お客様各位

カタログ等資料中の旧社名の扱いについて

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

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

2010年4月1日
ルネサスエレクトロニクス株式会社

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

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

ご注意書き

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

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

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

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

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

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

ユーザーズ・マニュアル

RA78K4 Ver.1.50以上

アセンブラ・パッケージ

言語編

対象デバイス
78K/IVシリーズ

資料番号 U15255JJ1V0UM00（第1版）
発行年月 August 2001 NS CP(K)

〔メ　モ〕

目次要約

Windows**および**WindowsNT**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。**

HP-UX**は，米国**Hewlett-Packard Corp. **の商標です。**

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

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

M7A 98.8

巻末にアンケート・コーナを設けております。このドキュメントに対するご意見をお気軽にお寄せください。

はじめに

このマニュアルは，RA78K4アセンブラ・パッケージ（以降，RA78K4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，RA78K4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RA78K4**アセンブラ・パッケージ　ユーザーズ・マニュアル　操作編**（U15254J）（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RA78K4に関する記述は，Ver. 1.50以上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ピュータ（78K/**Ⅳ**シリーズ）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

RA78K4全体の基本的な機能概要を説明します。

第2章　ソース・プログラムの記述方法

ソース・プログラムの記述方法，アセンブラ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，最大性能一覧表，索引を掲載しています。

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ピュータのユーザーズ・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ピュータのユーザーズ・マニュアル（ハードウエア編）をご覧ください。

【読 み 方】

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・プログラムの記述方法**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。

⋮	；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。

資　料　名		資料番号	
		和　　文	英　　文
RA78K4　アセンブラ・パッケージ	操作編	U15254J	U15254E
	言語編	このマニュアル	U15255E
	構造化アセンブラ・プリプロセッサ	U11743J	U11743E
CC78K4　Cコンパイラ	操作編	U15557J	作成予定
	言語編	U15556J	作成予定
SM78K4　システム・シミュレータ　Ver.1.40以上 Windows^{TM}ベース	レファレンス編	U10093J	U10093E
SM78Kシリーズ　システム・シミュレータ　Ver.1.40以上	外部部品ユーザ・オープン・インタフェース仕様編	U10092J	U10092E
ID78Kシリーズ　統合ディバッガ　Ver.2.30以上 Windowsベース	操作編	U15185J	U15185E
ID78K4　統合ディバッガ　Windowsベース	レファレンス編	U10440J	U10440E
RX78K4　リアルタイムOS	基礎編	U10603J	U10603E
	インストール編	U10604J	U10604E
	ディバッガ編	U10364J	－

目　　次

図の目次

表の目次

第1章　概　　説

この章では，マイクロコンピュータの開発におけるRA78K4の役割など，RA78K4の特徴について説明します。

1.1　アセンブラの概要

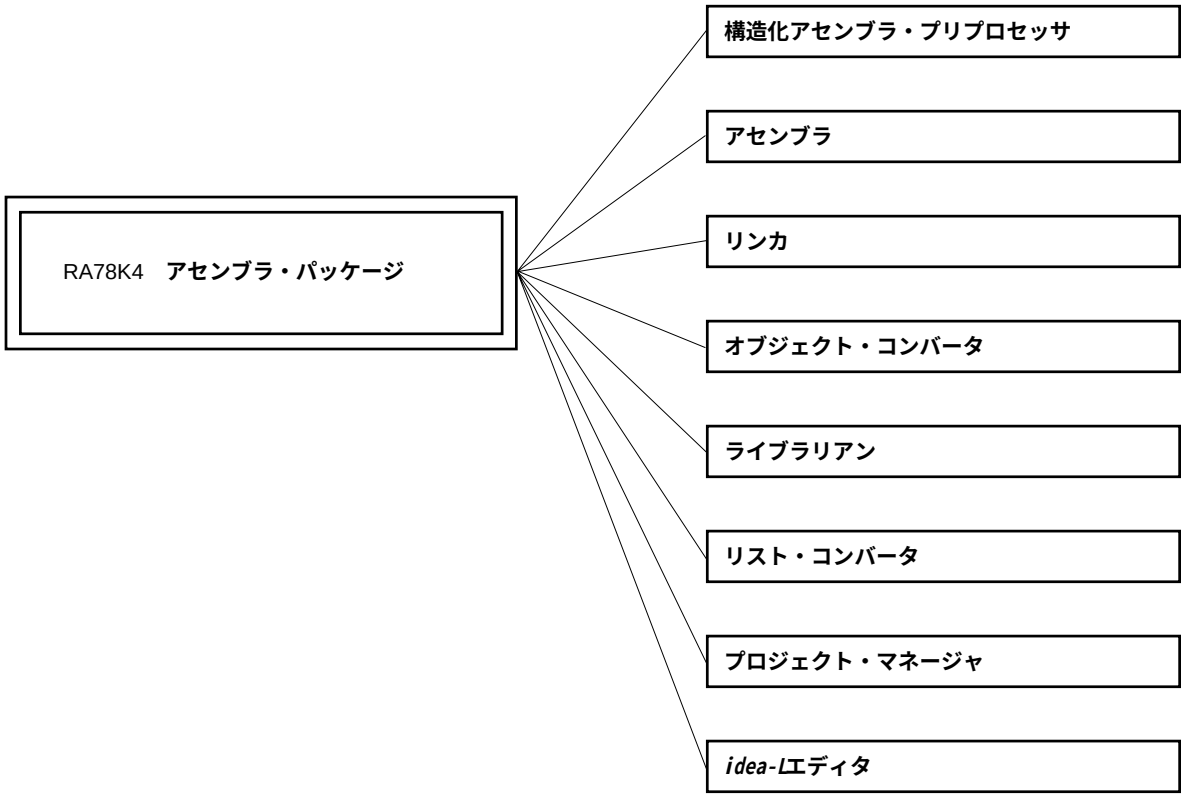
RA78K4アセンブラ・パッケージは，78K/IVシリーズのアセンブリ言語で記述されたソース・プログラムを機械語に変換する一連のプログラムの総称です。

RA78K4の中には，構造化アセンブラ・プリプロセッサ，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・コンバータの6つのプログラムがあります。

さらに，エディット，コンパイル／アセンブル，リンクからディバ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プロジェクト・マネージャもRA78K4に添付されています。

また，プロジェクト・マネージャにはエディタ（*idea-L*エディタ）が添付されています。

図1－1　RA78K4アセンブラ・パッケージ



1.1.1 アセンブラとは

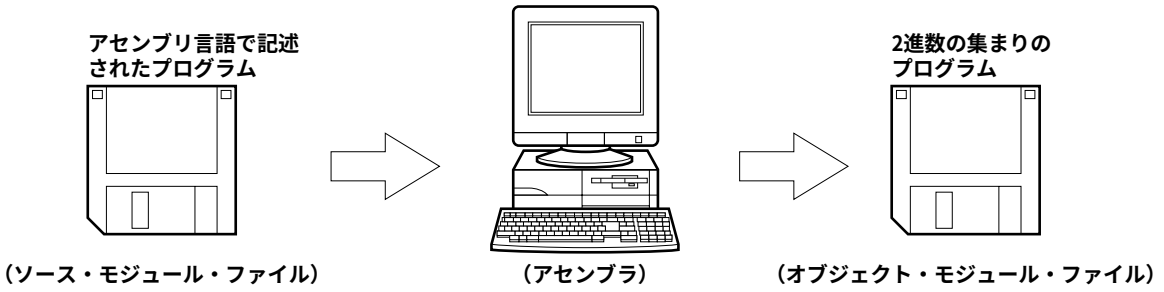
(1) アセンブリ言語と機械語

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させるためには，プログラムやデータが必要です。これを人間がプログラミングして，マイクロコンピュータのメモリ部に記憶させます。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，これを機械語といいます。機械語でプログラムを作るのは，人間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人間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言語体系をアセンブリ言語といいます。

マイクロコンピュータが扱えるのは機械語ですから，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。

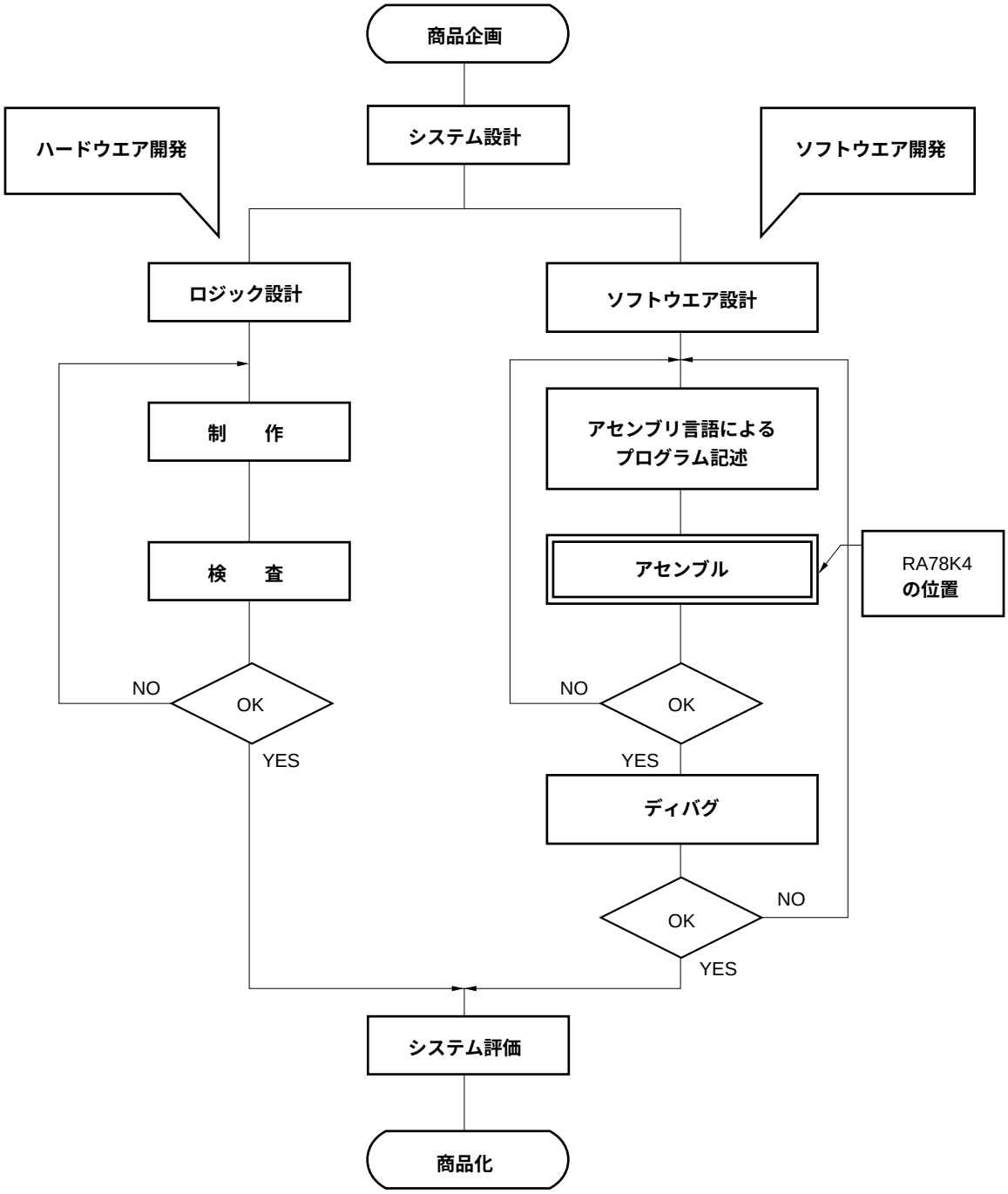
図1－2　アセンブラの流れ



(2) マイクロコンピュータ応用製品の開発とRA78K4の役割

アセンブリ言語でのプログラミングは，製品開発のどこに位置するのかを図1－3　マイクロコンピュータ応用製品の開発工程に示します。

図1－3　マイクロコンピュータ応用製品の開発工程



1.1.2 リロケータブル・アセンブラとは

アセンブラが変換した機械語は，マイクロコンピュータ内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，「各機械語がメモリ中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」とに大別できます。

・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を，絶対的（アブソリュート）なアドレスに配置します。

・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，一時的なアドレスを与えます。絶対的なアドレスは，リンカと呼ばれるプログラムにより行います。

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一度に作成すると，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大変になります。そこで，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・プログラミングといいます。

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す。

リロケータブル・アセンブラを使用してモジュラ・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。

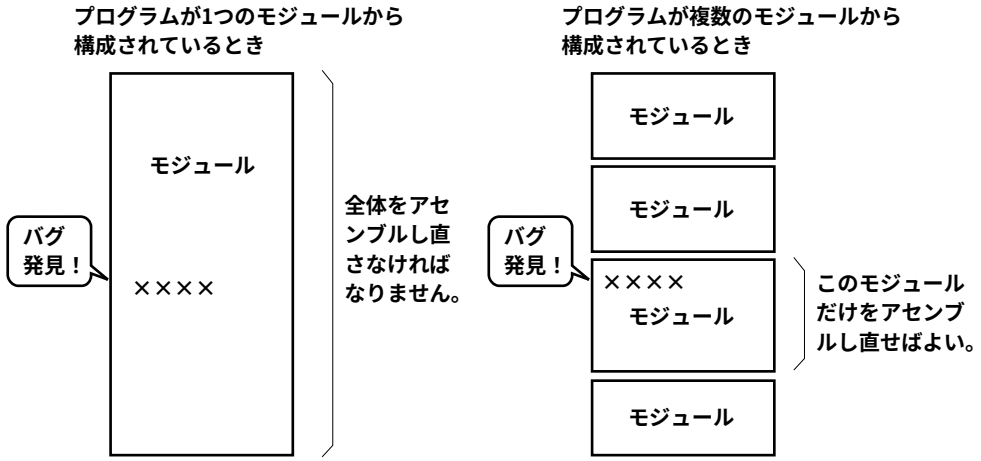
(1) 開発効率が上がる

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，プログラムを機能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，一部の修正の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せます。

これにより，ディバグ時間を短くできます。

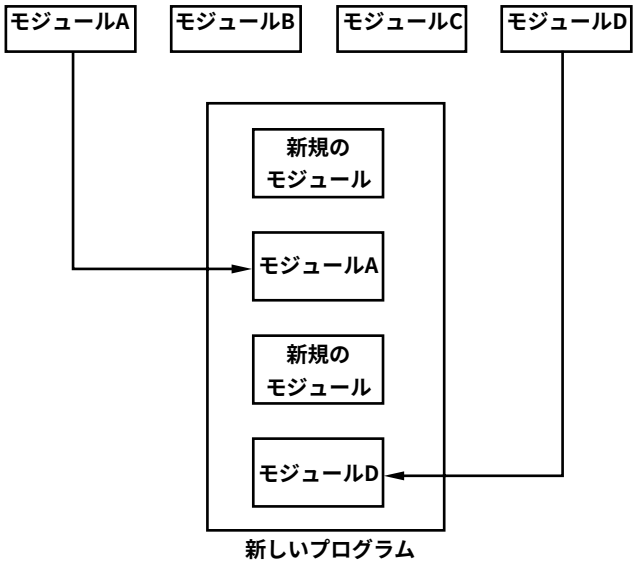
図1－4　アセンブルのやり直し



(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，他のプログラムの開発に利用できます。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できます。

図1－5　既成モジュールを利用したプログラム作成



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，次のことを頭に入れておいてください。

1.2.1 RA78K4の最大性能

(1) アセンブラの最大性能

表1－1　アセンブラの最大性能

項　目	最大性能	
	PC版	WS版
シンボル数（ローカル＋パブリック）	65535個[注1]	65535個[注1]
クロスレファレンス・リスト出力可能なシンボル数	65534個[注2]	65534個[注2]
1マクロ参照のマクロ・ボディ最大サイズ	1 Mバイト	1 Mバイト
全マクロ・ボディ合計のサイズ	10 Mバイト	10 Mバイト
1ファイル中のセグメント数	256個	256個
1ファイル中のマクロ，インクルード指定	10000個	10000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	10000個	10000個
リロケーション情報[注3]	65535個	65535個
行番号情報	65535個	65535個
1ファイル中のBR疑似命令数	32767個	32767個
1行の文字数	2048文字[注4]	2048文字[注4]
シンボル長	256文字	256文字
スイッチ名の定義数[注5]	1000個	1000個
スイッチ名の文字長[注5]	31文字	31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	8レベル

注1. XMSを使用します。XMSがなければファイルを使用します。

2. 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。

3. リロケーション情報とは，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。

4. 復帰／改行コードを含みます。1行に2049文字以上記述された場合，ワーニング・メッセージを出力し，2049文字以降は無視します。

5. 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。

(2) リンカの最大性能

表1－2　リンカの最大性能

項　目	最大性能	
	PC版	WS版
シンボル数（ローカル＋パブリック）	65535個	65535個
同一セグメントの行番号情報	65535個	65535個
セグメント数	65535個	65535個
入力モジュール数	1024個	1024個

1. 3　RA78K4の特徴

RA78K4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・プログラム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（BR疑似命令）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いいます。

(3) 条件付きアセンブル機能

ソース・プログラムの一部分を条件によりアセンブルするかしないかを設定できます。

ソース・プログラム中にディバグ文などを記述した場合，ディバグ文を機械語に変換するか，しないかを条件付きアセンブルのスイッチ設定により選択できます。ディバグ文が必要なくなった場合でも，ソース・プログラムに大幅な変更を加えることなくアセンブルできます。

(4) 汎用レジスタ名選択機能

汎用レジスタ表現として絶対名称（R0, R1, RP0…など）と機能名称（X, A, AX, …など）があります。なお，ソース・プログラム中で機能名称を記述するときは，必ず汎用レジスタ選択疑似命令（RSS疑似命令）を記述します。

RSS疑似命令はソース・プログラム中の汎用レジスタ表現として機能名称を記述することを可能にするものです。

第2章　ソース・プログラムの記述方法

この章では，ソース・プログラムの記述方法，記述様式，式と演算子などについて説明します。

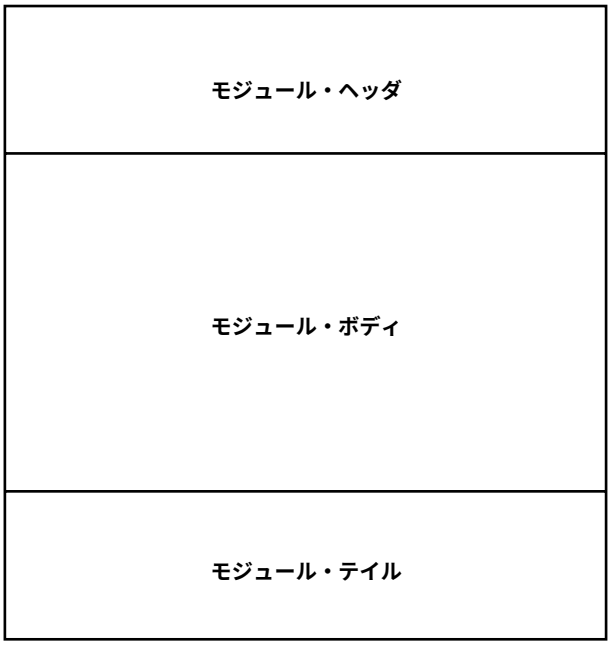
2. 1　ソース・プログラムの基本構成

1つのソース・プログラム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・プログラムとソース・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。

① モジュール・ヘッダ（module header）
② モジュール・ボディ（module body）
③ 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

モジュール・ヘッダには，表2－1　モジュール・ヘッダに記述できるものに示す制御命令が記述できます。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	アセンブラ・オプションと同様の機能を持つ制御命令は次のとおりです。 ・PROCESSOR ・XREF/NOXREF ・DEBUG/NODEBUG/DEBUGA/NODEBUGA ・TITLE ・SYMLIST/NOSYMLIST ・FORMFEED/NOFORMFEED ・WIDTH ・LENGTH ・TAB ・CHGSFR/CHGSFRA ・KANJICODE	第4章　制御命令
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	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は次のとおりです。 ・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がモジュール・ボディに記述できます。
さらに，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。
セグメントには，次の4つのセグメントがあり，それぞれ対応する疑似命令で定義します。

① コード・セグメント …… CSEG疑似命令で定義します。
② データ・セグメント …… DSEG疑似命令で定義します。
③ ビット・セグメント …… BSEG疑似命令で定義します。
④ 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

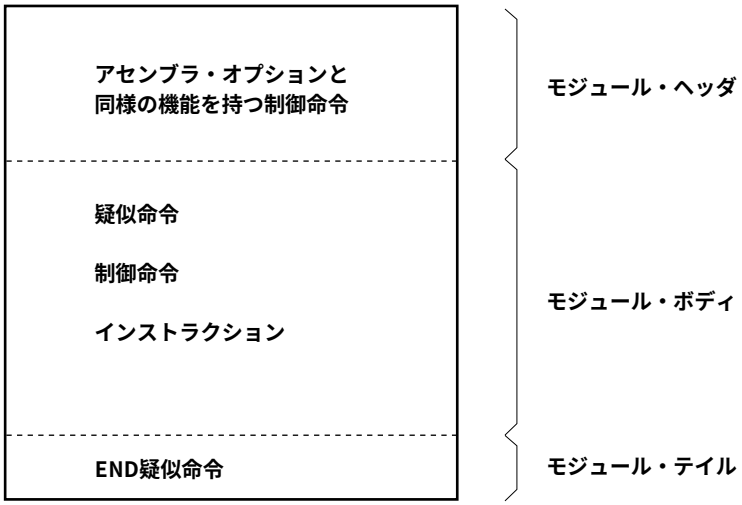
モジュール・テイルは，ソース・モジュールの終了を示すものです。END疑似命令を記述します。

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・プログラムの全体構成

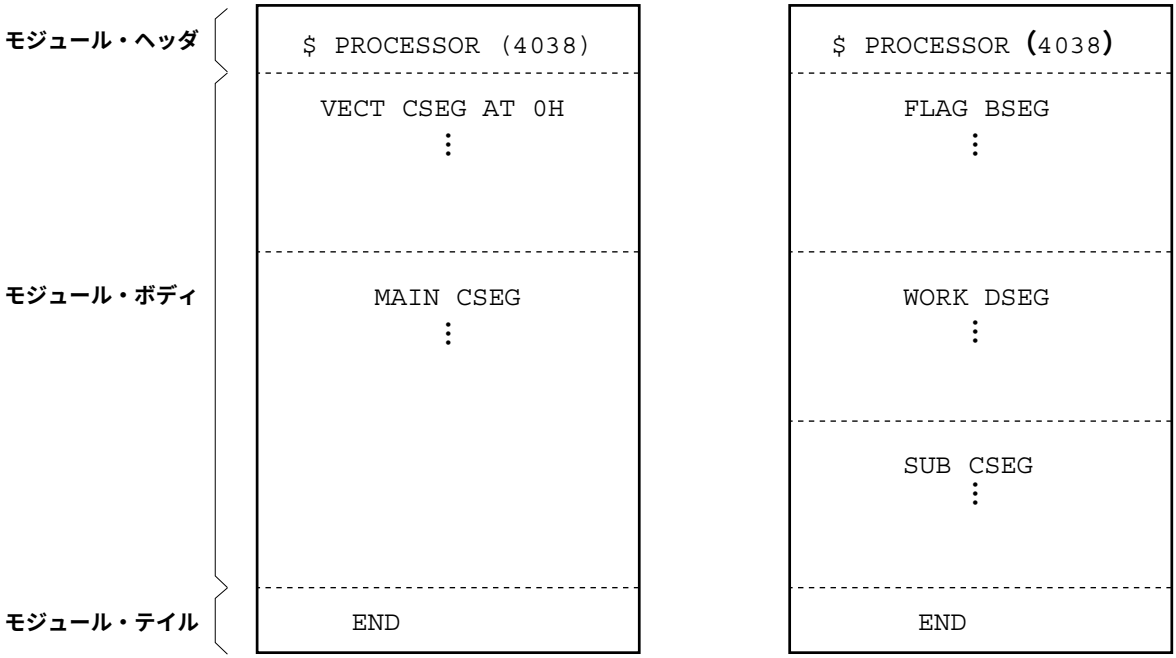
ソース・モジュール（ソース・プログラム）の全体構成は，次のようになります。

図2－2　ソース・モジュールの全体構成



また，ソース・モジュールの構成例を簡単に示すと，次のようになります。

図2－3　ソース・モジュールの構成例

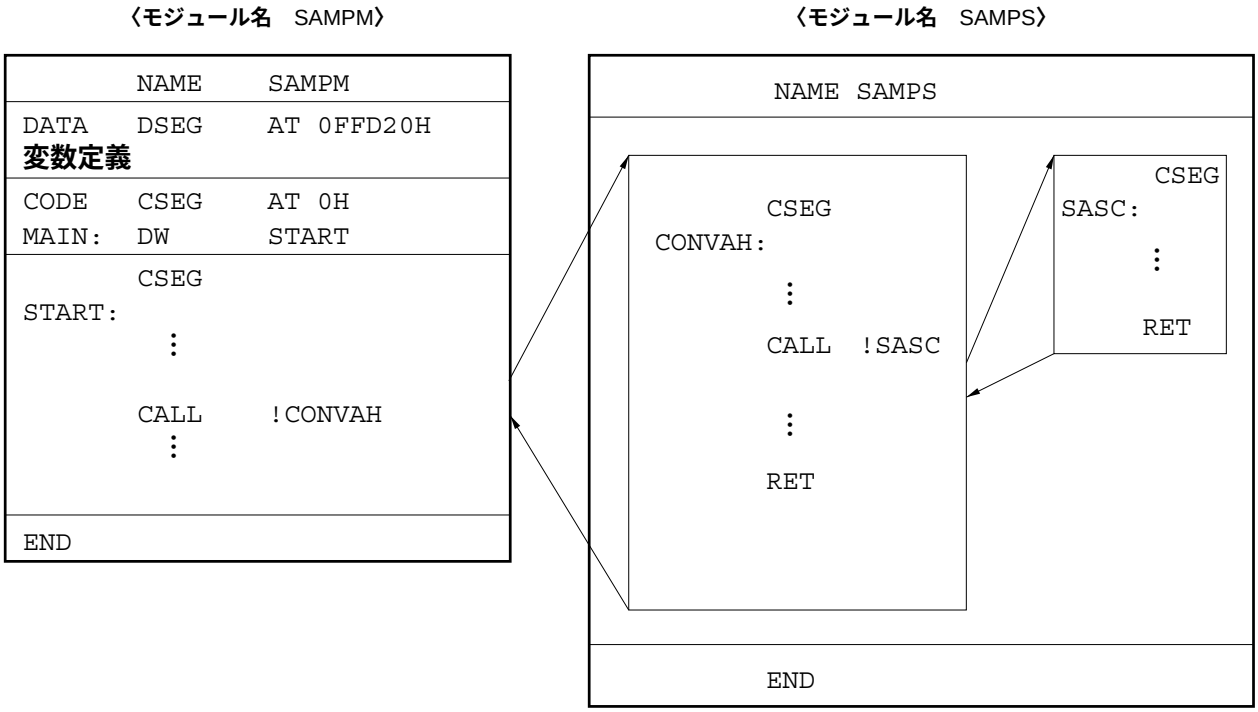


2.1.5　ソース・プログラム記述例

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

サンプル・プログラムの構成を簡単に示すと，次のようになります。

図2－4　サンプル・プログラムの構成



このサンプル・プログラムは，1つのソース・プログラムを2つのモジュールに分けたものです。モジュール「SAMPM」は，このプログラムのメイン・ルーチン，モジュール「SAMPS」は，メイン・ルーチン内で呼ばれるサブルーチンです。

＜メイン・ルーチン＞

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

        PUBLIC  MAIN, START                                                 ;(2)
        EXTRN   CONVAH                                                      ;(3)

DATA    DSEG    AT 0FFD20H                                                  ;(4)
HDTSA:  DS      1
STASC:  DS      2

CODE    CSEG    AT 0H                                                       ;(5)
MAIN:   DW      START

        CSEG                                                                ;(6)
        LOCATION 15
START:  MOV     RFM,#00
        MOVG    SP,#0FFE00H
        MOV     MM,#00
        MOV     STBC,#08H

        MOV     HDTSA,#1AH
        MOVG    WHL,#HDTSA              ;set hex 2-code data in WHL registor

        CALL    CONVAH                  ;convert ASCII <- HEX
                                        ;output BC-register <- ASCII code
        MOVG    TDE,#STASC              ;set DE <- store ASCII code table
        MOV     A,B
        MOV     [TDE+],A
        MOV     A,C
        MOV     [TDE+],A

        BR      $$

        END                                                                 ;(7)
```

(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定義

(4) データ・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0FFD20H番地から配置する）

(5) 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）

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

(7) モジュールの終了を宣言

＜サブルーチン＞

```
        NAME        SAMPS                                                   ;(8)
;************************************************
;*                                              *
;*   HEX -> ASCII Conversion Program            *
;*                                              *
;*               sub-routine                    *
;*                                              *
;*   input condition  : (HL) <- hex 2 code       *
;*                                              *
;*   output condition ; BC-register <-ASCII 2 code *
;*                                              *
;************************************************

        PUBLIC   CONVAH                                                     ;(9)

        CSEG                                                                ;(10)
CONVAH: MOV      A,#0
        ROL4     [WHL]              ;hex upper code load
        CALL     $!SASC
        MOV      B,A                ;store result

        MOV      A,#0
        ROL4     [WHL]              ;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(＋7)
SASC1:  ADD      A,#30H             ;bias(＋30)
        RET

        END                                                                 ;(11)
```

(8) モジュール名を宣言

(9) 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

(10) コード・セグメントの開始を宣言

(11) モジュールの終了を宣言

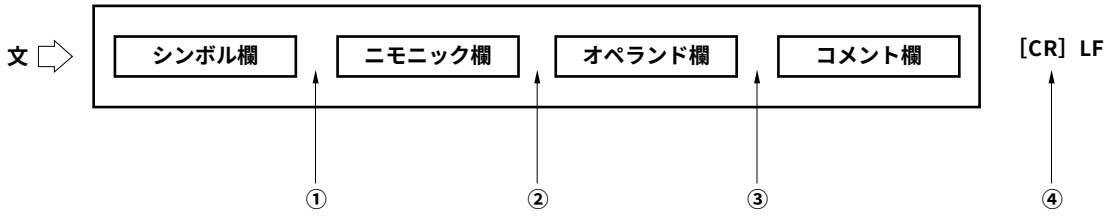
2. 2　ソース・プログラムの記述様式

2. 2. 1　文の構成

ソース・プログラムは，文（ステートメント）で構成します。

1つの文は，図2－5　**文の構成フィールド**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（コロンか空白かは，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）まで記述できます。

このとき，TABおよび単独のCRは，各々1文字として数えます。2049文字以上記述された場合には，ワーニング・メッセージを出力し，2049文字以降を無視します。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。

また，次のような行の記述ができます。

・空行（文の記述のない行）
・シンボル欄のみの行
・コメント欄のみの行

2.2.2　文字セット

ソース・ファイル中に記述できる文字は，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・プログラム上で命令の記述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。

【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

【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		区	各欄の区切り記号
HT（09H）	タブコード	切	空白相当文字
,	コンマ	り	オペランドの区切り文字
：	コロン	記	レーベル区切り記号
;	セミコロン	号	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	ア	加算演算子または正符号
－	マイナス	セ	減算演算子または負符号
＊	アスタリスク	ン	乗算演算子
／	スラッシュ	ブ	・除算演算子
		ラ	・／を持つオペランドを0→1, 1→0に反転し，操作することを意味する記号
.	ピリオド	演	ビット位置指定子
(，)	左・右かっこ	算	演算順序
＜，＞	不等号	子	比較演算子
=	等号		比較演算子
’	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 ・相対アドレシング指定記号	
$！	ドル・感嘆符	相対（16ビットの式）のアドレシング指定記号	
&	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・アドレシング指定記号	
！	感嘆符	絶対アドレシング指定記号	
！！	感嘆符（2つ）	16ビット空間の範囲での絶対アドレシング開始記号	
[%]	大かっこ・パーセント	インダイレクトかつ3バイト操作であることを表す記号	
[ ]	大かっこ	インダイレクト・アドレシング指定記号	

(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）の記述に用いる文字です。

【文字データの文字セット】

- 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 ’ で閉じるまで無視されます。
- 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）。
- Windowsの場合は，1AHはファイルの終わりと見なされるので，入力データとなり得ません。

(3) 注釈（コメント）用文字

コメントの記述に用いる文字です。

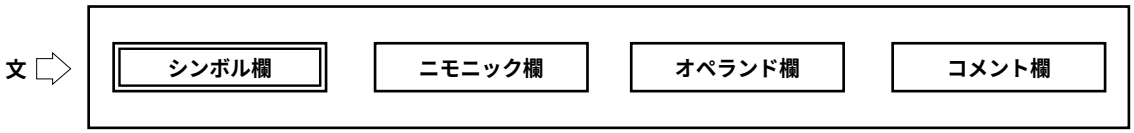
【コメントの文字セット】

・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“ ！ ”に置き換えられて出力されます。

2. 2. 3　文の構成フィールド

文を構成するフィールドについて説明します。

(1) 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・プログラムの内容がわかりやすくなります。

【シンボルの種類】

シンボルは，その使用目的，定義方法によって表2－2　シンボルの種類に示す種類に分けられます。

表2－2　シンボルの種類

シンボルの種類	使用目的	定義方法
ネーム	ソース・プログラム中で，数値データやアドレスとして使用します。	EQU，SET，DBIT疑似命令等のシンボル欄に記述します。
レーベル	ソース・プログラム中で，アドレス・データとして使用します。	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，他のモジュールで参照するときに使用します。	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	リンク時に使用するシンボルです。	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・ディバグ時に使用します。	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・プログラム中でマクロ参照時に使用します。	MACRO疑似命令のシンボル欄に記述します。

【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。

① 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，先頭文字として数字（0-9）は使用できません。

② シンボルの長さは，1-31文字です。
認識最大文字数を越えた文字は無視されます。

③ シンボルとして，予約語は使えません。
予約語を，付録A　予約語一覧に示します。

④ 同一シンボルを二度以上定義できません（ただし，SET疑似命令で定義したネームは，SET疑似命令で再定義できます）。

⑤ アセンブラは，シンボルの大文字／小文字を区別します。

⑥ シンボル欄にレーベルを記述する場合は，レー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’はレーベル
            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’レーベルです。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～Y1234567890123456     EQU     70H
```

250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～Y123456789012345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がレーベルとして定義されます。
```

【シンボルに関する注意事項】

??RAnnnn（n = 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なので，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に，アセンブラがセグメント名を自動生成します。表2－3　アセンブラ自動生成セグメント名に，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。

表2－3　アセンブラ自動生成セグメント名

セグメント名	疑似命令	再配置属性
?An	ORG疑似命令	n = 000000-FFFFFF
?CSEG	CSEG疑似命令	UNIT
?CSEGT0		CALLT0
?CSEGFX		FIXED
?CSEGFXA		FIXEDA
?CSEGB		BASE
?CSEGP		PAGE
?CSEGP64		PAGE64K
?DSEG	DSEG疑似命令	UNIT
?DSEGS		SADDR
?DSEGSP		SADDRP
?DSEGS2		SADDR2
?DSEGSP2		SADDRP2
?DSEGSA		SADDRA
?DSEGDT		DTABLE
?DSEGDTP		DTABLEP
?DSEGP		PAGE
?DSEGP64		PAGE64K
?DSEGG		GRAM
?BSEG	BSEG疑似命令	UNIT
?BSEGUP		UNITP
?BSEGS		SADDR
?BSEGSP		SADDRP
?BSEGS2		SADDR2
?BSEGSP2		SADDRP2
?BSEGSA		SADDRA
?BSEGG		GRAM

【シンボルの属性】

ネームおよびレーベルは，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，表2－4　シンボル属性と値に示す8種類のシンボル属性のことです。

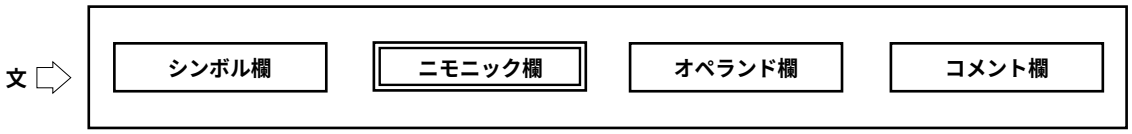
表2－4　シンボル属性と値

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：0-65535 16進表現：0H-FFFFH
ADDRESS	・レーベルとして定義されたシンボル ・レーベルをEQU，SET疑似命令で定義したネーム	
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	saddr領域
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       ；レーベル‘ START ’は，ADDRESS属性と，値80Hを持ちます。
BIT1      EQU     0FE20H.0     ；ネーム‘ BIT1 ’は，BIT属性と値0FE20H.0を持ちます。
```

(2) 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。

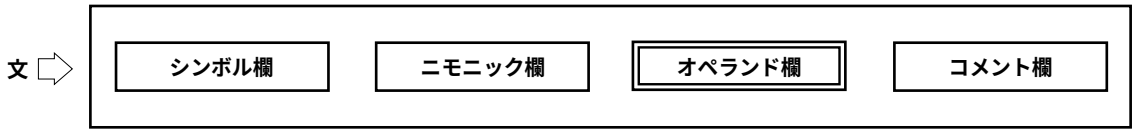
(正しい例)

```
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                     ；78K/IVシリーズの命令には，‘ZZZ’はありません。
```

(3) 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
- 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名
- シンボル
- 式
- ビット項
- マクロ・サービス・コントロール・ワード

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2.5　**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，開発対象となるマイクロコンピュータのユーザーズ・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。

【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・データともいいます。

定数には数値定数と文字列定数があります。

- 数値定数

数値定数として2進数，8進数，10進数，16進数が記述できます。

各数値定数の表現方法を**表**2－5　**数値定数の表記方法**に示します。

数値定数は，符号なしの24ビット・データとして処理されます。

値の範囲　0≦n≦16777215　（0FFFFFFH）

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。

表2－5　数値定数の表記方法

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’を付加します。	1101B 1101Y
8進数	数値の最後に文字‘O’または‘Q’を付加します。	74O 74Q
10進数	数値をそのまま記述します。 または，最後に文字‘D’または‘T’を付加します。	128 128D 128T
16進数	・数値の最後に文字‘H’を付加します。 ・先頭文字が‘A’，‘B’，‘C’，‘D’，‘E’，‘F’で始まる場合には，その前に‘0’を付加します。	8CH 0A6H

・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0-3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。

文字定数の表記例

```
'A'         ;41H
' '         ;20H
''''        ;27H
'''A'       ;2741H
''AA'       ;274141H
```

【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。
```

【レジスタ名】

オペランド欄に記述できる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**
- **3バイト・レジスタ**
- **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-R15, RP0-RP7**）での記述のほかに，機能名称（**X, A, B, C, D, E, H, L, AX, BC, DE, HL, VP, UP**）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，開発対象となるマイクロコンピュータのユーザーズ・マニュアルを参照してください。

【特殊文字】

オペランド欄に記述できる特殊文字は，表2－6　**オペランド欄に記述できる特殊文字に示すものです。**

表2－6　**オペランド欄に記述できる特殊文字**

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，コール命令のレラティブ（相対8ビット）アドレシングを示します。
$！	・分岐命令，コール命令のレラティブ（相対16ビット）アドレシングを示します。
！	・分岐命令，コール命令の16ビット絶対アドレシングを示します。 ・MOV命令の全メモリ空間指定可能なaddr16指定を示します。
！！	・分岐命令，コール命令の24/20絶対アドレシングを示します。 ・MOV命令の全メモリ空間指定可能なaddr24/addr20指定を示します。
#	・イミーデイエト・データを示します。
[ ]	・インダイレクト・アドレシングを示します。
[%]	・インダイレクト・アドレシングかつ3バイト命令を示します。

- **特殊文字の使用例**

```
アドレス        ソース・プログラム
100                 ADD  R15, R1
101             LOOP: INC  R1
103                 BR   $$-2     …①
106                 BR   !$+100H …②
```

① **オペランドの2番目の＄は**103H**番地を示します。“**BR　$LOOP**”と記述しても同様の動作をします。**

② **オペランドの2番目の＄は**106H**番地を示します。**

【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。**

【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     12345678
       MOVD    RP0, VALUE   ；MOV　D RP0, 12345678Hと同じ意味を持ちます。
```

【式】

式は，定数，ロケーション・アドレスを示す$，シンボ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**」と書き換えられます。**

【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. 4　**ビット位置指定子を参照してください。**

・ビット項の例

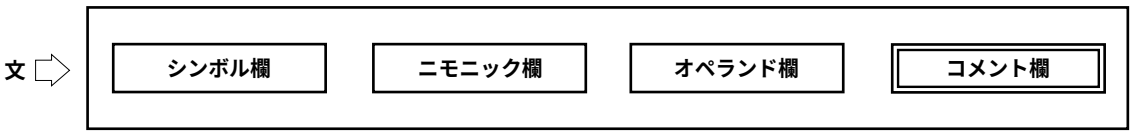
```
CLR1   A.5
SET1   1+0FE30H.3   ；オペランドの値は0FE31H.3です。
CLR1   0FE40H.4+2   ；オペランドの値は0FE40H.6です。
```

【マクロ・サービス・コントロール・ワード】

オペランドに記述できるマクロ・サービス・コントロール・ワードは，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。

注意　マクロ・サービス・コントロール・ワードはアブソリュートな値として処理するため，SFR**領域変更オプション（**-CSA**）または**SFR**領域変更制御命令（**$CHGSFRA**）を指定したときに，マクロ・サービス・コントロール・ワードをオペランドに指定した直接アドレシング命令でのアクセスは意図した結果が得られない場合があります。**

(4) 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・プログラムを作成できます。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。

コメント欄に記述できる文字は，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

DATA     DSEG     AT OFFD20H
HDTSA:   DS       1
STASC:   DS       2

CODE     CSEG     AT 0H
MAIN:    DW       START

         CSEG
         LOCATION 15
START:   MOV      RFM,#00
         MOVG     SP,#0FFE00H
         MOV      MM,#00
         MOV      STBC,#08H
         MOV      HDTSA,#1AH
         MOVG     WHL,#HDTSA          ;set hex 2-code data in WHL registor
         CALL     CONVAH              ;convert ASCII <- HEX
                                      ;output BC-register <- ASCII code
         MOVG     TDE,#STASC          ;set DE <- store ASCII code table
         MOV      A,B
         MOV      [TDE+],A
         MOV      A,C
         MOV      [TDE+],A
         BR       $$
         END
```

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$，ビット項および前述の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2－7　**演算子の種類**に示すものがあり，演算実行上の優先順位が表2－8　**演算子の優先順位**のとおり決められています。

演算の順序を変更するには，かっこ“（　）”を使います。

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2－7　演算子の種類

演算子の種類	演算子
算術演算子	＋符号，－符号，＋，－，＊，／，MOD
論理演算子	NOT, AND, OR, XOR
比較演算子	EQまたは = ，NEまたは＜＞，GTまたは＞， GEまたは＞ = ，LTまたは＜，LEまたは＜ =
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	（　）

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または = ，NEまたは＜＞，
　　GTまたは＞，GEまたは＞ = ，LTまたは＜，LEまたは＜ = ，SHR, SHL
N項演算子　　：MASK
その他　　：（　）

表2－8　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, DATAPOS, BITPOS, MASK
｜	2	＊，／，MOD, SHR, SHL
｜	3	＋，－
｜	4	AND
｜	5	OR, XOR
▼ 低い	6	EQまたは ＝ ，NEまたは＜＞，GTまたは＞，GEまたは＞ ＝ ，LTまたは＜，LEまたは＜ ＝

式の演算は，次の規則に従います。

① 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。
② 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。
③ 単項演算子の多重演算が可能です。

例 1 ＝ －－1 == 1
－1 ＝ －＋1 ＝ －1

④ 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フローした場合，オーバフローした値は無視します。
⑤ 定数が24ビット（0FFFFFFH）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。
⑥ 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。

2. 3. 1　演算子の機能

演算子の機能について説明します。

算術演算子　　算術演算子

(1) ＋（加算）

【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)
                    ⋮
```

【説　　明】

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」とも記述できます。

(2) －（減算）

【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     ; (b)
                    ⋮
```

【説　　明】

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，(b) は「BACK：　BR　！0FAH」とも記述できます。

(3) ＊（乗算）

【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    ; (c)
            ⋮
```

【説　　明】

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，（c）は「MOV A, #30H」とも記述できます。

(4) ／（除算）

【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   ; (d)
```

【説　　明】

「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，（d）は「MOV A, #5」とも記述できます。

算術演算子　　算術演算子

(5) MOD（剰余）

【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   ; (e)
```

【説　　明】

「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，（e）は「MOV A, #6」とも記述できます。

(6) ＋符号

【機　　能】

項の値をそのまま返します。

【使 用 例】

```
                    FIVE    EQU     +5
```

【説　　明】

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。

(7) －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
```

【説　　明】

－1は1の2の補数となります。

0000　0000　0000　0001の2つの補数は

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0FFFFHが定義されます。

論理演算子　　論理演算子

(1) NOT（否定）

【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,#NOT  3H     ; (a)
```

【説　　明】

3Hの論理否定をとります。

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

よって，0FFFCHを返します。

したがって，（a）は「MOVW AX, #0FFFCH」とも記述できます。

(2) AND（論理積）

【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  ; (b)
```

【説　　明】

6FAHと0FHの論理積をとります。

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

よって，0AHを返します。

したがって，（b）は「MOV A, #0AH」とも記述できます。

(3) OR（論理和）

【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   ; (c)
```

【説　　明】

0AHと1101Bの論理和をとります。

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

よって，0FHを返します。

したがって，(c) は「MOV A, #0FH」とも記述できます。

(4) XOR（排他的論理和）

【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   ; (d)
```

【説　　明】

9AHと9DHの排他的論理和をとります。

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

よって，7Hを返します。

したがって，(d) は「MOV A, #7H」とも記述できます。

比較演算子　　比較演算子

(1) EQまたは=（等しい）

【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)
```

【説　　明】

(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。

(2) NEまたは<>

【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          ; (c)
        MOV     A,#A1  NE   (A2+8H)       ; (d)
```

【説　　明】

(c) 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。

(d) 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。

(3) GTまたは＞（より大きい）

【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          ; (e)
        MOV     X,#A1  GT   (A2+10H)     ; (f)
```

【説　　明】

(e) 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。

(f) 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項の値と等しくなるので，00Hを返します。

(4) GEまたは>=　（より大きいか，または等しい）

【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          ; (g)
        MOV     X,#A1  GE   (A2+23H)     ; (h)
```

【説　　明】

(g) 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。

(h) 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。

比較演算子　　比較演算子

(5) LTまたは＜（より小さい）

【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           ; (i)
        MOV     X,#(A1+20H)  LT  A2     ; (j)
```

【説　　明】

(i) 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。

(j) 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項の値が等しくなるので，00Hを返します。

(6) LEまたは<= （より小さいか，または等しい）

【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。

LT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           ; (k)
        MOV     X,#(A1+7H)  LE  A2      ; (l)
```

【説　　明】

(k) 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。

(l) 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。

シフト演算子　　シフト演算子

(1) SHR（右シフト）

【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。

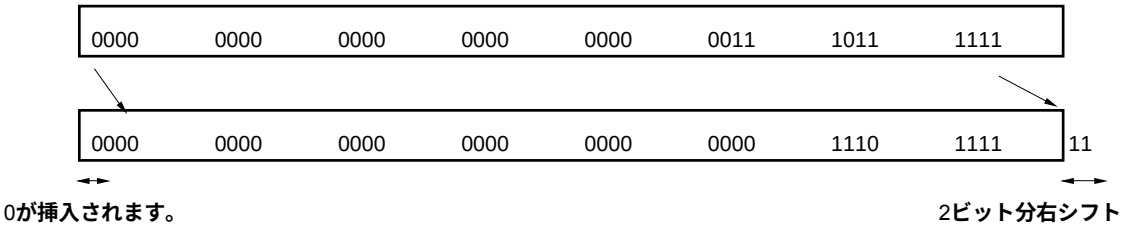
SHRの前後には，空白が必要です。

【使 用 例】

```
                    MOVW    RP1,#0003BFH  SHR  2     ; (a)
```

【説　　明】

0003BFHを2ビット分右シフトします。



よって，0000EFHを返します。

したがって（a）は，「MOV A, #0EFH」とも記述できます。

(2) SHL（左シフト）

【機　　能】

第1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。

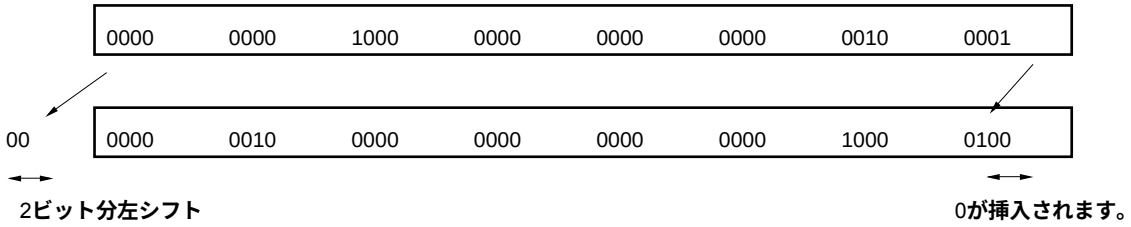
SHLの前後には，空白が必要です。

【使 用 例】

```
                    MOV     A,#800021H  SHL  2     ; (b)
```

【説　　明】

800021Hを2ビット分左シフトします。



よって，000084Hを返します。

したがって，（b）は，「MOV A, #84H」とも記述できます。

バイト分離演算子　　バイト分離演算子

(1) HIGH

【機　　能】

項の最下位16ビット中上位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  123456H     ; (a)
```

【説　　明】

MOV命令実行により，123456Hの下位16ビットの上位8ビット値34Hを返します。

したがって，(a) は「MOV A, #34H」とも記述できます。

(2) LOW

【機　　能】

項の最下位16ビット中下位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   123456H     ; (b)
```

【説　　明】

MOV命令実行により，123456Hの下位16ビットの下位8ビット値56Hを返します。

したがって，(b) は「MOV A, #56H」とも記述できます。

ワード分離演算子　　ワード分離演算子

(1) HIGHW

【機　　能】

項の32ビット中上位8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)
```

【説　　明】

MOVW命令実行により，12345678Hの32ビット中上位8ビット値12Hを返します。

したがって，(a) は「MOVW AX, #12H」とも記述できます。

(2) LOWW

【機　　能】

項の32ビット中下位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      ; (b)
```

【説　　明】

MOVW命令実行により，12345678Hの32ビット中下位16ビット値5678Hを返します。

したがって，(b) は「MOVW AX, #5678H」とも記述できます。

特殊演算子　　特殊演算子

(1) 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)
```

【説　　明】

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」とも記述できます。

(2) 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        ; (b)
```

【説　　明】

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。

(3) MASK

【機　　能】

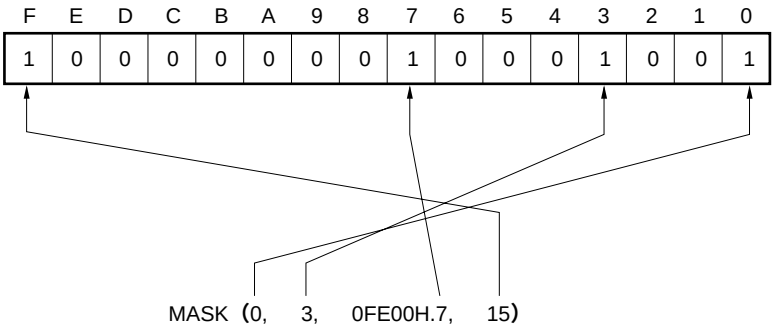
指定のビット位置に1，他を0にした16ビット値を返します。

【使 用 例】

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

【説　　明】

MOVW**命令実行により，**8089H**を返します。**



その他演算子　　その他演算子

(1) （ ）

【機　　能】

（ ）内の演算を，（ ）外の演算に先立って行います。

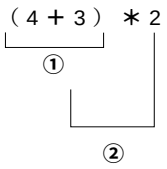
演算の優先順位を変更したいときに使います。

（ ）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】

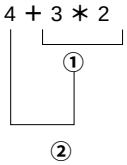
```
MOV     A,#（4+3）*2
```

【説　　明】



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2－8　**演算子の優先順位**を参照してください。

2.3.2　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，レーベルがあり，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。

(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種類に分かれます。

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**表2－9　リロケーション属性の種類**に示します。

表2－9　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・セグメント内のレーベル ・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，上記のレーベル，上記の$等，絶対値を定義したネーム
リロケータブル項	アセンブル時には値が決定しない項	・リロケータブル・セグメント内で定義したレーベル ・リロケータブル・セグメント内で定義したロケーション・アドレスを示す$ ・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]	他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レーベル ・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できる演算子は，‘＋’，‘－’，‘HIGH’，‘LOW’，‘HIGHW’，‘LOWW’の6つです。1つの式に外部参照シンボルは1つのみ記述可能です。その場合は必ず演算子“＋”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**表**2－10　**リロケーション属性による項と演算子の組み合わせ**のようになります。

表2－10　**リロケーション属性による項と演算子の組み合わせ（**1/2**）**

項のリロケーション属性 / 演算子の種類	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]
X ＊ Y	A	－	－	－
X ／ Y	A	－	－	－
X MOD Y	A	－	－	－
X SHL Y	A	－	－	－
X SHR Y	A	－	－	－
X EQ Y	A	－	－	A[注]
X LT Y	A	－	－	A[注]
X LE Y	A	－	－	A[注]
X GT Y	A	－	－	A[注]
X GE Y	A	－	－	A[注]
X NE Y	A	－	－	A[注]
X AND Y	A	－	－	－
X OR Y	A	－	－	－
X XOR Y	A	－	－	－
NOT X	A	A	－	－
＋ X	A	A	R	R
－ X	A	A	－	－

＜説　　明＞

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　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く，**X, Y **は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。**

表2－10　リロケーション属性による項と演算子の組み合わせ（2/2）

項のリロケーション属性 / 演算子の種類	X：ABS Y：ABS	X：ABS Y：REL	X：REL Y：ABS	X：REL Y：REL
HIGH X	A	A	R[注]	R[注]
LOW X	A	A	R[注]	R[注]
HIGHW X	A	A	R[注]	R[注]
LOWW X	A	A	R[注]	R[注]
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
MASK （X）	A	A	－	－

＜説　　明＞

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　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できる演算子は，‘＋’，‘－’，‘HIGH’，‘LOW’，‘HIGHW’，‘LOWW’の6つです（ただし，1つの式中に記述できる外部参照項は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表2－11　**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）**のようになります。

表2－11　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）

項のリロケーション属性 / 演算子の種類	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

＜説　明＞

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。

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。

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) 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種類に分かれます。

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表2－12　**演算におけるシンボル属性の種類**に示します。

表2－12　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**表2－13　シンボル属性による項と演算子の組み合わせ**のようになります。

表2－13　シンボル属性による項と演算子の組み合わせ

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

＜説　　明＞

ADDRESS　：ADDRESS項

NUMBER　：NUMBER項

A　　：演算結果がADDRESS項になります。

N　　：演算結果がNUMBER項になります。

－　　：演算不可

(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レーベルであると仮定します。

① 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」となりますので，この演算を表**2－10 **リロケーション属性による項と演算子の組み合わせ**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。**

② 演算とシンボル属性

「TABLE＋5H**」は「**ADDRESS**項＋**NUMBER**項」となりますので，この演算を表**2－13　**シンボル属性による項と演算子の組み合わせ**にあてはめます。

演算子の種類　……　X＋Y
項のシンボル属性　……　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。**

2.4　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。

ビット位置指定子　　ビット位置指定子

(1) .（ビット位置指定子）

【記述形式】

X［△］.［△］Y

└──────┘

ビット項

X（第1項）とY（第2項）の組み合わせ

X（第1項）		Y（第2項）
汎用レジスタ	A	式（0-7）
	X	式（0-7）
制御レジスタ	PSWL	式（0-7）
	PSWH	式（0-7）
特殊機能レジスタ	sfr[注]	式（0-7）
メモリ	[DE] [注]	式（0-7）
	[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，0FE20H-0FF1FHの範囲でなければなりません。ただし，CHGSFR制御命令またはアセンブラ・オプション（-CS）指定により，範囲が変わります。
　③ 外部参照シンボルを記述できます。
・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属性の式のみ記述できます。
　③ 外部参照シンボルは記述できません。

【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項の組み合わせを表2－14　**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**に示します。

表2－14　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

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

＜説　　明＞

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。

R　　：演算結果がリロケータブル項になります。

E　　：演算結果が外部参照項になります。

－　　：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表2－15　**ビット・シンボルが持つ値**に示します。

表2－15　ビット・シンボルが持つ値

オペランドの種類	シンボル値
A.bit1[注2]	1H.bit1
X.bit1[注2]	0H.bit1
PSWL.bit1[注2]	1FEH.bit1
PSWH.bit1[注2]	1FFH.bit1
sfr[注1]. bit1[注2]	0××××××H.bit1[注3]
式.bit1[注2]	0××××H.bit1[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。

2．bit1 = 0-7

3．0××××××Hは，sfrのアドレス

4．0××××Hは，式の値

【使 用 例】

```
        MOV1    CY,0FFD20H.3
        AND1    CY,A.5
        CLR1    P1.2
        SET1    1+0FFD30H.3         ;0FFD31H.3に等しい
        SET1    0FFD40H.4+2         ;0FFD40H.6に等しい
```

2.5　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すから，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，レーベル）

・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，レーベル）

2.5.1　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／レー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，また，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**表2－16　インストラクションのオペランド値の範囲**，**表2－17　疑似命令のオペランド値の範囲**に示します。

表2－16　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-0FFH	
word	16ビット値　0H-0FFFFH	
imm24	24ビット値　0H-0FFFFFFH	
saddr1	xFE00H-xFEFFH[注1]	
saddrg1	xFE00H-xFEFDH[注1]	
saddrp1	xFE00H-xFEFFHの偶数値[注1]	
saddr2[注2]	xFD20H-xFDFFH[注1]	
	xFF00H-xFF1FH[注1]	
saddrg2	xFD20H-xFDFFH[注1]	
saddrp2	xFD20H-xFDFFHの偶数値[注1]	
	xFF00H-xFF1FH[注1]	
sfr	xFF20H-xFFFFH[注1]	
sfrp	xFF20H-xFFFFHの偶数値[注1]	
addr24	0H-0FFFFFFH	
addr20[注3]	0H-0FCFFH, 10000H-FFFFFH	
addr16[注3]	MOVTBLW	xFE00H-xFEFFH[注1]
	その他の命令	0H-0FCFFH
MOVTBLWのaddr16[注3]	nFE00H-nFEFFH	
addr11	800H-0FFFH	
addr5	40H-7EHの偶数値	
bit	3ビット値　0-7	
n	3ビット値　0-7	
MOVTBLW, MACWのn8	8ビット（0H-FFH）	
locaddr	0H, 0FH	

注1．範囲は，対象とするターゲットの品種ごとに異なります。また，xのディフォールト値は，x = Fです。xは，SFR領域変更制御命令（CHGSFR）により範囲が変わります。

2．saddrp2の場合は，xFF00H-xFF1FHの範囲を除きます。

3．シンボルが奇数番地でも可能です。

表2－17　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-0FCFFH, 10000H-FFFFFH[注1]
	DSEG AT	0H-0FFFFFFH
	BSEG AT	0H-0FFFFFFH[注2]
	ORG	0H-0FFFFFFH
シンボル定義	EQU	24ビット値　0H-0FFFFFFH
	SET	24ビット値　0H-0FFFFFFH
メモリ初期化領域確保	DB	8ビット値　0H-0FFH
	DW	16ビット値　0H-0FFFFH
	DG	24ビット値　0H-0FFFFFFH
	DS	24ビット値　0H-0FFFFFFH
分岐命令自動選択	BR	0H-0FFEFFH
	CALL	0H-0FFEFFH
利用レジスタ選択	RSS	1ビット値　0, 1

注1．ディフォールトの範囲を示しています。範囲はSFR領域変更制御命令（CHGSFR）により変更可能です。詳細は，4. 8　SFR領域変更制御命令を参照してください。

2．0H-0FFFFFFHのうち，SFR領域は除きます。

2. 5. 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。

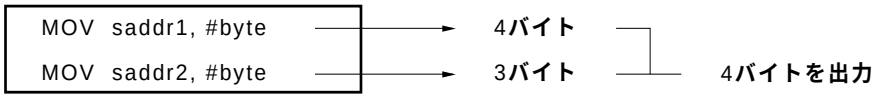
したがって，命令の要求するオペランドのサイズ以上のデータを記述すると，エラーとなります。なお，式の演算は，符号なし，32ビットで行います。評価結果が0FFFFFFH（24ビット）を越えた場合にはワーニング・メッセージを出力します。

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を行います。この場合も，評価チェックについては，表2－19　疑似命令のオペランドとしての記述可能なシンボルの性質の値の正当性チェック欄にあるとおり，一部のオペランドでは値のチェックをせずに必要な部分のみ取り出してオブジェクトに埋め込むので注意が必要です。

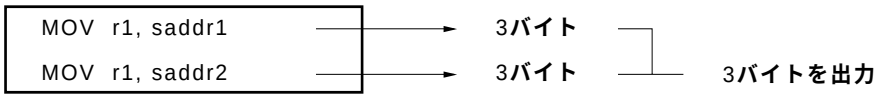
・saddr領域に関する注意事項

SADDR領域をアブソリュートな記述で前方参照／リロケータブルな記述で前・後方参照，かつsaddr1とsaddr2（またはsaddrg1とsaddrg2，以下同様に読み替えます）の両方のオペランドの記述形式をもつニモニックがある場合に，アセンブラはsaddr1とsaddr2の記述形式のうち長い方の形式である（同じであればどちらかで問題はありません）saddr1のオブジェクト・コード・サイズで出力します。

例1



例2



そのシンボルがsaddr1かsaddr2であるかはアブソリュートな記述で前方参照の場合にはパス1終了時点で判断可能で，オブジェクト・コードを出力するパス2ではsaddr1またはsaddr2のいずれかの適正なオブジェクト・コードを出力します。
このときに，パス1で決定したオブジェクト・サイズと異なる場合は異なるオブジェクト・サイズ分（通常は1バイト）をnopの“ 00 ”として出力し，ワーニング・エラー・メッセージ（W714）を出力します。

例

```
                DESG    AT  ????
        SYM:    DB      1
                MOV     SYM,#10H
```

アセンブラはSYMがsaddr1であれば，「MOV saddr1, #byte」のオブジェクト・コードXXXXXXXX（4バイト）を出力します。saddr2であれば「MOV saddr2, #byte」のオブジェクト・コードとしてYYYYYY00（3バイト＋00）を出力します。

リロケータブルな前・後方参照の場合はアセンブラ時ではそのシンボルがsaddr1かsaddr2であるかは判断できません。したがって，アセンブラはsaddr1とsaddr2の記述形式のうち長い方の形式であるsaddr1のオブジェクト・コードをリスト・ファイル（およびオブジェクト・ファイル）に出力します。
リンカはシンボルのアドレスを解決し，そのシンボルがsaddr1かsaddr2かを判断します。
リンカはsaddr1またはsaddr2のいずれかの適正なオブジェクト・コードに修正を行います。このときに，アセンブラで決定したオブジェクト・サイズと異なる場合は異なるオブジェクト・サイズ分（通常は１バイト）をnopの“ 00 ”として出力し，ワーニング・エラー・メッセージ（W714）を出力します。

例

```
                DESG
        SYM:    DB      1
                MOV     SYM,#10H
```

リンカはsaddr1であればアセンブラが出力したオブジェクト・コードXXXXXXXXを修正しません。saddr2**であれば「**MOV saddr2, #byte**」のオブジェクト・コード**YYYYYY**で修正した後に“** 00 **”を付加し**YYYYYY00**（4バイト）に修正します。**

・78K/Ⅱ，78K/Ⅲのソース・プログラムを利用されるユーザに対しサポートする機能

78K/Ⅱ，78K/Ⅲのソース・プログラムを78K/Ⅳで使われるお客様に対して，スタック操作命令をサポートする機能があります。

78K/Ⅱ，78K/Ⅲの記述可能であった“ MOV SP, #WORD ”を78K/Ⅳのプログラムは検出すると，ワーニング・メッセージ（W713）を出力し，“ MOVG SP, #imm24 ”のオブジェクト・コードを出力します。

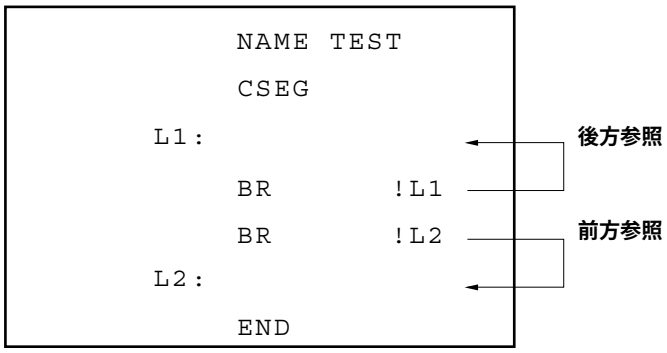
2. 5. 3　オペランドのシンボル属性，リロケーション属性

命令のオペランドとしてネーム，レーベル，$（ロケーション・カウンタを示す）を記述する場合，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. 3. 2　**演算の制限**参照）また，ネーム，レーベルの場合に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，レー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，レー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，レーベルがそれ以降の行で定義されている。

例



これらシンボル属性，リロケーション属性，ネーム，レーベルの参照方向の条件を**表**2－18　**インストラクションのオペランドの属性**，**表**2－19　**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**に示します。

表2－18　インストラクションのオペランドの属性

記述形式＼参照パターン		－	SET, EQU疑似命令で定義 後方	SET, EQU疑似命令で定義 前方	レーベルが存在するCSEG/DSEGセグメントの再配置属性またはEXTRN疑似命令に指定する再配置属性 saddr[注1] / saddrp 後方	saddr / saddrp 前方	saddr2[注1] / saddrp2 後方	saddr2 / saddrp2 前方	saddra[注1] 後方	saddra 前方	fixed / fixeda 後方	fixed / fixeda 前方	callt0 後方	callt0 前方	その他[注2] 後方	その他[注2] 前方	なし[注1, 3] 後方	なし[注1, 3] 前方	SFR予約語
byte		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×
word		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×
imm24		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×
saddr1		○	○	○	○	○	×	×	○	○	○	○	○	○	○	○	○	○	×
saddrg1		○	○	○	○[注4]	○[注4]	×	×	○	○	○	○	○	○	○	○	○	○	×
saddrp1		○	○	○	○[注4]	○[注4]	－	－	○	○	○	○	○	○	○	○	○	○	×
saddr2[注5]	注6	○	○	○[注8]	×	×	○[注4]	注4, 8 ○	○[注9]	○[注9]	－	－	－	－	－	－	－	－	－
	注7	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－	○[注10]
saddrg2		○	○	○[注8]	×	×	○[注4]	○[注4]	○[注9]	○[注9]	－	－	－	－	－	－	－	－	－
saddrp2	注6	○	○	○[注8]	×	×	○	○	○[注9]	○[注9]	－	－	－	－	－	－	－	－	－
	注7	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－	－	○[注11]
sfr		○[注12]	○[注13]	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○[注13]
sfrp		×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○[注3]
addr24		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×
addr20		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×
addr16		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×
MOVTBLWのaddr16		○	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
addr11		○	○	○	×	×	×	×	×	×	○	○	×	×	×	×	×	×	×
addr5		○	○	○	×	×	×	×	×	×	×	×	○	○	×	×	×	×	×
bit		○	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
n		○	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
MOVTBLW, MACWのn8		○	○	○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
locaddr		○	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×

＜説　明＞

前方　：前方参照を意味します。

後方　：後方参照を意味します。

○　：記述可能を意味します。

×　：エラーを意味します。

－　：記述不可能を意味します。

注1．extern**疑似命令の再配置属性を指定することによります。**

2．at, gram, unit, unitp, base**などの欄の左記以外の再配置属性**

3．16**ビット・アクセス可能な**sfr**予約語のみ**

4．**シンボルが奇数番地でも可能です。**

5．saddr2**の場合は，**nFF00H-nFF1FH**の範囲を除く**

6．nFD20H-nFDFFH

7．nFF00H-nFF1FH

8．**アセンブラが**nop**の“**00**”をオブジェクト・コードのうしろに付加する場合があります。詳細は，**2. 5. 2　**命令の要求するオペランドのサイズ**を参照してください。

9．**アセンブラは**saddr1/saddrg1**のオブジェクト・コードを出力し，場合によりリンカが**nop**の“**00**”を付加します。詳細は，**2. 5. 2　**命令の要求するオペランドのサイズ**を参照してください。

10．sfr**領域のうち**saddr2**アクセス可能な領域の**sfr**予約語で８ビット・アクセス可能な**sfr**予約語**

11．sfr**領域のうち**saddr2**アクセス可能な領域の**sfr**予約語で**16**ビット・アクセス可能な**sfr**予約語**

12．**外部アクセス領域の範囲で絶対式のみ可能です。**

13．**８ビット・アクセス可能な**sfr**予約語のみ**

表2－19　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, SADDR1, SADDR2						BIT						値の正当性チェック
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 / 疑似命令		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	
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	○	○	○	○	○	○	○	○	－	－	－	－	－	－	× (16)
DG	サイズ	○[注1]	－	－	－	－	－	－	－	－	－	－	－	－	－	－
	初期値	○	○	○	○	○	○	○	○	－	－	－	－	－	－	× (24)
DS		○	－	－	－	－	－	－	－	－	－	－	－	－	－	－
BR/CALL		○	－	－	－	－	－	－	－	－	－	－	－	－	－	－
RSS		○[注1]	－	－	－	－	－	－	－	－	－	－	－	－	－	◎

○：記述可能　　－：記述不可

＜説　　明＞

“値の正当性チェック”欄は，アブソリュートな式であればアセンブラで，リロケータブルな式や外部参照式であればリンカで行われる値の正当性のチェックです。

- **◎　　：値の範囲に従ったチェックを行います。**
- **○　　：アセンブラ内でアブソリュートなBITのみ，**saddr1.bit/sadr2.bit/**外部アクセス領域**.bit**の値の範囲に従ったチェックを行います。**
- **×（16）：計算結果の下位16ビットをオブジェクトに埋め込みます。**
- **×（24）：計算結果の下位24ビットをオブジェクトに埋め込みます。**
- **－　　：値の範囲がない場合**

注1．絶対式のみが記述可能です。

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。

ただし，ここでのADDRESS属性には，SADDR1属性とSADDR2属性を含みます。

- 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

以上の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

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

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，RA78K4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　疑似命令の概要

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。**表3－1　疑似命令一覧表**にその種類を示します。

- ソース・プログラム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。

表3－1　疑似命令一覧表

項番	疑似命令の種類	疑似命令
1	セグメント定義疑似命令	CSEG, DSEG, BSEG, ORG
2	シンボル定義疑似命令	EQU, SET
3	メモリ初期化，領域確保疑似命令	DB, DW, DG, DS, DBIT
4	リンケージ疑似命令	PUBLIC, EXTRN, EXTBIT
5	オブジェクト・モジュール名宣言疑似命令	NAME
6	自動選択疑似命令	BR, CALL
7	汎用レジスタ選択疑似命令	RSS
8	マクロ疑似命令	MACRO, LOCAL, REPT, IRP, EXITM, ENDM
9	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

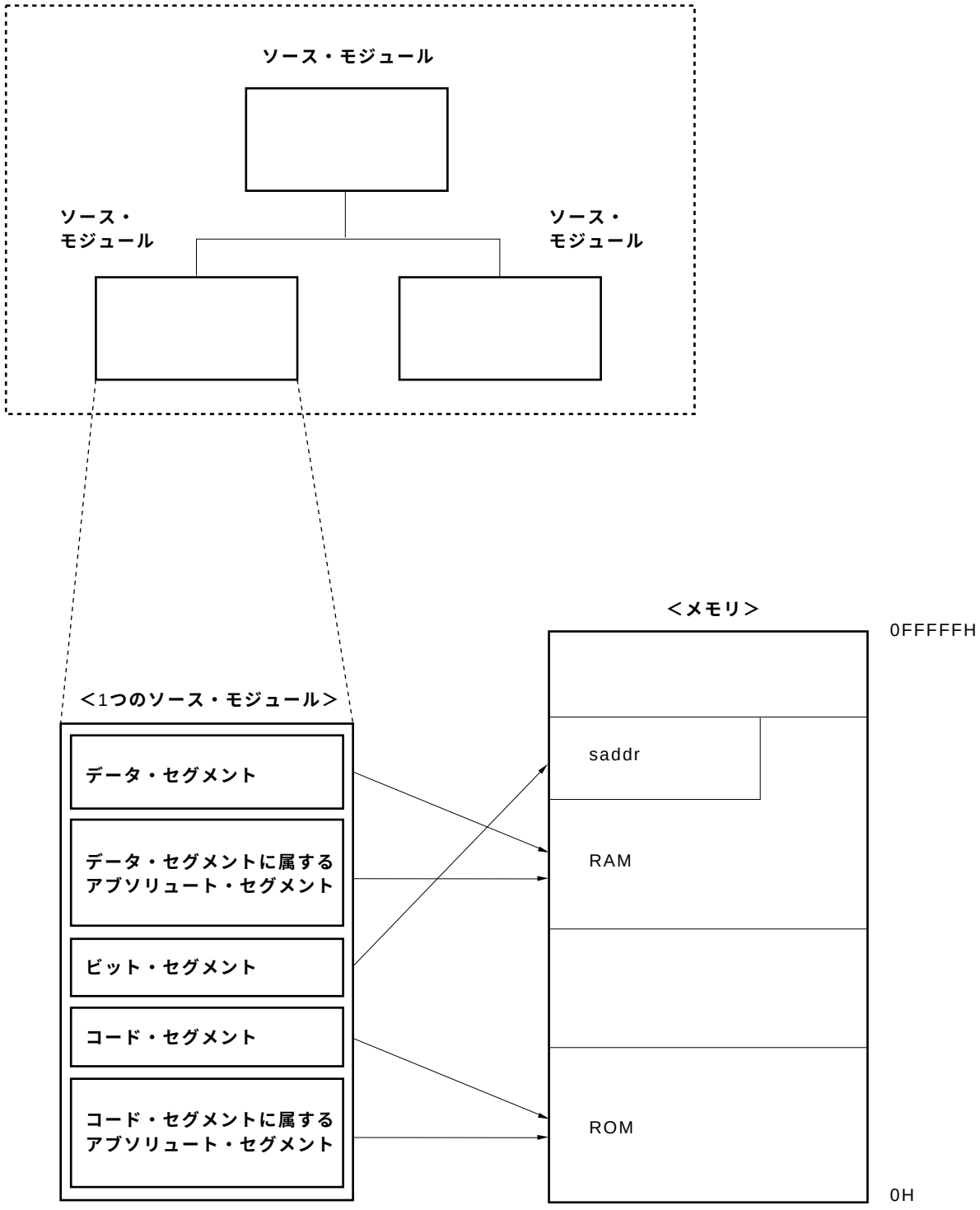
セグメントの種類により，メモリのどの範囲に配置されるかが決まります。
各セグメントの定義方法と配置されるメモリ・アドレスを**表**3－2　**セグメントの定義方法と配置されるメモリ・アドレス**に示します。

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。
セグメントの配置の例を，**図**3－1　**セグメントのメモリ配置**に示します。

図3ー1　セグメントのメモリ配置



CSEG　　code segment　　CSEG

（1）CSEG（code segment）

【記述形式】

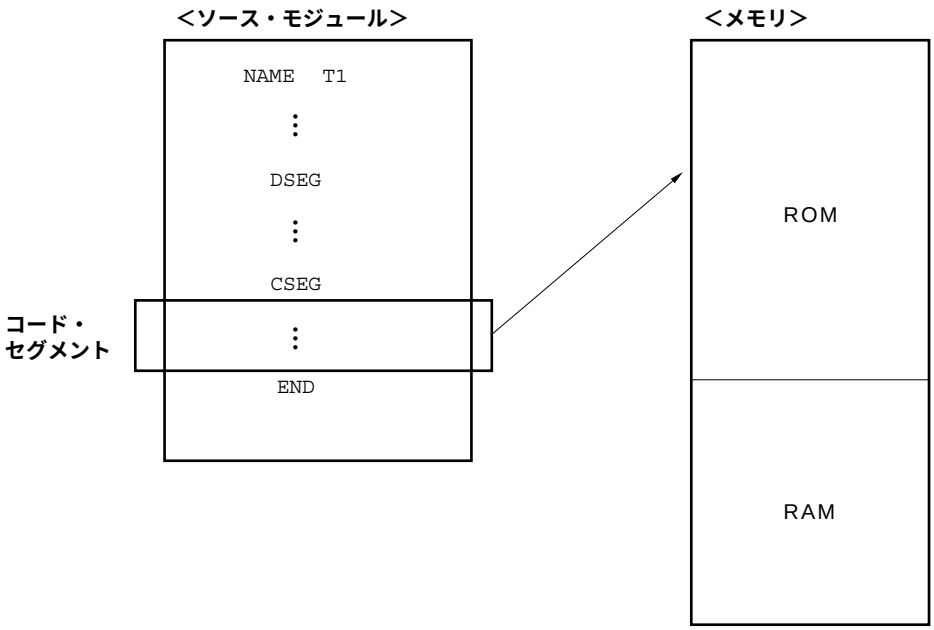
シンボル欄	ニモニック欄	オペランド欄	コメント欄
[セグメント名]	CSEG	[再配置属性]	[；コメント]

【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。

図3－2　コード・セグメントの再配置



【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（他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。

CSEG　　code segment　　CSEG

【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。再配置属性を表3－3 CSEGの再配置属性に示します。

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0040H-007FH番地内で先頭が2の倍数になるように配置することをアセンブラに指示します。1バイト命令“ CALLT ”でコールするサブルーチンのエントリ・アドレスを定義しているコード・セグメントの場合に指定してください。
FIXED	FIXED	指定セグメントの先頭を0800H-0FFFH番地に配置することをアセンブラに指示します。2バイト命令“ CALLF ”でコールするサブルーチンを定義しているコード・セグメントの場合に指定してください。
FIXEDA	FIXEDA	指定セグメントの先頭を0800H-0FFFH番地に配置し，終わりを0FCFFH[注]内に配置することをアセンブラに指示します。2バイト命令“ CALLF ”でコールするサブルーチンを定義しているコード・セグメントの場合に指定してください。
AT	AT絶対式	指定セグメントを絶対番地に配置します（0000H-0FCFFH, 10000H-0FFFFFH内）[注]。
UNIT	UNIT	指定セグメントを任意の位置へ配置します（0080H-0FCFFH, 10000H-0FFFFFH内）[注]。
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（0080H-0FCFFH, 10000H-0FFFFFH内）[注]。
BASE	BASE	指定セグメントを80H-0FCFFHに配置します。[注]
PAGE	PAGE	指定セグメントをxxx00H-xxxFFH内に配置します（0FFFFFH以内）。
PAGE64K	PAGE64K	64K境界にまたがらないように配置します（0H-0FCFFH, 10000H-FFFFFH）[注]。

注　範囲は，SFR領域変更制御命令（CHGSFR）により変更可能です。

CSEG　　code segment　　CSEG

・再配置属性が省略された場合，“ UNIT ”と解釈されます。
・表3－3　CSEGの再配置属性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・メッセージを出力し，“ 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・メッセージを出力し，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ィフォールトのセグメント名を与えます。
ディフォールト・セグメント名を表3－4　CSEGのディフォールト・セグメント名に示します。

表3－4　CSEGのディフォールト・セグメント名

再配置属性	ディフォールト・セグメント名
CALLT0	?CSEGT0
FIXED	?CSEGFX
FIXEDA	?CSEGFXA
BASE	?CSEGB
PAGE	?CSEGP
PAGE64K	?CSEGP64
UNIT（または省略時）	?CSEG
UNITP	?CSEGUP
AT	セグメント名省略不可 セグメント名を省略した場合はUNITと解釈され，?CSEGとなる

・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。
・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。
・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合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，小文字を区別します。

CSEG　　code segment　　CSEG

【使 用 例】

```
        NAME    SAMP1
C1      CSEG                            ; (1)

C2      CSEG    CALLT0                  ; (2)

        CSEG    FIXED                   ; (3)

C1      CSEG    CALLT0                  ; (4)

        CSEG                            ; (5)

        END
```

＜解　　説＞

(1) セグメント名が“ C1 ”，再配置属性が“ UNIT ”と解釈します。

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

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

(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 ”と解釈します。

DSEG　　data segment　　DSEG

(2) DSEG（data segment）

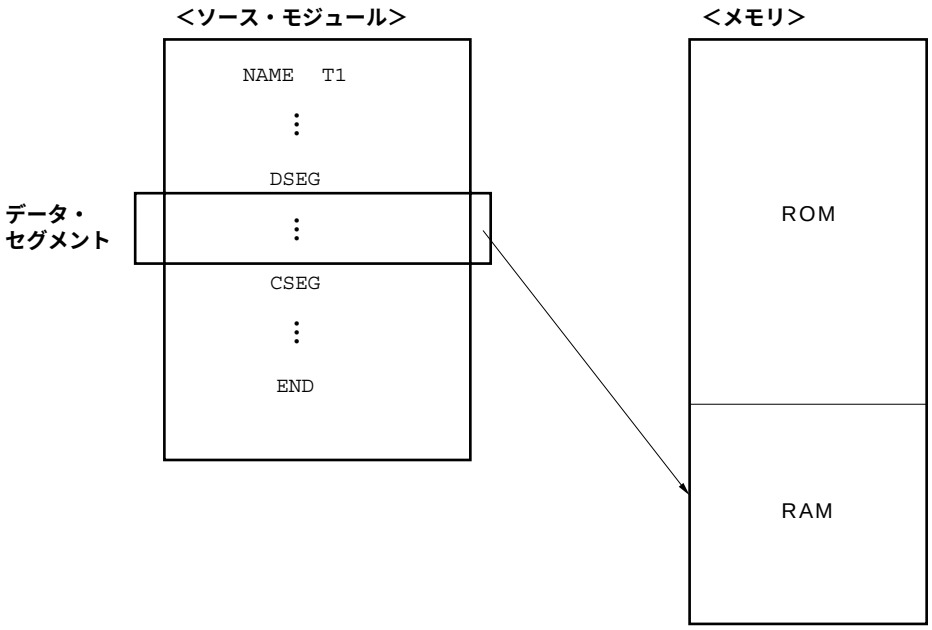
【記述形式】

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。

図3－3　データ・セグメントの再配置



【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レーベルを付けます。ソース・プログラムを記述する場合，このレーベルを利用します。データ・セグメントとして確保された領域は，RAM上で他の作業領域（スタック領域，汎用レジスタ領域，他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。

DSEG　　data segment　　DSEG

【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。再配置属性を表3－5　DSEGの再配置属性に示します。

表3－5　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saddr1領域に配置します（saddr1領域：0FE00H-0FEFFH[注1]）。
SADDR2	SADDR2	指定セグメントをsaddr2領域に配置します（saddr2領域：0FD20H-0FDFFH[注1, 2]）。
SADDRP	SADDRP	指定セグメントをsaddr1領域の偶数番地から配置します（saddr1領域：0FE00H-0FEFFH[注1]）。
SADDRP2	SADDRP2	指定セグメントをsaddr2領域の偶数番地から配置します（saddr2領域：0FD20H-0FDFFH[注1, 2]）。
SADDRA	SADDRA	指定セグメントをsaddr領域の任意の領域に配置します（saddr領域：0FD20H-0FEFFH（saddr1/saddr2領域）[注1, 2]）。
AT	AT絶対式	指定セグメントを絶対番地に配置します。
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（メモリ領域名“RAM”内[注1]）。
UNITP	UNITP	指定セグメントを任意の位置へ，偶数番地から配置します（メモリ領域名“RAM”内[注1]）。
DTABLE	DTABLE	指定セグメントをマクロ・サービス・コントロール領域内に配置します（マクロ・サービス・コントロール領域：0FE00H-0FEFFH[注1, 2]）。
DTABLEP	DTABLEP	指定セグメントをマクロ・サービス・コントロール領域内に偶数番地から配置します（マクロ・サービス・コントロール領域：0FE00H-0FEFFH[注1, 2]）。
LRAM	LRAM	指定セグメント名を周辺RAM領域（低速RAM内[注1]）に配置します。
GRAM	GRAM	指定セグメントを汎用スタティックRAM領域（高速RAM内）に配置します（汎用スタティックRAM領域：0FD00H-0FEFFH）。
PAGE	PAGE	指定セグメントをXXXX00H-XXXXFFHの任意の位置へ配置します（0FFFFFH以内）。
PAGE64K	PAGE64K	指定セグメントを64K境界にまたがらないように配置します（0H-0FCFFH, 10000H-FFFFFH[注2]）。

注1．各対象品種によってアドレスが異なる場合があります。

2．ディフォールトの範囲を示してあります。
範囲はSFR領域変更制御命令（CHGSFR）により変更可能です。

・再配置属性が省略された場合，“UNIT”と解釈されます。

・表3－5　DSEGの再配置属性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・メッセージを出力し，“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・メッセージを出力し，絶対式の値を0とみなし処理を続けます。

・DSEG疑似命令のシンボル欄にセグメント欄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ィフォールトのセグメント名を与えます。ディフォールト・セグメント名を表3－6　DSEGのディフォールト・セグメント名に示します。

DSEG　　data segment　　DSEG

表3－6　DSEGのディフォールト・セグメント名

再配置属性	ディフォールト・セグメント名
SADDR	?DSEGS
SADDRP	?DSEGSP
SADDR2	?DSEGS2
SADDRP2	?DSEGSP2
SADDRA	?DSEGA
UNIT（または省略時）	?DSEG
UNITP	?DSEGUP
DTABLE	?DSEGDT
DTABLEP	?DSEGDTP
PAGE	?DSEGP
PAGE64K	?DSEGP64
LRAM	?DSEGL
GRAM	?DSEGG
AT	セグメント名省略不可 UNITと解釈されて，?DSEGとなる

- 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。
- 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合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，小文字を区別します。

DSEG　　data segment　　DSEG

【使 用 例】

```
        NAME    SAMP1
        LOCATION 0H
        DSEG                                    ; (1)
WORK1:  DS      1
WORK2:  DS      2
        CSEG
        MOV     A,!WORK1                        ; (2)
        MOV     A,WORK1                         ; (3)
        MOVW    DE,#WORK2                       ; (4)
        MOVW    AX,[DE]
        MOVW    AX,WORK2                        ; (5)
        END
```

<解　　説>

(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。ディフォー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。リロケータブルなレーベル‘WORK1’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。リロケータブルなレーベル‘WORK2’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　　bit segment　　BSEG

(3) BSEG (bit segment)

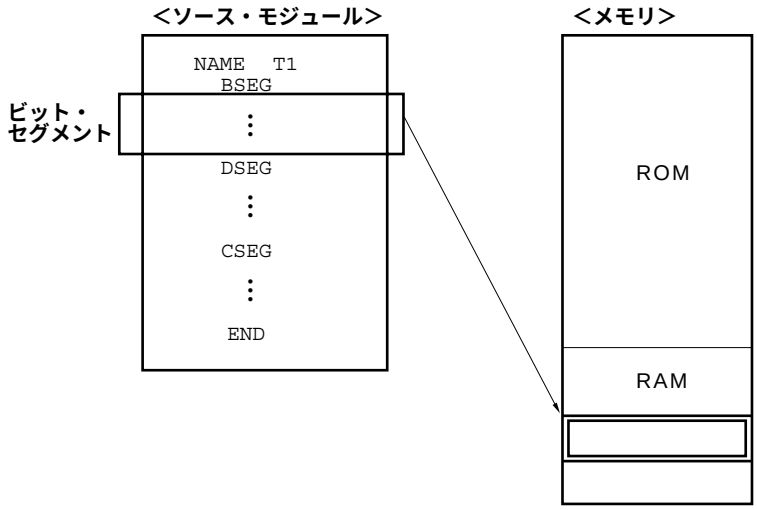
【記述形式】

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。

図3－4　ビット・セグメントの再配置



【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（使用例参照）。
- 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。再配置属性を表3－7　BSEGの再配置属性に示します。

BSEG　　bit segment　　BSEG

表3ー7　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（0H-0FFFFFFHでSFR領域以外の領域）。
SADDR	SADDR	指定セグメントをsaddr1領域内の任意の位置へ配置します（saddr1領域：0FE00H-0FEFFH[注1]）。
SADDR2	SADDR2	指定セグメントをsaddr2領域内の任意の位置へ配置します（saddr2領域：0FD20H-0FDFFH[注1, 2]）。
SADDRA	SADDRA	指定セグメントをsaddr領域内の任意の位置へ配置します（saddr領域：0FD20H-0FEFFH[注1, 2]）。
UNIT	UNITまたは指定なし	指定セグメントをsaddr1領域内の任意の位置へ配置します（saddr1領域：0FE00H-0FEFFH[注1]）。
GRAM	GRAM	指定セグメントを，汎用スタティックRAM内に配置します（高速RAM：0FD00H-0FEFFH[注1]）。
ARAM	ARAM	指定セグメントを全空間の任意の位置へ配置します（0H-0FFFFFHでSFR領域以外の領域）。

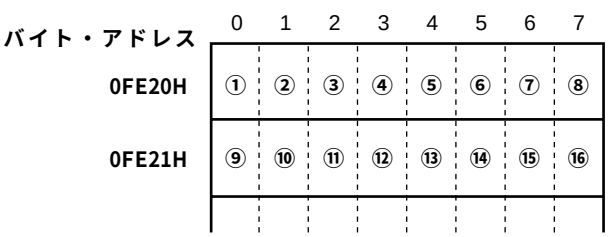
注1．各対象品種によってアドレスが異なる場合があります。

2．ディフォールトの範囲を示してあります。
範囲はSFR領域変更制御命令（CHGSFR）により変更可能です。

- 再配置属性が省略された場合，“ UNIT ”と解釈されます。
- 表3ー7　BSEGの再配置属性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・メッセージを出力し，“ 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“ 00xxxx.b ”の形式で表示します（バイト・アドレスは16進6桁，ビット位置は16進1桁（0-7））。

BSEG　　bit segment　　BSEG

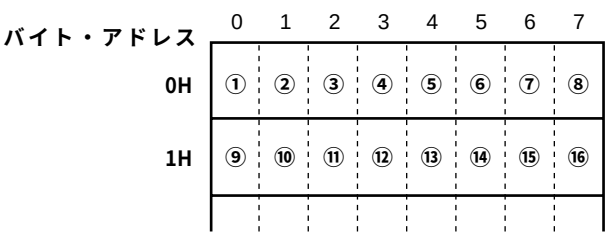
アブソリュートな場合



ロケーション・カウンタ

①0FE20H. 0	⑨0FE21H. 0
②0FE20H. 1	⑩0FE21H. 1
③0FE20H. 2	⑪0FE21H. 2
④0FE20H. 3	⑫0FE21H. 3
⑤0FE20H. 4	⑬0FE21H. 4
⑥0FE20H. 5	⑭0FE21H. 5
⑦0FE20H. 6	⑮0FE21H. 6
⑧0FE20H. 7	⑯0FE21H. 7

リロケータブルな場合



ロケーション・カウンタ

①0H. 0	⑨1H. 0
②0H. 1	⑩1H. 1
③0H. 2	⑪1H. 2
④0H. 3	⑫1H. 3
⑤0H. 4	⑬1H. 4
⑥0H. 5	⑭1H. 5
⑦0H. 6	⑮1H. 6
⑧0H. 7	⑯1H. 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
00FE20H.0	0000
00FE20H.1	0001
00FE20H.2	0002
⋮	⋮
00FE20H.7	0007
00FE21H.0	0008
00FE21H.1	0009
⋮	⋮
00FE80H.0	0300
⋮	⋮

BSEG　　bit segment　　BSEG

・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・メッセージを出力し，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ィフォールトのセグメント名を与えます。ディフォールト・セグメント名を表3－8　BSEGのディフォールト・セグメント名に示します。

表3－8　BSEGのディフォールト・セグメント名

再配置属性	ディフォールト・セグメント名
UNIT（または省略時）	?BSEG
UNITP	?BSEGUP
AT	セグメント名省略不可 UNITと解釈され，?BSEGとなる
SADDR	?BSEGS
SADDRP	?BSEGSP
SADDR2	?BSEGS2
SADDRP2	?BSEGSP2
SADDRA	?BSEGSA
GRAM	?BSEGG
ARAM	?BSEGA

・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（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。
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，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合わせて，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，小文字を区別します。

BSEG　　bit segment　　BSEG

【使 用 例】

```
        NAME    SAMP1

        FLAG    EQU     0FE20H
        FLAG0   EQU     FLAG.0              ; (1)
        FLAG1   EQU     FLAG.1              ; (1)

        BSEG                                ; (2)
        FLAG2   DBIT

        CSEG
        MOV1    CY, FLAG0                   ; (3)
        MOV1    CY, FLAG2                   ; (4)

        END
```

<解　説>

(1) バイト・アドレス境界を意識して，ビット・アドレス（0FE20Hのビット0，ビット1）を定義しています。

(2) 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“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。

(3) この記述は，「MOV1 CY, 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。

(4) この記述では，バイト・アドレスが意識されません。

ORG　　origin　　ORG

(4) ORG (origin)

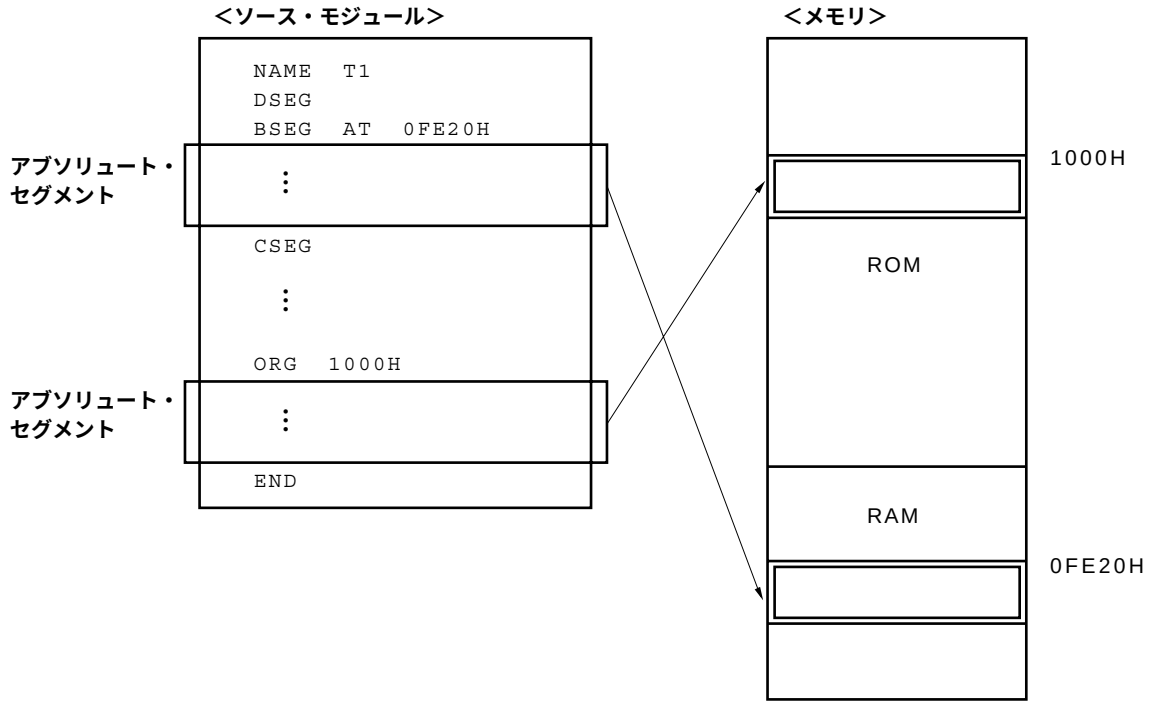
【記述形式】

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	絶対式	[；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。

図3－5　アブソリュート・セグメントの配置



【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。

ORG　　origin　　ORG

【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は記述できません。また，ビット・セグメントに属するアブソリュート・セグメントの記述はできません。**
- **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。**
- **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セグメント名を与えます。**

  “ ?Axxxxxx ”というセグメント名を与えます。xxxxxxには指定されたセグメントの先頭アドレス000000-FFFFFF**（16進6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／レーベルを記述する場合，そのネーム／レーベルは，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。**
- **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合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，小文字を区別します。**

ORG　　origin　　ORG

【使 用 例】

```
        NAME        SAMP1
        LOCATION    0H

        DSEG
        ORG         0FE20H                  ; (1)
SADR1:  DS          1
SADR2:  DS          1
SADR3:  DS          2

MAINO   ORG         100H
        MOV         A,SADR1                 ; (2)

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

<解　　説>

(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・アドレシング領域の先頭アドレスFE20H**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ますので，アセンブラが自動的に?A00FE20**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ませんので，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，EQU, SET疑似命令があります。

EQU　　equate　　EQU

（1）EQU（equate）

【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（使用例参照）。

【説　　明】

・EQU疑似命令のオペランドにネーム／レー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／レーベルを使います。
また，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。
・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。
（a）ADDRESS属性の式1－ADDRESS属性の式2
（b）ADDRESS属性の式1　比較演算子　ADDRESS属性の式2
（c）上記（a），（b）の中で，次の①，②があてはまる場合。
① ADDRESS属性の式1中のレーベル1とADDRESS属性の式2のレー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。
② レーベル1とレーベル2が別セグメントであり，属するセグメントの先頭からレー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。

EQU　　equate　　EQU

(d) HIGH　アブソリュートなADDRESS属性の式

(e) LOW　アブソリュートなADDRESS属性の式

(f) HIGHW　アブソリュートなADDRESS属性の式

(g) LOWW　アブソリュートなADDRESS属性の式

(h) DATAPOS　アブソリュートなADDRESS属性の式

(i) BITPOS　アブソリュートなADDRESS属性の式

(j) 上記(d)，(e)，(f)，(g)，(h)，(i)の中で，次の③があてはまる場合。

③　ADDRESS属性の式中のレーベルと，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できるビット値とその参照可能範囲を表3－9　ビット値を示すオペランドの表現形式に示します。

表3－9　ビット値を示すオペランドの表現形式

オペランドの種類	シンボル値	参照可能範囲
A.bit1[注1]	1.bit1	同一モジュール内でのみ参照可能
X.bit1[注1]	0.bit1	
PSWL.bit1[注1]	1FEH.bit1	
PSWH.bit1[注1]	1FFH.bit1	
sfr[注2].bit1[注1]	00FF××H[注3].bit1	
saddr.bit1[注1]	0nnnnnnH[注4].bit1	別モジュールから参照可能
式.bit1[注1]	0××××H[注3].bit1	

注1．bit1 = 0-7

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。

3．0×FF××Hはsfrのアドレス（LOCATION命令により異なります）

0××××Hは式の値

4．0nnnnnnHはsaddr領域

EQU　　equate　　EQU

【使 用 例】

```
        NAME    SAMP1
        LOCATION 0FH

WORK1   EQU     0FFE20H                 ; (1)
WORK10  EQU     WORK1.0                 ; (2)
P02     EQU     P0.2                    ; (3)
A4      EQU     A.4                     ; (4)
X5      EQU     X.5                     ; (5)
PSWL5   EQU     PSWL.5                  ; (6)
PSWH6   EQU     PSWH.6                  ; (7)

        MOV1    CY,WORK10               ; (8)
        MOV1    P0.2,CY                 ; (9)
        OR1     CY,A4                   ; (10)
        XOR1    CY,X5                   ; (11)
        SET1    PSWL5                   ; (12)
        CLR1    PSWH6                   ; (13)

        END
```

＜解　　説＞

(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) ‘ X.bit **’にあたるビット値‘** X.5 **’に，ネーム‘** X5 **’を割り当てます。**

(6) ‘ PSWL.bit **’にあたるビット値‘** PSWL.5 **’に，ネーム‘** PSWL5 **’を割り当てます。**

(7) ‘ PSWH.bit **’にあたるビット値‘** PSWH.6 **’に，ネーム‘** PSWH6 **’を割り当てます。**

(8) この記述は，「MOV1 CY, saddr.bit**」に該当します。**

(9) この記述は，「MOV1 sfr.bit, CY**」に該当します。**

(10) この記述は，「OR1 CY, A.bit**」に該当します。**

(11) この記述は，「XOR1 CY, X.bit**」に該当します。**

(12) この記述は，「SET1 PSWL.bit**」に該当します。**

(13) この記述は，「CLR1 PSWH.bit**」に該当します。**

EQU　　equate　　EQU

なお，（3），（4），（5），（6），（7）のように「sfr.bit」「A.bit」「X.bit」「PSWL.bit」「PSWH.bit」を定義したネームは，そのモジュール内でのみ参照できます。

「saddr.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できます（3. 5 (2) EXTBIT参照）。

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。

```
        Assemble list

 ALNO  STNO ADRS   OBJECT     M |  SOURCE STATEMENT

    1     1                            NAME    SAMP2
    2     2 000000  09C1FF00             LOCATION 0FH
    3     3
    4     4         (000FFE20)     WORK1   EQU     0FFE20H         ; (1)
    5     5         (0FFE20.0)     WORK10  EQU     WORK1.0         ; (2)
    6     6         (00FF00.2)     P02     EQU     P0.2            ; (3)
    7     7         (000001.4)     A4      EQU     A.4             ; (4)
    8     8         (000000.5)     X5      EQU     X.5             ; (5)
    9     9         (0001FE.5)     PSWL5   EQU     PSWL.5          ; (6)
   10    10         (0001FE.6)     PSWH6   EQU     PSWH.6          ; (7)
   11    11
   12    12 000004  3C080020             MOV1    CY,WORK10       ; (8)
   13    13 000008  081200               MOV1    P02,CY          ; (9)
   14    14 00000B  034C                 OR1     CY,A4           ; (10)
   15    15 00000D  0365                 XOR1    CY,X5           ; (11)
   16    16 00000F  0285                 SET1    PSWL5           ; (12)
   17    17 000011  0296                 CLR1    PSWH6           ; (13)
   18    18
   19    19                            END
```

＜解　　説＞

ビット値をネームとして定義している（2）-（7）の行には，アセンブル・リストのオブジェクト・コード欄には，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　　set　　SET

(2) SET（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でも記述できます。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，解析可能な限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。

SET　　set　　SET

【使 用 例】

```
        NAME        SAMP1
        LOCATION    0FH

COUNT   SET         10H                     ; (1)

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

COUNT   SET         20H                     ; (3)

        MOV         B,#COUNT                ; (4)
        END
```

＜解　　説＞

(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。

DB　　define byte　　DB

（1）DB（define byte）

【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
[レーベル：]	DB	{（サイズ） 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にはサイズ指定とみなし，そうでなければ初期値とみなします。

・DB疑似命令は，ビット・セグメント内では記述できません。

サイズ指定の場合：

・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。

・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・メッセージが出力され，初期化は行われません。

DB　　define byte　　DB

初期値指定の場合：

- **初期値として，次の2つが記述できます。**
  - **① 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。
  - **② 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。
- **初期値は，1行の範囲であれば，複数指定できます。**
- **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できます。**

【使 用 例】

```
        NAME       SAMP1
        LOCATION   0FH
        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)

        END
```

＜解　　説＞

(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　　define word　　DW

(2) DW（define word）

【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
[レーベル：]	DW	{（サイズ） 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にはサイズ指定とみなし，そうでなければ初期値とみなします。

・DW疑似命令は，ビット・セグメント内では記述できません。

サイズ指定の場合：

・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。

・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・メッセージが出力され，初期化は行われません。

DW　　define word　　DW

初期値指定の場合：

- **初期値として，次の2つが記述できます。**
  - **① 定数**
    16ビット以下の定数です。
  - **② 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。
- **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できます。**

【使 用 例】

```
        NAME        SAMP1
        LOCATION    0FH
        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
```

DW　　define word　　DW

＜解　　説＞

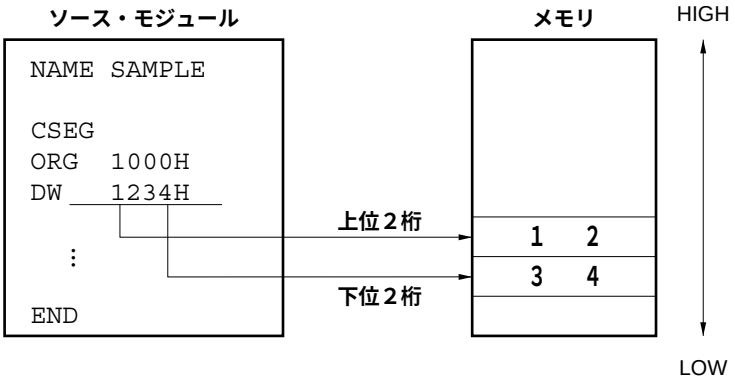
(1) サイズを指定しているので，それぞれのワード領域を値‘ 00H ’で初期化します。

(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　　dg　　DG

(3) DG（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
[レーベル：]	DG	{（サイズ） 初期値［，…］}	[；コメント]

【機　　能】

- 3バイト領域を初期化します。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3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24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にはサイズ指定とみなし，そうでなければ初期値とみなします。
- DG疑似命令は，ビット・セグメント内では記述できません。

サイズ指定の場合：

- 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×3バイト分の領域を‘00H’で初期化します。
- 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・メッセージが出力され，初期化は行われません。

DG　　dg　　DG

初期値指定の場合：

・初期値として，次の2つが記述できます。

① 定数

24ビット以下の定数です。

② 式

式の値は，24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。

・初期値の最上位1バイトがメモリのHIGH WORDアドレス[注]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バイトは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が記述できます。

注　HIGH WORDは，1バイトです。

【使 用 例】

```
        NAME    SAMP1
        LOCATION 0FH
DATA1:  DG      123456H,567890H             ; (1)
DATA2:  DG      (10)                        ; (2)

        END
```

＜解　　説＞

(1) 3バイトの領域を‘563412907856’の値で初期化します。

(2) 30バイト（10×3バイト）の領域が‘00H’で初期化されます。

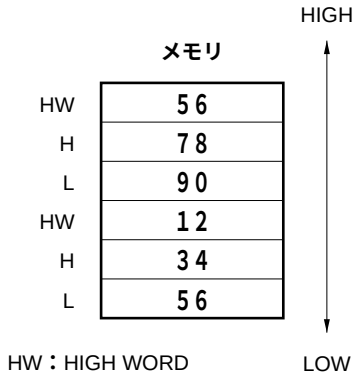
DG　　dg　　DG

注意　3**バイト値は，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      123456H,567890H
        ⋮
        END
```



HW：HIGH WORD
H：HIGH
L：LOW

DS　　define storage　　DS

(4) DS（define storage）

【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
[レーベル：]	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。
レーベルがある場合は，確保したメモリ領域の先頭アドレスの値をそのレーベルに割り付けます。
ソース・モジュールでは，このレー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ビットで評価します。
・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も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”中のレーベル1と“ADDRESS属性の式2”中のレーベル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) レーベル1とレーベル2が同一セグメントで，2つのレー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) レーベル1とレーベル2が別のセグメントで，属するセグメントの先頭からレー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

DS　　define storage　　DS

・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     1                     ; (2)
WORK2:  DS     2                     ; (3)
        CSEG
        MOVW   HL,#TABLE1
        MOV    A,!WORK1
        MOVW   BC,#WORK2
        END
```

＜解　　説＞

(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。レーベル‘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　　define bit　　DBIT

(5) DBIT（define bit）

【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。

【使 用 例】

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

        CSEG
        MOV1   CY, BIT1               ; (2)

        OR1    CY, BIT2               ; (3)

        END
```

＜解　　説＞

(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，「MOV1 CY, 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，「OR1 CY, 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，他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。このために，「使いたい」，「使ってもいいです」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，「他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，他のシンボルで参照してもいいよ」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。

- シンボルの外部定義宣言を行うもの：PUBLIC疑似命令
- 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

図3－6　2つのモジュール間のシンボルの関係

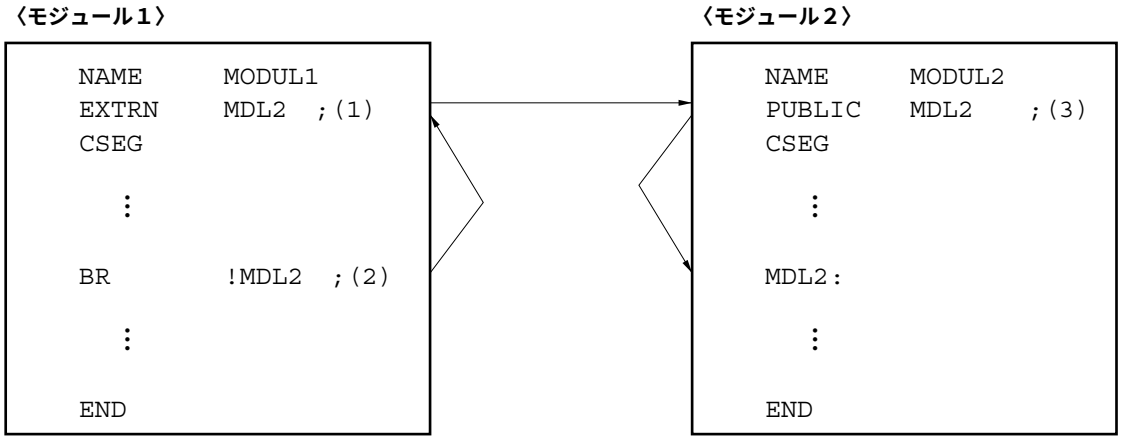


図3－6のモジュール1では，モジュール2の中で定義しているシンボル‘ MDL2 ’を（2）で参照しているため，（1）でEXTRN疑似命令により外部参照宣言を行っています。

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。

EXTRN　　external　　EXTRN

（1）EXTRN（external）

【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
[レーベル：]	EXTRN	{シンボル名[，…] SADDR2(シンボル名［，…］) BASE(シンボル名［，…］)}	[；コメント]

【機　　能】

・このモジュールで参照する他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】

・他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。

SADDR2（シンボル名［，…］）・・ saddr2領域として参照可能なシンボルとなる

BASE（シンボル名［，…］）・・・・・ 64 Kバイト内の（0H-0FFFFH内）領域のシンボルとして参照可能となる

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，参照可能となる

【説　　明】

・EXTRN疑似命令は，ソース・プログラム中のどこに記述してもかまいません（2.1　**ソース・プログラムの基本構成**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，他のモジュールで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・メッセージが出力されます。

・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できません。逆にEXTRN宣言したシンボルも他の疑似命令により再定義，宣言できません。

・saddr領域をEXTRN疑似命令で定義したシンボルで参照できます。

EXTRN　　external　　EXTRN

【使 用 例】

＜モジュール1＞

```
        NAME    SAMP1
        LOCATION 0FH
        EXTRN   SYM1,SYM2,SADDR2(SYM3),BASE(SYM4)    ; (1)
        CSEG
S1:     DW      SYM1                                 ; (2)
        MOV     A,SYM2                               ; (3)
        MOV     A,SYM3                               ; (4)
        BR      !SYM4                                ; (5)
        END
```

＜モジュール2＞

```
        NAME    SAMP2
        PUBLIC SYM1,SYM2,SYM3,SYM4                   ; (5)

        CSEG
SYM1    EQU     0FFH                                 ; (6)
DATA1   DSEG    SADDR
SYM2:   DB      012H                                 ; (7)
DATA2   DSEG    SADDR2
SYM3:   DB      034H                                 ; (8)
C1      CSEG    BASE
SYM4:   MOV     A, #20H                              ; (9)
        END
```

EXTRN　　external　　EXTRN

＜解　　説＞

(1) (2) と (3) で参照するシンボル‘ SYM1 ’，‘ SYM2 ’，‘ SYM3 ’，‘ SYM4 ’の外部参照宣言を行います。オペランド欄には，複数のシンボルが記述できます。

(2) シンボル‘ SYM1 ’を参照します。

(3) シンボル‘ SYM2 ’を参照します。saddr2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。64 K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 ’，‘ SYM4 ’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。

(9) シンボル‘ SYM4 ’を定義します。

EXTBIT　　external bit　　EXTBIT

(2) EXTBIT（external bit）

【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
[レーベル：]	EXTBIT	ビット・シンボル名[，…]	[；コメント]
		SADDR2(シンボル名 [，…])	
		SADDRA(シンボル名 [，…])	

【機　　能】

・本モジュールで参照する他のモジュールのsaddr.bitの値を持つビット・シンボルを宣言します。

【用　　途】

・他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できます。
・EXTBIT疑似命令で宣言されたシンボルは，他のモジュールで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。

EXTBIT　　external bit　　EXTBIT

【使 用 例】

＜モジュール1＞

```
        NAME    SAMP1
        EXTBIT  FLAG1,SADDR2(FLAG2,FLAG3),FLAG4      ; (1)
        CSEG
        MOV1    FLAG1,CY                             ; (2)
        AND1    CY,FLAG2                             ; (3)
        SET1    FLAG3                                ; (4)
        NOT1    FLAG4                                ; (5)
        END
```

＜モジュール2＞

```
        NAME    SAMP2
        PUBLIC  FLAG1,FLAG2,FLAG3,FLAG4              ; (6)
B1      BSEG    SADDR
FLAG1   DBIT                                         ; (7)
FLAG4   DBIT                                         ; (8)
B2      BSEG    SADDR2
FLAG2   DBIT                                         ; (9)
FLAG3   DBIT                                         ; (10)
        CSEG


        END
```

＜解　　説＞

(1) 参照するシンボル‘ FLAG1 **’，‘** FLAG2 **’，‘** FLAG3 **’，‘** FLAG4 **’の外部参照宣言を行います。オペランド欄には，複数のシンボルが記述できます。**

(2) シンボル‘ FLAG1 **’を参照します。この記述は，「**MOV1 saddr1. bit, CY**」に該当します。**

(3) シンボル‘ FLAG2 **’を参照します。この記述は，「**AND1 CY, saddr2. bit**」に該当します。**

(4) シンボル‘ FLAG3 **’を参照します。この記述は，「**SET1 saddr2. bit**」に該当します。**

(5) シンボル‘ FLAG4 **’を参照します。この記述は，「**NOT1 saddr1. bit**」に該当します。**

(6) シンボル‘ FLAG1 **’，‘** FLAG2 **’，‘** FLAG3 **’，‘** FLAG4 **’を定義します。**

(7) シンボル‘ FLAG1 **’を**SADDR1**領域のビット・シンボルとして定義します。**

(8) シンボル‘ FLAG4 **’を**SADDR1**領域のビット・シンボルとして定義します。**

(9) シンボル‘ FLAG2 **’を**SADDR2**領域のビット・シンボルとして定義します。**

(10) シンボル‘ FLAG3 **’を**SADDR2**領域のビット・シンボルとして定義します。**

PUBLIC　　public　　PUBLIC

(3) PUBLIC（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
[レーベル：]	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他のモジュールから参照できるよう宣言します。

【用　　途】

・他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宣言できます。2回目以降の宣言は無視されます。
・次のシンボルは，オペランドとして記述できません。
  ・SET疑似命令で定義したネーム
  ・同一モジュール内でEXTRN，EXTBIT疑似命令で定義したシンボル
  ・セグメント名
  ・モジュール名
  ・マクロ名
  ・モジュール内で定義されていないシンボル
  ・ビット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
  ・sfrをEQU疑似命令で定義したシンボル（ただし，sfr領域とsaddr領域のオーバラップしている箇所は除きます）。

PUBLIC　　public　　PUBLIC

【使 用 例】

3つのモジュールからなるプログラム例

＜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
        XOR1   CY, 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
        MOV1   CY, A2
        END
```

PUBLIC	public	PUBLIC

＜解　　説＞

(1) シンボルA1，A2が，他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，他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，他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。

NAME　　name　　NAME

（1）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
[レーベル：]	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，ディバッガによるシンボリック・ディバグ時に必要となります。

【説　　明】

・NAME疑似命令は，ソース・プログラム中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，シンボル記述上の規則2. 2. 3　**文の構成フィールド**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）と漢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8文字）がモジュール名になります。なお，Windows版では，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・メッセージが出力され，2回目以降の宣言を無視します。
・オペランド欄のモジュール名は8文字以内で指定してください。
・シンボル名の大文字，小文字を区別します。

NAME　　name　　NAME

【使 用 例】

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


        CSEG
        MOV    A,B


        END
```

＜解　　説＞

(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

無条件分岐命令で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 !addr16」，「BR $addr20」，「BR $!addr20」，「BR !!addr20」の4つがあります。また，CALL命令にも同様に「CALL !!addr20」，「CALL $!addr20」，「CALL !addr16」の3つがあります。これらの命令は，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しますが，命令のバイト数が異なるので，メモリ効率のよいプログラムを作成するためには，短いバイト数の命令を使用する必要がありますし，分岐命令を記述する際に，分岐先範囲も考慮する必要があります。これらのことを考慮してプログラムを作成するのは，かなり面倒です。

そこで，アセンブラ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。

BR　　branch　　BR

(1) BR (branch)

【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
[レーベル：]	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3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。

“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　!addr16”（3バイト）・・・・・・・ 分岐先が64 K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[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（BR　!addr16）に置き換えられます。

注　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，アセンブル時間が短縮できます。

BR　　branch　　BR

【説　　明】

・BR疑似命令は，コード・セグメント内でのみ使用できます。

・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‘ $ ’は記述できません。

・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 式中のレー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
000050H             BR      L1                      ; (1)
000052H             BR      L2                      ; (2)
000055H             BR      L3                      ; (3)
000058H             BR      L4                      ; (4)

00007DH     L1:
007FFFH     L2:
00FFFFH     L3:
010000H     L4:
                            END
```

＜解　　説＞

(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。

(3) このBR疑似命令は，分岐先が64 K以内なので3バイトの分岐命令（BR　!addr16）に置き換えられます。

(4) 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。

CALL　　call　　CALL

(2) CALL（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
[レーベル：]	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4バイトのCALL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。

“CALL　!addr16”（3バイト）　・・・・ 分岐先が64 K内の場合に使用可能
“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　!addr16）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。

CALL　　call　　CALL

【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 式中のレー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
000050H                 CALL    L1                      ; (1)
000053H                 CALL    L2                      ; (2)
000056H                 CALL    L3                      ; (3)

008052H       L1:
00FFFFH       L2:
010000H       L3:
                        END
```

＜解　　説＞

(1) 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。

(2) このCALL疑似命令は，分岐先が64 K以内なので3バイトの分岐命令（CALL　!addr16）に置き換えられます。

(3) 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. 8　汎用レジスタ選択疑似命令

78K/IVの汎用レジスタでは，PSW内のレジスタ・セット選択（RSS）フラグの値によって機能名称と絶対名称の対応が異なっています（表3－10　汎用レジスタの絶対名称と機能名称参照）。

これは，プログラム中で絶対名称の代わりに機能名称を記述する場合，RSSフラグの値によって実際にアクセスされるレジスタが異なり，生成されるオブジェクト・コードもRSSフラグの値により異なることを意味します。

汎用レジスタ選択疑似命令は，RSSフラグに設定されている値をアセンブラに知らせ，RSSフラグの値に対応したオブジェクト・コードを生成するためのものです。

表3－10　汎用レジスタの絶対名称と機能名称

(a) 8ビット・レジスタ

絶対名称	機能名称	
	RSS = 0	RSS = 1[注]
R0	X	
R1	A	
R2	C	
R3	B	
R4		X
R5		A
R6		C
R7		B
R8		
R9		
R10		
R11		
R12	E	E
R13	D	D
R14	L	L
R15	H	H

(b) 16ビット・レジスタ

絶対名称	機能名称	
	RSS = 0	RSS = 1[注]
RP0	AX	
RP1	BC	
RP2		AX
RP3		BC
RP4	VP	VP
RP5	UP	UP
RP6	DE	DE
RP7	HL	HL

(c) 24ビット・レジスタ

絶対名称	機能名称
RG4	VVP
RG5	UUP
RG6	TDE
RG7	WHL

注　RSS = 1として使用するのは，78K/IIIシリーズ用のプログラムを流用する場合だけにしてください。

備考1.　表中空欄の部分は，絶対名称を記述することにより，該当するレジスタをアクセスすることができます。

2.　R8-R11には機能名称はありません。

RSS　　register set select　　RSS

（1）RSS（register set select）

【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
[レーベル：]	RSS	評価値が0 or 1の絶対値	[；コメント]

【機　　能】

・オペランドで指定されたレジスタ・セット選択フラグの値をもとにして，プログラム中に記述された機能名称の汎用レジスタを対応する絶対名称の汎用レジスタに置き換えてオブジェクト・コードを生成します。汎用レジスタの機能名称と絶対名称の対応は，**表**3－10　**汎用レジスタの絶対名称と機能名称**を参照してください。

【用　　途】

・汎用レジスタの絶対名称の代わりに固有機能を活かした機能名称によるアドレシングを行う場合に，RSS疑似命令を使用します。
・汎用レジスタを機能名称で記述する場合，その時点でRSSフラグに設定されている値を必ずRSS疑似命令で宣言します。

【説　　明】

・レジスタ・セット選択（RSS）フラグは，PSWLのビット5です。

	7	6	5	4	3	2	1	0
PSWL	S	Z	RSS	AC	IE	P/V	0	CY

↑
RSSフラグ

・RSSフラグの値（0，1）をアセンブラに知らせるのがRSS疑似命令です。アセンブラはRSS疑似命令のオペランドの値により機能名称の汎用レジスタを絶対名称の汎用レジスタに置き換えてオブジェクト・コードを生成します。
・RSSフラグの値をインストラクションにより，セット，リセット，スイッチする場合には，そのインストラクションの直前または直後に必ずRSS疑似命令を記述し，アセンブラにRSSフラグの値を宣言します。インストラクションによりRSSフラグのセット，リセットを行っても，それにあわせてRSS疑似命令を記述しないと，期待するオブジェクト・コードが生成されません。
・RSS疑似命令は，次のRSS疑似命令，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有効です。したがって，RSS疑似命令は各セグメント単位で記述することが必要です。
・RSS疑似命令は，コード・セグメント内でのみ記述できます。
・セグメントが生成されていない状態でRSS疑似命令が出現した場合，アセンブラはその時点で，ディフォールト・セグメントとしてリロケータブルなコード・セグメントを生成します。このときのセグメント名は?CSEG，再配置属性はUNITです。

RSS　　register set select　　RSS

・RSS疑似命令のディフォールト値はRSS = 0です。

【使 用 例】

```
        NAME    SAMPLE
        LOCATION 0FH
        RSS     1                       ; (1)
        MOV     B,A                     ; (2)
SEG1    CSEG
SUB1:   MOV     B,A                     ; (3)
        MOV     A,C                     ; (4)
        RET

SEG2    CSEG
SUB2:   RSS     1                       ; (5)
        SET1    PSWL.5                  ; (6)
        MOV     B,A                     ; (7)
        RET
SUB3:   RSS     0                       ; (8)
        SWRS                            ; (9)
        MOV     B,A                     ; (10)
        RET
SUB4:   MOV     B,A                     ; (11)
        RET

SEG4    DSEG
VAR:    DW      0

SEG3    CSEG
SUB5:   MOV     B,A                     ; (12)
        RET
        END
```

RSS　　register set select　　RSS

＜解　　説＞

(1) セグメントが生成されます。

(2) この記述は，「MOV R7, R5」に該当します。

(3) RSSのアセンブラにおけるディフォールト値は‘ ０ ’です。RSS疑似命令の記述がないので，この記述は「MOV R3, R1」に該当します。

(4) この記述は，「MOV R1, R2」に該当します。

(5) (6) のRSSフラグをセットするインストラクションの直前（または直後）に，RSS疑似命令を記述します。

(7) この記述は，「MOV R7, R5」に該当します。

(8) (9) のRSSフラグをリセットするインストラクションの直前（または直後）に，RSS疑似命令を記述します。

(10) この記述は，「MOV R3, R1」に該当します。

(11) この記述は，「MOV R3, R1」に該当します。

(12) この記述は，「MOV R3, R1」に該当します。

RSS　　register set select　　RSS

使用例のアセンブル・リストで，生成されるオブジェクト・コードを次に示します。

```
     Assemble list

ALNO  STNO ADRS    OBJECT     M I  SOURCE STATEMENT

   1     1                              NAME    SAMPLE
   2     2 000000  09C1FF00             LOCATION 0FH
   3     3                              RSS     1             ;(1)
   4     4 000004  2475                 MOV     B,A           ;(2)
   5     5 ------                 SEG1  CSEG
   6     6 000000  2431           SUB1: MOV     B,A           ;(3)
   7     7 000002  D2                   MOV     A,C           ;(4)
   8     8 000003  56                   RET
   9     9
  10    10 ------                 SEG2  CSEG
  11    11 000000                 SUB2: RSS     1             ;(5)
  12    12 000000  0285                 SET1    PSWL.5        ;(6)
  13    13 000002  2475                 MOV     B,A           ;(7)
  14    14 000004  56                   RET
  15    15 000005                 SUB3: RSS     0             ;(8)
  16    16 000005  05FC                 SWRS                  ;(9)
  17    17 000007  2431                 MOV     B,A           ;(10)
  18    18 000009  56                   RET
  19    19 00000A  2431           SUB4: MOV     B,A           ;(11)
  20    20 00000C  56                   RET
  21    21
  22    22 ------                 SEG4  DSEG
  23    23 000000  0000           VAR:  DW      0
  24    24
  25    25 ------                 SEG3  CSEG
  26    26 000000  2431           SUB5: MOV     B,A           ;(12)
  27    27 000002  56                   RET
  28    28                              END
```

3.9　マクロ疑似命令

ソース・プログラムを記述する場合，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，MACRO，LOCAL，REPT，IRP，EXITMおよびENDMがあります。ここでは，マクロ疑似命令についての説明を行い，マクロの機能については，第5章　マクロで説明します。

MACRO　　macro　　MACRO

（1）MACRO（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	[仮パラメータ［，…］]	[；コメント]
	⋮		
	マクロ・ボディ		
	⋮		
	ENDM		[；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・ボディといいます）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・プログラム中で，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（マクロの参照）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，シンボル記述上の規則（2. 2. 3　**文の構成フィールド**）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（使用例参照）。
・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。
・仮パラメータが有効なのは，マクロ・ボディ内のみです。
・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／レーベルを，LOCAL疑似命令で宣言すれば，そのネーム／レーベルは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・ボディ内で他のマクロを参照すること）はREPT，IRP合わせて最大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。メモリが使える限り定義できます。
・クロスレファレンス・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は定義できません。定義された場合は，エラーを出力します。

MACRO　　macro　　MACRO

【使 用 例】

```
        NAME   SAMPLE

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

        ADMAC  10H,20H                  ; (3)

        END
```

＜解　　説＞

(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　　local　　LOCAL

(2) LOCAL（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，シンボル記述上の規則（2. 2. 3　**文の構成フィールド**）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“??RAn”（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??RAn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??RAnというシンボル名で参照できます。
・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エラー・メッセージを出力し，そのマクロ定義を空のマクロ・ボディとして登録します。参照された場合は，何も展開しません。
・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は記述できません。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，クロスレファレンス・リスト，シンボル・テーブル・リストには出力されません。

LOCAL　　local　　LOCAL

・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】

＜ソース・プログラム＞

```
        NAME    SAMPLE

MAC1    MACRO
        LOCAL   LLAB                    ; (1)
LLAB:
        BR      $LLAB                   ; (2)
        ENDM

REF1:   MAC1                            ; (3)

        BR      !LLAB                   ; (4)

REF2:   MAC1                            ; (5)

        END
```

マクロの定義

この記述は
エラーです。

＜解　　説＞

(1) シンボル名LLABをローカル・シンボルとして定義します。
(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。

```
    Assemble list

 ALNO  STNO ADRS   OBJECT   M | 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 F407, STNO  13 (   0) Undefined symbol reference 'LLAB'
***  ERROR F303, 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　　repeat　　REPT

(3) REPT (repeat)

【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
[レーベル：]	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・プログラム中で，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あわせて，ネスト・レベルの最大数8までネスティングできます。
・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できます。
・REPT-ENDMのブロック内にマクロ定義を記述できません。
・オペランド欄に記述する絶対式は，符号なし24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】

＜ソース・プログラム＞

```
        NAME    SAMP1
        CSEG
        REPT    3                       ; (1)
        INC     B
        DEC     C
        ENDM                            ; (2)
        END
```

← REPT-ENDMブロック

REPT　　repeat　　REPT

＜解　　説＞

(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　　indefinite repeat　　IRP

(4) IRP (indefinite repeat)

【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
[レーベル：]	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・プログラム中で，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個まで記述できます。
・IRP-ENDMブロック内では，マクロ参照，REPT，IRPを合わせたネスト・レベルの最大数8までネスティングできます。
・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できます。

【使 用 例】

＜ソース・プログラム＞

```
        NAME    SAMP1
        CSEG

        IRP     PARA,<0AH,0BH,0CH>      ; (1)
        ADD     A,#PARA
        MOV     [DE+],A                          ← IRP-ENDMブロック
        ENDM                            ; (2)

        END
```

IRP　　indefinite repeat　　IRP

＜解　　説＞

(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

        ADD     A,#0AH                  ; (3)
        MOV     [DE+],A
        ADD     A,#0BH                  ; (4)
        MOV     [DE+],A
        ADD     A,#0CH                  ; (5)
        MOV     [DE+],A

        END
```

(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　　exit from macro　　EXITM

(5) EXITM（exit from 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
[レーベル：]	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**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・メッセージを出力しますが，EXITMの処理は行います。
・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が現れた場合は，そのEXITM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】

・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。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。

EXITM　　exit from macro　　EXITM

＜ソース・プログラム＞

```
        NAME   SAMP1
MAC1    MACRO                          ; (1)
        NOT1   A.1
$       IF(SW1)                        ; (2)
        BT     A.1,$L1
        EXITM                          ; (3)
$       ELSE                           ; (4)
        MOV1   CY,A.1
        MOV    A,#0
$       ENDIF                          ; (5)
$       IF(SW2)                        ; (6)
        BR     [HL]
$       ELSE                           ; (7)
        BR     [DE]
$       ENDIF                          ; (8)
        ENDM                           ; (9)

        CSEG
$       SET(SW1)                       ; (10)
        MAC1                           ; (11)
        NOP
L1:     NOP
        END
```

マクロ・ボディ

IFブロック

ELSEブロック

IFブロック

ELSEブロック

マクロ参照

EXITM　　exit from macro　　EXITM

＜解　　説＞

(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。

このソースをアセンブルすると，次のようになります。

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

(6) ENDM（end 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
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
```

ENDM　　end macro　　ENDM

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. 10　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。**

END　　end　　END

(1) 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・メッセージが出力されます。

【使 用 例】

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

＜解　　説＞

(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　制御命令の概要

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・プログラム中に記述します。
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。
制御命令には，次の種類があります。

表4－1　制御命令一覧表

項番	制御命令の種類	制御命令
1	アセンブル対象品種指定制御命令	PROCESSOR
2	ディバグ情報出力制御命令	DEBUG/NODEBUG, DEBUGA/NODEBUGA
3	クロスレファレンス・リスト出力指定制御命令	XREF/NOXREF, SYMLIST/NOSYMLIST
4	インクルード制御命令	INCLUDE
5	アセンブル・リスト制御命令	EJECT, TITLE, SUBTITLE, LIST/NOLIST, GEN/NOGEN, COND/NOCOND, FORMFEED/NOFORMFEED, WIDTH, LENGTH, TAB
6	条件付きアセンブル制御命令	SET/RESET, IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF
7	SFR領域変更制御命令	CHGSFR/CHGSFRA
8	漢字コード指定制御命令	KANJICODE
9	その他	DGL, DGS, TOL_INF

制御命令は疑似命令と同様にソース・プログラム中に記述します。
また，**表4－1　制御命令一覧表**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できます。
制御命令とコマンド行のアセンブラ・オプションの対応を，**表4－2　制御命令とアセンブラ・オプション**に示します。

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
FORMFEED/NOFORMFEED	-LF/-NLF
TITLE	-LH
WIDTH	-LW
LENGTH	-LL
TAB	-LT
CHGSFR/CHGSFRA	-CS/-CSA
KANJICODE	-ZS/-ZE/-ZN

コマンド行での指定方法については，操作編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。

PROCESSOR　　processor　　PROCESSOR

(1) 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けます。

【説　　明】

・PROCESSOR制御命令は，モジュール・ヘッダ部にのみ記述できます。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・メッセージを出力しコマンド・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（4038）
$       DEBUG
$       XREF

        NAME      TEST

        CSEG
        ⋮
```

4.3　ディバグ情報出力制御命令

ディバ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ィバグ情報の出力を指定することができます。

DEBUG/NODEBUG　　debug/nodebug　　DEBUG/NODEBUG

（1）DEBUG/NODEBUG（debug/nodebug）

【記述形式】

```
〔△〕$〔△〕DEBUG                    ;省略時解釈
〔△〕$〔△〕NODEBUG
```

【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・ファイルに出力します。
- 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・ディバ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・ディバグ
  2. シンボルなしでのディバグ
  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。
- 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・ライン上の指定を優先します。
- 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。

(2) DEBUGA/NODEBUGA（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・ディバ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・ディバグ情報を付加しません。

【用　　途】

- DEBUGA制御命令は，アセンブラまたは構造化アセンブラのソース・レベルでディバグするときに使用します。なお，ソース・レベルでのディバグには，“統合ディバッガ”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・ソースなしでのディバグ
  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，後者優先とします。
- アセンブラ・ソース・ディバ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・ライン上の指定を優先します。
- 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，構造化アセンブラ・プリプロセッサでディバグ情報を出力して，コンパイル，構造化アセンブルした場合，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ィバ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，構造化アセンブラ・プリプロセッサが，アセンブラ・ソース中に制御文として出力します。

4.4　クロスレファレンス・リスト出力指定制御命令

クロスレファレンス・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レファレンス・リストの出力指定を行えます。

（1）XREF/NOXREF（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・ファイルにクロスレファレンス・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・ファイルにクロスレファレンス・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レファレンス・リストを出力します。
- アセンブルのたびにクロスレファレンス・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けます。

【説　　明】

- XREF/NOXREF制御命令は，モジュール・ヘッダ部のみに記述できます。
- XREF/NOXREF制御命令が複数指定された場合には，後者優先です。
- クロスレファレンス・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　　symlist/nosymlist　　SYMLIST/NOSYMLIST

(2) SYMLIST/NOSYMLIST（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，後者優先です。
- 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・ライン上の指定を優先します。
- 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・モジュール中で他のソース・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・プログラムの記述の手間を軽減できます。

INCLUDE　　include　　INCLUDE

(1) 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できます。

【説　　明】

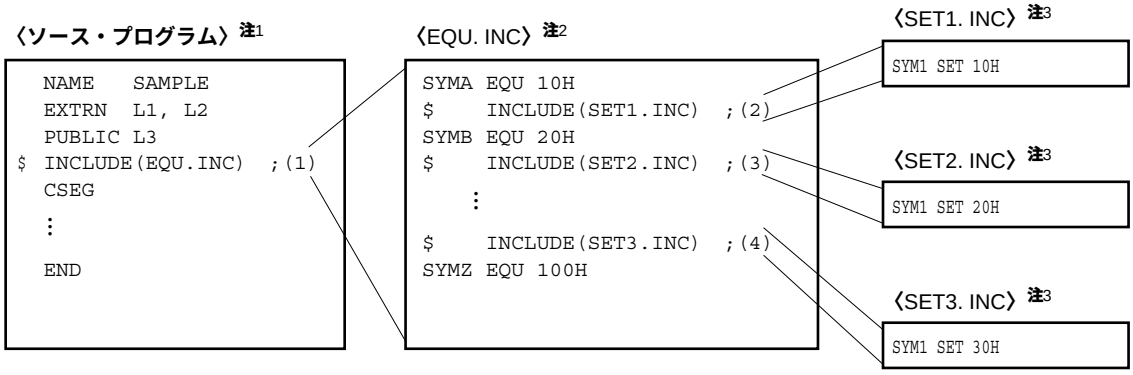
・INCLUDE制御命令は，通常のソースにのみ記述できます。
・アセンブラ・オプション（-I）でインクルード・ファイルのパス名やドライブ名を指定できます。
・インクルード・ファイルの読み込みパスのサーチの順番は次のとおりです。
(a) インクルード・ファイルがパス名なしで指定された場合
① ソース・ファイルのあるパス
② アセンブラ・オプション“-I”で指定されたパス
③ 環境変数INC78K4で指定されたパス
(b) インクルード・ファイルがパス名付きで指定された場合
ドライブ名または￥から始まるパス名付きで指定した場合には，インクルード・ファイル名に付いているパス，相対パス（先頭に￥がない）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（a）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。

INCLUDE　　include　　INCLUDE

・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・メッセージを出力し，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が現れた場合には，エラーが出力され，そのENDM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がない場合には，エラーが出力され，ENDMがあるものとして処理されます。

【使 用 例】



注1. ソース・ファイル中には，$ICを複数指定できます。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。

2. EQU. INCには，$ICを複数指定できます。

3. SET1. INC，SET2. INC，およびSET3. INC中には，$ICを指定できません。

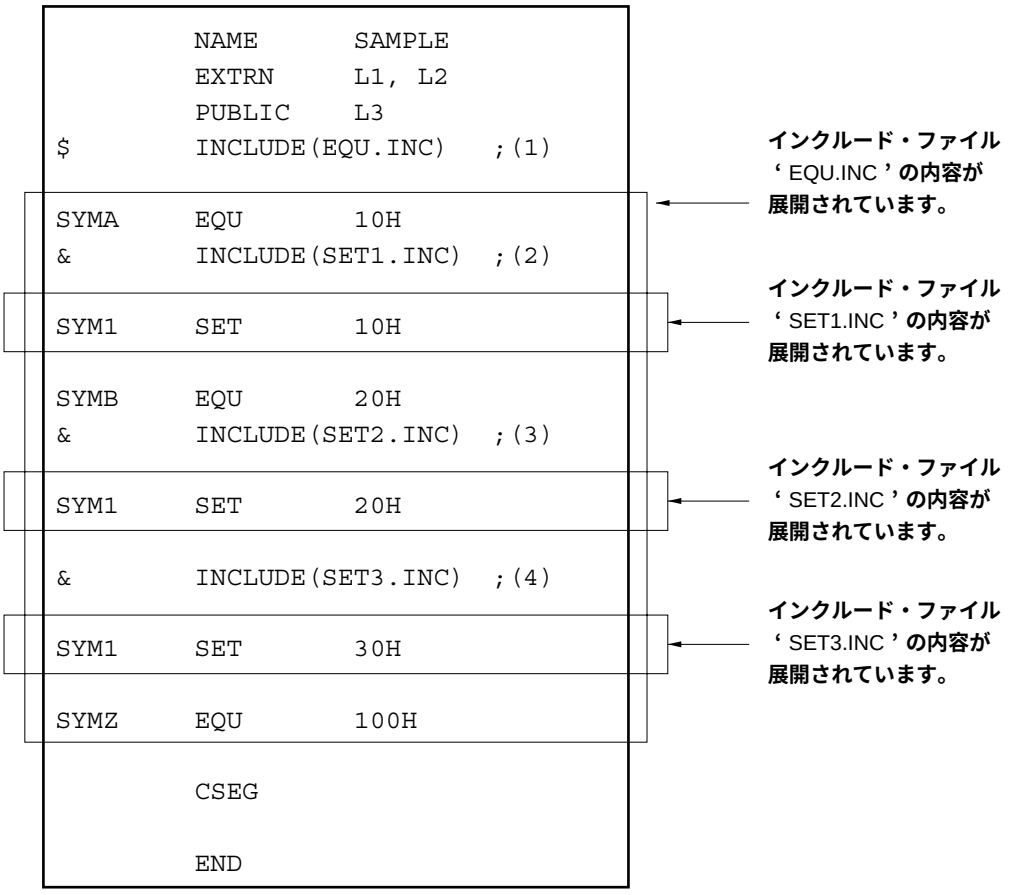
＜解　　説＞

(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)，(3)，(4) インクルード・ファイルとして‘ SET1. INC ’,‘ SET2. INC ’,‘ SET3. INC ’をそれぞれ指定しています。

このソース・プログラム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。

INCLUDE　　include　　INCLUDE



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　　eject　　EJECT

（1）EJECT（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”“-LLO”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。

-NP，-LLオプションについては，**操作編**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・メッセージを出力します。

【使 用 例】

ソース・モジュール

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

EJECT　　eject　　EJECT

＜解　　説＞

（1）EJECT制御命令により改ページを行い，アセンブル・リストは次のようになります。

```
        ⋮
        MOV     〔DE+〕,A
        BR      $$
$       EJECT                                          改ページ
        CSEG
        ⋮
        END
```

(2) LIST/NOLIST (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います。
- 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います。
  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
DATA2       EQU         11H
            ⋮
DATAX       EQU         20H
DATAY       EQU         20H
$           LIST                    ;(2)
            CSEG
            ⋮
            END
```

↑
アセンブル・リストに出力されません。
↓

＜解　　説＞

（1）NOLIST制御命令を指定していますので，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ますので，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　　generate/no generate　　GEN/NOGEN

(3) GEN/NOGEN（generate/no generate）

【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】

＜ソース・プログラム＞

```
        NAME    SAMP
$       NOGEN
ADMAC   MACRO   PARA1, PARA2
        MOV     A,#PARA1
        ADD     A,#PARA2
        ENDM
        CSEG
        ADMAC   10H, 20H
        END
```

アセンブル・リストは次のようになります。

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

        MOV     A,#10H
        AUD     A,#20H

        END
```

マクロ展開部は出力されません

＜解　　説＞

（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。

（4）COND/NOCOND（condition/no condition）

【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。

COND/NOCOND　　condition/no condition　　COND/NOCOND

【使 用 例】

〈ソース・プログラム〉

```
            NAME        SAMP

$           NOCOND
$           SET(SW1)

$           IF(SW1)

            MOV         A,#1H

$           ELSE
            MOV         A,#0H
            ENDIF

            END
```

この部分は，アセンブルしてもリストには，出力されません。

TITLE　　title　　TITLE

(5) TITLE (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レファレンス・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けます。

【説　　明】

- TITLE制御命令は，モジュール・ヘッダ部のみに記述できます。
- 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です。
- 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します。
- 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）によって指定できます。

TITLE　　title　　TITLE

【使 用 例】

<ソース・モジュール>

```
$       PROCESSOR (4038)
$       TITLE ('THIS IS TITLE')
        NAME    SAMPLE
$       EJECT
        CSEG
        END
```

アセンブル・リストは次ページのようになります（行数は72行と指定しています）。

TITLE　　title　　TITLE

```
78K/IV Series Assembler Vx.xx    THIS IS TITLE    Date:xx xxx xxxx  Page: 1

Command:  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 (4038)
    2     2                       $          TITLE ( ' THIS IS TITLE ' )
    3     3
    4     4                                  NAME     SAMP
    5     5
    6     6                       $          EJECT

78K/IV Series Assembler Vx.xx    THIS IS TITLE     Date:xx xxx xxxx  Page: 2

 ALNO  STNO ADRS  OBJECT          M I  SOURCE STATEMENT

    7     7 ------                          CSEG
    8     8
    9     9                                 END
   10    10

 Target chip : uPD784038
 Device file : Vx.xx
Assembly complete,     0 error (s)  and     0 warning (s)  found. (    0)
```

(6) SUBTITLE（subtitle）

【記述形式】

```
〔△〕$〔△〕SUBTITLE〔△〕（〔△〕,'文字列'〔△〕）
〔△〕$〔△〕ST〔△〕（〔△〕'文字列'〔△〕）　　;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できます。

【説　　明】

・SUBTITLE制御命令は，通常のソースにのみに記述できます。
・指定できる文字列は，72文字以内です。
　73文字以上記述された場合，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）が記述された場合には，エラーとし，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。

SUBTITLE　　subtitle　　SUBTITLE

【使 用 例】

＜ソース・モジュール＞

```
        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
```

＜解　　説＞

(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。**

SUBTITLE　　subtitle　　SUBTITLE

例のアセンブル・リストは，次のようになります（行数は80行です）。

```
78K/IV Series Assembler Vx.xx                    Date:xx xxx xxxx  Page: 1

      Assemble list

 ALNO  STNO ADRS  OBJECT     M I  SOURCE STATEMENT

    1     1                          NAME SAMP
    2     2
    3     3 ------                   CSEG
    4     4
    5     5                    $     SUBTITLE ( ' THIS IS SUBTITLE 1')  ; (1)
    6     6                    $     EJECT                            ; (2)
-------------------------------------------------------------------------------
78K/IV Series Assembler Vx.xx                    Date:xx xxx xxxx  Page: 2

THIS IS SUBTITLE 1

 ALNO  STNO ADRS  OBJECT     M I  SOURCE STATEMENT

    7     7
    8     8 ------                   CSEG
    9     9
   10    10                    $     SUBTITLE ( ' THIS IS SUBTITLE 2')  ; (3)
   11    11                    $     EJECT                            ; (4)
-------------------------------------------------------------------------------
78K/IV Series Assembler Vx.xx                    Date:xx xxx xxxx  Page: 3

THIS IS SUBTITLE 2

 ALNO  STNO ADRS  OBJECT     M I  SOURCE STATEMENT

   12    12
   13    13                    $     SUBTITLE ( ' THIS IS SUBTITLE 3')  ; (5)
   14    14                          END

 Target chip : uPD784038
 Device file : Vx.xx
Assembly complete,     0 error (s)  and     0 warning (s)  found. (    0)
```

FORMFEED/NOFORMFEED　formfeed/noformfeed　FORMFEED/NOFORMFEED

（7）FORMFEED/NOFORMFEED（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）がディフォールトで設定されます。
- FORMFEED/NOFORMFEED制御命令が複数指定された場合は，最後に指定した命令を有効とします。
- 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・ライン上の指定を優先します。
- 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　　width　　WIDTH

(8) WIDTH (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，72-250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を有効とします。
- 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・ライン上の指定を優先します。
- 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　　length　　LENGTH

(9) LENGTH (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を有効とします。
・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・ライン上の指定を優先します。
・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　　tab　　TAB

（10）TAB（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を有効とします。
- 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・ライン上の指定を優先します。
- 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か，しないか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には，IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF制御命令，およびSET/RESET制御命令があります。

これらの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ができます。

IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF　　IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF

(1) 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（**0000H**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（0000H**）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。**

IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF　　IF/_IF/ELSEIF/_ELSEIF/ELSE/ENDIF

- 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ィバグ文などを記述した場合，そのディバグ文を機械語に変換するか，しないかを条件付きアセンブルのスイッチ設定により選択できます。

【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　**文の構成フィールド**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8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（4. 7（2）SET/RESET参照）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・メッセージを出力し，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。この場合には，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いいます（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
```

＜解　　説＞

(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
```

＜解　　説＞

(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)            ; (1)
        text1
$       ELSEIF (SW2)        ; (2)
        text2
$       ELSEIF (SW3)        ; (3)
        text3
$       ELSE                ; (4)
        text4
$       ENDIF               ; (5)
          ⋮
        END
```

＜解　説＞

(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’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’の値が偽の場合，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’の値が偽で，かつスイッチ名‘SW2’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’と，(2) のスイッチ名‘SW2’の値がともに偽で，かつスイッチ名‘SW3’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’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(SWA=SWB)          ;(1)
        text1
$       ENDIF                ;(2)
          ⋮
        END
```

＜解　　説＞

(1) スイッチ名‘ SWA **’，‘** SWB **’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‘ SWA **’または‘** SWB **’のいずれかの値が真の場合，**text1**の部分がアセンブルされます。**

スイッチ名‘ SWA **’，‘** SWB **’ともに偽の場合，**text1**の部分はアセンブルされません。**

(2) 条件付きアセンブル範囲の終了を示します。

SET/RESET　　set/reset　　SET/RESET

(2) SET/RESET（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（00FFH）を与えます。
- RESET制御命令で指定したスイッチ名に，偽の値（0000H）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（00FFH）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（0000H）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。
  スイッチ名記述上の規則は，シンボル記述上の規則（2. 2. 3　**文の構成フィールド**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できます。また，一度RESETしたスイッチ名をSET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，クロスレファレンス・リストには出力されません。

SET/RESET　　set/reset　　SET/RESET

【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
```

＜解　　説＞

（1）スイッチ名‘ SW1 **’に真の値（**00FFH**）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（**0000H**）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　SFR領域変更制御命令

ユーザの希望によりチップを発注する際に，SFR領域を含むSFR周辺の領域（リロケータブル空間）が変更可能です。SFR領域変更制御命令は，その変更希望をアセンブラ，リンカでサポートするための制御命令です。

CHGSFR/CHGSFRA　CHanGe SFR area/CHanGe SFR area　CHGSFR/CHGSFRA

（1）CHGSFR/CHGSFRA（change sfr area/change sfr area）

【記述形式】

```
〔△〕$〔△〕CHGSFR（〔△〕絶対式n〔△〕）
〔△〕$〔△〕CHGSFRA
```

【省略時解釈】

・$CHGSFR（0FH）

【機　　能】

・CHGSFR制御命令は，SFR領域のアドレスを指定します。
　オペランドに0を指定した場合　　：0FD00H-0FFFFH
　オペランドに0FHを指定した場合　：0FFD00H-0FFFFFH
・CHGSFRA制御命令は，LOCATION命令の記述がないことおよびアブソリュート・セグメントの配置アドレスが，設定可能なすべてのSFR領域内にないことをチェックするように指示します。
　この制御命令は，使用しないでください。

【用　　途】

・SFR領域を変更したいときに記述します。

【説　　明】

・CHGSFR/CHGSFRA制御命令は，モジュール・ヘッダ部にのみに記述可能です。
・CHGSFR/CHGSFRA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を優先とします。
・SFR領域変更は，コマンド・ライン上でアセンブラ・オプション（-CS/-CSA）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。
・リンク時に，CHGSFR制御命令または-CSオプションを指定しているすべてのモジュールでの指定値は同一でなければ行けません。
　また，その値は，LOCATION命令での指定値と同一でなければなりません。
・アセンブラ・オプション（-NO）を指定した場合でも，CHGSFR/CHGSFRA制御命令の文法チェックは行われます。

4.9　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。

KANJICODE　　kanjicode　　KANJICODE

（1）KANJICODE（kanjicode）

【記述形式】

```
〔△〕$〔△〕KANJICODE〔△〕漢字コード
```

【省略時解釈】

・OSにより次のように解釈します。

Windows/HP-UX	：$KANJICODE SJIS
Sun OS	：$KANJICODE EUC

【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が記述できます。

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を優先とします。

・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. 10　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。ソース・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　マクロの概要

ソース・プログラム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・プログラム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）を記述するだけですみます。
  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できます。
- また，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]**

【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・ボディといいます）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
```

＜解　　説＞

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. 9 **マクロ疑似命令（**1**）**MACRO**（**macro**）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・プログラム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
[レーベル：]	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・メッセージが出力されます。

【使 用 例】

```
        NAME    SAMPLE
ADMAC   MACRO   PARA1, PARA2
        MOV     A, #PARA1
        ADD     A, #PARA2

        CSEG
        ⋮
        ADMAC   10H, 20H
        ⋮
        END
```

＜解　　説＞

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，他のステートメントと同様にアセンブルします。

【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
ADMAC   MACRO   PARA1, PARA2
        MOV     A, #PARA1
        ADD     A, #PARA2
        ENDM

        CSEG
        ⋮
        ADMAC   10H, 20H            ; (1)
        MOV     A, PARA1 10H
        ADD     A, PARA2 20H
        ⋮
        END
```

マクロ定義

マクロの展開

＜解　　説＞

(1) の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・プログラム内のすべてのステートメントから参照できます。
  したがって，そのシンボルを定義しているマクロを2回以上参照して，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. 9　**マクロ疑似命令（2）**LOCAL（local）参照）。
- 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できます。
- マクロ外からローカル・シンボルを参照できません。

【使 用 例】

<ソース・プログラム>

```
        NAME    SAMPLE
MAC1    MACRO
        LOCAL   LLAB                ; (1)
LLAB:
        :
GLAB:
        BR      LLAB                ; (2)
        BR      GLAB                ; (3)
        ENDM
        :
REF1:   MAC1                        ; (4)
        :
        BR      LLAB                ; (5)
        BR      GLAB                ; (6)
        :
REF2:   MAC1                        ; (7)
        :
        END
```

マクロの定義

マクロの参照

この記述は
エラーです。

マクロの参照

<解　説>

(1) **レーベル**LLAB**をローカル・シンボルとして定義します。**

(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**内でグローバル・シンボル**GLAB**を参照しています。**

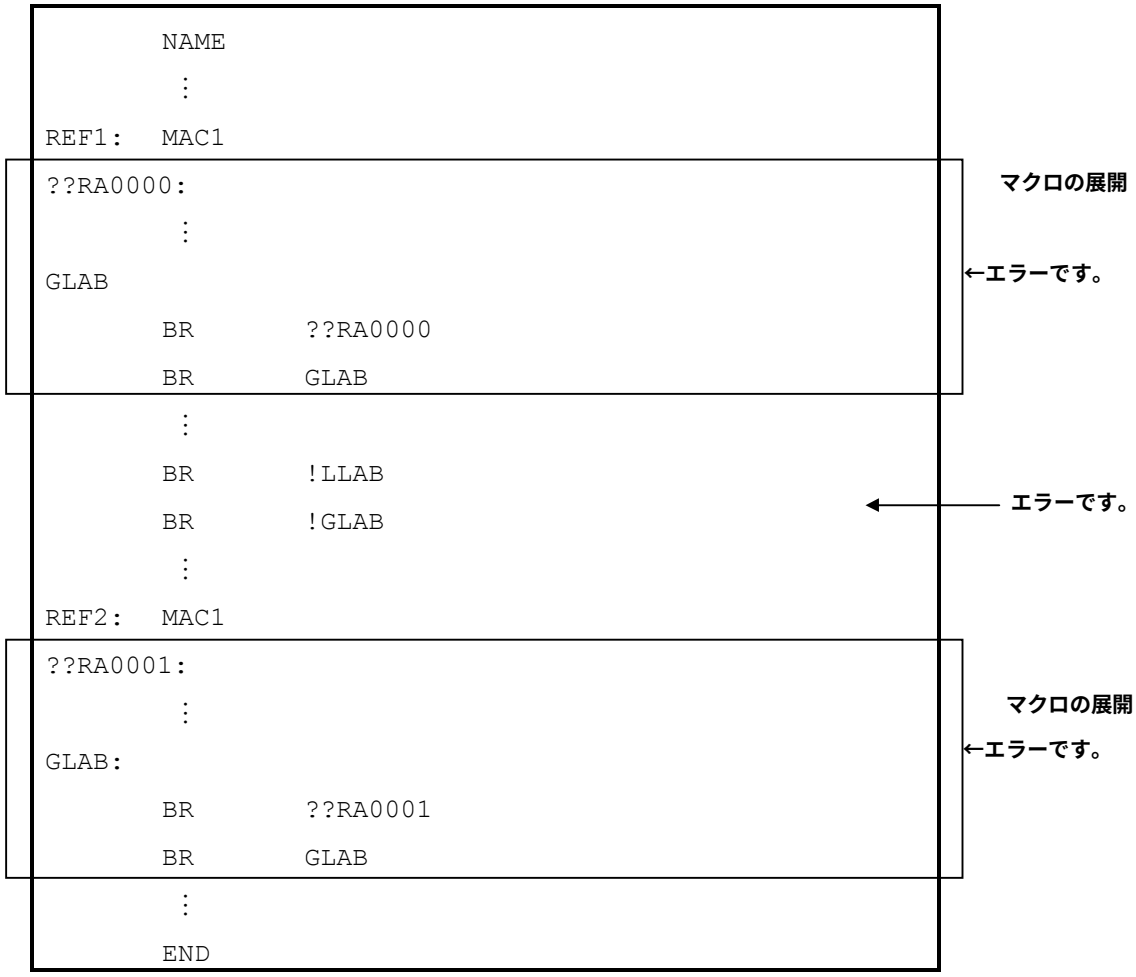
(4) **マクロ**MAC1**を参照しています。**

(5) **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。**

(6) **マクロ**MAC1**の定義外でグローバル・シンボル**GLAB**を参照しています。**

(7) **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**

使用例のソース・プログラムをアセンブルすると，次のように展開されます。



＜解　　説＞

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後に，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・クォートがはずされて渡されます。

・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      Z,<P>
        MOV      A, #Z
        ENDM
        ENDM

        MAC1     ‘10, 20, 30’
```

アセンブルするとMAC1は次のように展開されます。

```
        IRP      Z,<10, 20, 30>
        MOV      A, #Z
        ENDM
        MOV      A, #10
        MOV      A, #20
        MOV      A, #30
```

IRPの展開

第6章　製品活用法

この章では，RA78K4を利用してアセンブル作業を行う上で有効な活用法をいくつか紹介します。
RA78K4を利用してアセンブル作業を行う上で有効な活用法が，いろいろとあります。
ここでは，そのうちのいくつかを紹介します。

(1) アセンブラ起動時の手間を省く方法

アセンブラ起動時に必ず指定するアセンブル対象品種指定（-C）や漢字コード指定（-ZS/-ZE/-ZN）は，制御命令としてソース・モジュール中に記述しておくと便利です。特にアセンブル対象品種指定は省略不可能なのでソース・モジュール・ヘッダで指定しておけば，アセンブラ・プログラムの起動時にいちいちコマンド行で指定する手間が省けます。コマンド行で指定を忘れるとエラーとなり，やり直しをしなければなりません。

ほかに，クロスレファレンス・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（4038）
$     DEBUG
$     XRFF

      NAME              TEST

      CSEG
       ⋮
```

(2) メモリ効率のよいプログラムを開発する方法

ショート・ダイレクト・アドレシング領域は，ほかのデータ・メモリにくらべ短いバイト数の命令でアクセスできる領域です。

したがって，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。

そこでショート・ダイレクト・アドレシング領域は1モジュールで宣言するようにしておくと，ショート・ダイレクト・アドレシング領域に配置しようと思った変数がショート・ダイレクト・アドレシング領域に入りきれなかった場合，アクセス頻度の高い変数だけをショート・ダイレクト・アドレシング領域に配置する変更が容易になります。

モジュール1

```
        PUBLIC      TMP1, TMP2
WORK    DSEG        AT     0FE20H
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シンボルの6種類があります。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できません。

ソース・プログラムの各欄に記述できる予約語の種類と予約語一覧を次に示します。

シンボル欄	すべての予約語が記述できません。
ニモニック欄	機械語命令および疑似命令のみ記述できます。
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できます。
コメント欄	すべての予約語が記述できます。

sfr一覧は，各デバイスの**特殊機能レジスタ活用表**を参照してください。

割り込み要求ソース一覧は，各デバイス・ファイルの**使用上の留意点**を参照してください。

機械語命令，レジスタ名一覧は，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。

(1) 予約語一覧

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	BR	BSEG	CALL
	CALLT0	CSEG	DB	DBIT	DG
	DS	DSEG	DTABLE	DTABLEP	DW
	END	ENDM	EQU	EXITM	EXTBIT
	EXTRN	FIXED	FIXEDA	GRAM	IRP
	LOCAL	LRAM	MACRO	NAME	ORG
	PAGE	PAGE64K	PUBLIC	REPT	RSS
	SADDR	SADDR2	SADDRA	SADDRP	SADDRP2
	SET	UNIT	UNITP	SFR	SFRP
制御命令	CHGSFR	CHGSFRA	COND	DEBUG	DEBUGA
	EJ	EJECT	ELSE	ELSEIF	_ELSEIF
	ENDIF	FORMFEED	GEN	IC	IF
	_IF	INCLUDE	KANJICODE	LENGTH	LI
	LIST	NOCOND	NODEBUG	NODEBUGA	NOFORMFEED
	NOGEN	NOLI	NOLIST	NOSYMLIST	NOXR
	NOXREF	PC	PROCESSOR	RESET	SET
	ST	SUBTITLE	SYMLIST	TAB	TITLE
	TT	WIDTH	XR	XREF	
その他	DGL	DGS	TOL_INF		

付録B　疑似命令一覧

(1) 疑似命令一覧

項番	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		
1	[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	
2	[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	
3	[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	
4	[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
5	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
6	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
7	[レーベル：]	DB	{（サイズ）初期値［，…］}	[；コメント]	バイト・データ領域の確保／初期化	レーベル：シンボル 初期値の代わりに文字列を置くことができる
8	[レーベル：]	DW	{（サイズ）初期値［，…］}	[；コメント]	ワード・データ領域の確保／初期化	レーベル：シンボル
9	[レーベル：]	DG	{（サイズ）初期値［，…］}	[；コメント]	3バイト・データ領域の確保／初期化	レーベル：シンボル
10	[レーベル：]	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
11	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
12	[レーベル：]	PUBLIC	シンボル名［，…］	[；コメント]	外部定義名の宣言	
13	[レーベル：]	EXTRN	シンボル名［，…］	[；コメント]	外部参照名の宣言	
14	[レーベル：]	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。
15	[レーベル：]	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
16	[レーベル：]	BR	式	[；コメント]	分岐命令の自動選択	レーベル：シンボル

項番	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		
17	[レーベル：]	CALL	式	[；コメント]	CALL命令の自動選択	レーベル：シンボル
18	[レーベル：]	RSS	n	[；コメント]	レジスタ・セット選択フラグの値の宣言	n = 0, 1
19	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
20	[レーベル：]	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
21	[レーベル：]	REPT	絶対式	[；コメント]	マクロ展開時の繰り返しの指定	レーベル：シンボル
22	[レーベル：]	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	レーベル：シンボル
23	[レーベル：]	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
24	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
25	なし	END	なし	[；コメント]	ソース・モジュールの終了	

付録C　最大性能一覧

(1) アセンブラの最大性能

項　　目	最大性能	
	PC版	WS版
シンボル数（ローカル＋パブリック）	65535個[注1]	65535個[注1]
クロスレファレンス・リスト出力可能なシンボル数	65534個[注2]	65534個[注2]
1マクロ参照のマクロ・ボディ最大サイズ	1 Mバイト	1 Mバイト
全マクロ・ボディ合計のサイズ	10 Mバイト	10 Mバイト
1ファイル中のセグメント数	256個	256個
1ファイル中のマクロ，インクルード指定	10000個	10000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	10000個	10000個
リロケーション情報[注3]	65535個	65535個
行番号情報	65535個	65535個
1ファイル中のBR疑似命令数	32767個	32767個
1行の文字数	2048文字[注4]	2048文字[注4]
シンボル長	256文字	256文字
スイッチ名の定義数[注5]	1000個	1000個
スイッチ名の文字長[注5]	31文字	31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	8レベル

注1. XMSを使用します。XMSがなければファイルを使用します。

2. 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。
3. リロケーション情報とは，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。
4. 復帰／改行コードを含みます。1行に2049文字以上記述された場合，ワーニング・メッセージを出力し，2049文字以降は無視します。
5. 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。

(2) リンカの最大性能

項　　目	最大性能	
	PC版	WS版
シンボル数（ローカル＋パブリック）	65535個	65535個
同一セグメントの行番号情報	65535個	65535個
セグメント数	65535個	65535個
入力モジュール数	1024個	1024個

付録D　索　　引

D. 1　索引（英数字）

D. 2 索引（50音順）

〔メ　モ〕

お問い合わせ先

【技術的なお問い合わせ先】

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

電　話　：044−435−9494
FAX　：044−435−9608
E-mail　：info@lsi.nec.co.jp

【営業関係お問い合わせ先】

第一販売事業部

東　京　(03)3798−6106, 6107, 6108
大　阪　(06)6945−3178, 3200, 3208, 3212
広　島　(082)242−5504
仙　台　(022)267−8740
郡　山　(024)923−5591
千　葉　(043)238−8116

第二販売事業部

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

第三販売事業部

東　京　(03)3798−6151, 6155, 6586, 1622, 1623, 6156
水　戸　(029)226−1702
前　橋　(027)243−6060
鳥　取　(0857)27−5313
太　田　(0276)46−4014
名古屋　(052)222−2170, 2190
福　岡　(092)261−2806

【資料の請求先】

上記営業関係お問い合わせ先またはNEC特約店へお申しつけください。

【NECエレクトロンデバイス　ホームページ】

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

アンケート記入のお願い

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

［ドキュメント名］RA78K4　Ver.1.50以上　ユーザーズ・マニュアル　言語編
（U15255JJ1V0UM00（第1版））

［お名前など］（さしつかえのない範囲で）

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

1．ご評価（各欄に○をご記入ください）

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

2．わかりやすい所（第　章，第　章，第　章，第　章，その他　　）
理由［　　］

3．わかりにくい所（第　章，第　章，第　章，第　章，その他　　）
理由［　　］

4．ご意見，ご要望

5．このドキュメントをお届けしたのは
NEC販売員，特約店販売員，その他（　　）

ご協力ありがとうございました。
下記あてにFAXで送信いただくか，最寄りの販売員にコピーをお渡しください。

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

2000.6

キリトリ