適したアセンブラであり，モジュラ・プログラミングを行うことにより，次の利点があげられます。

(1) 開発効率が上がる

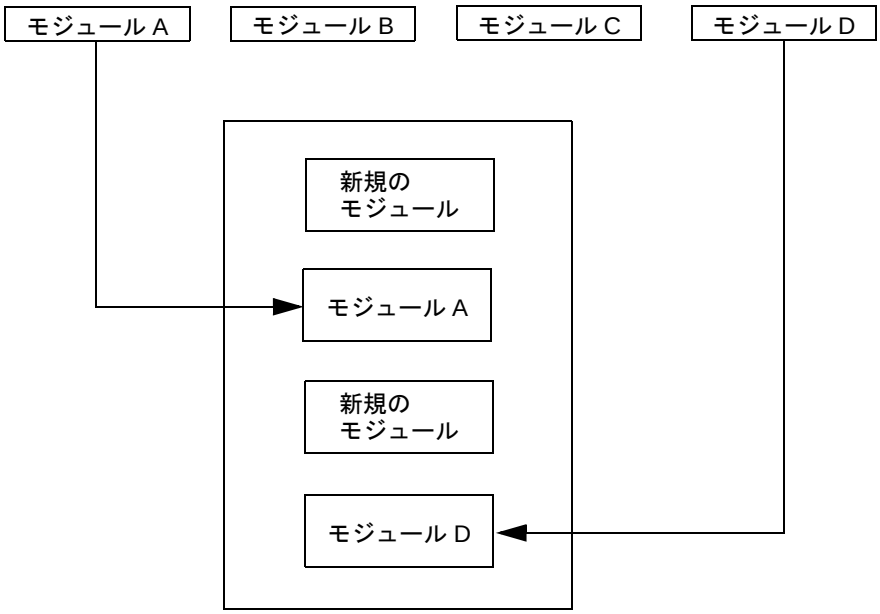
大きなプログラムを一度にプログラミングすることは困難です。このような場合，プログラムを 1 つ 1 つの機能単位ごとにモジュール分割すれば，複数の人数でプログラムの並行開発ができ，開発効率が上がります。

また，プログラム中にバグが発見された場合，一部の修正を行うために全プログラムをアセンブルすることなく，修正が必要なモジュールだけアセンブルし直すことができ，デバッグ時間を短くできます。



(2) 資源の活用ができる

以前に作成された信頼性，汎用性の高いモジュールを，ほかのプログラムの開発に再利用できます。このような汎用性の高いモジュールを蓄積することにより，新規にプログラム開発する部分を少なくすることができます。



新しいプログラム

1.2 プログラム開発をはじめる前に

実際にプログラム開発をはじめる前に，次のことを参照してください。

1.2.1 RA78K0R の最大値

(1) アセンブラの最大値

項目	最大値
シンボル数（ローカル + パブリック）	65,535 個
クロスリファレンス・リスト出力可能なシンボル数	65,534 個[注 1]
1 マクロ参照のマクロ・ボディ最大サイズ	1M バイト
全マクロ・ボディ合計のサイズ	10M バイト
1 ファイル中のセグメント数	256 個
1 ファイル中のマクロ，インクルード指定	10,000 個
1 インクルード・ファイル中のマクロ，インクルード指定	10,000 個
リロケーション情報[注 2]	65,535 個
行番号情報	65,535 個
1 ファイル中の BR/CALL 疑似命令数	32,767 個
ソース 1 行の文字数	2,048 文字[注 3]
シンボル長	256 文字
スイッチ名の定義数[注 4]	1,000 個
スイッチ名の文字長[注 4]	31 文字
セグメント名の文字長	8 文字
モジュール名（NAME 疑似命令）の文字長	256 文字
MACRO 疑似命令の仮パラメータ数	16 個
マクロ参照の実パラメータ数	16 個
IRP 疑似命令の実パラメータ数	16 個
マクロ・ボディ内のローカル・シンボル数	64 個
マクロ展開のローカル・シンボル数合計	65,535 個
マクロ（マクロ参照，REPT 疑似命令，IRP 疑似命令）のネスト数	8 レベル
TITLE 制御命令，-lh オプションで指定可能な文字数	60 文字[注 5]
SUBTITLE 制御命令で指定可能な文字数	72 文字
1 ファイル中のインクルード・ファイルのネスト数	8 レベル
条件アセンブルのネスト数	8 レベル

項目	最大値
-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
-d オプションで定義可能なシンボル数	30 個

注 1 モジュール名，セクション名の個数を引いた数です。
メモリを使用します。メモリがなければファイルを使用します。

注 2 アセンブラでシンボル値が解決できない場合に，リンカに渡す情報のことです。
たとえば，MOV 命令で外部参照シンボルを参照した場合，リロケーション情報が 2 個，.rel ファイルに生成されます。

注 3 復帰 / 改行コードを含みます。1 行に 2049 文字以上記述した場合，ワーニング・メッセージが出力され，2049 文字以降は無視されます。

注 4 スイッチ名は，SET/RESET 疑似命令で真 / 偽に設定され，$IF などで使用されるものです。

注 5 アセンブル・リスト・ファイルの 1 行の文字数指定（X とします）が 119 文字以下の場合，“X - 60” 文字以内とします。

(2) リンカの最大値

項目	最大値
シンボル数（ローカル + パブリック）	65,535 個
同一セグメントの行番号情報	65,535 個
セグメント数	65,535 個[注]
入力モジュール	1,024 個
メモリ領域名の文字長	256 文字
メモリ領域数	100 個[注]
-b オプションで指定可能なライブラリ・ファイル数	64 個
-i オプションで指定可能なライブラリ・ファイル・パス数	64 個

注 デフォルトで定義されているものを含みます。

1.3 RA78K0R の特徴

RA78K0R は，次の特徴的な機能を備えています。

(1) マクロ機能

ソース中で同じ命令群を何回も記述する場合，その一連の命令群を，1 つのマクロ名に対応させてマクロ定義をすることができます。

マクロ機能を利用することにより，コーディングの効率を上げることができます。

(2) 分岐命令の最適化機能

分岐命令自動選択疑似命令として，「BR」，「CALL」を備えています。

メモリ効率のよいプログラムを生成するためには，分岐命令の分岐先範囲に応じたバイトの分岐命令を記述する必要があります。しかし，分岐先範囲をいちいち意識して分岐命令を記述することは面倒です。BR 疑似命令，または CALL 疑似命令を記述することにより，アセンブラが分岐先範囲に応じて適切な分岐命令のコードを生成します。これを分岐命令の最適化機能と呼びます。

(3) 条件付きアセンブル機能

ソースの一部分を条件によりアセンブルする／しないに設定することができます。

ソース中にデバッグ文などを記述した場合，デバッグ文を機械語に変換する／しないを条件付きアセンブルのスイッチ設定により選択することができます。デバッグ文が必要なくなった場合でも，ソースに大幅な変更を加えることなく，アセンブルを行うことができます。

(4) 78K0 の互換マクロ機能

78K0 用アセンブラで作成したアセンブラ・ソース・ファイルをアセンブル可能にします。

下記の 78K0R で使用できない 78K0 の命令をソースの記述を変更せずにアセンブルしたい場合，-compati オプションを指定します。

78K0R で使用できない 78K0 の命令 ： DIVUW/ROR4/ROL4/ADJBA/ADJBS/CALLF/DBNZ

第 2 章　ソースの記述方法

この章では，ソースの記述方法，式と演算子などについて説明します。

2.1　基本構成

1 つのソースをいくつかのモジュールに分割して記述したとき，アセンブラの入力単位となる各モジュールをソース・モジュールと呼びます（プログラムが 1 つのモジュールからなるとき，ソースとソース・モジュールは同じ意味を持ちます）。

アセンブラの入力単位となるソース・モジュールは，大まかには次の 3 つの構成部分からなります。

- モジュール・ヘッダ（Module Header）
- モジュール・ボディ（Module Body）
- モジュール・テイル（Module Tail）

図 2-1　ソース・モジュールの構成

2.1.1 モジュール・ヘッダ

次に，モジュール・ヘッダに記述できる制御命令を示します。これらの制御命令は，モジュール・ヘッダ以外には記述できません。

また，モジュール・ヘッダは省略することが可能です。

表 2-1 モジュール・ヘッダに記述できるもの

記述できるもの	説明	説明箇所
アセンブラ・オプションと同様の機能を持つ制御命令	- PROCESSOR - XREF/NOXREF - DEBUG/NODEBUG/DEBUGA/NODEBUGA - TITLE - SYMLIST/NOSYMLIST - FORMFEED/NOFORMFEED - WIDTH - LENGTH - TAB - KANJICODE	第 4 章 制御命令
C コンパイラなどの上位プログラムが出力する特別な制御命令	- TOL_INF - DGS - DGL	

2.1.2 モジュール・ボディ

モジュール・ボディには，次のものは記述できません。

- アセンブラ・オプションと同様の機能を持つ制御命令

上記以外のすべての疑似命令，制御命令，インストラクションは，モジュール・ボディに記述可能です。

また，モジュール・ボディは，セグメントと呼ぶ単位に分割して記述します。

セグメントは，それぞれ対応する疑似命令で定義します。

- コード・セグメント

  CSEG 疑似命令で定義します。

- データ・セグメント

  DSEG 疑似命令で定義します。

- ビット・セグメント

  BSEG 疑似命令で定義します。

- アブソリュート・セグメント

  CSEG，DSEG，BSEG 疑似命令で，再配置属性に配置アドレス（AT 配置アドレス）を指示してセグメントを定義します。また，ORG 疑似命令で定義することもできます。

モジュール・ボディは，どのようなセグメントの組み合わせで構成してもかまいません。

ただし，データ・セグメントやビット・セグメントの定義は，コード・セグメントの定義よりも前で行うようにしてください。

2.1.3 モジュール・テイル

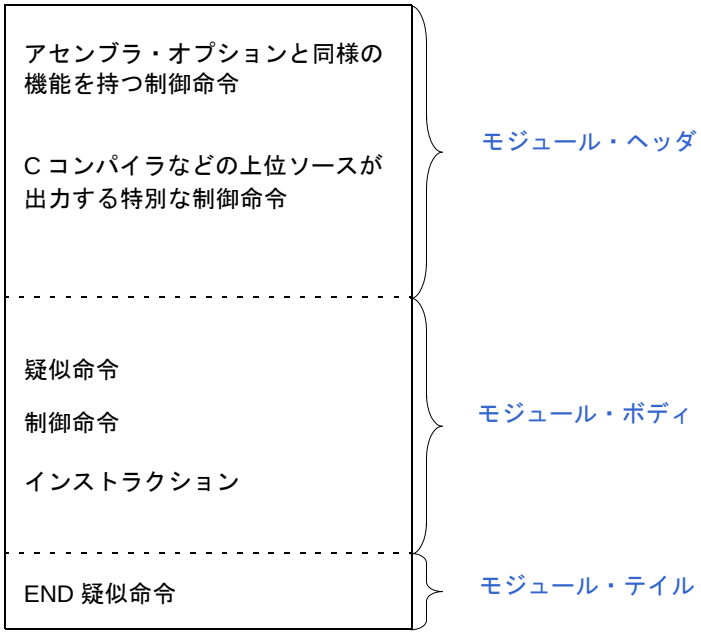
モジュール・テイルは，ソース・モジュールの終了を示すもので，END 疑似命令を記述します。

END 疑似命令のあとにコメント，空白，タブ，改行コード以外のものを記述すると，ワーニング・メッセージが出力され，それらは無視されます。

2.1.4 ソースの全体構成

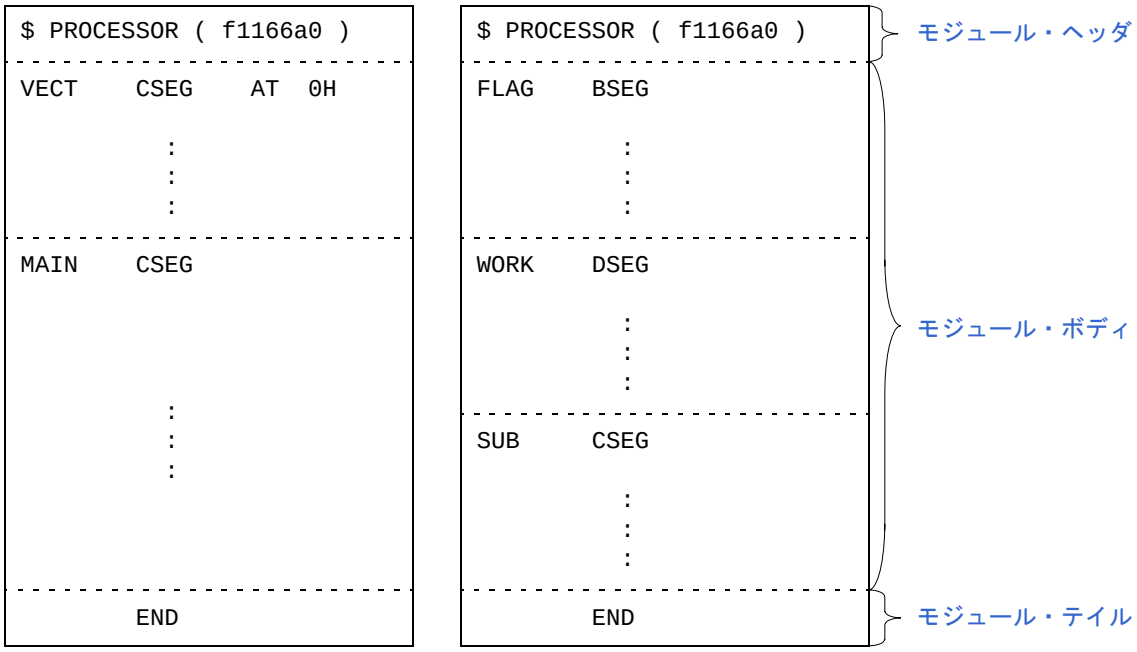
次に，ソース・モジュール（ソース）の全体構成を示します。

図 2-2 ソース・モジュールの全体構成



また，次に，ソース・モジュールの構成例を簡単に示します。

図 2-3 ソース・モジュールの構成例

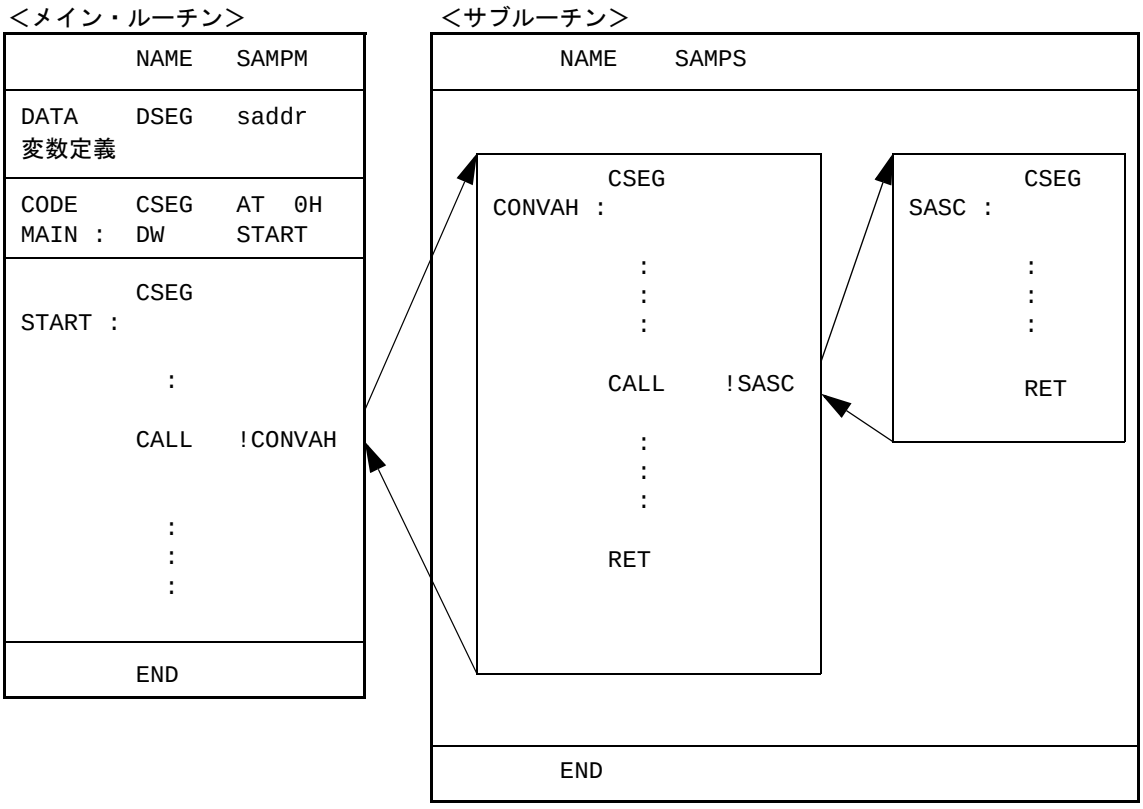


2.1.5 記述例

ここで，ソース・モジュール（ソース）の記述例をサンプル・プログラムとして示します。

次に，サンプル・プログラムの構成を簡単に示します。

図 2-4 サンプル・プログラムの構成



＜メイン・ルーチン＞

```
        NAME    SAMPM                       ; (1)
; ****************************************
;       HEX -> ASCII Conversion Program
;               main-routine
; ****************************************

PUBLIC  MAIN , START                        ; (2)
EXTRN   CONVAH                              ; (3)
EXTRN   _@STBEG                             ; (4)   ←エラー

DATA    DSEG    AT      0FFE20H             ; (5)
HDTSA : DS      1
STASC : DS      2

CODE    CSEG    AT      0H                  ; (6)
MAIN :  DW      START

        CSEG                                ; (7)
START :
        ; chip initialize
        MOVW    SP , #_@STBEG

        MOV     HDTSA , #1AH
        MOVW    HL , #LOWW ( HDTSA )        ; set hex 2-code data in HL registor

        CALL    !CONVAH                     ; convert ASCII <- HEX
                                            ; output BC-register <- ASCII code
        MOVW    DE , #LOWW ( STASC )        ; set DE <- store ASCII code table
        MOV     A , B
        MOV     [ DE ] , A
        INCW    DE
        MOV     A , C
        MOV     [ DE ] , A
        BR      $$

        END                                 ; (8)
```

(1) モジュール名を宣言

(2) ほかのモジュールから参照されるシンボルを，外部定義シンボルとして宣言

(3) ほかのモジュールで定義されているシンボルを，外部参照シンボルとして宣言

(4) リンカの -s オプションで生成されるスタック解決用シンボルを，外部参照シンボルとして宣言（リンクする際に -s オプションを指定しないとエラーになる）

(5) データ・セグメントの開始を宣言（saddr に配置する）

(6) コード・セグメントの開始を宣言（アブソリュート・セグメントとして 0H 番地から配置する）

(7) コード・セグメントの開始を宣言（アブソリュート・セグメントの終了を意味する）

(8) モジュールの終了を宣言

＜サブルーチン＞

```
        NAME    SAMPS           ; (1)
; ************************************************************
;       HEX -> ASCII Conversion Program
;       sub-routine
;
;   input condition     : ( HL )        <- hex 2 code
;   output condition    : BC-register   <- ASCII 2 code
; ************************************************************

PUBLIC  CONVAH                  ; (2)

        CSEG                    ; (3)
CONVAH :
        XOR     A , A
        ROL4    [ HL ]          ; hex upper code load (4)
        CALL    !SASC
        MOV     B , A           ; store result

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

; ************************************************************
;       subroutine   convert ASCII code
;
;   input       Acc ( lower 4bits )     <- hex code
;   output      Acc                     <- ASCII code
; ************************************************************

SASC :
        CMP     A , #0AH        ; check hex code > 9
        BC      $SASC1
        ADD     A , #07H        ; bias ( +7H )
SASC1 :
        ADD     A , #30H        ; bias ( +30H )
        RET

        END                     ; (5)
```

(1) モジュール名を宣言

(2) ほかのモジュールから参照されるシンボルを，外部定義シンボルとして宣言

(3) コード・セグメントの開始を宣言

(4) ROL4 命令は，78K0R ではサポートしていない 78K0 用の命令であるため，アセンブラ・オプション（-compati）の指定が必要です。

アセンブラ・オプション（-compati）については，「RA78K0R 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。

(5) モジュールの終了を宣言

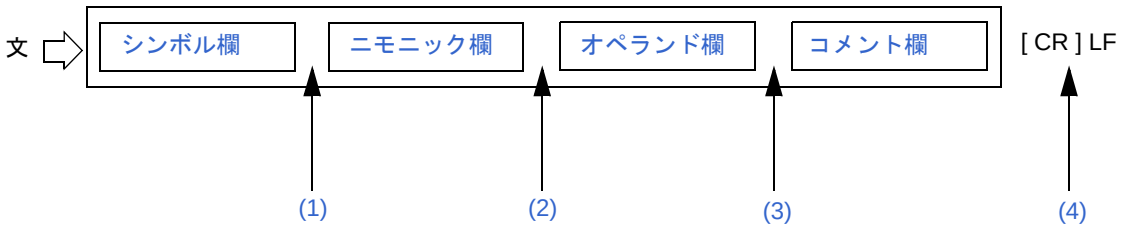
2.2 記述方法

2.2.1 構成

ソースは，文（ステートメント）で構成します。

1 つの文は，次に示す 4 つのフィールドで構成します。

図 2-5 文の構成フィールド



(1) シンボル欄とニモニック欄は，コロン（ : ），または 1 つ以上の空白（または TAB）で区切ります（コロンで区切るか空白で区切るかは，ニモニック欄で記述する命令により異なります）。

(2) ニモニック欄とオペランド欄は，1 つ以上の空白（または TAB）で区切ります。ニモニック欄に記述する命令によっては，オペランド欄が必要ない場合もあります。

(3) コメント欄を記述するときは，コメント欄の前にセミコロン（ ; ）を記述します。

(4) 各行の終わりは，LF で区切ります（LF の直前に CR が 1 つ存在してもかまいません）。

- 1 つの文は 1 行以内に記述します。1 行には最大 2048 文字（CR/LF を含む）まで記述することができます。このとき，TAB，および単独の CR は，それぞれ 1 文字として数えます。2049 文字以上記述した場合には，ワーニング・メッセージが出力され，2049 文字以降は無視されます。ただし，アセンブル・リストには 2049 文字以降も出力されます。
- 単独の CR は，アセンブル・リストには出力されません。
- 次のような行の記述が可能です。

(1) 空行（文の記述のない行）

(2) シンボル欄のみの行

(3) コメント欄のみの行

2.2.2 文字セット

ソース・ファイル中に記述可能な文字は，次の 3 つから構成されます。

- 言語文字
- 文字データ
- 注釈（コメント）用文字

(1) 言語文字

ソース上で命令を記述するために使用する文字です。

言語文字の文字セットには，英数字，および特殊文字があります。

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

表 2-3 特殊文字

文字	名称	主な用途	
?	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	各欄の区切り記号	区切り記号
HT (09H)	タブ・コード	空白相当文字	
,	コンマ	オペランドの区切り文字	
:	コロン	ラベル区切り記号	
;	セミコロン	コメント欄開始記号	
CR (0DH)	復帰コード	1 行の最終記号（アセンブラでは無視）	
LF (0AH)	改行コード	1 行の最終記号	
+	プラス	加算演算子，または正符号	アセンブラ演算子
-	マイナス	減算演算子，または負符号	
*	アスタリスク	乗算演算子	
/	スラッシュ	除算演算子	
.	ピリオド	ビット位置指定子	
( )	左右かっこ	演算順序	
< >	不等号	比較演算子	
=	等号	比較演算子	

文字	名称	主な用途
’	引用符	- 文字定数の開始・終了記号 - マクロのパラメータを 1 つにまとめる記号
$	ドル記号	- ロケーション・カウンタの値 - アセンブラ・オプションに相当する制御命令の開始記号 - 相対アドレッシング指定記号
&	アンパーサンド	コンカティネート記号（マクロ・ボディ内で使用）
#	シャープ	イミーディエト・アドレッシング指定記号
!	感嘆符	絶対アドレッシング指定記号
[ ]	大かっこ	インダイレクト・アドレッシング指定記号

(2) 文字データ

文字データは，文字列定数，文字列，および制御命令部（TITLE，SUBTITLE，INCLUDE）を記述するために使用する文字です。

注意 1 00H を除くすべての文字（漢字かなを含みます。ただし，OS によってコードは異なります）が記述可能です。00H を記述するとエラーとなり，それ以降，引用符（ ’ ）で閉じるまで無視されます。

注意 2 不正文字が入力された場合，アセンブラは，不正文字を“！”に置き換えてアセンブル・リストに出力します（CR（0DH）は，アセンブル・リストに出力されません）。

注意 3 Windows では，1AH をファイルの末尾と判断するため，入力データとはなりません。

(3) 注釈（コメント）用文字

コメントを記述するために使用する文字です。

注意 文字データの文字セットと同一です。ただし，00H が入力されてもエラーは出力されません。アセンブル・リストには“！”に置き換えて出力されます。

2.2.3 シンボル欄

シンボル欄には，シンボルを記述します。シンボルとは，数値データやアドレスなどに付けた名前のことです。

シンボルを使用することにより，ソースの内容がわかりやすくなります。

(1) シンボルの種類

シンボルは，その使用目的，定義方法によって，次に示す種類に分けられます。

シンボルの種類	使用目的	定義方法
ネーム	ソース中で，数値データやアドレスとして使用	EQU，SET，DBIT 疑似命令等のシンボル欄に記述します。
ラベル	ソース中で，アドレス・データとして使用	シンボルのあとにコロン（：）を付けることにより定義します。
外部参照名	あるモジュールで定義されたシンボルを，ほかのモジュールで参照するときに使用	EXTRN，EXTBIT 疑似命令のオペランド欄に記述します。
セグメント名	リンク時に使用	CSEG，DSEG，BSEG，ORG 疑似命令のシンボル欄に定義します。
モジュール名	シンボリック・デバッグ時に使用	NAME 疑似命令のオペランド欄に記述します。
マクロ名	ソース中で，マクロ参照時に使用	MACRO 疑似命令のシンボル欄に記述します。

注意 シンボル欄に記述可能なシンボルは，ネーム，ラベル，セグメント名，マクロ名の 4 種類です。

(2) シンボル記述上の規則

シンボルは，次の規則に基づいて記述します。

- シンボルは，英数字，および英字相当文字（?，@，_ ）で構成します。

  ただし，先頭文字に数字（0 - 9）は使用できません。

- シンボルの長さは，1 - 256 文字です。

  認識最大文字数を越えた文字は無視されます。

- シンボルとして，予約語は使用できません。

  予約語については，表 A-2 を参照してください。

- 同一シンボルを二度以上定義することはできません。

  ただし，SET 疑似命令で定義したネームは，SET 疑似命令で再定義することができます。

- アセンブラは，シンボルの大文字／小文字を区別します。

- シンボル欄にラベルを記述する場合は，ラベルの直後にコロン（ : ）を記述します。

＜正しいシンボルの例＞

```
CODE01  CSEG            ;  “CODE01”はセグメント名
VAR01   EQU     10H     ;  “VAR01”はネーム
LAB01 : DW      0       ;  “LAB01”はラベル
        NAME    SAMPLE  ;  “SAMPLE”はモジュール名
MAC1    MACRO           ;  “MAC1”はマクロ名
```

＜誤ったシンボルの例＞

```
1ABC    EQU     3       ; 先頭文字に数字は使用できません。
LAB     MOV     A , R0  ;  “LAB”ラベルです。ニモニック欄とコロン（ : ）で区切ります。
FLAG :  EQU     10H     ; ネームにはコロン（ : ）が必要ありません。
```

＜長いシンボルの例＞

```
A123456789B12 ～ Y123456789Z123456        EQU     70H
        257 文字                        ; 認識最大文字数（256 文字）を越えた文字“6”は
                                        ; 無視されます。
                                        ; A123456789B12 ～ Y12345678912345
                                        ; というシンボルが定義されていることになります。
```

＜シンボルのみからなる文の例＞

```
ABCD :                  ; ABCD がラベルとして定義されます。
```

(3) シンボルに関する注意事項

??RAnnnn（nnnn = 0000 - FFFF）というシンボルは，マクロ・ボディ内のローカル・シンボルが展開されるごとに，アセンブラによって自動的に置き換えられるシンボルであるため，二重定義しないように注意してください。

また，セグメント定義疑似命令でセグメント名が指定されなかったときは，アセンブラがセグメント名を自動生成します。次に，そのセグメントを示します。

同名で定義するとエラーとなります。

セグメント名	疑似命令	再配置属性
?A0nnnnn（nnnnn = 00000 - FFFFF）	ORG 疑似命令	（なし）
?CSEG	CSEG 疑似命令	UNIT
?CSEGUP		UNITP
?CSEGT0		CALLT0
?CSEGFX		FIXED
?CSEGSI		SECUR_ID
?CSEGB		BASE
?CSEGP64		PAGE64KP
?CSEGU64		UNIT64KP
?CSEGMIP		MIRRORP
?CSEGOB0		OPT_BYTE
?DSEG	DSEG 疑似命令	UNIT
?DSEGUP		UNITP
?DSEGS		SADDR
?DSEGSP		SADDRP
?DSEGBP		BASEP
?DSEGP64		PAGE64KP
?DSEGU64		UNIT64KP
?BSEG	BSEG 疑似命令	UNIT

(4) シンボルの属性

ネーム，およびラベルは，値と属性を持ちます。

値とは，定義された数値データやアドレス・データの値そのものです。

セグメント名，モジュール名，およびマクロ名は，値を持ちません。

属性とは，次に示すシンボル属性のことです。

属性の種類	区分	値
NUMBER	- 数値定数を割り付けたネーム - EXTRN 疑似命令で定義されたシンボル - 定数	10 進表現 ： 0 - 1048575 16 進表現 ： 00000H - FFFFFH （符号なし）
ADDRESS	- ラベルとして定義されたシンボル - ラベルを EQU，SET 疑似命令で定義したネーム	10 進表現 ： 0 - 1048575 16 進表現 ： 0H - FFFFFH
BIT	- ビット値として定義されたネーム - BSEG 内のネーム - EXTBIT 疑似命令で定義されたシンボル	0H - FFFFFH
SFR	SFR を EQU 疑似命令で定義したネーム	SFR 領域
SFRP	SFR を EQU 疑似命令で定義したネーム	
CSEG	CSEG 疑似命令で定義されたセグメント名	値を持ちません
DSEG	DSEG 疑似命令で定義されたセグメント名	
BSEG	BSEG 疑似命令で定義されたセグメント名	
MODULE	NAME 疑似命令で定義されたモジュール名 （指定されなかった場合は，入力ソース・ファイル名のプライマリ・ネームから作成されます）	
MACRO	MACRO 疑似命令で定義されたマクロ名	

＜例＞

```
TEN      EQU     10H          ; ネーム“TEN”は NUMBER 属性と値 10H を持ちます。
         ORG     80H
START :  MOV     A , #10H     ; ラベル“START”は，ADDRESS 属性と値 80H を持ちます。
BIT1     EQU     0FFE20H.0    ; ネーム“BIT1”は，BIT 属性と値 0FFE20H.0 を持ちます。
```

2.2.4 ニモニック欄

ニモニック欄には，インストラクションのニモニック，疑似命令，およびマクロ参照を記述します。

オペランドの必要なインストラクションや疑似命令の場合，ニモニック欄とオペランド欄を 1 つ以上の空白，または TAB で区切ります。

ただし，インストラクションの第 1 オペランドの先頭が“#”，“$”，“！”，“[”の場合には，ニモニックと第 1 オペランドの間に何もなくても，正常にアセンブルが行われます。

＜正しい例＞

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

＜誤った例＞

```
MOVA    #0H             ; ニモニック欄とオペランド欄の間に，空白がありません。
C ALL    !CONVAH        ; ニモニック中に空白があります。
ZZZ                     ; 78K0R の命令には，“ZZZ”はありません。
```

2.2.5 オペランド欄

オペランド欄には，インストラクションや疑似命令，およびマクロ参照の実行に必要なデータ（オペランド）を記述します。

各インストラクションや疑似命令により，オペランドを必要としないものや，複数のオペランドを必要とするものがあります。

2 個以上のオペランドを記述する場合には，各オペランドをコンマ（ , ）で区切ります。

オペランド欄に記述できるものは，次のものです。

- 定数（数値定数，文字列定数）
- 文字列
- レジスタ名
- 特殊文字（$ # ! [ ]）
- セグメント定義疑似命令の再配置属性
- シンボル
- 式
- ビット項

なお，各インストラクションや疑似命令により，要求するオペランドのサイズ，属性などが異なります。これらについては「2.15 オペランドの特性」を参照してください。

また，インストラクション・セットにおけるオペランドの表現形式と記述方法については，開発対象となる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。

以降に，オペランド欄に記述可能な各項目について説明します。

(1) 定数

定数は，それ自身で定まる値を持つもので，イミーディエト・データとも呼びます。

定数には，数値定数と文字列定数があります。

(a) 数値定数

数値定数として，2 進数，8 進数，10 進数，16 進数が記述可能です。

次に，各数値定数の表現方法を示します。

数値定数は，符号なしの 32 ビット・データとして処理されます。

値の範囲　0 ≦ n ≦ 0FFFFFFFFH

マイナスの値を記述するには，演算子のマイナス符号を使用します。

数値定数の種類	表記方法	表記例
2 進数	数値の最後に文字“B”，または“Y”を付加	1101B 1101Y
8 進数	数値の最後に文字“O”，または“Q”を付加	74O 74Q
10 進数	数値をそのまま記述，または最後に文字“D”，または“T”を付加	128 128D 128T
16 進数	- 数値の最後に文字“H”を付加 - 先頭文字が“A”，“B”，“C”，“D”，“E”，“F”で始まる場合には，その前に“0”を付加	8CH 0A6H

(b) 文字列定数

文字列定数は，「2.2.2 文字セット」で示した文字を引用符（'）で囲んだものです。

文字列定数は，アセンブルされた結果，パリティ・ビットを 0 とした 7 ビット ASCII コードに変換されます。

文字列の長さは 0 - 2 です。

引用符自体を文字列定数とする場合には，引用符を 2 個続けて記述します。

＜文字列定数の表記例＞

```
'ab'            ; 6162H
'A'             ; 0041H
'A'''           ; 4127H
' '             ; 0020H（空白１個）
```

(2) 文字列

文字列は，「2.2.2 文字セット」で示した文字を引用符（'）で囲んだものです。

文字列は，DB，CALL 疑似命令や TITLE，SUBTITLE 制御命令のオペランドに使用します。

＜文字列の使用例＞

```
        CSEG
MAS1 :  DB      'YES'           ; 文字列“YES”で初期化します。
MAS2 :  DB      'NO'            ; 文字列“NO”で初期化します。
```

(3) レジスタ名

オペランド欄に記述可能なレジスタとして，次のものがあります。

- 汎用レジスタ
- 汎用レジスタ・ペア
- 特殊機能レジスタ

汎用レジスタや汎用レジスタ・ペアは，絶対名称（R0 - R7，RP0 - RP3）での記述のほかに，機能名称（X，A，B，C，D，E，H，L，AX，BC，DE，HL）での記述も可能です。

なお，インストラクションの種類により，オペランド欄に記述可能なレジスタ名が異なります。各レジスタの記述方法の詳細については，開発対象となる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。

(4) 特殊文字

次に，記述可能な特殊文字を示します。

特殊文字	機能
$	- このオペランドを待つインストラクションが割り当てられているロケーション・アドレス（複数バイト命令の場合は 1 バイト目）を示します。 - 分岐命令の相対アドレッシングを示します。
!	- 分岐命令の絶対アドレッシングを示します。 - MOV 命令の全メモリ空間指定可能な addr16 指定を示します。
#	- イミーディエト・データを示します。
[ ]	- インダイレクト・アドレッシングを示します。

＜特殊文字の使用例＞

```
アドレス        ソース
100                     ADD     A , #10H
102             LOOP :  INC     A
103                     BR      $$ - 1          ; (1)
105                     BR      !$ + 100H       ; (2)
```

(1) オペランドの 2 番目の $ は，103H 番地を示します。“BR $ - 1”と記述しても同様に動作します。

(2) オペランドの 2 番目の $ は，105H 番地を示します。“BR $ + 100H”と記述しても同様に動作します。

(5) セグメント定義疑似命令の再配置属性

オペランド欄には，再配置属性を記述するとこができます。

再配置属性の詳細については，「3.2 セグメント定義疑似命令」を参照してください。

(6) シンボル

シンボルをオペランド欄に記述した場合は，そのシンボルに割り付けられたアドレス（または値）がオペランドの値になります。

＜シンボルの使用例＞

```
VALUE   EQU     1234H
        MOV     AX , #VALUE     ; MOV     AX , #1234H と記述することができます。
```

(7) 式

式は，定数，ロケーション・アドレスを示す $，ネーム，またはラベルを演算子で結合したものです。

インストラクションのオペランドとして数値表現可能なところに記述することができます。

式と演算子については，「2.3 式と演算子」を参照してください。

＜式の例＞

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

この記述例では，“TEN - 5H”が式です。

この式は，ネームと数値定数が -（マイナス）演算子で結合されています。式の値は“BH”です。

したがって，この記述は“MOV A , #0BH”と書き換えることが可能です。

(8) ビット項

ビット項は，ビット位置指定子によって得ることができます。

ビット項の詳細については，「2.14 ビット位置指定子」を参照してください。

＜ビット項の例＞

```
        CLR1    A.5
        SET1    1 + 0FFE30H.3   ;オペランドの値は 0FFE31H.3 です。
        CLR1    0FFE40H.4 + 2   ;オペランドの値は 0FFE40H.6 です。
```

2.2.6 コメント欄

コメント欄には，セミコロン（ ; ）のあとにコメント（注釈）を記述します。

コメント欄は，セミコロンからその行の改行コード，または EOF までです。

コメントを記述することにより，理解しやすいソースを作成できます。

コメント欄の記述は，機械語変換というアセンブル処理の対象とはならず，そのままアセンブル・リストに出力されます。

記述可能な文字は，「2.2.2 文字セット」に示すものです。

＜コメントの例＞

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

        PUBLIC  MAIN , START
        EXTRN   CONVAH
        EXTRN   @STBEG

DATA    DSEG    saddr
HDTSA:  DS      1
STASC:  DS      2

CODE    CSEG    AT 0H
MAIN :  DW      START

        CSEG
START :
                                ; chip initialize
        MOVW    SP , #_@STBEG

        MOV     HDTSA , #1AH
        MOVW    HL , #HDTSA     ; set hex 2-code data in HL register

        CALL    !CONVAH         ; convert ASCII <- HEX
                                ; output BC-register <- ASCII code

        MOVW    DE , #STASC     ; set DE <- store ASCII code table
        MOV     A , B
        MOV     [ DE ] , A
        INCW    DE
        MOV     A , C
        MOV     [ DE ] , A
        BR      $$

        END
```

コメント欄のみの行

コメント欄のみの行

コメント欄にコメントが記述されている行

2.3 式と演算子

式とは，シンボル，定数，ロケーション・アドレスを示す $，ビット項，前述の 4 つに演算子を付加したもの，または演算子で結合したものです。

式を構成する演算子以外の要素を項といい，記述された左側から順に第 1 項，第 2 項，…と呼びます。

演算子には表 2-4 に示すものがあり，演算実行上の優先順位が表 2-5 のように決められています。

演算の順序を変更するには，かっこ“( )”を使用します。

＜例＞

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

(1) では，“5 * (SYM+1)”が式です。“5”が第 1 項，“SYM”が第 2 項，“1”が第 3 項です。“*”，“+”，“( )”が演算子です。

表 2-4 演算子の種類

演算子の種類	演算子
算術演算子	+，-，*，/，MOD，+ 符号，- 符号
論理演算子	NOT，AND，OR，XOR
比較演算子	EQ (=)，NE (< >)，GT (>)，GE (>=)，LT (<)，LE (<=)
シフト演算子	SHR，SHL
バイト分離演算子	HIGH，LOW
ワード分離演算子	HIGHW，LOWW
特殊演算子	DATAPOS，BITPOS，MASK
その他の演算子	( )

上記の演算子は，単項演算子，特殊単項演算子，2 項演算子，N 項演算子，その他の演算子に分けられます。

単項演算子	+ 符号，- 符号，NOT，HIGH，LOW，HIGHW，LOWW
特殊単項演算子	DATAPOS，BITPOS
2 項演算子	+，-，*，/，MOD，AND，OR，XOR，EQ (=)，NE (<>)，GT (>)，GE (>=)，LT (<)，LE (<=)，SHR，SHL
N 項演算子	MASK
その他の演算子	( )

表 2-5 演算子の優先順位

優先度	優先順位	演算子
高い	1	+ 符号，- 符号，NOT，HIGH，LOW，HIGHW，LOWW，DATAPOS，BITPOS，MASK
	2	*，/，MOD，SHR，SHL
	3	+，-
	4	AND
	5	OR，XOR
低い	6	EQ (=)，NE (<>)，GT (>)，GE (>=)，LT (<)，LE (<=)

式の演算は，次の規則に従います。

- 演算の順序は，演算子の優先順位に従います。
  同一順位の場合は，左から右に演算されます。単項演算子の場合は，右から左に演算されます。
- かっこ“( )”の中の演算は，かっこの外の演算に先立って行われます。
- 単項演算子の多重演算が可能です。

【例】

```
1 = - - 1 == 1
-1 = -+1 = -1
```

- 式の演算は，符号なし 32 ビットで行います。
  演算中に 32 ビットを越えてオーバーフローした場合，オーバーフローした値は無視されます。
- 定数が 32 ビットを越える場合には，エラーとなり，その値は 0 とみなされて計算されます。
- 除算では，小数部分を切り捨てます。
  除算がゼロの場合は，エラーとなり，結果は 0 となります。
- 負の値は，2 の補数形式となります。
- 外部参照記号のアセンブル時の評価値はゼロです（評価値はリンク時に決定されます）。
- オペランド欄に記述した式の演算結果は，命令の要求を満たす値でなければなりません。
  8 ビット長のオペランドを要求される命令で，リロケータブル，または外部参照の式を記述した場合は，下位 8 ビットの値からオブジェクトが生成され，リロケーション情報には 16 ビットで必要な情報が出力されます。そして，リンカにおいて，決定された値が 8 ビットの範囲に収まるかのチェックがされ，オーバーフローすると，リンク時にエラーとなります。
  アブソリュートな式を記述した場合は，アセンブラ内で値が決定されるので，要求した範囲に収まるかのチェックが行われます。
  例えば，MOV 命令の場合，オペランドは 8 ビットなので，0H - 0FFH の範囲に入っていなければなりません。

＜正しい例＞

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

＜誤った例＞

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

＜式評価の例＞

式	評価値
2 + 4 * 5	22
(2 + 3) * 4	20
10/4	2
0 - 1	0FFFFFFFFH
-1 > 1	00H（偽）
EXT 注 + 1	1

注 EXT ： 外部参照記号

2.4 算術演算子

算術演算子には，次のものがあります。

- +
- -
- *
- /
- MOD
- + 符号
- - 符号

+

【機能】

- 第 1 項と第 2 項の値の和を返します。

【使用例】

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

- BR 命令により，“現在のロケーション・アドレス＋ 6 番地”へジャンプします。

  つまり，“100H + 6H = 106H”へジャンプします。

  したがって，(a) は“START:　BR　!106H”と記述することもできます。

-

【機能】

- 第 1 項と第 2 項の値の差を返します。

【使用例】

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

- BR 命令により，「“BACK”に割り付けられたアドレス－ 6 番地」へジャンプします。

  つまり，“100H - 6H = 0FAH”へジャンプします。

  したがって，(a) は“BACK:　BR　!0FAH”と記述することもできます。

*****

【機能】

- 第 1 項と第 2 項の値の積を返します。

【使用例】

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

- EQU 疑似命令により，ネーム“TEN”に 10H という値が定義されます。

“#”はイミーディエト・データを示します。

“TEN * 3”という式は“10H * 3”のことで，30H を返します。

したがって，(a) は“MOV A , #30H”と記述することもできます。

/

【機能】

- 第 1 項の値を第 2 項の値で割り，その値の整数部を返します。
  小数部は切り捨てられます。
  除数（第 2 項）が 0 の場合は，エラーとなります。

【使用例】

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

- “256 / 50 = 5　余り 6”となります。
  よって，整数部の 5 を返します。
  したがって，(a) は“MOV A , #5”と記述することもできます。

MOD

【機能】

- 第 1 項の値を第 2 項の値で割り，その値の余りを返します。

  除数が 0 の場合は，エラーとなります。

  MOD の前後には，空白が必要です。

【使用例】

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

- “256 / 50 = 5　余り 6”となります。

  よって，余りの 6 を返します。

  したがって，(a) は“MOV A , #6”と記述することもできます。

+ 符号

【機能】

- 項の値をそのまま返します。

【使用例】

```
FIVE    EQU     +5
```

- 項の値“5”をそのまま返します。

  EQU 疑似命令により，ネーム“FIVE”に 5 という値が定義されます。

- 符号

【機能】

- 項の値の 2 の補数をとった値を返します。

【使用例】

```
NO      EQU     -1
```

- “-1”は 1 の 2 の補数となります。

0000 0000 0000 0000 0000 0000 0000 0001 の 2 の補数は

1111 1111 1111 1111 1111 1111 1111 1111

となります。

よって，EQU 疑似命令により，ネーム“NO”に 0FFFFFFFFH が定義されます。

2.5 論理演算子

論理演算子には，次のものがあります。

- NOT
- AND
- OR
- XOR

NOT

【機能】

- 項のビットごとの論理否定をとり，その値を返します。

  NOT と項との間には，空白が必要です。

【使用例】

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

- “3H”の論理否定をとります。

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

よって，0FFFFFFFCH を返します。

したがって，(a) は“MOVW AX , #LOWW 0FFFFFFFCH”と記述することもできます。

AND

【機能】

- 第 1 項の値と第 2 項の値のビットごとの論理積をとり，その値を返します。

  AND の前後には，空白が必要です。

【使用例】

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

- “6FAH”と“0FH”の論理積をとります。

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

よって，“0AH”を返します。

したがって，(a) は“MOV A , #0AH”と記述することもできます。

OR

【機能】

- 第 1 項の値と第 2 項の値のビットごとの論理和をとり，その値を返します。

  OR の前後には，空白が必要です。

【使用例】

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

- “0AH”と“1101B”の論理和をとります。

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

よって，“0FH”を返します。

したがって，(a) は“MOV A , #0FH”と記述することもできます。

XOR

【機能】

- 第 1 項の値と第 2 項の値のビットごとの排他的論理和をとり，その値を返します。

  XOR の前後には，空白が必要です。

【使用例】

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

- “9AH”と“9DH”の排他的論理和をとります。

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

よって，“7H”を返します。

したがって，(a) は“MOV A , #7H”と記述することもできます。

2.6 比較演算子

比較演算子には，次のものがあります。

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

EQ (=)

【機能】

- 第 1 項の値と第 2 項の値が等しいときに 0FFH（真），等しくないときに 00H（偽）を返します。

  EQ の前後には，空白が必要です。

【使用例】

```
A1      EQU     12C4H
A2      EQU     12C0H

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

- (a) の場合

  “A1 EQ (A2 + 4H)”は，“12C4H EQ (12C0H + 4H) ”となります。

  このとき，第 1 項の値と第 2 項の値が等しいので，0FFH を返します。

- (b) の場合

  “A1 EQ A2”は，“12C4H EQ 12C0H”となります。

  このとき，第 1 項の値と第 2 の値が等しくないので，00H を返します。

NE (< >)

【機能】

- 第 1 項の値と第 2 項の値が等しくないときに 0FFH（真），等しいときに 00H（偽）を返します。

  NE の前後には，空白が必要です。

【使用例】

```
A1      EQU     5678H
A2      EQU     5670H

        MOV     A , #A1  NE   A2                ; (a)
        MOV     A , #A1  NE  ( A2 + 8H )        ; (b)
```

- (a) の場合

  “A1 NE A2”は“5678H NE 5670H”となります。

  このとき，第 1 項の値と第 2 項の値が等しくないので，0FFH を返します。

- (b) の場合

  “A1 NE (A2 + 8H) ”は“5678H NE (5670H + 8H) ”となります。

  このとき，第 1 項の値と第 2 項の値が等しいので，00H を返します。

GT (>)

【機能】

- 第 1 項の値が第 2 項の値より大きいときに 0FFH（真），等しいか小さいときに 00H（偽）を返します。

  GT の前後には，空白が必要です。

【使用例】

```
A1      EQU     1023H
A2      EQU     1013H

        MOV     A , #A1  GT   A2                ; (a)
        MOV     X , #A1  GT  ( A2 + 10H )       ; (b)
```

- (a) の場合

  “A1 GT A2”は“1023H GT 1013H”となります。

  このとき，第 1 項の値が第 2 項の値より大きいので，0FFH を返します。

- (b) の場合

  “A1 GT (A2 + 10H) ”は“1023H GT (1013H + 10H) ”となります。

  このとき，第 1 項の値が第 2 項の値と等しいので，00H を返します。

GE (>=)

【機能】

- 第 1 項の値が第 2 項の値より大きいか，等しいときに 0FFH（真），小さいときに 00H（偽）を返します。

  GE の前後には，空白が必要です。

【使用例】

```
A1      EQU     2037H
A2      EQU     2015H

        MOV     A , #A1  GE   A2                ; (a)
        MOV     X , #A1  GE  ( A2 + 23H )       ; (b)
```

- (a) の場合

  “A1 GE A2”は“2037H GE 2015H”となります。

  このとき，第 1 項の値が第 2 項の値より大きいので，0FFH を返します。

- (b) の場合

  “A1 GE (A2 + 23H) ”は“2037H GE (2015H + 23H) ”となります。

  このとき，第 1 項の値が第 2 項の値より小さいので，00H を返します。

LT (<)

【機能】

- 第 1 項の値が第 2 項の値より小さいときに 0FFH（真），等しいか大きいときに 00H（偽）を返します。

  LT の前後には，空白が必要です。

【使用例】

```
A1      EQU     1000H
A2      EQU     1020H

        MOV     A , #A1  LT  A2                 ; (a)
        MOV     X , # ( A1 + 20H )  LT  A2      ; (b)
```

- (a) の場合

  “A1 LT A2”は“1000H LT 1020H”となります。

  このとき，第 1 項の値が第 2 の値より小さいので，0FFH を返します。

- (b) の場合

  “(A1 + 20H) LT A2”は“(1000H + 20H) LT 1020H”となります。

  このとき，第 1 項の値と第 2 項の値が等しいので，00H を返します。

LE (<=)

【機能】

- 第 1 項の値が第 2 項の値より小さいか等しいときに 0FFH（真），大きいときに 00H（偽）を返します。

  LE の前後には，空白が必要です。

【使用例】

```
A1      EQU     103AH
A2      EQU     1040H

        MOV     A , #A1  LE  A2                 ; (a)
        MOV     X , # ( A1 + 7H )  LE  A2       ; (b)
```

- (a) の場合

  “A1 LE A2”は“103AH LE 1040H”となります。

  このとき，第 1 項の値が第 2 の値より小さいので，0FFH を返します。

- (b) の場合

  “(A1 + 7H) LE A2”は“(103AH + 7H) LE 1040H”となります。

  このとき，第 1 項の値が第 2 項の値より大きいので，00H を返します。

2.7 シフト演算子

シフト演算子には，次のものがあります。

- SHR
- SHL

SHR

【機能】

- 第 1 項の値を第 2 項で示す値（ビット数）分だけ右シフトし，その値を返します。

  上位ビットには，シフトされたビット数だけ 0 が挿入されます。

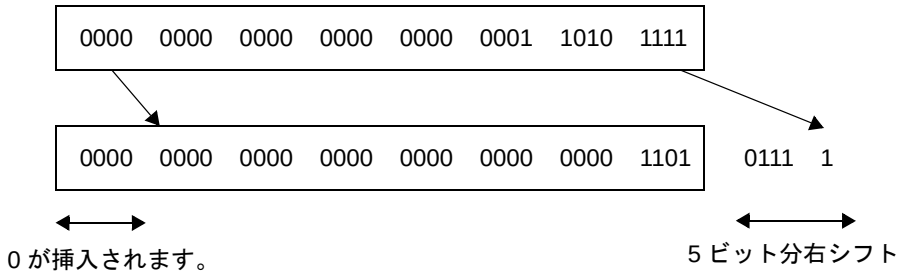
  SHR の前後には，空白が必要です。

  シフト数が 0 の場合は，第 1 項の値がそのまま返されます。シフト数が 32 を越えた場合は，自動的に 0 が埋め込まれます。

【使用例】

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

- “01AFH”を 5 ビット分右シフトします。



よって，“000DH”を返します。

したがって (a) は，“MOV A , #0DH”と記述することもできます。

SHL

【機能】

- 第 1 項の値を第 2 項で示す値（ビット数）分だけ左シフトし，その値を返します。

  下位ビットには，シフトされたビット数だけ 0 が挿入されます。

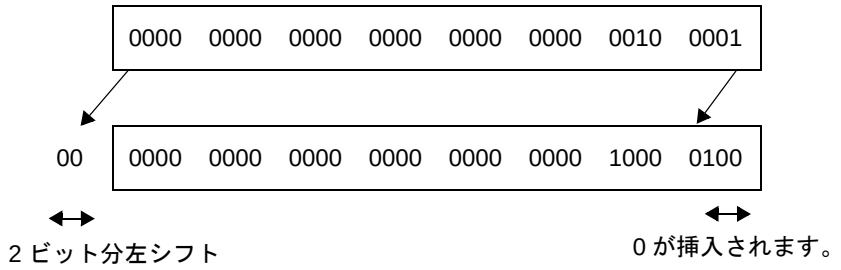
  SHL の前後には，空白が必要です。

  シフト数が 0 の場合は，第 1 項の値がそのまま返されます。シフト数が 32 を越えた場合は，自動的に 0 が埋め込まれます。

【使用例】

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

- “21H”を 2 ビット分左シフトします。



よって，“84H”を返します。

したがって，(a) は，“MOV A , #84H”と記述することもできます。

2.8 バイト分離演算子

バイト分離演算子には，次のものがあります。

- HIGH
- LOW

HIGH

【機能】

- 項の上位 8 ビットを返します。

  HIGH と項との間には，空白が必要です。

【使用例】

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

- MOV 命令実行により，1234H の上位 8 ビット値 12H を返します。

  したがって，(a) は“MOV A , #12H”と記述することもできます。

【備考】

SFR 名に対して HIGH 演算を行う場合は，次のように行います。

記述方法には，次の 2 つの方法があります。

```
HIGH △ SFR 名
```

または

```
HIGH[ △ ]([ △ ]SFR 名 [ △ ])
```

演算結果は，アブソリュートな NUMBER 属性の項となります。

ただし，SFR 名に他の演算を施すことはできません。

＜例＞

```
シンボル欄   ニーモニック欄   オペランド欄
             MOV          R0 , #HIGH PM0
             MOV          R1 , #HIGH PM1 + 1H     ; #( HIGH PM1 ) + 1 と同意
             MOV          R1 , #HIGH ( PM1 + 1H ) ; SFR 名に HIGH，LOW，HIGHW，LOWW
                                                  ; 以外の演算子が施されているので
                                                  ; エラー
```

LOW

【機能】

- 項の下位 8 ビットを返します。

  LOW と項との間には，空白が必要です。

【使用例】

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

- MOV 命令実行により，1234H の下位 8 ビット値 34H を返します。

  したがって，(b) は“MOV A , #34H”と記述することもできます。

【備考】

SFR 名に対して LOW 演算を行う場合は，以下のように行います。

記述方法には，次の 2 つの方法があります。

```
LOW △ SFR 名
```

または

```
LOW[ △ ]([ △ ]SFR 名 [ △ ])
```

演算結果は，アブソリュートな NUMBER 属性の項となります。

ただし，SFR 名に他の演算を施すことはできません。

＜例＞

```
シンボル欄   ニーモニック欄   オペランド欄
             MOV            R0 , #LOW PM0
             MOV            R1 , #LOW PM1 + 1H       ; #( LOW PM1 ) + 1 と同意
             MOV            R1 , #LOW ( PM1 + 1H )   ; SFR 名に HIGH，LOW，HIGHW，LOWW
                                                     ; 以外の演算子が施されているので
                                                     ; エラー
```

2.9 ワード分離演算子

ワード分離演算子には，次のものがあります。

- HIGHW
- LOWW

HIGHW

【機能】

- 項の上位 16 ビットを返します。

  HIGHW と項との間には，空白が必要です。

【使用例】

```
MOVW    AX , #HIGHW     12345678H       ; (a)

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

- MOVW 命令実行により，12345678H の上位 16 ビット値 1234H を返します。

  したがって，(a) は“MOVW AX , #1234H”と記述することもできます。

- (b) は，MOV 命令実行により，ラベル LAB の上位アドレスを ES レジスタに設定します。

【備考】

SFR 名に対して HIGHW 演算を行う場合は，以下のように行います。

記述方法には，次の 2 つの方法があります。

```
HIGHW △ SFR 名
```

または

```
HIGHW[ △ ]([ △ ]SFR 名 [ △ ])
```

演算結果は，アブソリュートな NUMBER 属性の項となります。

ただし，SFR 名に他の演算を施すことはできません。

＜例＞

```
シンボル欄   ニーモニック欄   オペランド欄
             MOVW          RP0 , #HIGHW PM0
             MOVW          RP1 , #HIGHW PM1 + 1H      ; #( HIGHW PM1 ) + 1 と同意
             MOVW          RP1 , #HIGHW ( PM1 + 1H ) ; SFR 名に HIGH，LOW，HIGHW，
                                                      ; LOWW 以外の演算子が施されて
                                                      ; いるのでエラー
```

LOWW

【機能】

- 項の下位 16 ビットを返します。

  LOWW と項との間には，空白が必要です。

【使用例】

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

- MOVW 命令実行により，12345678H の下位 16 ビット値 5678H を返します。

  したがって，(a) は“MOVW AX , #5678H”と記述することもできます。

【備考】

SFR 名に対して LOWW 演算を行う場合は，以下のように行います。

記述方法には，次の 2 つの方法があります。

```
LOWW △ SFR 名
```

または

```
LOWW[ △ ]([ △ ]SFR 名 [ △ ])
```

演算結果は，アブソリュートな NUMBER 属性の項となります。

ただし，SFR 名に他の演算を施すことはできません。

＜例＞

```
シンボル欄  ニーモニック欄  オペランド欄
            MOVW        RP0 , #LOWW PM0
            MOVW        RP1 , #LOWW PM1 + 1H      ; #( LOWW PM1 ) + 1 と同意
            MOVW        RP1 , #LOWW ( PM1 + 1H )  ; SFR 名に HIGH，LOW，HIGHW，
                                                  ; LOWW 以外の演算子が施されて
                                                  ; いるのでエラー
```

2.10 特殊演算子

特殊演算子には，次のものがあります。

- DATAPOS
- BITPOS
- MASK

DATAPOS

【機能】

- ビット・シンボルのアドレス部（バイト・アドレス）を返します。

【使用例】

```
SYM     EQU     0FE68H.6

        MOV     A , !DATAPOS  SYM       ; (a)
```

- EQU 疑似命令により，ネーム“SYM”に 0FE68H.6 という値が定義されます。

  “DATAPOS SYM”は“DATAPOS 0FE68H.6”ということで，0FE68H を返します。

  したがって，(a) は“MOV A , !0FE68H”と記述することもできます。

BITPOS

【機能】

- ビット・シンボルのビット部（ビット位置）を返します。

【使用例】

```
SYM     EQU     0FE68H.6

        CLR1    [ HL ].BITPOS  SYM
```

- EQU 疑似命令により，ネーム“SYM”に 0FE68H.6 という値が定義されます。

  “BITPOS.SYM”は“BITPOS 0FE68H.6”ということで，6 を返します。

  CLR1 命令実行により，[HL].6 を 0 クリアします。

MASK

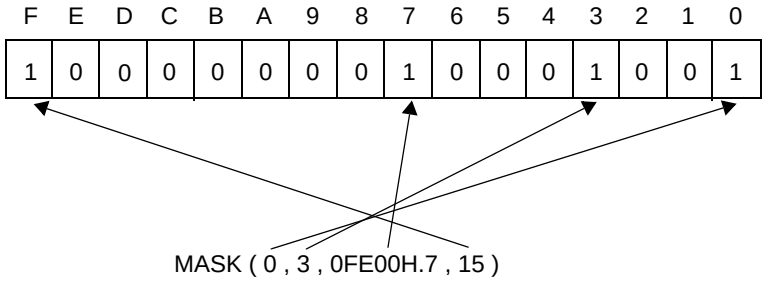
【機能】

- 指定のビット位置に 1，その他を 0 にした 16 ビット値を返します。

【使用例】

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

- MOVW 命令実行により，8089H を返します。



2.11 その他の演算子

その他の演算子には，次のものがあります。

- ( )

( )

【機能】

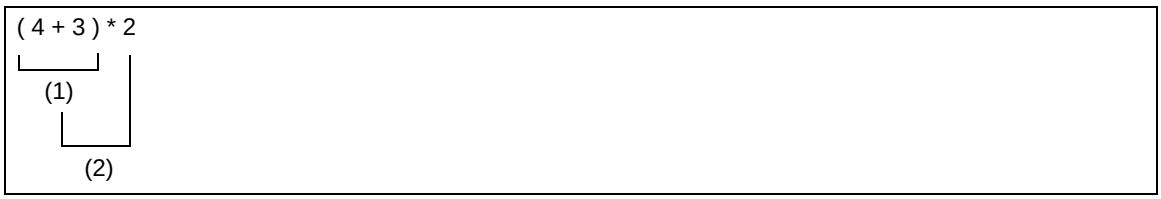
- ( ) 内の演算を，( ) 外の演算に先立って行います。

  演算の優先順位を変更したいときに使用します。

  ( ) が多重になっている場合は，一番内側の ( ) 内の式から演算します。

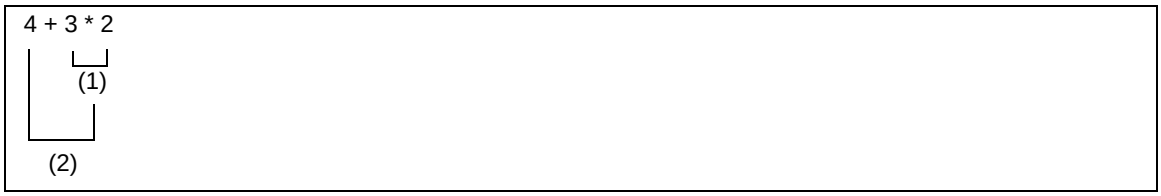
【使用例】

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



(1)，(2) の順で演算を行い，14 という値を返します。

( ) がなければ



(1)，(2) の順で演算を行い，10 という値を返します。

演算子の優先順位については，表 2-5 を参照してください。

2.12 演算の制限

式の演算は，項を演算子で結びつけて行います。項として記述できるものには，定数，$，ネーム，ラベルがあり，各項はリロケーション属性とシンボル属性を持ちます。

各項の持つリロケーション属性，シンボル属性の種類により，その項に対して演算可能な演算子が限られます。したがって，式を記述する場合には，式を構成する各項のリロケーション属性，シンボル属性に留意することが大切です。

2.12.1 演算とリロケーション属性

式を構成する各項は，リロケーション属性とシンボル属性を持ちます。

各項をリロケーション属性により分類すると，アブソリュート項，リロケータブル項，外部参照項の 3 種類に分けられます。

次に，演算におけるリロケーション属性の種類とその性質，およびそれに該当する項を示します。

表 2-6 リロケーション属性の種類

種類	性質	該当項
アブソリュート項	アセンブル時に値，定数が決定する項	- 定数 - アブソリュート・セグメント内のラベル - アブソリュート・セグメント内で定義したロケーション・アドレスを示す $ - 定数，上記のラベル，上記の $ 等，絶対値を定義したネーム
リロケータブル項	アセンブル時には値が決定しない項	- リロケータブル・セグメント内で定義したラベル - リロケータブル・セグメント内で定義したロケーション・アドレスを示す $ - リロケータブルなシンボルで定義したネーム
外部参照項[注]	ほかのモジュールのシンボルを外部参照する項	- EXTRN 疑似命令で定義したラベル - EXTBIT 疑似命令で定義したネーム

注 外部参照項を演算対象にすることができる演算子は，“+”，“-”，“HIGH”，“LOW”，“HIGHW”，“LOWW”の 6 つです。ただし，1 つの式に記述可能な外部参照項は，1 つだけです。その場合，必ず演算子“+”で結合されていなければなりません。

演算可能な演算子と項の組み合わせをリロケーション属性により分類すると，次のようになります。

表 2-7 リロケーション属性による項と演算子の組み合わせ（リロケータブル項）

演算子の種類	項のリロケーション属性			
	X：ABS Y：ABS	X：ABS Y：REL	X：REL Y：ABS	X：REL Y：REL
X + Y	A	R	R	—
X - Y	A	—	R	A 注 1
X * Y	A	—	—	—
X / Y	A	—	—	—
X MOD Y	A	—	—	—
X SHL Y	A	—	—	—
X SHR Y	A	—	—	—
X EQ Y	A	—	—	A 注 1
X LT Y	A	—	—	A 注 1
X LE Y	A	—	—	A 注 1
X GT Y	A	—	—	A 注 1
X GE Y	A	—	—	A 注 1
X NE Y	A	—	—	A 注 1
X AND Y	A	—	—	—
X OR Y	A	—	—	—
X XOR Y	A	—	—	—
NOT X	A	A	—	—
+ X	A	A	R	R
- X	A	A	—	—
HIGH X	A	A	R 注 2	R 注 2
LOW X	A	A	R 注 2	R 注 2
HIGHW X	A	A	R 注 2	R 注 2
LOWW X	A	A	R 注 2	R 注 2
MASK (X)	A	A	—	—
DATAPOS X.Y	A	—	—	—
BITPOS X.Y	A	—	—	—
MASK (X.Y)	A	—	—	—
DATAPOS X	A	A	R	R
BITPOS X	A	A	A	A

ABS ：アブソリュート項

REL ：リロケータブル項

A ：演算結果がアブソリュート項になります。

R ：演算結果がリロケータブル項になります。

－ ：演算不可

注 1 X，または Y が HIGH，LOW，HIGHW，LOWW，DATAPOS 演算を行ったリロケータブル項ではなく，X，Y がともに同一セグメントにある場合にかぎり，演算可能です。

注 2 X，または Y が HIGH，LOW，HIGHW，LOWW，DATAPOS 演算を行ったリロケータブル項でない場合にかぎり，演算可能です。

外部参照項を演算対象にすることができる演算子は，“+”，“-”，“HIGH”，“LOW”，HIGHW，LOWW の 6 つです。ただし，1 つの式に記述可能な外部参照項は，1 つだけです。

これらの演算子と外部参照項との実行可能な組み合わせをリロケーション属性により分類すると，次のようになります。

表 2-8 リロケーション属性による項と演算子の組み合わせ（外部参照項）

演算子の種類	項のリロケーション属性				
	X：ABS Y：EXT	X：EXT Y：ABS	X：REL Y：EXT	X：EXT Y：REL	X：EXT Y：EXT
X + Y	E	E	－	－	－
X - Y	－	E	－	－	－
+ X	A	E	R	E	E
HIGH X	A	E 注 1	R 注 2	E 注 1	E 注 1
LOW X	A	E 注 1	R 注 2	E 注 1	E 注 1
HIGHW X	A	E 注 1	R 注 2	E 注 1	E 注 1
LOWW X	A	E 注 1	R 注 2	E 注 1	E 注 1
MASK (X)	A	－	－	－	－
DATAPOS X.Y	－	－	－	－	－
BITPOS X.Y	－	－	－	－	－
MASK (X.Y)	－	－	－	－	－
DATAPOS X	A	E	R	E	E
BITPOS X	A	E	A	E	E

ABS ：アブソリュート項

EXT ：外部参照項

REL ：リロケータブル項

A ：演算結果がアブソリュート項になります。

E ：演算結果が外部参照項になります。

R ：演算結果がリロケータブル項になります。

－ ：演算不可

注 1 X，または Y が HIGH，LOW，HIGHW，LOWW，DATAPOS，BITPOS 演算を行った外部参照項でない場合にかぎり，演算可能です。

注 2 X，または Y が HIGH，LOW，HIGHW，LOWW，DATAPOS 演算を行ったリロケータブル項でない場合にかぎり，演算可能です。

2.12.2 演算とシンボル属性

式を構成する各項は，リロケーション属性に加えてシンボル属性を持ちます。

各項をシンボル属性により分類すると，NUMBER 項，ADDRESS 項の 2 種類に分けられます。

演算におけるシンボル属性の種類とそれに該当する項を，次に示します。

表 2-9 演算におけるシンボル属性の種類

シンボル属性の種類	該当項
NUMBER 項	- NUMBER 属性を持つシンボル - 定数
ADDRESS 項	- ADDRESS 属性を持つシンボル - ロケーション・カウンタを示す“$”

演算可能な演算子と項の組み合わせをシンボル属性により分類すると，次のようになります。

表 2-10 シンボル属性による項と演算子の組み合わせ

演算子の種類	項のシンボル属性			
	X：ADDRESS Y：ADDRESS	X：ADDRESS Y：NUMBER	X：NUMBER Y：ADDRESS	X：NUMBER Y：NUMBER
X + Y	A	A	A	N
X - Y	N	A	N	N
X * Y	N	N	N	N
X / Y	N	N	N	N
X MOD Y	N	N	N	N
X SHL Y	N	N	N	N
X SHR Y	N	N	N	N
X EQ Y	N	N	N	N
X LT Y	N	N	N	N
X LE Y	N	N	N	N
X GT Y	N	N	N	N
X GE Y	N	N	N	N
X NE Y	N	N	N	N
X AND Y	N	N	N	N
X OR Y	N	N	N	N
X XOR Y	N	N	N	N
NOT X	A	A	N	N
+ X	A	A	N	N
- X	A	A	N	N
HIGH X	A	A	N	N
LOW X	A	A	N	N
HIGHW X	A	A	N	N
LOWW X	A	A	N	N
DATAPOS X	A	A	N	N
MASK X	N	N	N	N

ADDRES ： ADDRESS 項

NUMBER ： NUMBER 項

A ： 演算結果が ADDRESS 項

N ： 演算結果が NUMBER 項

ー ： 演算不可

2.12.3 演算の制限についての確認方法

リロケーション属性，シンボル属性による演算の見方について，例を示します。

＜例＞

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

ここで，“TABLE”はリロケータブルなコード・セグメント中で定義されたラベルであると仮定します。

[演算とリロケーション属性]

“TABLE + 5H”は，“リロケータブル項＋アブソリュート項”となるので，この演算を表 2-7 にあてはめます。

演算子の種類 … X + Y

項のリロケーション属性 … X：REL，Y：ABS

したがって，演算結果は“R”，すなわちリロケータブル項になることがわかります。

[演算とシンボル属性]

“TABLE + 5H”は“ADDRESS 項＋ NUMBER 項”となるので，この演算を表 2-10 にあてはめます。

演算子の種類 … X + Y

項のリロケーション属性 … X：ADDRESS，Y：NUMBER

したがって，演算結果は“A”，すなわち ADDRESS 項となることがわかります。

2.13 絶対式の定義

絶対式は，アセンブルの途中で，その式を評価したときに値が確定しているような式を指します。
以下のいずれかに属するものを絶対式と呼びます。

- 定数
- 定数同士に演算を施した式（定数式）
- 定数，または定数式で定義された EQU シンボル，または SET シンボル
- 上記に演算を施した式

備考　シンボルは後方参照のみ可能です。

2.14 ビット位置指定子

ビット位置指定子（．）を使用することにより，ビット・アクセスが可能になります。

.

【記述形式】

```
X[ △ ].[ △ ]Y
└──────┘
   ビット項
```

X（第 1 項）		Y（第 2 項）
汎用レジスタ	A	式（0 - 7）
制御レジスタ	PSW	式（0 - 7）
特殊機能レジスタ	sfr 注	式（0 - 7）
メモリ	[HL] 注	式（0 - 7）

注 具体的な記述については，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。

【機能】

- 第 1 項にバイト・アドレスを指定するもの，第 2 項にビット位置を指定するものを指定します。これにより，ビット・アクセスが可能になります。

【説明】

- ビット位置指定子を使用したものをビット項と呼びます。
- ビット位置指定子には演算子との優先順位はなく，左辺を第 1 項，右辺を第 2 項と認識します。
- 第 1 項には，次の制限があります。
  (1) NUMBER，ADDRESS 属性の式，8 ビット・アクセスが可能な SFR 名，またはレジスタ名（A）が記述可能です。
  (2) 第 1 項にアブソリュートな式を記述する場合は，0H - 0FFFFFH の範囲でなければなりません。
  (3) 外部参照シンボルを記述することができます。
- 第 2 項には，次の制限があります。
  (1) 式の値は，0 - 7 の範囲です。範囲を越えた場合にはエラーとなります。
  (2) アブソリュートな NUMBER 属性の式のみ記述可能です。
  (3) 外部参照シンボルを記述することはできません。

【演算とリロケーション属性】

- リロケーション属性における第 1 項と第 2 項の組み合わせを，次に示します。

項の組み合わせ X :	ABS	ABS	REL	REL	ABS	EXT	REL	EXT	EXT
項の組み合わせ Y :	ABS	REL	ABS	REL	EXT	ABS	EXT	REL	EXT
X.Y	A	—	R	—	—	E	—	—	—

ABS ： アブソリュート項

REL ： リロケータブル項

EXT ： 外部参照項

A ： 演算結果がアブソリュート項になります。

E ： 演算結果が外部参照項になります。

R ： 演算結果がリロケータブル項になります。

ー ： 演算不可

【ビット・シンボルの値】

- EQU 疑似命令のオペランドにビット位置指定子を用いたビット項を記述しビット・シンボルを定義した場合，そのビット・シンボルが持つ値を，次に示します。

オペランドの種類	シンボル値
A.bit 注 2	1.bit
PSW.bit 注 2	FFFFAH.bit
sfr 注 1.bit 注 2	FFFXXH.bit 注 3
式 .bit 注 2	XXXXXH.bit 注 4

注 1 具体的な記述については，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。

注 2 bit = 0 - 7

注 3 FFFXXH は，sfr のアドレス

注 4 XXXXXH は，式の値

【使用例】

```
SET1    0FFE20H.3
SET1    A.5
CLR1    P1.2
SET1    1 + 0FFE30H.3    ; 0FFE31H.3 に等しい
SET1    0FFE40H.4 + 2    ; 0FFE40H.6 に等しい
```

2.15 オペランドの特性

オペランドを必要とする命令（インストラクション，および疑似命令）は，その種類により要求するオペランド値のサイズ，範囲，シンボル属性などが異なります。

たとえば，“MOV r , #byte”というインストラクションの機能は，「レジスタ r に，byte で示される値を転送する」ものです。このとき，レジスタ r は 8 ビット長のレジスタであるため，転送されるデータ“byte”のサイズは，8 ビット以下でなければなりません。

もし，“MOV R0 , #100H”と記述した場合には，第 2 オペランド“100H”のサイズが 8 ビット長を越えているため，アセンブル・エラーとなります。

このように，オペランドを記述する場合には，次のような注意が必要です。

- 値のサイズ，アドレス範囲がその命令のオペランドに適しているかどうか（数値やネーム，ラベル）
- シンボル属性がその命令のオペランドに適しているかどうか（ネーム，ラベル）

2.15.1 オペランドの値のサイズとアドレス範囲

命令のオペランドとして記述可能な数値／ネーム／ラベルの値のサイズとアドレス範囲には条件があります。

インストラクションの場合は，各インストラクションのオペランドの表現形式により，疑似命令の場合には命令の種類により，記述可能なオペランドのサイズとアドレス範囲に条件があります。

これらの条件を，次に示します。

表 2-11 インストラクションのオペランド値の範囲

オペランドの表現形式	値の範囲	
byte	8 ビット値 ： 0H - FFH	
word	word [B] word [C] word [BC]	- 数値定数，および NUMBER 属性のシンボルの場合 0H - FFFFH - ADDRESS 属性のシンボルの場合 以下の (1)，または (2) の領域内 (1) F0000H - FFFFFH (2) MAA=0 の時に RAM 空間にミラーされる領域（01000H - 0xxxxH），または MAA=1 の時に RAM 空間にミラーされる領域（11000H - 1xxxxH）注 1
	ES : word [B] ES : word [C] ES : word [BC]	- 数値定数 および NUMBER 属性のシンボルの場合 0H - FFFFH - ADDRESS 属性のシンボルの場合 0H - FFFFFH
	上記以外	16 ビット値 ： 0H - FFFFH
saddr	FFE20H - FFF1FH 注 4	
saddrp	FFE20H - FFF1FH の偶数値注 4	
sfr	FFF20H - FFFFFH ： 特殊機能レジスタ略号（SFR 略号），および数値定数，NUMBER 属性のシンボル注 5	
sfrp	FFF20H - FFFFFH ： 特殊機能レジスタ略号（16 ビット操作可能な SFR 略号／偶数値のみ），および数値定数，NUMBER 属性のシンボル注 5	
addr20	!!addr20	0H - FFFFFH
	$addr20	0H - FFFFFH，かつ分岐命令の次のアドレスから分岐先までが (-80H) - (+7FH) の範囲
	$!addr20	0H - FFFFFH，かつ分岐，コール命令の次のアドレスから分岐先までが (-8000H) - (+7FFFH) の範囲

オペランドの表現形式	値の範囲	
addr16	!addr16 （BR，CALL 命令）	0H - FFFFH (数値定数，シンボルともに，指定可能な範囲は同じ)
	!addr16 注 2 （BR，CALL 以外の命令）	- 数値定数，および NUMBER 属性のシンボルの場合注 3 0H - FFFFH - ADDRESS 属性のシンボルの場合注 3 以下の (1)，または (2) の領域内 (1) F0000H - FFFFFH (2) MAA=0 のときに RAM 空間にミラーされる領域（例：01000H - 0xxxxH），または MAA=1 のときに RAM 空間にミラーされる領域（例：11000H - 1xxxxH）注 1
	ES:!addr16	- 数値定数，および NUMBER 属性のシンボルの場合注 3 0H - FFFFH - ADDRESS 属性のシンボルの場合注 3 0H - FFFFFH
	!addr16.bit	- DBIT シンボル，SFBIT 属性，SABIT 属性のビット・シンボル，EQU 疑似命令で定義されたビット・シンボル（オペランドに ADDRESS 属性のシンボルを含む場合のみ） 以下の (1)，または (2) の領域内 (1) F0000H - FFFFFH (2) MAA=0 のときに RAM 空間にミラーされる領域（例：01000H - 0xxxxH），または MAA=1 のときに RAM 空間にミラーされる領域（例：11000H - 1xxxxH）注 1 - 上記以外のビットシンボル 0H - FFFFH
	ES : !addr16.bit	- DBIT シンボル，SFBIT 属性，SABIT 属性のビット・シンボル，EQU 疑似命令で定義されたビット・シンボル（オペランドに ADDRESS 属性のシンボルを含む場合のみ） 0H - FFFFFH - 上記以外のビットシンボル 0H - FFFFH
addr5	0080H - 00BFH（CALLT 命令テーブル領域／偶数値のみ）	
bit	3 ビット値　：0 - 7	
n	2 ビット値　：0 - 3	

注 1　RAM 空間にミラーされる領域のアドレス範囲は，デバイスによって異なります。詳細については，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。

注 2　sfr，2ndsfr をオペランドに記述したい場合は，!sfr，!2ndsfr の記述が可能です。これらは，!addr16 のオペランドとしてコードが出力されます。
2ndsfr は，“!”なしでも記述可能ですが，!addr16 のオペランドとしてコードが出力されます。

注 3　16 ビット・データの場合は，偶数アドレスのみとなります。

注 4　78K0 との互換性を保つため，数値定数，および NUMBER 属性のシンボルの場合のみ FE20H - FF1FH の範囲も記述可能となっています。

注 5 数値定数，および NUMBER 属性のシンボルの場合は，指定アドレスの SFR の書き込み／読み出しのアクセス・チェックを行いません。

【addr16，および word において，オペランドのシンボル属性によって取り得る範囲が異なる理由】

addr16，および word において，オペランドに記述されたシンボルの属性によって，そのオペランドの取り得る値の範囲は異なります。その理由について，次に示します。

なお，シンボルの属性については，「2.2.3 (4) シンボルの属性」を参照してください。

(1) !addr16（BR，CALL 以外の命令）

!addr16（BR，CALL 以外の命令）のオペランドで，数値定数，および NUMBER 属性のシンボルと ADDRESS 属性のシンボルで記述可能な範囲が異なっている理由について，＜例 1＞で説明します。

＜例 1＞

```
NUMBER0         EQU     0F100H                  ; (a)
NUMBER1         EQU     0F102H
NUMBER2         EQU     0F103H

D0              DSEG    AT  0FF100H
ADDRESS0:       DS      1
ADDRESS1:       DS      1
ADDRESS2:       DS      1

                CSEG
                MOV     !NUMBER0 , A            ; (b)
                MOV     !0F100H , A             ; (c)
                MOV     !ADDRESS0 , A           ; (d)
```

(a) は，NUMBER 属性のシンボルであり，このシンボルを !addr16 のオペランドに記述した場合について説明します。

「MOV !addr16 , A」の命令セットのオペランド !addr16 では，ダイレクト・アドレッシングが行われ，(b) では，「A レジスタの値を 0FF100H のアドレスへ転送する」という処理が行われます。(a) は，NUMBER 属性のシンボルであり，(c) と置き換えることが可能なので，!addr16 に記述した NUMBER 属性のシンボル NUMBER0，および数値 0F100H は，「0FF100H」のアドレスを指していることになります。

つまり，!addr16（BR，CALL 以外の命令）の「NUMBER 属性のシンボル」は「0H - FFFFH」の値を取ることができ，それは「F0000H - FFFFFH」のアドレスを指していることになります。

次に，ラベル ADDRESS0（ADDRESS 属性のシンボル）を使って，同様の処理を行う場合について説明します。

addr16 の範囲「0000H - FFFFH」に対して，(d) の ADDRESS0 のシンボル値は RAM 空間の「FxxxxH - FFFFFH」であるため，このままではエラーとなります。そこで，オペランドにラベル ADDRESS0（ADDRESS 属性のシンボル）を記述した場合は，オペランドの範囲を「F0000H - FFFFFH」とすることにより，記述の簡便化を図っています。

つまり，!addr16（BR，CALL 以外の命令）の「ADDRESS 属性のシンボル」は，「F0000H - FFFFFH」の値を取ることができ，そのままオペランドに記述することが可能です。

さらに，ROM 領域が RAM 領域にミラーされることに対して，!addr16 での対応が必要となります。

＜例 2 ＞において，MO のセグメントは，RAM 空間にミラーされる ROM 空間へ配置されます。MO のセグメントは，MAA=0 のときは「01000H - 0xxxxH」，MAA=1 のときは「11000H - 0xxxxH」へ配置されます。そのため，(e) の ADDRESS0 のシンボル値は「01000H - 0xxxxH，または 11000H - 1xxxxH」の値となります。このことから，ミラーされるセグメント中のシンボルを参照する (e) のような記述を可能とするため，!addr16 の範囲は「01000H - 0xxxxH，または 11000H - 1xxxxH」となっています。

つまり，!addr16（BR，CALL 以外の命令）の「ADDRESS 属性のシンボル」は，「01000H - 0xxxxH，または 11000H - 0xxxxH」の値も，そのままオペランドに記述することが可能です。

＜例 2 ＞

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

(2) ES:!addr16

ES:!addr16 のオペランドで，数値定数，および NUMBER 属性のシンボルと ADDRESS 属性のシンボルで記述可能な範囲が異なっている理由について，＜例 3 ＞で説明します。

＜例 3 ＞

```
DATA            CSEG    AT      12345H
ADDRESS0:       DB      12H
ADDRESS1:       DB      34H
ADDRESS2:       DB      56H

                CSEG
                MOV     ES , #HIGHW ADDRESS0    ; (f)
                MOV     A , ES:!ADDRESS0        ; (g)
```

(f)，(g) で「ADDRESS0 にあるデータを A レジスタへ転送する」という処理を行う場合について説明します。

addr16 の範囲「0000H - FFFFH」に対して，(g) の ADDRESS0 のシンボル値は「12345H」であるため，このままではエラーとなります。

そこで，ADDRESS0 の範囲を「0H - FFFFFH」とすることにより，(g) の記述を可能として，記述の簡便化を図っています。

つまり，ES:!addr16 のオペランドの「ADDRESS 属性のシンボル」は，「0H - FFFFFH」の値も，そのままオペランドに記述することが可能です。

(3) !addr16.bit，ES:!addr16.bit

!addr16.bit，ES:!addr16.bit のオペランドで，DBIT シンボル，SFBIT 属性，SABIT 属性のビット・シンボル，EQU 疑似命令で定義されたビット・シンボル（オペランドに ADDRESS 属性のシンボルを含む場合のみ）とそれ以外のシンボルで記述可能な範囲が異なっている理由について，＜例 4 ＞で説明します。

＜例 4 ＞

```
                BSEG
DBITSYM0        DBIT                            ; (h)
DBITSYM1        DBIT
DBITSYM2        DBIT

BIT1_PM0        EQU     PM0.1                   ; (i)
BIT2_P0         EQU     P0.2                    ; (j)

                DSEG
ADDRESS0:       DS      1
ADDRESS1:       DS      1
ADDRESS2:       DS      1

ADR_BIT0        EQU     ADDRESS0.0              ; (k)
ADR_BIT1        EQU     ADDRESS0.1
ADR_BIT2        EQU     ADDRESS0.2

                CSEG
                SET1    !DBITSYM0               ; (l)
                SET1    !BIT1_PM0               ; (m)
                SET1    !BIT2_P0                ; (n)
                SET1    !ADR_BIT0               ; (o)
```

(h) の DBIT シンボル，(i)，(j) の SFBIT 属性，SABIT 属性のビット・シンボル，(k) の EQU 疑似命令で定義されたビット・シンボル（オペランドに ADDRESS 属性のシンボルを含む場合のみ）を，(l) 〜 (o) のように !addr16.bit のオペランドにそのまま記述できるようにするため，シンボルの属性によって取り得る値の範囲が異なります。

同様の理由により，ES:!addr16.bit についても，シンボルの属性によって取り得る値の範囲が異なります。

(4) word

word のオペランドで，数値定数，および NUMBER 属性のシンボルと ADDRESS 属性のシンボルで記述可能な範囲が異なっている理由について，＜例 5 ＞で説明します。

＜例 5 ＞

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

word をオペランドとする word[B]，wor[C]，word[BC] では，(p) 〜 (r) のようにラベル（ADDRESS 属性のシンボル）を記述することが多いため，!addr16 と同様，ラベルでの記述を可能とすることで，記述の簡便化を図っています。
同様の理由により，ES:word[B]，ES:wor[C]，ES:word[BC] についても，記述の簡便化を図っています。

表 2-12 疑似命令のオペランド値の範囲

種類	疑似命令	値の範囲
セグメント定義	CSEG AT	0H - FFFFFH（SFR，2ndSFR を除く）
	DSEG AT	0H - FFFFFH（SFR，2ndSFR を除く）
	BSEG AT	0H - FFFFFH（SFR，2ndSFR を除く）
	ORG	0H - FFFFFH（SFR，2ndSFR を除く）
シンボル定義	EQU	20 ビット値　0H - FFFFFH
	SET	20 ビット値　0H - FFFFFH
メモリ初期化領域確保	DB	8 ビット値　0H - FFH
	DW	16 ビット値　0H - FFFFH
	DG	20 ビット値　0H - FFFFFH
	DS	16 ビット値　0H - FFFFH
分岐命令自動選択	BR/CALL	0H - FFFFFH

2.15.2 命令の要求するオペランドのサイズ

命令には，機械命令と疑似命令がありますが，オペランドとしてイミーディエト・データ，またはシンボルを要求する命令については，各命令により要求するオペランドのサイズが異なります。したがって，命令の要求するオペランドのサイズ以上のデータを記述すると，エラーとなります。

なお，式の演算は，符号なし，32 ビットで行います。評価結果が 0FFFFFFFFH（32 ビット）を越えた場合にはワーニング・メッセージが出力されます。

ただし，オペランドにリロケータブルなシンボル，または外部シンボルを記述した場合は，アセンブラ内では値が決定されないので，リンカにおいて値の決定と範囲のチェックが行われます。

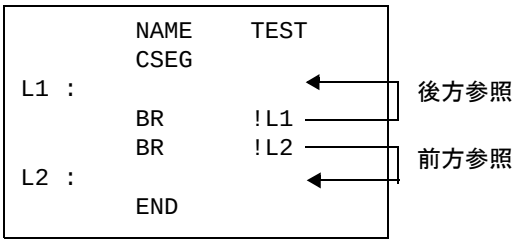
2.15.3 オペランドのシンボル属性，リロケーション属性

命令のオペランドとしてネーム，ラベル，$（ロケーション・カウンタを示す）を記述する場合は，それらの式の項としてのシンボル属性，リロケーション属性（「2.12 演算の制限」を参照），また，ネーム，ラベルの場合は，その参照方向の条件により，オペランドとして記述可能かどうかが異なります。

ネーム，ラベルの参照方向には，後方参照と前方参照があります。

- 後方参照：オペランドとして参照するネーム，ラベルがそれ以前の行で定義されている。
- 前方参照：オペランドとして参照するネーム，ラベルがそれ以降の行で定義されている。

＜例＞



次に，シンボル属性，リロケーション属性，ネーム，ラベルの参照方向の条件を示します。

表 2-13 オペランドとして記述可能なシンボルの性質

	シンボル属性	NUMBER		ADDRESS				NUMBER ADDRESS		sfr 予約語[注 1]	
	リロケーション属性	アブソリュート項		アブソリュート項		リロケータブル項		外部参照項			
	参照パターン	後方	前方	後方	前方	後方	前方	後方	前方	sfr	2ndsfr
記述形式	byte	○	○	○	○	○	○	○	○	×	×
	word	○	○	○	○	○	○	○	○	×	×
	saddr	○	○	○	○	○	○	○	○	○[注 3]	×
	saddrp	○	○	○	○	○	○	○	○	○[注 2,4]	×
	sfr	○	○	×	×	×	×	×	×	○[注 2,5]	×
	sfrp	○	○	×	×	×	×	×	×	○[注 2,6]	×
	addr20	○	○	○	○	○	○	○	○	×	×
	addr16	○	○	○	○	○	○	○	○	○[注 7]	○[注 7]
	addr5	○	○	○	○	○	○	○	○	×	×
	bit	○	○	×	×	×	×	×	×	×	×
	n	○	○	○	○	×	×	×	×	×	×

前方 ： 前方参照

後方 ： 後方参照

○ ： 記述可能

× ： エラー

ー ： 記述不可能

注 1 EQU 疑似命令のオペランドに，sfr，sfrp（saddr と sfr がオーバーラップしていない領域の sfr）を指定し，定義されたシンボルは後方参照のみとし，前方参照は禁止します。

注 2 オペランドの組み合わせに，saddr/saddrp を sfr/sfrp に入れ替えた組み合わせが存在する命令に対し，saddr 領域の sfr 予約語を記述した場合は，saddr/saddrp としてコードが出力されます。

注 3 saddr 領域の sfr 予約語

注 4 saddr 領域の sfrp 予約語

注 5 8 ビット・アクセス可能な sfr 予約語のみ

注 6 16 ビット・アクセス可能な sfr 予約語のみ

注 7 BR，CALL 以外の命令のオペランド !addr16 でのみ，!sfr，!2ndsfr の指定が可能です。

表 2-14　疑似命令のオペランドとして記述可能なシンボルの性質

	シンボル属性		NUMBER		ADDRESS，SADDR						BIT					
	リロケーション属性		アブソリュート項		アブソリュート項		リロケータブル項		外部参照項		アブソリュート項		リロケータブル項		外部参照項	
	参照方向		後方	前方	後方	前方	後方	前方	後方	前方	後方	前方	後方	前方	後方	前方
疑似命令	ORG		○ 注 1	－	－	－	－	－	－	－	－	－	－	－	－	－
	EQU 注 2		○	－	○	－	○ 注 3	－	－	－	○	－	○ 注 3	－	－	－
	SET		○ 注 1	－	－	－	－	－	－	－	－	－	－	－	－	－
	DB	サイズ	○ 注 1	－	－	－	－	－	－	－	－	－	－	－	－	－
		初期値	○	○	○	○	○	○	○	○	－	－	－	－	－	－
	DW	サイズ	○ 注 1	－	－	－	－	－	－	－	－	－	－	－	－	－
		初期値	○	○	○	○	○	○	○	○	－	－	－	－	－	－
	DG	サイズ	○ 注 1	－	－	－	－	－	－	－	－	－	－	－	－	－
		初期値	○	○	○	○	○	○	○	○	－	－	－	－	－	－
	DS		○ 注 4	－	－	－	－	－	－	－	－	－	－	－	－	－
	BR/CALL		○	－	－	－	－	－	－	－	－	－	－	－	－	－

○　：記述可能

－　：記述不可能

注 1　絶対式のみが記述可能です。

注 2　次のパターンを含む式を記述するとエラーとなります。

- ADDRESS 属性－ ADDRESS 属性
- ADDRESS 属性　比較演算子　ADDRESS 属性
- HIGH　アブソリュートな ADDRESS 属性
- LOW　アブソリュートな ADDRESS 属性
- HIGHW　アブソリュートな ADDRESS 属性
- LOWW　アブソリュートな ADDRESS 属性
- DATAPOS　アブソリュートな ADDRESS 属性
- MASK　アブソリュートな ADDRESS 属性
- 上記の 8 つで，演算結果が最適化の影響を受ける可能性がある場合

注 3　リロケータブルな項をオペランドに持つ HIGH/LOW/HIGHW/LOWW/DATAPOS/MASK 演算子によってできた項は許されません。

注 4　「3.4 メモリ初期化，領域確保疑似命令」を参照してください。

第 3 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，RA78K0R 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。

3.1 概要

インストラクションは，アセンブルの結果，オブジェクト・コード（機械語）に変換されますが，疑似命令は，原則としてオブジェクト・コードに変換されません。

疑似命令は，主に次の機能を持ちます。

- ソースの記述を容易にします。
- メモリの初期化や領域の確保を行います。
- アセンブラ，リンカがその処理を行うために必要となる情報を与えます。

次に，疑似命令の種類を示します。

表 3-1 疑似命令一覧

種類	疑似命令
セグメント定義疑似命令	CSEG，DSEG，BSEG，ORG
シンボル定義疑似命令	EQU，SET
メモリ初期化，領域確保疑似命令	DB，DW，DG，DS，DBIT
リンケージ疑似命令	EXTRN，EXTBIT，PUBLIC
オブジェクト・モジュール名宣言疑似命令	NAME
分岐命令自動選択疑似命令	BR，CALL
マクロ疑似命令	MACRO，LOCAL，REPT，IRP，EXITM，ENDM
アセンブル終了疑似命令	END

以降，各疑似命令について詳細な説明を行います。

説明の中で，[ ] は大かっこの中が省略可能であることを，… は同一の形式を繰り返すことを示します。

3.2 セグメント定義疑似命令

ソース・モジュールは，セグメント単位に分割して記述します。
この“セグメント”を定義するのが，セグメント定義疑似命令です。

セグメントには，次の 4 種類があります。

- コード・セグメント
- データ・セグメント
- ビット・セグメント
- アブソリュート・セグメント

セグメントの種類により，メモリのどの範囲に配置されるかが決まります。
次に，各セグメントの定義方法と配置されるメモリ・アドレスを示します。

表 3-2 セグメントの定義方法と配置されるメモリ・アドレス

セグメントの種類	定義方法	配置されるメモリ・アドレス
コード・セグメント	CSEG 疑似命令	内部，または外部の ROM アドレス内
データ・セグメント	DSEG 疑似命令	内部，または外部の RAM アドレス内
ビット・セグメント	BSEG 疑似命令	内部 RAM の saddr 領域内
アブソリュート・セグメント	CSEG，DSEG，BSEG 疑似命令で再配置属性に配置アドレス（AT 配置アドレス）を指示する	指定したアドレス

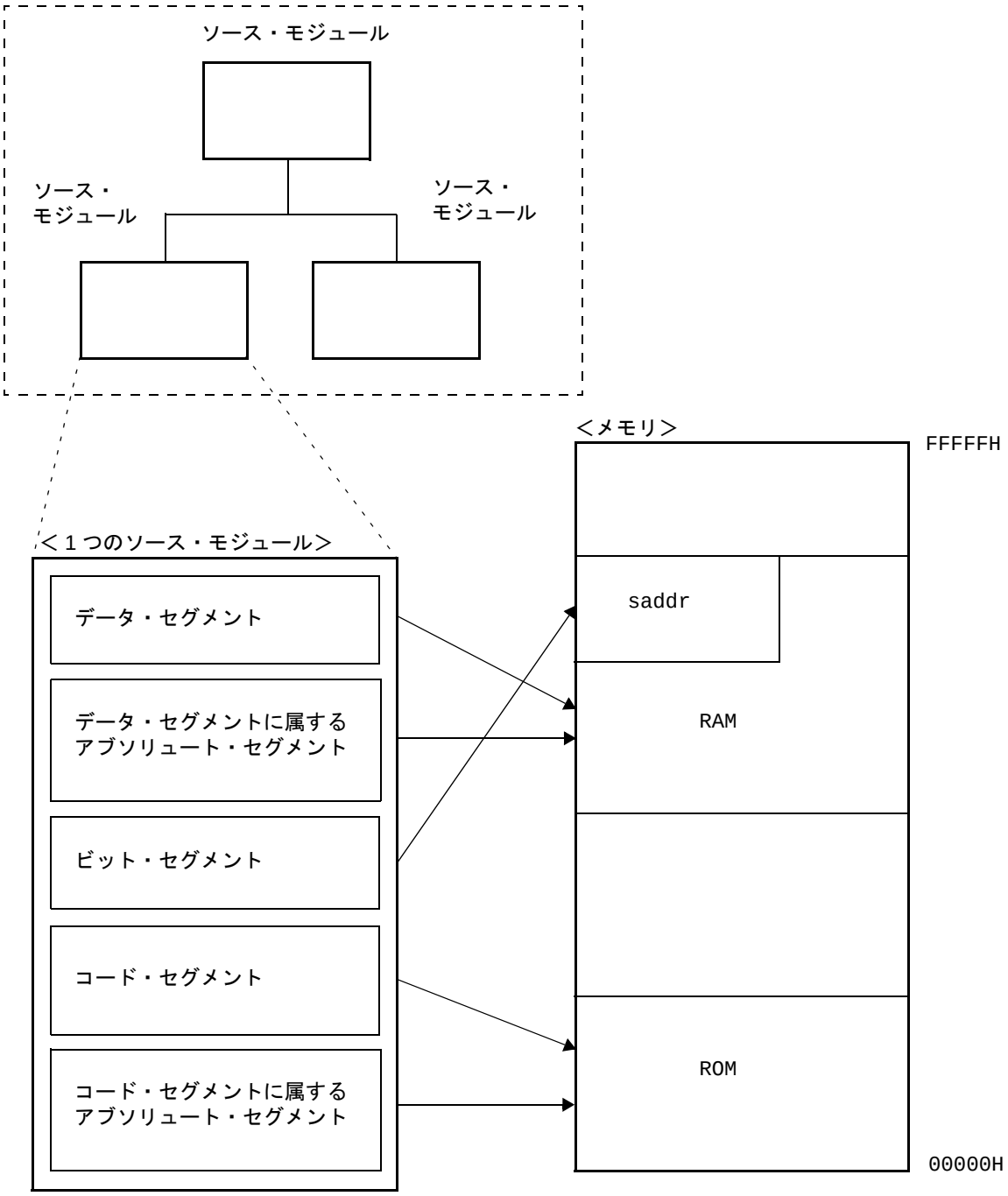
メモリの配置アドレスをユーザが決定したい場合には，アブソリュート・セグメントを記述します。スタック領域は，ユーザがデータ・セグメント内に領域を確保し，スタック・ポインタに設定する必要があります。

また，以下の領域には，セグメントを配置することはできません。

オプション・バイト領域	C0 ～ C2H（ユーザ・オプション・バイト），C3H（オンチップ・デバッグ・オプション・バイト）
セキュリティ ID を指定する場合	C4H ～ CDH 番地
オンチップ・デバッグ機能を使用する場合	02H ～ 03H，CE ～ D7H（オンチップ・デバッグ用） ユーザが -go オプションで指定するスタート・アドレスからプログラム・サイズ分の領域

セグメントの配置の例を，次に示します。

図 3-1 セグメントのメモリ配置



セグメント定義疑似命令には，次のものがあります。

- CSEG
- DSEG
- BSEG
- ORG

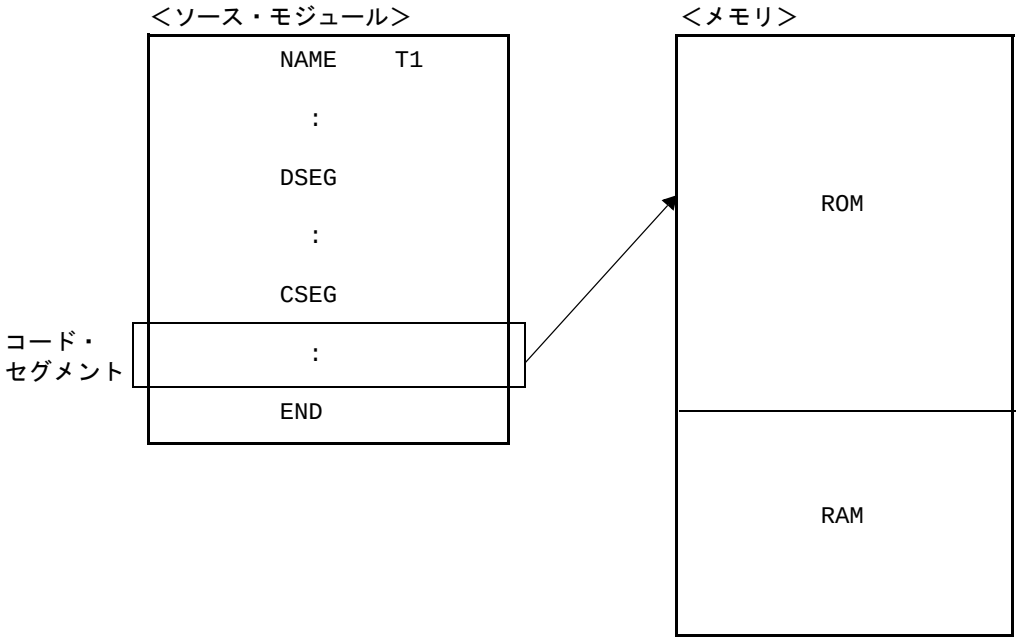
CSEG

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*セグメント名*]	CSEG	[*再配置属性*]	[; *コメント*]

【機能】

- CSEG 疑似命令は，アセンブラにコード・セグメントの開始を指示します。
- CSEG 疑似命令以降に記述した命令は，再びセグメント定義疑似命令（CSEG，DSEG，BSEG，ORG），または END 疑似命令が現れるまでコード・セグメントに属し，最終的に機械語に変換された時点で ROM アドレス内に配置されます。



【用途】

- CSEG 疑似命令で定義するコード・セグメントには，インストラクションや DB，DW 疑似命令等を記述します。
  ただし，そのセグメントを固定アドレスから配置する場合には，再配置属性欄に“AT 絶対式”を記述してください。
- サブルーチンなどの 1 つの機能を持つ単位の記述は，1 つのコード・セグメントとして定義します。
  その規模が比較的大きい場合や，そのサブルーチンに高い汎用性（ほかのプログラム開発にも流用できる）がある場合には，1 つのモジュールとして定義することをお勧めします。

【説明】

- コード・セグメントの開始アドレスは，ORG 疑似命令により指定できます。

  また，再配置属性欄に“AT 絶対式”を記述することによって，開始アドレスを指定することもできます。

- 再配置属性とは，セグメントの配置アドレスの範囲を限定するものです。

  次に，CSEG の再配置属性を示します。

表 3-3 CSEG の再配置属性

再配置属性	記述形式	説明
CALLT0	CALLT0	指定セグメントを 00080H - 000BFH 番地内で先頭が偶数番地になるように配置します。
FIXED	FIXED	指定セグメントを 000C0H - 0FFFFH 番地に配置します。
BASE	BASE	指定セグメントを 000C0H - 0FFFFH に配置します。
AT	AT 絶対式	指定セグメントを絶対番地に配置します（SFR，2ndSFR を除く）。
UNIT	UNIT	指定セグメントを任意の位置へ配置します（メモリ領域名“ROM”内 000C0H - EFFFFH）。
UNITP	UNITP	指定セグメントを任意の位置へ，先頭が偶数番地になるように配置します（メモリ領域名“ROM”内 000C0H - EFFFFH）。
IXRAM	IXRAM	指定セグメントを任意の位置へ配置します（メモリ領域名“ROM”内 000C0H - EFFFFH）。
SECUR_ID	SECUR_ID	セキュリティ ID 指定専用の属性です。セキュリティ ID 以外は指定しないでください。 指定セグメントを 000C4H - 000CDH 番地へ配置します。
PAGE64KP	PAGE64KP	64K 境界にまたがらない，先頭が偶数番地となるように配置します。 異なるファイルの同名セグメントは結合しません。
UNIT64KP	UNIT64KP	64K 境界にまたがらない，先頭が偶数番地となるように配置します。 同名セグメントは結合します。
MIRRORP	MIRRORP	MAA=0 のときに RAM 空間にミラーされる領域（01000H - 0xxxxH），または MAA=1 のときに RAM 空間にミラーされる領域（11000H - 1xxxxH）のいずれかの領域に配置します。[注]
OPT_BYTE	OPT_BYTE	ユーザ・オプション・バイト，およびオンチップ・デバッグ指定専用の属性です。ユーザ・オプション・バイト，およびオンチップ・デバッグ以外は指定しないでください。 指定セグメントを 000C0H - 000C3H 番地へ配置します。

注 RAM 空間にミラーされる領域のアドレス範囲は，デバイスによって異なります。

- 再配置属性を省略した場合，“UNIT”と解釈されます。
- 表 3-3 以外の再配置属性を指定した場合は，アセンブラはエラーを出力し，“UNIT”が指定されたものとみなします。また，各セグメントのサイズが領域のサイズを越えた場合には，エラーとなります。

- 再配置属性 AT で不当な絶対式を指定すると，アセンブラはエラーを出力し，絶対式の値を 0 とみなし，処理を続けます。
- CSEG 疑似命令のシンボル欄にセグメント名を記述することにより，そのコード・セグメントにネーム（名前）を付けることができます。セグメント名が省略されたコード・セグメントには，アセンブラが自動的にデフォルトのセグメント名を与えます。
  次に，CSEG のデフォルト・セグメント名を示します。

再配置属性	デフォルト・セグメント名
CALLT0	?CSEGT0
FIXED	?CSEGFX
UNIT（または省略時）	?CSEG
UNITP	?CSEGUP
IXRAM	?CSEGIX
BASE	?CSEGB
SECUR_ID	?CSEGSI
PAGE64KP	?CSEGP64
UNIT64KP	?CSEGU64
MIRRORP	?CSEGMIP
OPT_BYTE	?CSEGOB0
AT	セグメント名省略不可

- 再配置属性が AT の場合，セグメント名を省略するとエラーとなります。
- 再配置属性が同一であれば，複数のコード・セグメントに同一のセグメント名を与えることができます（ただし，AT の場合は同名セグメントは許されません）。
  これらは，アセンブラ内部で 1 つのセグメントとして処理されます。同名セグメントの再配置属性が異なる場合には，エラーとなります。したがって，再配置属性ごとの同名セグメントの数は 1 つです。
- コード・セグメントは分割記述が可能です。つまり，同一モジュール内に記述された同一再配置属性，同一セグメント名のコード・セグメントは，アセンブラ内部で連続したひとつのセグメントとして処理されます。

注意 1 再配置属性が AT の場合は，分割記述はできません。

注意 2 再配置属性が CALLT0 のアドレスが偶数となるように，必要ならば 1 バイトの間隙をおいてください。

- 別モジュール間での同名セグメントは，UNIT，CALLT0，FIXED，UNITP，BASE，PAGE64KP，UNIT64KP，MIRRORP，SECUR_ID の場合のみ記述することができ，リンク時に連続した 1 つのセグメントとして結合されます。
- セグメント名は，シンボルとして参照できません。
- アセンブラの出力するセグメントの総数は，ORG 疑似命令によるセグメントをあわせて，別名セグメントが 256 個までです。同名セグメントは 1 つと数えます。

- セグメント名の最大認識文字数は，8 文字です。
- セグメント名の大文字，小文字は区別されます。
- OPT_BYTE で，ユーザ・オプション・バイト，およびオンチップ・デバッグを指定します。
  ユーザ・オプション・バイト指定機能を持つデバイスに対して，ユーザ・オプション・バイトを指定していない場合には，各番地に“?CSEGOB0”というデフォルト・セグメントが定義され，デバイス・ファイルから読み込んだ初期値が設定されます。

【使用例】

```
        NAME    SAMP1
C1      CSEG            ; (1)

C2      CSEG    CALLT0  ; (2)

        CSEG    FIXED   ; (3)

C1      CSEG    CALLT0  ; (4)   ←エラー

        CSEG            ; (5)

        END
```

(1) セグメント名が“C1”，再配置属性が“UNIT”と解釈されます。

(2) セグメント名が“C2”，再配置属性が“CALLT0”と解釈されます。

(3) セグメント名が“?CSEGFX”，再配置属性が“FIXED”と解釈されます。

(4) (1) でセグメント名“C1”は再配置属性“UNIT”として定義されているので，エラーとなります。

(5) セグメント名が“?CSEG”，再配置属性が“UNIT”と解釈されます。

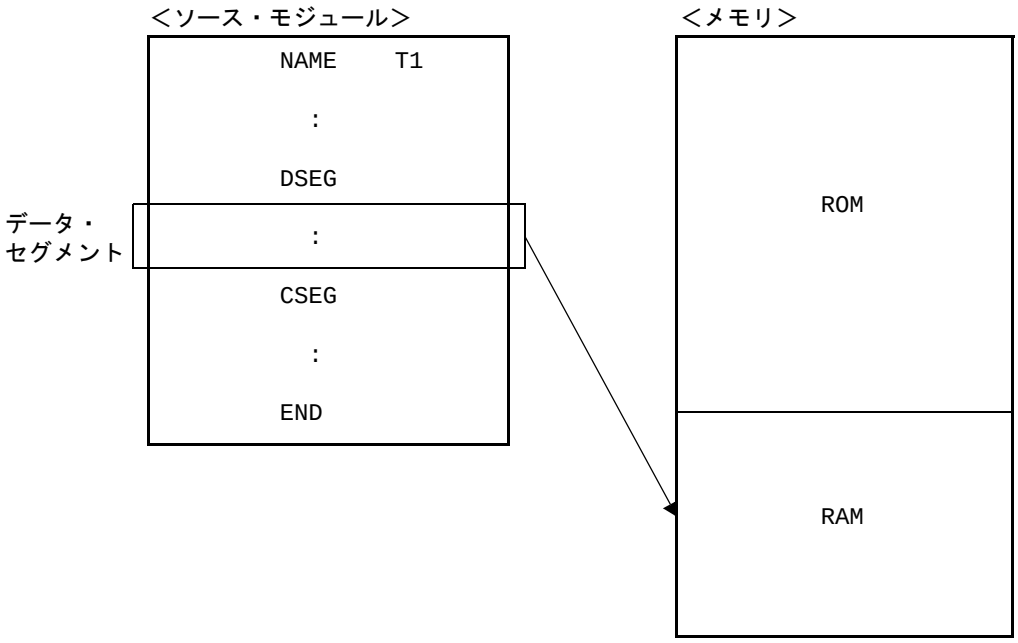
DSEG

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*セグメント名*]	DSEG	[*再配置属性*]	[; *コメント*]

【機能】

- DSEG 疑似命令は，アセンブラにデータ・セグメントの開始を指示します。
- DSEG 疑似命令以降，再びセグメント定義疑似命令（CSEG，DSEG，BSEG，ORG），または END 疑似命令が現れるまでに DS 疑似命令により定義されたメモリ領域は，データ・セグメントに属し，最終的に，RAM アドレス内に確保されます。



【用途】

- DSEG 疑似命令で定義するデータ・セグメントには，主に DS 疑似命令を記述します。
  データ・セグメントは，RAM 内に配置されます。したがって，データ・セグメント内にインストラクションを記述することはできません。
- データ・セグメントでは，プログラムで使用する RAM の作業領域を DS 疑似命令により確保し，それぞれの作業領域のアドレスにラベルを付けます。プログラムを記述する場合，このラベルを利用します。
  データ・セグメントとして確保された領域は，RAM 上でほかの作業領域（スタック領域，ほかのモジュールで定義された作業領域など）と重複しないよう，リンカにより配置されます。

汎用レジスタ領域と重複する場合，リンカはワーニング・メッセージを出力します。このワーニング・メッセージの出力レベルは，ワーニング・メッセージ出力指定オプション（-w）により切り替えることができます。

-w の値	チェック対象
0	チェックしない
1	RB0
2	RB0 ～ RB3

【説明】

- データ・セグメントの開始アドレスは，ORG 疑似命令により指定できます。

  また，再配置属性欄に“AT 絶対式”を記述することによって，開始アドレスを指定することもできます。

- 再配置属性とは，データ・セグメントの配置アドレスの範囲を限定するものです。

  次に，DESG の再配置属性を示します。

表 3-4 DSEG の再配置属性

再配置属性	記述形式	説明
SADDR	SADDR	指定セグメントを saddr 領域に配置します（saddr 領域：FFE20H - FFEFFH）。
SADDRP	SADDRP	指定セグメントを saddr 領域に先頭が偶数番地となるように配置します（saddr 領域：FFE20H - FFEFFH）。
AT	AT 絶対式	指定セグメントを絶対番地に配置します（SFR，2ndSFR を除く）。
UNIT	UNIT，または指定なし	指定セグメントを内部または外部の任意の位置へ配置します（メモリ領域名“RAM”内）。
UNITP	UNITP	指定セグメントを内部または外部の任意の位置へ，偶数番地から配置します（メモリ領域名“RAM”内）。
BASEP	BASEP	指定セグメントを内部 RAM 領域に先頭が偶数番地となるように配置します（ただし，saddr 領域（FxxxxH - FFEFFH）は含みません）。[注] ES 参照なしでアクセスしたいデータを配置するときに使用します。
PAGE64KP	PAGE64KP	メモリ領域名“RAM”内に，64K 境界にまたがらない，先頭が偶数番地となるように配置します。 異なるファイルの同名セグメントは結合しません。
UNIT64KP	UNIT64KP	メモリ領域名“RAM”内に，64K 境界にまたがらない，先頭が偶数番地となるように配置します。 同名セグメントは結合します。

注 xxxx に当てはまるアドレスは，デバイスに依存します。

- 78K0 用アセンブラでの再配置属性も記述可能で，“UNIT”と同様の配置を行います。

  次に，78K0 用 DSEG の再配置属性を示します。

再配置属性	記述形式
IHRAM	IHRAM
LRAM	LRAM
DSPRAM	DSPRAM
IXRAM	IXRAM

- 再配置属性が省略された場合，“UNIT”と解釈されます。
- 表 3-4 以外の再配置属性が指定された場合には，アセンブラはエラーを出力し，“UNIT”が指定されたものとみなします。また，各セグメントのサイズが領域のサイズを越えた場合には，エラーとなります。
- 再配置属性 AT で不当な絶対式を指定すると，アセンブラはエラーを出力し，絶対式の値を 0 とみなし，処理を続けます。
- データ・セグメント中に機械語命令（BR 疑似命令も含む）は記述できません。記述した場合はエラーとなり，その行は無視されます。
- DSEG 疑似命令のシンボル欄にセグメント名を記述することにより，そのデータ・セグメントにネーム（名前）を付けることができます。セグメント名が省略されたデータ・セグメントには，アセンブラが自動的にデフォルトのセグメント名を与えます。
  次に，DSEG のデフォルト・セグメント名を示します。

再配置属性	デフォルト・セグメント名
SADDR	?DSEGS
SADDRP	?DSEGSP
UNIT（または省略時）	?DSEG
UNITP	?DSEGUP
IHRAM	?DSEGIH
LRAM	?DSEGL
DSPRAM	?DSEGDSP
IXRAM	?DSEGIX
BASEP	?DSEGBP
PAGE64KP	?DSEGP64
UNIT64KP	?DSEGU64
AT	セグメント名省略不可

- 再配置属性が同一であれば，複数のデータ・セグメントに同一のセグメント名を与えることができます（ただし，AT の場合は同名セグメントは許されません）。
  これらは，アセンブラ内部で 1 つのセグメントとして処理されます。
- データ・セグメントは分割記述が可能です。つまり，同一モジュール内に記述された同一再配置属性，同一セグメント名のデータ・セグメントは，アセンブラ内部で連続したひとつのセグメントとして処理されます。

注意 1 再配置属性が AT の場合は，分割記述はできません。

注意 2 再配置属性が SADDR の場合は，DESG 疑似命令を記述した直後のアドレスが偶数となるように，必要ならば 1 バイトの間隙をおいてください。

- 再配置属性が SADDRP の場合，DSEG 疑似命令を記述した直後のアドレスが 2 の倍数になるように配置されます。
- 同名セグメントの再配置属性が異なる場合には，エラーとなります。したがって，再配置属性ごとの同名セグメントの数は 1 つです。
- 別モジュール間での同名セグメントは，UNIT，UNITP，SADDR，SADDRP，LRAM，IHRAM，DSPRAM，IXRAM，BASEP，PAGE64KP，UNIT64KP の場合のみ記述することができ，リンク時に連続した 1 つのセグメントとして結合されます。
- セグメント名はシンボルとして参照できません。
- アセンブラの出力するセグメントの総数は，ORG 疑似命令によるセグメントをあわせて，別名セグメントが 255 個までです。同名セグメントは 1 つと数えます。
- セグメント名の最大認識文字数は，8 文字です。
- セグメント名の大文字，小文字は区別されます。

【使用例】

```
        NAME    SAMP1
        DSEG                    ; (1)
WORK1 : DS      2
WORK2 : DS      1
        CSEG
        MOV     A , !WORK2      ; (2)
        MOV     A , WORK2       ; (3)   ←エラー
        MOVW    DE , #WORK1     ; (4)
        MOVW    AX , WORK1      ; (5)   ←エラー

        END
```

(1) DSEG 疑似命令により，データ・セグメントの開始を定義します。
再配置属性が省略されたので“UNIT”と解釈されます。デフォルトのセグメント名は“?DSEG”です。

(2) この記述は，“MOV A , !addr16”に該当します。

(3) この記述は，“MOV A , saddr”に該当します。
リロケータブルなラベル“WORK2”は“saddr”としては記述できません。したがって，(3) の記述はエラーです。

(4) この記述は，“MOVW rp , #word”に該当します。

(5) この記述は，“MOVW AX , saddrp”に該当します。
リロケータブルなラベル“WORK1”は“saddrp”としては記述できません。したがって，(5) の記述はエラーです。

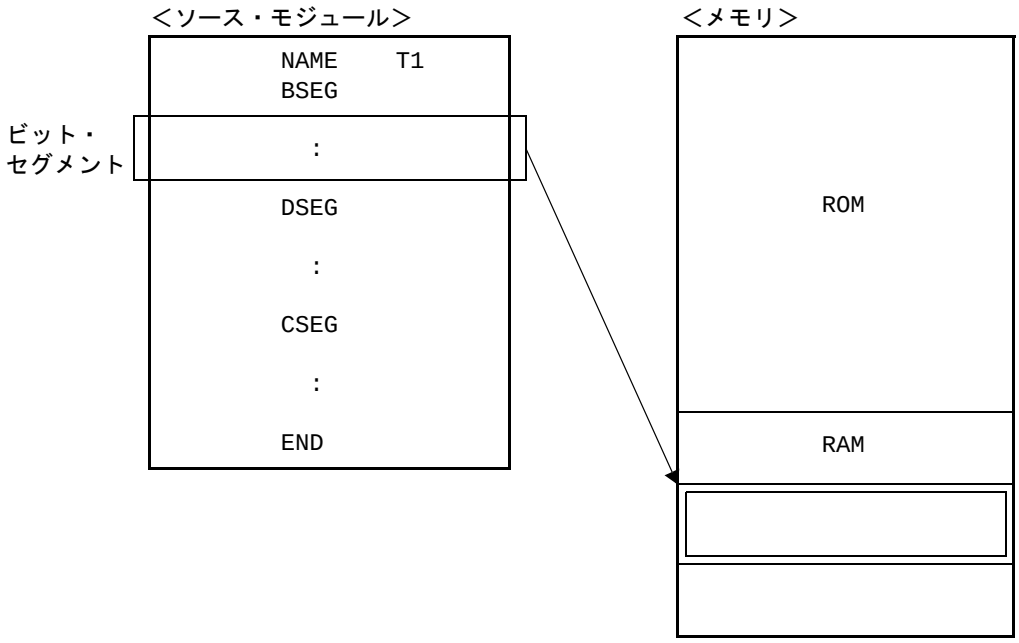
BSEG

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*セグメント名*]	BSEG	[*再配置属性*]	[; *コメント*]

【機能】

- BSEG 疑似命令は，アセンブラにビット・セグメントの開始を指示します。
- ビット・セグメントは，ソース・モジュール中で使用する RAM アドレスの定義を行うセグメントです。
- BSEG 疑似命令以降，再びセグメント定義疑似命令（CSEG，DSEG，BSEG），または END 疑似命令が現れるまでに DBIT 疑似命令により定義されたメモリ領域は，ビット・セグメントに属します。



【用途】

- BSEG 疑似命令で定義するビット・セグメントには，DBIT 疑似命令を記述します。
- ビット・セグメント内にインストラクションを記述することはできません。

【説明】

- ビット・セグメントの開始アドレスは，再配置属性欄に“AT 絶対式”を記述することによって指定することができます。
- 再配置属性とは，ビット・セグメントの配置アドレスの範囲を限定するものです。

  次に，BSEG の再配置属性を示します。

表 3-5 BSEG の再配置属性

再配置属性	記述形式	説明
AT	AT 絶対式	指定セグメントの先頭を絶対番地の 0 ビット目に配置します。ビット単位で指定することはできません（00000H - FFFFFH）（SFR，2ndSFR を除く）。
UNIT	UNIT，または指定なし	指定セグメントを任意の位置へ配置します（FFE20H - FFEFFH）。

- 再配置属性を省略した場合，“UNIT”と解釈されます。
- 表 3-5 以外の再配置属性が指定された場合には，アセンブラはエラーを出力し，“UNIT”が指定されたものとみなします。また，各セグメントのサイズが領域のサイズを越えた場合には，エラーとなります。
- アセンブラ，リンカでは，ビット・セグメント内のロケーション・カウンタを“0xxxxx.b”の形式で表示します（バイト・アドレスは 16 進 5 桁，ビット位置は 16 進 1 桁（0 - 7））。

＜アブソリュート＞

バイト・アドレス	ビット位置							
	0	1	2	3	4	5	6	7
0FFE20H	0FFE20H.0	0FFE20H.1	0FFE20H.2	0FFE20H.3	0FFE20H.4	0FFE20H.5	0FFE20H.6	0FFE20H.7
0FFE21H	0FFE21H.0	0FFE21H.1	0FFE21H.2	0FFE21H.3	0FFE21H.4	0FFE21H.5	0FFE21H.6	0FFE21H.7

＜リロケータブル＞

バイト・アドレス	ビット位置							
	0	1	2	3	4	5	6	7
0H	0H.0	0H.1	0H.2	0H.3	0H.4	0H.5	0H.6	0H.7
1H	1H.0	1H.1	1H.2	1H.3	1H.4	1H.5	1H.6	1H.7

備考 リロケータブルなビット・セグメント中でのバイト・アドレスは，セグメントの先頭からのバイト単位のオフセットを指定します。
なお，オブジェクト・コンバータが出力するシンボル・テーブルでは，ビットの定義を行う領域の先頭からのビット・オフセットで表示，出力されます。

シンボル値	ビット・オフセット
0FFE20H.0	0000
0FFE20H.1	0001
0FFE20H.2	0002
:	:
0FFE20H.7	0007
0FFE21H.0	0008
0FFE21H.1	0009
:	:
0FFE80H.0	0300
:	:

- 再配置属性 AT で不当な絶対式を指定すると，アセンブラはエラーを出力し，絶対式の値を 0 とみなし，処理を続けます。
- BSEG 疑似命令のシンボル欄にセグメント名を記述することにより，そのビット・セグメントにネーム（名前）を付けることができます。セグメント名が省略されたビット・セグメントには，アセンブラが自動的にデフォルトのセグメント名を与えます。
次に，BSEG のデフォルト・セグメント名を示します。

再配置属性	デフォルト・セグメント名
UNIT（または省略時）	?BSEG
AT	セグメント名省略不可

- 再配置属性が UNIT であれば，複数のデータ・セグメントに同一のセグメント名を与えることができます（ただし，AT の場合に同名セグメントは許されません）。
これらは，アセンブラ内部で 1 つのセグメントとして処理されます。したがって，再配置属性ごとの同名セグメントの数は 1 つです。
- 同じセグメント名のビット・セグメント同士は，同一の再配置属性 UNIT（AT の場合は同名セグメントは禁止）でなければなりません。
- 同一モジュール内に記述された同一セグメント名の再配置属性が UNIT ではないときは，エラーとなり，その行は無視されます。
- 別モジュール間での同名セグメントは，リンク時に連続した 1 つのセグメントとして結合されます。結合は，ビット単位で行われます。
- セグメント名は，シンボルとして参照できません。

- ビット・セグメントは，リンカにより 0H - FFFFFH に配置されます。
- ビット・セグメント内でラベルを記述することはできません。
- ビット・セグメント内で記述できる命令は，DBIT 疑似命令，EQU，SET，PUBLIC，EXTBIT，EXTRN，MACRO，REPT，IRP，ENDM 疑似命令，およびマクロ定義とマクロ参照のみです。これ以外のものが記述された場合には，エラーとなります。
- アセンブラの出力するセグメントの総数は，ORG 疑似命令によるセグメントをあわせて，別名セグメントが 256 個までです。同名セグメントは 1 つと数えます。
- セグメント名の最大認識文字数は，8 文字です。
- セグメント名の大文字，小文字は区別されます。

【使用例】

```
        NAME    SAMP1

FLAG    EQU     0FFE20H
FLAG0   EQU     FLAG.0          ; (1)
FLAG1   EQU     FLAG.1          ; (2)

        BSEG                    ; (3)
FLAG2   DBIT

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

        END
```

(1) バイト・アドレス境界を意識して，ビット・アドレス（0FFE20H のビット 0）を定義しています。

(2) バイト・アドレス境界を意識して，ビット・アドレス（0FFE20H のビット 1）を定義しています。

(3) BSEG 疑似命令により，ビット・セグメントを定義します。再配置属性が省略されているので，アセンブラは，再配置属性が“UNIT”，セグメント名が“?BSEG”と解釈します。
ビット・セグメント内では，DBIT 疑似命令により，ビット作業領域を 1 ビットごとに定義します。
ビット・セグメントは，モジュール・ボディのはじめの方に記述します。
ビット・セグメント内で定義したビット・アドレス FLAG2 は，バイト・アドレス境界を意識しないで配置されます。

(4) この記述は，“SET1 FLAG.0”と書き換えられます。ここで，FLAG は，バイト・アドレスを示します。

(5) この記述では，バイト・アドレスが意識されません。

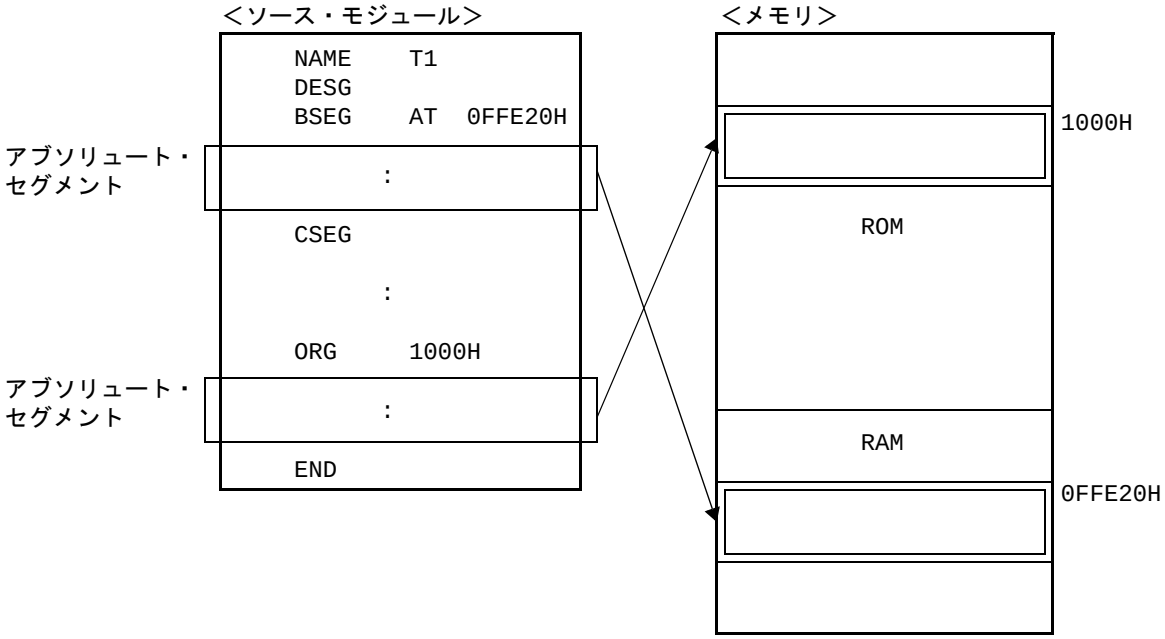
ORG

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*セグメント名*]	ORG	[*再配置属性*]	[; *コメント*]

【機能】

- ロケーション・カウンタに，オペランドで指定した式の値を設定します。
- ORG 疑似命令以降，再びセグメント定義疑似命令（CSEG，DSEG，BSEG，ORG），または END 疑似命令が現れるまでに記述された命令や，確保されたメモリ領域は，アブソリュート・セグメントに属し，オペランドで指定したアドレスから配置されます。



【用途】

- コード・セグメント，データ・セグメントを特定のアドレスから配置させる場合に，ORG 疑似命令を指定します。

【説明】

- ORG 疑似命令により定義されたアブソリュート・セグメントは，その直前に CSEG，DSEG 疑似命令により定義されたコード・セグメント，またはデータ・セグメントに属します。
  データ・セグメントに属するアブソリュート・セグメント内では，インストラクションを記述することはできません。また，ビット・セグメントに属するアブソリュート・セグメントを記述することはできません。

- ORG 疑似命令により定義されたコード，データ・セグメントは，再配置属性 AT のコード，データ・セグメントとして解釈されます。
- ORG 疑似命令のシンボル欄に，セグメント名を記述することにより，そのアブソリュート・セグメントにネームを付けることができます。
  セグメント名の最大認識文字数は 8 文字です。
- ORG 疑似命令によって定義されたモジュール内の同名セグメントの取り扱いは，CSEG 疑似命令，および DESG 疑似命令の AT 属性のセグメントと同一とします。
- ORG 疑似命令によって定義された別モジュール間の同名セグメントの取り扱いは，CSEG 疑似命令，および DESG 疑似命令の AT 属性のセグメントと同一とします。
- セグメント名が省略されたアブソリュート・セグメントには，アセンブラが自動的に“?A0nnnnn”というセグメント名を与えます。nnnnn は指定されたセグメントの先頭アドレスで，00000 - FFFFF（16 進 5 桁）が入ります。
- ORG 疑似命令以前に CSEG，DSEG 疑似命令の記述がない場合，そのアブソリュート・セグメントは，コード・セグメント中のアブソリュート・セグメントと解釈されます。
- ORG 疑似命令のオペランドとして，ネーム／ラベルを記述する場合，そのネーム／ラベルは，ソース・モジュール中ですでに定義されたアブソリュート項でなければなりません。
- 絶対式として不当なものを記述した場合，または絶対式の評価値が 00000H - FFEFFH を越える場合は，エラーとなり，アセンブラは絶対式の値を 00000H とみなして，処理を続けます。
- オペランドの絶対式は，符号なし 32 ビットで評価されます。
- セグメント名は，シンボルとして参照できません。
- アセンブラの出力するセグメントの総数は，セグメント定義疑似命令によるセグメントをあわせて，別名セグメントが 256 個までです。同名セグメントは 1 つと数えます。
- セグメント名の最大認識文字数は，8 文字です。
- セグメント名の大文字，小文字は区別されます。

【使用例】

```
        NAME    SAMP1

        DSEG
        ORG     0FFE20H         ; (1)
SADR1 : DS      1
SADR2 : DS      1
SADR3 : DS      2

MAIN0   ORG     100H
        MOV     A , SADR1       ; (2)   ←エラー

        CSEG                    ; (3)
MAIN1   ORG     1000H           ; (4)
        MOV     A , SADR2
        MOVW    AX , SADR3

        END
```

(1) データ・セグメントに属するアブソリュート・セグメントを定義します。
このアブソリュート・セグメントは，ショート・ダイレクト・アドレッシング領域の先頭アドレス FFE20H 番地から配置されます。セグメント名の指定を省略しているので，アセンブラが自動的に“?A0FFE20”というセグメント名を与えます。

(2) データ・セグメントに属するアブソリュート・セグメント内では，インストラクションの記述はできないため，エラーとなります。

(3) コード・セグメントの開始を宣言します。

(4) このアブソリュート・セグメントは，1000H 番地から配置されます。

3.3 シンボル定義疑似命令

シンボル定義疑似命令は，ソース・モジュールを記述する際に使用するデータにネーム（名前）を割り付けます。これにより，データ値の意味がはっきりし，ソース・モジュールの内容がわかりやすくなります。

シンボル定義疑似命令は，ソース・モジュール中で使用するネームの値をアセンブラに知らせるものです。

シンボル定義疑似命令には，次のものがあります。

- EQU
- SET

EQU

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
ネーム	EQU	*式*	[; *コメント*]

【機能】

- オペランドで指定した式の値と属性（シンボル属性，およびリロケーション属性）を持つネーム（名前）を定義します。

【用途】

- ソース・モジュールの中で使用する数値データをネームとして定義し，命令のオペランドに数値データの代わりに記述します。

  特に，ソース・モジュールの中で頻繁に使用する数値データは，ネームとして定義しておくことをお勧めします。何らかの理由により，ソース・モジュール中のあるデータ値を変更しなければならないとき，ネームとして定義しておけば，そのネームのオペランド値を変更するだけで済みます。

【説明】

- EQU 疑似命令は，ソース・プログラムのどこに記述してもかまいません。
- EQU 疑似命令で定義したシンボルは，SET，およびラベルとして再定義することはできません。また，SET で定義したシンボルやラベルも EQU 疑似命令で再定義することはできません。
- EQU 疑似命令のオペランドにネーム／ラベルを記述する場合は，すでにソース・モジュール中で定義されているネーム／ラベルを使用します。

  オペランドとして外部参照項を記述することはできません。

  SFR，SFR ビット名称は，記述可能です。
- リロケータブルな項をオペランドに持つ HIGH/LOW/HIGHW/LOWW/DATAPOS/BITPOS 演算子によってできた項を含む式を記述することはできません。
- オペランドに次のパターンを含む式を記述すると，エラーとなります。

  (1) ADDRESS 属性の式 1 － ADDRESS 属性の式 2

  上記で，次の (a)，(b) があてはまる場合。

  (a) ADDRESS 属性の式 1 中のラベル 1 と ADDRESS 属性の式 2 のラベルの間に，その場でオブジェクト・コードのバイト数が決定できない BR 疑似命令が記述されている場合。

  (b) ラベル 1 とラベル 2 が別セグメントであり，属するセグメントの先頭からラベルまでの間に，その場でオブジェクト・コードのバイト数が決定できない BR 疑似命令が記述されている場合。

  (2) ADDRESS 属性の式 1　比較演算子　ADDRESS 属性の式 2

  (3) HIGH　アブソリュートな ADDRESS 属性の式

(4) LOW　アブソリュートな ADDRESS 属性の式

(5) HIGHW　アブソリュートな ADDRESS 属性の式

(6) LOWW　アブソリュートな ADDRESS 属性の式

(7) DATAPOS　アブソリュートな ADDRESS 属性の式

(8) BITPOS　アブソリュートな ADDRESS 属性の式

(9) 上記 (3) 〜 (8) の中で，次の (a) があてはまる場合。

(a) ADDRESS 属性の式中のラベルと，属するセグメントの先頭の間にその場でオブジェクト・コードのバイト数が決定できない BR 疑似命令が記述されている場合。

- オペランドの記述形式にエラーがある場合，アセンブラはエラーを出力し，解析可能なかぎりの値をネームの値として登録します。
- EQU 疑似命令により定義したネームは，同一のソース・モジュール中では再定義することはできません。
- EQU 疑似命令でビット値を定義したネームは，値としてアドレスとビット位置を持ちます。
- EQU 疑似命令のオペランドとして記述可能なビット値とその参照可能範囲を，次に示します。

オペランドの種類	シンボル値	参照可能範囲
A.bit 注 1	1.bit	同一モジュール内でのみ参照可能
PSW.bit 注 1	0FFFFAH.bit	
sfr 注 2.bit 注 1	0FFFXXH 注 3.bit	
2ndsfr 注 2.bit 注 1	0FXXXXH 注 4.bit	
saddr.bit 注 1	0FFXXXH 注 5.bit	別モジュールから参照可能
式 .bit 注 1	0XXXXXH 注 6.bit	

注 1　bit = 0 - 7

注 2　具体的な記述については，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。

注 3　0FFFXXH ： sfr のアドレス

注 4　0FXXXXH ： 2ndsfr 領域

注 5　0FXXXXH ： saddr 領域（0FFE20H - 0FFF1FH）

注 6　0XXXXXH ： 0H - 0FFFFFH

【使用例】

```
        NAME    SAMP1

WORK1   EQU     0FFE20H         ; (1)
WORK10  EQU     WORK1.0         ; (2)
P02     EQU     P0.2            ; (3)
A4      EQU     A.4             ; (4)
PSW5    EQU     PSW.5           ; (5)

        SET1    WORK10          ; (6)
        SET1    P02             ; (7)
        SET1    A4              ; (8)
        SET1    PSW5            ; (9)

        END
```

(1) ネーム“WORK1”は，値“0FFE20H”と，シンボル属性“NUMBER”，リロケーション属性“ABSOLUTE”を持ちます。

(2)“saddr.bit”にあたるビット値“WORK1.0”に，ネーム“WORK10”を割り当てます。
オペランドに記述されている“WORK1”は，(1) で値“0FFE20H”と定義済みです。

(3)“sfr.bit”にあたるビット値“P0.2”に，ネーム“P02”を割り当てます。

(4)“A.bit”にあたるビット値“A.4”に，ネーム“A4”を割り当てます。

(5)“PSW.bit”にあたるビット値“PSW.5”に，ネーム“PSW5”を割り当てます。

(6) この記述は，“SET1 saddr.bit”に該当します。

(7) この記述は，“SET1 sfr.bit”に該当します。

(8) この記述は，“SET1 A.bit”に該当します。

(9) この記述は，“SET1 PSW.bit”に該当します。

なお，(4)，(5) のように“A.bit”，“PSW.bit”を定義したネームは，そのモジュール内でのみ参照することができます。
“sfr.bit”，“saddr.bit”，“式 .bit”を定義したネームは，外部定義シンボルとして別のモジュールからも参照することができます（「3.5 リンケージ疑似命令」を参照）。

使用例のソース・モジュールをアセンブルすると，次のようなアセンブル・リストが生成されます。

＜アセンブル・リスト＞

```
        Assemble list

ALNO    STNO    ADRS    OBJECT      M I SOURCE STATEMENT

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

ビット値をネームとして定義している (2) ～ (5) の行には，アセンブル・リストのオブジェクト・コード欄に定義されたネームの持つビット・アドレスの値が表示されています。

SET

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
ネーム	SET	*絶対式*	[; *コメント*]

【機能】

- オペランドで指定した式の値と属性（シンボル属性，およびリロケーション属性）を持つネーム（名前）を定義します。
- SET 疑似命令で定義したネームの値と属性は，同一モジュール内において SET 疑似命令により再定義できます。SET 疑似命令により定義したネームの値と属性は，再び同じネームを再定義するまで有効です。

【用途】

- ソース・モジュールの中で使用する変数をネームとして定義し，命令のオペランドに数値データ（変数）の代わりに記述します。
  ソース・モジュールの中で，ネームの値を変更したい場合には，再度 SET 疑似命令で同じネームに異なる数値データを定義できます。

【説明】

- オペランドには，絶対式を記述します。
- SET 疑似命令は，ソース・プログラムのどこに記述してもかまいません。
  ただし，SET 疑似命令でネームを定義したネームを前方参照することはできません。
- SET 疑似命令でネームを定義した文にエラーがあると，アセンブラはエラーを出力し，解析可能なかぎりの値をネームの値として登録します。
- EQU 疑似命令で定義したシンボルを SET 疑似命令で再定義することはできません。
  また，SET 疑似命令で定義したシンボルを EQU 疑似命令で再定義することもできません。
- ビット・シンボルは定義できません。

【使用例】

```
        NAME    SAMP1
COUNT   SET     10H             ; (1)

        CSEG
        MOV     B , #COUNT      ; (2)
LOOP :
        DEC     B
        BNZ     $LOOP

COUNT   SET     20H             ; (3)

        MOV     B , #COUNT      ; (4)

        END
```

(1) ネーム“COUNT”は，値“10H”と，シンボル属性“NUMBER”，リロケーション属性“ABSOLUTE”を持ちます。これらは，(3) の記述の直前まで有効です。

(2) レジスタ B には，“COUNT”の値 10H が転送されます。

(3) ネーム“COUNT”の値を，“20H”に変更します。

(4) レジスタ B には，“COUNT”の値 20H が転送されます。

3.4 メモリ初期化，領域確保疑似命令

メモリ初期化疑似命令は，プログラムで使用する定数データを定義します。
定義したデータの値は，オブジェクト・コードとして生成されます。
領域確保疑似命令は，プログラムで使用するメモリの領域を確保します。

メモリ初期化，領域確保疑似命令には，次のものがあります。

- DB
- DW
- DG
- DS
- DBIT

DB

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	DB	(*サイズ*)	[; *コメント*]
		または	
[*ラベル* :]	DB	*初期値* [, …]	[; *コメント*]

【機能】

- バイト領域を初期化します。
  初期化するバイト数は，サイズとして指定することができます。
- オペランドで指定された初期値で，メモリをバイト単位で初期化します。

【用途】

- プログラムで使用する式や文字列を定義するときに，DB 疑似命令を使用します。

【説明】

- オペランドがカッコ“ (”，“) ”で囲まれている場合はサイズ指定とみなされ，そうでない場合は初期値とみなされます。
  (1) サイズ指定の場合
    (a) オペランドにサイズを記述した場合，アセンブラは，指定されたバイト数分の領域を“00H”で初期化します。
    (b) サイズには，絶対式を記述します。サイズの記述が不正な場合，エラーが出力され，初期化は行われません。
  (2) 初期値指定の場合
    (a) 式
    式の値は 8 ビットのデータとして確保されます。したがって，式の値は 0H - 0FFH の間でなければなりません。8 ビットを越えた場合，下位 8 ビットがデータとして確保され，エラーが出力されます。
    (b) 文字列
    文字列が記述された場合，1 文字に対して，それぞれ 8 ビット ASCII コードが確保されます。
- DB 疑似命令は，ビット・セグメント内では記述することはできません。
- 初期値は，1 行の範囲であれば複数指定することができます。
- 初期値として，リロケータブルなシンボルや外部参照シンボルを含んだ式を記述することができます。

【使用例】

```
                NAME    SAMP1
                CSEG
WORK1 :         DB      ( 1 )                   ; (1)
WORK2 :         DB      ( 2 )                   ; (1)
                CSEG
MASSAG :        DB      'ABCDEF'                ; (2)
DATA1 :         DB      0AH , 0BH , 0CH         ; (3)
DATA2 :         DB      ( 3 + 1 )               ; (4)
DATA3 :         DB      'AB' + 1                ; (5)   ←エラー

                END
```

(1) サイズを指定しているので，それぞれのバイト領域を値“00H”で初期化します。

(2) 6 バイトの領域を文字列“ABCDEF”で初期化します。

(3) 3 バイトの領域を 0AH，0BH，0CH で初期化します。

(4) 4 バイトの領域を，00H で初期化します。

(5)“AB”+1 の値は，4143H (4142H + 1) で 0H - 0FFH の範囲を越えています。
したがって，この記述はエラーとなります。

DW

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	DW	(*サイズ*)	[; *コメント*]
		または	
[*ラベル* :]	DW	*初期値* [, …]	[; *コメント*]

【機能】

- ワード領域を初期化します。
  初期化するワード数は，サイズとして指定することができます。
- オペランドで指定された初期値で，メモリをワード（2 バイト）単位に初期化します。

【用途】

- プログラムで使用するアドレスやデータなどの 16 ビットの定数を定義するときに，DW 疑似命令を使用します。

【説明】

- オペランドがカッコ“ (”，“) ”で囲まれている場合はサイズ指定とみなされ，そうでない場合は初期値とみなされます。

  (1) サイズ指定の場合

  (a) オペランドにサイズを記述した場合，アセンブラは指定されたワード数分の領域を“00H”で初期化します。

  (b) サイズには，絶対式を記述します。サイズの記述が不正な場合，エラーが出力され，初期化は行われません。

  (2) 初期値指定の場合

  (a) 定数

  16 ビット以下の定数です。

  (b) 式

  式の値は，16 ビット・データとして確保されます。

  文字列は，初期値として記述できません。

- DW 疑似命令は，ビット・セグメント内では記述できません。
- 初期値の上位 2 桁がメモリの HIGH アドレスに，下位 2 桁がメモリの LOW アドレスに確保されます。
- 初期値は，1 行の範囲であれば複数指定することができます。
- 初期値として，リロケータブルなシンボルや外部参照シンボルを含んだ式が記述することができます。

【使用例】

```
                NAME    SAMP1
                CSEG
WORK1 :         DW      ( 10 )          ; (1)
WORK2 :         DW      ( 128 )         ; (1)
                CSEG
                ORG     10H
                DW      MAIN            ; (2)
                DW      SUB1            ; (2)

                CSEG
MAIN :
                CSEG
SUB1 :

DATA :          DW      1234H , 5678H   ; (3)

                END
```

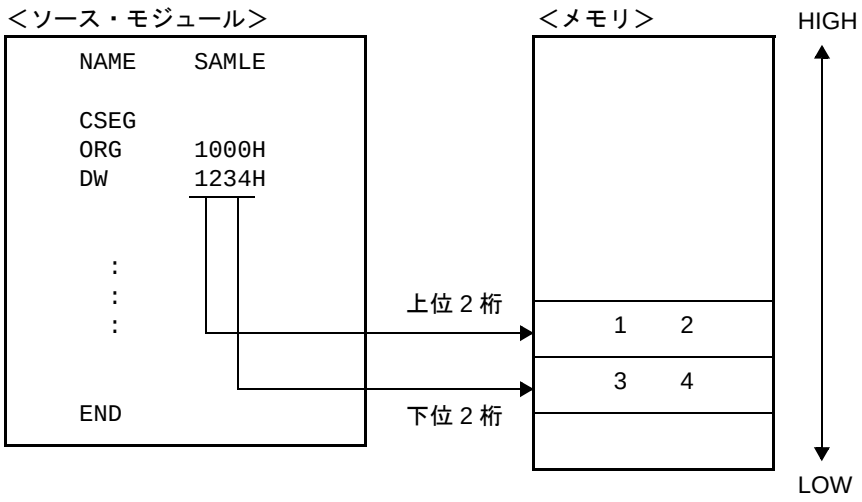
(1) サイズを指定しているので，それぞれのワード領域を値“00H”で初期化します。

(2) ベクタ・エントリ・アドレスを，DW 疑似命令で定義します。

(3) 2 ワードの領域を“34127856”の値で初期化します。

注意 ワード値は，上位 2 桁でメモリの HIGH アドレスを，下位 2 桁でメモリの LOW アドレスを初期化します。

【例】



DG

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	DG	(*サイズ*)	[; *コメント*]
		または	
[*ラベル* :]	DG	*初期値* [, …]	[; *コメント*]

【機能】

- 20 ビット領域を 32 ビット（4 バイト）単位でを初期化します。オペランドとして，初期値，あるいはサイズを指定することができます。
- オペランドで指定された初期値で，メモリを 4 バイト単位に初期化します。

【用途】

- プログラムで使用するアドレスやデータなどの 20 ビットの定数を定義するときに，DG 疑似命令を使用します。

【説明】

- オペランドがカッコ“ (”，“) ”で囲まれている場合はサイズ指定とみなされ，そうでない場合は初期値とみなされます。

(1) サイズ指定の場合

(a) オペランドにサイズを記述した場合，アセンブラは指定された数× 4 バイト分の領域を“00H”で初期化します。

(b) サイズには，絶対式を記述します。サイズの記述が不正な場合，エラーが出力され，初期化は行われません。

(2) 初期値指定の場合

(a) 定数

20 ビット以下の定数です。

(b) 式

式の値は，20 ビット・データとして確保されます。

文字列は，初期値として記述できません。

- DG 疑似命令は，ビット・セグメント内では記述できません。
- 初期値の最上位 1 バイトがメモリの HIGH WORD アドレスに，最下位 1 バイトがメモリの LOW アドレスに，最下位 2 バイト中上位 1 バイトがメモリの HIGH アドレスに確保されます。
- 初期値は，1 行の範囲であれば複数指定することができます。
- 初期値として，リロケータブルなシンボルや外部参照シンボルを含んだ式を記述することができます。

【使用例】

```
                NAME    SAMP1

DATA1 :         DG      12345H , 56789H         ; (1)
DATA2 :         DG      ( 10 )                  ; (2)

                END
```

(1) 4 バイトの領域を“4523010089670500”の値で初期化します。

(2) 40 バイト（10 × 4 バイト）の領域が“00H”で初期化されます。

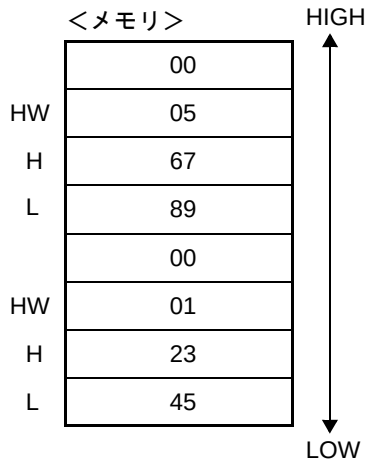
注意 20 ビット値は，最上位 1 バイトでメモリの HIGH WORD アドレスを，最下位 1 バイトでメモリの LOW アドレスを，最下位 2 バイト中上位 1 バイトで HIGH アドレスを初期化します。

【例】

＜ソース・モジュール＞

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

＜メモリ＞



HW : HIGH WORD
H : HIGH
L : LOW

DS

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	DS	*絶対式*	[; *コメント*]

【機能】

- オペランドで指定したバイト数分のメモリ領域を確保します。

【用途】

- DS 疑似命令は，主にプログラムで使用するメモリ（RAM）の領域を確保するときに使用します。
  ラベルがある場合は，確保したメモリ領域の先頭アドレスの値をそのラベルに割り付けます。ソース・モジュールでは，このラベルを使用してメモリを操作する記述をします。

【説明】

- 確保する領域の内容は，不定です。
- 絶対式は，符号なし 16 ビットで評価されます。
- オペランドの値が 0 のときは，領域は確保されません。
- DS 疑似命令は，ビット・セグメント内では記述することはできません。
- DS 疑似命令のシンボルは後方参照のみです。
- オペランドに記述できるのは，絶対式を拡張した次のものです。

  (1) 定数

  (2) 定数に演算を施した式（定数式）

  (3) 定数，または定数式で定義された EQU シンボル，または SET シンボル

  (4) ADDRESS 属性の式 1 － ADDRESS 属性の式 2
  （“ADDRESS 属性の式 1”中のラベル 1 と“ADDRESS 属性の式 2”中のラベル 2 は，リロケータブルな場合には，同一セグメント中で定義されていなければなりません）
  ただし，以下の場合にはエラーとなります。

  (a) ラベル 1 とラベル 2 が同一セグメントで，2 つのラベルの間にその場でオブジェクト・コードのバイト数が決定できない BR 疑似命令が記述されている場合

  (b) ラベル 1 とラベル 2 が別のセグメントで，属するセグメントの先頭からラベルまでの間に，その場でオブジェクト・コードのバイト数が決定できない BR 疑似命令が記述されている場合

  (5) (1) ～ (4) の式に演算を施した式

オペランドに記述することのできないものを次に示します。

(1) 外部参照シンボル

(2) ADDRESS 属性の式 1 － ADDRESS 属性の式 2 を EQU で定義したシンボル

(3) ADDRESS 属性の式 1 － ADDRESS 属性の式 2 の形で式 1，2 のいずれかにロケーション・カウンタ（$）が記述された場合

(4) ADDRESS 属性の式に HIGH/LOW/DATAPOS/BITPOS を施した式を EQU で定義したシンボル

【使用例】

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

                END
```

(1) 10 バイトの作業領域を確保しますが，領域の内容は不定です。ラベル“TABLE1”を，先頭アドレスに割り付けます。

(2) 1 バイトの作業領域を確保します。

(3) 2 バイトの作業領域を確保します。

DBIT

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ネーム*]	DBIT	*なし*	[; *コメント*]

【機能】

- ビット・セグメント中で 1 ビットのメモリ領域を確保します。

【用途】

- DBIT 疑似命令は，ビット・セグメント中で，ビット領域を確保するために使用します。

【説明】

- DBIT 疑似命令は，ビット・セグメント中でのみ記述します。
- 確保した領域の内容は，不定です。
- シンボル欄にネームを記述した場合，そのネームは値として，アドレスとビット位置を持ちます。
- 定義したネームは，saddr.bit，addr16.bit，ES:addr16.bit を要求される箇所に記述することができます。

【使用例】

```
        NAME    SAMPLE
        BSEG
BIT1    DBIT            ; (1)
BIT2    DBIT            ; (1)
BIT3    DBIT            ; (1)

        CSEG
        SET1    BIT1    ; (2)

        CLR1    BIT2    ; (3)

        END
```

(1) 1 ビットごとの領域を確保し，それぞれのアドレスとビット位置を値として持つネーム（BIT1，BIT2，BIT3）を定義します。

(2) この記述は，“SET1 saddr.bit”に該当します。
“saddr.bit”として，(1) で確保したビット領域のネーム BIT1 を記述します。

(3) この記述は，“CLR1 saddr.bit”に該当します。
“saddr.bit”として，ネーム BIT2 を記述します。

3.5 リンケージ疑似命令

リンケージ疑似命令は，ほかのモジュールで定義されているシンボルを参照する場合に，その関連性を明白にさせるためのものです。

1 つのプログラムがモジュール 1 とモジュール 2 に分けて作成されている場合を考えます。モジュール 1 中において，モジュール 2 中で定義されているシンボルを参照したい場合，お互いのモジュールで何の宣言もなくそのシンボルを使うわけにはいきません。このため，「使いたい」，「使ってもよい」の表示をそれぞれのモジュールで行う必要があります。

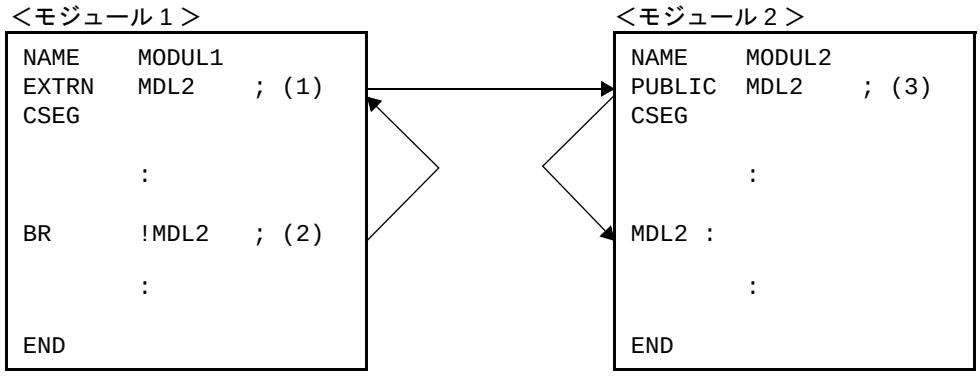
モジュール 1 では，「ほかのモジュール中で定義されているシンボルを参照したい」というシンボルの外部参照宣言をします。一方，モジュール 2 では，「そのシンボルは，ほかのシンボルで参照してもよい」というシンボルの外部定義宣言をします。

外部参照と外部定義という 2 つの宣言が有効に行われて，はじめてそのシンボルを参照することができます。

この相互関係を成立させるのが，リンケージ疑似命令であり，次の命令があります。

- シンボルの外部参照宣言を行うもの：EXTRN，および EXTBIT 疑似命令
- シンボルの外部定義宣言を行うもの：PUBLIC 疑似命令

図 3-2 2 つのモジュール間のシンボルの関係



上記のモジュール 1 では，モジュール 2 の中で定義しているシンボル“MDL2”を (2) で参照しているため，(1) で EXTRN 疑似命令により外部参照宣言を行っています。

モジュール 2 では，モジュール 1 から参照されるシンボル“MDL2”を (3) で，PUBLIC 疑似命令により外部定義宣言を行っています。

この外部参照，外部定義シンボルが正しく対応しているかどうかは，リンカによりチェックされます。

リンケージ疑似命令には，次のものがあります。

- EXTRN
- EXTBIT
- PUBLIC

EXTRN

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	EXTRN	*シンボル名* [, …]	[; *コメント*]
		または	
[*ラベル* :]	EXTRN	BASE(*シンボル名* [, …])	[; *コメント*]

【機能】

- このモジュールで参照するほかのモジュールのシンボル（ビット・シンボルを除く）を宣言します。

【用途】

- ほかのモジュールの中で定義されているシンボルを参照する場合には，必ずそのシンボルを EXTRN 疑似命令で外部参照宣言します。
- オペランドの記述形式により，以下の違いがあります。

BASE (*シンボル名* [, …])	64K バイト内の（0H - 0FFFF 内）領域のシンボルとして参照可能となります。
再配置属性なし	リンカが配置後，PUBLIC されたシンボルの領域にあわせて処理を行い，参照可能となります。

【説明】

- EXTRN 疑似命令は，ソース・プログラムのどこに記述してもかまいません（「2.1 基本構成」を参照してください）。
- オペランドには，コンマ（ , ）で区切って最大 20 個のシンボルを指定することができます。
- ビット値を持つシンボルを参照する場合は，EXTBIT 疑似命令で外部参照宣言をします。
- EXTRN 疑似命令で宣言されたシンボルは，ほかのモジュールで PUBLIC 疑似命令で宣言されていなければなりません。
- EXTRN 疑似命令で宣言されたシンボルをモジュール中で参照しなくても，エラーにはなりません。
- EXTRN 疑似命令のオペランドとして，マクロ名を記述することはできません（マクロ名については，「第 5 章 マクロ」を参照してください）。
- シンボルは，全モジュール中で一度だけ EXTRN 宣言できます。2 回目以降の宣言に対しては，ワーニングが出力されます。
- すでに宣言されたシンボルは，EXTRN 疑似命令のオペランドに記述することはできません。逆に，EXTRN 宣言したシンボルも，ほかの疑似命令により再定義，宣言することはできません。
- 64K バイト内の（0H - 0FFFFH 内）領域を EXTRN 疑似命令で定義したシンボルで参照することができます。“BASE(シンボル名)”の記述形式で宣言されたシンボル名は，64K バイト内へ参照可能です。

【使用例】

＜モジュール 1 ＞

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

        END
```

＜モジュール 2 ＞

```
        NAME    SAMP2
        PUBLIC  SYM1 , SYM2 , SYM3              ; (5)

        CSEG
SYM1    EQU     0FFH                            ; (6)
DATA1   DSEG    SADDR
SYM2 :  DB      012H                            ; (7)
C1      CSEG    BASE
SYM3 :  MOV     A , #20H                        ; (8)

        END
```

(1) (2) と (3) と (4) で参照するシンボル“SYM1”，“SYM2”，“SYM3”の外部参照宣言を行います。
オペランド欄には，複数のシンボルを記述することができます。

(2) シンボル“SYM1”を参照します。

(3) シンボル“SYM2”を参照します。saddr 領域を参照するコードを出力します。

(4) シンボル“SYM3”を参照します。64K バイト内（0H - 0FFFFH 内）の領域を参照するコードを出力します。

(5) シンボル“SYM1”，“SYM2”，“SYM3”を外部定義宣言します。

(6) シンボル“SYM1”を定義します。

(7) シンボル“SYM2”を定義します。

(8) シンボル“SYM3”を定義します。

EXTBIT

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	EXTBIT	*ビット・シンボル名* [, …]	[; *コメント*]

【機能】

- 本モジュールで参照するほかのモジュールのビット・シンボルを宣言します。

【用途】

- ほかのモジュールの中で定義されているビット値を持つシンボルを参照する場合には，必ずそのシンボルを EXTBIT 疑似命令で外部参照宣言します。

【説明】

- EXTBIT 疑似命令は，ソース・プログラムのどこに記述してもかまいません。
- オペランドには，コンマ（ , ）で区切って最大 20 個のシンボルを指定することができます。
- EXTBIT 疑似命令で宣言されたシンボルは，ほかのモジュールで PUBLIC 疑似命令で宣言されていなければなりません。
- シンボルは，1 モジュール中で一度だけ EXTBIT 宣言することができます。2 回目以降の宣言に対しては，ワーニングが出力されます。
- EXBIT 疑似命令で宣言されたシンボルをモジュール中で参照しなくても，エラーにはなりません。

【使用例】

＜モジュール 1 ＞

```
        NAME    SAMP1
        EXTBIT  FLAG1 , FLAG2   ; (1)
        CSEG
        SET1    FLAG1           ; (2)
        CLR1    FLAG2           ; (3)

        END
```

＜モジュール 2 ＞

```
        NAME    SAMP2
        PUBLIC  FLAG1 , FLAG2   ; (4)
        BSEG
FLAG1   DBIT                    ; (5)
FLAG2   DBIT                    ; (6)
        CSEG
        NOP

        END
```

(1) 参照するシンボル“FLAG1”，“FLAG2”の外部参照宣言を行います。

オペランド欄には，複数のシンボルを記述することができます。

(2) シンボル“FLAG1”を参照します。

この記述は，“SET1 saddr.bit”に該当します。

(3) シンボル“FLAG2”を参照します。

この記述は，“CLR1 saddr.bit”に該当します。

(4) シンボル“FLAG1”，“FLAG2”を定義します。

(5) シンボル“FLAG1”を SADDR 領域のビット・シンボルとして定義します。

(6) シンボル“FLAG2”を SADDR 領域のビット・シンボルとして定義します。

PUBLIC

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	PUBLIC	*シンボル名* [, …]	[; *コメント*]

【機能】

- オペランドに記述したシンボルをほかのモジュールから参照できるよう宣言します。

【用途】

- ほかのモジュールから参照されるシンボル（ビット・シンボルを含む）を定義している場合には，必ず，そのシンボルを PUBLIC 疑似命令で外部定義宣言します。

【説明】

- PUBLIC 疑似命令は，ソース・プログラムのどこに記述してもかまいません。
- オペランドには，コンマ（ , ）で区切って最大 20 個のシンボルを指定することができます。
- オペランドに記述するシンボルは，同一モジュール内で定義していなければなりません。
- シンボルは，全モジュール中で一度だけ PUBLIC 宣言することができます。2 回目以降の宣言は，無視されます。
- 各ビット領域にあるビットシンボルは，PUBLIC 宣言することが可能です。
- 次のシンボルは，オペランドとして記述することはできません。
  (1) SET 疑似命令で定義したネーム
  (2) 同一モジュール内で EXTRN，EXTBIT 疑似命令で定義したシンボル
  (3) セグメント名
  (4) モジュール名
  (5) マクロ名
  (6) モジュール内で定義されていないシンボル
  (7) SFBIT 属性を持つオペランドを EQU 疑似命令で定義したシンボル
  (8) sfr，2ndSFR を EQU 疑似命令で定義したシンボル（ただし，sfr 領域と saddr 領域のオーバーラップしている箇所は除きます。）

【使用例】

＜モジュール 1 ＞

```
        NAME    SAMP1
        PUBLIC  A1 , A2         ; (1)
        EXTRN   B1
        EXTBIT  C1

A1      EQU     10H
A2      EQU     0FFE20H.1

        CSEG
        BR      B1
        SET1    C1

        END
```

＜モジュール 2 ＞

```
        NAME    SAMP2
        PUBLIC  B1              ; (2)
        EXTRN   A1
        CSEG
B1 :
        MOV     C , #LOW ( A1 )

        END
```

＜モジュール 3 ＞

```
        NAME    SAMP3
        PUBLIC  C1              ; (3)
        EXTBIT  A2
C1      EQU     0FFE21H.0
        CSEG
        CLR1    A2

        END
```

(1) シンボル A1，A2 が，ほかのモジュールから参照されるシンボルであることを宣言します。

(2) シンボル B1 が，ほかのモジュールから参照されるシンボルであることを宣言します。

(3) シンボル C1 が，ほかのモジュールから参照されるシンボルであることを宣言します。

3.6 オブジェクト・モジュール名宣言疑似命令

オブジェクト・モジュール名宣言疑似命令は，アセンブラで生成するオブジェクト・モジュールにモジュール名を与えます。

オブジェクト・モジュール名宣言疑似命令には，次のものがあります。

- NAME

NAME

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	NAME	*オブジェクト・モジュール名*	[; *コメント*]

【機能】

- オペランドに記述したオブジェクト・モジュール名を，アセンブラの出力するオブジェクト・モジュールに与えます。

【用途】

- モジュール名は，デバッガによるシンボリック・デバッグ時に必要となります。

【説明】

- NAME 疑似命令は，ソース中・プログラムのどこに記述してもかまいません。
- モジュール名の規則については，「2.2.3 シンボル欄」を参照してください。
- モジュール名として指定できる文字は，OS でファイル名として許す文字から“ (”（28H），“) ”（29H）と 2 バイト文字を除いた文字とします。
- モジュール名を，その他の疑似命令，インストラクションのオペランドとして記述することはできません。
- NAME 疑似命令を省略すると，ソース・モジュール・ファイルのプライマリ・ネーム（先頭から 256 文字）がモジュール名になります。なお，プライマリ・ネームは，大文字に変換されて取り出されます。
  複数個指定した場合は，ワーニングが出力され，2 回目以降の宣言は無視されます。
- オペランド欄のモジュール名は，256 文字以内で指定してください。
- シンボル名の大文字，小文字は区別されます。

【使用例】

```
        NAME    SAMPLE  ; (1)
        DSEG
BIT1 :  DBIT

        CSEG
        MOV     A , B

        END
```

(1) モジュール名を SAMPLE として宣言します。

3.7 分岐命令自動選択疑似命令

無条件分岐命令において，分岐先アドレスをオペランドとして直接記述するものには，“BR !addr20”，“BR $addr20”の 2 つがあります。

これらの命令は，命令のバイト数が異なるため，メモリ効率のよいプログラムを作成するためには，ユーザが分岐先の範囲に応じて，どのオペランドが適しているかを選択して使用する必要があります。

そこで，RA78K0R が自動的に分岐先の範囲に応じて，2 バイト，または 3 バイトの分岐命令を選択する疑似命令を設けました。これを分岐命令自動選択疑似命令と呼びます。

分岐命令自動選択疑似命令には，次のものがあります。

- BR
- CALL

BR

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	BR	*式*	[; *コメント*]

【機能】

- オペランドで指定された式の値の範囲に応じて，アセンブラが自動的に 2 バイトから 4 バイトの BR 分岐命令を選択し，該当するオブジェクト・コードを生成します。

【用途】

- 次の分岐命令のうち，分岐先範囲を判断して，可能であれば命令バイト数の少ない命令を自動的に選択し，出力します。2 バイトの分岐命令で記述できるかどうか，はっきりしない分岐命令については，BR 疑似命令を使用します。

分岐命令	説明
“BR　$addr20”（2 バイト）	分岐先が BR 疑似命令の次のアドレス -80H から +7FH の範囲内で使用可能
“BR　!addr20”（3 バイト）	分岐先が 64K バイト内の場合，使用可能
“BR　$!addr20”（3 バイト）	相対距離を計算し，-8000H から +7FFFH の範囲内で使用可能
“BR　!!addr20”（4 バイト）	上記以外の場合に使用

なお，オペランド（分岐先）が，疑似命令とは異なるリロケータブル・セグメント内で，BASE 領域外に割り当てられる場合は，4 バイト命令に置き換えて出力します。
また，疑似命令とオペランド（分岐先）が異なるセグメントで，BASE 領域外に割り当てられ，かつ，別タイプの場合，オペランドがアブソリュート・セグメント内にあった場合でも 4 バイト命令に置き換えられます。
疑似命令と分岐先が別セグメントで BASE 領域内にある場合には，3 バイト命令（BR　!addr20）に置き換えられます。

備考　別タイプとは，BR 疑似命令がアブソリュート・セグメント内であれば，リロケータブルな別セグメント，BR 疑似命令がリロケータブル・セグメントであれば，アブソリュート・セグメントを示すことです。

- 2 バイトから 4 バイトのどの分岐命令を記述するべきかが明確に判断できる場合は，該当するインストラクションを記述するようにしてください。これにより，BR 疑似命令を記述する場合に比べ，アセンブル時間を短縮することができます。

【説明】

- BR 疑似命令は，コード・セグメント内でのみ使用可能です。
- BR 疑似命令のオペランドには，直接ジャンプ先を記述します。式の先頭に，現在のロケーション・カウンタを示す“$”を記述することはできません。
- 最適化の対象となるためには，次のような条件があります。

(1) 式中のラベル，または前方参照シンボルが 1 個以下。

(2) ADDRESS 属性の EQU シンボルが記述されていない。

(3) ADDRESS 属性の式ー ADDRESS 属性の式を EQU 定義したシンボルが記述されていない。

(4) ADDRESS 属性の式に HIGH/LOW/HIGHW/LOWW/DATAPOS/BITPOS を施した式が記述されていない。

これらの条件が満たされていない場合には，4 バイト命令となります。

【使用例】

```
アドレス                NAME    SAMPLE
              C1        CSEG    AT      50H
00050H                  BR      L1                      ; (1)
00052H                  BR      L2                      ; (2)
00055H                  BR      L3                      ; (3)

0007DH        L1 :
0FFFFH        L2 :
10000H        L3 :

              C2        CSEG    AT      20050H
20050H                  BR      L4                      ; (4)
27FFFH        L4 :

                                END
```

(1) この BR 疑似命令は，分岐先との相対距離が -80H から +7FH の範囲内なので，2 バイトの分岐命令（BR　$addr20）が生成されます。

(2) この BR 疑似命令は，分岐先が 64K 以内なので，3 バイトの分岐命令（BR　!addr20）に置き換えられます。

(3) この BR 疑似命令は，4 バイトの分岐命令（BR　!!addr20）に置き換えられます。

(4) この BR 疑似命令は，分岐先との相対距離が -8000H から +7FFFH の範囲内なので，3 バイトの分岐命令（BR　$!addr20）に置き換えられます。

CALL

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	CALL	*式*	[; *コメント*]

【機能】

- オペランドで指定された式の値の範囲に応じて，アセンブラが自動的に 3 バイトから 4 バイトの CALL 分岐命令を選択し，該当するオブジェクト・コードを生成します。

【用途】

- 次の分岐命令のうち，分岐先範囲を判断して，可能であれば命令バイト数の少ない命令を自動的に選択し，出力します。3 バイトの分岐命令で記述できるかどうか，はっきりしない分岐命令については，CALL 疑似命令を使用します。

分岐命令	説明
“CALL　!addr20”（3 バイト）	分岐先が 64K バイト内の場合，使用可能
“CALL　$!addr20”（3 バイト）	相対距離を計算し，-8000H から +7FFFH の範囲内で使用可能
“CALL　!!addr20”（4 バイト）	上記以外の場合に使用

なお，オペランド（分岐先）が，疑似命令とは異なるリロケータブル・セグメント内で，BASE 領域外に割り当てられる場合は，4 バイト命令に置き換えて出力します。
また，疑似命令とオペランド（分岐先）が異なるセグメントで，BASE 領域外に割り当てられ，かつ，別タイプ[注]の場合，オペランドがアブソリュート・セグメント内にあった場合でも 4 バイト命令に置き換えられます。
疑似命令と分岐先が別セグメントで BASE 領域内にある場合には，3 バイト命令（CALL　!addr20）に置き換えられます。

注 別タイプとは，CALL 疑似命令がアブソリュート・セグメント内であれば，リロケータブルな別セグメント，CALL 疑似命令がリロケータブル・セグメントであれば，アブソリュート・セグメントを示すことです。

- 3 バイト，または 4 バイトのどのコール命令を記述するべきかが明確に判断できる場合は，該当するインストラクションを記述するようにしてください。これにより，CALL 疑似命令を記述する場合に比べ，アセンブル時間を短縮することができます。

【説明】

- CALL 疑似命令は，コード・セグメント内でのみ使用可能です。
- CALL 疑似命令のオペランドには，直接コール先を記述します。
- 最適化の対象となるためには，次のような条件があります。

(1) 式中のラベル，または前方参照シンボルが 1 個以下。

(2) ADDRESS 属性の EQU シンボルが記述されていない。

(3) ADDRESS 属性の式－ ADDRESS 属性の式を EQU 定義したシンボルが記述されていない。

(4) ADDRESS 属性の式に HIGH/LOW/HIGHW/LOWW/DATAPOS/BITPOS を施した式が記述されていない。

これらの条件が満たされていない場合には，4 バイト命令となります。

【使用例】

```
アドレス                NAME    SAMPLE
              C1        CSEG    AT      50H
00050H                  CALL    L1                      ; (1)
00053H                  CALL    L2                      ; (2)
08052H        L1 :
0FFFFH        L2 :

              C2        CSEG    AT      20050H
20050H                  CALL    L3                      ; (3)
27FFFH        L3 :

                        END
```

(1) この CALL 疑似命令は，分岐先が 64K 以内なので，3 バイトの分岐命令（CALL　!addr20）に置き換えられます。

(2) この CALL 疑似命令は，4 バイトの分岐命令（CALL　!!addr20）に置き換えられます。

(3) この CALL 疑似命令は，分岐先との相対距離が -8000H から +7FFFH の範囲内なので，3 バイトの分岐命令（CALL　$!addr20）に置き換えられます。

3.8 マクロ疑似命令

ソースを記述する場合，使用頻度の高い一連の命令群をそのつど記述するのは面倒です。また，記述ミス増加の原因ともなります。

マクロ疑似命令により，マクロ機能を使用することにより，同じような一連の命令群を何回も記述する必要がなくなり，コーディングの効率を上げることができます。

マクロの基本的な機能は，一連の文の置き換えにあります。

マクロ疑似命令には，次のものがあります。

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

MACRO

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
マクロ名	MACRO : *マクロ・ボディ* : ENDM	[*仮パラメータ* [, …]]	[; *コメント*] [; *コメント*]

【機能】

- MACRO 疑似命令と ENDM 疑似命令の間に記述された一連の文（マクロ・ボディと呼びます）に対し，シンボル欄で指定したマクロ名を付け，マクロの定義を行います。

【用途】

- ソース中で，使用頻度の高い一連の文をマクロ定義しておきます。その定義以降では，定義されたマクロ名を記述するだけ（「5.2.2 マクロの参照」を参照）で，そのマクロ名に対応するマクロ・ボディが展開されます。

【説明】

- MACRO 疑似命令には，対応する ENDM 疑似命令がなければなりません。
- シンボル欄に記述するマクロ名の規則については，「2.2.3 シンボル欄」を参照してください。
- マクロを参照する場合は，ニモニック欄に定義済みのマクロ名を記述します。
- オペランド欄に記述する仮パラメータの規則については，シンボル記述上の規則と同じです。
- 1 つのマクロ疑似命令で指定できる仮パラメータは，16 個までです。
- 仮パラメータは，マクロ・ボディ内でのみ有効です。
- 仮パラメータとして予約語を記述すると，エラーとなります。ただし，ユーザ定義シンボルを記述した場合には，仮パラメータとしての認識が優先されます。
- 仮パラメータと実パラメータの個数は同じでなければなりません。
- マクロ・ボディ内で定義したネーム / ラベルを，LOCAL 疑似命令で宣言すれば，そのネーム／ラベルは 1 回のマクロ展開でのみ有効になります。
- マクロのネスティング（マクロ・ボディ内でほかのマクロを参照すること）は，REPT，IRP あわせて，最大 8 レベルまでです。
- 1 つのモジュール内でのマクロ定義の最大数には，特に制限はありません。メモリが使えるかぎり定義することができます。
- クロスリファレンス・リストには，仮パラメータの定義行，参照行，シンボル名は出力されません。
- マクロ・ボディ中に，2 つ以上のセグメントを定義することはできません。定義した場合は，エラーが出力されます。

【使用例】

```
        NAME    SAMPLE

ADMAC   MACRO   PARA1 , PARA2   ; (1)
        MOV     A , #PARA1
        ADD     A , #PARA2
        ENDM                    ; (2)

        ADMAC   10H , 20H       ; (3)

        END
```

(1) マクロ名“ADMAC”，2 つの仮パラメータ“PARA1”，“PARA2”を指定したマクロ定義をしています。

(2) マクロ定義の終わりを示します。

(3) マクロ ADMAC を参照しています。

LOCAL

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
なし	LOCAL	*シンボル名* [, …]	[; *コメント*]

【機能】

- オペランド欄で指定されたシンボル名は，そのマクロ・ボディ内でのみ有効なローカル・シンボルであることを宣言します。

【用途】

- マクロ・ボディ内でシンボルを定義しているマクロを 2 回以上参照すると，シンボルは二重定義エラーとなります。LOCAL 疑似命令を使用することにより，シンボルを定義しているマクロを複数回，参照することができます。

【説明】

- オペランド欄に記述するシンボル名の規則については，「2.2.3 シンボル欄」を参照してください。
- ローカル宣言されたシンボルは，展開されるごとに“??RAnnnn”（n = 0000 - FFFF）というシンボルに置き換えられます。置き換え後の“??RAnnnn”というシンボルは，グローバル・シンボルと同じ扱いとなり，シンボル・テーブルに登録され，“??RAnnnn”というシンボル名で参照することができます。
- マクロ・ボディ内でシンボルを定義し，そのマクロを 2 回以上参照すると，ソース・モジュール中でそのシンボルを 2 回以上定義することになってしまいます。このため，そのシンボルは，マクロ内でのみ有効なローカル・シンボルであると宣言します。
- LOCAL 疑似命令は，マクロ定義内でのみ使用できます。
- LOCAL 疑似命令は，オペランド欄で指定したシンボルを使用する前に記述しなければなりません（マクロ・ボディの先頭で記述してください）。
- 1 つのモジュール内で LOCAL 疑似命令により定義するシンボル名は，すべて別名でなければいけません（各マクロ内で使用するローカル・シンボル名に同一名は使えません）。
- オペランド欄で指定できるローカル・シンボル数は，1 行以内であればいくつでも定義することができます。ただし，マクロ・ボディ内での最大数は 64 個です。65 個以上のローカル・シンボルが宣言された場合はエラーが出力され，そのマクロ定義は空のマクロ・ボディとして登録されます。参照された場合は，何も展開されません。
- ローカル・シンボルを定義しているマクロは，ネストさせることができません。
- LOCAL 疑似命令で定義したシンボルをマクロ外から参照することはできません。
- シンボルとして予約語を記述することはできません。ただし，ユーザ定義シンボルと同じシンボルを記述した場合には，LOCAL シンボルとしての機能が優先されます。

- LOCAL 疑似命令のオペランドで宣言したシンボルは，クロスリファレンス・リスト，シンボル・テーブル・リストには出力されません。
- LOCAL 疑似命令の行は，展開時に出力されません。
- マクロ定義の仮パラメータと同名のシンボルを，そのマクロ定義の中で LOCAL 宣言した場合，エラーとなります。

【使用例】

```
        NAME    SAMPLE

        ; マクロの定義
MAC1    MACRO
        LOCAL   LLAB            ; (1)
LLAB :                          ;
        BR      $LLAB           ; (2)
        ENDM                    ;

        ; ソース本文
REF1 :  MAC1                    ; (3)

??RA0000 :
        BR      $??RA0000       ; (2)

        BR      !LLAB           ; (4)     ←エラー

REF2 :  MAC1                    ; (5)

??RA0001 :
        BR      $??RA0001       ; (2)

        END
```

(1) シンボル名 LLAB をローカル・シンボルとして定義しています。

(2) マクロ MAC1 内でローカル・シンボル LLAB を参照しています。

(3) マクロ MAC1 を参照しています。

(4) マクロ MAC1 の定義外でローカル・シンボル LLAB を参照しています。
この記述は，エラーになります。

(5) マクロ MAC1 を参照しています。

使用例のアセンブル・リストを次に示します。

＜アセンブル・リスト＞

```
        Assemble list

ALNO    STNO    ADRS    OBJECT  M I     SOURCE STATEMENT

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

REPT

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	REPT	*絶対式*	[; *コメント*]
	:		
	ENDM		[; *コメント*]

【機能】

- REPT 疑似命令と ENDM 疑似命令の間に記述された一連の文（REPT-ENDM ブロックと呼びます）を，オペランド欄で指定した式の値分だけ，アセンブラが繰り返し展開します。

【用途】

- ソース中で一連の文を連続して繰り返し記述する場合に，REPT，ENDM 疑似命令を使用します。

【説明】

- REPT 疑似命令に対応する ENDM 疑似命令がなければ，エラーとなります。
- REPT-ENDM ブロック内では，マクロ参照，REPT，IRP をあわせたネスト・レベルの最大数 8 までネスティングすることができます。
- REPT-ENDM のブロックの途中で EXITM が現れると，展開を中止します。
- REPT-ENDM のブロック内に，アセンブル制御命令を記述することができます。
- REPT-ENDM のブロック内に，マクロ定義を記述することはできません。
- オペランド欄に記述する絶対式は，符号なし 16 ビットで評価されます。
  絶対式が 0 の場合には，何も展開されません。

【使用例】

```
NAME    SAMP1
CSEG
    ; REPT-ENDM ブロック
REPT    3               ; (1)
        INC     B
        DEC     C
            ; ソース本文
ENDM                    ; (2)

END
```

(1) REPT-ENDM ブロックを 3 回連続して展開するよう，指示しています。

(2) REPT-ENDM ブロックの終了を示します。

アセンブルすると，REPT-ENDM ブロックは次のように展開されます。

＜アセンブル・リスト＞

```
NAME    SAMP1
CSEG
REPT    3
        INC     B
        DEC     C
ENDM
        INC     B
        DEC     C
        INC     B
        DEC     C
        INC     B
        DEC     C

END
```

(1)，(2) で定義された REPT-ENDM ブロックが，3 回展開されています。

アセンブル・リスト上には，ソース・モジュールの REPT 疑似命令による定義分 (1)，(2) は，表示されません。

IRP

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	IRP	*仮パラメータ* ,<[*実パラメータ* [, …]]>	[; *コメント*]
	:		
	ENDM		[; *コメント*]

【機能】

- IRP 疑似命令と ENDM 疑似命令の間にある一連の文（IRP-ENDM ブロックと呼びます）を，オペランドで指定された実パラメータ（左から順）で仮パラメータを置き換えながら，実パラメータの数だけ繰り返し展開します。

【用途】

- ソース中で，一部分だけ変数となる一連の文を連続して繰り返し記述したい場合に，IRP-ENDM 疑似命令を使用します。

【説明】

- IRP 疑似命令には対応する ENDM 疑似命令がなければなりません。
- 実パラメータは，16 個まで記述することができます。
- IRP-ENDM ブロック内では，マクロ参照，REPT，IRP をあわせたネスト・レベルの最大数 8 までネスティングすることができます。
- IRP-ENDM ブロックの途中に EXITM を記述すると，そこで展開を中止します。
- IRP-ENDM ブロックで，マクロを定義することはできません。
- IRP-ENDM ブロック内に，アセンブル制御命令を記述することができます。

【使用例】

```
NAME    SAMP1
CSEG

IRP     PARA , <0AH , 0BH , 0CH>        ; (1)
    ; IRP-ENDM ブロック
ADD     A , #PARA
MOV     [ DE ] , A
ENDM                                    ; (2)
    ; ソース本文
END
```

(1) 仮パラメータが PARA，実パラメータが 0AH，0BH，0CH の 3 個です。

仮パラメータ“PARA”を，実パラメータ“0AH”，“0BH”，“0CH”に置き換えながら，IRP-ENDM ブロックを実パラメータの数 3 回分展開することを指示します。

(2) IRP-ENDM ブロックの終了を示します。

アセンブルすると，IRP-ENDM ブロックは次のように展開されます。

＜アセンブル・リスト＞

```
NAME    SAMP1
CSEG
    ; IRP-ENDM ブロック
ADD     A , #0AH        ; (3)
MOV     [ DE ] , A
ADD     A , #0BH        ; (4)
MOV     [ DE ] , A
ADD     A , #0CH        ; (5)
MOV     [ DE ] , A
    ; ソース本文
END
```

使用例の (1)，(2) で定義された IRP-ENDM ブロックが，実パラメータの数 3 回分展開されています。

(3) PARA が 0AH に置き換えられました。

(4) PARA が 0BH に置き換えられました。

(5) PARA が 0CH に置き換えられました。

EXITM

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	EXITM	*なし*	[; *コメント*]

【機能】

- MACRO 疑似命令で定義されたマクロ・ボディの展開，および REPT-ENDM，IRP-ENDM による繰り返しを強制的に終了させます。

【用途】

- この機能は，主に MACRO 疑似命令で定義したマクロ・ボディ中で，条件付きアセンブル（「4.7 条件付きアセンブル制御命令」を参照）機能を用いている場合に使用します。
- マクロ・ボディ中で，条件付きアセンブル機能を組み合わせて使用している場合，EXITM 疑似命令で強制的にマクロを抜けないと，アセンブルされてはならない部分がアセンブルされてしまう場合があります。このようなときに，EXITM 疑似命令を使用します。

【説明】

- マクロ・ボディ中に EXITM 疑似命令を記述した場合，マクロ・ボディとしては，ENDM 疑似命令までが登録されます。
- EXITM 疑似命令は，マクロ展開時にのみマクロの終了を指示します。
- オペランド欄に何かの記述がある場合には，エラーが出力されますが，EXITM 疑似命令の処理は行われます。
- EXITM 疑似命令が現れると，アセンブルは，IF/_IF/ELSE/ELSEIF/_ELSEIF/ENDIF のネスティング・レベルを，そのマクロ・ボディに入ったときのネスティング・レベルまで強制的に戻します。
- マクロ・ボディ中に記述されたインクルード制御命令を展開したときに，インクルード・ファイル中の EXITM 疑似命令が現れた場合は，その EXITM 疑似命令を有効とし，そのレベルのマクロ展開を中止します。

【使用例】

```
        NAME    SAMP1
MAC1    MACRO                           ; (1)
          ; マクロボディ
        NOT1    CY
$       IF ( SW1 )                      ; (2)    ← IF ブロック
                BT      A.1 , $L1
                EXITM                   ; (3)
$       ELSE                            ; (4)    ← ELSE ブロック
                MOV1    CY , A.1
                MOV     A , #0
$       ENDIF                           ; (5)
$       IF ( SW2 )                      ; (6)    ← IF ブロック
                BR      [ HL ]
$       ELSE                            ; (7)    ← ELSE ブロック
                BR      [ DE ]
$       ENDIF                           ; (8)
          ; ソース本文
        ENDM                            ; (9)

        CSEG
$       SET ( SW1 )                     ; (10)
        MAC1                            ; (11)   ←マクロ参照
L1:     NOP

        END
```

(1) マクロ MAC1 は，マクロ・ボディ内で条件付きアセンブル機能（(2)，(4)-(8)）を使用しています。

(2) 条件付きアセンブルの IF ブロックを定義します。

スイッチ名 SW1 が真（非 0）の場合，IF ブロックがアセンブルされます。

(3) (4) 以降のマクロ・ボディの展開を強制的に終了します。

この (3) の記述がないと，マクロが展開されたとき，アセンブルは (6) 以降のアセンブル処理に移ります。

(4) 条件付きアセンブルの ELSE ブロックを定義します。

スイッチ名 SW1 が偽（0）の場合，ELSE ブロックがアセンブルされます。

(5) 条件付きアセンブルの終了を示します。

(6) 再び，条件付きアセンブルの IF ブロックを定義します。

スイッチ名 SW2 が真（非 0）の場合，これに続く IF ブロックがアセンブルされます。

(7) 条件付きアセンブルの ELSE ブロックを定義します。

スイッチ名 SW2 が偽（0）の場合，ELSE ブロックがアセンブルされます。

(8) (6)，(7) の条件付きアセンブルの終了を示します。

(9) マクロ・ボディの終了を示します。

(10)SET 制御命令で，スイッチ名 SW1 に真の値（非 0）を与え，条件付きアセンブルの条件を設定します。

(11)マクロ MAC1 を参照しています。

備考 使用例には，条件付きアセンブル制御命令を記述してあります。詳細については，「4.7 条件付きアセンブル制御命令」を参照してください。マクロ・ボディ，マクロ展開については，「第 5 章 マクロ」を参照してください。

使用例のアセンブル・リストを次に示します。

＜アセンブル・リスト＞

```
        NAME    SAMP1
MAC1    MACRO                   ; (1)
          :
        ENDM                    ; (9)
        CSEG
$       SET ( SW1 )             ; (10)
        MAC1                    ; (11)
          ; マクロ展開部
        NOT1    CY
$       IF ( SW1 )
                BT      A.1 , $L1
          ; ソース本文
L1 :    NOP

        END
```

(11) のマクロ参照により，マクロ MAC1 のマクロ・ボディが展開されています。

(10) でスイッチ名 SW1 に真の値を設定しているため，マクロ・ボディ内の最初の IF ブロックがアセンブルされます。IF ブロックの最後に EXITM 疑似命令があるため，それ以降のマクロ・ボディは展開されていません。

ENDM

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
なし	ENDM	*なし*	[; *コメント*]

【機能】

- マクロの機能として定義される一連のステートメントの終了をアセンブラに指示します。

【用途】

- MACRO 疑似命令，REPT 疑似命令，および IRP 疑似命令に続く一連のマクロ・ステートメントの最後には，必ず ENDM 疑似命令を記述します。

【説明】

- MACRO 疑似命令と ENDM 疑似命令の間に記述された一連のマクロ・ステートメントがマクロ・ボディとなります。
- REPT 疑似命令と ENDM 疑似命令の間に記述された一連のステートメントが，REPT-ENDM ブロックとなります。
- IRP 疑似命令と ENDM 疑似命令の間に記述された一連のステートメントが，IRP-ENDM ブロックとなります。

【使用例】

＜例 1：MACRO-ENDM ＞

```
        NAME    SAMP1
ADMAC   MACRO   PARA1 , PARA2
                MOV     A , #PARA1
                ADD     A , #PARA2
        ENDM
          :
        END
```

＜例 2：REPT-ENDM ＞

```
NAME    SAMP2
CSEG
  :
REPT    3
        INC     B
        DEC     C
ENDM
  :
END
```

＜例 3：IRP-ENDM ＞

```
NAME    SAMP3
CSEG
  :
IRP     PARA , <1 , 2 , 3>
        ADD     A , #PARA
        MOV     [ DE ] , A
ENDM
  :
END
```

3.9 アセンブル終了疑似命令

アセンブル終了疑似命令は，アセンブラにソース・モジュールの終了を指示します。ソース・モジュールの最後には，必ずアセンブル終了疑似命令を記述します。

アセンブラは，アセンブル終了疑似命令までをソース・モジュールとして処理します。したがって，REPT ブロックや IRP ブロックで，ENDM より前にアセンブル終了疑似命令があると，REPT ブロックと IRP ブロックは無効になります。

アセンブル終了疑似命令には，次のものがあります。

- END

END

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
なし	END	*なし*	[; *コメント*]

【機能】

- ソース・モジュールの終了をアセンブラに宣言します。

【用途】

- END 疑似命令は，ソース・モジュールの最後に必ず記述します。

【説明】

- アセンブラは，END 疑似命令が現れるまでソース・モジュールをアセンブルします。したがって，ソース・モジュールの最後には，END 疑似命令が必要です。
- END 疑似命令のあとにも，必ず改行コード（LF）を入力してください。
- END 疑似命令のあとに，空白，タブ，改行コード，コメント以外のステートメントがある場合には，ワーニングが出力されます。

【使用例】

```
NAME    SAMPLE
DSEG
  :
CSEG
  :
END                     ; (1)
```

(1) ソース・モジュールの最後には，必ず END 疑似命令を記述します。

第 4 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，アセンブラの動作に対し細かい指示を与えるものです。

4.1　概要

制御命令は，アセンブラの動作に対し細かい指示を与えるもので，ソース中に記述します。

制御命令は，オブジェクト・コード生成の対象とはなりません。

次に，制御命令の種類を示します。

表 4-1　制御命令一覧

制御命令の種類	制御命令
アセンブル対象品種指定制御命令	PROCESSOR
デバッグ情報出力制御命令	DEBUG/NODEBUG，DEBUGA/NODEBUGA
クロスリファレンス・リスト出力指定制御命令	XREF/NOXREF，SYMLIST/NOSYMLIST
インクルード制御命令	INCLUDE
アセンブル・リスト制御命令	EJECT，LIST/NOLIST，GEN/NOGEN，COND/NOCOND，TITLE，SUBTITLE，FORMFEED/NOFORMFEED，WIDTH，LENGTH，TAB
条件付きアセンブル制御命令	IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF，SET/RESET
漢字コード制御命令	KANJICODE
その他の制御命令	TOL_INF，DGS，DGL

制御命令は，疑似命令と同様に，ソース中に記述します。

また，表 4-1 で示した制御命令のうち，次に示すものは，アセンブラを起動するときにアセンブラ・オプションとしてコマンド行でも指定することができます。

表 4-2 制御命令とアセンブラ・オプション

制御命令	アセンブラ・オプション
PROCESSOR	-c
DEBUG/NODEBUG	-g/-ng
DEBUGA/NODEBUGA	-ga/-nga
XREF/NOXREF	-kx/-nkx
SYMLIST/NOSYMLIST	-ks/-nks
TITLE	-lh
FORMFEED/NOFORMFEED	-lf/-nlf
WIDTH	-lw
LENGTH	-ll
TAB	-lt
KANJICODE	-zs/-ze/-zn

コマンド行での指定方法については，「RA78K0R 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。

4.2 アセンブル対象品種指定制御命令

アセンブル対象品種指定制御命令は，ソース・モジュール・ファイル中でアセンブル対象品種を指定します。

アセンブル対象品種指定制御命令には，次のものがあります。

- PROCESSOR

PROCESSOR

【記述形式】

```
[ △ ]$[ △ ]PROCESSOR[ △ ]([ △ ] 品種名 [ △ ])
[ △ ]$[ △ ]PC[ △ ]([ △ ] 品種名 [ △ ])                ; 短縮形
```

【機能】

- PROCESSOR 制御命令は，ソース・モジュール・ファイル中でアセンブル対象品種を指定します。

【用途】

- アセンブル対象品種指定は，ソース・モジュール・ファイル，またはコマンド・ラインのどちらかで必ず指定しなければなりません。
- ソース・モジュール・ファイル中でアセンブル対象品種指定の記述がない場合，アセンブルのたびにアセンブル対象品種を指定しなければなりません。したがって，ソース・モジュール・ファイル中でアセンブル対象品種を指定しておくことにより，アセンブラ起動時の手間を省くことができます。

【説明】

- PROCESSOR 制御命令は，モジュール・ヘッダ部にのみ記述することができます。その他に記述した場合，アセンブラはアボートします。
- 指定可能な品種名ついては，各デバイスのユーザーズ・マニュアル，または「デバイス・ファイル　使用上の留意点」を参照してください。
- 指定した品種名がアセンブル対象品種と異なる場合，アセンブラはアボートします。
- PROCESSOR 制御命令は，複数指定することはできません。
- アセンブル対象品種指定は，コマンド・ライン上でアセンブラ・オプション（-c）によっても指定できます。ソース・モジュール・ファイル中とコマンド・ラインでの指定が異なる場合，アセンブラはワーニングを出力し，コマンド・ラインでの指定を優先します。
- アセンブラ・オプション（-c）を指定した場合でも，PROCESSOR 制御命令に対する文法チェックは行われます。
- ソース・モジュール・ファイル中，コマンド・ラインのどちらも指定されていない場合，アセンブラはアボートします。

【使用例】

```
$       PROCESSOR ( f1166a0 )
$       DEBUG
$       XREF

        NAME    TEST
          :
        CSEG
```

4.3 デバッグ情報出力制御命令

デバッグ情報出力制御命令は，ソース・モジュール・ファイル中で，オブジェクト・モジュール・ファイルに対してデバッグ情報の出力を指定することができます。

デバッグ情報出力制御命令には，次のものがあります。

- DEBUG/NODEBUG
- DEBUGA/NODEBUGA

DEBUG/NODEBUG

【記述形式】

```
[ △ ]$[ △ ]DEBUG          ; 省略時解釈
[ △ ]$[ △ ]DG             ; 短縮形
[ △ ]$[ △ ]NODEBUG
[ △ ]$[ △ ]NODG           ; 短縮形
```

【機能】

- DEBUG 制御命令は，オブジェクト・モジュール・ファイル中にローカル・シンボル情報を付加します。
- NODEBUG 制御命令は，オブジェクト・モジュール・ファイル中にローカル・シンボル情報を付加しません。なお，この場合にも，セグメント名はオブジェクト・モジュール・ファイルに出力されます。
- ローカル・シンボル情報とは，モジュール名，PUBLIC，EXTRN，EXTBIT シンボル以外のシンボルのことを示します。

【用途】

- DEBUG 制御命令は，ローカル・シンボルも含め，シンボリック・デバッグを行うときに使用します。
- NODEBUG 制御命令は，次の 3 種類の場合に使用します。

  (1) グローバル・シンボルのみのシンボリック・デバッグ

  (2) シンボルなしでのデバッグ

  (3) オブジェクトのみを必要とするとき（PROM による評価時など）

【説明】

- DEBUG/NODEBUG 制御命令は，モジュール・ヘッダ部のみに記述することができます。
- DEBUG/NODEBUG 制御命令を省略した場合，アセンブラは DEBUG 制御命令が指定されたと解釈して処理を行います。
- DEBUG/NODEBUG 制御命令が複数回記述された場合は，後者が優先されます。
- ローカル・シンボル情報の付加は，コマンド・ライン上でアセンブラ・オプション（-g/-ng）によっても指定することができます。
- ソース・モジュール・ファイル中とコマンド・ライン上で異なる指定が行われた場合，コマンド・ライン上の指定が優先されます。
- アセンブラ・オプション（-ng）を指定した場合でも，DEBUG/NODEBUG 制御命令に対する文法チェックは行われます。

DEBUGA/NODEBUGA

【記述形式】

```
[ △ ]$[ △ ]DEBUGA          ; 省略時解釈
[ △ ]$[ △ ]NODEBUGA
```

【機能】

- DEBUGA 制御命令は，オブジェクト・モジュール・ファイル中にアセンブラ・ソース・デバッグ情報を付加します。
- NODEBUGA 制御命令は，オブジェクト・モジュール・ファイル中にアセンブラ・ソース・デバッグ情報を付加しません。

【用途】

- DEBUGA 制御命令は，アセンブラのソース・レベルで，デバッグするときに使用します。なお，ソース・レベルでのデバッグには，“統合デバッガ”が必要です。
- NODEBUGA 制御命令は，次の 2 種類の場合に使用します。

  (1) アセンブラ・ソースなしでのデバッグ

  (2) オブジェクトのみを必要とするとき（PROM による評価時など）

【説明】

- DEBUGA/NODEBUGA 制御命令は，モジュール・ヘッダ部のみに記述することができます。
- DEBUGA/NODEBUGA 制御命令を省略した場合，アセンブラは DEBUGA 制御命令が指定されたと解釈して，処理を行います。
- DEBUGA/NODEBUGA 制御命令が複数回記述された場合は，後者が優先されます。
- アセンブラ・ソース・デバッグ情報の付加は，コマンド・ライン上でアセンブラ・オプション（-ga/-nga）によっても指定することができます。
- ソース・モジュール・ファイル中とコマンド・ライン上で異なる指定が行われた場合，コマンド・ライン上の指定が優先されます。
- アセンブラ・オプション（-nga）を指定した場合でも，DEBUGA/NODEBUGA 制御命令に対する文法チェックは行われます。
- C コンパイラでデバッグ情報を出力して，コンパイルした場合，その出力アセンブル・ソースをアセンブルするときには，デバッグ情報出力制御命令を記述しないでください。アセンブル時に必要な制御命令は，C コンパイラが，アセンブラ・ソース中に制御文として出力します。

4.4 クロスリファレンス・リスト出力指定制御命令

クロスリファレンス・リスト出力指定制御命令は，ソース・モジュール・ファイル中でクロスリファレンス・リストの出力指定を行うことができます。

クロスリファレンス・リスト出力指定制御命令には，次のものがあります。

- XREF/NOXREF
- SYMLIST/NOSYMLIST

XREF/NOXREF

【記述形式】

```
[ △ ]$[ △ ]XREF
[ △ ]$[ △ ]XR              ; 短縮形
[ △ ]$[ △ ]NOXREF          ; 省略時解釈
[ △ ]$[ △ ]NOXR            ; 短縮形
```

【機能】

- XREF 制御命令は，アセンブル・リスト・ファイルにクロスリファレンス・リストを出力することを指示します。
- NOXREF 制御命令は，アセンブル・リスト・ファイルにクロスリファレンス・リストを出力しないことを指示します。

【用途】

- ソース・モジュール・ファイルで定義された各シンボルがソース・モジュール中のどこでどれだけ参照されているか，また，アセンブル・リストの何行目の記述で，そのシンボルを参照しているのか，などの情報を知りたいときに，クロスリファレンス・リストを出力します。
- アセンブルのたびに，クロスリファレンス・リスト出力指定を行うような場合には，ソース・モジュール・ファイル中にこれらを記述することにより，アセンブラ起動時の手間を省くことができます。

【説明】

- XREF/NOXREF 制御命令は，モジュール・ヘッダ部のみに記述することができます。
- XREF/NOXREF 制御命令が複数指定された場合には，後者が優先されます。
- クロスリファレンス・リスト指定は，コマンド・ライン上でアセンブラ・オプション（-kx/-nkx）によって指定することができます。
- ソース・モジュール・ファイル中とコマンド・ライン上で異なる指定が行われた場合，コマンド・ライン上の指定が優先されます。
- アセンブラ・オプション（-np）を指定した場合でも，XREF/NOXREF 制御命令に対する文法チェックは行われます。

SYMLIST/NOSYMLIST

【記述形式】

```
[ △ ]$[ △ ]SYMLIST
[ △ ]$[ △ ]NOSYMLIST      ; 省略時解釈
```

【機能】

- SYMLIST 制御命令は，リスト・ファイルにシンボル・リストを出力することを指示します。
- NOSYMLIST 制御命令は，リスト・ファイルにシンボル・リストを出力しないことを指示します。

【用途】

- シンボル・リストを出力したい場合に使用します。

【説明】

- SYMLIST/NOSYMLIST 制御命令は，モジュール・ヘッダ部のみに記述できます。
- SYMLIST/NOSYMLIST 制御命令が複数指定された場合には，後者が優先されます。
- シンボル・リストの出力は，コマンド・ライン上でアセンブラ・オプション（-ks/-nks）によっても指定することができます。
- ソース・モジュール・ファイル中とコマンド・ライン上で異なる指定が行われた場合，コマンド・ライン上の指定が優先されます。
- アセンブラ・オプション（-np）を指定した場合でも，SYMLIST/NOSYMLIST 制御命令に対する文法チェックは行われます。

4.5 インクルード制御命令

インクルード制御命令は，ソース・モジュール中でほかのソース・モジュール・ファイルを引用する場合に使用します。

インクルード制御命令を有効に使用することにより，ソースの記述の手間を軽減することができます。

インクルード制御命令には，次のものがあります。

- INCLUDE

INCLUDE

【記述形式】

```
[ △ ]$[ △ ]INCLUDE[ △ ]([ △ ] ファイル名 [ △ ])
[ △ ]$[ △ ]IC[ △ ]([ △ ] ファイル名 [ △ ])              ; 短縮形
```

【機能】

- 指定されたファイルの内容を，指定された行以降に挿入展開し，アセンブルします。

【用途】

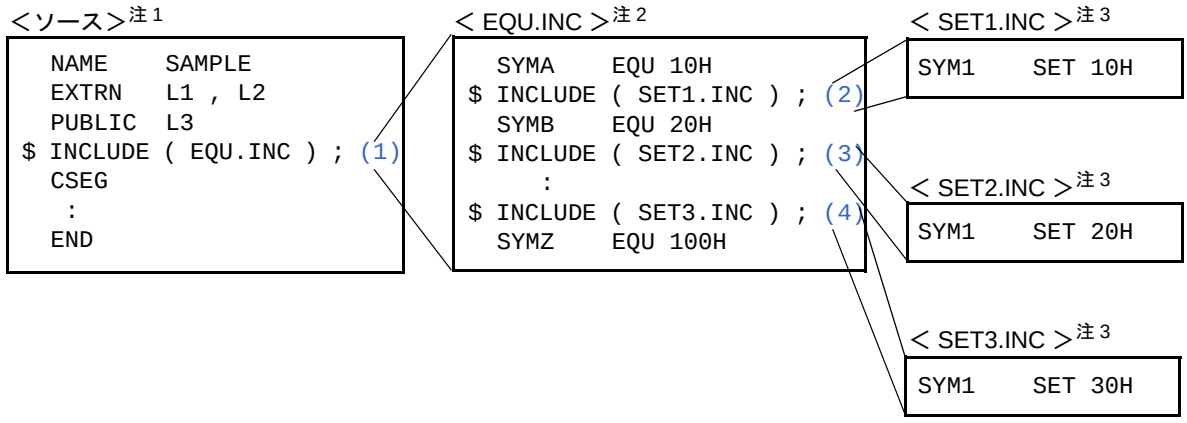
- 複数のソース・モジュール中で共通に記述する比較的大きな一連のステートメントを，1 つのファイル（インクルード・ファイル）としてまとめておきます。
  各ソース・モジュール中で，その一連のステートメントを引用する必要があるとき，INCLUDE 制御命令により，必要とするインクルード・ファイル名を指定します。
  これにより，ソース・モジュールの記述作業を軽減することができます。

【説明】

- INCLUDE 制御命令は，通常のソースにのみ記述することができます。
- アセンブラ・オプション（-I）で，インクルード・ファイルのパス名やドライブ名を指定することができます。
- インクルード・ファイルの読み込みパスのサーチの順番は，次のとおりです。
  (1) インクルード・ファイルがパス名なしで指定された場合
    (a) ソース・ファイルのあるパス
    (b) アセンブラ・オプション（-I）で指定されたパス
    (c) 環境変数 INC78K0R で指定されたパス
  (2) インクルード・ファイルがパス名付きで指定された場合
    ドライブ名，またはバックスラッシュ（ \ ）から始まるパス名付きで指定した場合には，インクルード・ファイル名に付いているパス，相対パス（先頭に \ がない）付きで指定された場合には，インクルード・ファイル名の前に (1) の順でパス名を付加します。
- インクルード・ファイルは，7 重までネスティングすることが可能です。つまり，ネスト・レベルの最大数は 8 です（ネスティングとは，インクルード・ファイル中で，別のインクルード・ファイルを指定することです）。
- インクルード・ファイルに，END 疑似命令の記述は必要ありません。
- インクルード・ファイルがオープンできない場合，アセンブラはアボートします。

- インクルード・ファイル中は，“IF，または _IF ～ ENDIF”の対応がとれた状態で，閉じなければなりません。もし，インクルード・ファイルの展開の入口の IF レベルと，展開終了直後の IF レベルの対応がとれていない場合，アセンブラはエラーを出力し，レベルを強制的に入口でのレベルに戻してアセンブルを続けます。
- インクルード・ファイル中でマクロを定義するときは，そのマクロ定義はインクルード・ファイル中で閉じていなければなりません。
  突然 ENDM 疑似命令が現れた場合には，エラーが出力され，その ENDM 疑似命令は無視されます。また，マクロ定義疑似命令があるのにそのインクルード・ファイル中に ENDM 疑似命令がない場合には，エラーが出力され，ENDM 疑似命令があるものとして処理されます。
- インクルード・ファイル中に，2 つ以上のセグメントを定義することはできません。定義した場合は，エラーが出力されます。

【使用例】



(1) インクルード・ファイルとして，“EQU.INC”を指定しています。

(2) インクルード・ファイルとして，“SET1.INC”を指定しています。

(3) インクルード・ファイルとして，“SET2.INC”を指定しています。

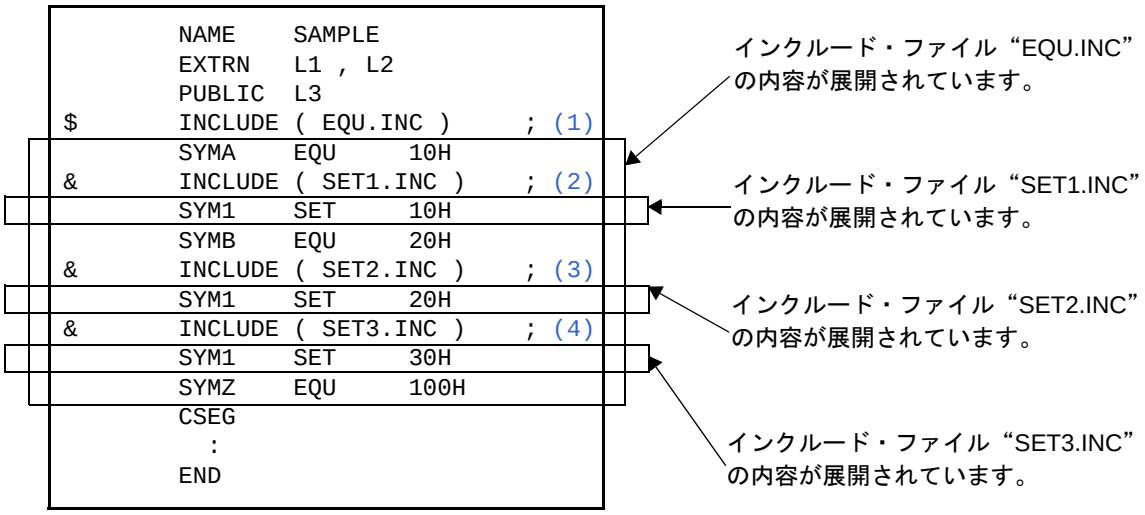
(4) インクルード・ファイルとして，“SET3.INC”を指定しています。

注 1 ソース・ファイル中には，$IC を複数指定することができます。また，同じインクルード・ファイルを複数指定することもできます。

注 2 EQU.INC には，$IC を複数指定することができます。

注 3 SET1.INC，SET2.INC，および SET3.INC 中には，$IC を指定することはできません。

このソースがアセンブルされると，インクルード・ファイルの内容が次のように展開されます。



4.6 アセンブル・リスト制御命令

アセンブル・リスト制御命令は，アセンブラの出力するアセンブル・リストに対して，改ページ，リスト出力をしない部分，サブタイトル・メッセージ出力などを指示するものです。

アセンブル・リスト制御命令には，次のものがあります。

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

EJECT

【記述形式】

```
[ △ ]$[ △ ]EJECT
[ △ ]$[ △ ]EJ              ; 短縮形
```

【省略時解釈】

- EJECT 制御命令は，指定していないものとします。

【機能】

- EJECT 制御命令は，アセンブル・リストの改ページをアセンブラに指示します。

【用途】

- ソース・モジュール中で改ページを行いたい箇所に記述します。

【説明】

- EJECT 制御命令は，通常のソースのみに記述することができます。
- EJECT 制御命令自身のイメージを出力したあとに，リストを改ページします。
- コマンド・ラインでアセンブラ・オプション（-np），（-ll0）の指定がある場合や，制御命令の指定によりリスト出力禁止状態の場合，EJECT 制御命令は無効です。
  アセンブラ・オプション（-np），（-ll0）については，「RA78K0R 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。
- EJECT 制御命令のあとに不正な記述があった場合，アセンブラはエラーを出力します。

【使用例】

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

(1) EJECT 制御命令により改ページを行います。

使用例のアセンブル・リストを次に示します。

＜アセンブル・リスト＞

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

LIST/NOLIST

【記述形式】

```
[ △ ]$[ △ ]LIST          ; 省略時解釈
[ △ ]$[ △ ]LI            ; 短縮形
[ △ ]$[ △ ]NOLIST
[ △ ]$[ △ ]NOLI          ; 短縮形
```

【機能】

- LIST 制御命令は，アセンブル・リストの出力開始位置をアセンブラに指示します。
- NOLIST 制御命令は，アセンブル・リストの出力中止位置をアセンブラに指示します。
  NOLIST 制御命令を指定したあと，次に LIST 制御命令が現れるまでのステートメントは，アセンブルされますがアセンブル・リストには出力されません。

【用途】

- NOLIST 制御命令は，リストの出力量を制限するために使用します。
- LIST 制御命令は，NOLIST 制御命令で指定したアセンブル・リスト出力中止の状態を，再びアセンブル・リスト出力の状態にする場合に使用します。
  NOLIST 制御命令と LIST 制御命令を組み合わせて使用することにより，出力するアセンブル・リストの量や印字内容を制御することができます。

【説明】

- LIST/NOLIST 制御命令は，通常のソースにのみに記述することができます。
- NOLIST 制御命令は，アセンブル・リストの出力を中止するもので，アセンブルを中止するものではありません。
- NOLIST 制御命令以降，LIST 制御命令の指定があると，LIST 制御命令以降のステートメントは再びアセンブル・リストに出力されます。記述した LIST/NOLIST 制御命令自身もアセンブル・リストに出力されます。
- LIST/NOLIST 制御命令を省略した場合には，すべてのステートメントがアセンブル・リストに出力されます。

【使用例】

```
        NAME    SAMP1
$       NOLIST          ; (1)
DATA1   EQU     10H     ; アセンブル・リストに出力されません。
DATA2   EQU     11H     ; アセンブル・リストに出力されません。
          :             ; アセンブル・リストに出力されません。
DATAX   EQU     20H     ; アセンブル・リストに出力されません。
DATAY   EQU     20H     ; アセンブル・リストに出力されません。
$       LIST            ; (2)
        CSEG
          :
        END
```

(1) NOLIST 制御命令を指定しているので，これ以降，(2) の LIST 制御命令までのステートメントは，アセンブル・リストに出力されません。
NOLIST 制御命令自身は出力されます。

(2) LIST 制御命令を指定しているので，これ以降のステートメントは，再びアセンブル・リストに出力されます。LIST 制御命令自身は出力されます。

GEN/NOGEN

【記述形式】

```
[ △ ]$[ △ ]GEN              ; 省略時解釈
[ △ ]$[ △ ]NOGEN
```

【機能】

- GEN 制御命令は，マクロ定義部，参照行，およびマクロ展開部をアセンブル・リストに出力します。
- NOGEN 制御命令はマクロ定義部，および参照行はアセンブル・リストに出力しますが，マクロ展開部は出力されません。

【用途】

- アセンブル・リストの出力量を制限するために使用します。

【説明】

- GEN/NOGEN 制御命令は，通常のソースのみに記述することができます。
- GEN/NOGEN 制御命令を省略した場合，マクロ定義部，参照行，およびマクロ展開部をアセンブル・リストに出力します。
- GEN/NOGEN 制御命令自身のイメージが出力されたあとに，リスト制御が行われます。
- NOGEN 制御命令でリスト出力中断後もアセンブルは続けられ，アセンブル・リスト上のステートメント・ナンバ（STNO）の値がカウント・アップされます。
- NOGEN 制御命令のあと，GEN 制御命令が指定された場合には，再びマクロ展開部の出力が開始されます。

【使用例】

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

        END
```

使用例のアセンブル・リストを次に示します。

＜アセンブル・リスト＞

```
        NAME    SAMP1
$       NOGEN                           ; (1)
ADMAC   MACRO   PARA1 , PARA2
                MOV     A , #PARA1
                ADD     A ,  #PARA2
        ENDM
        CSEG
        ADMAC   10H , 20H
        MOV     A , #10H                ; マクロ展開部は出力されません。
        AUD     A , #20H                ; マクロ展開部は出力されません。

        END
```

(1) NOGEN 制御命令が指定されているので，マクロ展開部はリストに出力されません。

COND/NOCOND

【記述形式】

```
[ △ ]$[ △ ]COND            ; 省略時解釈
[ △ ]$[ △ ]NOCOND
```

【機能】

- COND 制御命令は，条件アセンブルの条件成立部分，および不成立部分をアセンブル・リストへ出力します。
- NOCOND 制御命令は，条件アセンブルの条件成立部分のみをアセンブル・リストに出力し，条件不成立部分，および IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF が記述されている行は，出力されません。

【用途】

- アセンブル・リストの出力量を制限するために使用します。

【説明】

- COND/NOCOND 制御命令は，通常のソースにのみに記述することができます。
- COND/NOCOND 制御命令を省略した場合，条件アセンブルの条件成立部分，および不成立部分をアセンブル・リストへ出力します。
- COND/NOCOND 制御命令自身のイメージが出力されたあとに，リスト制御が行われます。
- NOCOND 制御命令で，リスト出力中断後も ALNO，STNO がカウント・アップされます。
- NOCOND 制御命令のあと，COND 制御命令が指定された場合には，再び条件不成立部分，および IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF が記述されている行の出力が開始されます。

【使用例】

```
        NAME    SAMP

$       NOCOND
$       SET ( SW1 )
$       IF ( SW1 )                      ; アセンブルしてもリストには，出力されません。
                MOV     A , #1H
$       ELSE                            ; アセンブルしてもリストには，出力されません。
                MOV     A , #0H         ; アセンブルしてもリストには，出力されません。
$       ENDIF                           ; アセンブルしてもリストには，出力されません。

        END
```

TITLE

【記述形式】

```
[ △ ]$[ △ ]TITLE[ △ ]([ △ ]' タイトル・ストリング'[ △ ])
[ △ ]$[ △ ]TT[ △ ]([ △ ]' タイトル・ストリング'[ △ ])          ; 短縮形
```

【省略時解釈】

- TITLE 制御命令は指定されていないものとし，アセンブル・リストのヘッダのタイトル欄は空白となります。

【機能】

- TITLE 制御命令は，アセンブル・リスト，シンボル・テーブル・リスト，およびクロスリファレンス・リストの各ページのヘッダのタイトル欄に，印字する文字列を指定します。

【用途】

- タイトルを各ページに印字することにより，リストの内容が一目でわかります。
- アセンブルのたびにタイトル指定を行うような場合，ソース・モジュール・ファイル中にこれらを記述することにより，アセンブラ起動時の手間を省くことができます。

【説明】

- TITLE 制御命令は，モジュール・ヘッダ部のみに記述することができます。
- TITLE 制御命令が複数指定された場合には，あとで指定したものが有効となります。
- タイトル・ストリングは，60 文字以内です。61 文字以上の場合には，先頭の 60 文字を有効とします。ただし，アセンブル・リスト・ファイルの 1 行の文字数指定（X とします）が 119 文字以下の場合，“X - 60”文字以内とします。
- タイトル・ストリングに引用符（’）を記述する場合には，2 個続けて記述します。
- 文字数が 0 の場合，タイトル欄は空白になります。
- タイトル・ストリングに「2.2.2 文字セット」にない不当文字が記述された場合は，“！”に置き換えてタイトル欄に出力されます。
- タイトル指定は，コマンド・ライン上でアセンブラ・オプション（-lh）によって指定することができます。

【使用例】

```
$       PROCESSOR (f1166a0 )
$       TITLE ( 'THIS IS TITLE' )
        NAME    SAMPLE
        CSEG
        MOV     A , B

        END
```

使用例のアセンブル・リストを次に示します（行数は 72 行と指定しています）。

＜アセンブル・リスト＞

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

Command :       -ll72 sample.asm
Para-file :
In-file :       sample.asm
Obj-file :      sample.rel
Prn-file :      sample.prn

         Assemble list

ALNO    STNO    ADRS    OBJECT  M I     SOURCE STATEMENT

 1       1                              $       PROCESSOR ( f1166a0 )
 2       2                              $       TITLE ( 'THIS IS TITLE' )
 3       3                                      NAME    SAMPLE
 4       4       ----                           CSEG
 5       5       00000   63                     MOV     A , B
 6       6
 7       7                                      END

Segment information :

ADRS    LEN     NAME

00000   00001H  ?CSEG

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

SUBTITLE

【記述形式】

```
[ △ ]$[ △ ]SUBTITLE[ △ ]([ △ ]'タイトル・ストリング'[ △ ])
[ △ ]$[ △ ]ST[ △ ]([ △ ]'タイトル・ストリング'[ △ ])          ; 短縮形
```

【省略時解釈】

- SUBTITLE 制御命令は指定されていないものとし，アセンブル・リストのヘッダのサブタイトル部は空白となります。

【機能】

- アセンブル・リストの各ページ・ヘッダのサブタイトル部に印字する文字列を指定します。

【用途】

- サブタイトルを各ページに印字することにより，アセンブル・リストの内容をわかりやすくします。
  サブタイトルは，各ページごとに印字する文字列を変更することができます。

【説明】

- SUBTITLE 制御命令は，通常のソースにのみに記述することができます。
- 指定可能な文字列は，72 文字までです。
  73 文字以上記述した場合，先頭の 72 文字が有効です。なお，全角文字は 2 文字，タブは 1 文字として数えます。
- SUBTITLE 制御命令は指定した文字列を，その次のページからサブタイトル部に印字します。ただし，ページの先頭行に指定した場合には，そのページから印字します。
- SUBTITLE 制御命令を指定しない場合は，サブタイトル部は空白です。
- 文字列に引用符（'）を記述する場合には，2 個続けて記述してください。
- 文字数が 0 の場合，サブタイトル欄は空白となります。
- 指定された文字列の中に「2.2.2 文字セット」にない不当文字が記述された場合には，“!”に置き換えてサブタイトル欄に出力します。CR（0DH）を記述した場合は，エラーとなり，リスト上には何も出力されません。00H を記述すると，それ以降，引用符（'）で閉じるまで出力されません。

【使用例】

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

        END
```

(1) 文字列“THIS IS SUBTITLE 1”を指定します。

(2) 改ページ指示です。

(3) 文字列“THIS IS SUBTITLE 2”を指定します。

(4) 改ページ指示です。

(5) 文字列“THIS IS SUBTITLE 3”を指定します。

使用例のアセンブル・リストを次に示します（行数は 80 行です）。

＜アセンブル・リスト＞

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

Command :       -cf1166a0 -ll80 sample.asm
Para-file :
In-file :       sample.asm
Obj-file :      sample.rel
Prn-file :      sample.prn
        Assemble list

ALNO  STNO  ADRS  OBJECT  M I SOURCE STATEMENT

 1     1                          NAME SAMP
 2     2    -----                      CSEG
 3     3                       $   SUBTITLE ( 'THIS IS SUBTITLE 1' )   ; (1)
 4     4                       $   EJECT                               ; (2)
-------------------------------------------- 改ページ ------------------------
78K0R Assembler Vx.xx                              Date:xx xxx xx  Page:2

THIS IS SUBTITLE 1

ALNO  STNO  ADRS  OBJECT  M I SOURCE STATEMENT

 5     5    -----                  CSEG
 6     6                       $   SUBTITLE ( 'THIS IS SUBTITLE 2' )   ; (3)
 7     7                       $   EJECT                               ; (4)
-------------------------------------------- 改ページ ------------------------
78K0R Assembler Vx.xx                              Date:xx xxx xx  Page:3

THIS IS SUBTITLE 2

ALNO  STNO  ADRS  OBJECT  M I SOURCE STATEMENT

 8     8                       $   SUBTITLE ( 'THIS IS SUBTITLE 3' )   ; (5)
 9     9
10    10                          END

Segment informations :

ADRS  LEN     NAME

00000 00000H  ?CSEG

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

FORMFEED/NOFORMFEED

【記述形式】

```
[ △ ]$[ △ ]FORMFEED
[ △ ]$[ △ ]NOFORMFEED      ; 省略時解釈
```

【機能】

- FORMFEED 制御命令は，リスト・ファイルの最後にフォーム・フィードを出力することを指示します。
- NOFORMFEED 制御命令は，リスト・ファイルの最後にフォーム・フィードを出力しないことを指示します。

【用途】

- アセンブル・リスト・ファイルの内容を印字したあとで，改ページしておきたい場合に使用します。

【説明】

- FORMFEED/NOFORMFEED 制御命令は，モジュール・ヘッダ部のみに記述することができます。
- アセンブル・リストをプリント・アウトするとき，プリント・アウトが終了しても印字が最終ページの途中だと，最後のページが出てこない場合があります。
  このような場合，FORMFEED 制御命令，またはアセンブラ・オプション（-lf）を使用して，アセンブル・リストの最後に FORMFEED コードを付けてください。
  なお，多くの場合，ファイルの終了により FORMFEED コードが送られるので，最後に FORMFEED コードがあると不要な白紙が 1 ページ送られてしまいます。これを防止するために，NOFORMFEED 制御命令，またはアセンブラ・オプション（-nlf）がデフォルトで設定されます。
- FORMFEED/NOFORMFEED 制御命令を複数指定した場合は，最後に指定した命令が有効となります。
- フォーム・フィードの出力は，コマンド・ライン上でアセンブラ・オプション（-lf/-nlf）によっても指定することができます。
- ソース・モジュール・ファイル中とコマンド・ライン上で異なる指定が行われた場合，コマンド・ライン上の指定が優先されます。
- アセンブラ・オプション（-np）を指定した場合でも，FORMFEED/NOFORMFEED 制御命令に対する文法チェックは行われます。

WIDTH

【記述形式】

[△]$[△]WIDTH[△]([△] *文字数* [△])

【省略時解釈】

$WIDTH (132)

【機能】

- WIDTH 制御命令は，リスト・ファイルの 1 行の最大文字数を指示します。
  なお，文字数の指定範囲は，72 - 260 です。

【用途】

- 各種リスト・ファイルの 1 行の文字数を変更したい場合に使用します。

【説明】

- WIDTH 制御命令は，モジュール・ヘッダ部のみに記述することができます。
- WIDTH 制御命令が複数指定された場合は，最後に指定した命令が有効となります。
- 行文字数は，コマンド・ライン上でアセンブラ・オプション（-lw）によっても指定することができます。
- ソース・モジュール・ファイル中とコマンド・ライン上で異なる指定が行われた場合，コマンド・ライン上の指定が優先されます。
- アセンブラ・オプション（-np）を指定した場合でも，WIDTH 制御命令に対する文法チェックは行われます。

LENGTH

【記述形式】

```
[ △ ]$[ △ ]LENGTH[ △ ]([ △ ] 行数 [ △ ])
```

【省略時解釈】

$LENGTH (66)

【機能】

- LENGTH 制御命令は，リスト・ファイルの 1 ページの行数を指示します。
  なお，行数の指定範囲は，0，および 20 - 32767 です。

【用途】

- アセンブル・リスト・ファイルの 1 ページの行数を変更したい場合に使用します。

【説明】

- LENGTH 制御命令は，モジュール・ヘッダ部のみに記述することができます。
- LENGTH 制御命令が複数指定された場合は，最後に指定した命令が有効となります。
- 行数は，コマンド・ライン上でアセンブラ・オプション（-ll）によっても指定することができます。
- ソース・モジュール・ファイル中とコマンド・ライン上で異なる指定が行われた場合，コマンド・ライン上の指定が優先されます。
- アセンブラ・オプション（-np）を指定した場合でも，LENGTH 制御命令に対する文法チェックは行われます。

TAB

【記述形式】

[△]$[△]TAB[△]([△] *展開文字数* [△])

【省略時解釈】

$TAB (8)

【機能】

- TAB 制御命令は，リスト・ファイルのタブの展開文字数を指示します。
  なお，展開文字数の指定範囲は，0 - 8 です。
- TAB 制御命令は，ソース・モジュール中の HT（Horizontal Tabulation）コードを，各種リスト上でいくつかのブランク（空白）に置き換えて出力する（タビュレーション処理）ための基本となる文字数を指定します。

【用途】

- TAB 制御命令で，各種リストの 1 行の文字数を少なく指定した場合に，HT コードによるブランクを少なくし文字数を節約します。

【説明】

- TAB 制御命令は，モジュール・ヘッダ部のみに記述することができます。
- TAB 制御命令が複数指定された場合は，最後に指定した命令が有効となります。
- タブの展開文字数は，コマンド・ライン上でアセンブラ・オプション（-lt）によっても指定することができます。
- ソース・モジュール・ファイル中とコマンド・ライン上で異なる指定が行われた場合，コマンド・ライン上の指定が優先されます。
- アセンブラ・オプション（-np）を指定した場合でも，TAB 制御命令に対する文法チェックは行われます。

4.7 条件付きアセンブル制御命令

条件付きアセンブル制御命令は，ソース・モジュール中のある一連のステートメントをアセンブルの対象とする／しないを，条件付きアセンブルのスイッチ設定により選択するものです。

条件付きアセンブル制御命令を有効に使用すると，ソース・モジュールをほとんど変更せずに，不必要なステートメントを除いたアセンブルを行うことができます。

条件付きアセンブル制御命令には，次のものがあります。

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

IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF

【記述形式】

```
[ △ ]$[ △ ]IF[ △ ]([ △ ] スイッチ名 [[ △ ]:[ △ ] スイッチ名 ] … [ △ ])
または [ △ ]$[ △ ]_IF △条件式
        :
[ △ ]$[ △ ]ELSEIF[ △ ]([ △ ] スイッチ名 [[ △ ]:[ △ ] スイッチ名 ] … [ △ ])
または [ △ ]$[ △ ]_ELSEIF △条件式
        :
[ △ ]$[ △ ]ELSE
        :
[ △ ]$[ △ ]ENDIF
```

【機能】

- アセンブル対象とするソース・ステートメントを限定するための条件を設定します。
  条件付きアセンブルの対象となるソース・ステートメントは，IF/_IF 制御命令から ENDIF 制御命令までです。
- IF/_IF 制御命令は，指定したスイッチ名，あるいは条件式の評価値が真（00H 以外）の場合，それ以降，次の条件付きアセンブル制御命令（ELSEIF/_ELSEIF/ELSE/ENDIF）が現れるまでのソース・ステートメントがアセンブルされます。そのあとのアセンブル処理は，ENDIF 制御命令の次のステートメントに移ります。
  スイッチ名，あるいは条件式の評価値が偽（00H）の場合，それ以降，次の条件付きアセンブル制御命令（ELSEIF/_ELSEIF/ELSE/ENDIF）が現れるまでのソース・ステートメントは，アセンブルされません。
- ELSEIF/_ELSEIF 制御命令は，それ以前に記述してあるすべての条件付きアセンブル制御命令の条件が不成立（評価値が偽）の場合にのみ，条件判定が行われます。
  ELSEIF/_ELSEIF 制御命令で指定するスイッチ名，あるいは条件式の評価値が真の場合，それ以降，次の条件付きアセンブル制御命令（ELSEIF/_ELSEIF/ELSE/ENDIF）が現れるまでのソース・ステートメントがアセンブルされます。そのあとのアセンブル処理は，ENDIF 制御命令の次のステートメントに移ります。
  ELSEIF/ELSEIF の評価値が偽の場合，それ以降，次の条件付きアセンブル制御命令（ELSEIF/_ELSEIF/ELSE/ENDIF）が現れるまでのソース・ステートメントはアセンブルされません。
- ELSE 制御命令については，それ以前に記述したすべての IF/_IF/ELSEIF/_ELSEIF 制御命令の条件が不成立（スイッチ名の値が偽）の場合，ELSE 制御命令以降 ENDIF 制御命令が現れるまでのソース・ステートメントがアセンブルされます。
- ENDIF 制御命令は，条件付きアセンブルの対象となるソース・ステートメントの終了をアセンブラに指示します。

【用途】

- ソース・モジュールを大幅に変更することなく，アセンブル対象となるソース・ステートメントを変更することができます。
- ソース・モジュール中に，プログラム開発中にのみ必要となるデバッグ文などを記述した場合，そのデバッグ文を機械語に変換する／しないを，条件付きアセンブルのスイッチ設定により選択することができます。

【説明】

- スイッチ名による条件判断を行う場合には，IF/ELSEIF 制御命令を使用し，条件式による条件判断を行う場合には，_IF/_ELSEIF 制御命令を使用します。
  両方を組み合わせて使用することもできます。つまり，1 つの IF，または _IF と ENDIF の対の中に，ELSEIF/_ELSEIF を組み合わせて使用することができます。
- 条件式には，絶対式を記述します。
- スイッチ名記述上の規則は，シンボル記述上の規則（「2.2.3 シンボル欄」を参照してください）と同じです。
  ただし，最大認識文字数は，常に 31 文字です。
- IF/ELSEIF 制御命令で複数のスイッチ名を指定する場合は，各スイッチ名をコロン（:）で区切ります。
  ただし，1 つのモジュール内で使用できるスイッチ名は，最大 5 つです。
- IF/ELSEIF 制御命令で複数のスイッチ名を指定した場合，そのいずれか 1 つの値が真であれば，条件成立します。
- IF/ELSEIF 制御命令で指定するスイッチ名の値は，SET/RESET 制御命令により設定します。
  したがって，IF/ELSEIF 制御命令で指定するスイッチの値が，前もってソース・モジュール中で SET/RESET 制御命令により設定されていない場合は，RESET されたものとみなされます。
- スイッチ名，または条件式に不適当な記述がある場合，アセンブラはエラーを出力し，条件判断を偽とします。
- この制御命令を記述する場合には，IF，または _IF と ENDIF を対応させてください。
- マクロ・ボディ中に本制御命令が記述され，本体の途中で EXITM の処理を行って，そのレベルのマクロを抜けた場合，IF レベルは，アセンブラによってマクロ・ボディの入口の IF レベルまで強制的に戻されます。この場合，エラーになりません。
- 1 つの IF-ENDIF 制御命令の間に，別の IF-ENDIF 制御命令を記述することをネスティングと呼びます（8 レベルまでのネスティングが可能です）。
- 条件付きアセンブルで，アセンブルをしないステートメントは，オブジェクト・コードは生成されませんが，アセンブル・リストにはそのまま出力されます。出力したくない場合は，$NOCOND 制御命令を使用します。

【使用例】

＜例 1＞

```
        text0
$       IF ( SW1 )      ; (1)
                text1
$       ENDIF           ; (2)
          :
        END
```

(1) スイッチ名“SW1”の値が真の場合，text1 の部分がアセンブルされます。
スイッチ名“SW1”の値が偽の場合，text1 の部分はアセンブルされません。
スイッチ名“SW1”の値は，text0 の部分で SET/RESET 制御命令により真／偽に設定されています。

(2) 条件付きアセンブル範囲の終了を示します。

＜例 2＞

```
        text0
$       IF ( SW1 )      ; (1)
                text1
$       ELSE            ; (2)
                text2
$       ENDIF           ; (3)
          :
        END
```

(1) スイッチ名“SW1”の値は，text0 の部分で SET/RESET 制御命令により真／偽に設定されています。
スイッチ名“SW1”の値が真の場合，text1 の部分をアセンブルし，text2 の部分はアセンブルされません。

(2) (1) のスイッチ名“SW1”の値が偽の場合，text1 の部分はアセンブルされず，text2 の部分がアセンブルされます。

(3) 条件付きアセンブル範囲の終了を示します。

＜例 3＞

```
        text0
$       IF ( SW1 : SW2 )        ; (1)
                text1
$       ELSEIF ( SW3 )          ; (2)
                text2
$       ELSEIF ( SW4 )          ; (3)
                text3
$       ELSE                    ; (4)
                text4
$       ENDIF                   ; (5)
          :
        END
```

(1) スイッチ名“SW1”，“SW2”，“SW3”の値は，text0 の部分で SET/RESET 制御命令により真／偽に設定されています。
スイッチ名“SW1”または“SW2”の値が真の場合，text1 の部分がアセンブルされ，text2，text3，text4 の部分はアセンブルされません。
スイッチ名“SW1”と“SW2”の値がともに偽の場合，text1 の部分はアセンブルされず，(2) 以降の条件付きアセンブルが行われます。

(2) (1) のスイッチ名“SW1”と“SW2”の値がともに偽で，かつスイッチ名“SW3”の値が真の場合，text2 の部分がアセンブルされ，text1，text3，text4 の部分はアセンブルされません。

(3) (1) のスイッチ名“SW1”，“SW2”と，(2) のスイッチ名“SW3”の値がともに偽で，かつスイッチ名“SW4”の値が真の場合，text3 の部分がアセンブルされ，text1，text2，text4 の部分はアセンブルされません。

(4) (1)，(2)，(3) のスイッチ名“SW1”，“SW2”，“SW3”，“SW4”の値がすべて偽の場合，text4 の部分がアセンブルされ，text1，text2，text3 の部分はアセンブルされません。

(5) 条件付きアセンブル範囲の終了を示します。

＜例 4＞

```
        text0
$       _IF ( SYMA )            ; (1)
                text1
$       _ELSEIF ( SYMB = SYMC ) ; (2)
                text2
$       ENDIF                   ; (3)
          :
        END
```

(1) シンボル名“SYMA”の値は，text0 の部分で EQU，または SET 疑似命令により，定義されています。
シンボル名“SYMA”の値が真（非 0）の場合，text1 の部分がアセンブルされ，text2 はアセンブルされません。

(2) シンボル名“SYMA”の値が偽（0）で SYMB と SYMC が同じ値をもつ場合，text2 がアセンブルされます。

(3) 条件付きアセンブル範囲の終了を示します。

SET/RESET

【記述形式】

```
[ △ ]$[ △ ]SET[ △ ]([ △ ] スイッチ名 [[ △ ]:[ △ ] スイッチ名 ] … [ △ ])
[ △ ]$[ △ ]RESET[ △ ]([ △ ] スイッチ名 [[ △ ]:[ △ ] スイッチ名 ] … [ △ ])
```

【機能】

- SET/RESET 制御命令は，IF/ELSEIF 制御命令で指定するスイッチ名に値を与えます。
- SET 制御命令で指定したスイッチ名に，真の値（0FFH）を与えます。
- RESET 制御命令で指定したスイッチ名に，偽の値（00H）を与えます。

【用途】

- IF/ELSEIF 制御命令で指定するスイッチ名に真の値（0FFH）を与えたいときは，SET 制御命令を記述します。
- IF/ELSEIF 制御命令で指定するスイッチ名に偽の値（00H）を与えたいときは，RESET 制御命令を記述します。

【説明】

- SET/RESET 制御命令では，スイッチ名を記述します。
  スイッチ名の記述上の規則は，シンボルの記述上の規則（「2.2.3 シンボル欄」を参照してください）と同じです。
  ただし，最大認識文字数は，常に 31 文字です。
- スイッチ名は，予約語，スイッチ名以外のユーザ定義シンボルと重複してもかまいません。
- SET/RESET 制御命令で複数のスイッチ名を指定する場合は，各スイッチ名をコロン（ : ）で区切ります。
  ただし，1 つのモジュール内で使用できるスイッチ名は，最大 1000 個です。
- 一度 SET したスイッチ名を RESET することができます。また，一度 RESET したスイッチ名を SET することができます。
- IF/ELSEIF 制御命令で指定するスイッチ名は，その制御命令を記述する以前のソース・モジュール中で，SET/RESET 制御命令により，少なくとも 1 回は定義しなければなりません。
- スイッチ名は，クロスリファレンス・リストには出力されません。

【使用例】

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

(1) スイッチ名“SW1”に真の値（0FFH）を与えます。

(2) (1) でスイッチ名“SW1”に真の値を与えていますので，text1 の部分がアセンブルされます。

(3) (2) から始まる条件付きアセンブル範囲の終了を示します。

(4) スイッチ名“SW1”と“SW2”に偽の値（00H）を与えます。

(5) スイッチ名“SW1”には (4) で偽の値を与えていますので，text2 の部分はアセンブルされません。

(6) スイッチ名“SW2”にも (4) で偽の値を与えていますので，text3 の部分もアセンブルされません。

(7) (5)，(6) のスイッチ名“SW1”，“SW2”の値がすべて偽のため，text4 の部分がアセンブルされます。

(8) (5) から始まる条件付きアセンブル範囲の終了を示します。

4.8 漢字コード制御命令

コメントに記述された漢字を，どの漢字コードで解釈するのかを指定する制御命令です。

漢字コード制御命令には，次のものがあります。

- KANJICODE

KANJICODE

【記述形式】

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

【省略時解釈】

- $KANJICODE SJIS

【機能】

- コメントに記述された漢字を，指定された漢字コードとして解釈します。
- 漢字コードは，SJIS/EUC/NONE を記述することができます。

  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。

【用途】

- コメント行の漢字の，漢字コードの解釈を指定するときに使用します。

【説明】

- KANJICODE 制御命令は，モジュール・ヘッダ部のみに記述することができます。
- KANJICODE 制御命令が，モジュール・ヘッダ部に複数回記述された場合は，その中でもっとも後者に記述された命令が優先されます。
- 漢字コード指定は，コマンド・ライン上でアセンブラ・オプション（-zs/-ze/-zn）によって指定することができます。
- ソース・モジュール中とコマンド・ライン上で異なる指定が行われた場合，コマンド・ライン上の指定が優先されます。
- コマンド・ライン上にオプションが指定された場合にも，KANJICODE 制御命令に対する文法チェックは行われます。

4.9 その他の制御命令

次に示す制御命令は，C コンパイラや構造化アセンブラ・プリプロセッサなどの上位プログラムが出力する特別な制御命令です。

- $TOL_INF
- $DGS
- $DGL

第 5 章 マクロ

この章では，マクロ機能の使い方について説明します。

プログラムの中で一連の命令群を何回も記述する場合に使用すると，便利な機能です。

5.1 概要

ソースの中で一連の命令群を何回も記述する場合，マクロ機能を使用すると便利です。

マクロ機能とは，MACRO，ENDM 疑似命令により，マクロ・ボディとして定義された一連の命令群をマクロ参照している箇所に展開することです。

マクロは，ソースの記述性を向上させるために使用するもので，サブルーチンとは異なります。

マクロとサブルーチンには，それぞれ次のような特徴があります。それぞれ目的に応じて有効に使用してください。

(1) サブルーチン

- プログラム中で何回も必要となる処理を 1 つのサブルーチンとして記述します。サブルーチンは，アセンブラにより一度だけ機械語に変換されます。
- サブルーチンの参照には，サブルーチン・コール命令（一般にはその前後に引数設定の命令が必要）を記述するだけで済みます。
  したがって，サブルーチンを活用することにより，プログラムのメモリを効率よく使用することができます。
- プログラム中の一連のまとまった処理をサブルーチン化することにより，プログラムの構造化を図ることができます（プログラムを構造化することにより，プログラム全体の構造が分かりやすくなり，プログラムの設計が容易になります）。

(2) マクロ

- マクロの基本的な機能は，命令群の置き換えです。
  MACRO，ENDM 疑似命令によりマクロ・ボディとして定義された一連の命令群が，マクロ参照時にその場所に展開されます。アセンブラは，マクロ参照を検出するとマクロ・ボディを展開し，マクロ・ボディの仮パラメータを参照時の実パラメータに置き換えながら，命令群を機械語に変換します。
- マクロは，パラメータを記述することができます。
  たとえば，処理手順は同じであるがオペランドに記述するデータだけが異なる命令群がある場合，そのデータに仮パラメータを割り当ててマクロを定義します。マクロ参照時には，マクロ名と実パラメータを記述することにより，記述の一部分だけが異なる種々の命令群に対処することができます。

サブルーチン化の手法が，メモリ・サイズの削減やプログラムの構造化を図るために用いられるのに対し，マクロは，コーディングの効率を向上させるために用いられます。

5.2 マクロの利用

5.2.1 マクロの定義

マクロの定義は，MACRO，ENDM 疑似命令により行います。

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
マクロ名	MACRO : ENDM	[*仮パラメータ* [, …]]	[; *コメント*] [; *コメント*]

【機能】

- MACRO 疑似命令と ENDM 疑似命令の間に記述された一連の文（マクロ・ボディと呼びます）に対し，シンボル欄で指定したマクロ名を取り付け，マクロの定義を行います。

【使用例】

```
ADMAC   MACRO   PARA1 , PARA2
        MOV     A , #PARA1
        ADD     A , #PARA2
        ENDM
```

上記の例は，PARA1 と PARA2 の 2 数を加算して，結果を A レジスタに格納する簡単なマクロ定義で，ADMAC というマクロ名が付けられています。PARA1，PARA2 が仮パラメータです。

詳細については，「3.8 マクロ疑似命令」を参照してください。

5.2.2 マクロの参照

マクロの参照を行う場合は，すでにマクロ定義されているマクロ名を，ソースのニモニック欄に記述します。

【記述形式】

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	*マクロ名*	[*実パラメータ* [, …]]	[; *コメント*]

【機能】

- 指定したマクロ名に割り付けられたマクロ・ボディを参照します。

【用途】

- マクロ・ボディを参照するときに，この形式の記述を使用します。

【説明】

- マクロ名は，参照以前に定義されていなければなりません。
- 実パラメータはコンマ（，）で区切って，1 行以内であれば最大 16 個まで記述することができます。
- 実パラメータの文字列中に，空白を記述することはできません。
- 実パラメータにコンマ（，），セミコロン（;），空白，TAB を記述する場合には，それらを含む文字列をシングルクォート（'）で囲ってください。
- 仮パラメータから実パラメータへの置き換えは，それぞれの記述順に対応して，左から順に行われます。仮パラメータと実パラメータの数が一致しない場合は，ワーニングが出力されます。

【使用例】

```
        NAME    SAMPLE
ADMAC   MACRO   PARA1 , PARA2
                MOV     A , #PARA1
                ADD     A , #PARA2
        ENDM

        CSEG
          :
        ADMAC   10H , 20H
          :
        END
```

すでに定義されているマクロ名“ADMAC”を参照しています。

10H，20H は実パラメータです。

5.2.3 マクロの展開

アセンブラは，マクロを次のように処理します。

- マクロの参照を見つけると，それに対応するマクロ・ボディをマクロ名の位置に展開します。
- 展開したマクロ・ボディのステートメントを，ほかのステートメントと同様にアセンブルします。

5.2.4 使用例

「5.2.2 マクロの参照」で参照されたマクロがアセンブルされ，展開されると，次のようになります。

```
        NAME    SAMPLE

        ; マクロ定義
ADMAC   MACRO   PARA1 , PARA2
                MOV     A , #PARA1
                ADD     A , #PARA2
        ENDM

        ; ソース本文
        CSEG
          :

        ; マクロの展開
        ADMAC   10H , 20H               ; (1)
        MOV     A , #10H
        ADD     A , #20H

        ; ソース本文
          :
        END
```

(1) マクロの参照により，マクロ・ボディが展開されます。
マクロ・ボディ内の仮パラメータは，実パラメータに置き換えられます。

5.3 マクロ内のシンボル

マクロ内で定義するシンボルには，グローバル・シンボルとローカル・シンボルの 2 種類があります。

(1) グローバル・シンボル

- ソース内のすべてのステートメントから参照することができます。
  したがって，そのシンボルを定義しているマクロを 2 回以上参照し，一連のステートメントが展開されると，シンボルは二重定義エラーとなります。
- LOCAL 疑似命令で定義されていないシンボルは，グローバル・シンボルです。

(2) ローカル・シンボル

- ローカル・シンボルは，LOCAL 疑似命令で定義します（「3.8 マクロ疑似命令」を参照してください）。
- ローカル・シンボルは，LOCAL 疑似命令で LOCAL 宣言されたマクロ内でのみ参照することができます。
- マクロ外からローカル・シンボルを参照することはできません。

【使用例】

```
        NAME    SAMPLE
          ; マクロの定義
MAC1    MACRO
        LOCAL   LLAB            ; (1)
LLAB :                          ; (2)
          :
GLAB :                          ; (3)
        BR      LLAB            ; (4)
        BR      GLAB            ; (5)
        ENDM
          :
          ; ソース本文
REF1 :  MAC1                    ; (6) ←マクロの参照
          :
        BR      LLAB            ; (7) ←エラー

REF2 :  MAC1                    ; (8) ←マクロの参照
          :
GLAB :                          ; (9) ←エラー
          :
        END
```

(1) ラベル LLAB をローカル・シンボルとして宣言しています。

(2) ラベル LLAB をローカル・シンボルとして定義しています。

(3) ラベル GLAB をグローバル・シンボルとして定義しています。

(4) マクロ MAC1 の定義内で，ローカル・シンボル LLAB を参照しています。

(5) マクロ MAC1 の定義内で，グローバル・シンボル GLAB を参照しています。

(6) マクロ MAC1 を参照しています。

(7) マクロ MAC1 の定義外で，ローカル・シンボル LLAB を参照しています。

この記述は，アセンブル時にエラーとなります。

(8) マクロ MAC1 を参照しています。

同一のマクロが 2 回参照されています。

(9) ラベル GLAB をグローバル・シンボルとして定義しています。

同一のラベルが 2 回定義されています。

この記述は，アセンブル時にエラーとなります。

使用例のアセンブル・リストを次に示します。

＜アセンブル・リスト＞

```
        NAME    SAMPLE
          :
REF1 :  MAC1
          ; マクロの展開
??RA0000 :
          :
GLAB :                          ←エラー
        BR      ??RA0000
        BR      GLAB
          ; ソース本文
          :
        BR      !LLAB           ←エラー
        BR      !GLAB
          :
REF2 :  MAC1
          ; マクロの展開
??RA0001 :
          :
GLAB :                          ←エラー
        BR      ??RA0001
        BR      GLAB
          ; ソース本文
          :
        END
```

グローバル・シンボル GLAB が，マクロ MAC1 内で定義されています。

マクロ MAC1 が 2 回参照されており，一連のステートメントが展開された結果，グローバル・シンボル GLAB は二重定義エラーとなります。

5.4 マクロ・オペレータ

マクロ・オペレータには，“&”と“'”の 2 種類があります。

(1) &（コンカティネート）

- コンカティネート記号は，マクロ・ボディ内で文字列と文字列を連結します。
  マクロ展開時には，コンカティネート記号の左右の文字列を連結し，コンカティネート自身は消滅します。
- コンカティネート記号は，マクロ定義時にシンボル中の“&”の前後を仮パラメータ，あるいは LOCAL シンボルとして認識することが可能であり，マクロ展開時にシンボル中の“&”の前後の仮パラメータ，あるいは LOCAL シンボルを評価してシンボル中に連結することができます。
- 引用符で囲まれた文字列中の“&”は，単なるデータとして扱われます。
- “&”を 2 つ続けると，1 つの“&”として扱われます。

【使用例】

＜マクロ定義＞

```
MAC     MACRO   P
LAB&P :                 ←仮パラメータの P が認識される
        D&B     10H
        DB      'P'
        DB      P
        DB      '&P'
        ENDM
```

＜マクロ参照＞

```
        MAC     1H
LAB1H :
        DB      10H     ← D と B が連結されて DB となる
        DB      'P'
        DB      1H
        DB      '&P'    ←引用符中の“&”は単なるデータとして扱われる
```

(2) '（シングル・クォート）

- マクロ参照行，および IRP の実パラメータの先頭，あるいは，区切り文字のあとに，文字列をシングル・クォートで囲んで記述すると，その文字列がそのまま 1 つの実パラメータとみなされます。
  実パラメータに渡されるときには，シングル・クォートが外されて渡されます。
- マクロ・ボディ中にシングル・クォートで囲まれた文字列がある場合には，単なるデータとして扱われます。
- シングル・クォートで囲まれた中にシングル・クォートを使用する場合には，“'”を 2 つ続けて記述します。

【使用例】

```
        NAME    SAMP
MAC1    MACRO   P
        IRP     Q , <P>
                MOV     A , #Q
        ENDM
ENDM

        MAC1    '10 , 20 , 30'
```

このソースをアセンブルすると，MAC1 は次のように展開されます。

＜アセンブル・リスト＞

```
IRP     Q , <10 , 20 , 30>
        MOV A , #Q
ENDM
        MOV     A , #10         ; IRP の展開
        MOV     A , #20         ; IRP の展開
        MOV     A , #30         ; IRP の展開
```

第 6 章 製品活用法

この章では，RA78K0R を利用してアセンブル作業を行ううえで有効な活用法をいくつか紹介します。

6.1 アセンブラ起動時の手間を省くには

アセンブラ起動時に，コマンド行で指定するデバイス種別指定（-c）や漢字コード指定（-zs/-ze/-zn）については，制御命令としてソース・モジュール中に記述しておくことができます。これにより，コマンド行での指定を省くことができます。

このほかに，クロスリファレンス・リスト出力指定なども，ソース・モジュールの先頭で指定しておくとよいでしょう。

＜例＞

```
$       PROCESSOR ( f1166a0 )
$       KANJICODE       SJIS
$       XRFF

        NAME    TEST

C1      CSEG
          :
        END
```

6.2 メモリ効率のよいプログラムを開発するには

ショート・ダイレクト・アドレッシング領域は，ほかのデータ・メモリに比べ，短いバイト数の命令でアクセスすることのできる領域です。この領域を効率的に使うことで，メモリ効率のよいプログラムを開発することができます。

ショート・ダイレクト・アドレッシング領域は，1 モジュールで宣言します。

これにより，ショート・ダイレクト・アドレッシング領域に配置しようとした変数が，ショート・ダイレクト・アドレッシング領域に入りきらなかった場合，アクセス頻度の高い変数だけをショート・ダイレクト・アドレッシング領域に配置する変更が容易になります。

【使用例】

＜モジュール 1 ＞

```
        PUBLIC  TMP1 , TMP2
WORK    DSEG    AT      0FFE20H
TMP1:   DS      2               ; word
TMP2:   DS      1               ; byte
```

＜モジュール 2 ＞

```
        EXTRN   TMP1 , TMP2
SAB     CSEG
        MOVW    TMP1 , #1234H
        MOV     TMP2 , #56H
          :
```

付録 A 予約語一覧

予約語には，機械語命令，疑似命令，制御命令，演算子，レジスタ名，および sfr シンボルがあります。

予約語は，アセンブラがあらかじめ予約している文字列で，所定の目的以外には転用することができません。

ソースの各欄に記述可能な予約語の種類と予約語一覧を，次に示します。

表 A-1 予約語の種類

種類	説明
シンボル欄	すべての予約語が記述不可
ニモニック欄	機械語命令，および疑似命令のみ記述可能
オペランド欄	演算子，sfr シンボル，およびレジスタ名のみ記述可能
コメント欄	すべての予約語が記述可能

表 A-2 予約語一覧

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

備考 制御命令の [ ] 内は，短縮形を表します。

なお，sfr 一覧については，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。

割り込み要求ソース一覧，機械語命令，レジスタ名一覧については，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。

付録 B 疑似命令一覧

表 B-1 疑似命令一覧

疑似命令				機能分類	備考
シンボル欄	ニモニック欄	オペランド欄	コメント欄		
[*セグメント名*]	CSEG	[*再配置属性*]	[; *コメント*]	コード・セグメントの開始宣言	
[*セグメント名*]	DSEG	[*再配置属性*]	[; *コメント*]	データ・セグメントの開始宣言	
[*セグメント名*]	BSEG	[*再配置属性*]	[; *コメント*]	ビット・セグメントの開始宣言	
[*セグメント名*]	ORG	*絶対式*	[; *コメント*]	アブソリュート・セグメントの開始宣言	オペランド式中でのシンボルの前方参照禁止
ネーム	EQU	*式*	[; *コメント*]	ネームの定義	ネーム：シンボル オペランド式中でのシンボルの前方参照，外部参照名の参照禁止
ネーム	SET	*絶対式*	[; *コメント*]	再定義可能なネームの定義	ネーム：シンボル オペランド式中でのシンボルの前方参照禁止
[*ラベル* :]	DB	(*サイズ*) または *初期値* [, …]	[; *コメント*]	バイト・データ領域の確保／初期化	ラベル：シンボル 初期値の代わりに文字列を置くことができる
[*ラベル* :]	DW	(*サイズ*) または *初期値* [, …]	[; *コメント*]	ワード・データ領域の確保／初期化	ラベル：シンボル
[*ラベル* :]	DG	(*サイズ*) または *初期値* [, …]	[; *コメント*]	4 バイト・データ領域の確保／初期化	ラベル：シンボル
[*ラベル* :]	DS	*絶対式*	[; *コメント*]	バイト・データ領域の確保	ネーム：シンボル オペランド式中でのシンボルの前方参照禁止
ネーム	DBIT	*なし*	[; *コメント*]	ビット・データ領域の確保	ネーム：シンボル オペランド式中でのシンボルの前方参照禁止

疑似命令				機能分類	備考
シンボル欄	ニモニック欄	オペランド欄	コメント欄		
[*ラベル* :]	EXTRN	*シンボル名* [, …] または BASE(*シンボル名* [, …])	[; *コメント*]	外部参照名の宣言	
[*ラベル* :]	EXTBIT	*ビット・シンボル名* [, …]	[; *コメント*]	外部参照名の宣言	シンボル名はビット値を持つものに限る
[ラベル :]	PUBLIC	*シンボル名* [, …]	[; *コメント*]	外部定義名の宣言	
[*ラベル* :]	NAME	*オブジェクト・モジュール名*	[; *コメント*]	モジュール名の定義	モジュール名：シンボル
[*ラベル* :]	BR	*式*	[; *コメント*]	分岐命令の自動選択	ラベル：シンボル
[*ラベル* :]	CALL	*式*	[; *コメント*]	分岐命令の自動選択	ラベル：シンボル
ネーム	MACRO	[*仮パラメータ* [, …]]	[; *コメント*]	マクロの定義	マクロ名：シンボル
[*ラベル* :]	LOCAL	*シンボル名* [, …]	[; *コメント*]	マクロ内でのみ有効なシンボルの定義	マクロ定義中のみ使用可
[*ラベル* :]	REPT	*絶対式*	[; *コメント*]	マクロ展開時の繰り返しの指定	ラベル：シンボル
[*ラベル* :]	IRP	*仮パラメータ* ,< [*実パラメータ* [, …]]>	[; *コメント*]	仮パラメータに実パラメータの値を割り付けて展開	ラベル：シンボル
[*ラベル* :]	EXITM	*なし*	[; *コメント*]	マクロ展開の中断	マクロ定義中のみ使用可
なし	ENDM	*なし*	[; *コメント*]	マクロ定義の終了	マクロ定義中のみ使用可
なし	END	*なし*	[; *コメント*]	ソース・モジュールの終了	

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。