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。

お客様各位

資料中の「日立製作所」、「日立XX」等名称の株式会社ルネサス テクノロジへの変更について

2003年4月1日を以って三菱電機株式会社及び株式会社日立製作所のマイコン、ロジック、アナログ、ディスクリート半導体、及びDRAMを除くメモリ（フラッシュメモリ・SRAM等）を含む半導体事業は株式会社ルネサス テクノロジに承継されました。従いまして、本資料中には「日立製作所」、「株式会社日立製作所」、「日立半導体」、「日立XX」といった表記が残っておりますが、これらの表記は全て「株式会社ルネサス テクノロジ」に変更されておりますのでご理解の程お願い致します。尚、会社商標・ロゴ・コーポレートステートメント以外の内容については一切変更しておりませんので資料としての内容更新ではありません。

ルネサステクノロジ ホームページ（http://www.renesas.com）

2003年4月1日
株式会社ルネサス テクノロジ
カスタマサポート部

RENESAS

ご注意

安全設計に関するお願い

1. 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ (http://www.renesas.com)などを通じて公開される情報に常にご注意ください。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任は負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。

User's Manual

H8S, H8/300 シリーズ クロスアセンブラ

ユーザーズマニュアル

ルネサスマイクロコンピュータ開発環境システム

HSS008ASCS2S

ルネサス エレクトロニクス
www.renesas.com

Rev.6.00　1999.01

はじめに

本マニュアルは、H8S, H8/300 シリーズ　クロスアセンブラについて説明したものです。

本マニュアルは、プログラミングの方法を説明した**「言語編」**、アセンブラの使い方を説明した**「操作編」**、および**「付録」**から構成されています。1 章～9 章が言語編、10 章～13 章が操作編です。

また、H8S, H8/300 シリーズ　クロスアセンブラ（Ver.2）からご使用になっている場合は、**付録 D** に旧バージョンとの相違点をまとめてありますので参照してください。

プログラムの開発にあたっては、次のマニュアルを合わせて参照してください。

実行命令の詳細について

- H8S/2600 シリーズ, H8S/2000 シリーズ　プログラミングマニュアル
- H8/300H シリーズ　プログラミングマニュアル
- H8/300 シリーズ　プログラミングマニュアル
- H8/300L シリーズ　プログラミングマニュアル

関連するソフトウェアについて

- H8S, H8/300 シリーズ　C/C++コンパイラ　ユーザーズマニュアル
- H シリーズ　リンケージエディタ、ライブラリアン、オブジェクトコンバータ　ユーザーズマニュアル
- H8S, H8/300 シリーズ　シミュレータ・デバッガ　ユーザーズマニュアル

【注】 MS-DOS **は、米国マイクロソフト社により管理されているオペレーティングシステムの名称です。**
UNIX **は、**X/Open **社により管理されているオペレーティングシステムの名称です。**

■表記上の注意事項

本マニュアルの説明の中で使用する記号は、次の内容を示しています。

{ }	いずれか１つを選択することを意味します。
[ ]	省略できることを意味します。
・・・	任意の回数だけ繰り返すことを意味します。
×	指定できないことを意味します。
△	１つ以上の空白（スペースまたはタブコード）を意味します。
(RET)	リターンキーの入力を意味します。
>、%	プロンプトを意味します。

コンソールによっては、「¥」が「＼」と表示されることがあります。

目次

言語編

第 1 章　言語編概要

第 2 章　コーディングの基本規則

第 3 章　プログラミング概要

第 4 章　実行命令

第 5 章　アセンブラ制御命令

第 6 章　ファイルインクルード機能

第 7 章　条件付きアセンブリ機能

第 8 章　マクロ機能

第 9 章　構造化アセンブリ機能

操作編

第 10 章　操作編概要

第 11 章　アセンブラの起動

第 12 章　コマンドラインオプション

第 13 章　モジュール間最適化の オプション・環境変数

第 14 章　モジュール間最適化の エラーメッセージ

付録

図目次

言語編

操作編

付録

表目次

言語編

操作編

付録

1. 言語編概要

第 1 章　目次

1.1 特長

本アセンブラには、次のような特長があります。

(1) 対象とする CPU

本アセンブラは、次の CPU を対象としています。

・H8S/2600 シリーズ
・H8S/2000 シリーズ
・H8/300H シリーズ
・H8/300 シリーズ
・H8/300L シリーズ

(2) プリプロセッサ機能

次に示すプリプロセッサ機能により、効率よくソースプログラムを記述できます。

・ファイルインクルード機能
・条件付きアセンブリ機能
・マクロ機能
・構造化アセンブリ機能

(3) 標準規格に準拠

実行命令、アセンブラ制御命令の名称（ニーモニック）は、IEEE-694 仕様で規定された命名規則にのっとっており、統一された体系となっています。

1.2 アセンブラの概要

本アセンブラは、アセンブリ言語ソースプログラムを、リロケータブルオブジェクトモジュールに変換します。

また、処理結果を表示したアセンブルリストを出力します。

本アセンブラは、次のソースプログラムを入力できます。

・エディタで作成したアセンブリ言語ソースプログラム

・H8S、H8/300 シリーズ　C/C++コンパイラが出力したアセンブリ言語ソースプログラム

本アセンブラが出力したオブジェクトモジュールは、次のソフトウェアに入力できます。

・H シリーズ　リンケージエディタ

・H シリーズ　ライブラリアン

・H8S、H8/300 シリーズ　シミュレータ・デバッガ

2. コーディングの基本規則

第 2 章　目次

2.1 ソースステートメント

ソースプログラムは、ソースステートメントの集まりです。

ソースステートメントは、1 行 8,192 文字以内の ASCII 文字および、かな・漢字・LATIN1 コードで記述します。

2.1.1 ソースステートメントの種類

ソースステートメントには、次のものがあります。

■命令ステートメント

命令ステートメントは、実行命令、アセンブラ制御命令、プリプロセッサ制御文、マクロ命令を記述したステートメントです。

■ラベルステートメント

ラベルステートメントは、ラベルとしてシンボルを記述しただけのステートメントです。分岐先や、データの場所を示します。

■コメントステートメント

コメントステートメントは、コメントを記述しただけのステートメントです。ソースプログラムに対する注釈を記述します。

■空ステートメント

空ステートメントは、何も記述していないステートメントです。ソースプログラムを区切って見やすくします。

2.1.2 ソースステートメントの構成

ソースステートメントは、次のフィールドで構成されます。

■ラベルフィールド

ラベルフィールドには、そのソースステートメントの位置を示すためのラベルとして、シンボルを記述します。

■オペレーションフィールド

オペレーションフィールドには、実行命令、アセンブラ制御命令、プリプロセッサ制御文、マクロ命令を記述します。

■オペランドフィールド

オペランドフィールドには、オペレーションフィールドに記述した命令のオペランドを記述します。

■コメントフィールド

コメントフィールドには、コメントを記述します。

各フィールドのソースステートメント内における位置は、次のとおりです。

ラベル	オペレーション	オペランド	コメント

2.1.3 ソースステートメントの書き方

(1) ラベルフィールド

ラベルフィールドのシンボルの書き方は、次のとおりです。

(a) 1カラム目から書き始める。

(b) 1カラム目から書き始めてコロン(:)で終了する。

(c) 2カラム目以降から書き始めてコロン(:)で終了する。

また、シンボルが必要ない場合は、記述する必要はありません。

■例

```
LABEL1
LABEL2:
        LABEL3:
```

(2) オペレーションフィールド

オペレーションフィールドの命令の書き方は、次のとおりです。

(a) ラベルフィールドにシンボルを記述しない場合

2カラム目以降から書き始める。

(b) ラベルフィールドにシンボルを記述する場合

シンボルとの間に1つ以上のスペースまたはタブを置いてから書き始める。

また、命令が必要ない場合は、記述する必要はありません。

■例

```
          ADD.B     R0L,R1L
LABEL1    ADD.B     R0L,R1L
LABEL2
```

(3) オペランドフィールド

オペランドフィールドの書き方は、次のとおりです。

(a) オペレーションフィールドの命令にオペランドが必要な場合

命令との間に1つ以上のスペースまたはタブを置いてから書き始める。

(b) オペレーションフィールドの命令にオペランドが必要ない場合

何も記述しない。

(c) オペレーションフィールドに命令の記述がない場合

何も記述しない。

■例

```
          ADD.B     R0L,R1L
          NOP
LABEL
```

(4) コメントフィールド

コメントフィールドのコメントの書き方は、次のとおりです。

(a) コメントを記述する場合

セミコロン（;）を置いてから書き始める。

【注】 コメントとして、かな・漢字を記述することができます。

(b) コメントを記述しない場合

何も記述しない。

■例

```
          ADD.B     R0L,R1L ; R1L=R0L+R1L
          NOP               ; NO OPERAND
LABEL                       ; LABEL STATEMENT
; この行はコメントステートメントです。
```

2.1.4 行の継続

ソースステートメントを複数の行に渡って記述することができます。

オペランドフィールドのコンマ（,）を区切りとして、以降のオペランドを次の行に記述します。このとき、継続する行には、1 カラム目にプラス（+）を記述します。

■例

```
        ADD.B    R0L,    ; SOURCE OPERAND
+                R1L     ; DESTINATION OPERAND
```

2.2 シンボル

シンボルは、ソースステートメントの位置を示すためのラベルとして、ソースステートメントのロケーションカウンタ値（アドレス）を値に持ちます。

また、シンボル値を設定するアセンブラ制御命令の場合は、設定した値を持ちます。

2.2.1 シンボルの書き方

(1) 使用できる文字

シンボルに使用できる文字は、次のとおりです。

- アルファベット（A～Z、a～z）
- 数字（0～9）
- アンダスコア（_）
- ドル記号（$）

アルファベット（A～Z、a～z）の大文字と小文字は区別します。

(2) 書き方

シンボルは、アルファベット（A～Z、a～z）、アンダスコア（_）、ドル記号（$）で書き始めます。

シンボルの文字数は、最大 251 文字です。

(3) シンボルに使用できない名称

次の名称は、シンボルとして使用できません。

- レジスタニーモニック（ER0～ER7、E0～E7、R0～R7、R0H～R7H、R0L～R7L、SP、CCR、EXR、MACH、MACL）
- 演算子（STARTOF、SIZEOF、HIGH、LOW、HWORD、LWORD）
- ロケーションカウンタ（$）
- 内部シンボル（_$$00000～_$$FFFFF）

また、シンボル名とセクション名は重複できません。

【注】 内部シンボルとは、アセンブラの内部処理のため必要なシンボルです。内部シンボルは、アセンブルリストやオブジェクトモジュールには出力されません。

2.2.2 シンボルの種類

シンボルは、そのシンボルが持つ値によって、次のように分類できます。

(1) 絶対シンボル

絶対シンボルは、アセンブルの時点で値が決定しているシンボルです。
絶対シンボルには、次のものがあります。

(a) 定数シンボル

定数値を値に持つシンボルです。

(b) 絶対アドレスシンボル

絶対アドレス値を値に持つシンボルです。

(2) 相対シンボル

相対シンボルは、アセンブルの時点で値が決定せず、リンケージエディタでオブジェクトモジュールを連結する際に値が決定するシンボルです。
相対シンボルには、次のものがあります。

(a) 相対アドレスシンボル

相対アドレス値を値に持つシンボルです。

(b) 外部参照シンボル

外部参照値を値に持つシンボルです。

なお、それぞれの値については、**「2.3　値」**を参照してください。

2.2.3 シンボル参照

シンボルの参照には、次の形態があります。

- 前方参照
- 後方参照
- 外部参照

【補足】

本マニュアルでは、「前方」「後方」という用語を、**図 2.1** に示す意味で使っています。

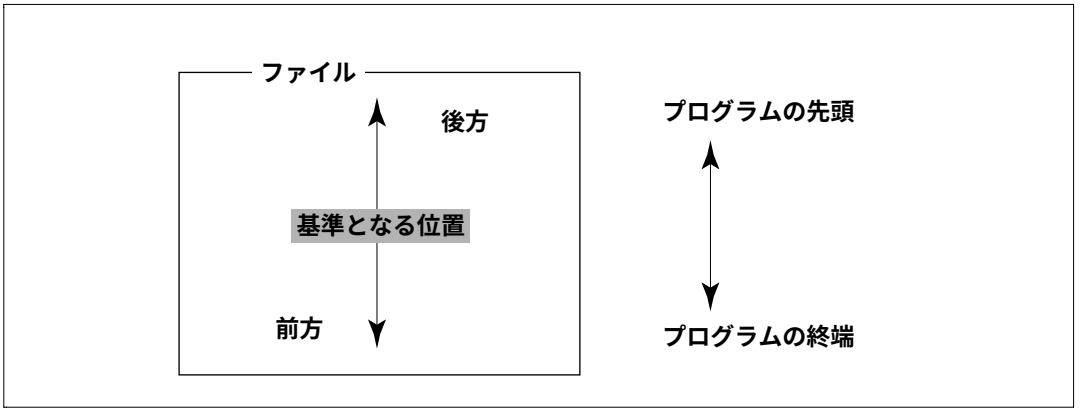


図 2.1 **「前方」「後方」の意味**

また、「外部」という用語を、**図 2.2** に示す意味で使っています。

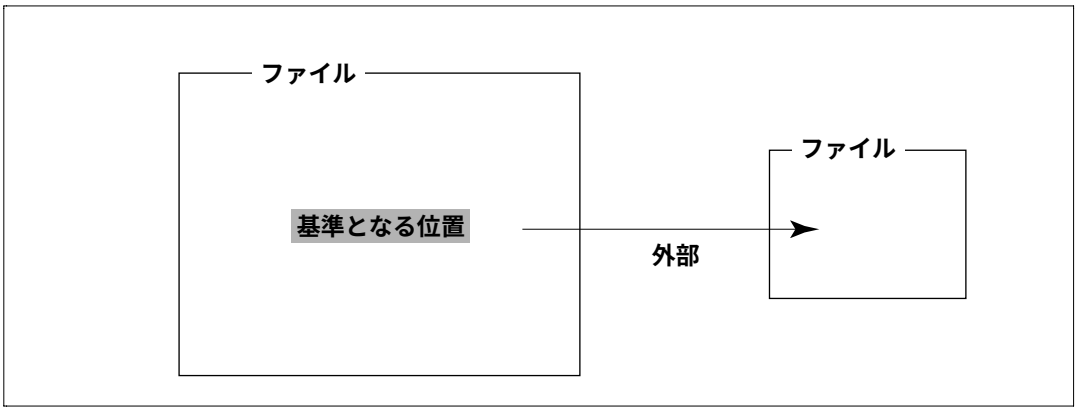


図 2.2 **「外部」の意味**

（1）前方参照

前方参照とは、前方で定義しているシンボルを参照することをいいます。

■例

```
        ～
        BRA     FORWARD     ; BRA は、分岐命令です。
                            ; 前方のシンボル FORWARD を参照します。
        ～
FORWARD:
        ～
```

(2) 後方参照

後方参照とは、後方で定義しているシンボルを参照することをいいます。

■例

```
                ～
BACK:
                ～
                BRA         BACK      ; BRA は、分岐命令です。
                                      ; 後方シンボル BACK を参照します。
                ～
```

(3) 外部参照

ソースプログラムが複数のファイルで構成される場合に、他のファイルで定義しているシンボルを参照することをいいます。

2.3 値

2.3.1 値の種類

値は、次のように分類できます。

(1) 絶対値

絶対値は、アセンブルの時点で値が決定しています。
絶対値には、次のものがあります。

(a) 定数値

・整数定数の値
・文字定数の値
・定数シンボルの値

(b) 絶対アドレス値

・絶対アドレスセクションのロケーションカウンタ値（アドレス）
・ダミーセクションのロケーションカウンタ値（アドレス）
・絶対アドレスシンボルの値

(2) 相対値

相対値は、アセンブルの時点で値が決定せず、リンケージエディタでオブジェクトモジュールを連結する際に値が決定します。
相対値には、次のものがあります。

(a) 相対アドレス値

・相対アドレスセクションのロケーションカウンタ値(アドレス)
・相対アドレスシンボルの値

(b) 外部参照値

・外部参照シンボルの値

また、シンボルの値は、そのシンボルの参照形態によって、次のように分類できます。

（1）後方参照値

後方参照シンボルの値

（2）前方参照値

前方参照シンボルの値

（3）外部参照値

外部参照シンボルの値

【注】 本マニュアルの説明の中で、「後方参照の絶対値」という場合は、次の値を意味します。

- **整数定数の値**
- **文字定数の値**
- **後方参照の定数シンボルの値**
- **絶対アドレスセクションのロケーションカウンタ値**
- **ダミーセクションのロケーションカウンタ値**
- **後方参照の絶対アドレスシンボルの値**

2.3.2 定数

定数には、整数定数と文字定数があります。

（1）整数定数

整数定数には、2 進数、8 進数、10 進数、16 進数があります。

整数定数は、基数と数値で記述します。

- 2 進数 ・・・・・・ 基数 B'と 2 進数で記述します。
- 8 進数 ・・・・・・ 基数 Q'と 8 進数で記述します。
- 10 進数 ・・・・・ 基数 D'と 10 進数で記述します。
- 16 進数 ・・・・・ 基数 H'と 16 進数で記述します。

基数を省略した場合は、通常は 10 進数になります（**「5.9.2 .RADIX 」**参照）。

■例

```
MOV.B   #B'00001010:8,R0L
MOV.B   #Q'012:8,R0L
MOV.B   #D'10:8,R0L
MOV.B   #H'0A:8,R0L
```

(2) 文字定数

文字定数は、4 文字以内の文字をダブルコーテーション（"）で囲んで記述します。

文字として指定できるのは、ASCII コードの H'09（タブ）と H'20（スペース）～H'7E (~) の文字です。また、シフト JIS コード、EUC コードのかな・漢字、もしくは LATIN1 コードを記述することができます。

文字として、かな・漢字を記述した場合は、必ずシフト JIS コードのときは SJIS コマンドラインオプション（**「12.7.1　SJIS」**参照）を、EUC コードのときは、EUC コマンドラインオプション（**「12.7.2　EUC」**参照）を指定してください。また、LATIN1 コードを記述したときは、LATIN1 コマンドラインオプション（**「12.7.4　LATIN1」**参照）を指定してください。

ダブルコーテーション（"）自体を文字として指定する場合は、2 つ続けて記述します。

■例

```
.CPU    2600A
MOV.B   #"A":8,R0L      ; A
MOV.B   #"""":8,R1L     ; "
MOV.W   #"AB":16,R2     ; AB
MOV.W   #"か":16,R1     ; か
MOV.L   #"漢字":32,ER0  ; 漢字
```

2.3.3 ロケーションカウンタ

ロケーションカウンタは、ドル記号（$）で参照します。

ロケーションカウンタの値は、そのソースステートメントの先頭のロケーションカウンタ値（アドレス）になります。

また、絶対アドレスセクションでロケーションカウンタを参照すると、絶対アドレス値が得られ、相対アドレスセクションでロケーションカウンタを参照すると、相対アドレス値が得られます。

■例

```
          .SECTION A,CODE,LOCATE=H'1000‥‥ [1]
ABS       .EQU     $‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ [2]
          MOV.W    R0,R1

          .SECTION B,CODE,ALIGN=2‥‥‥‥‥‥ [3]
REL       .EQU     $‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ [4]
          MOV.W    R0,R1
```

[1] 絶対アドレスセクションの宣言をしています。

[2] $は絶対アドレス値で H'1000 になります。

[3] 相対アドレスセクションの宣言をしています。

[4] $は相対アドレス値で H'0000 になります。

2.4 式

2.4.1 式の要素

式は、項と演算子とカッコで構成されます。

■項

項は、次のとおりです。

・定数
・ロケーションカウンタ（$）
・シンボル
・上記の項と演算子による演算結果

■演算子

表 2.1 に、演算子一覧を示します。

表 2.1 **演算子一覧**

演算区分	演算子	演算内容	書き方
算術演算	+	**単項プラス**	+**項**
	–	**単項マイナス**	–**項**
	+	**加算**	**項** 1+**項** 2
	–	**減算**	**項** 1–**項** 2
	＊	**乗算**	**項** 1＊**項** 2
	/	**除算**	**項** 1/**項** 2
論理演算	~	**単項否定**	~**項**
	&	**論理積**	**項** 1&**項** 2
	|	**論理和**	**項** 1|**項** 2
	~	**排他的論理和**	**項** 1~**項** 2
シフト演算	<<	**算術左シフト**	**項** 1<<**項** 2
	>>	**算術右シフト**	**項** 1>>**項** 2
セクション集合演算	STARTOF	**セクション集合の先頭アドレス**	STARTOF **セクション名**
	SIZEOF	**セクション集合のサイズ**	SIZEOF **セクション名**
抽出演算	HIGH	**上位バイト抽出**	HIGH **項**
	LOW	**下位バイト抽出**	LOW **項**
	HWORD	**上位ワード抽出**	HWORD **項**
	LWORD	**下位ワード抽出**	LWORD **項**

【注】 HWORD、LWORD **は、**H8S/2600、H8S/2000、H8/300H **のアドバンストモードと** H8S/2600、H8S/2000、H8/300H **のノーマルモードで使用できます。**H8/300、H8/300L **では使用できません。**

■カッコ

カッコは、演算の優先順位を変更します。

2.4.2 演算子の優先順位

表 2.2 に、演算子の優先順位を示します。

表 2.2 **演算子の優先順位**

優先順位	演算子	結合規則
1	+ − ~ STARTOF SIZEOF HIGH LOW HWORD LWORD	右から左の順に演算を処理する
2	* ／	左から右の順に演算を処理する
3	+ −	左から右の順に演算を処理する
4	<< >>	左から右の順に演算を処理する
5	&	左から右の順に演算を処理する
6	│ ~	左から右の順に演算を処理する

【注】 1. **優先順位１の演算子は、単項演算子です。**

2. **同一優先順位の演算子は、結合規則の方向に従って演算されます。**

■例

10+20/5・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 14 になります。
(10+20)/5 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 6 になります。
H'0000FFFF|H'00FF00FF&H'0F0F0F0F・・・・・・・・ H'000FFFFF になります。
(H'0000FFFF|H'00FF00FF)&H'0F0F0F0F ・・・・・ H'000F0F0F になります。
HIGH H'12345678 ・・・・・・・・・・・・・・・・・・・・・・・・・・・・ H'00000056 になります。
LOW HWORD H'12345678 ・・・・・・・・・・・・・・・・・・・・・・・ H'00000034 になります。
~H'000000C3・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・H'FFFFFF3C になります。
~H'000000C3&H'000000FF・・・・・・・・・・・・・・・・・・・・ H'0000003C になります。

【注】 **この例では、基数を省略した定数は** 10 **進数としています。**

2.4.3 演算の詳細

（1）STARTOF 演算

セクション集合の先頭アドレスを求めます。

指定したセクションがリンケージエディタで連結された後のセクション先頭アドレスを求めます。

（2）SIZEOF 演算

セクション集合のサイズを求めます。

指定したセクションがリンケージエディタで連結された後のセクションサイズを求めます。

■例

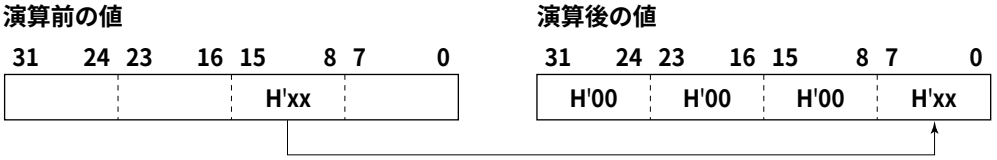
```
            .CPU 2600A
            .SECTION  INIT_RAM,DATA,ALIGN=2
            .RES      H'100
  ;
            .SECTION  INIT_DATA,DATA,ALIGN=2
  INIT_BGN  .DATA.L   STARTOF INIT_RAM ····· [1]
  INIT_END  .DATA.L   STARTOF INIT_RAM + SIZEOF INIT_RAM ····· [2]
  ;
  ;
            .SECTION MAIN,CODE,ALIGN=2
  INITAL:
            MOV.L   @INIT_BGN,ER1
            MOV.L   @INIT_END,ER2
            MOV.W   #0,R3
  LOOP:
            CMP.L   ER1,ER2
            BEQ     END
            MOV.W   R3,@ER1
            ADDS.L  #1,ER1
            BRA     LOOP
  END:
            SLEEP
            .END
```

セクション（INIT_RAM）のデータ領域をゼロで初期化します。

[1] セクション（INIT_RAM）の先頭アドレスを求めます。
[2] セクション（INIT_RAM）の最終アドレスを求めます。

（3）HIGH 演算

4バイト値の下位2バイトの上位バイトを抽出します。

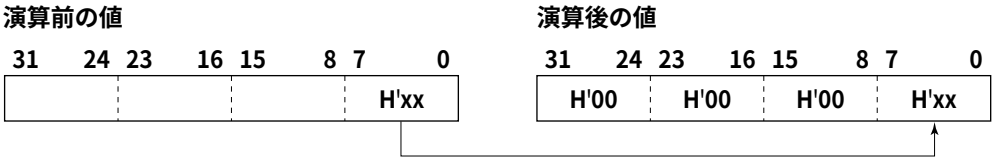


■例

```
LABEL     .EQU     H'00007FFF
          MOV.W    #HIGH LABEL,R0    ;R0=H'007F
```

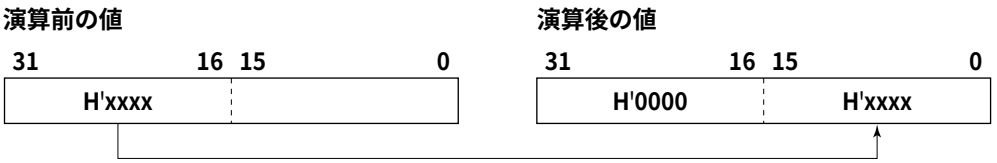
（4）LOW 演算

4バイト値の下位バイトを抽出します。



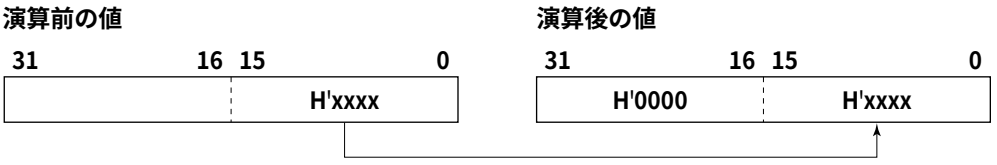
（5）HWORD 演算

4バイト値の上位2バイトを抽出します。



（6）LWORD 演算

4バイト値の下位2バイトを抽出します。



2.4.4 演算の注意事項

(1) 演算の仕様

演算は、すべて符号付き 32 ビット演算です。

オペランドのサイズが、バイトサイズ（8 ビット）、またはワードサイズ（16 ビット）であっても、演算は符号付き 32 ビット演算です。このため、次の例のような場合にエラーとなりますので注意してください。

■例

```
MOV.B  #~H'80:8,R0L ・・・・・・・・#~H'80:8 がエラーになります。
```

~H'80 の演算内容は、次のとおりです。

```
~      H'80
~H'00000080
 H'FFFFFF7F
```

H'FFFFFF7F は、8 ビット値の範囲外なのでエラーとなります。

これを回避するためには、次のようにしてください。

```
MOV.B  #H'7F:8,R0L ・・・・・・・・・・・・演算結果の値を直接記述する。
MOV.B  #~H'80&H'FF:8,R0L ・・・・・・論理積を用いて下位ビットを有効とする。
MOV.B  #LOW ~H'80:8,R0L ・・・・・・・下位バイト抽出を用いて下位 8 ビット
                                    を有効とする。
```

(2) 算術演算

(a) 乗算、除算では、項に相対値は指定できません。相対値は指定しないでください。

(b) 除算では、除数に 0 は指定できません。0 は指定しないでください。

(3) 論理演算

論理演算では、項に相対値は指定できません。相対値は指定しないでください。

(4) セクション集合演算

セクション集合演算では、セクション名以外は指定できません。セクション名以外は指定しないでください。

2.5 文字列

文字列は、文字をダブルコーテーション（"）で囲んで記述します。

文字として指定できるのは、ASCII コードの H'09（タブ）と H'20（スペース）～H'7E（~）の文字です。

また、シフト JIS コード、EUC コードのかな・漢字、もしくは LATIN1 コードを記述することができます。

文字として、かな・漢字を記述した場合は、必ずシフト JIS コードのときは SJIS コマンドラインオプション（**「12.7.1　SJIS」**参照）を、EUC コードのときは EUC コマンドラインオプション（**「12.7.2　EUC」**参照）を指定してください。また、LATIN1 コードを記述したときは、LATIN1 コマンドラインオプション（**「12.7.4　LATIN1」**参照）を指定してください。

ダブルコーテーション（"）自体を文字として指定する場合は、2 つ続けて記述します。

■例

```
.SDATA "ABCDEF"     ; ABCDEF
.SDATA "ABC""DE"    ; ABC"DE
.SDATA "漢字記述"     ; 漢字記述
```

2.6 ローカルラベル

ローカルラベルは、通常のラベルとラベルの間で局所的に有効なラベルです。ローカルラベルは、有効範囲外の他のラベルと衝突することがありませんので、他のラベル名を気にせず、局所的な制御を記述できます。

【注】 ローカルラベルは、デバッグ時には参照できません。

(1) 記述方法

ローカルラベルは、先頭が"?"で、それ以外の文字が以下の ASCII 文字からなる文字列です。

(a) 英大文字、英小文字（A～Z、a～z）
(b) 数字（0～9）
(c) アンダースコア（_）
(d) ドル（$）

ローカルラベルの文字数は 2 文字以上 16 文字以内（"?"を含む）です。

ローカルラベルは、通常のラベルと同様に、ラベルフィールドに指定することによって定義でき、オペランド内で参照できます。

【注】 ローカルラベルは以下の位置には指定できません。

(a) マクロ名
(b) セクション名
(c) オブジェクトモジュール名
(d) .ASSIGNA、.ASSIGNC、.EQU、.BEQU、.ASSIGN、.REG のラベルフィールド
(e) .EXPORT、.IMPORT、.GLOBAL のオペランド
(f) .DEFINE のラベルフィールド

(2) 有効範囲

ローカルラベルの有効範囲をローカルブロックといいます。ローカルブロックの区切りは、（ローカルラベルでない）通常のラベル、または.SECTION です。

このローカルブロック内で定義されたローカルラベルは、当該ブロック内で参照できます。

異なるブロックに属するローカルラベルは、綴りが同じであっても別のラベルと解釈し、エラーにはなりません。

【注】 .ASSIGNA、.ASSIGNC、.EQU、.BEQU、.ASSIGN、.REG、.DEFINE **のラベルは有効範囲の区切りとはみなしません。**

■例

```
LABEL1:                          ;  Local block 1 start
?0001:     CMP.W  R1,R2
           BEQ    ?0002
           BRA    ?0001
?0002:
LABEL2:                          ;  Local block 2 start
?0001:     CMP.W  R1,R2
           BGE    ?0002
           BRA    ?0001
?0002:
LABEL3:
```

3. プログラミング概要

第 3 章　目次

3.1 セクション

セクションは、プログラムの構成単位で命令やデータの集まりです。また、リンケージエディタでは、セクション単位でオブジェクトモジュールの連結やメモリ上の配置を行います。

プログラムを作成する際は、そのセクションの用途やメモリ上での配置によって、セクション属性、形式種別を選択します。

(1) セクション属性

セクション属性には、次のものがあります。

(a) コードセクション

コードセクションは、実行命令、またはプログラムの実行時に内容を変更しないデータを記述します。

コードセクションは、ROM に配置します。

(b) データセクション

データセクションは、プログラムの実行時に内容を変更しないデータ、または内容を変更するデータ領域を記述します。

データセクションは、前者は ROM に、後者は RAM に配置します。

(c) スタックセクション

スタックセクションは、初期値を持たない領域だけのセクションで、プログラムのスタックエリアやワークエリアとして使用します。

スタックセクションは、RAM に配置します。

(d) コモンセクション

コモンセクションは、プログラムの実行時に内容を変更しないデータ、または内容を変更するデータ領域を記述します。データを分割したプログラム間で共通に使用する場合に使用します。

コモンセクションは、前者は ROM に、後者は RAM に配置します。

(e) ダミーセクション

ダミーセクションは、初期値を持たない領域だけのセクションで、変数構造を記述します。

ダミーセクションは、オブジェクトモジュールには出力されません。

（2）形式種別

形式種別には、次のものがあります。

（a）絶対アドレス形式

絶対アドレス形式は、セクションをメモリ上のどのアドレスに配置すべきかをあらかじめソースプログラムで指定しておく形式です。

絶対アドレスセクション（絶対アドレス形式のセクション）のアドレスは、リンケージエディタで変更できません。

絶対アドレスセクションのアドレスは、.SECTION 制御命令の LOCATE で指定した値です。

（b）相対アドレス形式

相対アドレス形式は、セクションをメモリ上のどのアドレスに配置すべきかをソースプログラムでは指定しない形式です。

相対アドレスセクション（相対アドレス形式のセクション）のアドレスは、リンケージエディタでオブジェクトモジュールを連結する際に決定します。

また、相対アドレスセクションでは、リンケージエディタがセクションのアドレスを決定するときの境界調整数を設定できます。

リンケージエディタは、セクションのアドレスが境界調整数の倍数番地になるようにアドレスを決定します。

セクションの境界調整数は、コードセクションの実行命令のようにメモリの偶数番地に配置しなければならない場合などに有効です。

セクションの属性と形式種別は、.SECTION 制御命令で宣言します。

■例

```
        .CPU    2600A
        .OUTPUT DBG
SIZE    .EQU    8
;
        .SECTION A,CODE,ALIGN=2 ····· [1]
START                                          ┐
        MOV.L   #CONST:32,ER0                  ├ 相対アドレス形式のコードセクション
        MOV.L   #DATA:32,ER1                   │
        MOV.L   #SIZE:32,ER2                   │
LOOP                                           │
        CMP.L   #0:32,ER2                      │
        BEQ     EXIT                           │
        MOV.B   @ER0,R3L                       │
        MOV.B   R3L,@ER1                       │
        ADD.L   #1:32,ER0                      │
        ADD.L   #1:32,ER1                      │
        SUB.L   #1:32,ER2                      │
        BRA     LOOP                           │
EXIT                                           │
        SLEEP                                  │
        BRA     START                          ┘
;
        .SECTION B,DATA,LOCATE=H'00001000····· [2]
CONST                                          ┐
        .DATA.B H'01,H'02,H'03,H'04            ├ 絶対アドレス形式のデータセクション
        .DATA.B H'05,H'06,H'07,H'08            ┘
;
        .SECTION C,STACK,ALIGN=2 ····· [3]
DATA                                           ┐
        .RES.B  SIZE                           ├ 相対アドレス形式のスタックセクション
;                                              ┘
        .END    START
```

[1] 相対アドレス形式のコードセクションの宣言をしています。
[2] 絶対アドレス形式のデータセクションの宣言をしています。
[3] 相対アドレス形式のスタックセクションの宣言をしています。

3.2 外部参照と外部定義

ソースプログラムを分割して作成すると、別ソースプログラムのサブルーチンをコールしたり、データを参照したりします。

アセンブリ言語では、シンボルを参照することによってサブルーチンをコールしたり、データを参照したりするので、ソースプログラム中に定義していない別ソースプログラムのシンボルを参照することになります。

この別ソースプログラムのシンボルを参照するのが外部参照で、別ソースプログラムにシンボルを参照させるのが外部定義です。

外部参照シンボル、外部定義シンボルは、.IMPORT 制御命令、.EXPORT 制御命令で宣言します。

外部参照シンボルの宣言

```
        .CPU    2600A
        .OUTPUT DBG
;
        .IMPORT SUBRTN ····· [1]
        .IMPORT DATA1,DATA2····· [1]
;
        .SECTION A,CODE,ALIGN=2
MAIN
        MOV.L   #STACK:32,SP
        MOV.L   @DATA1:32,ER0
        MOV.L   @DATA2:32,ER1
        MOV.L   #0:32,ER2
        JSR     @SUBRTN:24
        SLEEP
        BRA     MAIN
;
        .SECTION B,STACK,ALIGN=2
        .RES.B H'500
STACK
;
        .END     MAIN
```

外部定義シンボルの宣言

```
        .CPU    2600A
        .OUTPUT DBG
;
        .EXPORT SUBRTN ····· [2]
        .EXPORT DATA1,DATA2····· [2]
;
        .SECTION A,CODE,ALIGN=2
SUBRTN
        MOV.L   ER0,ER2
        ADD.L   ER1,ER2
        RTS
;
        .SECTION C,DATA,ALIGN=1
DATA1
        .DATA.B H'01
DATA2
        .DATA.B H'02
;
        .END
```

[1] シンボルSUBRTN、DATA1、DATA2を外部参照しています。
[2] シンボルSUBRTN、DATA1、DATA2を外部定義しています。

図 3.1 外部参照シンボル、外部定義シンボルの宣言例

4. 実行命令

第 4 章　目次

4.1 実行命令の書き方

実行命令の書き方は、次のとおりです。

ラベル	オペレーション	オペランド
[**シンボル名**]	**ニーモニック** [.s]	[**アドレス形式** [,**アドレス形式**]]

s（**サイズ**）：{B｜W｜L}

（1）ラベルフィールド

必要に応じてシンボルを記述します。

シンボルは、そのソースステートメントのロケーションカウンタ値（アドレス）を値に持ちます。

（2）オペレーションフィールド

実行命令のニーモニックを記述します。

また、実行命令のニーモニックにピリオド（.）で区切って、オペランドのサイズを記述します。

サイズは、次のとおりです。

B‥‥‥ バイト　　（1バイト）
W ‥‥ ワード　　（2バイト）
L‥‥‥ ロングワード（4バイト）

指定できるサイズは、各実行命令によって異なります。

（3）オペランドフィールド

実行命令のオペランドを記述します。

実行命令のオペランドの数、オペランドのアドレス形式は、各実行命令によって異なります。

4.2 実行命令のサイズとアドレス形式

(1) H8S/2600 シリーズ

(a) H8S/2600 アドバンストモード

表 4.1 に、H8S/2600 アドバンストモードの実行命令のサイズを示します。

表 4.1 H8S/2600 アドバンストモードの実行命令のサイズ

項番	実行命令	サイズ	項番	実行命令	サイズ	項番	実行命令	サイズ
1	ADD	B W L (B)	26	DEC	B W L (B)	51	ROTL	B W L (B)
2	ADDS	L (L)	27	DIVXS	B W (B)	52	ROTR	B W L (B)
3	ADDX	B (B)	28	DIVXU	B W (B)	53	ROTXL	B W L (B)
4	AND	B W L (B)	29	EEPMOV	B W (B)	54	ROTXR	B W L (B)
5	ANDC	B (B)	30	EXTS	W L (W)	55	RTE	—
6	BAND	B (B)	31	EXTU	W L (W)	56	RTS	—
7	Bcc	—	32	INC	B W L (B)	57	SHAL	B W L (B)
8	BCLR	B (B)	33	JMP	—	58	SHAR	B W L (B)
9	BIAND	B (B)	34	JSR	—	59	SHLL	B W L (B)
10	BILD	B (B)	35	LDC	B W (B)	60	SHLR	B W L (B)
11	BIOR	B (B)	36	LDM	L (L)	61	SLEEP	—
12	BIST	B (B)	37	LDMAC	L (L)	62	STC	B W (B)
13	BIXOR	B (B)	38	MAC	—	63	STM	L (L)
14	BLD	B (B)	39	MOV	B W L (B)	64	STMAC	L (L)
15	BNOT	B (B)	40	MOVFPE	B (B)	65	SUB	B W L (B)
16	BOR	B (B)	41	MOVTPE	B (B)	66	SUBS	L (L)
17	BSET	B (B)	42	MULXS	B W (B)	67	SUBX	B (B)
18	BSR	—	43	MULXU	B W (B)	68	TAS	B (B)
19	BST	B (B)	44	NEG	B W L (B)	69	TRAPA	—
20	BTST	B (B)	45	NOP	—	70	XOR	B W L (B)
21	BXOR	B (B)	46	NOT	B W L (B)	71	XORC	B (B)
22	CLRMAC	—	47	OR	B W L (B)			
23	CMP	B W L (B)	48	ORC	B (B)			
24	DAA	B (B)	49	POP	W L (L)			
25	DAS	B (B)	50	PUSH	W L (L)			

【注】 ( ) ・・・・・・・・ サイズ指定を省略した場合のデフォルトサイズ

— ・・・・・・・・・ サイズ指定不可

表 4.2 に、H8S/2600 アドバンストモードのアドレス形式を示します。

表 4.2　H8S/2600 アドバンストモードのアドレス形式

項番	アドレス形式	書き方
1	レジスタ直接形式	{ERn｜En｜Rn｜RnH｜RnL}
2	レジスタ間接形式	@ERn
3	ディスプレースメント付レジスタ間接形式	@(d[:{16｜32}],ERn)
4	ポストインクリメントレジスタ間接形式	@ERn+
5	プリデクリメントレジスタ間接形式	@-ERn
6	絶対アドレス形式	@aa[:{8｜16｜24｜32}]
7	イミディエートデータ形式	#xx[:{8｜16｜32}]
8	メモリ間接形式	@@aa[:8]
9	プログラムカウンタ相対形式 （分岐命令のディスプレースメント）	d[:{8｜16}]
10	コントロールレジスタ	CCR、EXR、MACH、MACL

【注】 1. n ‥‥‥ レジスタ番号（0～7）
d ‥‥‥ ディスプレースメント
aa‥‥‥ 絶対アドレス
xx‥‥‥ イミディエートデータ
2. ER7 は、SP（スタックポインタ）と同じです。

【注】 FP（フレームポインタ）は、レジスタニーモニックではありません。
FP（フレームポインタ）をレジスタとして記述したい場合は、.REG 制御命令を使用してください。

(b) H8S/2600 ノーマルモード

表 4.3 に、H8S/2600 ノーマルモードの実行命令のサイズを示します。

表 4.3　H8S/2600 **ノーマルモードの実行命令のサイズ**

項番	実行命令	サイズ	項番	実行命令	サイズ	項番	実行命令	サイズ
1	ADD	B W L (B)	26	DEC	B W L (B)	51	ROTL	B W L (B)
2	ADDS	L (L)	27	DIVXS	B W (B)	52	ROTR	B W L (B)
3	ADDX	B (B)	28	DIVXU	B W (B)	53	ROTXL	B W L (B)
4	AND	B W L (B)	29	EEPMOV	B W (B)	54	ROTXR	B W L (B)
5	ANDC	B (B)	30	EXTS	W L (W)	55	RTE	—
6	BAND	B (B)	31	EXTU	W L (W)	56	RTS	—
7	Bcc	—	32	INC	B W L (B)	57	SHAL	B W L (B)
8	BCLR	B (B)	33	JMP	—	58	SHAR	B W L (B)
9	BIAND	B (B)	34	JSR	—	59	SHLL	B W L (B)
10	BILD	B (B)	35	LDC	B W (B)	60	SHLR	B W L (B)
11	BIOR	B (B)	36	LDM	L (L)	61	SLEEP	—
12	BIST	B (B)	37	LDMAC	L (L)	62	STC	B W (B)
13	BIXOR	B (B)	38	MAC	—	63	STM	L (L)
14	BLD	B (B)	39	MOV	B W L (B)	64	STMAC	L (L)
15	BNOT	B (B)	40	MOVFPE	B (B)	65	SUB	B W L (B)
16	BOR	B (B)	41	MOVTPE	B (B)	66	SUBS	L (L)
17	BSET	B (B)	42	MULXS	B W (B)	67	SUBX	B (B)
18	BSR	—	43	MULXU	B W (B)	68	TAS	B (B)
19	BST	B (B)	44	NEG	B W L (B)	69	TRAPA	—
20	BTST	B (B)	45	NOP	—	70	XOR	B W L (B)
21	BXOR	B (B)	46	NOT	B W L (B)	71	XORC	B (B)
22	CLRMAC	—	47	OR	B W L (B)			
23	CMP	B W L (B)	48	ORC	B (B)			
24	DAA	B (B)	49	POP	W L (W)			
25	DAS	B (B)	50	PUSH	W L (W)			

【注】　( ) ･･･････ **サイズ指定を省略した場合のデフォルトサイズ**

— ･･･････ **サイズ指定不可**

表 4.4 に、H8S/2600 ノーマルモードのアドレス形式を示します。

表 4.4　H8S/2600 ノーマルモードのアドレス形式

項番	アドレス形式	書き方
1	レジスタ直接形式	{ERn｜En｜Rn｜RnH｜RnL}
2	レジスタ間接形式	@ERn
3	ディスプレースメント付レジスタ間接形式	@(d[:16],ERn)
4	ポストインクリメントレジスタ間接形式	@ERn+
5	プリデクリメントレジスタ間接形式	@-ERn
6	絶対アドレス形式	@aa[:{8｜16}]
7	イミディエートデータ形式	#xx[:{8｜16｜32}]
8	メモリ間接形式	@@aa[:8]
9	プログラムカウンタ相対形式 （分岐命令のディスプレースメント）	d[:{8｜16}]
10	コントロールレジスタ	CCR、EXR、MACH、MACL

【注】 1. n ‥‥‥ レジスタ番号（0～7）
d ‥‥‥ ディスプレースメント
aa‥‥‥ 絶対アドレス
xx‥‥‥ イミディエートデータ
2. ER7 は、SP（スタックポインタ）と同じです。

【注】 FP（フレームポインタ）は、レジスタニーモニックではありません。
FP（フレームポインタ）をレジスタとして記述したい場合は、.REG 制御命令を使用してください。

（2）H8S/2000 シリーズ

（a）H8S/2000 アドバンストモード

表 4.5 に、H8S/2000 アドバンストモードの実行命令のサイズを示します。

表 4.5　H8S/2000 **アドバンストモードの実行命令のサイズ**

項番	実行命令	サイズ	項番	実行命令	サイズ	項番	実行命令	サイズ
1	ADD	B W L (B)	26	DIVXS	B W (B)	51	ROTXR	B W L (B)
2	ADDS	L (L)	27	DIVXU	B W (B)	52	RTE	—
3	ADDX	B (B)	28	EEPMOV	B W (B)	53	RTS	—
4	AND	B W L (B)	29	EXTS	W L (W)	54	SHAL	B W L (B)
5	ANDC	B (B)	30	EXTU	W L (W)	55	SHAR	B W L (B)
6	BAND	B (B)	31	INC	B W L (B)	56	SHLL	B W L (B)
7	Bcc	—	32	JMP	—	57	SHLR	B W L (B)
8	BCLR	B (B)	33	JSR	—	58	SLEEP	—
9	BIAND	B (B)	34	LDC	B W (B)	59	STC	B W (B)
10	BILD	B (B)	35	LDM	L (L)	60	STM	L (L)
11	BIOR	B (B)	36	MOV	B W L (B)	61	SUB	B W L (B)
12	BIST	B (B)	37	MOVFPE	B (B)	62	SUBS	L (L)
13	BIXOR	B (B)	38	MOVTPE	B (B)	63	SUBX	B (B)
14	BLD	B (B)	39	MULXS	B W (B)	64	TAS	B (B)
15	BNOT	B (B)	40	MULXU	B W (B)	65	TRAPA	—
16	BOR	B (B)	41	NEG	B W L (B)	66	XOR	B W L (B)
17	BSET	B (B)	42	NOP	—	67	XORC	B (B)
18	BSR	—	43	NOT	B W L (B)			
19	BST	B (B)	44	OR	B W L (B)			
20	BTST	B (B)	45	ORC	B (B)			
21	BXOR	B (B)	46	POP	W L (L)			
22	CMP	B W L (B)	47	PUSH	W L (L)			
23	DAA	B (B)	48	ROTL	B W L (B)			
24	DAS	B (B)	49	ROTR	B W L (B)			
25	DEC	B W L (B)	50	ROTXL	B W L (B)			

【注】　( ) ･･･････ **サイズ指定を省略した場合のデフォルトサイズ**

— ･･･････ **サイズ指定不可**

表 4.6 に、H8S/2000 アドバンストモードのアドレス形式を示します。

表 4.6　H8S/2000 アドバンストモードのアドレス形式

項番	アドレス形式	書き方
1	**レジスタ直接形式**	{ERn｜En｜Rn｜RnH｜RnL}
2	**レジスタ間接形式**	@ERn
3	**ディスプレースメント付レジスタ間接形式**	@(d[:{16｜32}],ERn)
4	**ポストインクリメントレジスタ間接形式**	@ERn+
5	**プリデクリメントレジスタ間接形式**	@-ERn
6	**絶対アドレス形式**	@aa[:{8｜16｜24｜32}]
7	**イミディエートデータ形式**	#xx[:{8｜16｜32}]
8	**メモリ間接形式**	@@aa[:8]
9	**プログラムカウンタ相対形式** **（分岐命令のディスプレースメント）**	d[:{8｜16}]
10	**コントロールレジスタ**	CCR、EXR

【注】 1. n ‥‥‥ **レジスタ番号（**0～7**）**

d ‥‥‥ **ディスプレースメント**

aa‥‥‥ **絶対アドレス**

xx‥‥‥ **イミディエートデータ**

2. ER7 **は、**SP**（スタックポインタ）と同じです。**

【注】 FP**（フレームポインタ）は、レジスタニーモニックではありません。**
FP**（フレームポインタ）をレジスタとして記述したい場合は、**.REG **制御命令を使用してください。**

(b) H8S/2000 ノーマルモード

表 4.7 に、H8S/2000 ノーマルモードの実行命令のサイズを示します。

表 4.7　H8S/2000 **ノーマルモードの実行命令のサイズ**

項番	実行命令	サイズ	項番	実行命令	サイズ	項番	実行命令	サイズ
1	ADD	B W L (B)	26	DIVXS	B W (B)	51	ROTXR	B W L (B)
2	ADDS	L (L)	27	DIVXU	B W (B)	52	RTE	—
3	ADDX	B (B)	28	EEPMOV	B W (B)	53	RTS	—
4	AND	B W L (B)	29	EXTS	W L (W)	54	SHAL	B W L (B)
5	ANDC	B (B)	30	EXTU	W L (W)	55	SHAR	B W L (B)
6	BAND	B (B)	31	INC	B W L (B)	56	SHLL	B W L (B)
7	Bcc	—	32	JMP	—	57	SHLR	B W L (B)
8	BCLR	B (B)	33	JSR	—	58	SLEEP	—
9	BIAND	B (B)	34	LDC	B W (B)	59	STC	B W (B)
10	BILD	B (B)	35	LDM	L (L)	60	STM	L (L)
11	BIOR	B (B)	36	MOV	B W L (B)	61	SUB	B W L (B)
12	BIST	B (B)	37	MOVFPE	B (B)	62	SUBS	L (L)
13	BIXOR	B (B)	38	MOVTPE	B (B)	63	SUBX	B (B)
14	BLD	B (B)	39	MULXS	B W (B)	64	TAS	B (B)
15	BNOT	B (B)	40	MULXU	B W (B)	65	TRAPA	—
16	BOR	B (B)	41	NEG	B W L (B)	66	XOR	B W L (B)
17	BSET	B (B)	42	NOP	—	67	XORC	B (B)
18	BSR	—	43	NOT	B W L (B)			
19	BST	B (B)	44	OR	B W L (B)			
20	BTST	B (B)	45	ORC	B (B)			
21	BXOR	B (B)	46	POP	W L (W)			
22	CMP	B W L (B)	47	PUSH	W L (W)			
23	DAA	B (B)	48	ROTL	B W L (B)			
24	DAS	B (B)	49	ROTR	B W L (B)			
25	DEC	B W L (B)	50	ROTXL	B W L (B)			

【注】　( ) ･･･････ **サイズ指定を省略した場合のデフォルトサイズ**

— ･･･････ **サイズ指定不可**

表 4.8 に、H8S/2000 ノーマルモードのアドレス形式を示します。

表 4.8　H8S/2000 ノーマルモードのアドレス形式

項番	アドレス形式	書き方
1	レジスタ直接形式	{ERn｜En｜Rn｜RnH｜RnL}
2	レジスタ間接形式	@ERn
3	ディスプレースメント付レジスタ間接形式	@(d[:16],ERn)
4	ポストインクリメントレジスタ間接形式	@ERn+
5	プリデクリメントレジスタ間接形式	@-ERn
6	絶対アドレス形式	@aa[:{8｜16}]
7	イミディエートデータ形式	#xx[:{8｜16｜32}]
8	メモリ間接形式	@@aa[:8]
9	プログラムカウンタ相対形式 （分岐命令のディスプレースメント）	d[:{8｜16}]
10	コントロールレジスタ	CCR、EXR

【注】 1. n …… レジスタ番号（0～7）
d …… ディスプレースメント
aa …… 絶対アドレス
xx …… イミディエートデータ
2. ER7 は、SP（スタックポインタ）と同じです。

【注】 FP（フレームポインタ）は、レジスタニーモニックではありません。
FP（フレームポインタ）をレジスタとして記述したい場合は、.REG 制御命令を使用してください。

(3) H8/300H シリーズ

(a) H8/300H アドバンストモード

表 4.9 に、H8/300H アドバンストモードの実行命令のサイズを示します。

表 4.9　H8/300H **アドバンストモードの実行命令のサイズ**

項番	実行命令	サイズ	項番	実行命令	サイズ	項番	実行命令	サイズ
1	ADD	B W L (B)	26	DIVXS	B W (B)	51	RTE	—
2	ADDS	L (L)	27	DIVXU	B W (B)	52	RTS	—
3	ADDX	B (B)	28	EEPMOV	B W (B)	53	SHAL	B W L (B)
4	AND	B W L (B)	29	EXTS	W L (W)	54	SHAR	B W L (B)
5	ANDC	B (B)	30	EXTU	W L (W)	55	SHLL	B W L (B)
6	BAND	B (B)	31	INC	B W L (B)	56	SHLR	B W L (B)
7	Bcc	—	32	JMP	—	57	SLEEP	—
8	BCLR	B (B)	33	JSR	—	58	STC	B W (B)
9	BIAND	B (B)	34	LDC	B W (B)	59	SUB	B W L (B)
10	BILD	B (B)	35	MOV	B W L (B)	60	SUBS	L (L)
11	BIOR	B (B)	36	MOVFPE	B (B)	61	SUBX	B (B)
12	BIST	B (B)	37	MOVTPE	B (B)	62	TRAPA	—
13	BIXOR	B (B)	38	MULXS	B W (B)	63	XOR	B W L (B)
14	BLD	B (B)	39	MULXU	B W (B)	64	XORC	B (B)
15	BNOT	B (B)	40	NEG	B W L (B)			
16	BOR	B (B)	41	NOP	—			
17	BSET	B (B)	42	NOT	B W L (B)			
18	BSR	—	43	OR	B W L (B)			
19	BST	B (B)	44	ORC	B (B)			
20	BTST	B (B)	45	POP	W L (L)			
21	BXOR	B (B)	46	PUSH	W L (L)			
22	CMP	B W L (B)	47	ROTL	B W L (B)			
23	DAA	B (B)	48	ROTR	B W L (B)			
24	DAS	B (B)	49	ROTXL	B W L (B)			
25	DEC	B W L (B)	50	ROTXR	B W L (B)			

【注】　( ) ･･･････ **サイズ指定を省略した場合のデフォルトサイズ**

— ･･･････ **サイズ指定不可**

表 4.10 に、H8/300H アドバンストモードのアドレス形式を示します。

表 4.10　H8/300H アドバンストモードのアドレス形式

項番	アドレス形式	書き方
1	**レジスタ直接形式**	{ERn｜En｜Rn｜RnH｜RnL}
2	**レジスタ間接形式**	@ERn
3	**ディスプレースメント付レジスタ間接形式**	@(d[:{16｜24}],ERn)
4	**ポストインクリメントレジスタ間接形式**	@ERn+
5	**プリデクリメントレジスタ間接形式**	@-ERn
6	**絶対アドレス形式**	@aa[:{8｜16｜24}]
7	**イミディエートデータ形式**	#xx[:{8｜16｜32}]
8	**メモリ間接形式**	@@aa[:8]
9	**プログラムカウンタ相対形式** **（分岐命令のディスプレースメント）**	d[:{8｜16}]
10	**コントロールレジスタ**	CCR

【注】 1. n …… **レジスタ番号**（0～7）
d …… **ディスプレースメント**
aa …… **絶対アドレス**
xx …… **イミディエートデータ**
2. ER7 **は、**SP**（スタックポインタ）と同じです。**

【注】 FP**（フレームポインタ）は、レジスタニーモニックではありません。**
FP**（フレームポインタ）をレジスタとして記述したい場合は、**.REG **制御命令を使用してください。**

（b）H8/300H ノーマルモード

表 4.11 に、H8/300H ノーマルモードの実行命令のサイズを示します。

表 4.11　H8/300H **ノーマルモードの実行命令のサイズ**

項番	実行命令	サイズ	項番	実行命令	サイズ	項番	実行命令	サイズ
1	ADD	B W L (B)	26	DIVXS	B W (B)	51	RTE	—
2	ADDS	L (L)	27	DIVXU	B W (B)	52	RTS	—
3	ADDX	B (B)	28	EEPMOV	B W (B)	53	SHAL	B W L (B)
4	AND	B W L (B)	29	EXTS	W L (W)	54	SHAR	B W L (B)
5	ANDC	B (B)	30	EXTU	W L (W)	55	SHLL	B W L (B)
6	BAND	B (B)	31	INC	B W L (B)	56	SHLR	B W L (B)
7	Bcc	—	32	JMP	—	57	SLEEP	—
8	BCLR	B (B)	33	JSR	—	58	STC	B W (B)
9	BIAND	B (B)	34	LDC	B W (B)	59	SUB	B W L (B)
10	BILD	B (B)	35	MOV	B W L (B)	60	SUBS	L (L)
11	BIOR	B (B)	36	MOVFPE	B (B)	61	SUBX	B (B)
12	BIST	B (B)	37	MOVTPE	B (B)	62	TRAPA	—
13	BIXOR	B (B)	38	MULXS	B W (B)	63	XOR	B W L (B)
14	BLD	B (B)	39	MULXU	B W (B)	64	XORC	B (B)
15	BNOT	B (B)	40	NEG	B W L (B)			
16	BOR	B (B)	41	NOP	—			
17	BSET	B (B)	42	NOT	B W L (B)			
18	BSR	—	43	OR	B W L (B)			
19	BST	B (B)	44	ORC	B (B)			
20	BTST	B (B)	45	POP	W L (W)			
21	BXOR	B (B)	46	PUSH	W L (W)			
22	CMP	B W L (B)	47	ROTL	B W L (B)			
23	DAA	B (B)	48	ROTR	B W L (B)			
24	DAS	B (B)	49	ROTXL	B W L (B)			
25	DEC	B W L (B)	50	ROTXR	B W L (B)			

【注】　( ) ･･･････ **サイズ指定を省略した場合のデフォルトサイズ**

— ･･･････ **サイズ指定不可**

表 4.12 に、H8/300H ノーマルモードのアドレス形式を示します。

表 4.12　H8/300H ノーマルモードのアドレス形式

項番	アドレス形式	書き方
1	レジスタ直接形式	{ERn｜En｜Rn｜RnH｜RnL}
2	レジスタ間接形式	@ERn
3	ディスプレースメント付レジスタ間接形式	@(d[:16],ERn)
4	ポストインクリメントレジスタ間接形式	@ERn+
5	プリデクリメントレジスタ間接形式	@-ERn
6	絶対アドレス形式	@aa[:{8｜16}]
7	イミディエートデータ形式	#xx[:{8｜16｜32}]
8	メモリ間接形式	@@aa[:8]
9	プログラムカウンタ相対形式 （分岐命令のディスプレースメント）	d[:{8｜16}]
10	コントロールレジスタ	CCR

【注】 1. n …… レジスタ番号（0～7）
d …… ディスプレースメント
aa …… 絶対アドレス
xx …… イミディエートデータ
2. ER7 は、SP（スタックポインタ）と同じです。

【注】 FP（フレームポインタ）は、レジスタニーモニックではありません。
FP（フレームポインタ）をレジスタとして記述したい場合は、.REG 制御命令を使用してください。

（4）H8/300、H8/300L シリーズ

表 4.13 に、H8/300、H8/300L の実行命令のサイズを示します。

表 4.13　H8/300、H8/300L の実行命令のサイズ

項番	実行命令	サイズ	項番	実行命令	サイズ	項番	実行命令	サイズ
1	ADD	B W (B)	26	DIVXU	B (B)	51	SHLL	B (B)
2	ADDS	W (W)	27	EEPMOV	—	52	SHLR	B (B)
3	ADDX	B (B)	28	INC	B (B)	53	SLEEP	—
4	AND	B (B)	29	JMP	—	54	STC	B (B)
5	ANDC	B (B)	30	JSR	—	55	SUB	B W (B)
6	BAND	B (B)	31	LDC	B (B)	56	SUBS	W (W)
7	Bcc	—	32	MOV	B W (B)	57	SUBX	B (B)
8	BCLR	B (B)	33	MOVFPE	B (B)	58	XOR	B (B)
9	BIAND	B (B)	34	MOVTPE	B (B)	59	XORC	B (B)
10	BILD	B (B)	35	MULXU	B (B)			
11	BIOR	B (B)	36	NEG	B (B)			
12	BIST	B (B)	37	NOP	—			
13	BIXOR	B (B)	38	NOT	B (B)			
14	BLD	B (B)	39	OR	B (B)			
15	BNOT	B (B)	40	ORC	B (B)			
16	BOR	B (B)	41	POP	W (W)			
17	BSET	B (B)	42	PUSH	W (W)			
18	BSR	—	43	ROTL	B (B)			
19	BST	B (B)	44	ROTR	B (B)			
20	BTST	B (B)	45	ROTXL	B (B)			
21	BXOR	B (B)	46	ROTXR	B (B)			
22	CMP	B W (B)	47	RTE	—			
23	DAA	B (B)	48	RTS	—			
24	DAS	B (B)	49	SHAL	B (B)			
25	DEC	B (B)	50	SHAR	B (B)			

【注】　( ) ･･･････ サイズ指定を省略した場合のデフォルトサイズ

— ･･･････ サイズ指定不可

【注】 項番 33、34 の実行命令は、H8/300L シリーズでは、使用できません。

表 4.14 に、H8/300、H8/300L のアドレス形式を示します。

表 4.14　H8/300、H8/300L のアドレス形式

項番	アドレス形式	書き方
1	レジスタ直接形式	{Rn｜RnH｜RnL}
2	レジスタ間接形式	@Rn
3	ディスプレースメント付レジスタ間接形式	@(d[:16],Rn)
4	ポストインクリメントレジスタ間接形式	@Rn+
5	プリデクリメントレジスタ間接形式	@-Rn
6	絶対アドレス形式	@aa[:{8｜16}]
7	イミディエートデータ形式	#xx[:{8｜16}]
8	メモリ間接形式	@@aa[:8]
9	プログラムカウンタ相対形式 （分岐命令のディスプレースメント）	d[:8]
10	コントロールレジスタ	CCR

【注】 1. n …… レジスタ番号（0～7）
d …… ディスプレースメント
aa …… 絶対アドレス
xx …… イミディエートデータ
2. ER7 は、SP（スタックポインタ）と同じです。

【注】 FP（フレームポインタ）は、レジスタニーモニックではありません。
FP（フレームポインタ）をレジスタとして記述したい場合は、.REG 制御命令を使用してください。

5. アセンブラ制御命令

第5章　目次

5.1 アセンブラ制御命令

アセンブラ制御命令（以下、制御命令）は、アセンブラが解釈して実行する命令です。制御命令を記述するとアセンブラはその指示に従ってアセンブルします。

表 5.1 に、アセンブラ制御命令一覧を示します。

表 5.1 **アセンブラ制御命令一覧**

分類	制御命令	機能	参照
CPU に関するもの	.CPU	CPU の指定	5.2.1
セクション、ロケーションカウンタに関するもの	.SECTION	セクションの宣言	5.3.1
	.ORG	ロケーションカウンタ値の設定	5.3.2
	.ALIGN	ロケーションカウンタ値の境界調整	5.3.3
シンボルに関するもの	.EQU	シンボル値の設定	5.4.1
	.ASSIGN	シンボル値の設定	5.4.2
	.REG	レジスタ別名の設定	5.4.3
	.BEQU	ビットデータ名の設定	5.4.4
データ領域確保に関するもの	.DATA	整数データ確保	5.5.1
	.DATAB	整数データブロック確保	5.5.2
	.SDATA	文字列データ確保	5.5.3
	.SDATAB	文字列データブロック確保	5.5.4
	.SDATAC	計数付き文字列データ確保	5.5.5
	.SDATAZ	ゼロ終端文字列データ確保	5.5.6
	.RES	整数データ領域確保	5.5.7
	.SRES	文字列データ領域確保	5.5.8
	.SRESC	計数付き文字列データ領域確保	5.5.9
	.SRESZ	ゼロ終端文字列データ領域確保	5.5.10
外部参照、外部定義に関するもの	.IMPORT	外部参照シンボル宣言	5.6.1
	.EXPORT	外部定義シンボル宣言	5.6.2
	.GLOBAL	外部参照、外部定義シンボル宣言	5.6.3

分類	制御命令	機能	参照
オブジェクトモジュールに関するもの	.OUTPUT	オブジェクトの出力、デバッグ情報の出力	5.7.1
	.DEBUG	デバッグシンボル情報の出力	5.7.2
	.LINE	デバッグ情報のファイル名、行番号の変更	5.7.3
	.DISPSIZE	ディスプレースメントサイズの設定	5.7.4
アセンブルリストに関するもの	.PRINT	アセンブルリストの出力	5.8.1
	.LIST	ソースプログラムリストの部分出力	5.8.2
	.FORM	リストサイズの設定	5.8.3
	.HEADING	リストヘッディング	5.8.4
	.PAGE	改ページ	5.8.5
	.SPACE	空行出力	5.8.6
その他	.PROGRAM	オブジェクトモジュール名の設定	5.9.1
	.RADIX	基数の設定	5.9.2
	.END	ソースプログラムの終了	5.9.3

5.2 CPU に関する制御命令

5.2.1 .CPU（CPU の指定）

(1) 書式

ラベル	オペレーション	オペランド
×	.CPU	CPU 種別 [:アドレス空間のビット幅]

CPU 種別：{2600A｜2600N｜2000A｜2000N｜300HA｜300HN｜300｜300L}

CPU 種別	アドレス空間のビット幅	デフォルトサイズ
2600A	20, 24, 28, 32	24
2000A	20, 24, 28, 32	24
300HA	20, 24	24

(2) 説明

アセンブルするソースプログラムの対象とする CPU を指定します。
CPU 種別がアドバンストモードのみ、アドレス空間のビット幅が指定できます。
アクセス可能な範囲は、アドレス空間のビット幅の指定により異なります。
アセンブラは、指定した CPU 用にアセンブルします。

CPU 種別は、次のとおりです。

2600A:32、2600A:28、2600A[:24]、2600A:20・・・・・・・H8S/2600 アドバンストモード
2600N・・・H8S/2600 ノーマルモード
2000A:32、2000A:28、2000A[:24]、2000A:20・・・・・・・H8S/2000 アドバンストモード
2000N・・・H8S/2000 ノーマルモード
300HA[:24]、300HA:20・・・・・・・・・・・・・・・・・・・・・・・・・H8/300H アドバンストモード
300HN ・・H8/300H ノーマルモード
300 ・・・H8/300
300L・・H8/300L

本制御命令は、ソースプログラムの最初に記述してください。アセンブルリストに関する制御命令を除いて、ソースプログラムの最初にない場合は、エラーになります。

本制御命令は、1 回限り有効です。

本制御命令は、CPU コマンドラインオプションの指定がない場合に有効です。
本制御命令および CPU コマンドラインオプションを省略した場合は、H38CPU 環境変数で設定した CPU 種別が有効となります。

H38CPU 環境変数の使用については、**「11.2　H38CPU 環境変数」**を参照してください。

（3）例

```
.CPU            300HA:20

.SECTION A,CODE,ALIGN=2
MOV.W R0,R1
MOV.W R0,R2
```

H8/300H アドバンストモードの 1M モード用にアセンブルします。

5.3 セクション、ロケーションカウンタに関する制御命令

5.3.1 .SECTION（セクションの宣言）

(1) 書式

ラベル	オペレーション	オペランド
×	.SECTION	セクション名［,セクション属性［,形式種別］］

セクション属性：{CODE｜DATA｜STACK｜COMMON｜DUMMY}
形式種別：{LOCATE=先頭アドレス｜ALIGN=境界調整数}

(2) 説明

セクションの開始、再開を宣言します。

(a) セクションの開始

セクションを開始し、セクション名、セクション属性、形式種別を設定します。

■セクション名

セクション名を設定します。セクション名の書き方は、シンボル名の書き方と同じです。また、セクション名は大文字と小文字を区別します。

■セクション属性

セクションの属性を設定します。セクション属性は、次のとおりです。

CODE・・・・・・・・・コードセクション
DATA・・・・・・・・・データセクション
STACK・・・・・・・・スタックセクション
COMMON・・・・・コモンセクション
DUMMY ・・・・・・ダミーセクション

セクション属性を省略した場合は、CODE を設定します。

■形式種別

形式種別を設定します。形式種別は、次のとおりです。

LOCATE=先頭アドレス ・・・・・・絶対アドレス形式
ALIGN=境界調整数・・・・・・・・・・相対アドレス形式

形式種別を省略した場合は、ALIGN=2 を設定します。

・絶対アドレス形式

絶対アドレス形式では、セクションの先頭アドレスを設定します。

先頭アドレスは、後方参照の絶対値で指定します。

先頭アドレスの最大値は、次のとおりです。

（1）H8S/2600 アドバンストモード

2600A:32 ・・・・・・・・・・・・・・・・・ H'FFFFFFFF
2600A:28 ・・・・・・・・・・・・・・・・・ H'0FFFFFFF
2600A[:24]・・・・・・・・・・・・・・・・ H'00FFFFFF
2600A:20 ・・・・・・・・・・・・・・・・・ H'000FFFFF

（2）H8S/2600 ノーマルモード

2600N・・・・・・・・・・・・・・・・・・・ H'0000FFFF

（3）H8S/2000 アドバンストモード

2000A:32 ・・・・・・・・・・・・・・・・・ H'FFFFFFFF
2000A:28 ・・・・・・・・・・・・・・・・・ H'0FFFFFFF
2000A[:24]・・・・・・・・・・・・・・・・ H'00FFFFFF
2000A:20 ・・・・・・・・・・・・・・・・・ H'000FFFFF

（4）H8S/2000 ノーマルモード

2000N・・・・・・・・・・・・・・・・・・・ H'0000FFFF

（5）H8/300H アドバンストモード

300HA[:24] ・・・・・・・・・・・・・・・・ H'00FFFFFF
300HA:20・・・・・・・・・・・・・・・・・・ H'000FFFFF

（6）H8/300H ノーマルモード

300HN ・・・・・・・・・・・・・・・・・・・ H'0000FFFF

（7）H8/300

300 ・・・・・・・・・・・・・・・・・・・・・ H'0000FFFF

（8）H8/300L

300L・・・・・・・・・・・・・・・・・・・・・ H'0000FFFF

・相対アドレス形式

相対アドレス形式では、セクションの境界調整数を設定します。

リンケージエディタでは、オブジェクトモジュールを連結するときに相対アドレスセクションの先頭アドレスを境界調整数の倍数に補正します。

境界調整数は、後方参照の絶対値で指定します。

境界調整数には、2^nの値が指定できます。

本制御命令で、セクションを宣言しなかった場合は、デフォルトセクションとして次のように設定します。

```
.SECTION P,CODE,ALIGN=2
```

(b) セクションの再開

ソースプログラム中にすでに存在するセクションを再開します。

セクションの再開では、すでに存在するセクションのセクション名を指定します。セクション属性と形式種別は、最初に宣言したものを継続します。

(3) 例

```
          .SECTION A,CODE,ALIGN=2 ····· [1]
          MOV.W   R0,R1
          .SECTION B,DATA,LOCATE=H'1000 ···· [2]
DATA1
          .DATA.W H'0001
          .SECTION A····· [3]
          MOV.W   R0,R3
```

[1] セクション A を開始します。セクション名は A、セクション属性はコードセクション、形式種別は相対アドレス形式で境界調整数は 2 です。

[2] セクション B を開始します。セクション名は B、セクション属性はデータセクション、形式種別は絶対アドレス形式で先頭アドレスは H'1000 番地です。

[3] セクション A を再開します。セクション名は A、セクション属性はコードセクション、形式種別は相対アドレス形式で境界調整数は 2 です。

5.3.2 .ORG（ロケーションカウンタ値の設定）

（1）書式

ラベル	オペレーション	オペランド
×	.ORG	ロケーションカウンタ値

（2）説明

セクション内のロケーションカウンタ値を指定した値に変更します。

ロケーションカウンタ値は、後方参照の絶対値、または後方参照の相対アドレスセクションのアドレス値で指定します。

ロケーションカウンタ値の最大値は、次のとおりです。

（1）H8S/2600 アドバンストモード

2600A:32 ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'FFFFFFFF
2600A:28 ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'0FFFFFFF
2600A[:24] ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'00FFFFFF
2600A:20 ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'000FFFFF

（2）H8S/2600 ノーマルモード

2600N‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'0000FFFF

（3）H8S/2000 アドバンストモード

2000A:32 ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'FFFFFFFF
2000A:28 ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'0FFFFFFF
2000A[:24] ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'00FFFFFF
2000A:20 ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'000FFFFF

（4）H8S/2000 ノーマルモード

2000N‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'0000FFFF

（5）H8/300H アドバンストモード

300HA[:24]‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'00FFFFFF
300HA:20‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'000FFFFF

（6）H8/300H ノーマルモード

300HN ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'0000FFFF

（7）H8/300

300 ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'0000FFFF

（8）H8/300L

300L ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ H'0000FFFF

また、絶対アドレスセクションで指定する場合は、ロケーションカウンタ値はセクションの先頭アドレス以降の値で指定します。

本制御命令を絶対アドレスセクションで指定した場合は、設定したロケーションカウンタ値は絶対アドレスになり、相対アドレスセクションで指定した場合は、相対アドレスになります。

(3) 例

```
          .SECTION A,DATA,LOCATE=H'0100
DATA1
          .DATA.W H'0001
          .DATA.W H'0002
          .ORG    H'0200 ·····  [1]
DATA2
          .DATA.W H'0003
          .DATA.W H'0004
```

[1] ロケーションカウンタ値を絶対 H'0200 番地に変更します。

```
          .SECTION B,DATA,ALIGN=2
DATA3
          .DATA.W H'0001
          .DATA.W H'0002
          .ORG    H'0300 ·····  [2]
DATA4
          .DATA.W H'0003
          .DATA.W H'0004
```

[2] ロケーションカウンタ値をセクション A の相対 H'0300 番地に変更します。

5.3.3 .ALIGN（ロケーションカウンタ値の境界調整）

（1）書式

ラベル	オペレーション	オペランド
×	.ALIGN	**境界調整数**

（2）説明

セクション内のロケーションカウンタ値を境界調整数の倍数に補正します。

境界調整数は、後方参照の絶対値で指定します。
境界調整数には、2^nの値が指定できます。
境界調整数は、セクション先頭からの相対アドレスで行います。
境界調整数は、アドレス空間の指定により異なります。

（3）例

```
            .CPU 2600A
            .SECTION A,DATA,ALIGN=2 ····· [1]
DATA1
            .DATA.W H'0001
DATA2
            .DATA.B H'02 ·················· [2]
            .ALIGN  2····················· [3]
DATA3
            .DATA.W H'0003
            .END
```

[1] オブジェクトモジュールを連結する際に、相対アドレスセクションの先頭アドレスを 2 の倍数番地（偶数番地）に補正します。

[2] 1 バイトのデータを確保するため、次のデータのロケーションカウンタ値が奇数番地になります。

[3] ロケーションカウンタ値を 2 の倍数番地（偶数番地）に補正しています。

5.4 シンボルに関する制御命令

5.4.1 .EQU（シンボル値の設定）

(1) 書式

ラベル	オペレーション	オペランド
×	.EQU	値

(2) 説明

シンボルに値を設定します。
値は、後方参照の絶対値、後方参照のアドレス値、または外部参照値*で指定します。
本制御命令で定義したシンボルの値は、変更できません。

(3) 例

```
SYM1        .EQU    1
SYM2        .EQU    2

            .SECTION A,CODE,ALIGN=2
            MOV.B   #SYM1:8,R0L ・・・・・・・・・・ MOV.B #1:8,R0L と同じです。
            MOV.B   #SYM2:8,R1L ・・・・・・・・・・ MOV.B #2:8,R1L と同じです。
```

シンボル SYM1 に 1、SYM2 に 2 を設定しています。

【注】 * 外部参照値、外部参照値＋定数、外部参照値ー定数を記述できます。外部参照値を用いて定義した.EQU 名を外部定義（EXPORT）やデバッガで参照することはできません。

5.4.2 .ASSIGN（シンボル値の設定）

（1）書式

ラベル	オペレーション	オペランド
シンボル名	.ASSIGN	**値**

（2）説明

シンボルに値を設定します。

値は、後方参照の絶対値、または後方参照のアドレス値で指定します。

本制御命令で定義したシンボルは、再び本制御命令で指定することにより値を変更できます。

本制御命令で定義したシンボルを参照した場合の値は、次の例のようになります。

■例

```
        MOV.B   #SYM:8,R0L ···· SYM の値は 0 です。
SYM     .ASSIGN 1
        MOV.B   #SYM:8,R1L ···· SYM の値は 1 です。
SYM     .ASSIGN 2
        MOV.B   #SYM:8,R2L ···· SYM の値は 2 です。
```

本制御命令で定義したシンボルは、外部定義できません。

本制御命令で定義したシンボルは、シミュレータ・デバッガで参照できません。

（3）例

```
SYM1    .ASSIGN 1
SYM2    .ASSIGN 2
        .SECTION A,CODE,ALIGN=2
        MOV.B   #SYM1:8,R0L·····MOV.B #1:8,R0L と同じです。
        MOV.B   #SYM2:8,R1L·····MOV.B #2:8,R1L と同じです。
SYM2    .ASSIGN 3
        MOV.B   #SYM1:8,R2L·····MOV.B #1:8,R2L と同じです。
        MOV.B   #SYM2:8,R3L·····MOV.B #3:8,R3L と同じです。
```

シンボル SYM1 に 1、SYM2 に 2 を設定しています。SYM2 は値を 3 に変更しています。

5.4.3 .REG（レジスタ別名の設定）

（1）書式

ラベル	オペレーション	オペランド
シンボル名	.REG	（レジスタ名）

（2）説明

レジスタ名に別名をつけます。

レジスタ名は、次のとおりです。

・単一レジスタ　1 つのレジスタにレジスタ別名をつけます。レジスタを指定できる箇所にはすべて指定できます。レジスタ名には、汎用レジスタを指定できます。

・複数レジスタ　2 つ以上のレジスタにレジスタ別名をつけます。ただし、CPU 種別が H8S/2600 シリーズ、H8S/2000 シリーズに限ります。LDM 命令、STM 命令と本制御命令のオペランドに指定できます。レジスタ名には、32 ビット汎用レジスタを指定できます。

レジスタ名の書き方は、次のとおりです。

表 5.2　レジスタ別名の設定

指定方法	説明	使用例
単一レジスタ	R0～R7,E0～E7,ER0～ER7 **のうち 1 つを指定します。**	SINGLREG .REG (R0) **レジスタ** R0 **に** SINGLREG **という別名をつけています。**
複数レジスタ	**ハイフン（-）で区切って、範囲の形式で一度に複数のレジスタを指定します。左側のレジスタの番号より右側のレジスタの番号が小さいとエラーとなり、本制御命令を無視します。**	RNG1 .REG (ER0-ER3) **4 個のレジスタ** ER0,ER1,ER2, ER3 **に** RNG1 **という別名をつけています。** RNG2 .REG (ER3-ER0) **右側の番号の方が小さいのでエラーです。**＊
レジスタ別名の再指定	**あらかじめ定義したレジスタ別名をオペランドに指定します。**	ER00 .REG (ER0-ER3) ER01 .REG (ER00) **4 個のレジスタ** ER0～ER3 **に** ER01 **という別名をつけています。**

【注】　本制御命令で定義したレジスタ別名は、再定義できません。

本制御命令で定義したレジスタ別名は、本制御命令以降のソースステートメントに対して有効です。

本制御命令で定義したレジスタ別名は、外部定義できません。

本制御命令で定義したシンボルは、シミュレータ・デバッガで参照できません。

＊　**指定できるレジスタの組み合わせは、次のとおりです。**

(ER0-ER1)、(ER2-ER3)、(ER4-ER5)、(ER6-ER7)、(ER0-ER2)、(ER4-ER6)、(ER0-ER3)、(ER4-ER7)

（3）例

```
        .CPU 2600A
RLST1:  .REG (R0)·····························[1]
RLST2:  .REG (ER0-ER2)·····················[2]
        MOV.W   RLST1,@R6
        LDM.L   @SP+,(RLST2)
        STM.L   (RLST2),@-SP
```

[1] R0 のレジスタを RLST1 に指定します。
[2] ER0、ER1、ER2 の 3 個のレジスタを RLST2 に指定します。

5.4.4 .BEQU（ビットデータ名の設定）

（1）書式

ラベル	オペレーション	オペランド
シンボル名	.BEQU	**ビット番号,置換シンボル名**

（2）説明

ビット操作命令の対象となるメモリ上の 1 ビットデータに名称をつけます。

ビットデータ名は、ビット操作命令のオペランドに指定できます。指定されたビットデータ名は、#xx,@aa 形式に置換されます。

■例

```
BIT0    .BEQU   0,ADDRESS ・・・・ビットデータ名を定義しています。
        BSET.B  BIT0:8 ・・・・・・・・ビット操作命令のオペランドに指定しています。
```

BSET.B BIT0:8 は、BSET.B #0,@ADDRESS:8 としてコード生成します。

ビット番号は、後方参照の絶対値で指定します。
ビット番号には、0～7 の値が指定できます。

CPU 種別が H8S/2600 シリーズ、H8S/2000 シリーズの場合
置換シンボル名は、8 ビット絶対アドレス形式（@aa:8）、16 ビット絶対アドレス形式（@aa:16）、32 ビット絶対アドレス形式（@aa:32）に指定できるシンボルを指定します。

CPU 種別が H8/300H シリーズ、H8/300 シリーズ、H8/300L シリーズの場合
置換シンボル名は、8 ビット絶対アドレス形式（@aa:8）に指定できるシンボルを指定します。

ビットデータ名を指定できるビット操作命令は、次のとおりです。

BSET、BCLR、BNOT、BTST、BAND、BIAND、BOR、BIOR、BXOR、BIXOR、BLD、BILD、BST、BIST

本制御命令で定義したビットデータ名は、本制御命令以降のソースステートメントに対して有効です。

本制御命令で定義したビットデータ名は、外部定義できません。
本制御命令で定義したビットデータ名は、シミュレータ・デバッガで参照できません。

(3) 例

```
        .CPU    2600A:32
AD1     .EQU    H'FFFFFF00
AD2     .EQU    H'FFFF8000
AD3     .EQU    H'FF000000
AD1B0   .BEQU   0,AD1
AD1B1   .BEQU   1,AD1
AD2B2   .BEQU   2,AD2
AD2B3   .BEQU   3,AD2
AD3B4   .BEQU   4,AD3
AD3B5   .BEQU   5,AD3

        .SECTION A,CODE,ALIGN=2
        BSET.B  AD1B0 ····· BSET.B #0,@AD1:8 と同じです。
        BSET.B  AD1B1 ····· BSET.B #1,@AD1:8 と同じです。
        BCLR.B  AD2B2 ····· BCLR.B #2,@AD2:16 と同じです。
        BCLR.B  AD2B3 ····· BCLR.B #3,@AD2:16 と同じです。
        BNOT.B  AD3B4 ····· BNOT.B #4,@AD3:32 と同じです。
        BNOT.B  AD3B5 ····· BNOT.B #5,@AD3:32 と同じです。
```

ビットデータ名の内容は、次のとおりです。

```
AD1B0····· H'FFFFFF00 番地のビット 0
AD1B1····· H'FFFFFF00 番地のビット 1
AD2B2····· H'FFFF8000 番地のビット 2
AD2B3····· H'FFFF8000 番地のビット 3
AD3B4····· H'FF000000 番地のビット 4
AD3B5····· H'FF000000 番地のビット 5
```

5.5 データ領域確保に関する制御命令

5.5.1 .DATA（整数データ確保）

（1）書式

ラベル	オペレーション	オペランド
[**シンボル名**]	.DATA[.s]	**整数データ**[,**整数データ**…]

s（サイズ）：{B｜W｜L}

（2）説明

整数データを指定したサイズに従って確保します。

サイズは、次のとおりです。

B・・・・・・・バイト　　（1バイト）
W ・・・・・・ワード　　（2バイト）
L ・・・・・・・ロングワード（4バイト）

下線部は、指定を省略した場合の設定です。

サイズによって指定できる整数データの値は、次のとおりです。

B・・・・・・・–128～255
W ・・・・・・–32,768～65,535
L ・・・・・・・–2,147,483,648～4,294,967,295

整数データは、アドレス空間の指定により異なります。

（3）例

```
.SECTION A,DATA,ALIGN=2
.DATA.W H'0102,H'0304
.DATA.B H'05,H'06,H'07,H'08
```

次のようにデータを確保します。

01	02	03	04	05	06	07	08

5.5.2 .DATAB（整数データブロック確保）

（1）書式

ラベル	オペレーション	オペランド
[**シンボル名**]	.DATAB[.s]	**ブロック数**,**整数データ**

s（サイズ）：{B｜W｜L}

（2）説明

ブロック数分の整数データを指定したサイズに従って確保します。

サイズは、次のとおりです。

B ・・・・・バイト　　（1バイト）
W・・・・・ワード　　（2バイト）
L・・・・・・ロングワード（4バイト）

下線部は、指定を省略した場合の設定です。

ブロック数は、後方参照の絶対値で指定します。

ブロック数には、1以上の値が指定できます。

サイズによって指定できる整数データの値は、次のとおりです。

B ・・・・・–128～255
W・・・・・–32,768～65,535
L・・・・・・–2,147,483,648～4,294,967,295

ブロック数は、アドレス空間の指定により異なります。

（3）例

```
.SECTION A,DATA,ALIGN=2
.DATAB.W 2,H'0102
.DATAB.B 2,H'03
```

次のようにデータを確保します。

01	02	01	02	03	03

5.5.3 .SDATA（文字列データ確保）

（1）書式

ラベル	オペレーション	オペランド
[**シンボル名**]	.SDATA	"**文字列**"[,"**文字列**"…]

（2）説明

文字列データを確保します。

文字列は、文字をダブルコーテーション（"）で囲んで指定します。
ダブルコーテーション（"）自体を文字として指定する場合は、2つ続けて記述します。

文字列に制御コードを指定する場合は、ダブルコーテーション（"）で囲んだ文字列の直後に制御コードをアングルブラケット（<>）で囲んで記述します。

"文字列"<制御コード>

制御コードは、後方参照の絶対値で指定します。

（3）例

```
.SECTION A,DATA,ALIGN=2
.SDATA  "ABC""DEF"
.SDATA  "GHI"＜H'0D＞＜H'0A＞
```

次のようにデータを確保します。

A	B	C	"	D	E	F	G	H	I	0D	0A

5.5.4 .SDATAB（文字列データブロック確保）

（1）書式

ラベル	オペレーション	オペランド
[**シンボル名**]	. SDATAB	**ブロック数**,"**文字列**"

（2）説明

ブロック数分の文字列データを確保します。

ブロック数は、後方参照の絶対値で指定します。
ブロック数には、1 以上の値が指定できます。
ブロック数は、アドレス空間の指定により異なります。

文字列は、文字をダブルコーテーション（"）で囲んで指定します。
ダブルコーテーション（"）自体を文字として指定する場合は、2 つ続けて記述します。

文字列に制御コードを指定する場合は、ダブルコーテーション（"）で囲んだ文字列の直後に制御コードをアングルブラケット（<>）で囲んで記述します。

"文字列"<制御コード>

制御コードは、後方参照の絶対値で指定します。

（3）例

```
.SECTION A,DATA,ALIGN=2
.SDATAB 2,"A""B"
.SDATAB 2,"EF"<H'00>
```

次のようにデータを確保します。

A	"	B	A	"	B	E	F	00	E	F	00

5.5.5 .SDATAC（計数付き文字列データ確保）

（1）書式

ラベル	オペレーション	オペランド
[**シンボル名**]	.SDATAC	"**文字列**"[,"**文字列**"…]

（2）説明

計数付き文字列データを確保します。

文字列データを確保する際に、文字列データの先頭に1バイトの計数（文字列のバイト数を示すデータ）を付加して確保します。計数には、計数自体の1バイトは含みません。

文字列は、文字をダブルコーテーション（"）で囲んで指定します。
ダブルコーテーション（"）自体を文字として指定する場合は、2つ続けて記述します。

文字列に制御コードを指定する場合は、ダブルコーテーション（"）で囲んだ文字列の直後に制御コードをアングルブラケット（<>）で囲んで記述します。

"文字列"<制御コード>

制御コードは、後方参照の絶対値で指定します。

（3）例

```
.SECTION A,DATA,ALIGN=2
.SDATAC "AB""CDE","FG"<H'0D><H'0A>
```

次のようにデータを確保します。

06	A	B	"	C	D	E	04	F	G	0D	0A

5.5.6 .SDATAZ（ゼロ終端文字列データ確保）

（1）書式

ラベル	オペレーション	オペランド
[**シンボル名**]	.SDATAZ	"**文字列**"[,"**文字列**"…]

s（サイズ）：{B｜W｜L}

（2）説明

ゼロ終端文字列データを確保します。

文字列データを確保する際に、文字列データの末尾に1バイトの0を付加して確保します。

文字列は、文字をダブルコーテーション（"）で囲んで指定します。
ダブルコーテーション（"）自体を文字として指定する場合は、2つ続けて記述します。

文字列に制御コードを指定する場合は、ダブルコーテーション（"）で囲んだ文字列の直後に制御コードをアングルブラケット（<>）で囲んで記述します。

"文字列"<制御コード>

制御コードは、後方参照の絶対値で指定します。

（3）例

```
.SECTION A,DATA,ALIGN=2
.SDATAZ "AB""CD","EFG"＜H'0D＞＜H'0A＞
```

次のようにデータを確保します。

A	B	"	C	D	00	E	F	G	0D	0A	00

5.5.7 .RES（整数データ領域確保）

（1）書式

ラベル	オペレーション	オペランド
[**シンボル名**]	.RES[.s]	**領域数**

s（サイズ）：｛B｜W｜L｝

（2）説明

整数データの領域を確保します。

指定したサイズの整数データ領域を領域数分だけ確保します。

サイズは次のとおりです、

B ・・・・・バイト　　（1 バイト）
W・・・・・ワード　　（2 バイト）
L・・・・・・ロングワード（4 バイト）

下線部は、指定を省略した場合の設定です。

領域数は、後方参照の絶対値で指定します。
領域数には、1 以上の値が指定できます。
領域数は、アドレス空間の指定により異なります。

（3）例

```
.SECTION A,DATA,ALIGN=2
.RES.W  10
.RES.B  255
```

20 バイトの領域と 255 バイトの領域を確保します。

5.5.8 .SRES（文字列データ領域確保）

（1）書式

ラベル	オペレーション	オペランド
[**シンボル名**]	.SRES	**領域サイズ**[,**領域サイズ…**]

（2）説明

文字列データの領域を確保します。

指定した領域サイズ（バイト数）分の領域を確保します。

領域サイズは、後方参照の絶対値で指定します。
領域サイズには、1 以上の値が指定できます。
領域サイズは、アドレス空間の指定により異なります。

（3）例

```
.SECTION A,DATA,ALIGN=2
.SRES   10,20,30
.SRES   255
```

10 バイト、20 バイト、30 バイトの領域と 255 バイトの領域を確保します。

5.5.9 .SRESC（計数付き文字列データ領域確保）

（1）書式

ラベル	オペレーション	オペランド
[**シンボル名**]	.SRESC	**領域サイズ**[,**領域サイズ…**]

（2）説明

計数付き文字列データの領域を確保します。

指定した領域サイズ（バイト数）に、計数の 1 バイトを加えた領域を確保します。

領域サイズは、後方参照の絶対値で指定します。
領域サイズには、0～255 の値が指定できます。

（3）例

```
.SECTION A,DATA,ALIGN=2
.SRESC  10,20,30
.SRESC  255
```

11 バイト、21 バイト、31 バイトの領域と 256 バイトの領域を確保します。

5.5.10 .SRESZ（ゼロ終端文字列データ領域確保）

（1）書式

ラベル	オペレーション	オペランド
[**シンボル名**]	.SRESZ	**領域サイズ**[,**領域サイズ…**]

（2）説明

ゼロ終端文字列データの領域を確保します。

指定した領域サイズ（バイト数）に、ゼロ終端の 1 バイトを加えた領域を確保します。

領域サイズは、後方参照の絶対値で指定します。
領域サイズには、0～255 の値が指定できます。

（3）例

```
.SECTION A,DATA,ALIGN=2
.SRESZ  10,20,30
.SRESZ                      255
```

11 バイト、21 バイト、31 バイトの領域と 256 バイトの領域を確保します。

5.6 外部参照、外部定義に関する制御命令

5.6.1 .IMPORT（外部参照シンボル宣言）

（1）書式

ラベル	オペレーション	オペランド
×	.IMPORT	**シンボル名**［:8｜:16］［,**シンボル名**［:8｜:16］…］

（2）説明

外部参照シンボルを宣言します。

別ソースプログラム内で定義したシンボルを参照する場合に宣言します。

ソースプログラム内で定義したシンボルは、外部参照シンボルの宣言はできません。

また、シンボル名にアクセスサイズ（:8または:16）を付けることにより、当該シンボルを 8 または 16 ビット絶対アドレス形式でアクセスします。 ただし、アクセスサイズ指定は、後方参照シンボルのみ有効となり、前方参照シンボルはアクセスサイズ指定がないものとして扱います。

本制御命令による宣言は、アクセスサイズの有無に関わらず最初に指定したシンボルを有効とし、2 度目以降の宣言は無効となります。

シンボルを外部参照するには、外部参照シンボル宣言に対応して、そのシンボルが定義してある別ソースプログラムで外部定義シンボル宣言（.EXPORT）する必要があります。

（3）例

```
        .CPU    2600A
        .IMPORT PROG·····PROG を外部参照シンボル宣言しています。
        .IMPORT sym1:8·····sym1 を 8 ビット絶対アドレスの外部参照シンボル宣言しています。
        .SECTION A,CODE,ALIGN=2
START
        MOV.L   #STACK:32,SP
        MOV.B   @sym1:8,ROL···· 8 ビット絶対アドレス形式で参照しています。
        JSR     @PROG:24····· PROG を参照しています。
        SLEEP
        BRA     START
        .SECTION B,STACK,ALIGN=2
        .RES.B  H'500
STACK
        .END    START
```

5.6.2 .EXPORT（外部定義シンボル宣言）

（1）書式

ラベル	オペレーション	オペランド
×	.EXPORT	**シンボル名**［:8｜:16］［,**シンボル名**［:8｜:16］…］

（2）説明

外部定義シンボルを宣言します。

ソースプログラム内で定義したシンボルが、別ソースプログラムから参照される場合に宣言します。

外部定義シンボルに指定できるシンボルは、次のとおりです。

・絶対値を持つシンボル

・アドレス値を持つシンボル

ただし、.ASSIGN 制御命令で定義したシンボル、ダミーセクションのアドレス値を持つシンボルは指定できません。

また、シンボル名にアクセスサイズ（:8または:16）を付けることにより、当該シンボルを 8 または 16 ビット絶対アドレス形式でアクセスします。ただし、アクセスサイズ指定は後方参照シンボルのみ有効となり、前方参照シンボルはアクセスサイズ指定がないものとして扱います。本制御命令による宣言は、アクセスサイズの有無に関わらず、最初に指定したシンボルを有効とし、2 度目以降の宣言は無効となります。

シンボルを外部参照させるには､外部定義シンボル宣言に対応して､そのシンボルを参照する別ソースプログラムで外部参照シンボル宣言（.IMPORT）する必要があります。

（3）例

```
          .CPU    2600A
          .EXPORT PROG ·····PROG を外部定義シンボル宣言しています。
          .EXPORT sym1:16·····sym1 を 16 ビット絶対アドレスの外部定義シ
                                ンボル宣言しています。
          .SECTION A,CODE,ALIGN=2
PROG  ·····PROG を定義しています。
          MOV.L   ER0,ER1
          MOV.L   ER0,ER3
          RTS
sym1 ·····sym1 を定義
          .RES.B  1
          .END
```

5.6.3 .GLOBAL（外部参照、外部定義シンボル宣言）

(1) 書式

ラベル	オペレーション	オペランド
×	.GLOBAL	**シンボル名**［:8｜:16］［,**シンボル名**［:8｜:16］…］

(2) 説明

外部参照シンボル、外部定義シンボルを宣言します。

別ソースプログラム内で定義したシンボルを参照する場合と、ソースプログラム内で定義したシンボルが、別ソースプログラムから参照される場合に宣言します。

本制御命令では、ソースプログラム内で定義していないシンボルを外部参照シンボルとし、ソースプログラム内で定義しているシンボルを外部定義シンボルとします。

外部定義シンボルに指定できるシンボルは、次のとおりです。

・絶対値を持つシンボル

・アドレス値を持つシンボル

ただし、.ASSIGN 制御命令で定義したシンボル、ダミーセクションのアドレス値を持つシンボルは指定できません。

また、シンボル名にアクセスサイズ（:8または:16）を付けることにより、当該シンボルを 8 または 16 ビット絶対アドレス形式でアクセスします。ただし、アクセスサイズ指定は、後方参照シンボルのみ有効となり、前方参照シンボルはアクセスサイズ指定がないものとして扱います。

本制御命令による宣言は、アクセスサイズの有無に関わらず、最初に指定したシンボルを有効とし、2 度目以降の宣言は無効となります。

(3) 例

```
        .CPU    2600A
        .GLOBAL PROG2・・・・・・・・・・・・外部参照シンボルとして宣言しています。
        .GLOBAL PROG3・・・・・・・・・・・・外部定義シンボルとして宣言しています。
;
        .SECTION C,CODE,ALIGN=2
PROG2 ・・・・・・・・・・・・・・・・・・・・・・・・・・・PROG2 を定義しています。
        MOV.L   ER0,ER1
        JSR     @PROG3:24・・・・・・・・PROG3 を参照しています。
        MOV.L   ER1,ER2
        RTS
;
        .END
```

5.7 オブジェクトモジュールに関する制御命令

5.7.1 .OUTPUT（オブジェクトの出力、デバッグ情報の出力）

（1）書式

ラベル	オペレーション	オペランド
×	.OUTPUT	**出力種別**[,**出力種別**]

出力種別：{OBJ｜NOOBJ｜DBG｜NODBG}

（2）説明

オブジェクトモジュールの出力、デバッグ情報の出力を指定します。

（1）オブジェクトモジュールの出力

オブジェクトモジュールの出力、出力抑止を指定します。

出力種別は、次のとおりです。

OBJ‥‥‥‥‥出力

NOOBJ‥‥‥‥出力抑止

指定を省略した場合は、OBJ を設定します。

本指定は、1 回限り有効です。

本指定は、[NO] OBJECT コマンドラインオプションの指定がない場合に有効です。

（2）デバッグ情報の出力

デバッグ情報の出力、出力抑止を指定します。

出力種別は、次のとおりです。

DBG ‥‥‥‥‥出力

NODBG ‥‥‥出力抑止

指定を省略した場合は、NODBG を設定します。

本指定は、1 回限り有効です。

本指定は、[NO] DEBUG コマンドラインオプションの指定がない場合に有効です。

本指定は、オブジェクトモジュールの出力時に有効です。

(3) 例

```
.OUTPUT OBJ,DBG
```

オブジェクトモジュールを出力、デバッグ情報を出力します。

5.7.2 .DEBUG（デバッグシンボル情報の出力）

(1) 書式

ラベル	オペレーション	オペランド
×	.DEBUG	**出力種別**

出力種別：{ON｜OFF}

(2) 説明

デバッグ情報中のデバッグシンボル情報の出力、出力抑止を指定します。

ソースプログラム中のシンボルのうち、デバッグに必要なものだけを出力したい場合に使用します。

出力種別は、次のとおりです。

ON ・・・・・・・・・・・・・出力

OFF・・・・・・・・・・・・・出力抑止

出力種別の初期設定は、ONです。

本制御命令は何回でも指定でき、指定内容は本制御命令以降のソースステートメントに対して有効です。

本制御命令は、デバッグ情報の出力時に有効です。

(3) 例

```
        .OUTPUT DBG
        .DEBUG  OFF
        .SECTION A,CODE,ALIGN=2
START
        MOV.W   @DAT:16,R1

        .SECTION B,DATA,ALIGN=2
        .DEBUG  ON
DAT
        .DATA.W H'1234
```

シンボル DAT をデバッグシンボル情報に出力します。

5.7.3 .LINE（デバッグ情報のファイル名、行番号の変更）

（1）書式

ラベル	オペレーション	オペランド
×	.LINE	["**ファイル名**",]**行番号**

（2）説明

本制御命令は、デバッグ情報のファイル名、行番号の変更を行います。
本制御命令は、C/C++ソースレベルのデバッグを支援します。

C/C++コンパイラは、アセンブリソースプログラムを出力した際、本制御命令を埋め込みます。
これによりコンパイラが出力したアセンブリソースプログラムをアセンブルしても、C/C++ソースレベルのデバッグができます。
本制御命令を指定した次の行からアセンブラが管理するファイル名、行番号は、本制御命令で指定した内容に切り替わります。
本制御命令で指定したファイル名、行番号は本制御命令が記述されているファイル内でのみ有効です。

(3) 例

```
> ch38 -cpu=2600a –code=asmcode –debug aaa.c
```

Cソースプログラム（aaa.c）

```
int func()                  /*1*/
{                           /*2*/
    int i,j;                /*3*/
    j=0;                    /*4*/
    for(i=1;i<=10;i++)      /*5*/
        j+=i;               /*6*/
    return(j);              /*7*/
}                           /*8*/
```

→

アセンブリソースプログラム

```
      .CPU     2600A
      .EXPORT  _func
      .SECTION P,CODE,ALIGN=2
      .LINE    "/asm/code/aaa.c",1
_func:            ;function:func
      .LINE    2
      .LINE    4
      SUB.W    E0,E0
      .LINE    5
      MOV.W    #1:16,R0
      .LINE    5
L5:
      .LINE    6
      ADD.W    R0,E0
      .LINE    5
      INC.W    #1,R0
      CMP.W    #10:16,R0
      BLE      L5:8
      .LINE    7
      MOV.W    E0,R0
      .LINE    8
      RTS
      .END
```

図 5.1　.LINE **使用例**

5.7.4 .DISPSIZE（ディスプレースメントサイズの設定）

（1）書式

ラベル	オペレーション	オペランド
×	.DISPSIZE	対象項目＝ビット数［,対象項目＝ビット数…］

対象項目：{FBR｜XBR｜FRG｜XRG｜FWD｜XTN｜ALL}

（2）説明

分岐命令のディスプレースメント、ディスプレースメント付レジスタ間接形式のディスプレースメントが前方参照値、外部参照値の場合のディスプレースメントのデフォルトサイズを設定します。

本制御命令の対象となるのは、ディスプレースメントサイズ（:8、:16、:24、:32）の指定がないディスプレースメントです。

対象項目は、次のとおりです。

FBR ・・・・・・・・・・・・・前方参照の分岐命令
XBR ・・・・・・・・・・・・・外部参照の分岐命令
FRG ・・・・・・・・・・・・・前方参照のディスプレースメント付レジスタ間接形式
XRG ・・・・・・・・・・・・・外部参照のディスプレースメント付レジスタ間接形式
FWD・・・・・・・・・・・・・FBR、FRG の同時指定
XTN ・・・・・・・・・・・・・XBR、XRG の同時指定
ALL ・・・・・・・・・・・・・FBR、XBR、FRG、XRG の同時指定

ビット数は、後方参照の絶対値で指定します。

ビット数は、次のとおりです。

H8S/2600 アドバンストモード・・・・・・・FBR=8、16、XBR=8、16、FRG=16、32、XRG=16、32、FWD=16、XTN=16、ALL=16
H8S/2600 ノーマルモード・・・・・・・・・・・FBR=8、16、XBR=8、16、FRG=16、XRG=16
H8S/2000 アドバンストモード・・・・・・・FBR=8、16、XBR=8、16、FRG=16、32、XRG=16、32、FWD=16、XTN=16、ALL=16
H8S/2000 ノーマルモード・・・・・・・・・・・FBR=8、16、XBR=8、16、FRG=16、XRG=16
H8/300H アドバンストモード ・・・・・・・FBR=8、16、XBR=8、16、FRG=16、24、XRG=16、24、FWD=16、XTN=16、ALL=16
H8/300H ノーマルモード ・・・・・・・・・・・FBR=8、16、XBR=8、16、FRG=16、XRG=16
H8/300 ・・・・・・・・・・・・・・・・・・・・・・・・FBR=8、XBR=8、FRG=16、XRG=16
H8/300L ・・・・・・・・・・・・・・・・・・・・・・・FBR=8、XBR=8、FRG=16、XRG=16

下線部は、指定を省略した場合の設定です。

本制御命令は何回でも指定でき、指定内容は本制御命令以降のソースステートメントに対して有効です。

FBR は、OPTIMIZE コマンドラインオプションと BR_RELATIVE コマンドラインオプションの指定がない場合に有効です。

本制御命令は、H8S/2600、H8S/2000、H8/300H のアドバンストモードと H8S/2600、H8S/2000、H8/300H のノーマルモードで有効です。

H8/300、H8/300L では、FBR=8、XBR=8、FRG=16、XRG=16 固定なので意味を持ちません。

(3) 例

```
        .CPU    2600A

        .SECTION A,CODE,ALIGN＝2
        .DISPSIZE FBR=16･･･････････････ [1]
        BRA     SYM ･･･････････････････ BRA SYM:16 と同じです。

        .DISPSIZE FBR=8･･･････････････ [2]
        BRA     SYM ･･･････････････････ BRA SYM:8 と同じです。
SYM
        MOV.W   R0,R1
```

[1] 前方参照の分岐命令のディスプレースメントサイズを 16 ビットに設定します。

[2] 前方参照の分岐命令のディスプレースメントサイズを 8 ビットに設定します。

5.8 アセンブルリストに関する制御命令

5.8.1 .PRINT（アセンブルリストの出力）

（1）書式

ラベル	オペレーション	オペランド
×	.PRINT	出力種別[,出力種別…]

出力種別：｛LIST｜NOLIST｜SRC｜NOSRC｜CREF｜NOCREF｜SCT｜NOSCT｝

（2）説明

アセンブルリストの出力、ソースプログラムリストの出力、クロスリファレンスリストの出力、セクション情報リストの出力を指定します。

（a）アセンブルリストの出力

アセンブルリストの出力、出力抑止を指定します。

出力種別は、次のとおりです。

LIST ・・・・・・・・・・・・出力

NOLIST ・・・・・・・・・・出力抑止

指定を省略した場合は、NOLIST を設定します。

本指定は、1 回限り有効です。

本指定は、[NO] LIST コマンドラインオプションの指定がない場合に有効です。

（b）ソースプログラムリストの出力

ソースプログラムリストの出力、出力抑止を指定します。

出力種別は、次のとおりです。

SRC ・・・・・・・・・・・・出力

NOSRC・・・・・・・・・・・出力抑止

指定を省略した場合は、SRC を設定します。

本指定は、1 回限り有効です。

本指定は、[NO] SOURCE コマンドラインオプションの指定がない場合に有効です。

本指定は、アセンブルリストの出力時に有効です。

(c) クロスリファレンスリストの出力

クロスリファレンスリストの出力、出力抑止を指定します。

出力種別は、次のとおりです。

CREF ・・・・・・・・・・・・出力
NOCREF ・・・・・・・・・出力抑止

指定を省略した場合は、CREF を設定します。

本指定は、1 回限り有効です。

本指定は、[NO] CROSS_REFERENCE コマンドラインオプションの指定がない場合に有効です。

本指定は、アセンブルリストの出力時に有効です。

(d) セクション情報リストの出力

セクション情報リストの出力、出力抑止を指定します。

出力種別は、次のとおりです。

SCT・・・・・・・・・・・・・出力
NOSCT・・・・・・・・・・・出力抑止

指定を省略した場合は、SCT を設定します。

本指定は、1 回限り有効です。

本指定は、[NO]SECTION コマンドラインオプションの指定がない場合に有効です。

本指定は、アセンブルリストの出力時に有効です。

(3) 例

```
        .PRINT  LIST,SRC,NOCREF,NOSCT
;
        .SECTION A,CODE,ALIGN=2
START
        MOV.W   R0,R1
        MOV.W   R0,R2
```

アセンブルリストにソースプログラムリストだけを出力します。

5.8.2 .LIST（ソースプログラムリストの部分出力）

（1）書式

ラベル	オペレーション	オペランド
×	.LIST	出力種別[,出力種別…]

出力種別：{ON｜OFF｜COND｜NOCOND｜DEF｜NODEF｜CALL｜NOCALL｜EXP｜NOEXP｜STR｜NOSTR｜CODE｜NOCODE}

（2）説明

ソースプログラムリストのソースステートメントの部分出力、プリプロセッサ機能のソースステートメントの部分出力、オブジェクトコード表示行の部分出力を指定します。

（a）ソースステートメントの部分出力

ソースステートメントの部分的な出力、出力抑止を指定します。

出力種別は、次のとおりです。

ON ‥‥‥‥ 出力

OFF‥‥‥‥ 出力抑止

出力種別の初期設定は、ONです。

（b）プリプロセッサ機能のソースステートメントの部分出力

プリプロセッサ機能のソースステートメントの部分的な出力、出力抑止を指定します。

出力種別は、次のとおりです。

COND‥‥‥‥‥ 条件付き不成立出力
NOCOND‥‥‥‥ 条件付き不成立出力抑止
DEF‥‥‥‥‥‥ 定義出力
NODEF‥‥‥‥‥ 定義出力抑止
CALL ‥‥‥‥‥ コール出力
NOCALL ‥‥‥‥ コール出力抑止
EXP‥‥‥‥‥‥ 展開出力
NOEXP‥‥‥‥‥ 展開出力抑止
STR‥‥‥‥‥‥ 構造化展開出力
NOSTR‥‥‥‥‥ 構造化展開出力抑止

本指定は、ソースステートメントの部分出力時に有効です。

出力種別の初期設定は、COND、DEF、CALL、EXP、STRです。

出力種別の内容は、次のとおりです。

条件付き不成立 ‥‥‥‥ .AIF の不成立部分
定義 ‥‥‥‥‥‥‥‥‥ マクロ定義部分、
.AREPEAT、.AWHILE 定義部分、
.INCLUDE 制御文、
.ASSIGNA、.ASSIGNC 制御文
コール ‥‥‥‥‥‥‥‥ マクロコール文、
構造化アセンブリ制御文、
.AIF、.AENDI 制御文
展開 ‥‥‥‥‥‥‥‥‥ マクロ展開部分、
.AREPEAT、.AWHILE 展開部分
構造化展開 ‥‥‥‥‥‥ 構造化アセンブリ展開部分

本指定は、[NO] SHOW コマンドラインオプションの指定がない場合に有効です。

(c) オブジェクトコード表示行の部分出力

制御命令のオブジェクトコード表示がソースステートメントの行数を超える部分の部分的な出力、出力抑止を指定します。

出力種別は、次のとおりです。

CODE‥‥‥ 出力
NOCODE‥‥ 出力抑止

出力種別の初期設定は、CODE です。

本指定は、[NO]SHOW コマンドラインオプションの指定がない場合に有効です。

本制御命令は、リスト上に出力しません。

本制御命令は何回でも指定でき、指定内容は本制御命令以降のソースステートメントに対して有効です。

本制御命令は、ソースプログラムリストの出力時に有効です。

(3) 例

```
        .PRINT  LIST
;
        .LIST   OFF····················· [1]
        .INCLUDE "bbb.h"
        .LIST   ON····················· [2]
;
        .SECTION A,CODE,ALIGN=2
START
        MOV.W   R0,R1
        MOV.W   R0,R2
```

ソースステートメントの出力を部分的に抑止しています。[1]～[2]間のソースステートメントは、ソースプログラムリスト上に出力しません。

```
        .PRINT  LIST
        .LIST   NOSTR ·················· [3]
;
        .SECTION A,CODE,ALIGN=2
START
        .IF.B (R0L<EQ>R1L) ············ [4]
                MOV.W  R0,R2
        .ELSE·························· [4]
                MOV.W  R0,R3
        .ENDI·························· [4]
```

[3]で構造化展開の出力抑止を指定しています。このため、[4]で展開したソースステートメントは、ソースプログラムリスト上に出力しません。

5.8.3 .FORM（リストサイズの設定）

（1）書式

ラベル	オペレーション	オペランド
×	.FORM	**リストサイズ**[,**リストサイズ**]

リストサイズ：{LIN＝行数｜COL＝桁数}

（2）説明

アセンブルリストのリストサイズを設定します。

リストサイズは、次のとおりです。

LIN=行数・・・・・・・・・1 ページの行数

COL=桁数・・・・・・・・1 行の桁数

リストサイズの初期設定は、LIN=60、COL=132 です。

（1）1 ページの行数

1 ページの行数を指定します。

行数は、後方参照の絶対値で指定します。

行数の値には、20～255 が指定できます。20 より小さい場合は 20 を、255 より大きい場合は 255 を設定します。この場合、エラーは表示しません。

本指定は、LINES コマンドラインオプションの指定がない場合に有効です。

（2）1 行の桁数

1 行の桁数を指定します。

桁数は、後方参照の絶対値で指定します。

桁数の値には、79～255 が指定できます。79 より小さい場合は 79 を、255 より大きい場合は 255 を設定します。この場合、エラーは表示しません。

本指定は、COLUMNS コマンドラインオプションの指定がない場合に有効です。

設定したリストサイズは、本制御命令以降のページに対して有効です。

(3) 例

```
.FORM     LIN=60,COL=80
```

アセンブルリストのリストサイズを、1 ページ 60 行、1 行 80 桁にします。

5.8.4 .HEADING（リストヘッディング）

(1) 書式

ラベル	オペレーション	オペランド
×	.HEADING	"**文字列**"

(2) 説明

ソースプログラムリストのページヘッダに表示するタイトルを設定します。

文字列は、文字をダブルコーテーション（"）で囲んで指定します。
ダブルコーテーション（"）自体を文字として指定する場合は、2 つ続けて記述します。

文字列は、60 文字まで指定できます。ただし、60 文字を超えた場合でもエラーは表示しません。

ページヘッダのタイトルは、本制御命令がリストの 1 行目にある場合はそのページから、本制御命令がリストの途中にある場合は次のページから表示します。

(3) 例

```
        .HEADING "MODULE NAME=""MAIN"""
        .PRINT  LIST
;
        .SECTION A,CODE,ALIGN=2
MAIN
        MOV.W   R0,R1
        MOV.W   R0,R2
```

ページヘッダのタイトルに、MODULE NAME="MAIN"と表示します。

5.8.5 .PAGE（改ページ）

(1) 書式

ラベル	オペレーション	オペランド
×	.PAGE	×

(2) 説明

ソースプログラムリストを改ページします。

本制御命令がリストの1行目にある場合は、改ページしません。

本制御命令は、リスト上に出力しません。
本制御命令は、ソースプログラムリストの出力時に有効です。

(3) 例

```
        .PRINT  LIST
;
        .INCLUDE "bbb.h"
;
        .PAGE
        .SECTION A,CODE,ALIGN＝2・・・・・・ [1]
START
        MOV.W   R0,R1
        MOV.W   R0,R2

;
        .PAGE
        .SECTION B,DATA,ALIGN＝2・・・・・・ [2]
DAT
        .DATA.W H'0001
        .DATA.W H'0002
```

改ページして[1]、[2]の行をリストの1行目にします。

5.8.6 .SPACE（空行出力）

（1）書式

ラベル	オペレーション	オペランド
×	.SPACE	[**行数**]

（2）説明

ソースプログラムリストに、指定した行数分の空行を出力します。

行数は、後方参照の絶対値で指定します。

行数には、1～50 の値が指定できます。1 より小さい場合は 1 を、50 より大きい場合は 50 を設定します。この場合、エラーは表示しません。

行数を省略した場合は、1 を設定します。

指定した行数が、1 ページを超えた部分の空行は出力しません。

出力した空行には、行番号などいっさいの表示をしません。

本制御命令は、リスト上に出力しません。

本制御命令は、ソースプログラムリストの出力時に有効です。

（3）例

```
        .PRINT  LIST
;
        .SECTION A,CODE,ALIGN=2
START
        MOV.W   R0,R1
        MOV.W   R0,R2·····[1]
        .SPACE  5
        MOV.W   R0,R3·····[2]
```

ソースプログラムリスト上の[1]と[2]の間には、5 行の空行が入ります。

5.9 その他の制御命令

5.9.1 .PROGRAM（オブジェクトモジュール名の設定）

(1) 書式

ラベル	オペレーション	オペランド
×	.PROGRAM	**オブジェクトモジュール名**

(2) 説明

オブジェクトモジュールにオブジェクトモジュール名を設定します。

オブジェクトモジュール名は、H シリーズ　リンケージエディタ、H シリーズ　ライブラリアンなどで使用します。

オブジェクトモジュール名は、シンボル名の書き方と同じです。

本制御命令を省略した場合は、オブジェクトファイル名の主ファイル名を設定します。

本指定は、1 回限り有効です。

(3) 例

```
        .PROGRAM main
;
        .SECTION A,CODE,ALIGN=2
START
        MOV.W   R0,R1
        MOV.W   R0,R2
```

オブジェクトモジュール名に main を設定します。

5.9.2 .RADIX（基数の設定）

（1）書式

ラベル	オペレーション	オペランド
×	.RADIX	基数種別

基数種別：{B｜Q｜D｜H}

（2）説明

整数定数のデフォルト基数を設定します。

基数種別は、次のとおりです。

B ・・・・・・・・・ 2 進数
Q ・・・・・・・・・ 8 進数
D ・・・・・・・・・ 10 進数
H ・・・・・・・・・ 16 進数

基数種別の初期設定は、D です。

本制御命令の対象となるのは、基数の指定がない整数定数です。

ただし、条件付きアセンブリ機能の制御文のオペランドには適用しません。条件付きアセンブリ機能の制御文で基数を省略した場合は、10 進数とみなします。

基数の指定がない 16 進数を記述する場合は、先頭が 0～9 でなければなりません。先頭が A～F で始まる記述はシンボル参照とみなし、本制御命令の対象になりません。この場合は先頭に 0 をつけてください。

本制御命令は何回でも指定でき、指定内容は本制御命令以降のソースステートメントに対して有効です。

（3）例

```
.SECTION A,DATA,ALIGN=2
.DATA.W -32768          .DATA.W -D'32768 と同じです。
.DATA.W 65535           .DATA.W D'65535 と同じです。
.RADIX   H
.DATA.W 8000            .DATA.W H'8000 と同じです。
.DATA.W 0FFFF           .DATA.W H'FFFF と同じです。
```

5.9.3 .END（ソースプログラムの終了）

(1) 書式

ラベル	オペレーション	オペランド
×	.END	[**実行開始アドレス**]

(2) 説明

ソースプログラムの終了を示します。

本制御命令が出現した時点で、アセンブラはアセンブルを終了します。

実行開始アドレスには、シミュレータ・デバッガでシミュレーションを開始するときのアドレスを指定します。

実行開始アドレスには、コードセクションのアドレスを設定します。
実行開始アドレスは、絶対値、またはアドレス値で指定します。
実行開始アドレスは、アドレス空間の指定により異なります。

(3) 例

```
        .CPU    2600A
        .OUTPUT DBG
;
        .SECTION A,CODE,ALIGN=2
START
        MOV.L   #0:32,ER0
        MOV.L   #1:32,ER1
        MOV.L   #2:32,ER2
        BRA     START
;
        .END    START
```

シミュレータ・デバッガでは、START のアドレスからシミュレーションを開始します。

6. ファイルインクルード機能

第 6 章　目次

6.1 ファイルインクルード機能の概要

ファイルインクルード機能は、アセンブルするソースファイルに別ソースファイルを取り込む機能です。

6.1.1 ファイルインクルード

ファイルインクルードは、.INCLUDE で指定したインクルードファイルを.INCLUDE の直後に取り込みます。

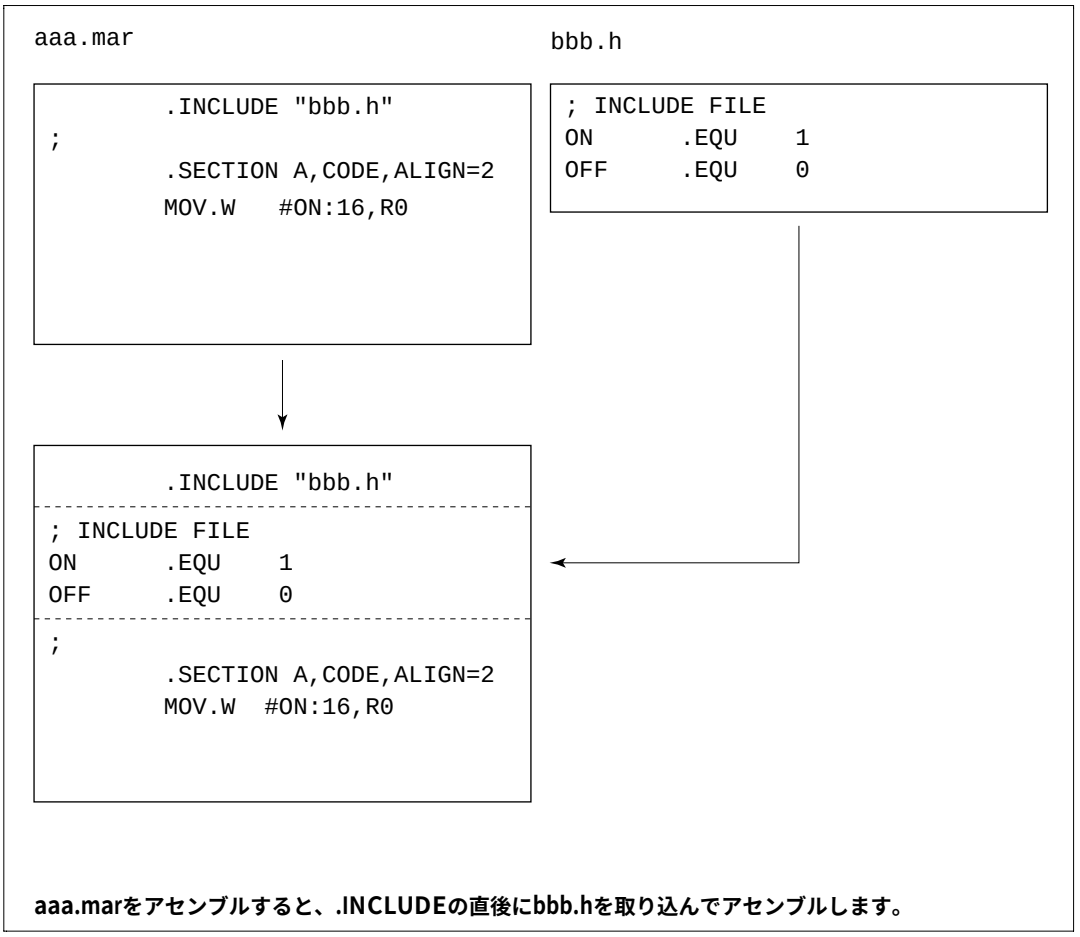


図 6.1 ファイルインクルード使用例

6.2 ファイルインクルード機能に関する制御文

6.2.1 .INCLUDE（ファイルインクルード）

（1）書式

ラベル	オペレーション	オペランド
×	.INCLUDE	"**ファイル名**"

（2）説明

ファイルをインクルードします。

ファイル名は、ダブルコーテーション（"）で囲んで指定します。

ファイル名は、ホストマシンの標準的な指定方法に従います。

ファイル名にファイル形式がない場合は、指定どおりのファイル名が有効になります。

インクルードファイルから、さらにファイルをインクルードすることもできます。ただし、許されるインクルードのネストは 30 までです。

本制御命令で指定したディレクトリ名は、INCLUDE コマンドラインオプションにより変更することができます。

（3）例

```
        .CPU    2600A
;
        .INCLUDE "import.h"
;
        .SECTION A,CODE,ALIGN=2
START
        MOV.L   #STACK:32,SP
        JSR     @PROG1:24
        JSR     @PROG2:24
;
        .END
```

ファイル import.h をインクルードします。

7. 条件付きアセンブリ機能

第 7 章　目次

7.1 条件付きアセンブリ機能の概要

条件付きアセンブリ機能は、ソースプログラムの一部分をアセンブルするかしないかを指定する機能です。

7.1.1 プリプロセッサ変数

プリプロセッサ変数は、主に条件付きアセンブリ機能で使用する変数で、整数型と文字型の 2 種類があります。

(1) 整数型プリプロセッサ変数

整数型プリプロセッサ変数は、.ASSIGNA 制御命令、ASSIGNA コマンドラインオプションで整数値を設定します。また、.ASSIGNA 制御命令では整数値の再設定ができます。

整数型プリプロセッサ変数を参照するときは、プリプロセッサ変数の先頭に円記号とアンパサンド（¥&）をつけます。

■例

```
PRE       .ASSIGNA 1・・・・・・・・・・・PRE に整数値 1 を設定しています。

          .AIF ¥&PRE EQ 1     .AIF 1 EQ 1 と同じです。
          MOV.W   R0,R1
          .AENDI
```

(2) 文字型プリプロセッサ変数

文字型プリプロセッサ変数は、.ASSIGNC 制御命令、ASSIGNC コマンドラインオプションで文字列を設定します。また、.ASSIGNC 制御命令では文字列の再設定ができます。

文字型プリプロセッサ変数を参照するときは、プリプロセッサ変数の先頭に円記号とアンパサンド（¥&）をつけます。

■例

```
PRE     .ASSIGNA "ON"・・・・・・・・・・・・・・・PRE に文字列 ON を設定しています。

        .AIF "¥&PRE" EQ "ON" ・・・・・・・.AIF "ON" EQ "ON"と同じです。
        MOV.W    R0,R1
        .AENDI
```

7.1.2 置換シンボル

.DEFINE 制御文で定義します。

ソースプログラムの一部分を指定によって、置き換えることができます。

コーディングは、次のようになります。

■コーディング例

```
SYM1:     .DEFINE "R1"
          ～
          MOV.W          SYM1,R0 ; MOV.W R1,R0 に置き換えられます。
          ～
```

7.1.3 条件付きアセンブル

ソースプログラムの一部分をアセンブルするか否か、条件によって切り替えることができます。

条件つきアセンブリの条件には、関係演算子で判別する比較型条件つきアセンブルと置換シンボルで判別する定義型条件つきアセンブルがあります。

(1) 比較型条件つきアセンブル

比較型条件つきアセンブルは、条件の成立か不成立かによりアセンブルする範囲を切り替えます。

コーディングは、次のようになります。

```
 ～
 .AIF   比較型条件
      条件が成立したときアセンブルする部分

 .AELIF   比較型条件
        条件が成立したときアセンブルする部分
 .AELSE
        全ての条件が成立しないときアセンブルする部分  この部分は省略可能
 .AENDI
```

■コーディング例

```
    ～
.AIF "¥&FLAG" EQ "ON"
 MOV R0,R3      ; FLAGが"ON"のときアセンブルします。
 MOV R1,R4      ;
 MOV R2,R5      ;
.AELSE
 MOV R3,R0      ; FLAGが"ON"でないときアセンブルします。
 MOV R4,R1      ;
 MOV R5,R2      ;
.AENDI
    ～
```

(2) 定義型条件つきアセンブル

定義型条件つきアセンブルは、置換シンボルが定義されているか否かによりアセンブルする範囲を切り替えます。

コーディングは、次のようになります。

```
～
.AIFDEF   比較型条件
    条件の置換シンボルが定義されていないときアセンブルする部分
.AELSE
      置換シンボルが定義されていないときアセンブルする部分   この部分は省略可能
.AENDI
```

■コーディング例

```
    ～
.AIFDEF FLAG
 MOV R0,R3      ; .AIFDEF制御文より後方で
 MOV R1,R4      ; FLAGが.DEFINE制御文で定義されているとき
 MOV R2,R5      ; アセンブルします。
.AELSE
 MOV R3,R0      ; .AIFDEF制御文より後方で
 MOV R4,R1      ; FLAGが.DEFINE制御文で定義されていないとき
 MOV R5,R2      ; アセンブルします。
.AENDI
    ～
```

7.1.4 繰り返し展開

繰り返し展開は、.AREPEAT～.AENDR 間のソースステートメントを、.AREPEAT で指定した回数分だけ繰り返し展開し、展開した部分をアセンブルします。

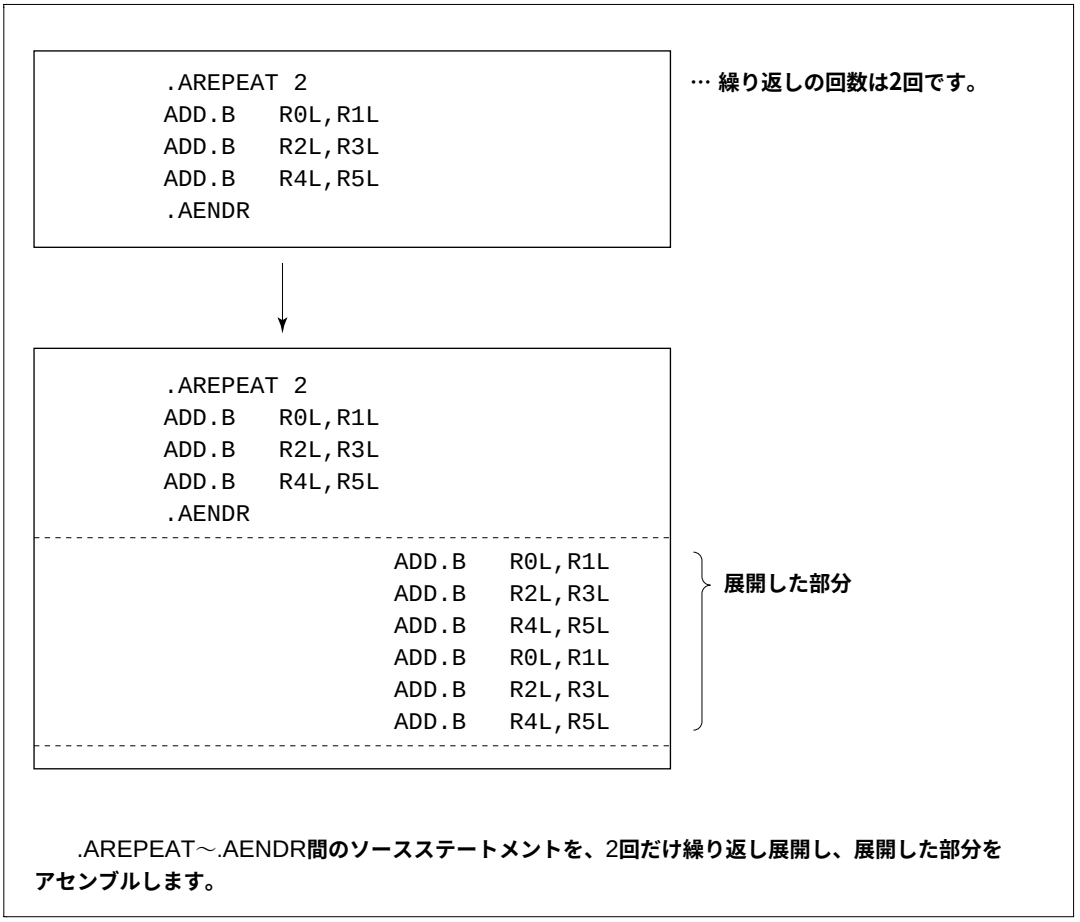


図 7.1 .AREPEAT **使用例**

7.1.5 条件付き繰り返し展開

条件付き繰り返し展開は、.AWHILE～.AENDW 間のソースステートメントを、.AWHILE で指定した条件が成立している間だけ繰り返し展開し、展開した部分をアセンブルします。

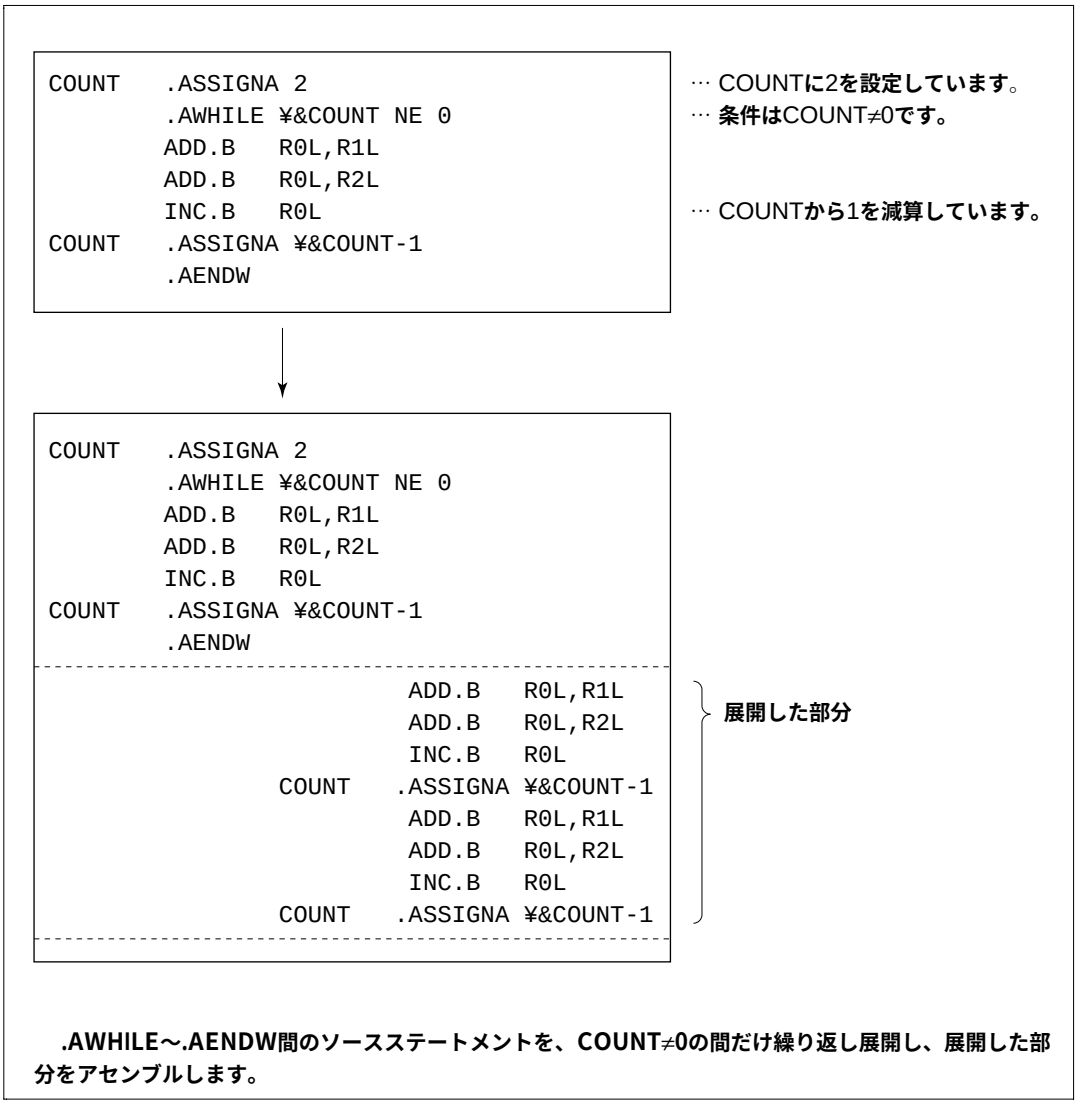


図 7.2 .AWHILE **使用例**

7.2 条件付きアセンブリ機能に関する制御文

7.2.1 .ASSIGNA（整数型プリプロセッサ変数）

（1）書式

ラベル	オペレーション	オペランド
プリプロセッサ変数名	.ASSIGNA	**値**

（2）説明

整数型プリプロセッサ変数に、値を設定します。

（1）整数型プリプロセッサ変数の定義

プリプロセッサ変数名の書き方は、シンボル名の書き方と同じです。ただし、文字数は最大 32 文字です。

値の指定は、次のとおりです。

・定数
・既に定義したプリプロセッサ変数
・上記を項とする式

本制御文で定義したプリプロセッサ変数は、再び本制御文で指定することにより値を変更できます。

定義したプリプロセッサ変数は、本制御文以降のソースステートメントに対して有効です。

本指定は、ASSIGNA コマンドラインオプションの指定がない場合に有効です。

（2）プリプロセッサ変数の参照

プリプロセッサ変数の参照方法は、次のとおりです。

¥&プリプロセッサ変数名［'］

アポストロフィ（'）は、プリプロセッサ変数名とソースステートメントの区別を明確にしたい場合に記述します。

参照した整数型プリプロセッサ変数は、設定した値（基数を省略した 10 進数）に置換されます。

プリプロセッサ変数を参照できる箇所は、次のとおりです。

・.ASSIGNA、.ASSIGNC

・.AIF、.AREPEAT、.AWHILE、.ALIMIT

・マクロ本体（.MACRO～.ENDM 間のソースステートメント）

(3) 例

```
FLAG    .ASSIGNA 1・・・・・・・・・・・・・・・・・・・・FLAG に 1 を設定します。
;
        .SECTION A,CODE,ALIGN=2
START
        .AIF ¥&FLAG EQ 1・・・・・・・・・・・・・・.AIF 1 EQ 1 と同じです。
        MOV.W   R0,R2
        .AENDI

        .AIF ¥&FLAG EQ 2・・・・・・・・・・・・・・.AIF 1 EQ 2 と同じです。
        MOV.W   R1,R2
        .AENDI
;
FLAG    .ASSIGNA 2・・・・・・・・・・・・・・・・・・・・FLAG の値を 2 に変更します。
;
        .AIF ¥&FLAG EQ 1・・・・・・・・・・・・・・.AIF 2 EQ 1 と同じです。
        MOV.W   R3,R5
        .AENDI

        .AIF ¥&FLAG EQ 2・・・・・・・・・・・・・・.AIF 2 EQ 2 と同じです。
        MOV.W   R4,R5
        .AENDI
```

整数型プリプロセッサ変数 FLAG を.AIF で参照しています。

7.2.2 .ASSIGNC（文字型プリプロセッサ変数）

（1）書式

ラベル	オペレーション	オペランド
プリプロセッサ変数名	.ASSIGNC	"**文字列**"

（2）説明

文字型プリプロセッサ変数に、文字列を設定します。

（a）文字型プリプロセッサ変数の定義

プリプロセッサ変数名の書き方は、シンボル名の書き方と同じです。ただし、文字数は最大 32 文字です。

文字列は、文字をダブルコーテーション（"）で囲んで指定します。
ダブルコーテーション（"）自体を文字として指定する場合は、2 つ続けて記述します。

本制御文で定義したプリプロセッサ変数は、再び本制御文で指定することにより文字列を変更できます。

定義したプリプロセッサ変数は、本制御文以降のソーステートメントに対して有効です。

本指定は、ASSIGNC コマンドラインオプションの指定がない場合に有効です。

（b）プリプロセッサ変数の参照

プリプロセッサ変数の参照方法は、次のとおりです。

¥&プリプロセッサ変数名[']

アポストロフィ（'）は、プリプロセッサ変数名とソーステートメントの区別を明確にしたい場合に記述します。

参照した文字型プリプロセッサ変数は、設定した文字列（文字列を囲んだダブルコーテーション（"）を除く）に置換されます。

プリプロセッサ変数を参照できる箇所は、次のとおりです。

- .ASSIGNA、.ASSIGNC
- .AIF、.AREPEAT、.AWHILE

・マクロ本体（.MACRO～.ENDM 間のソースステートメント）

(3) 例

```
FLAG    .ASSIGNC "ON"・・・・・・・・・・・・・・・・・FLAG に文字列 ON を設定します。
;
        .SECTION A,CODE,ALIGN=2
START
        .AIF "¥&FLAG" EQ "ON" ・・・・・・・・.AIF "ON" EQ "ON"と同じです。
        MOV.W   R0,R2
        .AENDI
;
        .AIF "¥&FLAG" EQ "OFF" ・・・・・・・.AIF "ON" EQ "OFF"と同じです。
        MOV.W   R1,R2
        .AENDI
;
FLAG    .ASSIGNC "OFF"・・・・・・・・・・・・・・・・FLAG を文字列 OFF に変更します。
;
        .AIF "¥&FLAG" EQ "ON" ・・・・・・・・.AIF "OFF" EQ "ON"と同じです。
        MOV.W   R3,R5
        .AENDI

        .AIF "¥&FLAG" EQ "OFF" ・・・・・・・.AIF "OFF" EQ "OFF"と同じです。
        MOV.W   R4,R5
        .AENDI
```

文字型プリプロセッサ変数 FLAG を.AIF で参照しています。

7.2.3 .DEFINE 制御命令（文字列の置き換えを定義）

(1) 書式

ラベル	オペレーション	オペランド
シンボル[:]	.DEFINE	"**文字列**"

(2) 説明

.DEFINE は、シンボルを対応する文字列に置き換えることを指定します。.DEFINE で定義したシンボルを置換シンボルと呼びます。

.DEFINE と.ASSIGNC との違いは以下の点です。

(a) .ASSIGNC で定義したシンボルは、プリプロセッサ文でしか使用できませんが、.DEFINE で定義したシンボルは、任意のステートメントで使用できます。

(b) .ASSIGNA、.ASSIGNC で定義したシンボルは、「¥&シンボル」の形式で参照しますが、.DEFINE で定義したシンボルは、「シンボル」の形式で参照します。

(c) .DEFINE は、再定義できません。

(d) .DEFINE で定義できるシンボルの文字数は最大 32 文字です。

(3) 例

■例 1

```
SYM1:   .DEFINE    "R1"
        MOV.W      SYM1,R0          ;MOV.W R1,R0 に置き換えられます。
```

【注】 1. **コマンドラインオプションで置換シンボルが定義されている場合、同名のシンボルに対する** .DEFINE **は無効になります。**

2. **先頭が** a～f **または、**A～F **で始まる** 16 **進数は、**.DEFINE **で同名のシンボルが定義された場合、置換対象になります。置換対象外にするには、先頭に** 0 **を指定してください。**

■例 2

```
A0:     .DEFINE                 "0"
        MOV.B  #H'A0,R0L        ; MOV.B  #H'0,R0L に置き換えられます。
        MOV.B  #H'0A0,R0L       ; 置き換えられません。
```

【注】 3. **基数（**B'、Q'、D'、H'**）は、**.DEFINE **で同名のシンボルが定義された場合、置**

換対象になります。B、D、Q、H、b、d、q、h **一文字のシンボルを定義するときは注意してください。**

■例 3

```
B:        .DEFINE       "H"
          MOV.W  #B'10,R0          ;MOV.W  #H'10,R0 に置き換えられます。
```

7.2.4 .AIF（条件付きアセンブル）

（1）書式

ラベル	オペレーション	オペランド
×	.AIF	**項** 1 **関係演算子 項** 2
[×	.AELIF	**項** 1 **関係演算子 項** 2]
[×	.AELSE	×]
×	.AENDI	×

関係演算子：｛EQ｜NE｜GT｜LT｜GE｜LE｝

（2）説明

条件付きアセンブルを行います。

.AIF～.AELIF～.AELSE～.AENDI 間に記述したソースステートメントのうち、条件が成立した部分をアセンブルします。
.AELIF、.AELSE は、省略することができます。

項は、値、または文字列で指定します。ただし、値と文字列の比較はできません。値と文字列の比較をすると、条件不成立になります。

値は、後方参照の絶対値、またはプリプロセッサ変数で指定します。

文字列は、文字、またはプリプロセッサ変数をダブルコーテーション（"）で囲んで指定します。
ダブルコーテーション（"）自体を文字として指定する場合は、2 つ続けて記述します。

関係演算子の条件は、次のとおりです。

EQ‥‥‥‥‥項1=項2

NE‥‥‥‥‥項1≠項2

GT‥‥‥‥‥項1>項2

LT‥‥‥‥‥項1<項2

GE‥‥‥‥‥項1≧項2

LE‥‥‥‥‥項1≦項2

(3) 例

```
        ～
.AIF    ¥&TYPE EQ 1
MOV.W   R0,R3           ; TYPEが1のときアセンブルします。
MOV.W   R1,R4           ;
MOV.W   R2,R5           ;
.AELIF  ¥&TYPE EQ 2
MOV.W   R2,R0           ; TYPEが2のときアセンブルします。
MOV.W   R3,R1           ;
MOV.W   R4,R5           ;
.AELSE
MOV.W   R3,R0           ; TYPEが1でも2でもないのときアセンブルします。
MOV.W   R4,R1           ;
MOV.W   R5,R2           ;
.AENDI
```

7.2.5 .AIFDEF（シンボル定義の有無による条件付きアセンブル）

(1) 書式

```
.AIFDEF 置換シンボル
 置換シンボル定義時にアセンブルするソースステートメント
.AELSE
 置換シンボル未定義時にアセンブルするソースステートメント
.AENDI
```

(2) 説明

.AIFDEF～.AELSE～.AENDI は、アセンブルするか否かを置換シンボルの定義によって切り替えます。.AELSE は、省略できます。

置換シンボルは、-DEFINE または.DEFINE で定義します。

記述した置換シンボルがコマンドラインオプションで定義されている、または.AIFDEF より後方のソースステートメントで定義されている場合、条件成立となります。

記述した置換シンボルが.AIFDEF より前方のソースステートメントで定義されている、または定義がない場合、条件不成立となります。

(3) 例

```
.AIFDEF     FLAG
      MOV.W R1,R2       ;FLAGが.DEFINEで定義されているとき
      MOV.W #0,R1       ;アセンブルします。
.AELSE
      MOV.W R2,R1       ;FLAGが.DEFINEで定義されていないとき
      MOV.W #0,R2       ;アセンブルします。
.AENDI
```

7.2.6 .AREPEAT（繰り返し展開）

（1）書式

ラベル	オペレーション	オペランド
×	.AREPEAT	**回数**
×	.AENDR	×

（2）説明

繰り返し展開を行います。

.AREPEAT～.AENDR 間に記述したソースステートメントを、指定した回数分だけ繰り返し展開します。

回数は、後方参照の絶対値、またはプリプロセッサ変数で指定します。
回数には、1 以上の値が指定できます。0 以下の値を指定した場合は、展開しません。

（3）例

```
.AREPEAT 2
SHAL.B      R0L
SHAL.B      R1L
SHAL.B      R2L
.AENDR
```

2 回だけ繰り返し展開します。

```
.AREPEAT 3
.DATA.W H'0001,H'0002
.DATA.W H'0003,H'0004
.AENDR
```

3 回だけ繰り返し展開します。

7.2.7 .AWHILE（条件付き繰り返し展開）

（1）書式

ラベル	オペレーション	オペランド
×	.AWHILE	項1　関係演算子　項2
×	.AENDW	×

関係演算子：｛EQ｜NE｜GT｜LT｜GE｜LE｝

（2）説明

条件による繰り返し展開を行います。

.AWHILE～.AENDW 間に記述したソースステートメントを、条件が成立している間だけ繰り返し展開します。

項は、値、または文字列で指定します。ただし、値と文字列の比較はできません。値と文字列の比較をすると、条件不成立になります。

値は、後方参照の絶対値、またはプリプロセッサ変数で指定します。

文字列は、文字、またはプリプロセッサ変数をダブルコーテーション（"）で囲んで指定します。
ダブルコーテーション（"）自体を文字として指定する場合は、2つ続けて記述します。

関係演算子の条件は、次のとおりです。

EQ‥‥‥‥項1＝項2
NE‥‥‥‥項1≠項2
GT‥‥‥‥項1＞項2
LT‥‥‥‥項1＜項2
GE‥‥‥‥項1≧項2
LE‥‥‥‥項1≦項2

条件付き繰り返し展開は、最終的に条件を不成立にして展開を終了します。
条件が不成立にならない場合は、展開回数が65,535回になるまで展開処理を繰り返します。

繰り返し展開の上限値は、.ALIMIT制御文で指定することができます。

(3) 例

```
COUNT   .ASSIGNA 2·····COUNTに2を設定しています。
        .AWHILE ¥&COUNT NE 0·····条件はCOUNT≠0です。
        ADD.B   R0L,R1L
        ADD.B   R0L,R2L
        INC.B   R0L
COUNT   .ASSIGNA ¥&COUNT-1·····COUNTから1を減算しています。
        .AENDW
```

COUNT≠0の間だけ繰り返し展開します。

```
STOP    .ASSIGNA 0·····STOPに0を設定しています。
        .AWHILE ¥&STOP LE 10·····条件はSTOP≦10です。
        ADD.B   R0L,R1L
        ADD.B   R0L,R2L
        INC.B   R0L
STOP    .ASSIGNA ¥&STOP+3·····STOPに3を加算しています。
        .AENDW
```

STOP≦10の間だけ繰り返し展開します。

7.2.8 .ALIMIT（条件付き繰り返し展開回数の上限値の設定）

(1) 書式

ラベル	オペレーション	オペランド
×	.ALIMIT	**上限値**

(2) 説明

条件付き繰り返し展開回数の上限値を設定します。

上限値の指定は、次のとおりです。

・定数
・既に定義したプリプロセッサ変数
・上記を項とする式

本制御命令で指定した条件付き繰り返し展開回数の上限値は、再び本制御命令で指定することにより、値を変更できます。

本制御命令で指定した条件付き繰り返し展開回数の上限値は、本制御命令以降のソースステートメントに対して有効です。

本制御命令を指定しない場合、展開回数の上限値を、65,535 とします。

(3) 例

```
COUNT   .ASSIGNA 3
        .ALIMIT 10 ‥‥‥‥‥‥‥‥‥‥繰り返し展開回数の上限値に、10 を設定しています。
        .AWHILE ¥&COUNT NE 4
        ADD. B R0L, R1L ‥‥‥‥‥‥‥‥‥‥‥‥‥‥[1]
        ADD. B R0L, R1L ‥‥‥‥‥‥‥‥‥‥‥‥‥‥[1]
        INC. B R0L ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥[1]
COUNT   .ASSIGNA ¥&COUNT-1‥‥‥‥‥‥‥‥‥‥‥‥[1]
        .AENDW
```

COUNT≠4 の間だけ[1]を展開します。

10 回展開した後、ウォーニング 854 を出力し、繰り返し展開を中断終了します。

7.2.9 .EXITM（展開の中断終了）

（1）書式

ラベル	オペレーション	オペランド
×	.EXITM	×

（2）説明

.AREPEAT、.AWHILE の繰り返し展開を中断終了します。

本制御文は、.AREPEAT～.AENDR 間、.AWHILE～.AENDW 間で指定します。各展開では、本制御文が出現した時点で、展開を中断終了します。

本制御文は、マクロ展開の中断終了にも使用します。マクロ命令と繰り返し展開を組み合わせて使用する場合は、本制御文の位置に注意してください。

（3）例

```
COUNT   .ASSIGNA 0
        .AWHILE 1 EQ 1・・・・・・・・・・無限展開（常に条件成立）を指定しています。
        ADD.B   R0L,R1L
        ADD.B   R2L,R3L
COUNT   .ASSIGNA ¥&COUNT+1
        .AIF ¥&COUNT EQ 2 ・・・・・・・条件は COUNT=2 です。
        .EXITM
        .AENDI
        .AENDW
```

COUNT が更新され.AIF の条件が成立すると、.EXITM がアセンブルされます。.EXITM がアセンブルされた時点で、.AWHILE の展開を中断終了します。

8. マクロ機能

第 8 章　目次

8.1 マクロ機能の概要

マクロ機能は、よく使われる命令の組み合わせに名称をつけて定義し、その名称を記述することにより命令の組み合わせを呼び出す機能です。

8.1.1 マクロ定義とマクロコール

マクロ定義は、.MACRO～.ENDM 間のソースステートメント（マクロ本体）をマクロ命令として定義し、マクロ名（マクロ命令の名称）をつけます。

マクロコールは、マクロ名をオペレーションフィールドに記述し、マクロ命令を呼び出します。

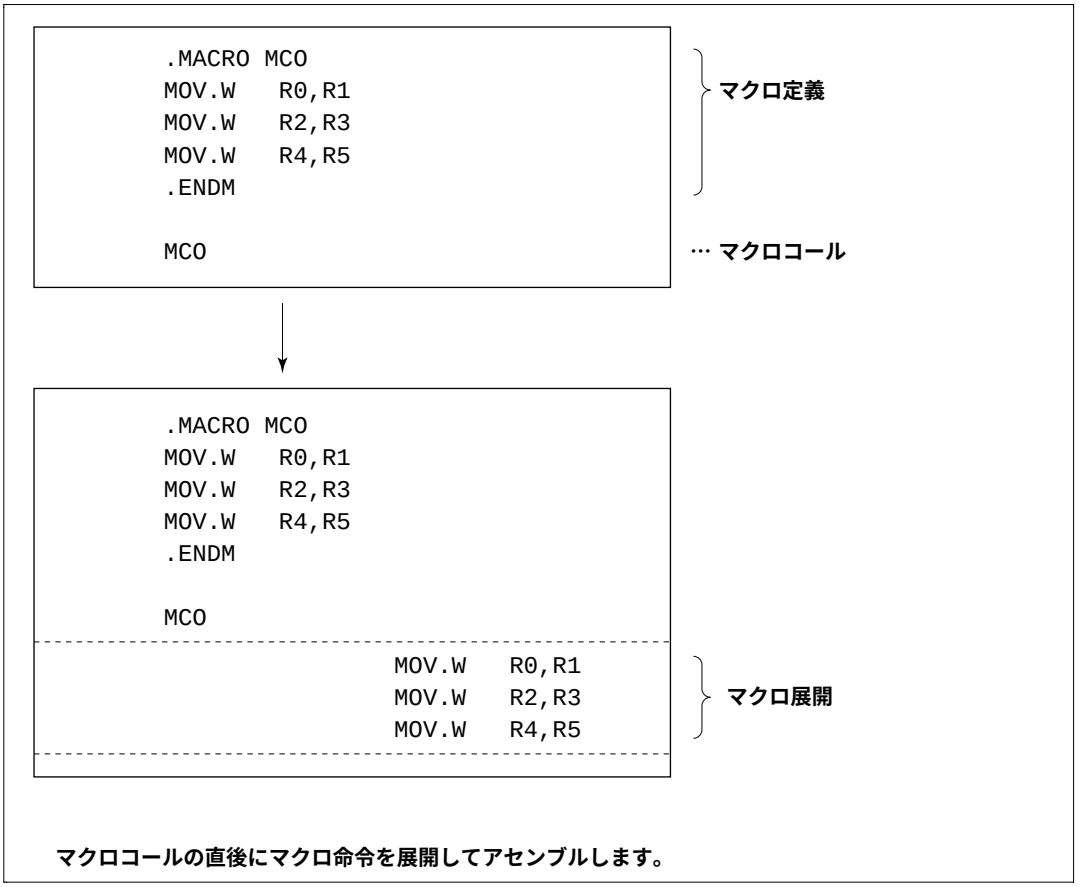


図 8.1 .MACRO **使用例**

また、マクロ命令はパラメータ（マクロパラメータ）を使用することにより、マクロ本体の一部を任意の文字に置換して展開できます。

手順は、次のとおりです。

（1）.MACRO のオペランドに、仮引数を宣言します。

（2）マクロ本体に、.MACRO の仮引数と同じ仮引数を記述します。このとき、仮引数の先頭に円記号（¥）をつけて、仮引数であることを示します。

（3）マクロコールで、仮引数に対応させてマクロパラメータを指定します。

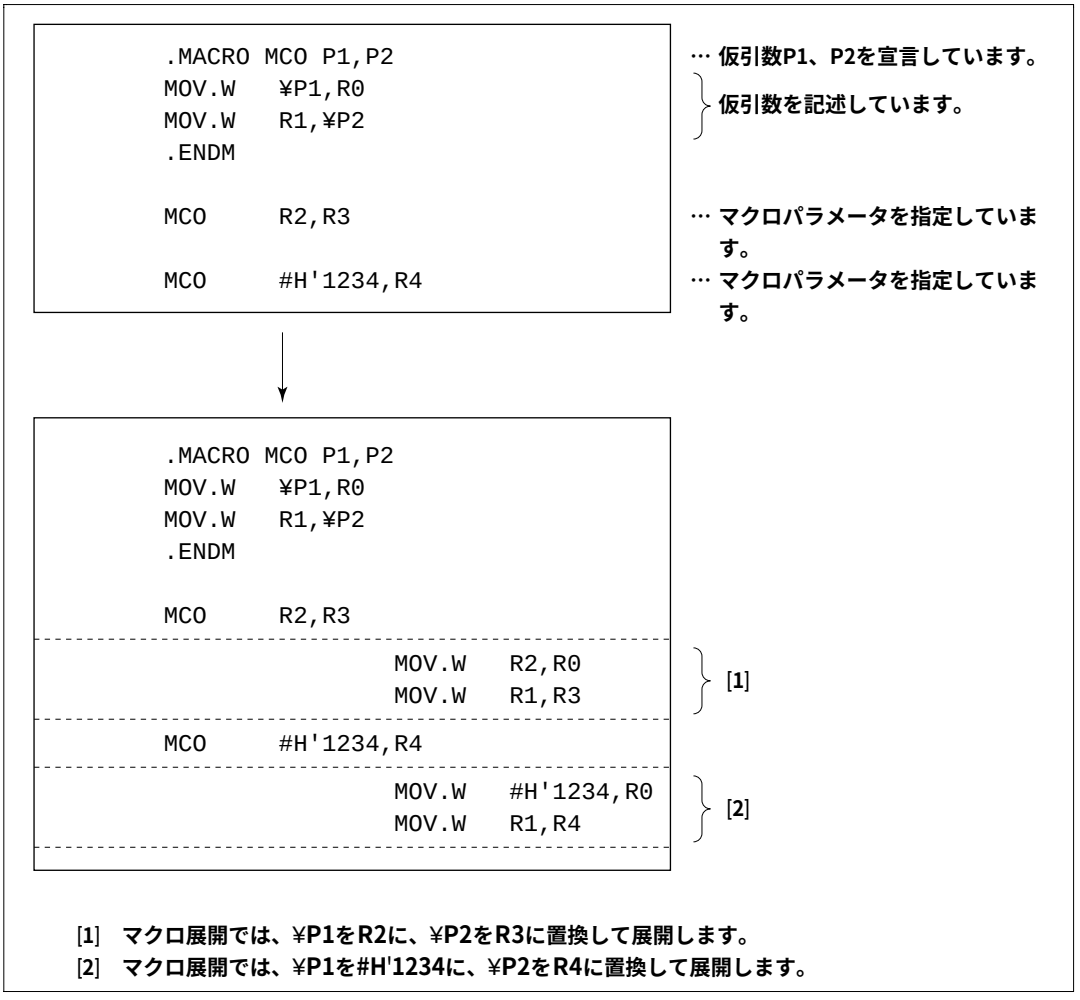


図 8.2 .MACRO（仮引数）使用例

8.2 マクロ機能に関する制御文

8.2.1 .MACRO（マクロ定義）

(1) 書式

ラベル	オペレーション	オペランド
×	.MACRO	マクロ名［△仮引数［,仮引数…］］
×	.ENDM	×

仮引数：仮引数名［=デフォルト］

(2) 説明

マクロ命令を定義します。

.MACRO～.ENDM 間のソースステートメント（マクロ本体）を、マクロ名に対するマクロ命令として定義します。

定義したマクロ命令は、マクロコールで呼び出します。

(a) マクロ名

マクロ名は、マクロ命令につけた名前です。マクロコールで使用します。

マクロ名の書き方は、シンボル名の書き方と同じです。ただし、実行命令、制御命令（ピリオド（.）を除く）、制御文（ピリオド（.）を除く）のニーモニックは指定できません。

また、マクロ名は大文字と小文字を区別しません。

(b) 仮引数

仮引数は、マクロ展開でマクロ本体の一部を置換して展開したい場合に指定します。

置換する文字列は、マクロコールのマクロパラメータで指定します。

■仮引数名の書き方

仮引数名の書き方は、シンボル名の書き方と同じです。ただし、文字数は最大 32 文字です。

また、仮引数名は大文字と小文字を区別します。

■仮引数の参照

マクロ本体では、置換したい部分に仮引数名を記述します。

マクロ本体での仮引数の参照方法は、次のとおりです。

¥仮引数名［'］

アポストロフィ（'）は、仮引数名とソーステートメントの区別を明確にしたい場合に記述します。

(c) 仮引数のデフォルト

仮引数には、仮引数のデフォルトを設定できます。仮引数のデフォルトには、マクロコールのマクロパラメータを省略した場合に置換する文字列を指定します。

仮引数のデフォルトに次の文字を含む場合は、文字列をダブルコーテーション（"）、またはアングルブラケット（<>）で囲んでください。

・スペース
・タブ
・コンマ（,）
・セミコロン（;）
・ダブルコーテーション（"）
・アングルブラケット（<>）

マクロ展開では、文字列を囲んだダブルコーテーション（"）やアングルブラケット（<>）は取り除いて置換します。

マクロ命令の定義には、次の制限があります。

（1）マクロ命令は、次の場所では定義できません。

・マクロ本体（.MACRO～.ENDM 間）
・.AREPEAT～.AENDR 間
・.AWHILE～.AENDW 間

（2）マクロ本体には、.END は記述できません。

（3）.ENDM のラベルフィールドには、シンボルは記述できません。.ENDM のラベルフィールドにシンボルを記述した場合は、.ENDM を無視します。この場合、エラーは表示しません。

(4) 例

```
.MACRO MCO P1,P2  ····· [1]
MOV.W   ¥P1,R1 ····· [1]
MOV.W   ¥P2,R3 ····· [1]
MOV.W   R4,R5 ····· [1]
.ENDM

MCO R0,R2 ····· [2]
```

[1] マクロ命令 MCO を定義しています。仮引数 P1、P2 を宣言しています。

[2] マクロコールです。マクロパラメータに仮引数 P1 に対して R0 を、P2 に対して R2 を指定しています。
マクロ展開では、¥P1 を R0 に、¥P2 を R2 に置換して展開します。

```
.MACRO MCO P1=R0,P2=R2 ····· [3]
MOV.W   ¥P1,R1
MOV.W   ¥P2,R3
MOV.W   R4,R5
.ENDM

MCO     ,R6 ····· [4]
```

[3] 仮引数のデフォルトとして、P1 に対して R0 を、P2 に対して R2 を設定しています。

[4] マクロコールです。仮引数 P1 に対するマクロパラメータは省略しています。P2 に対しては R6 を指定しています。
マクロ展開では、¥P1 を R0 に、¥P2 を R6 に置換して展開します。

8.2.2 マクロ本体

(1) 説明

マクロ本体（.MACRO～.ENDM 間のソースステートメント）が持つ機能を説明します。

(a) 仮引数の参照

マクロ展開で、マクロパラメータと置換したい部分に仮引数名を記述します。

仮引数の参照方法は、次のとおりです。

¥仮引数名［'］

アポストロフィ（'）は、仮引数名とソースステートメントの区別を明確にしたい場合に記述します。

■例

```
        .MACRO MCO P1
        MOV.W   ¥P1,R1
        .ENDM

        MCO     R0
```

(b) プリプロセッサ変数（.ASSIGNA、.ASSIGNC）の参照

マクロ本体では、プリプロセッサ変数を参照できます。

プリプロセッサ変数の参照方法は、次のとおりです。

¥&プリプロセッサ変数名［'］

アポストロフィ（'）は、プリプロセッサ変数名とソースステートメントの区別を明確にしたい場合に記述します。

■例

```
        .MACRO MCO
        MOV.W   ¥&PRE,R1
        .ENDM

PRE     .ASSIGNC "R0"
        MCO
```

(c) マクロ生成番号

マクロ生成番号は、マクロ展開で固有の 5 桁の 10 進数（00000～99999）を展開します。

マクロ生成番号の書き方は、次のとおりです。

¥@

マクロ本体にラベルがある場合などは、複数回マクロコールするとシンボル名が重複してしまいます。

こういう場合に、シンボル名の一部としてマクロ生成番号を記述しておくと、マクロコールのたびに固有のシンボル名となり、重複を避けることができます。

■例

```
        .MACRO MCO
        BEQ     SYM¥@:8
SYM¥@
        .ENDM

        MCO
        MCO
```

(d) マクロ処理除外

マクロ本体に円記号（¥）があると、マクロの置換処理の対象になります。したがって、円記号（¥）をただの文字として記述したい場合は、マクロの置換処理から除外する必要があります。

マクロ処理除外の書き方は、次のとおりです。

¥（マクロ処理除外文字列）

マクロ展開では、円記号（¥）とカッコは取り除きます。

また、マクロ処理除外では、文字列操作関数も置換処理から除外します。

■例

```
        .MACRO MCO
        ¥(MOV.B   #"¥",R0L)
        .ENDM

        MCO
```

(e) マクロ内コメント

マクロ本体のコメントを、マクロ展開では展開したくない場合にマクロ内コメントを記述します。

マクロ内コメントの書き方は、次のとおりです。

¥;コメント

■例

```
.MACRO MCO P1
MOV.W   ¥P1,R1  ¥; ¥P1:Rn
.ENDM

MCO     R0
```

(f) 文字列操作関数

マクロ本体には、文字列操作関数を記述できます。

文字列操作関数には、次のものがあります。

.LEN 関数 ・・・・・・・・・・文字列の長さ（**「8.2.5 .LEN 関数」**参照）

.INSTR 関数 ・・・・・・・・・文字列の検索（**「8.2.6 .INSTR 関数」**参照）

.SUBSTR 関数・・・・・・・・文字列の切り出し（**「8.2.7 .SUBSTR 関数」**参照）

8.2.3 マクロコール

(1) 書式

ラベル	オペレーション	オペランド
[シンボル名]	マクロ名	[マクロパラメータ] [,[マクロパラメータ]…]

マクロパラメータ：[仮引数名=]文字列

(2) 説明

マクロ命令を呼び出します。

呼び出すマクロ命令は、マクロコール以前にマクロ定義（.MACRO）します。

マクロパラメータには、マクロ展開で置換する文字列を指定します。この場合、マクロ名に対応するマクロ定義（.MACRO）で、仮引数を宣言しておく必要があります。

(a) マクロパラメータの指定方法

マクロパラメータの指定方法には、位置指定とキーワード指定があります。

■位置指定

マクロ定義（.MACRO）で宣言した仮引数の並び順と、マクロパラメータの並び順を一致させて指定する方法です。

■キーワード指定

マクロ定義（.MACRO）で宣言した仮引数の仮引数名に、イコール（=）で区切って指定する方法です。

(b) マクロパラメータの書き方

マクロパラメータに次の文字を含む場合は、文字列をダブルコーテーション（"）、またはアングルブラケット（<>）で囲んでください。

- スペース
- タブ
- コンマ（,）
- セミコロン（;）
- ダブルコーテーション（"）
- アングルブラケット（<>）

マクロ展開では、文字列を囲んだダブルコーテーション（"）やアングルブラケット（<>）は取り除いて置換します。

（3）例

```
.MACRO MCO P1,P2･･･････････････ [1]
MOV.W   ¥P1,R1･････････････････ [1]
MOV.W   ¥P2,R3･････････････････ [1]
MOV.W   R4,R5 ･････････････････ [1]
.ENDM   [1]

MCO R0,R2･･･････････････････････ [2]

MCO P1=R0,P2=R2･･･････････････ [3]
```

[1] マクロ命令 MCO を定義しています。仮引数 P1、P2 を宣言しています。
[2] マクロパラメータを位置指定しています。
[3] マクロパラメータをキーワード指定しています。

[2]と[3]のマクロ展開では、仮引数¥P1 は R0 に、¥P2 は R2 に置換して展開します。

8.2.4 .EXITM（展開の中断終了）

(1) 書式

ラベル	オペレーション	オペランド
×	.EXITM	×

(2) 説明

マクロ展開を中断終了します。

本制御文は、マクロ本体（.MACRO～.ENDM 間）で指定します。マクロ展開では、本制御文が出現した時点で、展開を中断終了します。

本制御文は、.AREPEAT、.AWHILE の繰り返し展開の中断終了にも使用します。マクロ命令と繰り返し展開を組み合わせて使用する場合は、本制御文の位置に注意してください。

(3) 例

```
.MACRO MCO P1
MOV.W   R0,R1·····[1]
MOV.W   R2,R3·····[1]
MOV.W   R4,R5·····[1]
¥P1
MOV.W   R6,R7
.ENDM

MCO     .EXITM
```

.EXITM を展開し、マクロ展開を中断終了します。[1]の部分が展開されます。

8.2.5 .LEN 関数（文字列操作関数）

(1) 書式

.LEN[△]（"文字列"）

(2) 説明

文字列の長さを数え、文字数を基数を省略した 10 進数に置換します。

文字列は、文字をダブルコーテーション（"）で囲んで指定します。
ダブルコーテーション（"）自体を文字として指定する場合は、2 つ続けて記述します。

文字列には、マクロの仮引数、プリプロセッサ変数を指定できます。

.LEN（"¥仮引数名"）
.LEN（"¥&プリプロセッサ変数名"）

本関数を記述できるのは、マクロ本体（.MACRO～.ENDM 間）だけです。
本関数に文法上の誤りがある場合は、置換しません。この場合、エラーは表示しません。

(3) 例

```
.MACRO MCO P1
.SRES    .LEN（"¥P1"）
.ENDM

MCO      ABCDEF·····[1]
MCO      GHI ·····[2]
```

[1]. SRES 6 を展開します。
[2]. SRES 3 を展開します。

8.2.6 .INSTR 関数（文字列操作関数）

（1）書式

.INSTR[△]（**"文字列 1","文字列 2"[,検索開始位置]**）

（2）説明

文字列 1 に文字列 2 が含まれているかを検索し、文字列 1 の先頭を 0 とした検出位置を基数を省略した 10 進数に置換します。

文字列 1 に文字列 2 が含まれていない場合は、－1 に置換します。

文字列は、文字をダブルコーテーション（"）で囲んで指定します。

ダブルコーテーション（"）自体を文字として指定する場合は、2 つ続けて記述します。

検索開始位置は、文字列 1 の先頭を 0 とした検索を開始する位置を指定します。省略した場合は、0 を設定します。

検索開始位置は、後方参照の絶対値で指定します。値は、0 以上が指定できます。

文字列、検索開始位置には、マクロの仮引数、プリプロセッサ変数を指定できます。

.INSTR（"¥仮引数名","¥仮引数名",¥仮引数名）

.INSTR("¥&プリプロセッサ変数名","¥&プリプロセッサ変数名",¥&プリプロセッサ変数名）

本関数を記述できるのは、マクロ本体（.MACRO～.ENDM 間）だけです。

本関数に文法上の誤りがある場合は、置換しません。この場合、エラーは表示しません。

（3）例

```
.MACRO MCO P1
.DATA.W .INSTR（"ABCDEFG","¥P1",0）
.ENDM

MCO     CDE ･･･････････････････ [1]
MCO     H･･･････････････････････ [2]
```

[1] .DATA.W 2 を展開します。

[2] .DATA.W -1 を展開します。

8.2.7 .SUBSTR 関数（文字列操作関数）

（1）書式

.SUBSTR[△]（**"文字列",切り出しの開始位置,切り出しの長さ**）

（2）説明

文字列から、文字列の先頭を 0 とした切り出しの開始位置から切り出しの長さ分の文字列を切り出し、ダブルコーテーション（"）で囲んだ文字列に置換します。

文字列は、文字をダブルコーテーション（"）で囲んで指定します。
ダブルコーテーション（"）自体を文字として指定する場合は、2 つ続けて記述します。

切り出しの開始位置、切り出しの長さは、後方参照の絶対値で指定します。
切り出しの開始位置は、0 以上が指定できます。
切り出しの長さは、1 以上が指定できます。

切り出しの開始位置、切り出しの長さが不適当な場合は、空文字（""）に置換します。

切り出しの開始位置、切り出しの長さには、マクロの仮引数、プリプロセッサ変数を指定できます。

.SUBSTR（"¥仮引数名",¥仮引数名,¥仮引数名）
.SUBSTR（"¥&プリプロセッサ変数名",¥&プリプロセッサ変数名,¥&プリプロセッサ変数名）

本関数を記述できるのは、マクロ本体（.MACRO～.ENDM 間）だけです。
本関数に文法上の誤りがある場合は、置換しません。この場合、エラーは表示しません。

（3）例

```
.MACRO MCO P1
.SDATA   .SUBSTR（"ABCDEFG",0,¥P1）
.ENDM

MCO      2･･･････････････････････････････[1]
MCO      5･･･････････････････････････････[2]
```

[1] .SDATA "AB"を展開します。
[2] .SDATA "ABCDE"を展開します。

9. 構造化アセンブリ機能

第９章　目次

9.1 構造化アセンブリ機能の概要

構造化アセンブリ機能は、処理を選択したり、繰り返したりする命令を展開する機能です。

9.1.1 処理の選択（.IF）

処理の選択（.IF）は、.IF～.ELSE～.ENDI（.ELSE は省略可能）間のソースステートメントを、.IF で指定した条件に従って、選択実行するような命令を展開します。

.IF で展開した命令を実行すると、次のようになります。

（1）.IF の条件を判定します。

（2）.IF の条件が成立した場合は、.IF～.ELSE 間のソースステートメントを実行します。
.ELSE の省略時は、.IF～.ENDI 間のソースステートメントを実行します。

（3）.IF の条件が不成立の場合は、.ELSE～.ENDI 間のソースステートメントを実行します。
.ELSE の省略時は、.IF～.ENDI 間のソースステートメントは実行しません。

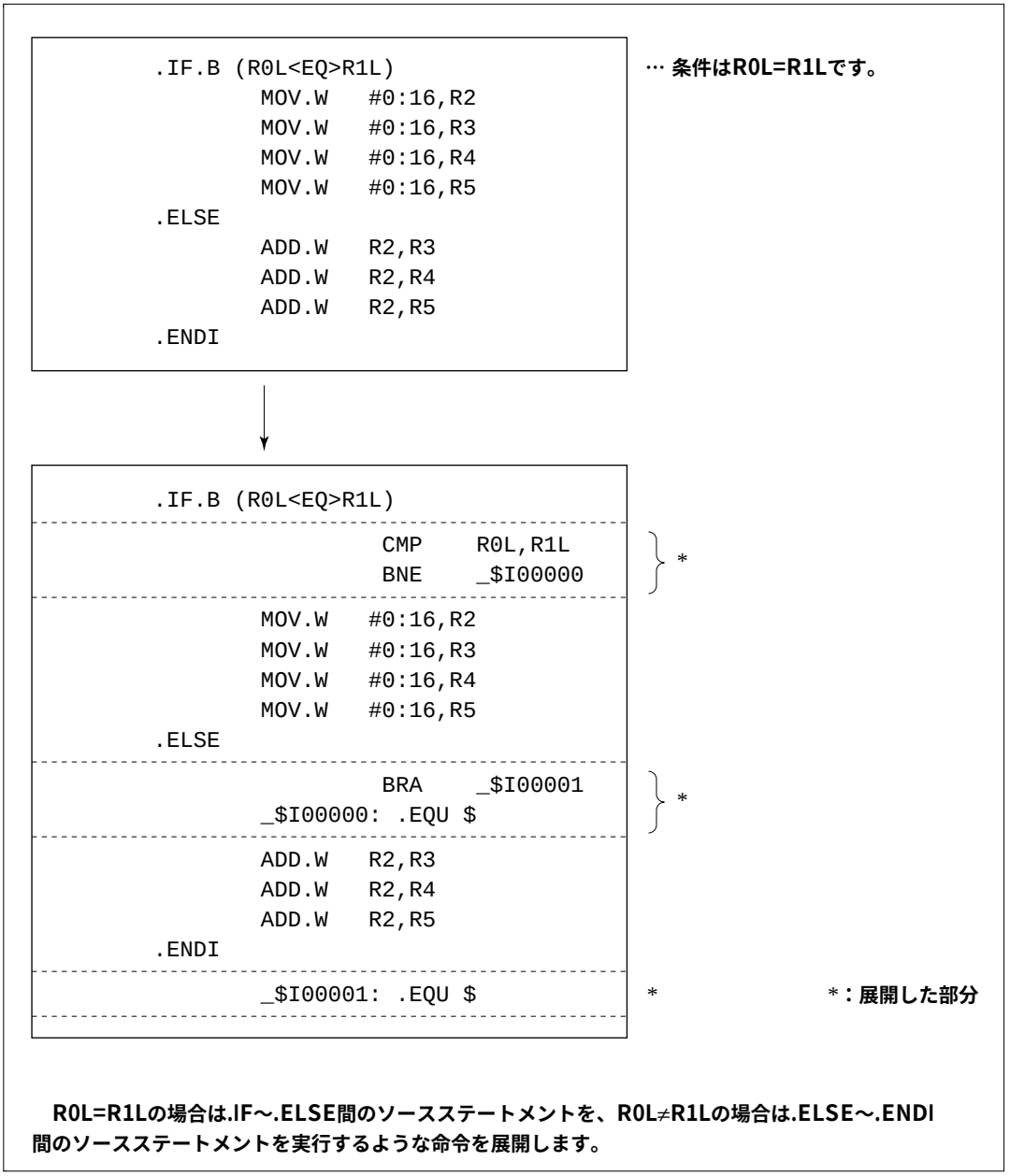

.IF.B (R0L<EQ>R1L)
MOV.W #0:16,R2
MOV.W #0:16,R3
MOV.W #0:16,R4
MOV.W #0:16,R5
.ELSE
ADD.W R2,R3
ADD.W R2,R4
ADD.W R2,R5
.ENDI
… 条件はR0L=R1Lです。
.IF.B (R0L<EQ>R1L)
CMP R0L,R1L
BNE _$I00000
*
MOV.W #0:16,R2
MOV.W #0:16,R3
MOV.W #0:16,R4
MOV.W #0:16,R5
.ELSE
BRA _$I00001
*
_$I00000: .EQU $
ADD.W R2,R3
ADD.W R2,R4
ADD.W R2,R5
.ENDI
_$I00001: .EQU $
*
*：展開した部分
R0L=R1Lの場合は.IF～.ELSE間のソースステートメントを、R0L≠R1Lの場合は.ELSE～.ENDI間のソースステートメントを実行するような命令を展開します。

図 9.1 .IF **使用例**

9.1.2 処理の選択（.SWITCH）

処理の選択（.SWITCH）は、.CASE～.BREAK 間、.OTHERS～.ENDS 間のソースステートメントを、.SWITCH と.CASE で指定した条件に従って、選択実行するような命令を展開します。

.SWITCH で展開した命令を実行すると、次のようになります。

（1）.SWITCH と.CASE の条件を判定します。.CASE との条件は、上から順番に判定します。
（2）.SWITCH と.CASE の条件が成立した時点で、該当する.CASE～.BREAK 間のソースステートメントを実行します。
（3）.SWITCH と.CASE の条件がすべて不成立の場合は、.OTHERS～.ENDS 間のソースステートメントを実行します。

```
        .SWITCH.B (R0L)
        .CASE #0                                    … 条件はR0L=0です。
                MOV.W R1,R4
        .BREAK
        .CASE#1                                     … 条件はR0L=1です。
                MOV.W R2,R4
        .BREAK
        .OTHERS
                MOV.W R3,R4
        .ENDS
```

↓

```
        .SWITCH.B (R0L)
        .CASE #0
                        CMP     #0,R0L          } *
                        BNE     _$S00000        }
                MOV.W   R1,R4
        .BREAK
                        BRA     _$S00001
        .CASE #1
                _$S00000: .EQU  $               }
                        CMP     #1,R0L          } *
                        BNE     _$S00002        }
                MOV.W R2,R4
        .BREAK
                        BRA     _$S00001        *
        .OTHERS
                _$S00002: .EQU  $               *
                MOV.W R3,R4
        .ENDS
                _$S00001: .EQU  $               *
```

*：展開した部分

R0L=0の場合は.CASE #0～.BREAK間のソースステートメントを、R0L=1の場合は.CASE #1～.BREAK間のソースステートメントを、それ以外の場合は.OTHERS～.ENDS間のソースステートメントを実行するような命令を展開します。

図 9.2　.SWITCH **使用例**

9.1.3 処理の繰り返し（.FOR[U]）

処理の繰り返し（.FOR [U]）は、.FOR [U]～.ENDF 間のソースステートメントを、.FOR［U］で指定した条件に従って、条件が成立している間だけ繰り返し実行するような命令を展開します。

処理の繰り返し（.FOR [U]）には、符号付き範囲で繰り返しの条件判定をする.FOR と、符号なし範囲で処理の繰り返しの条件判定をする.FORU があります。

.FOR [U]では、条件判定に使用するループカウンタとループカウンタの初期値、終値、増分値を指定します。

.FOR [U]で展開した命令を実行すると、次のようになります。

（1）ループカウンタに初期値を設定します。
（2）ループカウンタと終値で、条件を判定します。
（3）条件が成立した場合は、.FOR [U]～.ENDF 間のソースステートメントを実行します。また、ループカウンタを増分値で更新します。

（4）条件が不成立の場合は、.FOR [U]～.ENDF 間のソースステートメントは実行しません。また、処理の繰り返しを終了します。

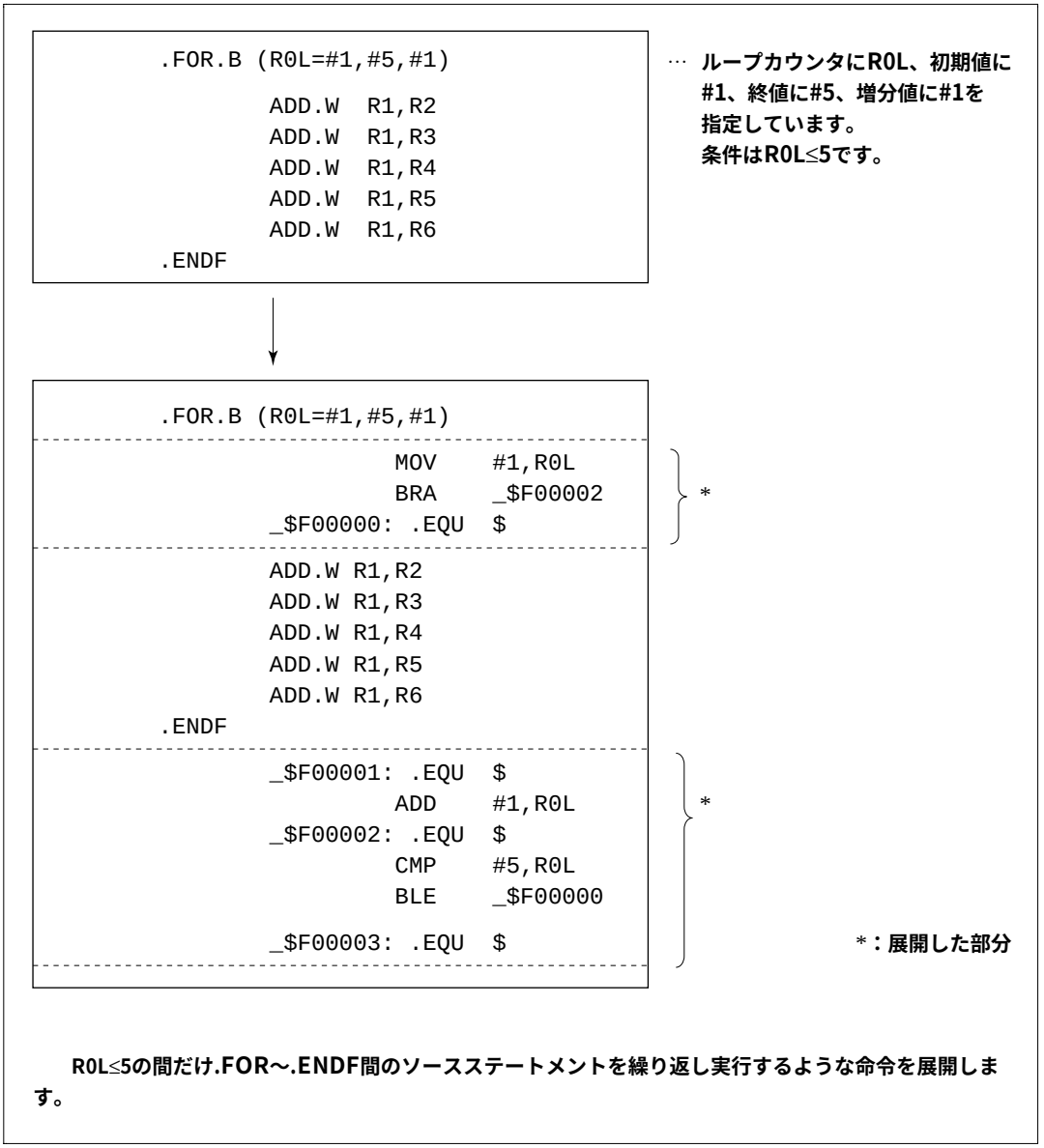

.FOR.B (R0L=#1,#5,#1)
ADD.W R1,R2
ADD.W R1,R3
ADD.W R1,R4
ADD.W R1,R5
ADD.W R1,R6
.ENDF
… ループカウンタにR0L、初期値に#1、終値に#5、増分値に#1を指定しています。
条件はR0L≦5です。
.FOR.B (R0L=#1,#5,#1)
MOV #1,R0L
BRA _$F00002
_$F00000: .EQU $
*
ADD.W R1,R2
ADD.W R1,R3
ADD.W R1,R4
ADD.W R1,R5
ADD.W R1,R6
.ENDF
_$F00001: .EQU $
ADD #1,R0L
_$F00002: .EQU $
CMP #5,R0L
BLE _$F00000
_$F00003: .EQU $
*
*：展開した部分
R0L≦5の間だけ.FOR～.ENDF間のソースステートメントを繰り返し実行するような命令を展開します。

図 9.3 .FOR **使用例**

9.1.4 処理の繰り返し（.WHILE）

処理の繰り返し（.WHILE）は、.WHILE～.ENDW 間のソースステートメントを、.WHILE で指定した条件に従って、条件が成立している間だけ繰り返し実行するような命令を展開します。

.WHILE で展開した命令を実行すると、次のようになります。

（1）条件を判定します。

（2）条件が成立した場合は、.WHILE～.ENDW 間のソースステートメントを実行します。

（3）条件が不成立の場合は、.WHILE～.ENDW 間のソースステートメントは実行しません。また、処理の繰り返しを終了します。

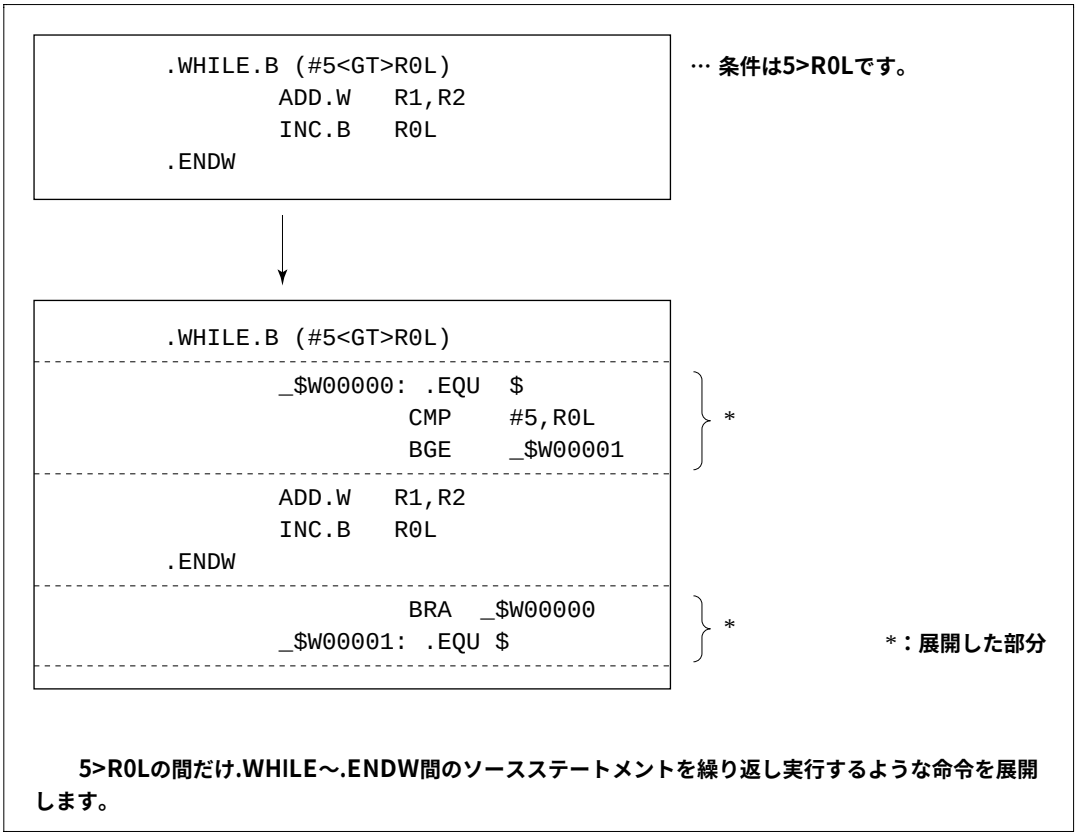


図 9.4 .WHILE **使用例**

9.1.5 処理の繰り返し（.REPEAT）

処理の繰り返し(.REPEAT)は、.REPEAT～.UNTIL 間のソースステートメントを、.UNTIL で指定した条件に従って、条件が成立するまで繰り返し実行するような命令を展開します。

.REPEAT で展開した命令を実行すると、次のようになります。

（1）.REPEAT～.UNTIL 間のソースステートメントを実行します。

（2）条件を判定します。

（3）条件が不成立の場合は、.REPEAT～.UNTIL 間のソースステートメントを実行します。

（4）条件が成立した場合は、処理の繰り返しを終了します。

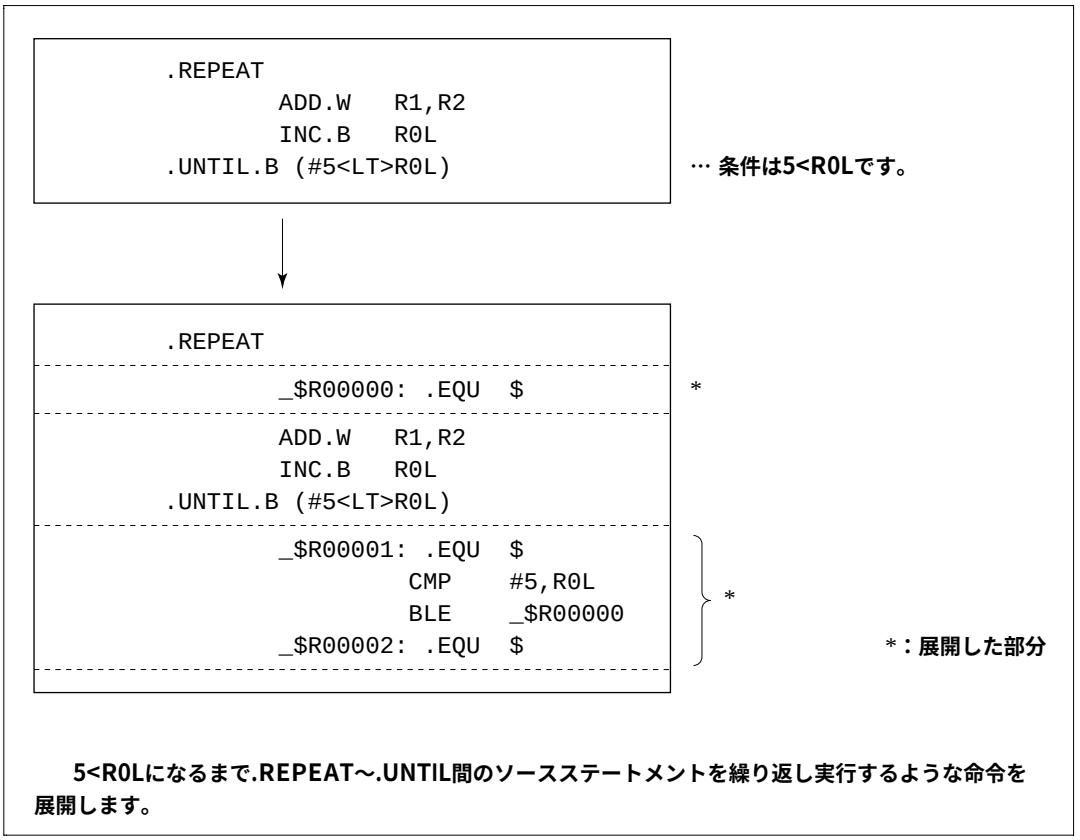


図 9.5 .REPEAT 使用例

9.1.6 構造化アセンブリ機能に関する注意事項

（1）構造化アセンブリ機能は、各制御文に対して一定の形式の命令、シンボルを展開する機能であり、最適化を行なうものではありません。
したがって、各制御文で指定できる内容は、展開する命令の仕様により制限されます。また、効率の悪い命令、不必要なシンボルを展開する場合があります。

■例

```
.IF.B (R0L<LT>#10) ・・・・・・・・・・  展開した命令がエラーになります。
        MOV.M     R1,R2
.ENDI
```

.IF では CMP 命令が展開されていますが、CMP R0L,#10 と展開されるためエラーになります。この場合は、次のようにしてください。

```
.IF.B (#10<GT>R0L) ・・・・・・・・・・  CMP #10,R0L と展開されます。
        MOV.W     R1,R2
.ENDI
```

（2）各制御文では、次の形式のシンボルを展開します。

.IF・・・・・・・・・・・_$I00000～_$I99999
.SWITCH・・・・・_$S00000～_$S99999
.FOR［U］ ・・・_$F00000～_$F99999
.WHILE ・・・・・・_$W00000～_$W99999
.REPEAT・・・・・_$R00000～_$R99999

したがって、同じ名称のシンボルは使用できません。

9.2 構造化アセンブリ機能に関する制御文

表 9.1 に、コンディションコード一覧を示します。

表 9.1 は、後述の構造化アセンブリ制御文で使用するコンディションコードの条件を示しています。各構造化アセンブリ制御文の説明と合わせて参照してください。

表 9.1 **コンディションコード一覧**

項番	コンディションコード	比較実行型のときの条件	コンディションコード 指定型のときの条件
1	＜EQ＞	項 1＝項 2	Z=1
2	＜NE＞	項 1≠項 2	Z=0
3	＜GT＞	項 1＞項 2（符号付き比較）	Z∨（N⊕V）＝0
4	＜LT＞	項 1＜項 2（符号付き比較）	N⊕V=1
5	＜GE＞	項 1≧項 2（符号付き比較）	N⊕V=0
6	＜LE＞	項 1≦項 2（符号付き比較）	Z∨（N⊕V）=1
7	＜HI＞	項 1＞項 2（符号なし比較）	C∨Z=0
8	＜LO＞＜CS＞	項 1＜項 2（符号なし比較）	C=1
9	＜HS＞＜CC＞	項 1≧項 2（符号なし比較）	C=0
10	＜LS＞	項 1≦項 2（符号なし比較）	C∨Z=1
11	＜VC＞		V=0
12	＜VS＞		V=1
13	＜PL＞		N=0
14	＜MI＞		N=1
15	＜T＞		常に条件成立
16	＜F＞		常に条件不成立

【注】： CCR（コンディションコードレジスタ）の N（ネガティブ）フラグ

Z ・・・・・・・・・・・・・・ CCR の Z（ゼロ）フラグ

V ・・・・・・・・・・・・・・ CCR の V（オーバフロー）フラグ

C ・・・・・・・・・・・・・・ CCR の C（キャリ）フラグ

∨ ・・・・・・・・・・・・・・ 論理和

⊕ ・・・・・・・・・・・・・・ 排他的論理和

9.2.1 .IF（処理の選択）

（1）書式

ラベル	オペレーション	オペランド
×	.IF［.s］［:d］	{（項1<cc>項2）｜（<cc>）}
[×	.ELSE［:d］	×]
×	.ENDI	×

s（サイズ） ： **{B｜W｜L}**
d（分岐サイズ） ： **{8｜16}**
cc（コンディションコード） ： **{EQ｜NE｜GT｜LT｜GE｜LE｜HI｜LO｜HS｜LS｜CC｜CS｜VC｜VS｜PL｜MI｜T｜F}**

（2）説明

.IF で条件判定した結果に従って、ソースステートメントを選択実行します。

条件が成立した場合は、.IF～.ELSE 間のソースステートメントを、条件が成立しない場合は、.ELSE～.ENDI 間のソースステートメントを実行します。

.ELSE は省略できます。この場合は、条件が成立した場合に.IF～.ENDI 間のソースステートメントを実行します。

条件判定には、次の 2 種類があります。

(a) 比較実行型

比較実行型は、2 つの項をコンディションコードの条件で判定します。

項には、CMP 命令に指定できるアドレス形式を指定します。

(b) コンディションコード指定型

コンディションコード指定型は、CCR（コンディションコードレジスタ）の状態で条件判定します。

サイズには、比較実行型で比較する項のサイズを指定します。コンディションコード指定型では意味を持ちません。

サイズは、次のとおりです。

B ・・・・・・・・ バイト（1 バイト）
W ・・・・・・・・ ワード（2 バイト）
L ・・・・・・・・ ロングワード（4 バイト）

下線部は、指定を省略した場合の設定です。

分岐サイズには、.IF と.ELSE の分岐サイズがあります。

.IF の分岐サイズには、.IF から.ELSE まで、または.IF から.ENDI まで（.ELSE 省略時）の分岐サイズを指定します。

.ELSE の分岐サイズには、.ELSE から.ENDI までの分岐サイズを指定します。

分岐サイズは、次のとおりです。

8 ・・・・・・・・ 8 ビット

16 ・・・・・・・・ 16 ビット

指定を省略した場合の設定は、

「5.7.4 .DISPSIZE」の FBR

「12.3.3 BR__RELATIVE」

「12.3.4 ［NO］OPTIMIZE」

を参照してください。

コンディションコードの条件については、**「表 9.1 コンディションコード一覧」**を参照してください。

（3）制限事項

（1）H8/300、H8/300L では、サイズに L は指定できません。

（2）H8/300、H8/300L では、分岐サイズに 16 は指定できません。

（3）.IF～.ELSE 間、.ELSE～.ENDI 間、.IF～.ENDI 間（.ELSE 省略時）のソースステートメントのサイズは、指定した分岐サイズを超えることはできません。

分岐サイズに対するソースステートメントの最大サイズは、次のとおりです。

8 ・・・・・・・・ 100 バイト程度

16 ・・・・・・・・ 32,700 バイト程度

（4）本制御文を使用すると_$I00000～_$I99999 のシンボルを展開します。したがって、同じ名称のシンボルは使用できません。

(4) 例

```
.IF.B（R0L<EQ>R1L）
        ADD.B   #1:8,R0L・・・・・・・・・・・・・・・ [1]
        MOV.W   R2,R3 ・・・・・・・・・・・・・・・・・ [1]
        MOV.W   R4,R5 ・・・・・・・・・・・・・・・・・ [1]
.ELSE
        ADD.B   #1:8,R1L・・・・・・・・・・・・・・・ [2]
        MOV.W   R3,R2 ・・・・・・・・・・・・・・・・・ [2]
        MOV.W   R5,R4 ・・・・・・・・・・・・・・・・・ [2]
.ENDI
```

比較実行型の例です。

R0L＝R1L の場合は[1]を、R0L≠R1L の場合は[2]を実行します。

```
.IF.B（#H'10<LT>R0L）
        MOV.W   R1,R2 ・・・・・・・・・・・・・・・・・ [3]
        MOV.W   R1,R3 ・・・・・・・・・・・・・・・・・ [3]
        MOV.W   R1,R4 ・・・・・・・・・・・・・・・・・ [3]
.ENDI
```

比較実行型の例です。

H'10<R0L（符号付き比較）の場合に[3]を実行します。

```
.IF（<NE>）
        ADD.B   #5:8,R0L・・・・・・・・・・・・・・・ [4]
.ELSE
        MOV.B   R0L,R1L・・・・・・・・・・・・・・・・ [5]
.ENDI
```

コンディションコード指定型の例です。

CCR（コンディションコードレジスタ）の Z（ゼロ）フラグが 0 の場合は[4]を、1 の場合は[5]を実行します。

```
        .IF（<VS>）
        MOV.B   #0:8,R0L･･･････････････････ [6]
        MOV.B   #0:8,R1L･･･････････････････ [6]
        MOV.B   #0:8,R2L･･･････････････････ [6]
        .ENDI
```

コンディションコード指定型の例です。

CCR（コンディションコードレジスタ）の V（オーバフロー）フラグが 1 の場合に[6]を実行します。

```
.IF.B（#0＜LE＞R0L）
        .IF.B（#50＜GE＞R0L）
                MOV.B   R0L,R1L･･･････ [7]
                MOV.B   R0L,R2L･･･････ [7]
                MOV.B   R0L,R3L･･･････ [7]
        .ENDI
.ENDI
```

.IF を組み合わせた例です。

0≦R0L≦50（符号付き比較）の場合に[7]を実行します。

9.2.2 .SWITCH（処理の選択）

（1）書式

ラベル	オペレーション	オペランド
×	.SWITCH［.s］	{（レジスタ）｜（CCR）}
×	.CASE［:d］	{項｜<cc>}
[×	.BREAK［:d］	×]
×	.CASE［:d］	{項｜<cc>}
[×	.BREAK［:d］	×]
	⋮	
[×	.OTHERS	×]
×	.ENDS	×

s（サイズ）　：{B｜W｜L}
d（分岐サイズ）　：{8｜16}
cc（コンディションコード）：{EQ｜NE｜GT｜LT｜GE｜LE｜HI｜LO｜HS｜LS｜CC｜CS｜VC｜VS｜PL｜MI｜T｜F}

（2）説明

.SWITCH と.CASE で条件判定した結果に従って、ソースステートメントを選択実行します。

.SWITCH と.CASE で条件が成立した場合は、該当する.CASE～.BREAK 間のソースステートメントを、条件が成立しない場合は、.OTHERS～.ENDS 間のソースステートメントを実行します。

.SWITCH と.CASE の条件判定は、上から順番に判定します。

.BREAK を省略した場合は、直後の.CASE～.BREAK 間、.OTHERS～.ENDS 間のソースステートメントまで実行します。

条件判定には、次の 2 種類があります。

（1）比較実行型

比較実行型は、レジスタと項が等しいか判定します。

.SWITCH には、レジスタを指定します。

.CASE には、CMP 命令のソースオペランドに指定できるアドレス形式を指定します。

（2）コンディションコード指定型

コンディションコード指定型は、CCR（コンディションコードレジスタ）の状態で条件判定します。

.SWITCH には、CCR を指定します。

.CASE には、コンディションコードを指定します。

サイズには、比較実行型で比較するレジスタと項のサイズを指定します。コンディションコード指定型では意味を持ちません。

サイズは、次のとおりです。

B・・・・・・・・・バイト（1 バイト）

W ・・・・・・・・ワード（2 バイト）

L ・・・・・・・・・ロングワード（4 バイト）

下線部は、指定を省略した場合の設定です。

分岐サイズには、.CASE と.BREAK の分岐サイズがあります。

.CASE の分岐サイズには、.CASE から次に現れる.CASE、.OTHERS、.ENDS までの分岐サイズを指定します。

.BREAK の分岐サイズには、.BREAK から.ENDS までの分岐サイズを指定します。

分岐サイズは、次のとおりです。

8 ・・・・・・・・・8 ビット

16 ・・・・・・・・16 ビット

指定を省略した場合の設定は、

「5.7.4　.DISPSIZE」の FBR

「12.3.3　BR_RELATIVE」

「12.3.4　［NO］OPTIMIZE」

を参照してください。

コンディションコードの条件については、**「表 9.1　コンディションコード一覧」**を参照してください。

(3) 制限事項

(1) H8/300、H8/300L では、サイズに L は指定できません。

(2) H8/300、H8/300L では、分岐サイズに 16 は指定できません。

(3) 各.CASE に対応するソースステートメント、各.BREAK～.ENDS 間のソースステートメントのサイズは、指定した分岐サイズを超えることはできません。
分岐サイズに対するソースステートメントの最大サイズは、次のとおりです。
8・・・・・・ 100 バイト程度
16・・・・・ 32,700 バイト程度

(4) 本制御文を使用すると_$S00000～_$S99999 のシンボルを展開します。したがって、同じ名称のシンボルは使用できません。

(4) 例

```
.SWITCH.B（R0L）
.CASE #0
        MOV.W   R1,R4 ･･･････････････････ [1]
.BREAK
.CASE #1
        MOV.W   R2,R4 ･･･････････････････ [2]
.BREAK
.OTHERS
        MOV.W   R3,R4 ･･･････････････････ [3]
.ENDS
```

比較実行型の例です。
R0L=0 の場合は[1]を、R0L=1 の場合は[2]を、それ以外の場合は[3]を実行します。

```
.SWITCH（CCR）
.CASE ＜CS＞
        MOV.W   R0,R3 ･･･････････････････ [4]
.BREAK
.CASE ＜MI＞
        MOV.W   R1,R3 ･･･････････････････ [5]
.ENDS
```

コンディションコード指定型の例です。

CCR（コンディションコードレジスタ）のC（キャリ）フラグが1の場合は[4]を、N（ネガティブ）フラグが1の場合は[5]を実行します。

```
.SWITCH.B（R0L）
  .CASE #0
  .CASE #1
  .CASE #2
          MOV.W   R1,R3 ･･･････････････････ [6]
  .BREAK
  .CASE #3
          MOV.W   R2,R3 ･･･････････････････ [7]
  .ENDS
```

.CASEに対する.BREAKを省略した例です。　R0L=0、R0L=1、R0L=2の場合は[6]を、R0L=3の場合は[7]を実行します。

9.2.3 .FOR［U］（処理の繰り返し）

（1）書式

ラベル	オペレーション	オペランド
×	.FOR［U］ ［.s］ ［:d］	（ループカウンタ＝初期値,終値 ［, ［ {+｜-} ］ 増分値］）
×	.ENDF	×

s（サイズ） ： {B｜W｜L}
d（分岐サイズ） ： {8｜16}

（2）説明

.FOR [U]のループカウンタと終値で条件判定し、条件が成立している間だけ.FOR［U］～.ENDF 間のソースステートメントを繰り返し実行します。

処理の繰り返し（.FOR [U]）には、符号付き範囲で繰り返しの条件判定をする.FOR と、符号なし範囲で繰り返しの条件判定をする.FORU があります。

サイズには、ループカウンタ、初期値、終値、増分値のサイズを指定します。
サイズは、次のとおりです。

B・・・・・・・・・バイト（1 バイト）
W ・・・・・・・・ワード（2 バイト）
L・・・・・・・・・ロングワード（4 バイト）

下線部は、指定を省略した場合の設定です。

分岐サイズには、.FOR [U]から.ENDF までの分岐サイズを指定します。
分岐サイズは、次のとおりです。

8・・・・・・・・・8 ビット
16・・・・・・・・16 ビット

指定を省略した場合の設定は、

「5.7.4 .DISPSIZE」の FBR
「12.3.3 BR__RELATIVE」
「12.3.4 ［NO］OPTIMIZE」

を参照してください。

オペランドの内容は、次のとおりです。

(a) ループカウンタ＝初期値

ループカウンタの初期値を設定します。

ループカウンタには、レジスタを指定します。

初期値には、MOV 命令のソースオペランドに指定できるアドレス形式を指定します。

(b) 終値

ループカウンタと比較する終値を指定します。

処理の繰り返しの条件は、次のとおりです。

増分方向が増加方向 ・・・・・・・ ループカウンタ≦終値

増分方向が減少方向 ・・・・・・・ ループカウンタ≧終値

終値には、CMP 命令のソースオペランドに指定できるアドレス形式を指定します。

(c) 増分値

1 回のループごとにループカウンタに加算、または減算する増分値を指定します。

増分方向には、増加方向の場合はプラス（+）、減少方向の場合はマイナス（–）を指定します。

指定を省略した場合は、プラス（+）を設定します。

増分値には、ADD、SUB 命令のソースオペランドに指定できるアドレス形式を指定します。

指定を省略した場合は、+#1 を設定します。

ループカウンタの数値範囲を示します。無限ループとなる場合がありますので、数値範囲には十分注意してください。

制御文	増分方向	サイズ	ループカウンタの数値範囲（初期値～終値）
.FOR	+	B	-128 ～ 126
		W	-32,768 ～ 32,766
		L	-2,147,483,648 ～ 2,147,483,646
	–	B	127 ～ -127
		W	32,767 ～ -32,767
		L	2,147,483,647 ～ -2,147,483,647
.FORU	+	B	0 ～ 254
		W	0 ～ 65,534
		L	0 ～ 4,294,967,294
	–	B	255 ～ 1
		W	65,535 ～ 1
		L	4,294,967,295 ～ 1

（4）制限事項

（1）H8/300、H8/300L では、サイズに L は指定できません。

（2）H8/300、H8/300L では、分岐サイズに 16 は指定できません。

（3）.FOR［U］～.ENDF 間のソースステートメントのサイズは、指定した分岐サイズを超えることはできません。
分岐サイズに対するソースステートメントの最大サイズは、次のとおりです。
8 ・・・・・・・・・ 100 バイト程度
16 ・・・・・・・・ 32,700 バイト程度

（4）本制御文を使用すると_$F00000～_$F99999 のシンボルを展開します。したがって、同じ名称のシンボルは使用できません。

(5) 例

```
.FOR.B（R0L＝#1,#10,+#1）
        ADD.B   R0L,R1L ･･･････････････････ [1]
.ENDF
```

.FOR の例です。
ループカウンタは R0L、初期値は#1、終値は#10、増分値は+#1 です。
R0L≦10（符号付き比較）の間だけ[1]を繰り返し実行します。

```
.FOR.W（R0＝R1,R2,-R3）
        ADD.B   #1:8,R5L ･･･････････････････ [2]
.ENDF
```

.FOR の例です。
ループカウンタは R0、初期値は R1、終値は R2、増分値は-R3 です。
R0≧R2（符号付き比較）の間だけ[2]を繰り返し実行します。

```
.FORU.B（R0L＝#1,#200,+#1）
        ADD.W   R1,R2 ･･･････････････････････ [3]
        ADD.W   R3,R4 ･･･････････････････････ [3]
        ADD.W   R5,R6 ･･･････････････････････ [3]
.ENDF
```

.FORU の例です。
ループカウンタは R0L、初期値は#1、終値は#200、増分値は+#1 です。
R0L≦200（符号なし比較）の間だけ[3]を繰り返し実行します。

```
.FORU.L（ER0=#H'00000100,#H'000001FC,+#4）
        MOV.L   @ER0,ER2･･･････････････････････････ [4]
        MOV.L   ER2,@（H'00001100:32,ER1）･･･････････ [4]
        ADDS.L  #4,ER1 ･･･････････････････････････ [4]
.ENDF
```

.FORU の例です。
ループカウンタは ER0、初期値は#H'00000100、終値は#H'000001FC、増分値は+#4 です。
ER0≦H'000001FC（符号なし比較）の間だけ[4]を繰り返し実行します。

```
.FOR.B （R0L=#1,#10,+#1）･･･････････････ [5]
        .FOR.B （R1L=#1,#10,+#1）･･････ [6]
                MOV.B   R0L,R2L
                ADD.B   R1L,R2L
                MOV.B   R2L,@ER3
                ADDS.L  #1,ER3
        .ENDF･･･････････････････････････ [6]
.ENDF ･･･････････････････････････････････ [5]
```

.FOR を組み合わせた例です。
[5]の.FOR の繰り返しの中で、[6]の.FOR の繰り返しを実行します。

9.2.4 .WHILE（処理の繰り返し）

（1）書式

ラベル	オペレーション	オペランド
×	.WHILE［.s］［:d］	{（項1<cc>項2）｜（<cc>）}
×	.ENDW	×

s（サイズ） ： **{B｜W｜L}**
d（分岐サイズ） ： **{8｜16}**
cc（コンディションコード） ： **{EQ｜NE｜GT｜LT｜GE｜LE｜HI｜LO｜HS｜LS｜CC｜CS｜VC｜VS｜PL｜MI｜T｜F}**

（2）説明

.WHILE で条件判定し、条件が成立している間だけ.WHILE～.ENDW 間のソースステートメントを繰り返し実行します。

条件判定には、次の 2 種類があります。

（a）比較実行型

比較実行型は、2 つの項をコンディションコードの条件で判定します。

項には、CMP 命令に指定できるアドレス形式を指定します。

（b）コンディションコード指定型

コンディションコード指定型は、CCR（コンディションコードレジスタ）の状態で条件判定します。

サイズには、比較実行型で比較する項のサイズを指定します。コンディションコード指定型では意味を持ちません。

サイズは、次のとおりです。

B・・・・・・・・・バイト（1 バイト）
W ・・・・・・・・ワード（2 バイト）
L・・・・・・・・・ロングワード（4 バイト）

下線部は、指定を省略した場合の設定です。

分岐サイズには、.WHILE から.ENDW までの分岐サイズを指定します。
分岐サイズは、次のとおりです。

8 ・・・・・・・・・ 8 ビット
16 ・・・・・・・・ 16 ビット

指定を省略した場合の設定は、

「5.7.4　.DISPSIZE」の FBR
「12.3.3　BR__RELATIVE」
「12.3.4　［NO］OPTIMIZE」

を参照してください。

コンディションコードの条件については、**「表 9.1　コンディションコード一覧」**を参照してください。

（3）制限事項

（a）H8/300、H8/300L では、サイズに L は指定できません。

（b）H8/300、H8/300L では、分岐サイズに 16 は指定できません。

（c）.WHILE～.ENDW 間のソースステートメントのサイズは、指定した分岐サイズを超えることはできません。
分岐サイズに対するソースステートメントの最大サイズは、次のとおりです。

8 ・・・・・・・・・ 100 バイト程度
16 ・・・・・・・・ 32,700 バイト程度

（d）本制御文を使用すると_$W00000～_$W99999 のシンボルを展開します。したがって、同じ名称のシンボルは使用できません。

（4）例

```
.WHILE.B（#50<GT>R0L）
        ADD.W   R1,R2 ･･･････････････････ [1]
        SUB.W   R1,R3 ･･･････････････････ [1]
        ADD.B   #1:8,R0L･･･････････････････ [1]
.ENDW
```

比較実行型の例です。
50＞R0L（符号付き比較）の間だけ[1]を繰り返し実行します。

```
.WHILE.W （R0<LS>R1）
        SUB.B   R2L,R3L･･･････････････････ [2]
        SUB.B   R2L,R4L･･･････････････････ [2]
        SUB.W   R5,R1 ･･･････････････････ [2]
.ENDW
```

比較実行型の例です。
R0≦R1（符号なし比較）の間だけ[2]を繰り返し実行します。

```
.WHILE（<NE>）
        MOV.L   @ER2,ER4･･･････････････････ [3]
        MOV.L   ER4,@ER3･･･････････････････ [3]
        ADDS.L  #4,ER2 ･･･････････････････ [3]
        ADDS.L  #4,ER3 ･･･････････････････ [3]
        SUB.B   R1L,R0L ･･･････････････････ [3]
.ENDW
```

コンディションコード指定型の例です。
CCR（コンディションコードレジスタ）のZ（ゼロ）フラグが0の間だけ[3]を繰り返し実行します。

```
.WHILE（<PL>）
        MOV.L   ER2,@ER1･･･････････････････ [4]
        ADDS.L  #4,ER1 ･･･････････････････ [4]
        MOV.L   ER3,@ER1･･･････････････････ [4]
        ADDS.L  #4,ER1 ･･･････････････････ [4]
        ADD.W   #-1,R0 ･･･････････････････ [4]
.ENDW
```

コンディションコード指定型の例です。

CCR（コンディションコードレジスタ）のN（ネガティブ）フラグが0の間だけ[4]を繰り返し実行します。

```
.WHILE.L（#H'000200<HI>ER0） ･････････････ [5]
        MOV.B   #0:8,R2L
        MOV.B   #H'FF:8,R3L
        .WHILE.B（#H'00<NE>R3L）･･･････ [6]
                MOV.B   @ER0,R3L
                ADDS.L  #1,ER0
                ADD.B   #1:8,R2L
        .ENDW･･･････････････････････････ [6]
        MOV.B   R2L,@ER1
        ADDS.L  #1,ER1
.ENDW ･･･････････････････････････････････ [5]
```

.WHILEを組み合わせた例です。

[5]の.WHILEの繰り返しの中で、[6]の.WHILEを繰り返し実行します。

9.2.5 .REPEAT（処理の繰り返し）

（1）書式

ラベル	オペレーション	オペランド
×	.REPEAT	×
×	.UNTIL［.s］	{（項1<cc>項2）｜（<cc>）}

s（サイズ）　　：{B｜W｜L}
cc（コンディションコード）：{EQ｜NE｜GT｜LT｜GE｜LE｜HI｜LO｜HS｜LS｜CC｜CS｜VC｜VS｜PL｜MI｜T｜F}

（2）説明

.UNTIL で条件判定し、条件が成立するまで.REPEAT～.UNTIL 間のソースステートメントを繰り返し実行します。

.UNTIL で条件判定するため、必ず 1 回は.REPEAT～.UNTIL 間のソースステートメントを実行します。

条件判定には、次の 2 種類があります。

(a) 比較実行型

比較実行型は、2 つの項をコンディションコードの条件で判定します。

項には、CMP 命令に指定できるアドレス形式を指定します。

(b) コンディションコード指定型

コンディションコード指定型は、CCR（コンディションコードレジスタ）の状態で条件判定します。

サイズには、比較実行型で比較する項のサイズを指定します。コンディションコード指定型では意味を持ちません。

サイズは、次のとおりです。

B・・・・・・・・・バイト（1 バイト）
W ・・・・・・・・ワード（2 バイト）
L・・・・・・・・・ロングワード（4 バイト）

下線部は、指定を省略した場合の設定です。

コンディションコードの条件については、**「表 9.1　コンディションコード一覧」**を参照してください。

(3) 制限事項

(1) H8/300、H8/300L では、サイズに L は指定できません。

(2) .REPEAT～.UNTIL 間のソースステートメントのサイズは、次のサイズを超えることはできません。

H8S/2600 アドバンストモード‥‥‥‥ 32,700 バイト程度
H8S/2600 ノーマルモード‥‥‥‥‥‥‥ 32,700 バイト程度
H8S/2000 アドバンストモード‥‥‥‥ 32,700 バイト程度
H8S/2000 ノーマルモード‥‥‥‥‥‥‥ 32,700 バイト程度
H8/300H アドバンストモード ‥‥‥‥ 32,700 バイト程度
H8/300H ノーマルモード ‥‥‥‥‥‥‥ 32,700 バイト程度
H8/300 ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ 100 バイト程度
H8/300L ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ 100 バイト程度

(3) 本制御文を使用すると_$R00000～_$R99999 のシンボルを展開します。したがって、同じ名称のシンボルは使用できません。

(4) 例

```
.REPEAT
        MOV.L   @ER0,ER2‥‥‥‥‥‥‥‥‥‥‥‥‥ [1]
        MOV.L   ER2,@ER1‥‥‥‥‥‥‥‥‥‥‥‥‥ [1]
        ADDS.L  #4,ER0 ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ [1]
        ADDS.L  #4,ER1 ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ [1]
.UNTIL.L（#H'001000<LS>ER0）
```

比較実行型の例です。
H'001000≦ER0（符号なし比較）が成立するまで[1]を繰り返し実行します。

```
.REPEAT
        ADD.W   R2,R3 ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ [2]
        ADD.W   R2,R4 ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥ [2]
        SUB.B   R1L,R0L ‥‥‥‥‥‥‥‥‥‥‥‥‥‥ [2]
.UNTIL（<EQ>）
```

コンディションコード指定型の例です。
CCR（コンディションコードレジスタ）の Z（ゼロ）フラグが 1 になるまで[2]を実行します。

9.2.6 .BREAK（処理の繰り返しの中断終了）

（1）書式

ラベル	オペレーション	オペランド
×	.BREAK［:d］	×

d（分岐サイズ）：｛8｜16｝

（2）説明

.FOR［U］、.WHILE、.REPEAT の繰り返し処理で、.BREAK 以下のソースステートメントを実行しないで繰り返し処理を終了します。具体的には、.FOR［U］、.WHILE、.REPEAT の繰り返し処理で、それぞれ.ENDF、.ENDW、.UNTIL へ分岐して繰り返し処理を終了します。

分岐サイズには、.BREAK から.ENDF、.ENDW、.UNTIL までの分岐サイズを指定します。
分岐サイズは、次のとおりです。

8 ・・・・・・・・・ 8 ビット

16 ・・・・・・・・ 16 ビット

指定を省略した場合の設定は、

「5.7.4 .DISPSIZE」の FBR

「12.3.3 BR_RELATIVE」

「12.3.4 ［NO］OPTIMIZE」

を参照してください。

本制御文は、.SWITCH でも使用します。
.SWITCH での使用方法については、「9.2.2 .SWITCH」を参照してください。

（3）制限事項

H8/300、H8/300L では、分岐サイズに 16 は指定できません。

（4）例

```
.WHILE（<T>）
        .IF.B（#10＜LE＞R0L）
                .BREAK
        .ENDI
        ADD.W   R1,R2
        INC.B   R0L
.ENDW
```

10≦R0L の場合に繰り返し処理を中断終了します。

9.2.7 .CONTINUEーーー処理の繰り返しの中断継続

(1) 書式

ラベル	オペレーション	オペランド
×	.CONTINUE［:d]	×

d（分岐サイズ）：｛8｜16｝

(2) 説明

.FOR [U]、.WHILE、.REPEAT の繰り返し処理で、.CONTINUE 以下のソースステートメントを実行しないで繰り返し処理を継続します。具体的には、.FOR [U]、.WHILE、.REPEAT の繰り返し処理で、それぞれの繰り返し処理の条件判定に分岐します。

分岐サイズには、.CONTINUE から.ENDF、.WHILE、.UNTIL までの分岐サイズを指定します。

分岐サイズは、次のとおりです。

8 ・・・・・・・ 8 ビット

16 ・・・・・・ 16 ビット

指定を省略した場合の設定は、

「5.7.4 .DISPSIZE」の FBR

「12.3.3 BR_RELATIVE」

「12.3.4 ［NO］OPTIMIZE」

を参照してください。

(3) 制限事項

H8/300、H8/300L では、分岐サイズに 16 は指定できません。

(4) 例

```
.WHILE.B（#10<GT>R0L）
        INC.B   R0L
        INC.B   R1L
        .IF.B（#10<LT>R1L）
                .CONTINUE
        .ENDI
        ADD.W   R2,R3 ・・・・・・・・・・・・・・・・・ [1]
.ENDW
```

10<R1L の場合には[1]を実行しません。

10. 操作編概要

第 10 章　目次

10.1 アセンブラの位置付け

図 10.1 に、プログラム開発におけるアセンブラの位置付けを示します。

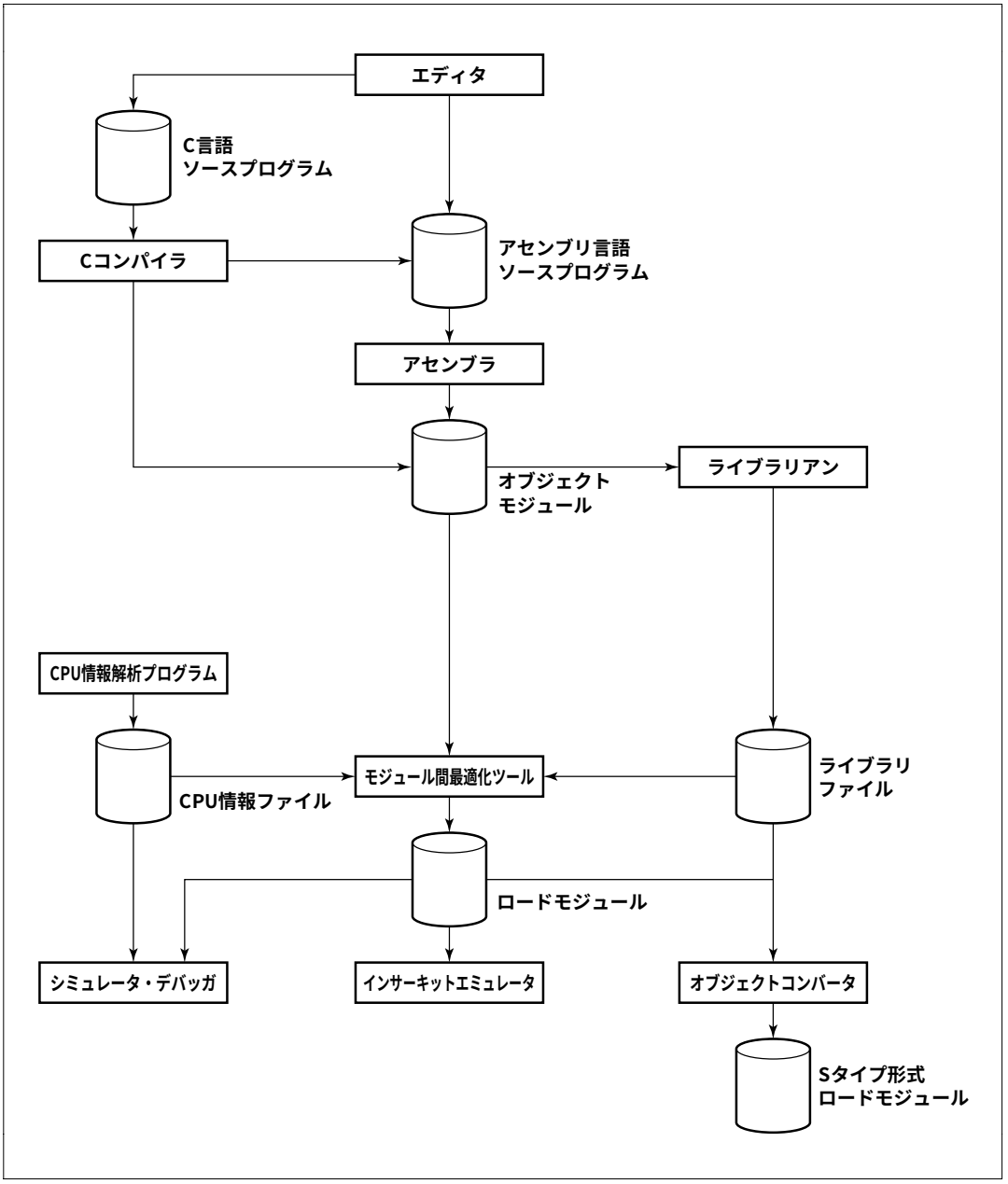


図 10.1 **アセンブラの位置付け**

10.2 入出力構成

図 10.2 に、アセンブラの入出力構成を示します。

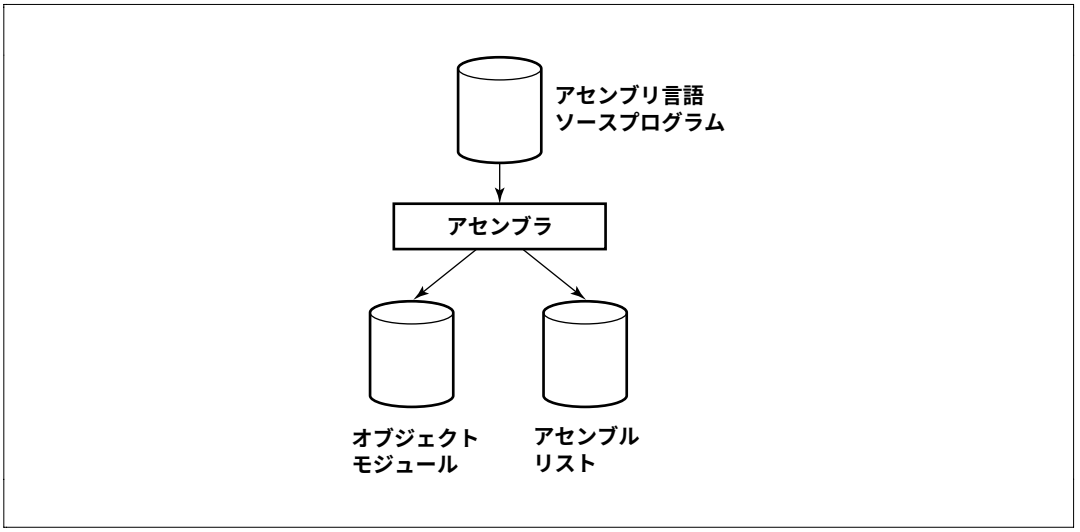


図 10.2 **入出力構成**

11. アセンブラの起動

第 11 章　目次

11.1 コマンドの形式

アセンブラは、コマンドラインで入力ファイル名、出力ファイル名などを指定して起動します。また、コマンドラインオプションでアセンブル方法を指定することができます。

アセンブラを起動するコマンドラインの形式は、次のとおりです。

```
> asm38  △入力ファイル名［,入力ファイル名…］  ［［△］-コマンドラインオプション［［△］-コマンドラインオプション…］］
  [1]              [2]                                  [3]
```

[1] アセンブラの起動コマンドです。

[2] アセンブルするソースファイルの名称を指定します。

2 つ以上の入力ファイルを指定する場合は、コンマ（,）で区切って指定します。

2 つ以上の入力ファイルを指定した場合は、入力ファイルを指定順に連結して 1 つのソースファイルとしてアセンブルします。

[3] コマンドラインオプションを指定します。

■例

```
>  asm38  aaa.mar -cpu=2600a -list -debug
```

ソースファイル aaa.mar をアセンブルします。

【注】 MS-DOS 版では、コマンドラインオプションの区切りにはハイフン（-）とスラッシュ（/）の両方が指定できます。

11.2 H38CPU 環境変数

アセンブラは、H38CPU 環境変数に設定した CPU 用にアセンブルします。
CPU 種別の設定方法は、以下のとおりです。

（1）UNIX の場合

（a）C Shell の場合

setenv　H38CPU*　CPU 種別［:アドレス空間のビット幅］

（b）Bourne/Korn Shell の場合

H38CPU*＝CPU 種別［:アドレス空間のビット幅］
export H38CPU*

（2）MS-DOS の場合

SET H38CPU*＝CPU 種別［:アドレス空間のビット幅］

指定できる CPU 種別は 2600A、2600N、2000A、2000N、300HA、300HN、300、300L です。
CPU 種別がアドバンストモードの場合、アドレス空間のビット幅が指定できます。
表 11.1 に、アドレス空間のビット幅を示します。

表 11.1　アドレス空間のビット幅

項番	CPU 種別	アドレス空間のビット幅	デフォルトサイズ
1	2600A	32、28、24、20	24
2	2000A	32、28、24、20	24
3	300HA	24、20	24

本環境変数で指定した CPU 種別は、CPU コマンドラインオプション、かつ.CPU 制御命令の指定がない場合に有効です。

【注】 ＊　本環境変数は、必ず大文字で指定してください。

11.3 使用するファイル

アセンブラでは、次の 3 種類のファイルを使用します。

（1）ソースファイル

ソースプログラムのファイルです。

（2）オブジェクトファイル

オブジェクトモジュールを出力するファイルです。

（3）アセンブルリストファイル

アセンブルリストを出力するファイルです。

（1）～（3）については、各ファイルの指定でファイル形式を省略した場合、次のように設定します。

・ソースファイル ・・・・・・・・・・・・・・・xxx.mar、xxx.src
・オブジェクトファイル ・・・・・・・・・xxx.obj
・アセンブルリストファイル ・・・・・xxx.lis

【注】 MS-DOS では、ファイルの拡張子は大文字で設定されます。

11.4 OS へのリターン値

アセンブラは、その実行結果をリターン値として OS（オペレーティングシステム）へ返します。

表 11.2 に、OS へのリターン値を示します。

表 11.2 OS **へのリターン値**

項番	実行結果	リターン値	
		MS-DOS	UNIX
1	**正常終了**	0	0
2	**ウォーニング**	0	0
3	**エラー**	2	1
4	**フェイタルエラー**	4	1

OS へのリターン値は、ABORT コマンドラインオプションの指定により変更できます。ABORT コマンドラインオプションについては、**「**12.3.6 ABORT**」**を参照してください。

12. コマンドラインオプション

第 12 章　目次

12.1 コマンドラインオプションの種類

アセンブラの起動時に、コマンドラインオプションを指定することで、アセンブル方法を指定することができます。

表 12.1 に、コマンドラインオプション一覧を示します。

表 12.1 **コマンドラインオプション一覧**

分類	コマンドラインオプション	機能	参照
CPU **に関するもの**	CPU	CPU **の指定**	12.2.1
オブジェクトモジュールに関するもの	[NO]OBJECT	**オブジェクトの出力**	12.3.1
	[NO]DEBUG	**デバッグ情報の出力**	12.3.2
	BR_RELATIVE	**ディスプレースメントサイズの設定**	12.3.3
	[NO]OPTIMIZE	**最適化の指定**	12.3.4
	GOPTIMIZE	**モジュール間最適化情報の出力**	12.3.5
	[NO]EXCLUDE	**未参照外部参照シンボルの情報の出力抑止**	12.3.6
	ABORT	OS **へのリターン値の変更、エラー時のオブジェクト出力抑止**	12.3.7
	EXPAND	**プリプロセッサの展開結果の出力**	12.3.8
	ABS8 ABS16	8 **または** 16 **ビット絶対アドレス形式アクセスシンボルの指定**	12.3.9
アセンブルリストに関するもの	[NO]LIST	**アセンブルリストの出力**	12.4.1
	[NO]SOURCE	**ソースプログラムリストの出力**	12.4.2
	[NO]CROSS_REFERENCE	**クロスリファレンスリストの出力**	12.4.3
	[NO]SECTION	**セクション情報リストの出力**	12.4.4
	[NO]SHOW	**ソースプログラムリストの部分出力**	12.4.5
	LINES	**リスト行数の設定**	12.4.6
	COLUMNS	**リスト桁数の設定**	12.4.7
ファイルインクルード機能に関するもの	INCLUDE	**インクルードファイルのディレクトリ指定**	12.5.1
条件付きアセンブリ機能に関するもの	ASSIGNA	**整数型プリプロセッサ変数の定義**	12.6.1
	ASSIGNC	**文字型プリプロセッサ変数の定義**	12.6.2
	DEFINE	**文字列の置き換えの定義**	12.6.3
文字列、コメント内文字に関するもの	SJIS	**文字列、コメント内の日本語記述**	12.7.1
	EUC	**文字列、コメント内の日本語記述**	12.7.2
	OUTCODE	**オブジェクトコード内の漢字コード指定**	12.7.3

分類	コマンドラインオプション	機能	参照
文字列、コメント内文字に関するもの	LATIN1	**文字列・コメント内の** LATIN1 **コード記述**	12.7.4
コマンドラインの指定方法に関するもの	SUBCOMMAND	**サブコマンドファイルの指定**	12.8.1

12.2 CPU に関するコマンドラインオプション

12.2.1 CPU（CPU の指定）

(1) 書式

CPU=CPU 種別［：アドレス空間のビット幅］

CPU 種別： {2600A｜2600N｜2000A｜2000N｜300HA｜300HN｜300｜300L}

CPU 種別	アドレス空間のビット幅	デフォルトサイズ
2600A	20, 24, 28, 32	24
2000A	20, 24, 28, 32	24
300HA	20, 24	24

下線部は省略形です。

(2) 説明

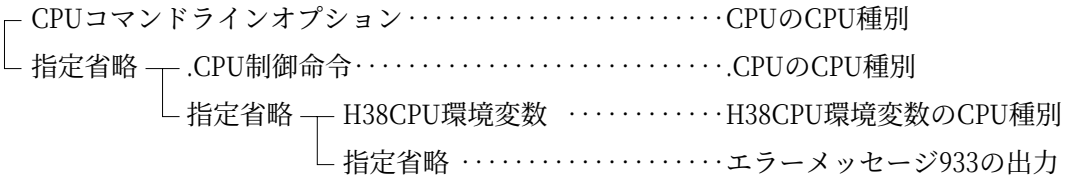
アセンブルするソースプログラムの対象とする CPU を指定します。
CPU 種別がアドバンストモードのみアドレス空間のビット幅が指定できます。
アクセス可能な範囲は、アドレス空間のビット幅の指定により異なります。

CPU 種別は、次のとおりです。

CPU＝2600A:32・・・・・・・・・・・・・・・・・・・・H8S/2600 アドバンストモード
CPU＝2600A:28・・・・・・・・・・・・・・・・・・・・H8S/2600 アドバンストモード
CPU＝2600A[:24] ・・・・・・・・・・・・・・・・・・H8S/2600 アドバンストモード
CPU＝2600A:20・・・・・・・・・・・・・・・・・・・・H8S/2600 アドバンストモード
CPU＝2600N ・・・・・・・・・・・・・・・・・・・・・H8S/2600 ノーマルモード
CPU＝2000A:32・・・・・・・・・・・・・・・・・・・・H8S/2000 アドバンストモード
CPU＝2000A:28・・・・・・・・・・・・・・・・・・・・H8S/2000 アドバンストモード
CPU＝2000A[:24] ・・・・・・・・・・・・・・・・・・H8S/2000 アドバンストモード
CPU＝2000A:20・・・・・・・・・・・・・・・・・・・・H8S/2000 アドバンストモード
CPU＝2000N ・・・・・・・・・・・・・・・・・・・・・H8S/2000 ノーマルモード
CPU＝300HA[:24]・・・・・・・・・・・・・・・・・・H8/300H アドバンストモード
CPU＝300HA:20 ・・・・・・・・・・・・・・・・・・・H8/300H アドバンストモード
CPU＝300HN・・・・・・・・・・・・・・・・・・・・・H8/300H ノーマルモード
CPU＝300・・・・・・・・・・・・・・・・・・・・・・・H8/300
CPU＝300L ・・・・・・・・・・・・・・・・・・・・・・H8/300L

アセンブラは、指定した CPU 用にアセンブルします。

(3) 制御命令との関係



(4) 例

```
> asm38 aaa.mar -cpu=2600a:32
```

H8S/2600 アドバンストモードの 4G モード用にアセンブルします。

```
> asm38 aaa.mar -cpu=2600a:28
```

H8S/2600 アドバンストモードの 256M モード用にアセンブルします。

```
> asm38 aaa.mar -cpu=2600a:24
```

H8S/2600 アドバンストモードの 16M モード用にアセンブルします。

```
> asm38 aaa.mar -cpu=2600a:20
```

H8S/2600 アドバンストモードの 1M モード用にアセンブルします。

12.3 オブジェクトモジュールに関するコマンドラインオプション

12.3.1 [NO]OBJECT（オブジェクトの出力）

(1) 書式

OBJECT［=ファイル名］
NOOBJECT

下線部は省略形です。

(2) 説明

オブジェクトモジュールの出力、出力抑止を指定します。

OBJECT・・・・・・・・・・・・出力
NOOBJECT・・・・・・・・・出力抑止

ファイル名には、オブジェクトモジュールを出力するファイル名を指定します。

ファイル名を省略した場合は、ソースファイルのファイル形式を.obj にしたファイル名を設定します。

また、指定したファイル名でファイル形式を省略した場合は、ファイル形式に.obj を設定します。

【注】 ファイル名には、入力ファイル名と重複する名称を指定しないでください。

(3) 制御命令との関係

- OBJECTコマンドラインオプション・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・出力
- NOOBJECTコマンドラインオプション ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・出力抑止
- 指定省略
  - .OUTPUT OBJ ・・・出力
  - .OUTPUT NOOBJ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・出力抑止
  - 指定省略 ・・出力

(4) 例

```
> asm38 aaa.mar -object
```

オブジェクトモジュールを出力します。ファイル名は、aaa.obj です。

```
> asm38 aaa.mar -noobject
```

オブジェクトモジュールを出力しません。

12.3.2 [NO]DEBUG（デバッグ情報の出力）

（1）書式

DEBUG
NODEBUG

下線部は省略形です。

（2）説明

デバッグ情報の出力、出力抑止を指定します。

DEBUG ・・・・・・・・・・・・出力
NODEBUG ・・・・・・・・・出力抑止

デバッグ情報を出力する場合、オブジェクトファイルを出力するディレクトリに「dwfinf」という名前のディレクトリを自動生成し、オブジェクトファイル名と同じ主ファイル名で拡張子「dwi」のデバッグ付加情報（ELF/WARF 付加情報）ファイルを出力します。

本コマンドラインオプションは、オブジェクトモジュールの出力時に有効です。

（3）制御命令との関係

- DEBUGコマンドラインオプション ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・出力
- NODEBUGコマンドラインオプション ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・出力抑止
- 指定省略
  - .OUTPUT DBG ・・出力
  - .OUTPUT NODBG ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・出力抑止
  - 指定省略 ・・・出力抑止

（4）例

```
> asm38 aaa.mar -debug
```

デバッグ情報を出力します。

```
> asm38 aaa.mar -nodebug
```

デバッグ情報を出力しません。

12.3.3 BR_RELATIVE（ディスプレースメントサイズの設定）

（1）書式

BR_RELATIVE=**ビット数**

ビット数： {8｜16}

下線部は省略形です。

（2）説明

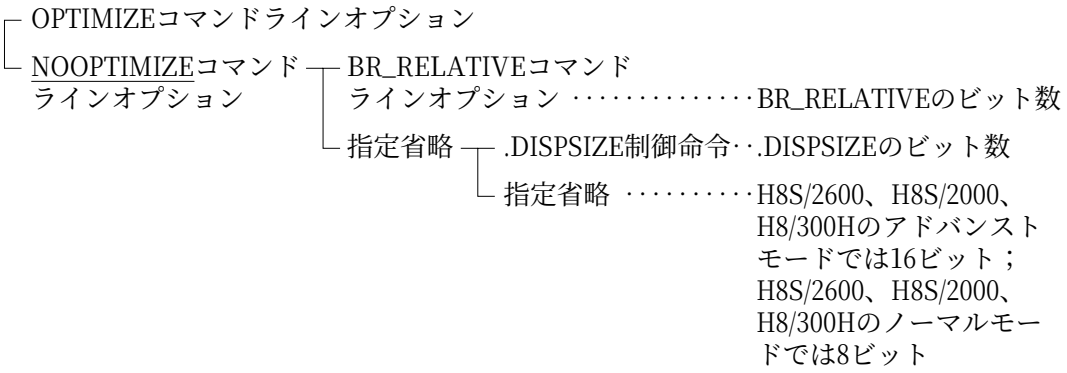
分岐命令のディスプレースメントが前方参照値の場合のディスプレースメントのデフォルトサイズを設定します。

BR_RELATIVE＝8‥‥‥‥ 8 ビット
BR_RELATIVE＝16‥‥‥‥ 16 ビット

本コマンドラインオプションの対象となるのは、ディスプレースメントサイズ（:8、:16）の指定かつ、OPTIMIZE コマンドラインオプションの指定がない場合のディスプレースメントです。

本コマンドラインオプションは、H8S/2600、H8S/2000、H8/300H のアドバンストモードと H8S/2600、H8S/2000、H8/300H のノーマルモードで有効です。
H8/300、H8/300L では、BR_RELATIVE＝8 固定なので意味を持ちません。

（3）制御命令との関係



下線部は、指定を省略した場合の設定です。

ディスプレースメントサイズの最適化については、
「12.3.4　[NO]OPTIMIZE」を参照してください。

（4）例

```
> asm38 aaa.mar -br_relative=16
```

前方参照の分岐命令のディスプレースメントのデフォルトサイズを 16 ビットにします。

```
> asm38 aaa.mar -br_relative=8
```

前方参照の分岐命令のディスプレースメントのデフォルトサイズを 8 ビットにします。

12.3.4 [NO]OPTIMIZE（最適化の指定）

（1）書式

OPTIMIZE
NOOPTIMIZE

下線部は省略形です。

（2）説明

PC 相対形式、ディスプレースメント付きレジスタ間接形式のディスプレースメントサイズと絶対アドレス形式の絶対アドレスサイズの最適化、最適化抑止を指定します。

本コマンドラインオプションの対象となるのは、ディスプレースメントサイズ（:8、:16）、絶対アドレスの確保サイズ（:8、:16、:24、:32）の指定がない実行命令です。

OPTIMIZE・・・・・・・・・・最適化
NOOPTIMIZE・・・・・・・最適化抑止

PC 相対形式のディスプレースメントの値によってディスプレースメントサイズを次のように設定します。

CPU 種別が H8S/2600 アドバンストモードで最適化を指定しない場合

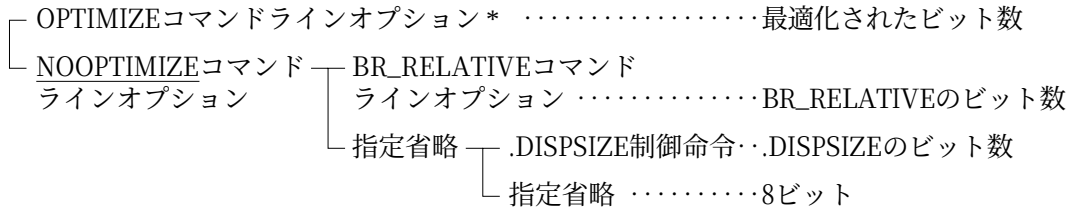
ディスプレースメント値	絶対値	–32768～32767	16ビット*
	相対値		16ビット
	外部参照値		16ビット

CPU 種別が H8S/2600 アドバンストモードで最適化を指定した場合

ディスプレースメント値	絶対値	–128～127	8ビット
		–32768～–129、128～32767	16ビット
	相対値		16ビット
	外部参照値		16ビット

【注】 * 命令より前方に定義した絶対シンボルを参照した場合に限ります。

（3）制御命令との関係



下線部は、指定を省略した場合の設定です。

【注】 ＊ OPTIMIZE **コマンドラインオプションは、オブジェクトモジュールの出力に関するコマンドラインオプション（**BR_RELATIVE**）、制御命令（**.DISPSIZE**）より優先します。**

（4）例

```
> asm38 aaa.mar -optimize
```

オブジェクトモジュールの最適化を行います。

```
> asm38 aaa.mar
```

オブジェクトモジュールの最適化を行いません。

12.3.5 GOPTIMIZE（モジュール間最適化情報の出力）

（1）書式

GOPTIMIZE

下線部は省略形です。

（2）説明

GOPTIMIZE オプションが指定された場合は、オブジェクトファイル出力ディレクトリに「ch38iop」という名前のディレクトリを自動生成し、オブジェクトファイル名と同じファイル名で拡張子「iop」の付加情報ファイルを出力します。

GOPTIMIZE オプションの指定がない場合は、付加情報ファイルは出力しません。

モジュール間最適化ツールの詳細は、「**13．モジュール間最適化のオプション・環境変数**」を参照してください。

（3）例

```
asm38 file.src -GOPTIMIZE
```

ch38iop というディレクトリを生成し、file.iop というファイルを出力します。

（4）注意事項

コードセクション内にデータ領域を確保する制御命令がある場合、モジュール間最適化ツールでエラーとなります。

12.3.6 [NO]EXCLUDE（未参照外部参照シンボルの情報の出力抑止）

（1）書式

EXCLUDE
NOEXCLUDE

下線部は省略形です。

（2）説明

未参照外部参照シンボルのデバッグ情報の出力、出力抑止を指定します。

EXCLUDE・・・・・・・・・・出力抑止
NOEXCLUDE・・・・・・・出力

下線部は、指定を省略した場合の設定です。

未参照外部参照シンボルの情報を出力抑止することにより、オブジェクトモジュールのサイズを小さくできます。

（3）例

```
> asm38 aaa.mar -exclude
```

未参照外部参照シンボルの情報を出力しません。

```
> asm38 aaa.mar -noexclude
```

未参照外部参照シンボルの情報を出力します。

12.3.7 ABORT（OS へのリターン値の変更、エラー時のオブジェクト出力抑止）

（1）書式

ABORT=**エラーレベル**

エラーレベル： {WARNING｜ERROR}

下線部は省略形です。

（2）説明

エラーレベルの指定とアセンブル結果で OS へのリターン値を変更します。

OS へのリターン値を**表** 12.2 に示します。

表 12.2 ABORT **オプション指定時の** OS **へのリターン値**

発生個数			**オプション指定時の** OS **へのリターン値**			
			ABORT＝WARNING		ABORT＝ERROR	
ウォーニング	**エラー**	**致命的エラー**	MS-DOS	UNIX	MS-DOS	UNIX
0	0	0	0	0	0	0
1 **以上**	0	0	2	1	0	0
―	1 **以上**	0	2	1	2	1
―	―	1 **以上**	4	1	4	1

下線部は、指定を省略した場合の設定です。
OS へのリターン値が 1 以上の場合は、オブジェクトモジュールの出力を抑止します。
本コマンドラインオプションは、オブジェクトモジュールの出力時に有効です。

（3）例

```
> asm38 aaa.mar -abort=warning
```

ウォーニングが 1 個以上発生した場合は、MS-DOS 版では、OS へのリターン値が 2 のため、オブジェクトモジュールの出力を抑止します。

UNIX 版では、OS へのリターン値が 1 のため、オブジェクトモジュールの出力を抑止します。

```
> asm38 aaa.mar -abort=error
```

エラーおよびフェイタルエラーがない場合は、MS-DOS 版では、OS へのリターン値が 0 のため、オブジェクトモジュールを出力します。

UNIX 版では、OS へのリターン値が 0 のため、オブジェクトモジュールを出力します。

12.3.8 EXPAND（プリプロセッサ展開結果の出力）

（1）書式

-EXPAND［＝出力ファイル名］

下線部は省略形です。

（2）説明

-EXPAND は、マクロ展開、条件付きアセンブル、構造化アセンブル、ファイルのインクルードを行なった後のアセンブラソースファイルを出力するコマンドラインオプションです。

本オプションを指定すると、オブジェクトの生成は行ないません。

出力ファイルの指定を省略すると、次のようになります。

・ファイル型（拡張子）の指定を省略した場合

ファイル型は exp になります。

・主ファイル名、ファイル型（拡張子）とも指定を省略した場合

主ファイル名は入力ソースファイル（1つめに指定したもの）と同じ、ファイル型は exp になります。

入力ファイルと出力ファイルに同じ名前を指定しないでください。

（3）例

```
> asm38 aaa.mar -EXPAND
```

プリプロセッサ展開結果をファイルに出力します。

ファイル名は aaa. exp です。

12.3.9 ABS8/ABS16（8 または 16 ビット絶対アドレス形式シンボルの指定）

（1）書式

-ABS8［＝シンボル［,シンボル･･･］］
-ABS16［＝シンボル［,シンボル･･･］］

（2）説明

8 ビットまたは 16 ビット絶対アドレス形式でアクセスするシンボルを指定します。
シンボル省略時は、全ての外部参照／定義シンボルを対象とします。

ABS8 コマンドラインオプションと ABS16コマンドラインオプションを同時に指定した場合、指定方法により扱いが異なります。

・-ABS8 -ABS16 と指定した場合
すべての外部シンボルは 16 ビット
・-ABS8=sym -ABS16 と指定した場合
sym のみ 8 ビット、その他は 16 ビット
・-ABS8=sym -ABS16=sym と指定した場合
sym は 16 ビット、その他は CPU により決定

表 12.3 に、アクセスサイズの優先順位を示します。

表 12.3 **アクセスサイズの優先順位**

優先順位		アクセスサイズの指定
1	高	絶対アドレス形式の確保サイズ（:XX）
2	↕	.IMPORT/.EXPORT/.GLOBALのアクセスサイズ
3	低	ABS8/ABS16コマンドラインオプション

（3）例

```
> asm38 aaa.mar -ABS8=sym1 -ABS16
```

絶対アドレス形式において、外部シンボルを指定する場合、
sym1 は 8 ビット、他の外部シンボルは 16 ビット絶対アドレス形式でアクセスします。

```
> asm38 aaa.mar -CPU=2600A -ABS8=sym1 -ABS16=sym2,sym3,sym4
aaa.mar の内容
.IMPORT sym1,sym2,sym3,sym5
.IMPORT sym4:8
MOV.B  @sym1,R1H          ;8 ビット（-ABS8 指定）
```

```
MOV.B  @sym2,R1H          ;16 ビット（-ABS16 指定）
MOV.B  @sym3:8,R1H        ;8 ビット（確保サイズ指定）
MOV.B  @sym4,R1H          ;8 ビット
                          ;（.IMPORT のアクセスサイズ指定）
MOV.B  @sym5,R1H          ;32 ビット（指定なし）
MOV.B  @(sym1+sym2),R1H ;8 ビット*1
                          ;（-ABS8 と-ABS16 混在指定）
```

【注】 *1 **絶対アドレス形式に外部シンボルを複数記述した場合、アクセスサイズは、最小のアクセスサイズを適用します。**

12.4 アセンブルリストに関するコマンドラインオプション

12.4.1 [NO]LIST（アセンブルリストの出力）

(1) 書式

LIST［=ファイル名］
NOLIST［=ファイル名］

下線部は省略形です。

(2) 説明

アセンブルリストの出力、出力抑止を指定します。

LIST ‥‥‥ 出力
NOLIST ‥‥ 出力抑止

ファイル名には、アセンブルリストを出力するファイル名を指定します。
ファイル名を省略した場合は、次のようにします。

LIST ‥‥‥ ソースファイルのファイル形式を.lisにしたファイル名を設定します。
NOLIST ‥‥ エラーが発生した行を画面に表示します。

また、指定したファイル名でファイル形式を省略した場合は、ファイル形式に.lis を設定します。

NOLIST でファイル名を指定した場合は、エラーが発生した行だけのアセンブルリストをファイルに出力します。

【注】 ファイル名には、入力ファイル名と重複する名称は指定しないでください。

(3) 制御命令との関係

- LISTコマンドラインオプション ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥出力
- NOLISTコマンドラインオプション‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥出力抑止
- 指定省略
  - .PRINT LIST ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥出力
  - .PRINT NOLIST ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥出力抑止
  - 指定省略 ‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥出力抑止

(4) 例

```
> asm38 aaa.mar -list
```

アセンブルリストを出力します。ファイル名は、aaa.lis です。

```
> asm38 aaa.mar -nolist
```

アセンブルリストを出力しません。

12.4.2 [NO]SOURCE（ソースプログラムリストの出力）

（1）書式

SOURCE
NOSOURCE

下線部は省略形です。

（2）説明

アセンブルリストのソースプログラムリストの出力、出力抑止を指定します。

SOURCE ・・・・・・・・・・出力
NOSOURCE ・・・・・・・・出力抑止

本コマンドラインオプションは、アセンブルリストの出力時に有効です。

（3）制御命令との関係

- SOURCEコマンドラインオプション ・・・・・・・・・・出力
- NOSOURCEコマンドラインオプション ・・・・・・・・・・出力抑止
- 指定省略
  - .PRINT SRC ・・・・・・・・・・出力
  - .PRINT NOSRC ・・・・・・・・・・出力抑止
  - 指定省略 ・・・・・・・・・・出力

（4）例

```
> asm38 aaa.mar -list -source
```

アセンブルリストにソースプログラムリストを出力します。

```
> asm38 aaa.mar -list -nosource
```

アセンブルリストにソースプログラムリストを出力しません。

12.4.3 [NO]CROSS_REFERENCE（クロスリファレンスリストの出力）

（1）書式

CROSS_REFERENCE
NOCROSS_REFERENCE

下線部は省略形です。

（2）説明

アセンブルリストのクロスリファレンスリストの出力、出力抑止を指定します。

CROSS_REFERENCE ・・・・・・・・ 出力
NOCROSS_REFERENCE・・・・・・ 出力抑止

本コマンドラインオプションは、アセンブルリストの出力時に有効です。

（3）制御命令との関係

- CROSS_REFERENCEコマンドラインオプション ・・・・・・・・・・・・・・・・・・・・・・・・・出力
- NOCROSS_REFERENCEコマンドラインオプション ・・・・・・・・・・・・・・・・・・・・・・出力抑止
- 指定省略
  - .PRINT CREF ・・・出力
  - .PRINT NOCREF ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・出力抑止
  - 指定省略 ・・出力

（4）例

```
> asm38 aaa.mar -list -cross_reference
```

アセンブルリストにクロスリファレンスリストを出力します。

```
> asm38 aaa.mar -list -cross_reference
```

アセンブルリストにクロスリファレンスリストを出力しません。

12.4.4 [NO]SECTION（セクション情報リストの出力）

（1）書式

SECTION
NOSECTION

下線部は省略形です。

（2）説明

アセンブルリストのセクション情報リストの出力、出力抑止を指定します。

SECTION・・・・・・・・・・・・・・・・・・ 出力
NOSECTION・・・・・・・・・・・・・・・・ 出力抑止

本コマンドラインオプションは、アセンブルリストの出力時に有効です。

（3）制御命令との関係

- SECTIONコマンドラインオプション ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・出力
- NOSECTIONコマンドラインオプション・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・出力抑止
- 指定省略
  - .PRINT SCT ・・・出力
  - .PRINT NOSCT・・・出力抑止
  - 指定省略 ・・出力

（4）例

```
> asm38 aaa.mar -list -section
```

アセンブルリストにセクション情報リストを出力します。

```
> asm38 aaa.mar -list -nosection
```

アセンブルリストにセクション情報リストを出力しません。

12.4.5 [NO]SHOW（ソースプログラムリストの部分出力）

（1）書式

SHOW[=**出力種別[,出力種別 …]**]
NOSHOW[=**出力種別[,出力種別 …]**]

出力種別：{CONDITIONALS｜DEFINITIONS｜CALLS｜EXPANSIONS｜STRUCTURED｜CODE}

下線部は省略形です。

【注】 MS-DOS 版では、複数の出力種別を指定する場合、カッコで囲むことができます。

（2）説明

ソースプログラムリストのプリプロセッサ機能のソースステートメントの部分出力、出力抑止、オブジェクトコード表示行の部分出力、出力抑止を指定します。

SHOW‥‥‥‥‥‥‥‥‥‥ 出力
NOSHOW‥‥‥‥‥‥‥‥‥ 出力抑止

出力種別は、次のとおりです。

CONDITIONALS‥‥‥‥‥‥ 条件付き不成立
DEFINITIONS ‥‥‥‥‥‥‥ 定義
CALLS ‥‥‥‥‥‥‥‥‥‥ コール
EXPANSIONS ‥‥‥‥‥‥‥ 展開
STRUCTURED ‥‥‥‥‥‥‥ 構造化展開
CODE ‥‥‥‥‥‥‥‥‥‥ オブジェクトコード表示行

出力種別の内容は、次のとおりです。

条件付き不成立 ‥‥‥‥‥‥ .AIF の不成立部分
定義 ‥‥‥‥‥‥‥‥‥‥ マクロ定義部分、
.AREPEAT、.AWHILE 定義部分、
.INCLUDE 制御文、
.ASSIGNA、.ASSIGNC 制御文
コール ‥‥‥‥‥‥‥‥‥ マクロコール文、
構造化アセンブリ制御文、
.AIF、.AENDI 制御文
展開 ‥‥‥‥‥‥‥‥‥‥ マクロ展開部分、
.AREPEAT、.AWHILE 展開部分

構造化展開 ・・・・・・・・・・・・・・ 構造化アセンブリ展開部分
オブジェクトコード表示行 ・・・ オブジェクトコード表示がソースステートメントの行数を超える部分

出力種別で指定した項目を出力、出力抑止します。出力種別を省略した場合は、すべての項目を出力、出力抑止します。

本コマンドラインオプションは、ソースプログラムリストの出力時に有効です。

（3）制御命令との関係

- SHOWコマンドラインオプション ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・出力
- NOSHOWコマンドラインオプション ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・出力抑止
- 指定省略
  - .LIST出力種別（出力） ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・出力
  - .LIST出力種別（出力抑止） ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・出力抑止
  - 指定省略 ・・・出力

（4）例

```
> asm38 aaa.mar -list -show
```

アセンブルリストにすべての項目を出力します。

```
> asm38 aaa.mar -list -noshow=expansions,structured
```

アセンブルリストに展開と構造化展開を出力しません。

```
> asm38 aaa.mar -list -noshow=code
```

アセンブルリストにソースステートメントの行数を越える部分のオブジェクトコード表示行を出力しません。

12.4.6 LINES（リスト行数の設定）

（1）書式

LINES=**行数**

下線部は省略形です。

（2）説明

アセンブルリストの 1 ページの行数を設定します。

行数に指定できる値は、20～255 です。

本コマンドラインオプションは、アセンブルリストの出力時に有効です。

（3）制御命令との関係

- LINESコマンドラインオプション ……LINESの行数
- 指定省略
  - .FORM制御命令 …….FORMの行数
  - 指定省略 ……60行

（4）例

```
> asm38 aaa.mar -list -lines=40
```

アセンブルリストの 1 ページの行数を 40 行にします。

12.4.7 COLUMNS（リスト桁数の設定）

（1）書式

COLUMNS=**桁数**

下線部は省略形です。

（2）説明

アセンブルリストの 1 行の桁数を設定します。

桁数に指定できる値は、79～255 です。

本コマンドラインオプションは、アセンブルリストの出力時に有効です。

（3）制御命令との関係

- COLUMNSコマンドラインオプション ……COLUMNSの桁数
- 指定省略
  - .FORM制御命令 ……FORMの桁数
  - 指定省略 ……132桁

（4）例

```
> asm38 aaa.mar -list -columns=80
```

アセンブルリストの桁数を 80 桁にします。

12.5 ファイルインクルード機能に関するコマンドラインオプション

12.5.1 INCLUDE（インクルードファイルのディレクトリ指定）

（1）書式

INCLUDE=**ディレクトリ名［,ディレクトリ名 …］**

下線部は省略形です。

（2）説明

.INCLUDE 制御命令でインクルードするファイルのディレクトリ名を指定します。

ディレクトリ名はホストマシンの標準的な指定方法に従います。

ディレクトリ名の指定数は、コマンドラインで 1 行入力可能な限り有効です。

インクルードファイルのサーチは、ディレクトリ名に指定した順に行ないます。インクルードファイルが見つかった場合、それ以降に指定したディレクトリ名は無効となります。

（3）制御命令との関係

INCLUDEコマンドラインオプション ‥‥[1] .INCLUDE制御命令のディレクトリ
[2] INCLUDEコマンドライオプション指定のディレクトリ*

指定省略‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥‥.INCLUDE制御命令のディレクトリ

【注】 * .INCLUDE **制御命令で指定したディレクトリの前に** INCLUDE **コマンドラインオプションで指定したディレクトリを付加します。**

（4）例

```
> asm38 aaa.mar -include=/usr/tmp,/tmp （UNIX の場合）
> asm38 aaa.mar -include=¥usr¥tmp,¥tmp （MS-DOS の場合）
```

（aaa.mar 内で.INCLUDE "file.h"指定の場合）

file.h をカレントディレクトリ、/usr/tmp、/tmp の順にサーチします。

12.6 条件付きアセンブリ機能に関するコマンドラインオプション

12.6.1 ASSIGNA（整数型プリプロセッサ変数の定義）

（1）書式

ASSIGNA=**プリプロセッサ変数名=整数定数［,プリプロセッサ変数名＝整数定数 …］**

下線部は省略形です。

（2）説明

プリプロセッサ変数に、整数定数を設定します。

プリプロセッサ変数名の書き方は、シンボル名の書き方と同じです。ただし、文字数は最大 32 文字です。

ホスト OS が UNIX の場合は、プリプロセッサ変数名の中にドル記号（$）がある場合、ドル記号（$）の直前に円記号（¥）を指定します。

整数定数は、基数（B'、Q'、D'、H'）と数値を組み合わせて指定します。基数を省略し、数値のみを指定した場合は 10 進数として扱います。

ホスト OS が UNIX の場合は、基数表示のアポストロフィ（'）の直前に円記号（¥）を指定します。

整数定数に指定できる値の範囲は、-2,147,483,648～4,294,967,295 です。ただし、負の値を設定する場合は、10 進以外の基数で指定してください。

（3）制御命令との関係

- ASSIGNAコマンドラインオプション
  - .ASSIGNA制御命令* ……ASSIGNAコマンドラインオプション指定の値
  - 指定省略 ……ASSIGNAコマンドラインオプション指定の値
- 指定省略 ― .ASSIGNA制御命令 ……….ASSIGNA制御命令指定の値

【注】 * ASSIGNA **コマンドラインオプションでプリプロセッサ変数に値を設定した場合、当該プリプロセッサ変数への**.ASSIGNA **制御命令による定義がすべて無効になります。**

（4）例

```
> asm38 aaa.mar -assigna=x=0
```

プリプロセッサ変数 x に値 0 を設定します。ソースプログラム内のプリプロセッサ変数 x のすべての参照箇所¥&x を 0 に設定します。

```
> asm38 aaa.mar -assigna=_¥$=H¥'FF      （UNIX の場合）
> asm38 aaa.mar -assigna=_$=H'FF        （MS-DOS の場合）
```

プリプロセッサ変数_$に値 H'FF を設定します。ソースプログラム内のプリプロセッサ変数_$のすべての参照箇所¥&_$を H'FF に設定します。

12.6.2 ASSIGNC（文字型プリプロセッサ変数の定義）

（1）書式

ASSIGNC=**プリプロセッサ変数名＝"文字列"［,プリプロセッサ変数名="文字列"　…］**

下線部は省略形です。

（2）説明

プリプロセッサ変数に、文字列を設定します。

プリプロセッサ変数名の書き方は、シンボル名の書き方と同じです。ただし、文字数は最大 32 文字です。

ホスト OS が UNIX の場合は、プリプロセッサ変数名の中にドル記号（$）がある場合、ドル記号（$）の直前に円記号（¥）を指定します。

文字列は文字をダブルコーテーション（"）で囲んで指定します。

ホスト OS が UNIX の場合は、文字列の中に次の文字を指定する際、直前に円記号（¥）を指定します。また、前後に文字列を指定する場合は、前後の文字列をダブルコーテーション（"）で囲みます。

- イクスクラメーション（!）
- ダブルコーテーション（"）
- ドル（$）
- 逆コーテーション（'）

文字列には、255 文字（バイト）まで指定できます。

（3）制御命令との関係

- ASSIGNCコマンドラインオプション
  - .ASSIGNC制御命令* ……ASSIGNCコマンドラインオプション指定の文字列
  - 指定省略 ……ASSIGNCコマンドラインオプション指定の文字列
- 指定省略 ― .ASSIGNC制御命令 ……….ASSIGNC制御命令指定の文字列

【注】 * ASSIGNC **コマンドラインオプションでプリプロセッサ変数に文字列を設定した場合、当該プリプロセッサ変数への**.ASSIGNC **制御命令による定義がすべて無効になります。**

（4）例

```
> asm38 aaa.mar -assignc=x="OK"
```

プリプロセッサ変数 x に文字列 OK を設定します。ソースプログラム内のプリプロセッサ変数 x のすべての参照箇所¥&x を文字列 OK に設定します。

```
> asm38 aaa.mar -assignc=_¥$="ON"¥!"OFF"      （UNIX の場合）
> asm38 aaa.mar -assignc=_$="ON!OFF"   （MS-DOS の場合）
```

プリプロセッサ変数_$に文字列 ON!OFF を設定します。ソースプログラム内のプリプロセッサ変数_$のすべての参照箇所¥&_$を文字列 ON!OFF に設定します。

12.6.3 DEFINE オプション（文字列の置き換えの定義）

（1）書式

DEFINE=**置換シンボル**="**文字列**"[,**置換シンボル**="**文字列**"...]

下線部は省略形です。

（2）説明

DEFINE オプションは、プリプロセッサで置換シンボルを対応する文字列に置き換えることを定義します。

DEFINE オプションと ASSIGNC オプションの機能の違いは、.DEFINE と.ASSIGNC の機能の違いに対応します。

（3）例

```
asm38 file.src -DEFINE=Z="ABC"
```

（4）注意事項

文字列には、255 文字（バイト）まで指定できます。ただし、ホスト OS が MS-DOS の場合、コマンドラインの制限により指定できる文字列は短くなります。

12.7 文字列、コメント内の日本語記述に関するコマンドラインオプション

12.7.1 SJIS（文字列、コメント内の日本語記述）

（1）書式

SJIS

（2）説明

文字列、コメント内の日本語記述を可能とします。

SJIS・・・・・・・・・・・・文字列、コメント内の日本語は、シフト JIS コードとして解釈
指定省略 ・・・・・・・・文字列内の日本語は、ウォーニングとして無視
コメント内の日本語は、ホストマシンに依存する日本語コードとして解釈

【注】 本コマンドラインオプションは、EUC、LATIN1 **コマンドラインオプションと一緒に指定しないでください。**

（3）例

```
> asm38 aaa.mar -sjis
```

文字列内の日本語をシフト JIS コードとして解釈します。

12.7.2 EUC（文字列、コメント内の日本語記述）

（1）書式

EUC

（2）説明

文字列、コメント内の日本語記述を可能とします。

EUC ・・・・・・・・・・・・文字列、コメント内の日本語は、EUC コードとして解釈
指定省略 ・・・・・・・・文字列内の日本語は、ウォーニングとして無視
コメント内の日本語は、ホストマシンに依存する日本語コードとして解釈

【注】 本コマンドラインオプションは、SJIS、LATIN1 コマンドラインオプションと一緒に指定しないでください。

（3）例

```
> asm38 aaa.mar -euc
```

文字列内の日本語を EUC コードとして解釈します。

12.7.3 OUTCODE（オブジェクトコード内の漢字コード指定）

（1）書式

OUTCODE=**漢字コード**
漢字コード： {SJIS|EUC}

下線部は省略形です。

（2）説明

OUTCODE は、ソースファイル内の日本語記述を指定した漢字コードに変換し、オブジェクトファイルに出力します。

OUTCODE とソースファイル内の漢字コード（-SJIS、-EUC）指定との組合せを**表** 12.4 に示します。

表 12.4 OUTCODE **とソースファイル内の漢字コードの組合せ**

OUTCODE **の**	**ソースファイル内の漢字コード**		
漢字コード	-SJIS	-EUC	**指定なし**
SJIS	**シフト** JIS **コード**	**シフト** JIS **コード**	**シフト** JIS **コード**
EUC	EUC **コード**	EUC **コード**	EUC **コード**
指定なし	**シフト** JIS **コード**	EUC **コード**	**デフォルト漢字コード**

デフォルト漢字コードを**表** 12.5 に示します。

表 12.5 **デフォルト漢字コード**

SPARC **ステーション**	EUC **コード**
HP9000/700 **シリーズ**	**シフト** JIS **コード**
PC9800 **シリーズ** IBM PC **およびその互換機**	**シフト** JIS **コード**

（3）例

```
asm38 file.src -SJIS -OUTCODE=EUC
```

シフト JIS コードのソースファイルを EUC コードに変換し、オブジェクトファイルを出力します。

12.7.4 LATIN1（文字列、コメント内の欧州コード記述）

（1）書式

-LATIN1

（2）説明

文字列、コメント内の欧州コード記述を可能にします。

LATIN1 ・・・・・・・・・・・・文字列、コメント内の文字は、LATIN1 コードとして解釈

指定省略 ・・・・・・・・・・・文字列、コメント内の欧州コードは、ホストマシンに依存する日本語コードとして解釈

【注】 本コマンドラインオプションは、SJIS、EUC および OUTCODE コマンドラインオプションと一緒に指定しないでください。

（3）例

```
> asm38 aaa.mar -LATIN1
```

文字列、コメント内の文字を、LATIN1 コードとして解釈します。

12.8 コマンドラインの指定方法に関するコマンドラインオプション

12.8.1 SUBCOMMAND（サブコマンドファイルの指定）

（1）書式

SUBCOMMAND=**サブコマンドファイル名**

下線部は省略形です。

（2）説明

サブコマンドファイルには、コマンドラインに展開する入力ファイル名とコマンドラインオプションを指定します。

サブコマンドファイルの書式は次のとおりです。

（1）通常のコマンドライン指定と同じ順番で、入力ファイル名とコマンドラインオプションを指定してください。

（2）1 行に 1 つの入力ファイル名またはコマンドラインオプションを指定してください。

【注】 本コマンドラインオプションは、コマンドラインの末尾に指定してください。
本コマンドラインオプションは、サブコマンドファイル内に指定しないでください。

（3）例

```
> asm38 aaa.mar -subcommand=aaa.sub
```

サブコマンドファイルの内容をコマンドラインに展開し、アセンブルします。

aaa.sub の内容

```
bbb.mar
-list
-noobj
```

展開例は、次のとおりです。

```
> asm38 aaa.mar,bbb.mar -list -noobj
```

13. モジュール間最適化のオプション・環境変数

第 13 章　目次

13.1 コマンドラインの形式

モジュール間最適化ツールを起動するコマンドラインの形式は次のとおりです。

```
optlnk38［△<オプション>... ］
    <オプション>:<オプション>［= <パラメタ>［, <パラメタ>. . . ］］
```

13.2 サブコマンドファイルの書式

サブコマンドファイルの書式は次のとおりです。

```
<サブコマンド>△［<パラメタ>［,<パラメタ>. . . ］］［△;<コメント>］
```

【注意】

1. **パラメタがファイル名の場合は、パラメタの区切りに空白を指定できます。**
2. **サブコマンドを１行に記述できない場合は、&を用いて継続指定できます。**
3. **コメントの開始を示すセミコロン(;)と、サブコマンドまたはパラメタの間は必ず１個以上の空白が必要です。**

ユニット名として、ソースファイル名称を指定します。アセンブリソースファイルの場合は、アセンブル時に任意のユニット名を指定することができます。

13.3 最適化機能オプション／サブコマンド

オプション／サブコマンドと短縮形および省略時解釈の一覧を**表** 13.1 に示します。英大文字は短縮形指定時の文字を示します。下線は省略時解釈を示します。

また日立統合化開発環境の対応するダイアログメニューをタブ名［項目］で示します。ダイアログメニューの詳細は、日立統合化開発環境のユーザーズマニュアルを参照してください。

表 13.1 **最適化機能オプション／サブコマンド一覧**

No	項目	オプション／サブコマンド名	パラメタ	ダイアログメニュー	指定内容
1	**最適化内容の指定**	OPtimize		Optimize[Optimize]	
			STring_unify	[Unify Strings]	**定数／文字列の統合**
			| SYmbol_delete	[Eliminate Dead Code]	**未参照シンボルの削除**
			| Variable_access	[Use Short Addressing]	**短絶対アドレッシングモードの活用**
			| Register	[Reallocate Registers]	**レジスタの再割付**
			| SAMe_code	[Eliminate Same Code]	**共通コードの統合**
			| Function_call	[Use Indirect Call/Jump]	**間接アドレッシングモードの活用**
			| Branch	[Optimize Branches]	**分岐命令の最適化**
			| SPeed	[Speed]	**スピード重視の最適化**(op=st,sy,v,r,b)
			| SAFe	[Safe]	**安全な最適化**(op=st,r,b)
		NOOPtimize	―	Optimize[Optimize]	**最適化の抑止指定**
2		SAMESize	<size> (**省略時：**1E) size:16 **進数**	Optimize [Eliminated Size]	**共通コード統合の対象となるサイズの指定**
3	**最適化抑止**	SYmbol_forbid	<**シンボル名**> [,<**シンボル名**>...]	Optimize[Forbid Item] [Forbid Elimination of Dead Code to]	**未参照シンボル削除の最適化を抑止する変数／関数名を指定**
4		SAMECode_forbid	<**関数名**> [,<**関数名**>...]	Optimize[Forbid Item] [Forbid Elimination of Same Code to]	**共通コード統合の最適化を抑止する関数名を指定**
5		Variable_forbid	<**変数名**> [,<**変数名**>...]	Optimize[Forbid Item] [Forbid Use of Short Addressing to]	**短絶対アドレッシングモード活用の最適化を抑止する変数名を指定**

No	項目	オプション／サブコマンド名	パラメタ	ダイアログメニュー	指定内容
6	**最適化抑止**	Function_forbid	<**関数名**> [,<**関数名**>...]	Optimize[Forbid Item] [Forbid Use of Indirect Call/Jump to]	**間接アドレッシングモード活用の最適化を抑止する関数名を指定**
7		ABSolute_forbid	<addr>[+<size>] [,<addr>[+<size>]...] addr,size:16 **進数**	Optimize[Forbid Item] [Forbid Memory Allocation in]	**アドレス割付の対象外となるアドレス領域を指定**
8	**最適化情報**	Mlist	<**ファイル名**>	List [Optimization File]	**最適化情報リストの出力**
9		SHow	Symbol | Reference	List [Optimization File] [Symbol] [Reference]	**リスト出力内容指定** **シンボル最適化情報出力** **シンボル参照回数出力**
10		INFormation	―	Optimize [Output Information]	**最適化された関数名を画面表示**
11	**サブコマンドファイル**	SUbcommand	<**ファイル名**>	― [Use External Subcommand File]	**サブコマンドファイルの指定**

（1）最適化の指定

■オプション／サブコマンド形式

オプション　：`OPtimize[=<パラメタ>[,<パラメタ>...]]`
　　`NOOPtimize`
サブコマンド　：`OPtimize[△<パラメタ>[,<パラメタ>...]]`
　　`NOOPtimize`
パラメタ　：`STring_unify | SYmbol_delete | Variable_access | Register |SAMe_code | Function_call | Branch | SPeed | SAFe`

■ダイアログメニュー

`Optimize[Optimize]`

■説明

optimize 指定時、モジュール間最適化を実行します。また、パラメタを指定することにより、最適化内容を指定することができます。各パラメタの内容を**表** 13.2 に示します。

nooptimize 指定時、モジュール間最適化を実行せずに、リンケージ処理のみ行います。

【注意】

optimize **オプション／サブコマンドのパラメタは、指定されたパラメタの論理和が有効となります。例えば、**optimize=speed,same_code **が指定された場合、**optimize=string_unify,symbol_delete,variable_access, register,branch,same_code

が有効になります。すなわち、optimize=function_call 以外の全ての最適化を実施します。

表 13.2 optimize オプション／サブコマンドのパラメター一覧

パラメタ	説明	最適化対象
パラメタなし	全ての最適化を実行します。optimize=string_unify, symbol_delete, variable_access, register, same_code, function_call, branch を指定した時と同じです。	—
string_unify	const 属性を持つ定数／文字列に対し、同一値定数および同一文字列の統合を、モジュール間にわたって実施します。 const 属性を持つ定数／文字列には、次のものが含まれます。 ・C/C++プログラム中で const 宣言した変数 ・文字列データの初期値	コンパイラ出力オブジェクトのみ対象
symbol_delete	一度も参照のない変数／関数を削除します。この最適化を指定する場合は、必ず entry サブコマンドを指定してください。	コンパイラ出力オブジェクトのみ対象
variable_access	8 ビットおよび 16 ビット絶対アドレッシングモードでアクセス可能な領域に空きがあれば、アクセス回数の多い変数を割り当て、当該変数のアクセスコード最適化を行います。	変数：C/C++定義変数 対象アクセスコード： 全てのオブジェクトプログラム
register	関数の呼出関係を解析し、冗長なレジスタ退避・回復コードを削除します。また、呼出前後のレジスタ使用状況により、使用レジスタ番号を変更することもあります。	コンパイラ出力オブジェクトのみ対象
same_code	複数の同一命令列をサブルーチン化して、コードサイズを削減します。	コンパイラ出力オブジェクトのみ対象
function_call	0〜0xFF の範囲に空きがあれば、アクセス回数の多い関数のアドレスを割り当てる最適化を行います。	全てのオブジェクトプログラムを対象
branch	プログラムの配置情報に基づいて、分岐命令サイズを最適化します。また、他の最適化項目をひとつでも実行すると、本最適化は指定の有無に関わらず、必ず実行します。	全てのオブジェクトプログラムを対象
speed	最適化項目のうち、同一命令列のサブルーチン化のようなオブジェクトスピード低下を招く可能性のある最適化以外を実施します。optimize=string_unify, symbol_delete, variable_access, register,branch を指定したときと同じ効果になります。	—
safe	メモリ割り付け位置が固定でなければならない変数や、スピードを優先したい関数など、部分的に最適化を抑止したい場合があります。optimize=safe は、変数や関数の属性によって制限される可能性のある最適化以外を実施します。optimize=safe は、optimize=string_unify, register, branch を指定したときと同じ効果になります。	—

(2) 共通コード最適化サイズの指定

■オプション／サブコマンド形式

オプション　　：SAMESize=<パラメタ>

サブコマンド　：SAMESize△<パラメタ>

パラメタ　　：<数値>

■ダイアログメニュー

Optimize[Eliminated Size]

■説明

optimize=same_code で、最適化の対象となる共通コードのサイズを指定します。このオプションで指定するサイズは、オブジェクトプログラムの実際のバイト数を指します。optimize=same_code オプションが有効でない場合には、本オプション／サブコマンドは無視されます。

<数値>：

16 進数で指定します。指定したサイズ以上の命令列について、共通コードのサブルーチン化を実施します。本オプション／サブコマンド省略時は、samesize=1E を仮定します。指定できる範囲は 8≦ 数値 ≦ 7FFF です。

(3) 未参照シンボル削除を抑止するシンボルの指定

■オプション／サブコマンド形式

オプション　　：SYmbol_forbid=<パラメタ>[,<パラメタ>...]

サブコマンド　：SYmbol_forbid△<パラメタ>[,<パラメタ>...]

パラメタ　　：<シンボル名>

■ダイアログメニュー

Optimize[Forbid Item->Forbid Elimination of Dead Code to]

■説明

未参照シンボル削除（optimize=symbol_delete）の最適化を抑止する変数名／関数名を指定します。optimize=symbol_delete オプションが有効でない場合には、本オプション／サブコマンドは無視されます。

<シンボル名>：

変数名、C 関数名は C/C++プログラム中での定義名の先頭に_を付加します。

C++関数の場合は、引数列を含めたプログラム中の定義名と同じ名前をダブルクォーテーションで囲んで指定します。（例：“ f(int) ”）

最適化によって削除してはならない変数名、関数名を指定してください。

(4) 共通コード統合を抑止する関数の指定

■オプション／サブコマンド形式

オプション　　：SAMECode_forbid=<パラメタ>[,<パラメタ>...]

サブコマンド　：SAMECode_forbid△<パラメタ>[,<パラメタ>...]

パラメタ　　：<関数名>

■ダイアログメニュー

Optimize[Forbid Item->Forbid Elimination of Same Code to]

■説明

共通コード統合（optimize=same_code）の最適化を抑止する関数名を指定します。optimize=same_code オプションが有効でない場合は、本オプション／サブコマンドは無視されます。

<関数名>：

C 関数の場合は、プログラム中での定義名の先頭に_を付加します。C++関数の場合は、引数列を含めたプログラム中の定義名と同じ名前をダブルクォーテーションで囲んで指定します。（例：“ f(int) ”）

共通コード統合の最適化は、コードサイズ削減には効果がありますが、実行速度が低下する可能性があります。スピードを重視する関数など、共通コード統合化を抑止したい関数名を指定してください。

（5）短絶対アドレス領域割り付けを抑止する変数の指定

■オプション／サブコマンド形式

オプション　　：Variable_forbid=<パラメタ>[,<パラメタ>...]

サブコマンド　：Variable_forbid△<パラメタ>[,<パラメタ>...]

パラメタ　　：<変数名>

■ダイアログメニュー

Optimize[Forbid Item->Forbid Use of Short Addressing to]

■説明

短絶対アドレッシングモード活用（optimize=variable_access）の最適化を抑止する変数名を指定します。 optimize=variable_access オプションが有効でない場合は、本オプション／サブコマンドは無視されます。

<シンボル名>：

変数名は C/C++プログラム中での定義名の先頭に_を付加します。

C/C++プログラム内で定義し、アセンブリプログラム内で間接参照しているなど、アドレスを変更してはならない変数名を指定してください。

（6）間接アドレス呼び出しを抑止する関数の指定

■オプション／サブコマンド形式

オプション　　：Function_forbid=<パラメタ>[,<パラメタ>...]

サブコマンド　：Function_forbid△<パラメタ>[,<パラメタ>...]

パラメタ　　：<関数名>

■ダイアログメニュー

Optimize[Forbid Item->Forbid Use of Indirect Call/Jump to]

■説明

間接アドレッシングモード活用（optimize=function_call）の最適化を抑止する関数名を指定します。optimize=function_call オプションが有効でない場合には、本オプション／サブコマンドは無視されます。

<関数名>：

C 関数の場合は、プログラム中での定義名の先頭に_を付加します。C++関数の場合は、引数列を含めたプログラム中の定義名と同じ名前をダブルクォーテーションで囲んで指定します。（例：“ f(int) ”）

間接アドレッシングモードの活用は、コードサイズ削減には効果がありますが、実行速度が低下する可能性があります。スピードを重視する関数など、間接アドレッシングモードの使用を抑止したい関数名を指定してください。

（7）最適化による再割り付け使用を禁止する領域の指定

■オプション／サブコマンド形式

オプション　　：ABSolute_forbid=<パラメタ>[,<パラメタ>...]

サブコマンド　：ABSolute_forbid△<パラメタ>[,<パラメタ>...]

パラメタ　　：<アドレス>[+<size>]

■ダイアログメニュー

Optimize[Forbid Item->Forbid Memory Allocation in]

■説明

最適化時に使用できないアドレス領域を指定します。

<アドレス>[+<サイズ>]：

アドレス、サイズとも 16 進数で指定します。また、アドレス値が A～Fの値で始まる場合は、先頭に 0 を付加してください。optimize=function_call オプションや optimize=variable_access オプションでは、オブジェクト性能を向上するために、積極的にメモリ割り付けの変更を行います。I/O 領域やエミュレータ予約領域など、最適化で使用できないアドレス領域を指定してください。

（8）最適化情報リストの出力指定

■オプション／サブコマンド形式

オプション　　：Mlist[=<ファイル名>]

サブコマンド　：Mlist△<ファイル名>

■ダイアログメニュー

List[Optimization File]

■説明

シンボルの参照回数および最適化情報をファイルに出力します。

ファイル名が省略された場合は、出力ファイル.lop になります。ファイル名の拡張子が

省略された場合は、lop になります。

nooptimize オプション／サブコマンド指定時は、シンボルの参照回数のみ出力します。

（9）最適化情報リスト内容の指定

■オプション／サブコマンド形式

オプション　　：SHow =<パラメタ>[,<パラメタ>]

サブコマンド　：SHow△<パラメタ>[,<パラメタ>]

パラメタ　　：Symbol | Reference

■ダイアログメニュー

List[Optimization File]

■説明

最適化情報リストの出力内容を指定します。

symbol パラメタ指定時には、シンボルの最適化情報を出力します。

reference パラメタ指定時には、シンボルの参照回数情報を出力します。

mlist オプション／サブコマンドが指定されていない場合は、本オプション／サブコマンドは無効です。また、nooptimize オプション／サブコマンド指定時は、symbol パラメタは無効です。

（10）最適化対象関数名の表示

■オプション／サブコマンド形式

オプション　　：INFormation

サブコマンド　：INFormation

パラメタ　　：なし

■ダイアログメニュー

Optimize[Output Information]

■説明

最適化された関数名の表示を指定します。

（11）サブコマンドファイルの指定

■オプション／サブコマンド形式

オプション　　：SUbcommand=<パラメタ>

パラメタ　　：<ファイル名>

■ダイアログメニュー

― [Output Information]

■説明

サブコマンドファイルを指定します。リンケージ機能は、サブコマンドのみサポートしているため、本オプションを省略することはできません。

<ファイル名>：

リンケージエディタ用サブコマンドファイル名称を指定します。
サブコマンドファイル名には、-を含めることはできません。

【注意】

1. **リンケージエディタ用サブコマンドファイルは、最適化ツール用サブコマンドファイルとして指定することができます。但し、最適化ツール用サブコマンドを指定したサブコマンドファイルを、H シリーズリンケージエディタのサブコマンドファイルとして指定した場合、エラーになります。**
2. print **サブコマンドを指定した場合、マップリストには最適化ツール出力のテンポラリサブコマンドファイル名が出力されます。このテンポラリサブコマンドファイルは、最適化ツール終了後削除されます。**

13.4 リンケージ機能サブコマンド

Hシリーズリンケージエディタでサポートしている表13.3のサブコマンドを指定できます。サブコマンドの大文字は短縮形指定時の文字を示します。また、下線は省略時解釈を示します。

また、日立統合化開発環境の対応するダイアログメニューを、タブ名[項目]で示します。ダイアログメニューの詳細は、日立統合化開発環境のユーザーズマニュアルを参照してください。

表 13.3 リンケージ機能サブコマンド一覧

項番	分類	サブコマンド	ダイアログメニュー	機能
1	ファイル制御	Input	Input [Input Files]	入力ファイルの指定
2		LIBrary NOLIBrary	Input [Input Files]	ライブラリファイルの指定
3		ENTry	Input [Use Entry Point]	実行開始アドレスの指定
4		DEFine	Input [Defines]	外部参照シンボルの強制定義
5		Form = A | R	Output [Load Module Type]	アブソリュート形式で出力 リロケータブル形式で出力
6		DEBug NODEBug	Output [Debug Information]	デバッグ情報の出力
		SDebug		デバッグ情報をファイル出力
7		ELf	Output [Object Format]	ELF フォーマット出力
		SYSROF		sysrof フォーマット出力
		SYSROFPlus		sysrof フォーマット出力で、 dwarf デバッグ情報出力可能
8		Output NOOutput	Output [Load Module Path]	出力ファイルの指定
9		ROm	Output [ROM to RAM Mapped Sections]	ROM 化支援機能の指定
10		EXCLude NOEXCLude	Output [Exclude Unreferenced External Symbol]	未参照ライブラリの結合抑止
11		FSymbol	Output [Symbol Address File]	シンボルアドレスのファイル出力指定

項番	分類	サブコマンド	ダイアログメニュー	機能
12	ファイル制御	Print NOPrint	List [Generate Map File]	リンケージリストファイルの出力指定
13		DIrectory	—	ディレクトリ名置き換え指定
14		STart	Section	セクションのアドレス指定
15		ALign_section	Section [Align section]	境界調整が異なるセクションの結合指定
16		CHeck_section	Verify [Check for Unlinked Section]	アドレス未指定セクションのチェック
17		Cpu	Verify [Use CPU Information File]	CPU 情報ファイルによるメモリ割り付けチェック
18		CPUCheck	Verify [Stop Linkage on CPU Information warning]	CPU 情報ファイルによるメモリ割り付けチェック時のエラー出力
19	エラーチェック	Udf NOUdf	—	未定義シンボルの表示指定
20		UDFCheck	Verify->Check for Undefined Symbol	未定義シンボル存在時のエラー出力
21	その他	EXCHange	—	ユニットの強制置換
22		REname	—	シンボル名の変更
23		DELete	—	シンボルの削除
24		EXIt	—	リンケージ処理終了指定
25		ECho NOECho	—	サブコマンドのエコーバック

（1）入力ファイルの指定

■サブコマンド形式

```
Input△<ファイル名>[,<ファイル名>...]
```

■ダイアログメニュー

```
Input[Input Files]
```

■説明

入力ファイル名を指定します。複数ファイル名を指定する場合は、空白で区切ることもできます。

入力ファイル名に拡張子の指定がない場合は、.obj を仮定します。

入力ファイルとして、ライブラリファイル内のモジュールを"<ライブラリ名>(<モジュール名>)"の形式で指定することもできます。ライブラリ名に拡張子の指定がない場合は、.lib を仮定します。

■使用例

(a) オブジェクトモジュールファイルを入力します。

```
INPUT△main            ; main.obj を入力
```

(b) 複数のファイルを入力する場合はカンマまたは空白で区切ります。

```
INPUT△main△tan.obj  ; main.obj、tan.obj を入力
```

(c) ライブラリファイル内のモジュールを入力します。

```
INPUT△funclib(sin,cos),tan.o  ; funclib.lib 内のモジュール sin、cos
を入力
```

(2) ライブラリファイルの指定

■サブコマンド形式

```
LIBrary△<ファイル名>[,<ファイル名>...]
NOLIBrary
```

■ダイアログメニュー

```
Input[Input Files]
```

■説明

ライブラリファイルの入力有無を指定します。

library サブコマンドはライブラリの入力およびライブラリ名を指定します。入力ファイルとして指定されたファイル間でのリンケージ処理後に、未解決の外部参照シンボルをライブラリからサーチします。

複数ファイル名を指定する場合は、空白で区切ることもできます。

ライブラリ名に拡張子の指定がない場合は、.lib を仮定します。

nolibrary サブコマンド指定時は、ライブラリを入力しません。

■使用例

(a) ライブラリファイルを指定します。

```
LIBRARY△syslib        ; syslib.lib を入力
```

(b) 複数のファイルを入力する場合はカンマまたは空白で区切ります。

```
LIBRARY△system△debug; system.lib と debug.lib を入力
```

(3) 実行開始アドレスの指定

■サブコマンド形式

ENTry△<外部定義シンボル名>

■ダイアログメニュー

Input[Use Entry Point]

■説明

実行開始アドレスを指定します。指定できるシンボル名は、外部定義シンボルだけです。

#pragma entry 指定と ENTRY を同時に指定した場合、ENTRY 指定を有効とします。

entry サブコマンドが指定されていない場合、optimize=symbol_delete, register が無効になります。

■使用例

```
ENTRY△_INIT
```

INIT 関数をエントリとして指定シンボル名はアセンブリプログラムで参照するシンボル名を指定します。 C プログラムの場合は先頭に下線(_)を付加します。 C++プログラムの場合は引数列を含めたプログラム中の定義名と同じ名前をダブルクォーテーションで囲んで指定します。（例：“ init(void) ”）

(4) 外部参照シンボルの強制定義

■サブコマンド形式

DEFine△<外部定義シンボル名>(<パラメタ>)

<パラメタ>:　<数値> | <アドレス> | <外部定義シンボル名>

■ダイアログメニュー

Input[Defines]

■説明

外部参照シンボルを、指定された値、アドレスまたは外部定義シンボルの値で強制的に定義します。

数値、アドレスは先頭が数字の 16 進数で指定します。先頭が A～F の場合は 0 を付加してください。また、外部定義シンボルは、既に定義済みのシンボルを指定してください。

リロケータブルロードモジュール（form△r）指定時には、本サブコマンドは無効になります。

■使用例

(a) 未定義シンボルを強制定義します。

```
DEFINE△PORT10(0FFE8)      ; 未定義シンボル"PORT10"を"FFE8"で定義
```

(b) 未定義シンボルを外部定義シンボルと同値として定義します。

DEFINE△MAIN_RTN(PRG_EXIT)； MAIN_RTN を PRG_EXIT と同じ値を持つシンボルとして定義

(5) 出力ファイルの指定

■サブコマンド形式

Output△[<ファイル名>]

NOOutput

■ダイアログメニュー

Output[Load Module Path]

■説明

ロードモジュール出力有無を指定します。

output サブコマンドはロードモジュールの出力および出力ファイル名を指定します。ファイル名の拡張子が省略された場合は、form=a のときには.abs、form=r のときには.rel になります。

またファイル名が省略された場合は、<先頭に入力されたファイル名>に.abs または.rel の拡張子を付加したファイル名になります。

nooutput サブコマンド指定時はロードモジュールを出力しません。

■使用例

```
output△sample.abs
```

出力ファイル名を sample.abs に指定します。

(6) ROM 化支援機能の指定

■サブコマンド形式

ROm△(<セクション名 1>,<セクション名 2>)[,(<セクション名 1>,<セクション名 2>)...]

<セクション名 1>： ROM 割り付け用セクション名（コンパイラ出力セクション名）

<セクション名 2>： RAM 割り付け用セクション名

■ダイアログメニュー

Output[Rom to Ram Mapped Sections]

■説明

初期化データ領域を ROM、RAM 上に二重に確保します。

コンパイラ出力の初期化データ領域セクションを ROM 上に割り付けるセクション（セクション名１）として指定します。セクション名２にサイズ０以外の存在するセクションを指定した場合はエラーになります。

■使用例

```
ROM△(D,R)
START△P,C,D(200),R,B(8000)
```

D セクションと同サイズの R セクションを確保し、D セクション内シンボル参照個所の

R セクション上のアドレスでリロケーションします。

start サブコマンドを用いて、D セクションを ROM 上、R セクションを RAM 上アドレスに割り付けてください。

（7）オブジェクトフォーマットの指定

■サブコマンド形式

```
ELf
SYSROF
SYSROFPlus
```

■ダイアログメニュー

```
Output[Object Format]
```

■説明

オブジェクトフォーマットを指定します。本サブコマンドの指定を省略した場合、SYSROF（従来フォーマット）のオブジェクトファイルを出力します。

デバッグ情報出力サブコマンド（DEBUG/SDEBUG）との組み合せで、5 種類のオブジェクトフォーマットを選択できます。使用するデバッガに合わせて選択してください。

表 13.4　使用可能なデバッガとオプション／サブコマンドの関係

使用可能なデバッガ	オプション／サブコマンド	
	オブジェクトフォーマット	デバッグ情報出力
3rd party 製 ELF/DWARF サポートのデバッガ	ELF	DEBUG
日立統合化マネージャ（Ver.4）+ E7000	SYSROFPLUS	SDEBUG
日立統合化マネージャ（Ver.3）+ E7000	SYSROF	SDEBUG
日立デバッキングインタフェース（Ver.2）+ E6000	SYSROF	DEBUG
日立デバッキングインタフェース（Ver.3）+ E6000	ELF	SDEBUG

【注意】

リンク時に ELF または SYSROFPLUS オプションを指定した場合、旧バージョンのコンパイラ、リンケージエディタ、ライブラリアンで生成したオブジェクトプログラム、ライブラリのデバッグ情報は削除されます。

■使用例

（a）ELF フォーマットのロードモジュールファイル sample.abs とデバッグ情報（DWARF フォーマット）ファイル sample.dwf を出力します。

```
ELF
SDEBUG
output sample.abs
  :
```

（b）ELF フォーマットのロードモジュールファイル sample.abs を出力します。デバッグ情報（DWARF フォーマット）はロードモジュール内に出力します。

```
ELF
DEBUG
output sample.abs
   :
```

（c）SYSROF フォーマットのロードモジュールファイル sample.abs とデバッグ情報（SYSROF フォーマット）ファイル sample.dbg を出力します。

```
SYSROF
SDEBUG
output sample.abs
   :
```

（d）SYSROF フォーマットのロードモジュールファイル sample.abs を出力します。デバッグ情報（SYSROF フォーマット）はロードモジュール内に出力します。

```
SYSROF
DEBUG
output sample.abs
   :
```

（e）SYSROFPLUS サブコマンドと DEBUG サブコマンドは同時に指定できません。指定した場合はエラーになります。

```
SYSROFPLUS
DEBUG                    ; エラー3010 を出力
   :
```

（8）ロードモジュール形式の指定

■サブコマンド形式

```
Form△ A | R
```

■ダイアログメニュー

```
Output[Load Module Type]
```

■説明

出力ロードモジュール形式を指定します。

form△a 指定時は、アブソリュート形式ロードモジュールを出力します。

form△r 指定時は、リロケータブル形式ロードモジュールを出力します。

rom または start サブコマンド指定時は、form△r は無効です。

本サブコマンドの省略時解釈は、form△a です。

（9）デバッグ情報出力の指定

■サブコマンド形式

DEBug

SDebug

NODEBug

■ダイアログメニュー

Output[Debug Information]

■説明

デバッグ情報の出力有無を指定します。

debug 指定時は、ロードモジュール中にデバッグ情報を出力します。

sdebug 指定時は、ロードモジュールと別ファイルとしてデバッグ情報ファイルを出力します。

nodebug 指定時は、デバッグ情報を出力しません。

debug または sdebug 指定時に、全ての入力ファイルにデバッグ情報が含まれていない場合はウォーニングメッセージを表示します。

debug、sdebug 指定は、オブジェクトフォーマット指定との組み合わせでデバッグ情報フォーマットが以下のように規定されます。

オブジェクトフォーマット指定	デバッグ情報指定	デバッグ情報フォーマット
elf	debug	dwarf
	sdebug	dwarf
sysrof	debug	sysrof（従来フォーマット）
	sdebug	sysrof（従来フォーマット）
sysrofplus	debug	**指定不可**
	sdebug	dwarf

【注意】

1. C++**プログラムのソースレベルデバッグを行う場合、デバッグ情報フォーマットが** dwarf **になるように、オブジェクトフォーマット、デバッグ情報指定をしてください。**
2. **使用例は「（**7**）オブジェクトフォーマットの指定」を参照してください。**

（10）未参照ライブラリの結合抑止

■サブコマンド形式

EXCLude

NOEXCLude

■ダイアログメニュー

Output[Exclude Unreferenced External Symbol]

■説明

アセンブリプログラム内で外部参照（.import）宣言した未参照シンボルの結合有無を指定します。

exclude サブコマンド指定時は、未参照シンボルの結合を抑止します。

noexclude サブコマンド指定時は、未参照シンボルを結合します。

form△r 指定時、noexclude 指定は無効になります。

本サブコマンドの省略時解釈は、noexclude です。

■使用例

```
exclude
  :
```

アセンブリプログラム内で.import 宣言した未参照シンボルの結合抑止を指定します。

【注意】 exclude **サブコマンドは** input **サブコマンドより前に指定してください。**

（11）シンボルアドレスのファイル出力指定

■サブコマンド形式

FSymbol <セクション名>[,<セクション名>...]

■ダイアログメニュー

Output[Exclude Unreferenced External Symbol]

■説明

リンケージ機能で解決した外部定義シンボルをアセンブラ制御命令形式でファイルに出力します。

出力するファイル名は、ロードモジュール名に拡張子".fsy"を付加したファイルとなります。

指定したセクション名が存在しない場合、ウォーニングメッセージを表示し、処理を継続します。指定したセクションが全て存在しない場合、ファイルは出力されません。

form△r 指定時、本サブコマンドは無効です。

■使用例

```
fsymbol△sct2,sct3
output△sample.abs
```

セクション sct2 と sct3 の外部定義シンボルを sample.fsy ファイルに出力します。

【ファイル sample.fsy の出力例】

```
;H SERIES LINKAGE EDITOR GENERATED FILE   1998.10.10
;fsymbol = sct2, sct3
;SECTION NAME = sct2
       .export sym1
sym1:  .equ     h'00FF0080      ;sct2内外部定義シンボル、アドレスを.equ形
                                ;式で出力
          :
```

sample.fsy をアセンブルし、リンク時に入力ファイルとして指定すると、sct2、sct3 セクションのオブジェクトファイルをリンクすることなく、外部参照シンボルのアドレスを解決できます。

本機能は、複数のロードモジュールで共通 ROM を参照する場合にご使用ください。

（12）リンケージリストファイルの出力指定

■サブコマンド形式

```
Print  <ファイル名>
NOPrint
```

■ダイアログメニュー

```
Output[Generate Map File]
```

■説明

リンケージリストファイル出力の有無を指定します。

print サブコマンドはリンケージリストファイル出力およびリストファイル名を指定します。

ファイル名の拡張子が省略された場合は、.map を付加したファイル名になります。ファイル名は省略できません。

noprint サブコマンド指定時は、リンケージリストファイルを出力しません。

（13）ディレクトリ名置き換え指定

■サブコマンド形式

```
DIrectory  <シンボル名>（<ディレクトリ名>）
```

■ダイアログメニュー

なし

■説明

ディレクトリ名をシンボル名に置き換えて指定することができます。

登録したディレクトリ名の参照は、シンボル名を$と/（ＰＣ版では$と¥）で囲んで指定します。

ディレクトリ名は 16 個まで指定できます。

■使用例

```
DIRECTORY △ symbol(dir1/dir2)
```

```
  INPUT △ $symbol/file1.obj
```

ディレクトリ名 dir1/dir2 をシンボル名 symbol として登録します。

$symbol/を dir1/dir2/に置き換え、ファイル名を dir1/dir2/file1.obj とします。

（14）セクションのアドレス指定

■サブコマンド形式

```
STart  <セクション名>[[[,<セクション名>...]
          [:<セクション名>[,<セクション名>...]...](<先頭アドレス>)]...]
```

■ダイアログメニュー

```
Section[Section Start Address]
```

■説明

セクションの開始アドレスを指定します。開始アドレスの指定がないセクションは、最終割り付けアドレスに続いて割り付けます。

またセクション群をコロン（：）で区切って、同一アドレスへの複数セクションの割り付けを指定できます。

アドレスは、先頭が数字で始まる 16 進数で指定します。

■使用例

```
        :
  ROM   (D1,R1),(D2,R2)
  START  P,C,D1,D2(8000)
  START  R1,B:R2(0D0000)
```

P、C、D1、D2 セクションを指定した順に結合し、8000 番地から割り付けます。

R1、B セクションを結合し、D0000 から割り付けます。また、R2 セクションも D0000 から割り付けます。

（15）境界調整が異なるセクションの結合指定

■サブコマンド形式

```
ALign_section
```

■ダイアログメニュー

```
Section[Align Section]
```

■説明

同一名で境界調整数が異なるセクションでも、同一のアドレスに割り付けます。本サブコマンド省略時には、同一名で境界調整数が異なるセクションは別セクションとして扱い、後から入力されたセクションを、開始アドレス未指定セクションとして扱います。

（16）アドレス未指定セクションのチェック

■サブコマンド形式

```
CHeck_section
```

■ダイアログメニュー

```
Verify[Check for Unlinked Section]
```

■説明

start オプションで指定しなかったセクションが存在するとき、ウォーニングメッセージを出力します。

（17）CPU 情報ファイルによるメモリ割り付けチェック

■サブコマンド形式

```
Cpu  <ファイル名>
```

■ダイアログメニュー

```
Verify[Use CPU Information File]
```

■説明

CPU 情報ファイルに基づき、割り付けメモリ範囲が使用可能であるかどうかをチェックします。

本サブコマンドが指定されない場合、optimize=variable_access, function_call が無効になります。

CPU 情報ファイルは、CPU 情報作成ツール（cia38）を用いて作成します。

（18）CPU 情報ファイルによるメモリ割り付けチェック時のエラー出力

■サブコマンド形式

```
CPUCheck
```

■ダイアログメニュー

```
Verify[Stop Linkage on CPU Information Warning]
```

■説明

cpu サブコマンド指定時、CPU 情報ファイルで指定したメモリレイアウトと矛盾したメモリ割り付けを行った場合、エラーを出力してリンケージ処理を終了します。

cpu サブコマンドを指定していない場合は、本サブコマンドは無効です。

（19）未定義シンボルの表示指定

■サブコマンド形式

```
Udf
NOUdf
```

■ダイアログメニュー

なし（常に udf が有効になります）

■説明

未定義シンボルが存在する場合、メッセージを出力します。リロケータブル形式（form=r）指定時、本サブコマンドは無効です。

（20）未定義シンボル存在時のエラー出力

■サブコマンド形式

```
UDFCheck
```

■ダイアログメニュー

```
Verify[Check for Undefined Symbol]
```

■説明

udf サブコマンド指定時、未定義シンボルが存在する場合、エラーを出力してリンケージ処理を終了します。udf サブコマンドが指定されていない場合、本サブコマンドは無効です。

（21）ユニットの強制置換

■サブコマンド形式

```
EXCHange  <ファイル名>[(<ユニット名>[,<ユニット名>...])]
```

■ダイアログメニュー

なし

■説明

指定ファイル中の指定ユニットを、リンケージ処理中のロードモジュール内同名ユニットと入れ替えます。ファイル名として指定できるのは、コンパイラ、アセンブラ出力のオブジェクトファイルかリロケータブルロードモジュールです。

ファイル名に拡張子の指定がない場合は、.obj を仮定します。

（22）シンボル名の変更

■サブコマンド形式

```
REname       <パラメタ>[,<パラメタ>...]
<パラメタ>:  un=<ユニット名>(<ユニット名>) |
             er=<ユニット名>.<外部参照シンボル名>(<外部参照シンボル名>) |
             ed=<ユニット名>.<外部定義シンボル名>(<外部定義シンボル名>)
```

■ダイアログメニュー

なし

■説明

ユニット名、外部定義シンボル名、外部参照シンボル名を括弧内で指定された名前に変更します。

本サブコマンド以降に指定した最初の input サブコマンドの入力ファイルのみが対象になります。

また rename サブコマンドの直後に指定できるのは、以下の５つのサブコマンドのみです。

・input

・exchange

・rename

・delete

・abort

(23) シンボルの削除

■サブコマンド形式

```
DELete      <パラメタ>[,<パラメタ>...]
<パラメタ>:  un=<ユニット名> | ed=<ユニット名>.<外部定義シンボル名>
```

■ダイアログメニュー

なし

■説明

ユニット名、外部定義シンボル名、外部参照シンボル名を削除します。

本サブコマンド以降に指定した最初の input サブコマンドの入力ファイルのみが対象になります。

また delete サブコマンドの直後に指定できるのは､以下の５つのサブコマンドのみです。

・input

・exchange

・rename

・delete

・abort

(24) リンケージ処理終了指定

■サブコマンド形式

```
EXIt
```

■ダイアログメニュー

なし

■説明

リンケージ処理を終了します。サブコマンドファイルの最後に必ず指定してください。

(25) サブコマンドのエコーバック

■サブコマンド形式

```
ECho
NOECho
```

■ダイアログメニュー

なし

■説明

echo 指定時、サブコマンドをコンソール表示します。

noecho 指定時、サブコマンドをコンソール表示しません。
本サブコマンドの省略時解釈は、noecho です。

13.5 モジュール間最適化の環境変数

モジュール間最適化ツールで使用する環境変数の使用方法を**表** 13.5 に示します。

表 13.5 **モジュール間最適化の環境変数**

No.	環境変数	説明
1	path	**モジュール間最適化ツール本体の格納ディレクトリを指定します。** **指定フォーマット：** PC **版** C> path= <**モジュール間最適化ツール本体パス名**>;%path% unix **版 C シェル** %set path =(<**モジュール間最適化ツール本体パス名**> $path) **ボーン** %PATH=:<**モジュール間最適化ツール本体パス名**>[:<**既存パス名**>...] **シェル** %export PATH
2	HLNK_LIBRARY1 HLNK_LIBRARY2 HLNK_LIBRARY3	**デフォルトライブラリ名を指定します。** **ライブラリサブコマンドで指定したライブラリを優先してリンクします。その後未解決のシンボルがある場合、デフォルトライブラリを** 1,2,3 **の順に入力します。** **指定フォーマット：** PC **版** C> set HLNK_LIBRARY1= <**ライブラリ名** 1> C> set HLNK_LIBRARY2= <**ライブラリ名** 2> C> set HLNK_LIBRARY3= <**ライブラリ名** 3> unix **版 C シェル** %setenv HLNK_LIBRARY1 <**ライブラリ名** 1> %setenv HLNK_LIBRARY2 <**ライブラリ名** 2> %setenv HLNK_LIBRARY3 <**ライブラリ名** 3> **ボーン** %HLNK_LIBRARY1=<**ライブラリ名** 1> **シェル** %export HLNK_LIBRARY1 %HLNK_LIBRARY2=<**ライブラリ名** 2> %export HLNK_LIBRARY2 %HLNK_LIBRARY3=<**ライブラリ名** 3> %export HLNK_LIBRARY3
3	HLNK_TMP	**テンポラリファイルを作成するディレクトリを指定します。この環境変数の指定がない場合は、カレントディレクトリにテンポラリファイルを作成します。** **指定フォーマット：** PC **版** C> set HLNK_TMP= <**テンポラリファイルパス名**> unix **版 C シェル** %setenv HLNK_TMP <**テンポラリファイルパス名**> **ボーンシェル** % HLNK_TMP = <**テンポラリファイルパス名**> %export HLNK_TMP

14. モジュール間最適化のエラーメッセージ

第 14 章　目次

14.1 モジュール間最適化のエラーメッセージ

本章では、以下の形式でモジュール間最適化ツールの出力するエラーメッセージとエラー内容を説明します。

エラー番号	エラーメッセージ
	エラー内容

エラーレベルは、エラーの重要度に従い、4種に分類されます。

表 14.1 モジュール間最適化ツールのエラーレベル

0～999	インフォメーション	処理を継続し、ロードモジュールを出力します。
1000～1999	ウォーニング	処理を継続し、ロードモジュールを出力します。
2000～2999	エラー	処理を継続します。ロードモジュールは出力しません。
3000～3999	フェータル	処理を中断します。

表 14.2 モジュール間最適化ツールのエラーメッセージ

0010	<ユニット名 1> IS REPLACED WITH <ユニット名 2> (<ファイル名>)
	<ユニット名 1>を(ファイル名)中の<ユニット名 2>に置き換えました。
0020	<外部名 1> IS RENAMED TO <外部名 2>
	<外部名 1>を<外部名 2>に変更しました。
0030	<外部名> IS DELETED
	<外部名>を削除しました。
0040	DUPLICATE UNIT - (<ユニット名>) IN (<ファイル名>) IS DELETED
	<ユニット名>のユニットを複数見つけたため、<ファイル名>中のユニット名を削除しました。
0050	<外部参照シンボル名> CANNOT BE DEFINED
	<外部参照シンボル名>が見つからないため、強制定義できません。
0060	<外部名> CANNOT BE RENAMED
	<外部名>が見つからないため、変更できません。
0070	<外部名> CANNOT BE DELETED
	<外部名>が見つからないため、削除できません。
0080	<ユニット名> CANNOT BE REPLACED
	<ユニット名>が見つからないため、置き換えができません。
0200	<最適化種別> OPTIMIZE:<セクション名> SECTION IS CREATED
	<最適化種別>の最適化によって<セクション名>を作成しました。

0210	<最適化種別> OPTIMIZE:<ユニット名>.<シンボル名> MOVED <セクション名> SECTION
	<最適化種別>の最適化によって<ユニット名>.<シンボル名>を<セクション名>に移動しました。
0220	<最適化種別> OPTIMIZE:<ユニット名>.<シンボル名> IS CREATED
	<最適化種別>の最適化によって<ユニット名>.<シンボル名>を作成しました。
0230	<最適化種別> OPTIMIZE:<ユニット名>.<シンボル名> IS DELETED
	<最適化種別>の最適化によって<ユニット名>.<シンボル名>を削除しました。
0240	<ユニット名>.<シンボル名> IS OPTIMIZED
	<ユニット名>.<シンボル名>を最適化しました。
1010	DUPLICATE OPTION/SUBCOMMAND(<オプション／サブコマンド名>)
	同じオプションまたはサブコマンドを重複して指定しています。後に指定したオプションまたはサブコマンドが有効になります。
1020	IDENTIFIER CHARACTER EXCEEDS 251(<名前>)
	251 文字を超える名前（ユニット名、セクション名、シンボル名）を指定しています。251 文字までが有効になります。
1040	DUPLICATE SYMBOL(<シンボル名>)
	外部定義シンボルが重複しています。先に現われた外部定義シンボルが有効になります。
1050	UNDEFINED EXTERNAL SYMBOL(<ユニット名>.<シンボル名>)
	未定義の外部シンボルを参照しています。外部参照は無効になり、0 を仮定します。
1060	REDEFINED SYMBOL(<シンボル名>)
	定義済みのシンボルを DEFINE オプション／サブコマンドで定義しています。DEFINE オプション／サブコマンドの指定を無視します。
1070	SECTION ATTRIBUTE MISMATCH(<セクション名>)
	属性または境界調整数の異なる同名セクションを入力しました。別セクションとして扱います。
1080	RELOCATION SIZE OVERFLOW(<ユニット名>.<セクション名> - <オフセット値>)
	リロケーションの結果がリロケーションサイズを超えました。
1090	ENTRY POINT MULTIPLY DEFINED
	実行開始アドレスの指定があるオブジェクトモジュールを複数指定しています。先に現われた実効開始アドレスの指定が有効になります。
1110	DUPLICATE SECTION NAME(<セクション名>)
	プション／サブコマンドで同一セクション名を指定しています。最初に指定したセクション名を有効にします。

1120	ILLEGAL CPU INFORMATION FILE FORMAT
	CPU 情報ファイルのファイル形式が正しくありません。
1130	CONFLICTING DEVICE TYPE
	入力オブジェクトモジュールの対象 CPU と異なる CPU 情報ファイルを指定しています。
1140	SECTION IS NOT IN SAME MEMORY AREA(<セクション名>:xxxx-yyyy)
	セクションが一つのメモリ領域に入りきらず、xxxx 番地から yyyy 番地が異なるメモリ領域に割り付けられています。
1150	INACCESSIBLE ADDRESS RANGE(<セクション名>)
	セクションが使用できない領域に割り付けられています。
1160	INVALID CPU OPTION/SUBCOMMAND
	ロードモジュールファイルをリロケータブル形式に指定して、CPU オプション／サブコマンドを指定しています。
1170	ADDRESS SPACE DUPLICATE
	セクションが重複しています。
1180	INVALID UDF OPTION/SUBCOMMAND
	出力ロードモジュール形式がアブソリュート指定に対し、NOUDF オプション／サブコマンドを指定しています。NOUDF オプション／サブコマンドを無視します。
1190	RELOCATION VALUE IS ODD(<ユニット名>.<セクション名> - <オフセット値>)
	ディスプレースメントに対するリロケーション結果が奇数になりました。最下位ビットを切り捨てます。
1200	START ADDRESS NOT SPECIFIED FOR SECTION(<セクション名>)
	START オプション／サブコマンドで指定していないセクションが存在します。
1210	CANNOT FIND SECTION(<セクション名>)
	指定したセクションが見つかりません。
1220	TOO LONG SUBCOMMAND LINE
	ディレクトリ名の置き換えで文字数が 511 文字を超えました。511 文字までを有効とします。
1230	TOO MANY DIRECTORY COMMANDS
	DIRECTORY サブコマンドで 16 個を超えたディレクトリを指定しています。16 個までを有効とします。
1240	NO DEBUG INFORMATION
	デバッグ情報の全くないファイルに対して DEBUG、SDEBUG サブコマンドを指定しています。コンパイル、アセンブル時にデバッグオプションを指定してください。

1250	CANNOT SET ENTRY POINT
	出力ロードモジュールがリロケータブル形式のとき、実行開始アドレスに定数の外部参照シンボルを指定しています。出力ロードモジュールをアブソリュート形式にするか、実行開始アドレスの指定を削除してください。
1260	TOO LONG CHARACTER NUMBER(FSYMBOL)
	FSYMBOL サブコマンドで指定したセクション内のシンボルの文字数が 238 文字を超えています。
1270	EXTERNAL SYMBOL 0（<セクション名>）
	FSYMBOL サブコマンドで指定したセクション内に外部定義シンボルが存在しません。
1280	ILLEGAL SYMBOL REFERENCE （<シンボル名>）
	START サブコマンドで同一アドレスに割り付けたセクション間でシンボルを参照しています。
1600	INVALID SYMBOL_FORBID OPTION
	SYMBOL_FORBID の指定が無効です。
1610	INVALID SAMECODE_FORBID OPTION
	SAMECODE_FORBID の指定が無効です。
1620	INVALID VARIABLE_FORBID OPTION
	VARIABLE_FORBID の指定が無効です。
1630	INVALID FUNCTION_FORBID OPTION
	FUNCTION_FORBID の指定が無効です。
1640	INVALID ABSOLUTE_FORBID OPTION
	ABSOLUTE_FORBID の指定が無効です。
1700	CANNOT FIND SYMBOL SPECIFIED SYMBOL_FORBID(<シンボル名>)
	SYMBOL_FORBID で指定したシンボル名が見つかりません。
1710	CANNOT FIND SYMBOL SPECIFIED SAMECODE_FORBID(<シンボル名>)
	SAMECODE_FORBID で指定したシンボル名が見つかりません。
1720	CANNOT FIND SYMBOL SPECIFIED VARIABLE_FORBID(<シンボル名>)
	VARIABLE_FORBID で指定したシンボル名が見つかりません。
1730	CANNOT FIND SYMBOL SPECIFIED FUNCTION_FORBID(<シンボル名>)
	FUNCTION_FORBID で指定したシンボル名が見つかりません。
1800	<最適化種別> OPTIMIZE:SECTION OVERLAP
	<最適化種別>の最適化でサイズ増加により隣接するセクションと重複しました。 <最適化種別>の最適化指定を無効にします。

1810	DIFFERENT SYMBOL ASSIGNED TO A GLOBAL REGISTER AMONG FILES (<**シンボル名**>:<**レジスタ番号**>)
	グローバルレジスタに割り付けるシンボル名、レジスタ番号がファイル間で異なります。
1820	STACK ACCESS SIZE OVERFLOW
	レジスタ最適化でスタックアクセスコードがコンパイラのスタック量制限値を超えました。レジスタ最適化指定を無視します。
1840	RELOCATION VALUE EXISTS(<**ユニット名**>.<**セクション名**> - <**オフセット値**>)
	定数-ラベルの式があります。<ユニット名>を最適化対象外にします。
1850	CANNOT OPTIMIZE REGISTER ALLOCATION(FUNCTION CALL NEST TOO DEEP)
	関数、呼び出しのネストが深すぎるため、レジスタ最適化指定を無視します。
1860	CANNOT OPTIMIZE REGISTER ALLOCATION(REGPARAM 2/3 MIXED)
	コンパイラオプション regparam=2 **を指定したオブジェクトと** regparam=3 **を指定したオブジェクトが混在しています。レジスタ最適化を無視します。**
2010	ILLEGAL SUBCOMMAND/OPTION
	不正なサブコマンド名（またはオプション名）を指定しています。
2020	SYNTAX ERROR
	指定されたサブコマンド（またはオプション）に構文上の不正があります。
2030	TOO LONG SUBCOMMAND LINE
	サブコマンドの長さが 511 **文字を超えています。**
2040	ILLEGAL SUBCOMMAND SEQUENCE
	サブコマンドの指定順序が不正です。
2070	ILLEGAL SECTION NAME(<**セクション名**>)
	不正なセクション名を指定しています。
2080	ILLEGAL SYMBOL NAME(<**シンボル名**>)
	不正なシンボル名を指定しています。
2100	TOO MANY INPUT FILES
	入力ファイル数が 65535 **個を超えています。**
2110	CANNOT FIND FILE(<**ファイル名**>)
	指定したファイルが見つかりません。
2120	CANNOT FIND UNIT(<**ユニット名**>)
	指定したユニットが見つかりません。
2130	CANNOT FIND MODULE(<**モジュール名**>)
	指定したモジュールが見つかりません。
2140	DUPLICATE START ADDRESS SPECIFIED
	同じ先頭アドレスを重複して指定しています。

2170	SUBCOMMAND COMMAND IN SUBCOMMAND FILE
	サブコマンドファイル中に SUBCOMMAND サブコマンドを指定しています。
2190	INVALID ADDRESS(<アドレス>)
	指定したアドレスが CPU のアドレス範囲を超えています。
2200	TOO MANY ROM COMMANDS
	ROM サブコマンドで 64 組を超えたセクションを指定しています。
2210	CANNOT CREATE ABSOLUTE MODULE(<モジュール名>)
	未定義の外部参照シンボルが存在しています。
2220	DIVISION BY ZERO IN RELOCATION VALUE(<ユニット名>.<セクション名> - <オフセット値>)
	0 除算を含むオブジェクトファイルを入力しました。
2600	COMPILER SUPPLEMENTARY INFORMATION FILE MISMATCH(<ファイル名>)
	コンパイラ付加情報ファイルの作成日付がオブジェクトと一致しません。
2610	ILLEGAL DUPLICATE SYMBOL(<シンボル名>)
	外部定義シンボルが重複しています。
2700	ILLEGAL ADDRESS SPECIFIED ABSOLUTE_FORBID OPTION(<アドレス>+<サイズ>)
	絶対アドレスの最適化抑止指定範囲が CPU のアドレス範囲を超えました。
2730	ILLEGAL SAMESIZE SPECIFIED
	共通コードサイズ指定が正しくありません。
2750	NOT SPECIFIED ENTRY SUBCOMMAND
	optimize=symbol_delete を指定していますが、entry サブコマンド指定がありません。
3010	ILLEGAL COMMAND PARAMETER
	不正なコマンドパラメータを指定しています。
3020	CANNOT OPEN FILE(<ファイル名>)
	ファイルをオープンできません。
3030	CANNOT READ INPUT FILE(<ファイル名>)
	ファイルを読み込むことができません。
3040	CANNOT WRITE OUTPUT FILE(<ファイル名>)
	ファイルに書き込むことができません。
3050	CANNOT CLOSE FILE(<ファイル名>)
	ファイルをクローズできません。
3060	ILLEGAL FILE FORMAT(<ファイル名>)
	指定したファイルのフォーマットが不正です。または、RENAME サブコマンドで指定した外部シンボル名が既に存在します。

3070	ILLEGAL RECORD FORMAT(<**ファイル名**>)
	指定したファイル中に不正なレコードがあります。または、除数が 0 **の除算があります。**
3080	SECTION ADDRESS OVERFLOW(<**セクション名**>)
	セクションの割り付けアドレスが CPU **で許されるアドレス範囲を超えています。**
3090	ADDRESS OVERFLOW
	指定したアドレスが CPU **で許されるアドレス範囲を超えています。**
3100	MEMORY OVERFLOW
	最適化ツールが内部で使用するメモリ領域を割り当てることができません。
3110	PROGRAM ERROR(<nnn>)
	最適化ツールの内部処理で何らかの障害が発生しました。プログラムエラー番号(nnn)**を確認の上、当社営業担当までご連絡ください。**
3120	ILLEGAL START ADDRESS ALIGNMENT(<**アドレス**>)
	オブジェクトモジュールの境界調整数と矛盾するアドレスを指定しています。
3140	CANNOT FIND SECTION(<**セクション名**>)
	指定したセクションが見つかりません。
3190	AUTOPAGE SPECIFIED AT NON-PAGE TYPE
	非ページタイプの入力ファイルに対して AUTOPAGE **オプション／サブコマンドを指定しています。**
3220	PAGE ADDRESS SPECIFIED AT NON-PAGE TYPE
	非ページタイプの入力ファイルに対してページアドレスを指定しています。
3230	SECTION SPECIFIED AT ROM OPTION/SUBCOMMAND DOES NOT EXIST (<**セクション名**>)
	ROM **コマンドで指定したセクションが存在しません。**
3250	ILLEGAL START SECTION(<**セクション名**>)
	START **コマンドで指定したセクションの属性が不正です。**
3260	CANNOT READ
	指定したファイル（標準入力を含む）から入力できません。
3270	SYMBOL ADDRESS OVERFLOW(<**シンボル名**>)
	シンボルの割り付けアドレスが CPU **のアドレス範囲を超えています。**
3280	ILLEGAL ROM SECTION(<**セクション名**>)
	ROM **オプション／サブコマンドの指定で、転送先セクションにサイズ** 0 **以外のセクションあるいは絶対番地セクションを指定しています。または、転送元と転送先のセクションの属性が異なっています。**

3290	INVALID MEMORY MAP
	CPU 情報ファイルと矛盾したメモリに割り付けています。または、異なるメモリ種別にまたがって割り付けています。
3300	ILLEGAL FILE FORMAT (INPUT ABSOLUTE FILE)
	アブソリュートロードモジュールを入力ファイルに指定しています。
3310	ILLEGAL FILE FORMAT (MISMATCH OBJECT FORMAT VERSION)
	オブジェクト形式の異なるファイルを入力しました。
3320	ILLEGAL FILE FORMAT (INPUT MISMATCH CPU TYPE)
	H シリーズ、SH シリーズ以外のファイルを入力しました。
3330	CANNOT OPEN INTERNAL FILE
	中間ファイルをオープンできません。ディスク容量に空きがないか、またはディスクにハード的なエラーがある場合があります。確認の上、再実行してください。
3340	CANNOT CLOSE INTERNAL FILE
	中間ファイルをクローズできません。ディスク容量に空きがないか、またはディスクにハード的なエラーがある場合があります。確認の上、再実行してください
3350	CANNOT DELETE INTERNAL FILE
	中間ファイルを削除できません。ディスク容量に空きがないか、またはディスクにハード的なエラーがある場合があります。確認の上、再実行してください
3360	CANNOT OUTPUT INTERNAL FILE
	中間ファイルに書き込みできません。ディスク容量に空きがないか、またはディスクにハード的なエラーがある場合があります。確認の上、再実行してください
3370	CANNOT READ INTERNAL FILE
	中間ファイルから入力できません。ディスク容量に空きがないか、またはディスクにハード的なエラーがある場合があります。確認の上、再実行してください
3390	TOO MANY UNITS
	指定したユニット数が 65,535 を超えました。
3400	TOO MANY SECTIONS
	指定したセクション数が 65,535 を超えました。
3700	CANNOT OPEN CPU INFORMATION FILE
	CPU 情報ファイルがオープンできません。
3710	CANNOT OPEN INTERNAL FILE
	中間ファイルがオープンできません。
3720	CANNOT WRITE INTERNAL FILE
	中間ファイルに書き込むことができません。
3730	CANNOT CLOSE INTERNAL FILE
	中間ファイルをクローズできません。

3740	CANNOT EXECUTE(<**ロードモジュール名**>)
	opt38 **または** lnk **を起動できません。正しくインストールされているか確認してください。**
3750	CANNOT CREATE INTERNAL FILE
	中間ファイルを作成することができません。
3760	INTERRUPT BY USER
	処理中に標準入力端末から「(CNTL)C**」コマンドによる割り込みを検出しました。**
3770	CANNOT ANALYZE OBJECT(<**ユニット名**>)
	オブジェクトコードを解析することができません。プログラムセクション内の.DATA **制御命令を削除するか、または当該ユニットのアセンブル時の** GOPTIMIZE **オプション指定を外してください。**
3800	TOO MANY EXTERNAL DEFINE SYMBOLS(<**ユニット名**>)
	ユニット内の定義シンボル数が 65535 **を超えました。ファイルを分割するか、または当該ユニットのコンパイル、アセンブル時の** GOPTIMIZE **オプション指定を外してください。**
3810	TOO MANY EXTERNAL REFERENCE SYMBOLS(<**ユニット名**>)
	ユニット内の参照シンボル数が 65535 **を超えました。ファイルを分割するか、または当該ユニットのコンパイル、アセンブル時の** GOPTIMIZE **オプション指定を外してください。**
3820	TOO MANY SECTIONS(<**ユニット名**>)
	ユニット内のセクション数が 65535 **を超えました。ファイルを分割するか、または当該ユニットのコンパイル、アセンブル時の** GOPTIMIZE **オプション指定を外してください。**
3830	<**最適化種別**> OPTIMIZE:SECTION OVERLAP
	<**最適化種別**>**の最適化でサイズ増加により隣接するセクションと重複しました。**

付録

付録　目次

付録 A. 制限事項

表 A.1 に、ソースプログラムの記述に関する制限事項を示します。

表 A.1 ソースプログラムの記述に関する制限事項

項番	項目	制限
1	文字	ASCII 文字*1
2	1 行文字数	8,192 文字まで
3	アセンブル行数	65,535 行まで（展開した行を含む）
4	文字定数	4 文字まで
5	シンボル文字数	251 文字まで*3
6	シンボル数	65,535 個まで
7	外部参照シンボル数	65,535 個まで
8	外部定義シンボル数	65,535 個まで
9	セクションの最大サイズ*2	H8S/2600 アドバンストモード … H'FFFFFFFF バイトまで H8S/2600 ノーマルモード…… H'00010000 バイトまで H8S/2000 アドバンストモード … H'FFFFFFFF バイトまで H8S/2000 ノーマルモード…… H'00010000 バイトまで H8/300H アドバンストモード…. H'01000000 バイトまで H8/300H ノーマルモード …… H'00010000 バイトまで H8/300…… H'0000FFFF バイトまで H8/300L …… H'0000FFFF バイトまで
10	セクション数	65,535 個まで
11	ファイルインクルード	ネストは 30 レベルまで
12	文字列長	255 文字まで
13	ファイル名長	251 文字まで

【注】 *1 SJIS、EUC、LATIN1 コマンドラインオプションの指定により、かな・漢字および、LATIN1 コードが使用できます。

*2 セクションの最大サイズは、アドレス空間の指定により異なります。

*3 プリプロセッサ変数名、DEFINE 置換シンボル名、マクロ名および、マクロ仮引数名は、32 文字までです。

付録 B.　アセンブルリスト

B.1　アセンブルリストの構成

アセンブルリストは、次のリストで構成されています。

(1) ソースプログラムリスト

ソースプログラムに関する情報を示します。

(2) クロスリファレンスリスト

ソースプログラムのシンボルに関する情報を示します。

(3) セクション情報リスト

ソースプログラムのセクションに関する情報を示します。

B.2 ソースプログラムリスト

ソースプログラムリストの内容は、次のとおりです。

(1) リスト行番号

(2) ロケーションカウンタ値

絶対アドレスセクションの場合は絶対アドレスを、相対アドレスセクションの場合は相対アドレスを表示します。

(3) オブジェクトコード

(4) ソース行番号

ソースファイル内でのソースステートメントの行番号です。アセンブラが展開したソースステートメントに対しては、行番号は表示しません。

(5) 展開区分

プリプロセッサ機能のソースステートメント区分です。

展開区分には、次のものがあります。

I・・・・・・・ ファイルインクルード

C・・・・・・ 条件付きアセンブルの成立、繰り返し展開、条件付き繰り返し展開

M ・・・・・ マクロ展開

S ・・・・・・ 構造化アセンブリ展開

展開区分 I には、インクルードのネストレベルを併せて表示します。

(6) ソースステートメント

```
  1                                      1              .CPU    2600A:32
  2                                      2     ;
  3 00000000                             3              .SECTION AAA,CODE,ALIGN=2
  4 00000000                             4     START
  5 00000000 7A0700000000                5              MOV.L   #STACK:32,SP
  6 00000006 F800                        6              MOV.B   #0:8,R0L
  7 00000008 6AA800000000                7              MOV.B   R0L,@ANS:32
  8 0000000E 7A0200001000                8              MOV.L   #DATA:32,ER2
  9                                      9              .FOR.B (R1L=#1,#8,+#1)
 10 00000014 F901                          S                             MOV    #1,R1L
 11 00000016 5800000A                      S                             BRA    _$F00002
 12          0000001A                      S                     _$F00000: .EQU  $
 13 0000001A 6828                       10                       MOV.B   @ER2,R0L
 14 0000001C 0B02                       11                       ADDS.L  #1,ER2
 15 0000001E 5E000000                   12                       JSR     @CHANGE:24
 16                                     13                    .ENDF
 17          00000022                      S                     _$F00001: .EQU  $
 18 00000022 8901                          S                             ADD    #1,R1L
 19          00000024                      S                     _$F00002: .EQU  $
 20 00000024 A908                          S                             CMP    #8,R1L
 21 00000026 4FF2                          S                             BLE    _$F00000
 22          00000028                      S                     _$F00003: .EQU  $
 23 00000028 0180                       14              SLEEP
 24 0000002A 40D4                       15              BRA     START
 25                                     16    ;
 26 0000002C                            17    CHANGE
 27 0000002C 6A2900000000               18              MOV.B   @ANS:32,R1L
 28                                     19              .IF.B (R1L<LT>R0L)
 29 00000032 1C98                          S                             CMP  R1L,R0L
 30 00000034 58F00006                      S                             BLE  _$I00000
 31 00000038 6AA800000000               20                       MOV.B   R0L,@ANS:32
 32                                     21              .ENDI
 33          0000003E                      S                     _$I00000: .EQU  $
 34          0000003E                      S                     _$I00001: .EQU  $
 35 0000003E 5470                       22              RTS
 36                                     23    ;
 37 00001000                            24              .SECTION BBB,DATA,LOCATE=H'00001000
 38 00001000                            25    DATA
 39 00001000 03020405                   26              .DATA.B H'03,H'02,H'04,H'05
 40 00001004 01080607                   27              .DATA.B H'01,H'08,H'06,H'07
 41                                     28    ;
 42 00000000                            29              .SECTION CCC,DATA,ALIGN=2
 43 00000000                            30    ANS
 44 00000000 00000001                   31              .RES.B   1
 45                                     32    ;
 46 00000000                            33              .SECTION DDD,STACK,ALIGN=2
 47 00000000 00000500                   34              .RES.B   H'500
 48 00000500                            35    STACK
 49                                     36    ;
 50          00000000                   37              .END     START
(1)  (2)       (3)                     (4)  (5)                    (6)
 *****TOTAL ERRORS       0
 *****TOTAL WARNINGS     0
```

図 B.1　**ソースプログラムリスト例**

B.3 クロスリファレンスリスト

クロスリファレンスリストの内容は、次のとおりです。

(1) シンボル名

(2) セクション名

シンボルが含まれるセクションの名称です。最大 8 文字まで表示します。

(3) シンボル属性

シンボルの属性です。シンボルの属性には、次のものがあります。

表示なし ・・・・・・・ラベル定義
EQU ・・・・・・・・・・・.EQU 定義
ASGN・・・・・・・・・・.ASSIGN 定義
IMPT ・・・・・・・・・・外部参照
EXPT ・・・・・・・・・・外部定義
SCT・・・・・・・・・・・セクション名
REG ・・・・・・・・・・・.REG 定義
MDEF・・・・・・・・・・二重定義
UDEF ・・・・・・・・・・未定義

(4) シンボル値

シンボルの値です。8 桁の 16 進数で表示します。

(5) シンボル定義、参照のリスト行番号

シンボルを定義、参照している行のリスト行番号です。定義行にはアスタリスク（*）を表示します。

```
*** CROSS REFERENCE LIST

NAME                            SECTION  ATTR VALUE              SEQUENCE

AAA                             AAA      SCT  00000000      3*
ANS                             CCC           00000000      7    27    31
                                                            43*
BBB                             BBB      SCT  00001000     37*
CCC                             CCC      SCT  00000000     42*
CHANGE                          AAA           0000002C     15    26*
DATA                            BBB           00001000      8    38*
DDD                             DDD      SCT  00000000     46*
STACK                           DDD           00000500      5    48*
START                           AAA           00000000      4*   24    50
_$F00000                        AAA      EQU  0000001A     12*   21
_$F00001                        AAA      EQU  00000022     17*
_$F00002                        AAA      EQU  00000024     11    19*
_$F00003                        AAA      EQU  00000028     22*
_$I00000                        AAA      EQU  0000003E     30    33*
_$I00001                        AAA      EQU  0000003E     34*
          (1)                    (2)      (3)  (4)             (5)
```

図 B.2　**クロスリファレンスリスト例**

B.4 セクション情報リスト

セクション情報リストの内容は、次のとおりです。

(1) セクション名

(2) セクション属性

セクションの属性です。形式種別とセクション属性を表示します。

(a) 形式種別

ABS ・・・・・・・・・・絶対アドレス形式

REL ・・・・・・・・・・相対アドレス形式

(b) セクション属性

CODE・・・・・・・・・コードセクション

DATA・・・・・・・・・データセクション

STACK・・・・・・・・スタックセクション

COMMON・・・・・コモンセクション

DUMMY ・・・・・・ダミーセクション

(3) セクションサイズ

セクションのサイズです。16 進数で表示します。

(4) セクション先頭アドレス

絶対アドレスセクションの先頭アドレスです。相対アドレスセクションには表示しません。

```
*** SECTION DATA LIST

SECTION                          ATTRIBUTE    SIZE            START

AAA                              REL-CODE    0000040
BBB                              ABS-DATA    0000008          001000
CCC                              REL-DATA    0000001
DDD                              REL-STACK   0000500
                  (1)                (2)        (3)             (4)
```

図 B.3 セクション情報リスト例

付録 C.　エラーメッセージ

C.1　エラーメッセージの種類

エラーメッセージには、次の 3 種類があります。

（1）起動時のエラー

アセンブラを起動するコマンドラインに対するエラーです。ファイルの指定、コマンドラインオプションの指定に誤りがある場合などに表示します。

起動時のエラーが発生した場合は、ソースプログラムをアセンブルしません。

表示形式は、次のとおりです。

（a）MS-DOS 版

ファイル名　行番号　エラー内容

（b）UNIX 版

"ファイル名",　line　行番号:　エラー内容

起動時のエラーのエラー番号は 0～99 番です。

（2）ソースプログラムのエラー

ソースプログラムの文法エラーです。エラーのあるソースステートメントの直後に表示します。

表示形式は、次のとおりです。

（a）MS-DOS 版

画面表示 ･･･････････････ ファイル名　行番号　エラー内容

アセンブルリスト中 ････ *****ERROR　　エラー番号（ファイル名）

*****WARNING　エラー番号（ファイル名）

（b）UNIX 版

画面表示 ･･･････････････ "ファイル名",　line　行番号:　エラー内容

アセンブルリスト中 ････ *****ERROR　　エラー番号（ファイル名）

*****WARNING　エラー番号（ファイル名）

ソースプログラムのエラー番号は、次のように分類されています。

100 番台・・・・・・・・・ソースプログラム全般の文法に関するエラー
200 番台・・・・・・・・・シンボルに関するエラー
300 番台・・・・・・・・・オペレーション、オペランドに関するエラー
400 番台・・・・・・・・・演算に関するエラー
500 番台・・・・・・・・・アセンブラ制御命令に関するエラー
600 番台・・・・・・・・・プリプロセッサ制御文に関するエラー
800 番台・・・・・・・・・ソースプログラム全般のウォーニング

(3) 致命的なエラー

アセンブラの動作環境に対するエラーです。アセンブルするソースプログラムのサイズに対してメモリ容量、ディスク容量が不足している場合などに表示します。

致命的なエラーが発生した場合は、ソースプログラムのアセンブルを中止します。

表示形式は、次のとおりです。

(a) MS-DOS 版

ファイル名　行番号　エラー内容

(b) UNIX 版

"ファイル名",　line　行番号:　エラー内容

致命的なエラーのエラー番号は 900 番台です。

C.2 起動時のエラー

表 C.1 に、起動時のエラー一覧を示します。

表 C.1　**起動時のエラー一覧**

10	NO INPUT FILE SPECIFIED
	内容：　入力ファイルの指定がない。 **対策：　入力ファイルを指定してください。**
20	CANNOT OPEN FILE **ファイル名**
	内容：　ファイルをオープンできない。 **対策：　ファイル名、ディレクトリ名などを見直してください。**
30	INVALID COMMAND PARAMETER
	内容：　コマンドラインオプションに誤りがある。または、サブコマンドファイルの内容に誤りがある。 **対策：　コマンドラインオプションを見直してください。または、サブコマンドファイルの内容を見直してください。**
40	CANNOT ALLOCATE MEMORY
	内容：　入力ファイルの処理中にメモリが足りなくなった。 **対策：　通常発生しません。もし発生した場合は、当社営業担当までご連絡ください。**
50	COMPLETED FILE NAME TOO LONG **ファイル名**
	内容：　ディレクトリを含めたファイル名が長い。 **対策：　ディレクトリの構造を簡単にしてください。** **補足：　本メッセージはウォーニング扱いとし、アセンブルは続行します。** **本メッセージが表示された場合は、シミュレータ・デバッガで正しくデバッグできない場合があります。**

C.3 ソースプログラムのエラー

表 C.2 に、ソースプログラムのエラー一覧を示します。

表 C.2 ソースプログラムのエラー一覧（1）

100	OPERATION TOO COMPLEX
	内容： 演算が複雑である。 対策： 演算の内容を簡単にしてください。
101	SYNTAX ERROR IN SOURCE STATEMENT
	内容： ソースステートメントに構文上の誤りがある。 対策： ソースステートメント全体を見直してください。
102	SYNTAX ERROR IN DIRECTIVE
	内容： ソースステートメントに構文上の誤りがある。 対策： ソースステートメント全体を見直してください。
103	.END NOT FOUND
	内容： プログラムに.END がない。 対策： .END を記述してください。
104	LOCATION COUNTER OVERFLOW
	内容： ロケーションカウンタ値（アドレス）が最大値を超えた。 対策： プログラムを縮小してください。
105	ILLEGAL INSTRUCTION IN STACK SECTION
	内容： スタックセクションに記述できない命令を記述した。 対策： 命令を削除してください。
106	TOO MANY ERRORS
	内容： エラーの数が多いので表示を打ち切ります。 対策： ソースステートメント全体を見直してください。
108	ILLEGAL CONTINUATION LINE
	内容： 誤った継続行を記述した。 対策： 継続行の記述を訂正してください。
109	LINE NUMBER OVERFLOW
	内容： プログラムの行数が 65,535 行を超えた。 対策： プログラムを分割してください。
200	UNDEFINED SYMBOL REFERENCE
	内容： 参照しているシンボルが定義されていない。 対策： シンボルを定義してください。

表 C.2　ソースプログラムのエラー一覧（2）

201	ILLEGAL SYMBOL OR SECTION NAME
	内容：　レジスタニーモニック、演算子、ロケーションカウンタをシンボル名、セクション名に指定した。 対策：　シンボル名、セクション名を訂正してください。
202	ILLEGAL SYMBOL OR SECTION NAME
	内容：　シンボル名、セクション名に誤りがある。 対策：　シンボル名、セクション名を訂正してください。
203	ILLEGAL LOCAL LABEL
	内容：　ローカルラベルの指定に誤りがある。 対策：　ローカルラベルの指定を訂正してください。
300	ILLEGAL MNEMONIC
	内容：　命令のニーモニックが誤っている。 対策：　命令のニーモニックを訂正してください。
301	TOO MANY OPERANDS OR ILLEGAL COMMENT
	内容：　命令のオペランドが多い。または、コメントが誤っている。 対策：　オペランド、またはコメントを訂正してください。
304	LACKING OPERANDS
	内容：　命令のオペランドが少ない。 対策：　オペランドを訂正してください。
306	SYNTAX ERROR IN REGISTER LIST
	内容：　複数レジスタの指定に誤りがある。 対策：　複数レジスタの指定を訂正してください。
307	ILLEGAL ADDRESSING MODE
	内容：　命令のオペランドに指定できないアドレス形式を指定した。 対策：　アドレス形式を訂正してください。
308	SYNTAX ERROR IN OPERAND
	内容：　命令のオペランドに構文上の誤りがある。 対策：　オペランドを訂正してください。
400	CHARACTER CONSTANT TOO LONG
	内容：　文字定数が 4 文字を超えた。 対策：　文字定数を訂正してください。

表 C.2　ソースプログラムのエラー一覧（3）

402	ILLEGAL VALUE IN OPERAND
	内容：　命令のオペランドに範囲外の値を指定した。 対策：　値を訂正してください。 補足：　範囲外の部分を無視してオブジェクトコードを生成します。
403	ILLEGAL OPERATION FOR RELATIVE VALUE
	内容：　相対値に対して乗除算、論理演算を行なった。 対策：　演算の内容を訂正してください。
404	ILLEGAL IMMEDIATE DATA
	内容：　#1、2、4、#0〜3、#0〜7 のオペランドで、値に相対値を指定した。 対策：　相対値は指定できません。値を訂正してください。
407	MEMORY OVERFLOW
	内容：　演算の途中でメモリが足りなくなった。 対策：　演算の内容を簡単にしてください。
408	DIVISION BY ZERO
	内容：　0 除算を行なった。 対策：　演算の内容を訂正してください。
409	REGISTER IN EXPRESSION
	内容：　演算にレジスタを指定した。 対策：　演算の内容を訂正してください。
411	INVALID STARTOF/SIZEOF OPERAND
	内容：　STARTOF 演算、SIZEOF 演算にセクション名以外を指定した。 対策：　演算の内容を訂正してください。
412	ILLEGAL SYMBOL IN EXPRESSION
	内容：　シフト演算で相対値を指定した。 対策：　演算内容を訂正してください。
500	SYMBOL NOT FOUND
	内容：　ラベルフィールドにシンボルがない。 対策：　シンボルを記述してください。
501	ILLEGAL ADDRESS VALUE IN OPERAND
	内容：　.SECTION、.ORG に範囲外の値を指定した。 対策：　値を訂正してください。

表 C.2　ソースプログラムのエラー一覧（4）

502	ILLEGAL SYMBOL IN OPERAND
	内容：　後方参照の絶対値を指定するオペランドに、相対値、前方参照値を指定した。または、未定義のシンボルを指定した。 対策：　値を訂正してください。
503	UNDEFINED EXPORT SYMBOL
	内容：　.EXPORT で外部定義したシンボルをプログラム内で定義していない。 対策：　シンボルを外部定義から削除してください。
504	INVALID RELATIVE SYMBOL IN OPERAND
	内容：　.EQU、.ASSIGN に、外部参照値、前方参照値を指定した。または、セクションの先頭からのオフセット値で表現できない相対値を指定した。 対策：　値を訂正してください。
505	ILLEGAL OPERAND
	内容：　制御命令のオペランドの内容に誤りがある。 対策：　オペランドの内容を訂正してください。
506	ILLEGAL OPERAND
	内容：　制御命令のオペランドの内容に誤りがある。 対策：　オペランドの内容を訂正してください。
508	ILLEGAL VALUE IN OPERAND
	内容：　制御命令のオペランドに範囲外の値を指定した。 対策：　値を訂正してください。
510	ILLEGAL BOUNDARY VALUE
	内容：　.SECTION、.ALIGN の境界調整数に誤りがある。 対策：　境界調整数を訂正してください。
511	ILLEGAL DISPLACEMENT SIZE
	内容：　.DISPSIZE のビット数に誤りがある。 対策：　ビット数を訂正してください。
512	ILLEGAL EXECUTION START ADDRESS
	内容：　.END の実行開始アドレスに誤りがある。 対策：　実行開始アドレスを訂正してください。
513	ILLEGAL REGISTER NAME
	内容：　.REG のレジスタ名に誤りがある。 対策：　レジスタ名を訂正してください。

表 C.2　ソースプログラムのエラー一覧（5）

514	INVALID EXPORT SYMBOL
	内容：　.EXPORT、.GLOBAL に外部定義できないシンボルを指定した。 対策：　シンボルを外部定義から削除してください。
516	EXCLUSIVE DIRECTIVES
	内容：　制御命令の指定内容が矛盾している。 対策：　関連する制御命令を含めて見直してください。
517	INVALID VALUE IN OPERAND
	内容：　.ORG に外部参照値、前方参照値を指定した。または、他セクションの相対アドレス値を指定した。 対策：　値を訂正してください。
518	INVALID IMPORT SYMBOL
	内容：　.IMPORT にプログラム内で定義しているシンボルを指定した。 対策：　シンボルを外部参照から削除してください。
520	ILLEGAL .CPU DIRECTIVE POSITION
	内容：　.CPU がプログラムの最初にない。または、.CPU を複数回指定した。 対策：　.CPU はプログラムの最初に指定してください。または、.CPU の指定を 1 回にしてください。
521	ILLEGAL SYMBOL IN OPERAND
	内容：　OPTIMIZE オプション指定時において、定数値を指定するオペランドにアドレスを値に持つシンボル、または、ロケーションカウンタ値を指定した。 対策：　値を訂正してください。 補足：　アドレスを値に持つシンボル、ロケーションカウンタ値を指定する場合は、OPTIMIZE コマンドラインオプションを指定しないでください。
523	ILLEGAL OPERAND
	内容：　制御命令のオペランドの内容に誤りがある。 対策：　オペランドの内容を訂正してください。
524	ILLEGAL ADDRESSING SPACE SIZE
	内容：　.CPU 制御命令のオペランドに、許されないアドレス空間のビット幅を指定した。 対策：　アドレス空間のビット幅を訂正してください。
525	ILLEGAL .LINE DIRECTIVE POSITION
	内容：　マクロ展開中または条件付き繰り返し展開中に.LINE がある。 対策：　.LINE を削除してください。
526	STRING TOO LONG
	内容：　指定した文字列が 255 文字を超えた。 対策：　指定文字列を 255 文字以下になるようにしてください。

表 C.2　ソースプログラムのエラー一覧（6）

600	INVALID CHARACTER
	内容：　不当な文字がある。 対策：　ソースステートメント全体を見直してください。
601	INVALID DELIMITER
	内容：　区切り文字に誤りがある。 対策：　ソースステートメント全体を見直してください。
602	INVALID CHARACTER STRING FORMAT
	内容：　文字列に誤りがある。 対策：　ソースステートメント全体を見直してください。
603	SYNTAX ERROR IN SOURCE STATEMENT
	内容：　ソースステートメントに構文上の誤りがある。 対策：　ソースステートメント全体を見直してください。
604	ILLEGAL SYMBOL IN OPERAND
	内容：　後方参照の絶対値を指定するオペランドに、相対値、前方参照値を指定した。または、未定義のシンボルを指定した。 対策：　値を訂正してください。
610	MULTIPLE MACRO NAMES
	内容：　マクロ定義（.MACRO）でマクロ名が重複した。 対策：　マクロ名を訂正してください。
611	MACRO NAME NOT FOUND
	内容：　.MACRO のオペランドにマクロ名がない。 対策：　マクロ名を記述してください。
612	ILLEGAL MACRO NAME
	内容：　.MACRO のマクロ名に誤りがある。 対策：　マクロ名を訂正してください。 補足：　マクロ名には、実行命令、制御命令（ピリオド（.）を除く）、制御文（ピリオド（.）を除く）のニーモニックは指定できません。
613	ILLEGAL .MACRO DIRECTIVE POSITION
	内容：　マクロ本体(.MACRO～.ENDM 間)、.AREPEAT～.AENDR 間、.AWHILE～.AENDW 間に.MACRO がある。 対策：　.MACRO を削除してください。
614	MULTIPLE MACRO PARAMETERS
	内容：　マクロ定義（.MACRO）の仮引数の宣言で、仮引数名が重複した。 対策：　仮引数名を訂正してください。

表 C.2　ソースプログラムのエラー一覧（7）

615	ILLEGAL .END DIRECTIVE POSITION
	内容：　マクロ本体（.MACRO**～**.ENDM **間）に**.END **がある。** **対策：　**.END **を削除してください。**
616	MACRO DIRECTIVES MISMATCH
	内容：　**.ENDM **が.MACRO **に対応していない。または、**.EXITM **がマクロ本体（**.MACRO **～**.ENDM **間）、**.AREPEAT**～**.AENDR **間、**.AWHILE**～**.AENDW **間以外にある。** **対策：　**.ENDM**、**.EXITM **を削除してください。**
618	MACRO EXPANSION TOO LONG
	内容：　マクロ展開で 1 **行の文字数が** 8,192 **文字を超えた。** **対策：　**8,192 **文字以下になるように訂正してください。**
619	ILLEGAL MACRO PARAMETER
	内容：　マクロコールでマクロパラメータの仮引数名に誤りがある。または、マクロ本体（.MACRO**～**.ENDM **間）の仮引数名に誤りがある。** **対策：　仮引数名を訂正してください。** **補足：　マクロ本体の仮引数名が誤りの場合は、マクロ展開時にエラーになります。**
620	UNDEFINED PREPROCESSOR VARIABLE
	内容：　参照しているプリプロセッサ変数が定義されていない。 **対策：　プリプロセッサ変数を定義してください。**
621	ILLEGAL .END DIRECTIVE POSITION
	内容：　マクロ展開中に.END **がある。** **対策：　**.END **を削除してください。**
622	')' NOT FOUND
	内容：　マクロ処理除外の閉じカッコがない。 **対策：　マクロ処理除外の閉じカッコを指定してください。**
623	SYNTAX ERROR IN STRING FUNCTION
	内容：　文字列操作関数に構文上の誤りがある。 **対策：　文字列操作関数を訂正してください。**
624	MACRO PARAMETERS MISMATCH
	内容：　マクロコールで位置指定のマクロパラメータの数が多い。 **対策：　マクロパラメータの数を訂正してください。**
630	SYNTAX ERROR IN OPERAND
	内容：　構造化アセンブリ制御文のオペランドに構文上の誤りがある。 **対策：　ソースステートメント全体を見直してください。**

表 C.2　ソースプログラムのエラー一覧（8）

631	END DIRECTIVE MISMATCH
	内容：　対になる制御文で、終了の制御文が一致しない。 対策：　制御文を見直してください。
632	SYNTAX ERROR IN OPERAND
	内容：　構造化アセンブリ制御文のオペランドのコンディションコードに誤りがある。 対策：　コンディションコードを訂正してください。
633	ILLEGAL .BREAK OR .CONTINUE DIRECTIVE POSITION
	内容：　.BREAK、.CONTINUE が.FOR［U］～.ENDF 間、.WHILE～.ENDW 間、.REPEAT～.UNTIL 間以外にある。 対策：　.BREAK、.CONTINUE を削除してください。
634	EXPANSION TOO LONG
	内容：　構造化アセンブリ展開で 1 行の文字数が 8,192 文字を超えた。 対策：　8,192 文字以下になるように訂正してください。
640	SYNTAX ERROR IN OPERAND
	内容：　条件付きアセンブリ制御文のオペランドに構文上の誤りがある。 対策：　ソースステートメント全体を見直してください。
641	INVALID RELATIONAL OPERATOR
	内容：　条件付きアセンブリ制御文のオペランドの関係演算子に誤りがある。 対策：　関係演算子を訂正してください。
642	ILLEGAL .END DIRECTIVE POSITION
	内容：　.AREPEAT～.AENDR 間、.AWHILE～.AENDW 間に.END がある。 対策：　.END を削除してください。
643	DIRECTIVE MISMATCH
	内容：　.AREPEAT、.AWHILE に対する.AENDR、.AENDW が対になっていない。 対策：　制御文を見直してください。
644	ILLEGAL .AENDW OR .AENDR DIRECTIVE POSITION
	内容：　.AIF～.AENDI 間に.AENDW、.AENDR がある。 対策：　.AENDW、.AENDR を削除してください。
645	EXPANSION TOO LONG
	内容：　.AREPEAT、.AWHILE 展開で 1 行の文字数が 8,192 文字を超えた。 対策：　8,192 文字以下になるように訂正してください。
650	INVALID INCLUDE FILE
	内容：　.INCLUDE のファイル名に誤りがある。 対策：　ファイル名を訂正してください。

表 C.2　ソースプログラムのエラー一覧（9）

651	CANNOT OPEN INCLUDE FILE
	内容：　.INCLUDE のファイルをオープンできない。 対策：　ファイル名を訂正してください。
652	INCLUDE NEST TOO DEEP
	内容：　ファイルインクルードのネストが 30 レベルを超えた。 対策：　ネストを 30 レベル以下にしてください。
653	SYNTAX ERROR IN OPERAND
	内容：　.INCLUDE のオペランドに構文上の誤りがある。 対策：　オペランドを訂正してください。
660	.ENDM NOT FOUND
	内容：　.MACRO に対する.ENDM がない。 対策：　.ENDM を記述してください。
661	END DIRECTIVE NOT FOUND
	内容：　構造化アセンブリ制御文で終了の制御文が足りない。 対策：　終了の制御文を記述してください。
662	ILLEGAL .END DIRECTIVE POSITION
	内容：　.AIF～.AENDI 間に.END がある。 対策：　.END を削除してください。
663	ILLEGAL .END DIRECTIVE POSITION
	内容：　インクルードファイル中に.END がある。 対策：　.END を削除してください。
664	ILLEGAL .END DIRECTIVE POSITION
	内容：　.AIF～.AENDI 間に.END がある。 対策：　.END を削除してください。
665	ILLEGAL SYMBOL IN OPERAND
	内容：　OPTIMIZE コマンドラインオプション指定時において、プリプロセッサ制御命令にプリプロセッサ変数以外のシンボルを指定した。 対策：　シンボルを訂正してください。 補足：　プリプロセッサ変数以外のシンボルを指定する場合は、OPTIMIZE コマンドラインオプションを指定しないでください。
667	EXPANSION TOO LONG
	内容：　.DEFINE で 1 行の文字数が 8,192 文字を超えた。 対策：　8,192 文字以下になるように訂正してください。

表 C.2　ソースプログラムのエラー一覧（10）

668	ILLEGAL VALUE IN OPERAND
	内容：　.AIFDEF のオペランドに誤りがある。 対策：　.AIFDEF のオペランドは、置換シンボルを指定してください。
669	STRING TOO LONG
	内容：　指定した文字列が 255 文字を超えた。 対策：　指定文字列を 255 文字以下になるようにしてください。
800	SYMBOL NAME TOO LONG
	内容：　シンボル名が 32 または 251 文字を超えた。 対策：　シンボル名を訂正してください。 補足：　プリプロセッサ変数名または DEFINE 置換シンボル名は 32 文字まで、それ以外のシンボル名は 251 文字までを有効とします。
801	MULTIPLE SYMBOLS
	内容：　シンボルの定義でシンボル名が重複した。 対策：　シンボル名を訂正してください。
805	ILLEGAL OPERATION SIZE
	内容：　構造化アセンブリ制御文の分岐サイズ（:8、:16）に誤りがある。 対策：　分岐サイズを訂正してください。
807	ILLEGAL OPERATION SIZE
	内容：　命令のサイズに誤りがある。 対策：　サイズを訂正してください。
808	ILLEGAL CONSTANT SIZE
	内容：　整数定数の解釈サイズ（.xx）に誤りがある。 対策：　解釈サイズを訂正してください。 補足：　解釈サイズには、バイト（.B）、ワード（.W）があり、それぞれ 1 バイト、2 バイトの符号付きの値として解釈します。
810	TOO MANY OPERANDS
	内容：　命令のオペランドが多い。 対策：　オペランドを訂正してください。
811	ILLEGAL SYMBOL DEFINITION
	内容：　ラベルフィールドにシンボルを指定した。 対策：　シンボルを削除してください。
812	SECTION OR MODULE NAME TOO LONG
	内容：　セクション名、モジュール名が 251 文字を超えた。 対策：　セクション名、モジュール名を訂正してください。 補足：　251 文字までを有効とします。

表 C.2　ソースプログラムのエラー一覧（11）

813	SECTION ATTRIBUTE MISMATCH
	内容：　セクションの再開で、異なるセクション属性、形式種別を指定した。 対策：　セクション属性、形式種別を訂正してください。 補足：　絶対アドレスセクションの再開では、LOCATE=先頭アドレスの指定はできません。
814	ILLEGAL OBJECT CODE SIZE
	内容：　確保サイズ（:8、:16、:24、:32）に誤りがある。 対策：　確保サイズを訂正してください。 補足：　#xx:2、#xx:3 はマニュアル記述上の記号です。アセンブラでは記述できません。
815	MULTIPLE MODULE NAMES
	内容：　.PROGRAM でモジュール名を複数回設定した。 対策：　設定は 1 回にしてください。
816	START ODD ADDRESS
	内容：　偶数バイトのデータまたはデータ領域を奇数アドレスから確保した。 対策：　偶数アドレスに訂正してください。
817	OPERATION SIZE MISMATCH
	内容：　バイトサイズ（.B）に対して、@-SP、@SP+を指定した。 対策：　SP（スタックポインタ）が奇数値になるため使用しないでください。 補足：　そのままのオブジェクトコードを生成します。
818	ILLEGAL ACCESS SIZE
	内容：アクセスサイズ（:8、:16）に誤りがある。 対策：アクセスサイズを訂正してください。
825	ILLEGAL INSTRUCTION IN DUMMY SECTION
	内容：　ダミーセクションに記述できない命令を記述した。 対策：　命令を削除してください。
830	OPERATION SIZE MISMATCH
	内容：　バイトサイズ（.B）に対して、ERn、Rn を指定した。または、ワードサイズ（.W）に対して、ERn を指定した。 対策：　レジスタを訂正してください。 補足：　バイトサイズでは RnL、ワードサイズでは Rn としてオブジェクトコードを生成します。
832	MULTIPLE 'P' DEFINITIONS
	内容：　シンボル P とデフォルトセクション P とが重複した。 対策：　P が重複しないようにしてください。

表 C.2　ソースプログラムのエラー一覧（12）

835	ILLEGAL VALUE IN OPERAND
	内容：　命令のオペランドに範囲外の値を指定した。 **対策：　値を訂正してください。** **補足：　範囲外の部分を無視してオブジェクトコードを生成します。**
836	CONSTANT SIZE OVERFLOW
	内容：　整数定数の値が整数定数の解釈サイズ（.B、.W）の範囲外である。 **対策：　整数定数の値を訂正してください。** **補足：　解釈サイズには、バイト（.B）、ワード（.W）があり、それぞれ 1 バイト、2 バイトの符号付きの値として解釈します。**
837	SOURCE STATEMENT TOO LONG
	内容：　ソースステートメントの 1 行の文字数が 8,192 文字を超えた。 **対策：　8,192 文字以下になるように訂正してください。**
838	ILLEGAL CHARACTER CODE
	内容：　コメント、文字列以外にシフト JIS コード、EUC コードまたは LATIN1 コードを指定した。または、SJIS、EUC および LATIN1 コマンドラインオプションの指定がない。 **対策：　シフト JIS コード、EUC コードおよび LATIN1 コードはコメント、文字列内に指定してください。または、SJIS、EUC、LATIN1 コマンドラインオプションを指定してください。**
850	ILLEGAL SYMBOL DEFINITION
	内容：　ラベルフィールドにシンボルを指定した。 **対策：　シンボルを削除してください。**
851	MACRO SERIAL NUMBER OVERFLOW
	内容：　マクロ生成番号が 99999 を超えた。 **対策：　マクロコールの回数を減らしてください。**
852	UNNECESSARY CHARACTER
	内容：　制御文のオペランドの終了後に文字がある。 **対策：　オペランドを訂正してください。**
853	NEGATIVE IMMEDIATE VALUE
	内容：　.FOR［U］の増分値に、#-xx を記述した。 **対策：　-#xx に訂正してください。** **補足：　そのまま.FOR［U］を展開します。**
854	.AWHILE ABORTED BY .ALIMIT
	内容：　展開回数が.ALIMIT 制御文で設定した上限値に達したため、展開を中断した。 **対策：　繰り返し展開する条件を見直してください。**

C.4 致命的なエラー

表 C.3 に、致命的なエラー一覧を示します。

表 C.3　**致命的なエラー一覧（1）**

900	FILE NAME TOO LONG
	内容：　ディレクトリを含めたファイル名が長い。 **対策：　ディレクトリの構造を簡単にしてください。**
901	SOURCE FILE INPUT ERROR
	内容：　ソースファイルで入力エラーが発生した。 **対策：　ディスクの容量が十分か確認してください。**
902	MEMORY OVERFLOW
	内容：　メモリが足りなくなった。（ソースプログラムが大きすぎて処理できない） **対策：　プログラムを分割してください。**
903	LISTING FILE OUTPUT ERROR
	内容：　リストファイルで出力エラーが発生した。 **対策：　ディスクの容量が十分か確認してください。**
904	OBJECT FILE OUTPUT ERROR
	内容：　オブジェクトファイルで出力エラーが発生した。 **対策：　ディスクの容量が十分か確認してください。**
905	MEMORY OVERFLOW
	内容：　メモリが足りなくなった。（ソースプログラム行に関する情報を処理できない） **対策：　プログラムを分割してください。**
906	MEMORY OVERFLOW
	内容：　メモリが足りなくなった。（シンボルに関する情報を処理できない） **対策：　プログラムを分割してください。**
907	MEMORY OVERFLOW
	内容：　メモリが足りなくなった。（セクションに関する情報を処理できない） **対策：　プログラムを分割してください。**
908	SECTION OVERFLOW
	内容：　セクション数が 65,535 **を超えた。** **対策：　プログラムを分割してください。**
909	SYMBOL OVERFLOW
	内容：　シンボル数が 65,535 **を超えた。** **対策：　プログラムを分割してください。**

表 C.3　致命的なエラー一覧（2）

910	SOURCE LINE NUMBER OVERFLOW
	内容：　リスティング行番号が 65,535 を超えた。 対策：　プログラムを分割してください。
911	IMPORT SYMBOL OVERFLOW
	内容：　外部参照シンボル数が 65,535 を超えた。 対策：　外部参照の数を減らしてください。
912	EXPORT SYMBOL OVERFLOW
	内容：　外部定義シンボル数が 65,535 を超えた。 対策：　外部定義の数を減らしてください。
933	LACKING CPU SPECIFICATION
	内容：　CPU 種別が設定されていない。 対策：　CPU 種別を CPU コマンドラインオプション、.CPU 制御命令、H38CPU 環境変数のいずれかで指定してください。
935	SUBCOMMAND FILE INPUT ERROR
	内容：　サブコマンドファイルで入力エラーが発生した。 対策：　ディスクの容量が十分か確認してください。
936	SUPPLEMENT FILE OUTPUT ERROR
	内容：　付加情報ファイルの出力時に、エラーが発生した。 対策：　ディスクの空き容量またはディレクトリの構造を確認してください。ディスク上の不要なファイルを削除するなどして、必要な空き容量を確保してください。または、ディレクトリの構造を簡単にしてください。
954	MEMORY OVERFLOW
	内容：　メモリが足りなくなった。（ソースプログラム行に関する情報を処理できない） 対策：　プログラムを分割してください。
955	LOCAL BLOCK NUMBER OVERFLOW
	内容：　ローカルラベルの有効範囲であるローカルブロックの個数が 100,000 を超えた。 対策：　ソースプログラムを分割してください。
956	EXPAND FILE INPUT/OUTPUT ERROR
	内容：　プリプロセッサ展開出力のファイル出力時にエラーが発生した。 対策：　ディスクの空き容量を確認してください。ディスク上に不要なファイルを削除するなどして、必要な空き領域を確保してください。
957	MEMORY OVERFLOW
	内容：　メモリが足りない。 対策：　ソースプログラムを分割してください。

致命的なエラーで、対策に示した方法でもエラーが回避できない場合は、当社営業担当までご連絡ください。

付録 D.　旧バージョンとの相違点

新バージョン（H8S, H8/300 シリーズクロスアセンブラ Ver. 3）と、旧バージョン（H8S, H8/300 シリーズクロスアセンブラ Ver. 2）の相違点を示します。

D.1　新規機能

(1) コマンドラインオプション

表 D.1 に、新バージョンで追加したコマンドラインオプションを示します。

表 D.1　コマンドラインオプションの追加

オプション	機能	参照個所
EXPAND	マクロ展開、条件付きアセンブル、構造化アセンブル、ファイルのインクルードを行なった後のアセンブラソースファイルを出力します。	12.3.8
LATIN1	文字列、コメント内の LATIN1 コード記述をサポートします。	12.7.4
ABS8 ABS16	8 または 16 ビット絶対アドレス形式でアクセスするシンボルを指定します。	12.3.9

D.2　機能改善

(1) アセンブラ制御命令のアクセスサイズ指定

外部シンボルを 8 または 16 ビット絶対アドレス形式でアクセス指定する機能をサポートしました。

アセンブラ制御命令.IMPORT,.EXPORT および.GLOBAL にアクセスサイズ（:8 または:16）を付けて指定します。アクセスサイズ省略時は、CPU のアドレス空間により決定します。

■例

```
> asm38 aaa.mar -CPU=300HA
aaa.mar の内容
        .IMPORT     sym1:8
        .EXPORT     sym2:16
        .GROBAL     sym3
sym2    .EQU        H'00FF8000
        MOV.B       @sym1,R1H      ;8 ビット
```

```
MOV.B          @sym2,R2H      ;16 ビット
MOV.B          @sym3,R3H      ;24 ビット
      :
      :
```

(2) アドレス形式の拡張

アドバンストモードのディスプレースメント付きレジスタ間接形式において、ディスプレースメントに 16 ビット絶対アドレスの指定範囲が変更になりました。

アクセス範囲は、アドレス空間のビット幅により異なります。

表 D.2 に、アドバンストモードのアクセス範囲を示します。

表 D.2　**アドバンストモードのアクセス範囲**

項番	CPU **種別：ビット幅**	**変更前のアクセス範囲**	**変更後のアクセス範囲**
1	H8S/2600A:20 H8S/2000A:20 H8/300HA:20	H'00000000〜H'00007FFF H'FFFF8000〜H'FFFFFFFF	H'00000000〜H'00007FFF H'000F8000〜H'000FFFFF H'FFFF8000〜H'FFFFFFFF
2	H8S/2600A［:24］ H8S/2000A［:24］ H8/300HA［:24］		H'00000000〜H'00007FFF H'00FF8000〜H'00FFFFFF H'FFFF8000〜H'FFFFFFFF
3	H8S/2600A:28 H8S/2000A:28		H'00000000〜H'00007FFF H'0FFF8000〜H'0FFFFFFF H'FFFF8000〜H'FFFFFFFF
4	H8S/2600A:32 H8S/2000A:32		H'00000000〜H'00007FFF H'FFFF8000〜H'FFFFFFFF

■例

```
. CPU   2600A:24
MOV.B   @(H'00FF8000,ER1),R2L
;旧バージョンでは、「MOV.B@(H'00FF8000:32,ER1),R2L」と解釈します。
;新バージョンでは、「MOV.B@(H'00FF8000:16,ER1),R2L」と解釈します。

MOV.B   @(H'00FFFFFF:16,ER1),R2L
;旧バージョンでは、エラー402 となりオブジェクトファイルは出力しません。
;新バージョンでは、「MOV.B@(H'00FFFFFF:16,ER1),R2L」と解釈します。
```

(3) デフォルト空間サイズの変更

H8S/2000, 2600 アドバンストモードのデフォルト空間サイズを 32 ビットから 24 ビット

へ変更しました。

CPU 種別	アドレス空間のビット幅	デフォルトサイズ
2600A	20, 24, 28, 32	24
2000A	20, 24, 28, 32	24
300HA	20, 24	24

（4）制限事項の変更および追加

表 D.3 に、変更および追加した制限事項を示します。

表 D.3　**制限事項**

項番	項目	制限
1	**文字**	ASCII **文字、シフト** JIS **コード、**EUC **コード、**LATIN1 **コード**
2	1 **行文字数**	8,192 **文字まで**
3	**シンボル文字数**	251 **文字まで**[*1]
4	**文字列長**	255 **文字まで**
5	**ファイル名長**	251 **文字まで**

【注】　*1 **プリプロセッサ変数名、**DEFINE **置換シンボル名、マクロ名およびマクロ仮引数名は、**32 **文字までです。**

付録 E.　JIS コード表

表 E.1　JIS **コード表**

上位4ビット / 下位4ビット	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	”	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	¥	l	|
D	CR	GS	–	=	M	]	m	}
E	SO	RS	.	>	N	^	n	―
F	SI	US	/	?	O	―	o	DEL

枠内の文字が使用できます。

H8S, H8/300 シリーズ クロスアセンブラ
ユーザーズマニュアル

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

ADJ-702-038E