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。

User's Manual

M32Rファミリ用 C/C++コンパイラパッケージ V.5.00

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
www.renesas.com

Rev.1.00　2005.07

- Microsoft、MS-DOS、Windows および Windows NT は、米国 Microsoft Corporation の米国およびその他の国における商標または登録商標です。
- HP-UX は、米国 Hewlett-Packard Company のオペレーティングシステムの名称です。
- Sun、Solaris、Java およびすべての Java 関連の商標およびロゴは、米国およびその他の国における米国 Sun Microsystems, Inc.の商標または登録商標です。
- UNIX は、The Open Group の米国ならびにその他の国における登録商標です。
- Linux は、Linus Torvalds 氏の米国およびその他の国における登録商標あるいは商標です。
- Turbolinux の名称およびロゴは、Turbolinux, Inc.の登録商標です。
- IBM および AT は、米国 International Business Machines Corporation の登録商標です。
- HP 9000 は、米国 Hewlett-Packard Company の商品名称です。
- SPARC および SPARCstation は、米国 SPARC International, Inc.の登録商標です。
- Intel, Pentium は、米国 Intel Corporation の登録商標です。
- Adobe および Acrobat は、Adobe Systems Incorporated（アドビシステムズ社）の登録商標です。
- Netscape および Netscape Navigator は、米国およびその他の諸国の Netscape Communications Corporation 社の登録商標です。

その他すべてのブランド名および製品名は個々の所有者の登録商標もしくは商標です。

安全設計に関するお願い

- 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

- 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株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズ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。
- 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、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは責任を負いません。
- 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、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは、予告なしに、本資料に記載した製品又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、事前に株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又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。
- 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、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズはその責任を負いません。
- 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。株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは、適用可否に対する責任は負いません。
- 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、株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店へご照会ください。
- 本資料の転載、複製については、文書による株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズの事前の承諾が必要です。
- 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株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店までご照会ください。

製品の内容及び本書についてのお問い合わせ先

インストーラが生成する以下のテキストファイルに必要事項を記入の上、ツール技術サポート窓口support_tool@renesas.comまで送信ください。

¥SUPPORT¥製品名¥SUPPORT.TXT

株式会社ルネサス ソリューションズ

ツール技術サポート窓口	support_tool@renesas.com
ユーザ登録窓口	regist_tool@renesas.com
ホームページ	http://www.renesas.com/jp/tools

目次

はじめに

M3T-CC32R（以下CC32Rと略す）は、ルネサス32ビットRISCマイクロコンピュータM32Rファミリ用のソフトウェア開発支援ツール一式です。CC32Rは組み込み制御システムの開発に適した豊富な機能を提供します。CC32Rのマニュアルセットでは、CC32Rを使ってM32Rシステムをターゲットとしたプログラミングを行うための情報を提供します。

対象読者

CC32Rマニュアルセットは、M32Rシステム上で動作するプログラムをCまたはアセンブリ言語で開発するプログラマーを対象としています。したがって、本マニュアルの読者は、プログラミング言語（Cまたはアセンブリ言語）、ターゲットとなるM32Rファミリ、開発環境（使用するホストマシンやOS等）についての基礎知識があることを前提としています。

関連マニュアル

M32Rファミリ用の開発に関連するマニュアルを以下に示します。

■ M32Rファミリ ユーザーズマニュアル
■ M32Rファミリソフトウェアマニュアル

詳細は、「ルネサスマイコン」のサイトを参照してください。
URLは、
http://www.renesas.com/jpn/
です。

参考文献

ANSI-C言語仕様の詳細については以下の文献を参照してください。

■ ANSI/ISO 9899-1990 American National Standard for Programming Languages - C
(AmericanNationalStandardsInstitute,Inc.発行)

マニュアルの表記規則

CC32R マニュアルセットでは、とくに説明がない限り、以下のような記述規則に従っています。

■ 記号

表記	意味
斜体文字	実際に指定する場合、斜体文字部分を適する値や文字に置き換える。
a |*b*	| は複数の項目を区切り、どれか一つを選択することを示す。*a* |*b* ならば、*a*か*b*のいずれかを選択する。
[ ]	[ ] 内の項目は省略可能。
...	... の前に記述された項目が複数指定可能。
	記述省略。
<RET>	リターンキー入力。

■ 用語 (1/2)

表記	意味
M32R M32Rx	ルネサス32ビットRISCマイクロコンピュータ。
M32Rシステム	M32Rファミリを搭載したシステム。
CC32R	CC32Rクロスツールキット。
Cコンパイラ（cc32R）	CC32Rに含まれるCコンパイラ。
アセンブラ（as32R）	CC32Rに含まれるアセンブラ。
リンカ（lnk32R）	CC32Rに含まれるリンカ。
ライブラリアン（lib32R）	CC32Rに含まれるライブラリアン。
マップジェネレータ（map32R）	CC32Rに含まれるマップジェネレータ。
ロードモジュールコンバータ（lmc32R）	CC32Rに含まれるロードモジュールコンバータ。
リリースノート	CC32Rの製品パッケージに含まれているリリースに関する資料（はじめにお読みください）。
Cプログラム	C言語で記述されたプログラム。
アセンブリプログラム	アセンブリ言語で記述されたプログラム。
ソースファイル	Cやアセンブリ言語などのプログラミング言語のソースコードが書かれたテキストファイル。
オブジェクトモジュール	Cおよびアセンブリ言語のソースコードを、M32Rに対応したマシンコードに翻訳した結果のオブジェクトファイル。Cコンパイラおよびアセンブラによって生成される。
ロードモジュール	リンク済みのオブジェクトモジュールで、M32Rシステム上で実行可能な形式のファイル。リンカやロードモジュールコンバータによって生成される。

■ 用語 (2/2)

表記	意味
リンクマップ	オブジェクトモジュールやロードモジュールの、セクション配置情報と外部定義シンボル情報をリスト出力したもの。マップジェネレータによって生成される。
ライブラリ	M32Rに対応したライブラリファイル。ライブラリアンによって生成される。
C標準ライブラリ	CC32Rに含まれる、ANSI-C準拠のライブラリファイル。
ユーザーライブラリ	ユーザーがライブラリアンによって作成したライブラリ。
リターン値	呼び出された関数が処理結果として呼び出し側に返す値。関数値。
ローカル変数	関数内だけで有効な変数。局所変数。
Rx	任意のM32R汎用レジスタ。
CRx	任意のM32R制御レジスタ。
ＥＷＳ	エンジニアリングワークステーションの略称。
ＯＳ	オペレーティングシステムの略称。
デフォルト	ユーザーが値を指定しなかった場合に有効となる規定値。または、ユーザーが動作指定しなかった場合の処理。
空白文字	タブやスペースキーによって入力される空白文字。

本マニュアルの構成

本マニュアルの構成を以下に示します。

●第１部　アセンブラ　as32R

- ■ 第１章　アセンブラの概要
- ■ 第２章　アセンブラの起動方法
- ■ 第３章　アセンブリ言語仕様
- ■ 第４章　一般命令行の記述方法
- ■ 第５章　擬似命令行の記述方法
- ■ 第６章　マクロ命令行の記述方法
- ■ 第７章　アセンブラのメッセージ
- ■ 付録Ａ　一般命令一覧
- ■ 付録Ｂ　擬似命令リファレンス
- ■ 付録Ｃ　マクロ命令リファレンス
- ■ 付録Ｄ　アセンブルリストファイル
- ■ 付録Ｅ　M32R/EUC#5拡張命令対応
- ■ 付録Ｆ　浮動小数点対応
- ■ 付録Ｇ　制限事項

●第２部　リンカ　lnk32R

- ■ 第１章　リンカlnk32Rの概要
- ■ 第２章　リンカの起動
- ■ 第３章　ロードモジュールの生成
- ■ 第４章　セクション
- ■ 第５章　ROM化
- ■ 第６章　リンカのメッセージ

●第３部　マップジェネレータ　map32R

- ■ 第１章　マップジェネレータmap32Rの概要
- ■ 第２章　マップジェネレータの起動
- ■ 第３章　リンクマップファイル
- ■ 第４章　アクセス制御ファイル生成機能
- ■ 第５章　csvシンボルマップファイル生成機能
- ■ 第６章　マップジェネレータのメッセージ

●第４部　ライブラリアン　lib32R

- ■ 第１章　ライブラリアンlib32Rの概要
- ■ 第２章　ライブラリアンの起動
- ■ 第３章　ライブラリアンの出力
- ■ 第４章　ライブラリアンのメッセージ

●第５部　ロードモジュールコンバータlmc32R

- ■ 第１章　ロードモジュールコンバータlmc32Rの概要
- ■ 第２章　ロードモジュールコンバータの起動
- ■ 第３章　ロードモジュールコンバータの使用方法
- ■ 第４章　Sフォーマット
- ■ 第５章　ロードモジュールコンバータのメッセージ

第１部

アセンブラ　as32R

第１章
アセンブラas32Rの概要

1.1 as32Rの概要

1.1.1 機能

as32R は M3T-CC32R クロスツールキットに含まれているアセンブラで、以下のような機能があります。

■ アセンブリ言語ソースファイルをアセンブルし、オブジェクトモジュールを生成します。

■ as32R は以下のツールを呼び出します。
- a032R（マクロプロセッサ）
- a132R（アセンブルプロセッサ）
- alis32R(リスト生成ツール)
- parafilt(パラレルプロセッサ)

as32R が生成したオブジェクトモジュールからロードモジュールを得るには、CC32R クロスツールキットに含まれるリンカ（lnk32R）を使用します。

1.1.2 特長

本アセンブラの特長を以下に示します。

■ オペランドサイズの最適化

オペランドサイズによって複数の長さの命令が選択できる場合、最短の命令を選択します。

■ 命令位置の自動補正機能

ワード境界に配置する必要のある命令の位置を自動的に調整します。

■ 32 ビット即値操作のための数値補正機能

32 ビット定数やアドレスを簡潔に記述できる数値補正機能を備えています。

■ マクロ機能をサポートしています。

第２章
アセンブラの起動方法

2.1 アセンブラを起動するには

2.1.1 アセンブラの起動手順

アセンブラを起動するためには、環境変数を設定（2.1.2参照）した後、規則に従ってコマンド「as32R」を入力し、それを実行します（2.1.3参照）。

2.1.2 環境変数の設定

環境変数M32RBIN、M32RINC、M32RLIB、M32RTMPに正しいディレクトリを設定します（通常はインストール時に設定します）。設定方法は「M3T-CC32Rクロスツールキット V.XX Release Xリリースノート」を参照してください。環境変数の設定を省略すると、デフォルトのディレクトリを設定したことになります。

表2.1 環境変数のデフォルト

環境変数	デフォルトのディレクトリ
M32RBIN	/usr/local/M32R/bin
M32RINC	/usr/local/M32R/include
M32RLIB	/usr/local/M32R/lib
M32RTMP	/tmp

2.1.3 コマンド行の記述規則

アセンブラの起動コマンド「as32R」の、コマンド行の入力書式および規則を以下に示します。各起動オプションの詳細は2.2節、入出力ファイルについての詳細は2.1.4～2.1.6節を参照してください。

```
as32R  [-g] [-V] [-w] [-o output_filename] [-l list_filename]
       [-I dir] [-D[=def]] [-m32r] [-m32rx] [input_filename]
        <RET>
```

※[ ]で囲まれていない項目　　：必ず入力しなければならない項目
※[ ]で囲まれた項目　　：必要に応じて入力する項目
※　－の付いた記号　　：起動オプション（詳細は2.2節参照）
※<RET>　　：リターンキー入力

図2.1　as32Rコマンドの入力書式

- 各項目（コマンド名、オプション、入力ファイル名）の間は、1文字以上の空白文字で区切ります。
- オプションとそのパラメータとの間には任意の空白文字が入力できます。矛盾するオプションの組み合わせがあれば、後から指定したオプションが有効となります。
- コマンド行に入力できる文字数は255文字までです（リターンキー入力を除く）。
- *filename*にはアセンブリ言語ソースファイルを指定します。指定できるファイル数は1個です（複数指定不可）。
- オブジェクトモジュールをリンクするにはCC32Rクロスツールキットに含まれるリンカlnk32Rを使用してください（as32Rにオブジェクトのリンク機能はありません）。

2.1.4 入力ファイルの条件

アセンブラ処理できる入力ファイルの条件を表2.2に示します。これらの条件を満たさないファイルはアセンブルできません。

表2.2 入力ファイルの条件

条件項目	条件内容
入力可能なファイル	アセンブリ言語ソースファイル。 ファイル名の拡張子は必ずしも「.ms」である必要はありません。
記述命令	入力するアセンブリ言語ソースは、M32R一般命令、アセンブラの擬似命令およびマクロ命令のみで記述されていること。
使用できる名前の長さ	モジュール名　：206文字まで シンボル名　：243文字まで セクション名　：243文字まで プリプロセッサ変数　：32文字まで マクロ名　：32文字まで
使用できる名前の数	名前の数は、一度にそれぞれ65535個まで処理できます。ただし、開発環境のシステムのメモリ容量によっては制限を受けます。

2.1.5 出力ファイル名の命名

出力ファイル名は -o *output_filename* で指定した名前となります。この指定がなかった場合、デフォルトで表2.3のように命名されます。

表2.3 出力ファイル名（デフォルト）

ファイル名	内容
file.mo	アセンブルの結果出力される、オブジェクトモジュールファイル。 ファイル名は、ソースファイル名の拡張子が「.mo」に置き換わったもの（ソースファイル名に拡張子がない場合は、ソースファイル名に拡張子「.mo」を付加したもの）。

2.1.6 リストファイル名の命名

リストファイル名は -l *list_filename* で指定した名前となります。

2.2 アセンブラの起動オプション

2.2.1 アセンブラ起動オプション一覧

表2.4に、アセンブラの各起動オプションの機能について示します。

表2.4 アセンブラ起動オプション （1/2）

オプション	機能
-g	デバッグ時に必要な情報（デバッグ情報）をオブジェクトモジュールファイルに出力します。
-I *dir*	マクロ命令.INCLUDEで指定した、相対パス表記のインクルードファイルの検索対象となるディレクトリを指定します。インクルードファイルの検索順序は以下の①～③のとおりです。 ① ソースファイルの存在するディレクトリ ② 本オプションで指定されたディレクトリ*dir* ③ 環境変数M32RINCに設定されているディレクトリ （環境変数が設定されていない場合、 /usr/local/M32R/include） 本オプションは10個まで複数指定できます（10個を越えて指定されたディレクトリは無視されます）。その場合、指定順にインクルードファイルが検索されます。 「-I」と「*dir*」の間には空白文字があってもなくてもかまいません。
-l *list_filename*	*list_filename*で指定した名前のリストファイルを生成します。「-l」と「*list_filename*」の間には空白文字があってもなくてもかまいません。本オプションを省略すると、リストファイルは生成されません。リストファイルの詳細については付録D「アセンブルリストファイル」を参照してください。
-o *output_filename* （o は小文字）	出力ファイル名を*output_filename*にします。このオプションを省略した場合、入力ファイル名の拡張子を.moに変換したファイル名の付いたオブジェクトモジュールを生成します。入力ファイル名が拡張子を持たない場合、入力ファイル名の末尾に.moを付加したファイル名のオブジェクトモジュールを出力します。
-v	as32Rのバージョンを標準エラー出力に出力します。本オプションを指定した場合、他のオプションは無視され、アセンブル処理は行われません。
-w	ワーニングメッセージの表示を抑止します。
-D *name*[=*def*]	*def*で指定する文字列を *name* というマクロ文字変数に定義します。*def* を省略した場合は、*name*＝1 と解釈します。本オプションは、マクロ命令 .ASSIGNC と同等の機能です。

表2.4 アセンブラ起動オプション（2/2）

オプション	機能
-m32r	M32R の命令をアセンブルします。並列命令とM32Rx固有の命令は処理できません。本アセンブラで特定のCPUを指定するオプションを使用しない場合には、本オプションが指定されたものとしてアセンブルされます。
-m32rx	M32Rの命令に加えて、M32Rx で追加/変更された命令もアセンブルします。さらにM32Rxの並列命令も処理できます。MULHI 命令等のアキュムレータを指定する命令において、アキュムレータの指定を省略できます（アキュムレータ指定がなければ、本アセンブラが A0 を指定されたものとして自動的に処理します）。 本オプションについては2.2.2節も参照してください。
-m32re5	M32R/ECU#5拡張命令(M32R-FPUコアのFPU命令)を有効にします。また、浮動小数点定数で非正規化数になるものは0.0に切り詰めます。
-zdiv	M32R/ECUシリーズでの整数剰除算の問題を回避するため、DIV系命令の直後にNOP命令を挿入します。

2.2.2 M32Rx 並列命令の記述について

本アセンブラは、M32Rx の並列命令に対応しています。M32Rx の並列命令についての詳細は、「M32Rx ソフトウェアマニュアル」を参照してください。また、アセンブラでの記述については、４章の「M32Rx 並列命令」をも参照してください。

■ 並列命令記述時の注意事項

○ 並列に記述できる命令

並列に記述できる命令は、命令カテゴリの組合せに限定されています（４章「M32Rx 並列命令」を参照）それ以外の記述をした場合、アセンブラは、以下のエラーメッセージを出力して、以降の処理を行いません。

（エラーメッセージ）

```
a132R: Òxxxó, line 1: error: invalid parallel category
```

○ オペランド干渉について

並列命令の実行において、同時に同じリソース に対して書き込みが行われた場合(オペランドの干渉)、M32Rx では動作は保証されていません。オペランドの干渉は、条件ビット(C)を含む PSW、CBR といった制御レジスタにおいても同様の依存関係が成り立ち、動作は保証されていませんので、注意してください。

アセンブラはオペランド干渉をチェックする機能を備えており、それに違反すると次のエラーメッセージを出力します。

a132R: "xxx", line 1: error: write to the same destination register

オペランドの干渉については「M32Rx ソフトウェアマニュアル」を参照してください。

第３章
アセンブリ言語仕様

本章では、アセンブラが処理するM32R用アセンブリ言語の、基本的な仕様について説明します。本章で単に「ソース」とある場合はアセンブリ言語ソースプログラムを意味しています。

3.1 ソース行の基本構成

アセンブリ言語ソースプログラムは行単位で構成されます（1行につき1命令を記述します）。1行は次の様に定義されます。

■ ソースファイルの先頭から、最初の改行文字まで。
■ 改行文字の直後の文字から、次の改行文字またはソースファイルの末尾まで。

1行は改行文字で完結となります。1行の文字数に制限はありません。1行は次の4つのフィールドで構成されています。各フィールドは必要に応じて省略できます。

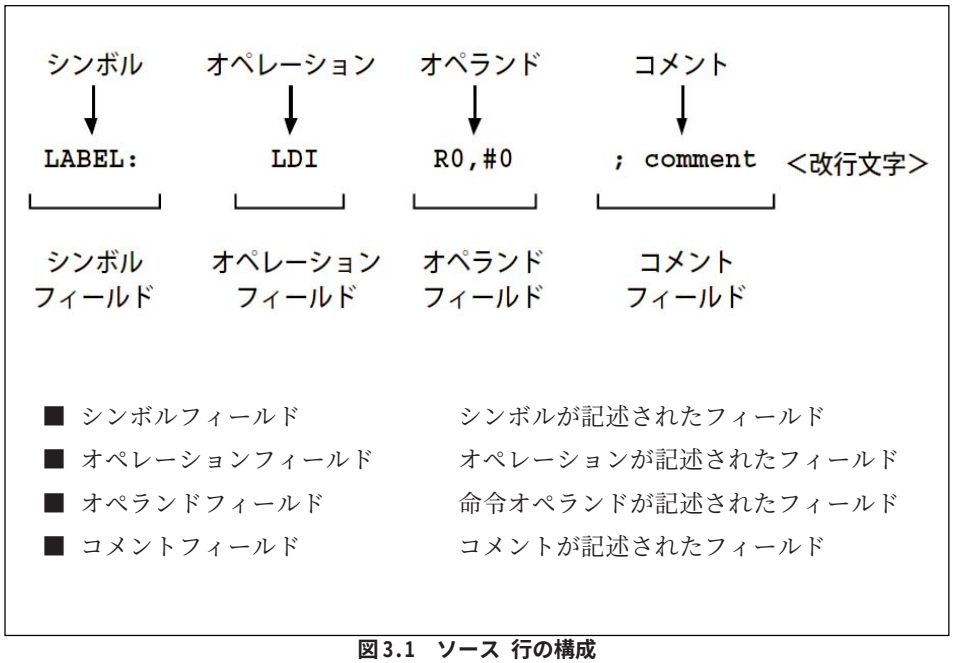


図3.1 ソース 行の構成

シンボルフィールド、オペレーションフィールド、およびオペランドフィールドの間には、それぞれ１つ以上の空白文字が必要です。

3.1.1節より、各フィールドについて説明します。なお、表記上の規則は以下のとおりです。

△　　：省略できる一文字以上の空白文字の入力（スペースおよびタブによる入力）
▲　　：省略できない一文字以上の空白文字入力（スペースおよびタブによる入力）
<CR>　：改行文字

3.1.1　シンボルフィールド

シンボルフィールドは、ソース行中でシンボル（3.6「シンボルの記述規則」参照）、またはマクロ命令のプリプロセッサ変数を記述するためのフィールドです。

通常、シンボルは第１カラム目から記述しますが、第２カラム目以降に記述することも可能です。シンボルの記述の際には以下のことに注意してください。

■ 第１カラム目からシンボルを記述する場合

以下に示す形式で記述します。

シンボル：*コメント*
シンボル　；*コメント*<CR>
シンボル<CR>
シンボル；*コメント*<CR>
↑
第１カラム目

第１カラム目からコロン（:）まで、および、第１カラム目から次に空白文字、改行文字、およびセミコロン（;）のいずれかが現れる直前までのフィールドがシンボルフィールドとなります。そのフィールド内の文字列がシンボルとなります。

第１カラム目からシンボルを記述する場合のみ、コロン（:）を省略できます（記述することもできます）。

第１カラム目から記述されたシンボルは、予約語（3.4「予約語」参照）と一致していてもシンボルと認識されます。

■ 第２カラム目以降にシンボルを記述する場合

以下に示す形式で記述します。

△***シンボル***：　　；*コメント*<CR>
△***シンボル***　　：*コメント*<CR>
↑
第１カラム目

行頭に（第１カラム目からシンボルの先頭文字の直前まで）に空白文字を挿入する場合です。この場合、シンボルの最後に必ずコロン（:）を付加してください。

第１カラム目からコロン（:）までがシンボルフィールドとなります。このうちコロン（:）と空白文字を除いた部分がシンボルとなります。

第２カラム目以降から記述されたシンボルが予約語と一致している場合はエラーとなります。

シンボルフィールドの記述例を以下に示します（図3.2。太字部分がシンボル）。

```
第１カラム目
↓
SYM1    LDI       R1,#0     …… ラベルシンボルの定義。
SYM0:   .EQU      10        …… 値シンボルの定義。
  SYM2:                     …… 2カラム目以降に記述する場合は最後に
                                コロン（:）が必須。
```

図３.２　シンボルフィールドの記述例

3.1.2　オペレーションフィールド

オペレーションフィールドは、ソース行中でオペレーションを記述するためのフィールドです。オペレーションには命令オペコードと擬似命令およびマクロ命令コードがあります。以下に、オペレーションフィールドの記述形式を示します。

▲オペレーション <CR>
▲オペレーション △;*コメント* <CR>
▲オペレーション ▲*オペランド* <CR>
▲オペレーション ▲*オペランド*△;*コメント* <CR>
シンボル：***▲オペレーション***
シンボル:***▲オペレーション*** △;*コメント* <CR>
シンボル:　***▲オペレーション*** ▲*オペランド*△;*コメント* <CR>
↑
第１カラム目

オペレーションは必ず第２カラム目以降から記述しなければなりません。オペレーションフィールドは、シンボルフィールドや後ろに続くオペランドフィールドとは、１文字以上の空白文字で区切ります。

オペレーションフィールドの記述例を以下に示します（図3.3。太字部分がオペレーション）。

```
      MV    R0,R1      ····· シンボルがなくても必ず2カラム目以降から
                             記述する。
SYM1  LDI   R1,#0      ····· シンボルフィールドとオペレーションフィールド
  ↑    ↑                     の間は空白が必要。
空白文字入力
```

図3.3　オペレーションフィールドの記述例

3.1.3　オペランドフィールド

オペランドフィールドは、ソース行中でオペランドを記述するためのフィールドです。オペランドには、各種命令のオペランドと一般命令の補正オプション（HIGH, SHIGH, LOW）があります。以下に、オペランドフィールドの記述形式を示します。

▲*オペレーション* ▲***オペランド*** <CR>
▲*オペレーション* ▲***オペランド1***, ***オペランド2***<CR>
▲*オペレーション* ▲***オペランド***△;*コメント* <CR>
シンボル: ▲*オペレーション* ▲***オペランド***△;*コメント* <CR>
↑
第１カラム目

オペランドを複数記述する場合は、オペランドどうしをコンマ（,）で区切ってください。オペレーションフィールドとオペランドフィールドの間は、１文字以上の空白文字で区切ってください。なお、オペランドを必要としない命令の行にはオペランドフィールドはありません。

オペランドフィールドの記述例を以下に示します（図3.4。太字部分がオペランド）。

```
          空白文字入力
              ↓
SYM1   LDI    R1,#0        ····· 複数のオペランドはコンマ（,）で区切る。
                                  オペレーションフィールドとオペランド
                                  フィールドの間は空白が必要。

       NOP                 ····· オペランドフィールドがない命令行もある。

       SETH   R0,#HIGH(H'ffffffff)
                           ····· 補正オプション（下線部）はオペラン
                                  フィールドで指定。
```

図3.4　オペランドフィールドの記述例

3.1.4 コメントフィールド

コメントフィールドとは、ソース行中でコメントを記述するためのフィールドです。コメントとは、アセンブル対象外の、ユーザーの任意の情報の記述です。以下に、コメントフィールドの記述形式を示します。

△；***コメント*** <CR>
コメント以外のフィールド△；***コメント*** <CR>
↑
第１カラム目

コメントの記述は必ずセミコロン（;）から始めます。アセンブラはセミコロン（;）から改行文字の直前までをコメントフィールドと認識します。ただし、ダブルクォーテーション（"）で囲まれた文字列中のセミコロン（;）はコメントフィールドの先頭文字とは見なしません。

コメントフィールドはどの行でも記述可能です（省略も可能）。コメントフィールドでは改行文字を除く文字セットの文字がすべて使用できます。

コメントフィールドの記述例を以下に示します（図3.5。太字部分がコメント）。

```
LDI   R0,#10  ;Loads 10 into R0
;LDI R0,#10   ;Loads 10 into R0 …… 行頭に；があれば行全体が
                                    コメントとなる。
;comment
```

図3.5 コメントフィールドの記述例

マクロ命令 .MACRO～ .ENDM によるマクロ定義内には、マクロ展開の対象にならないコメントが記述できます。詳しくは第６章「マクロ命令の記述方法」を参照してください。

3.2 ソース行の種類

ソース行には以下の種類があります。

■ 一般命令行　　M32R 用命令が記述されている行。この行はアセンブラによって M32R 対応のオブジェクトコードに変換されます。

■ 擬似命令行　　アセンブラ用の擬似命令が記述されている行。アセンブラにアセンブル処理の諸指示を行うものです。

■ マクロ命令行　　マクロ命令によってマクロ定義を記述する行。

■ コメント行　　コメントだけが記述されている行。この行はアセンブル処理の対象外となります。

■ 空白行　　何も記述されていない行です（改行文字入力のみの行)。この行はコメント行同様、アセンブル処理の対象外となります。

■ シンボル行　　シンボルのみが記述されている行（シンボルフィールドのみ、または、シンボルフィールドとコメントフィールドのみからなる行)。シンボルにはその行のロケーションカウンタが設定されます。

3.3 使用できる文字

アセンブリ言語ソースプログラム中で使用可能な文字を以下に示します。

表3.1 文字セット（1/2）

分類	文字	ASCIIコード	名称（備考）
英字	A～Z	H'41～H'5A	英大文字
	a～z	H'61～H'7A	英小文字
数字	0～9	H'30～H'39	数字
英数字	英字と数字をあわせた呼称		
特殊文字	"	H'22	ダブルクォーテーション（各記号の特別な意味を示す）
	#	H'23	シャープ
	$	H'24	ドルマーク（ シンボルに使用可能）
	&	H'26	アンパサンド
	'	H'27	シングルクォーテーション
	(	H'28	左カッコ
	)	H'29	右カッコ
	*	H'2A	アスタリスク
	+	H'2B	プラス
	,	H'2C	コンマ
	-	H'2D	マイナス
	.	H'2E	ピリオド
	/	H'2F	スラッシュ
	:	H'3A	コロン
	;	H'3B	セミコロン
	<	H'3C	不等号（小なり）
	=	H'3D	等号
	>	H'3E	不等号（大なり）
	@	H'40	アットマーク
	\	H'5C	円マークまたはバックスラッシュ
	_	H'5F	アンダーライン （シンボルに使用可能）
	|	H'7C	論理和
	~	H'7E	チルダ
空白文字	(SP)	H'20	スペース
	(HT)	H'09	タブ

表3.1　文字セット（2/2）

分類	文字	ASCIIコード	名称（備考）
改行文字	(CR)	H'0D	改行文字（キャリッジリターン）
	(LF)	H'0A	改行文字（ラインフィード）
	(FF)	H'0C	改行文字（フィードフォワード）
その他	上記以外で計算機で使用できる文字がある場合はコメントでのみ使用可能。		

3.4 予約語

アセンブラは以下の識別子を予約語と解釈します。これらには英大文字／英小文字による区別はありません。

- レジスタ名
- 特殊シンボル
- ニーモニック

3.4.1 レジスタ名

レジスタ名とはM32Rファミリのレジスタを表す予約語で、以下のものがあります。

■ 汎用レジスタ名　　Rx（R0～R15）　SP

※ R15（スタックポインタ）は、R15またはSPのどちらでも指定できます。

■ 制御レジスタ名　　CRx（CR0～CR15）PSW　CBR　SPI　SPU　BPC

※ これらは一般命令MVFC、MVTCのオペランドでのみ使用します。

■ アキュムレータ名
（ M32Rx の場合 ）　　A0　A1

※ アキュムレータは乗算命令「MUL」でも使用され、この命令の実行の際はアキュムレータA0、A1 の値が破壊されるので注意してください。

※ M32Rx仕様拡張命令　MVTACHI、MVTACLO、MVFACHI、MVFACLO、MVFACMI　のオペランドでのみ使用します。

3.4.2 特殊シンボル

特殊シンボルとは、オペランドで指定する予約シンボルで、以下のものがあります。

SIZEOF　SHIGH　HIGH　LOW

3.4.3 ニーモニック

ニーモニックとは、一般命令、擬似命令、またはマクロ命令の、命令を表す予約語です（LD、.PROGRAMなど）。

■ 一般命令ニーモニック

LD　ST　MV　ADD　など

■ 擬似命令ニーモニック

.ALIGN　.PROGRAM　.SECTION　.END
.EXPORT　.IMPORT　.GLOBAL　.EQU
.ASSIGN　.DATA　.DATAB　.SDATA
.SDATAB　.RES

■ マクロ命令ニーモニック

.AIF　.AELSE　.AENDI　.AREPEAT　.AENDR
.ASSINGA　.ASSIGNC　.AWHILE　.AENDW　.EXITM
.INCLUDE　.INSTR　.LEN　.SUBSTR　.MACRO
.ENDM

3.5 名前の記述規則

名前とは、以下の名前を表す文字列です。

■ ユーザー定義できる名前

- モジュール名（.PROGRAM 擬似命令で指定。予約語も使用可能）
- シンボル名（予約語は使用不可。3.7「シンボルの記述規則」参照）
- セクション名（.SECTION 擬似命令で指定。予約語は使用不可）
- プリプロセッサ変数
- マクロ名

■ ユーザー定義できない名前

- 予約語（レジスタ名、特殊シンボル、ニーモニック）

名前の記述規則を以下に示します。

■ 1 文字目に使用できる文字

英字、ドルマーク（$)、アンダーライン（_）のいずれか。
先頭文字に数字は使用できません。

■ 2 文字目以降に使用できる文字

英数字、ドルマーク（$)、アンダーライン（_）のいずれか。

■ 記述できる文字数

・ モジュール名	206 文字まで
・ シンボル名	243 文字まで
・ セクション名	243 文字まで
・ プリプロセッサ変数、マクロ名	それぞれ 32 文字まで

■ 英大文字／英小文字の区別

ユーザー定義できる名前の場合は区別あり。
ユーザー定義できない名前の場合は区別なし。

ユーザーが名前を定義する場合は上記の規則に従って記述します。なお、その際以下の点に注意してください。

■ モジュール名以外のものに、予約語と同じ名前は使用できません。
■ シンボルとセクション名など、ユーザー定義の名前は重複してはいけません。

3.6 シンボルの記述規則

シンボルとは、アドレスやユーザーが指定する式の値を、シンボル名に置き換えたものです。シンボルには以下のものがあります。

■ 値シンボル　　式の値を割り当てたシンボルです。擬似命令 .EQU、.ASSIGN で定義します。

擬似命令 .ASSIGN で定義した値シンボルは、擬似命令 .ASSIGN で再定義できます。

■ ラベルシンボル　　シンボルを宣言した行のロケーションカウンタ値（アドレス値）を割り当てたシンボルです。絶対形式セクション内のラベルシンボルには絶対アドレス値、相対形式セクション内のラベルシンボルには相対アドレス値が割り当てられます。

シンボルの記述規則を以下に示します（シンボル名の付け方は、名前の記述規則に従ってください）。

■ 記述フィールド　　シンボルフィールドまたはオペランドフィールド。

■ 定義方法　　シンボルを定義するには、シンボルをシンボルフィールドに記述します。定義行のオペレーションフィールドに擬似命令 .EQU、.ASSIGN があれば、オペランドで指定する値が割り当てられます（値シンボル）。それ以外の場合は、定義行に対応するロケーションカウンタ値が割り当てられます（ラベルシンボル）。

■ 参照方法　　定義したシンボルは命令のオペランドで参照できます。オペランドの式の中で、シンボルを使ってアドレス値や即値を指定できます。

シンボルの定義例と参照例を以下に示します（図3.6）。

```
            .SECTION  program
;
VAL_SYM0:   .EQU      10              ;値シンボルの定義
VAL_SYM1:   .ASSIGN   20              ;値シンボルの定義
VAL_SYM1:   .ASSIGN   30              ;.ASSIGNで定義したシンボルの再定義
            SETH      R0,#VAL_SYM0    ;値シンボルを参照
            SETH      R1,#VAL_SYM1    ;値シンボルを参照
;
LABEL0:                               ;命令のない行でラベルシンボルを定義
LABEL1:     MV        R5,R0           ;命令行でラベルシンボルを定義
            BL        LABEL0          ;ラベルシンボルの参照
;
            .END
```

図3.6　シンボルの定義と参照

3.7 プリプロセッサ変数の記述規則

プリプロセッサ変数はマクロ命令のオペランドとして使用します。それ以外の命令では使用できません。プリプロセッサ変数には、算術変数と文字変数の２種類があります。

■ 算術変数　　マクロ命令 .ASSIGNA のオペランドで指定した算術式の値を割り当てた変数です。算術式については、第６章「マクロ命令行の記述方法」を参照してください。

■ 文字変数　　マクロ命令 .ASSIGNC のオペランドで指定した文字式を割り当てた変数です。文字式については、第６章「マクロ命令行の記述方法」を参照してください。変数の宣言例と参照例を図3.7に示します。

```
          .SECTION    program
;
V_VAL:    .ASSIGNA    10                ;算術変数を定義します。
;
          .AREPEAT    \&V_VAL           ;算術変数を参照します。
          NOP
          .AENDR
;
C_VAL:    .ASSIGNC    "ABC"             ;文字変数を定義します。
;
          .AIF        \&C_VAL EQ "ABC" ;文字変数を参照します。
          MV          R0,R2
          .AELSE
          MV          R0,R3
          .AENDI
;
          .END
```

図3.7　プリプロセッサ変数の宣言と参照

注意

プリプロセッサ変数をオペランドで参照する場合、名前の先頭に「 \& 」を付加してください。

3.8 式の記述規則

式とは、即値、相対アドレス値、または、絶対アドレス値を表す表現です。式は、代数規則に従って、1つまたは複数の項を演算子で結合したものです。以下に式を構成する項と演算子を示します。

■ 項　　定数、シンボル名、セクション名
(定数どうし、または、定数が割り付けられたシンボルどうしの演算からなる式をとくに定数式といいます)。

■ 演算子　　算術演算子、論理演算子、シフト演算子

以下に式の記述規則を記します。

■ 記述フィールド　　式はオペランドフィールドに記述します。

■ データタイプ　　アセンブラは、式の値を 32 ビット符号付き整数と見なします。

■ 制限事項　　相対値（相対形式セクション内で定義したラベルシンボル）および外部参照シンボル（.IMPORT 擬似命令で定義されたシンボル）は、乗除算、シフト演算、および論理演算の項として使用できません。

演算結果については以下のことに注意してください。

■ 同一セクションの相対値どうしの減算結果は絶対値となります。ただし、相対値どうしがセクション名の場合は、減算結果は相対値になります。

■ 演算の結果オーバーフローが生じた場合、オーバーフローしたこと自体は無視されます。ただし、演算結果が各種命令で許されるデータサイズを超えた場合は、その時点でエラーとなります。

■ オーバーフローによって演算結果が意味を持たなくなった場合でも、演算結果（32 ビット範囲内）が使用されます。たとえば LOW 補正オプションを使用した以下の例のような場合、アセンブラ実行時にエラーとはなりません。

例：
```
DATA1:    .equ  H'7FFFFFFF
          LD    R0,@( LOW(DATA1+DATA1+DATA1), R1)
```

上記例の場合、(DATA1+DATA1+DATA1) の演算結果は 32 ビットの範囲をオーバーフローします。しかし、LOW 補正オプションによって計算結果の下位 16 ビットだけが処理対象となるため、エラーとはなりません。これに対し、以下の例のような

場合はエラーとなります。

例：
```
DATA2:    .equ  H'7FFF
          LD    R0,@(DATA2+DATA2, R1)
```

上記の場合にはLOW補正オプションがなく、DATA2+DATA2 の値がLD命令の許す16ビットのディスプレースメントを超えるためにエラーとなります。

||||注意||||

マクロ命令で使用する式は、6.3.2「式（マクロ命令用）」の内容に従ってください。

3.8.1 式に記述できる定数

式で記述できる定数の表現方法を表3.2に示します。

表3.2 定数の種類

種類	記述例	記述規則
2進数	B'10010001	接頭子 ：B'またはb'
		使用可能数字 ： 1または0
8進数	Q'6072	接頭子 ： Q'（大文字キュー）またはq'（小文字キュー）
		使用可能数字 ： 0～7
	01234	先頭が0（ゼロ）で、0～7で構成される定数も8進数とみなされる。
10進数	D'9423	接頭子 ：D'またはd'
		使用可能数字 ： 0～9
	1234	先頭は0（ゼロ）以外で、0～9で構成される定数も10進数とみなされる。
16進数	H'A05	接頭子 ：H'、h'、0X（ゼロ大文字X）、または0x（ゼロ小文字x）
	0XaA84	使用可能数字 ： 0～9, a～f, A～F
文字定数	"CNST"	・ASCII文字列をダブルクォーテーション（"）で囲んで記述する。 ・文字列は最大4文字まで（32ビット長）指定可能。 ・文字定数はASCIIコードを表す。 たとえば、"ABC" という文字定数は、H'414243を指定する定数。 ・ダブルクォーテーションを文字として書く場合、 ダブルクォーテーションを2つ重ねて書く("")ことで、1つのダブルクォーテーションを表す。

||||注意||||

負数の取り扱いにおける注意事項： 本アセンブラでは、式の評価時に、2の補数表現を負数として取り扱いません。したがって、以下に示す値は、それぞれ異なる値として扱われます。

```
-1
H'FFFF FFFF
```

3.8.2 シンボル名による値の指定

値シンボルでは即値、ラベルシンボルではロケーションカウンタ値（相対アドレス値または絶対アドレス値）が指定できます。式中にシンボルがある場合、アセンブラはそのシンボルに定義されている値を参照します。シンボルの参照について詳しくは、3.6「シンボルの記述規則」を参照してください。

3.8.3 セクション名による値の指定

セクション名は、そのセクションの先頭アドレスを示します。相対形式セクションのセクション名は、リンク後の同一セクション全体の先頭アドレスを表します。絶対形式セクションのセクション名は、同一ソースファイル中の１番目に記述したセクションの先頭アドレスを表します。また、

SIZEOF(*セクション名*)

あるいは、

sizeof *セクション名*

と表記することでリンク後のセクション全体の大きさを表すことができます

3.8.4 演算子

式の演算子には、以下の算術演算子、論理演算子、およびシフト演算子があります。

■ 算術演算子

表3.3 算術演算子の種類

演算子	名称	例
+	単項正	+ 6
-	単項負	-7
+	二項加算	9 + 6
-	二項減算	8 - 3
*	二項乗算	4 * 7
/	二項除算	5 / 2
%	二項剰余	17 % 5

■ 論理演算子

表3.4　論理演算子の種類

演算子	名称	例
~	単項論理否定	~1
&	二項論理積	H'F3 & H'31
|	二項論理和	H'FFFF | H'1356
~	二項排他的論理和	B'1111 ~B'0110

※~（チルダ）は、単項論理否定および二項排他的論理和の両方に使用されます。

■ シフト演算子

表3.5　シフト演算子

演算子	名称	例
<<	二項左シフト演算子	0x400 << 2
>>	二項右シフト演算子	0x800 >> 1 通常、二項右シフト演算は論理シフトで 、左項が明示的に負の場合に限り算術右シフトとなる。

式の演算子の優先度は以下のとおりです。

表3.6　演算子の優先順位

優先順位	演算子	名称
1	()	カッコ
2	＋、－、~	単項正、単項負、単項論理否定
3	＊、/、%	二項乗算、二項除算、二項剰余
4	＋、－	二項加算、二項減算
5	<<、>>	左シフト、右シフト
5	&	二項論理積
6	|	二項論理和
7	~	二項排他的論理和

第４章
一般命令行の記述方法

本章では、一般命令（M32R ファミリ用アセンブリ言語命令）を記述する、一般命令行の記述方法について説明します。

4.1 一般命令の種類

M32R の一般命令は、以下の６つの機能グループに分類できます。付録 B に、機能グループ別に各命令の概要を示します。各命令の詳細については、M32R ソフトウェアマニュアルを参照してください。

- ロード／ストア命令
- 転送命令
- 演算命令（比較、算術演算、論理演算、シフト）
- 分岐命令
- EIT 関連命令　　※ EIT :Exception,Interrupt,Trap
- DSP 機能用命令

4.2 一般命令行の構成

一般命令行では、オペレーションフィールドにM32Rの一般命令ニーモニックを、オペランドフィールドにそのオペランドを記述します。

一般命令行の構成と記述例を以下に示します。

■ 一般命令行の構成

▲*一般命令ニーモニック* ▲[*オペランド*[,*オペランド...*]] <CR>
△*シンボル*:▲*一般命令ニーモニック* ▲[*オペランド*[,*オペランド...*]] <CR>

※ [ ]で囲まれた項目　　省略可能な記述
※ ▲　　省略不可能な空白
※ △　　省略可能な空白
※ <CR>　　改行文字入力

■ 記述例

```
LABEL0:  LD       R1,@R1          ; memory to register
         LD24     R0,#h'FF0000
         ST       R1,@R0
         JMP      R14
```

↑ シンボル　↑ 一般命令ニーモニック　↑ オペランド　↑ コメント

注意

行の第１カラム目から命令ニーモニックを記述することはできません。行頭がニーモニックの場合は、必ずその前に空白文字またはタブを入れて下さい。

4.3 一般命令オペランド文法

一般命令のオペランドは以下のような文法で記述できます。

■ オペランド（imm は整数即値、label はラベルを示します）

```
Rn | CRn | @Rn | @(Rn) | @(disp,Rn) |
#imm[:8|:16] | #imm[:24] |
@+Rn | @-Rn | @Rn+ | @Rn+ |
label[:8|:24] | label[:16]
```

■ Rn（汎用レジスタ名）

```
R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 | R10 |
R11 | R12 | R13 | R14 | R15 | SP
```

■ CRn（制御レジスタ名）

```
CR0 | CR1 | CR2 | CR3 | CR4 | CR5 | CR6 | CR7 | CR8 |
CR9 | CR10 | CR11 | CR12 | CR13 | CR14 | CR15 |
CBR | SPI |  SPU | PSW
```

■ An（アキュムレータ名、M32Rx の場合）

```
A0 | A1
```

命令によっては、オペランドの整数即値に対して補正オプション（HIGH、SHIGH、LOW）を指定できる場合があります。各命令におけるオペランドの記述形式は M32R のソフトウェアマニュアルで確認してください。

4.4 演算サイズ指定

同じ演算を行う命令でも、演算サイズ（処理対象データのサイズ）に応じてバリエーションがあります。取り扱うデータの演算サイズに適した命令を選択してください。

例： レジスタからメモリへのデータ転送命令のバリエーション

ST 命令　　レジスタからメモリへの32bitデータ転送
STB 命令　　レジスタからメモリへの8bitデータ転送
STH 命令　　レジスタからメモリへの16bitデータ転送

整数即値の場合、演算サイズは指定不要です。アセンブラが、指定した値を表現できる最小のサイズ（8 または 16 ビットのいずれか）を自動選択します。ただし、指定された値を表現できない場合はエラーとなります。

例： `LDI  Rdst, #imm`

即値 imm は自動的に、8 ビット以内で表現できる場合は 8 ビットデータ、
8 ビットより多くのビットが必要な場合は 16 ビットデータとして、
それぞれ処理されます。

4.5 補正オプションの記述方法

M32R の命令セットでは 1 命令で扱える即値は 24 ビットまでです。24 ビットを超える即値を操作するには、上位 16 ビット と下位 16 ビットの 2 命令 に分けて取り扱います。アセンブラには、この操作を記述するための特殊シンボルとして補正オプション HIGH、SHIGH、LOW が用意されています。補正オプションを使用すると、32 ビット即値から必要とする 16 ビット即値を簡潔に得られます。

補正オプションが使える命令は以下のとおりです。

■ HIGH、SHIGH オプションが使用できる命令　　SETH
■ LOW オプションが使用できる命令　　LD　LDB　LDH　LDUB　LDUH
ST　STB　STH　OR3

表4.1に補正オプションの記述規則を示します。

表4.1 補正オプションの記述形式

補正オプション	記述形式と説明
HIGH	**SETH Rdst,#HIGH(imm)** 即値immの上位 16ビットのみがSETH命令の即値オペランドとして扱われます。SETH命令の実行によって、Rdstにはimmの下位16ビットを0にした値が転送されます。
SHIGH	**SETH Rdst,#SHIGH(imm)** 即値immの下位 16ビットの符号拡張を補正する値を加えた、immの上位16ビットをSETH命令の即値オペランドとします。
LOW	**OR3 Rdst,Rsrc,#LOW(imm)** 即値immの下位16ビットのみがOR3命令の即値オペランドとして扱われます。この命令より前に、SETH Rsrc,#HIGH(imm) を記述することで、Rdstに32ビットの即値が設定できます。 ***mnemonic* Rdst,@(LOW(imm),Rsrc)** （***mnemonic*** ：LD|LDB|LDH|LDUB|LDUH|ST|STB|STH） immの下位16ビットが***mnemonic*** 命令のディスプレースメントとして扱われます。このディスプレースメントは符号拡張されます。したがって、この命令により前に、 SETH Rsrc,#SHIGH(imm) を記述することでRdstにアドレスimmのメモリ内容が転送されます。

以下に補正オプションの記述例を示します（例１～例６）。

例１：
```
SETH    Rdst, #HIGH(imm)
```
即値immの上位 16bitを、Rdst の上位16bit に格納します。Rdst の下位16bitは0 になります。

例２：
```
SETH    Rdst, #SHIGH(imm)
```
即値immの上位16bit をRdst の上位16bit に格納します。Rdst の下位16bit は0 になります。このとき、Rdst に #immの下位16bit を符号つき整数として加算した結果が#imm になるように、Rdstは補正されます。

例３：
```
LD      Rdst, @(LOW(disp), Rsrc)
```
ディスプレースメントdispの下位16bitを32bitに 符号拡張し、その32bit値をRsrcの内容に加算します。加算後の値（ アドレス）が示すメモリから32bitのデータを読み出し、Rdst に格納します。この場合、dispの下位16bitのみを用いるので、ディスプレースメントの値は範囲外にはなりません。

例４：32bit 即値をレジスタに格納する場合
```
SETH    Rdst, #HIGH(imm_32)
OR3     Rdst, Rdst, #LOW(imm_32)
```
※OR3 命令では#LOW(imm_32) はゼロ拡張されます。

例５：　レジスタに32bitで示されるアドレスのメモリの内容をロードする場合

```
SETH    Rwork, #SHIGH(imm_32)
LD      Rdst, @(LOW(imm_32), Rwork)
```

※LD 命令ではLOW(imm_32) は符号拡張されます。

［補足］　SHIGH 補正オプションについて

SHIGH 補正オプションは、SETH 命令で 32 ビットのアドレス値の上位ハーフワードを設定する場合に使用します。以下の例 6 において、仮に imm_32 の値がアドレス H'**FC00**8000 であるとします。

例６：
```
SETH  R1,#SHIGH(imm_32)
LD    R2,@(LOW(imm_32),R1)
```

上記2行目のLD命令のディスプレースメントは、LOW(imm_32)=H'FFFF8000　のように符号拡張されます。したがって、もしHIGH補正オプションを使うと、次に示すようにimm_32で表されたアドレスが正しく得られないことになります。

```
R1 = #HIGH(imm_32) = H'FC000000
#LOW(imm_32)+R1 = H'FFFF8000+H'FC000000
                = H'FBFF8000
```

そこでSHIGH補正オプションを次のように定義します。

■ imm_32 の下位ハーフワードの最上位ビットが1（負）のとき

```
#SHIGH(imm_32) = #HIGH(imm_32)+1
```

■ imm_32 の下位ハーフワードの最上位ビットが0（正）のとき

```
#SHIGH(imm_32) = #HIGH(imm_32)
```

したがって例6の場合、#SHIGH(imm_32)=H'FC01 から、以下のようにSHIGH補正オプションを使用することで正しいアドレスが得られます。

```
R1 = #SHIGH(imm_32) = H'FC010000
LOW(imm_32)+R1 = H'FFFF8000+H'FC010000
               = H'FC008000
```

4.6 アドレッシングモードの種類と選択

M32R ファミリは、処理対象データを指定する方法として、以下のようなアドレッシングモードをサポートしています。

- レジスタ直接
- レジスタ間接
- レジスタ相対間接
- レジスタ間接＋レジスタ更新
  （プリインクリメント付きレジスタ間接、プリデクリメント付きレジスタ間接、ポストインクリメント付きレジスタ間接）
- 整数即値
- PC 相対

一般命令のオペランドを記述する場合、これらのアドレッシングモードのうちのいずれかを使用します。なお、使用可能なアドレッシングモードは命令によって異なるので、M32R の ソフトウェアマニュアルで確認してください。

表4.2に、各アドレッシングモードにおけるオペランドの表記方法と、その表記によって処理対象となるデータについて示します。

表4.2 アドレッシングモード別オペランド表記と意味

アドレッシングモード	オペランド表記	処理対象
レジスタ直接	Rn または CRn または An	汎用レジスタRnの内容、または制御レジスタCRnの内容、 または、アキュムレータの内容
レジスタ間接	@Rn	「汎用レジスタRnの内容」番地のメモリの内容
レジスタ相対間接	@(disp,Rn)	「汎用レジスタRnの内容＋ディスプレースメント」番地のメモリの内容
整数即値	#imm[:8|:16] または #imm[:24]	式の値
プリインクリメント付きレジスタ間接	@+Rn	「汎用レジスタRnの内容＋4」番地のメモリの内容
プリデクリメント付きレジスタ間接	@-Rn	「汎用レジスタRnの内容－4」番地のメモリの内容
ポストインクリメント付きレジスタ間接	@Rn+	「汎用レジスタRnの内容」番地のメモリの 内容。 Rnの内容は、参照後に4加算して更新される。
PC相対	label[:8|:24] または label[:16]	命令の置かれた番地と式（label）の指す番地との差。 通常アセンブラでは分岐先に置いたラベルを記述する。

※ Rn は任意の汎用レジスタ、CRn は任意の制御レジスタ、An はアキュムレータ disp はディスプレースメント値、imm は整数即値を意味します。

4.7 オペランドの記述方法（アドレッシングモード別）

4.7.1 レジスタ直接

処理対象　指定したレジスタの値

記述方法　`Rn`　または、`CRn`　または 、`An`

Rn　：汎用レジスタ（R0 ～ R15、SP）
CRn　：制御レジスタ（CR0 ～ CR15）
An　：アキュムレータ（A0、A1）

- 汎用レジスタ名、制御レジスタ名または、アキュムレータ名を記述します。制御レジスタはMVTC およびMVFC 命令のオペランドにのみ記述できます。また、アキュムレータは、MVTACHI、MVTACLO、MVFACHI、MVFACLO、MVFACMI のオペランドにのみ記述できます。
- 汎用レジスタ名のR15 はSP とも記述できます。制御レジスタは、PSW などの名称でも記述できます。レジスタの対応はM32R のソフトウェアマニュアルを参照してください。
- このオペランド記述によって、レジスタ、アキュムレータの内容が参照されます。

例　R5

4.7.2 レジスタ間接

処理対象　指定したレジスタの値（アドレス）が示すメモリの内容

記述方法　`@Rn`

Rn　：汎用レジスタ（R0 ～ R15、SP）

- @ に続けて、汎用レジスタ名を記述します。
- このオペランド記述によって、汎用レジスタ値（アドレス）が示すメモリの内容が参照されます。

例　@R5

4.7.3 レジスタ相対間接

処理対象　指定したレジスタの値（アドレス）にディスプレースメントを加算し、その加算した結果（アドレス）が示すメモリの内容

記述方法

```
@(disp,Rn)
```

disp　：ディスプレースメント
Rn　：汎用レジスタ（R0～R15、SP）

- @に続けて、ディスプレースメント、コンマ（,）、汎用レジスタ名の順に記述します。
- ディスプレースメントには式を記述します。
- このオペランド記述によって、指定したレジスタの値（アドレス）にディスプレースメントを加算し、その加算した結果（アドレス）が示すメモリの内容が参照されます。

例

```
@(data,R7)
@(label,R8)
@(-4, R9)
@(H'FEDC,R5)
```

4.7.4 整数即値（immediate）

処理対象　imm（式）の値

記述方法

#imm[:8|:16]	または、	#imm[:24]

imm　　：式
:8|:16　：式のサイズ（式を表現できるビット数のうち小さい方）
:24　　：式のサイズ

- # に続けて式 imm を記述します。
- 1つの命令で2つにディスプレースメントサイズが選択可能な場合、式の後に :8 または :16 を付記することで、そのどちらかを指定できます。その指定が無い場合はアセンブラが自動的に選択します。
- :24 は LD24 命令でのみ使用可能です。
- トラップ命令、シフト命令の場合、imm には以下の制限があります。
  制限：　式に外部参照シンボルを含んではいけない。
  式の値は絶対値であること。
  式に前方参照の ASSIGN 擬似命令で定義したシンボルを含んではいけない。
  式の範囲に制限がある。TRAP 命令は 0 ～ 15、シフト命令は 0 ～ 31。
- このオペランド記述によって、式の値が参照されます。

例

```
#B'1010
#(symbol + H'1)
#-7
```

4.7.5 プリインクリメント付きレジスタ間接

処理対象 汎用レジスタ値にオペランドサイズ分の値を加えた後の、汎用レジスタ値（アドレス）の示すメモリの内容

記述方法

```
@+Rn
```

Rn ：汎用レジスタ（R0 ～ R15、SP）

- @ に続けて、プラス（+）、汎用レジスタ名の順に記述します。
- このオペランド記述によって、汎用レジスタの値にオペランドサイズ分の値（4）を加えた後、汎用レジスタの値（アドレス）が示すメモリの内容が参照されます。

例 `@+R11`

4.7.6 プリデクリメント付きレジスタ間接

処理対象 汎用レジスタ値からオペランドサイズ分の値を引いた後の、汎用レジスタ値（アドレス）の示すメモリの内容

記述方法

```
@-Rn
```

Rn ：汎用レジスタ（R0 ～ R15、SP）

- @ に続けて、マイナス（-）、汎用レジスタ名の順に記述します。
- このオペランド記述によって、汎用レジスタの値からオペランドサイズ分の値（4）を引いた後、汎用レジスタの値（アドレス）が示すメモリの内容が参照されます。

例 `@-R13`

4.7.7 ポストインクリメント付きレジスタ間接

処理対象　汎用レジスタの示すアドレスの内容

記述方法

```
@Rn+
```

Rn　：汎用レジスタ（R0 ～ R15、SP）

- @ に続けて、汎用レジスタ名、プラス（+）の順に記述します。
- このオペランド記述によって、汎用レジスタの値（アドレス）の示すメモリの内容が参照されます。参照後、汎用レジスタ値はオペランドサイズ分の値（4）を加算した値に更新されます。

例

```
@R1+
```

4.7.8 PC相対

処理対象　命令の置かれたアドレスと分岐先アドレスとの差（ディスプレースメントサイズ）。

記述方法

```
label[:8|:24]
```

または、

```
label[:16]
```

label　：行き先ラベル
:8|:24　：ディスプレースメントサイズ
:16　：ディスプレースメントサイズ

- label は PC 相対アドレッシングモードの分岐命令で使用されます。
- ディスプレースメントには、命令により 8、16、および 24 ビットのサイズがあります。
- ディスプレースメントは、32 ビットに符号拡張したあと、左へ 2 ビットシフトした値が使用されます。ただし、通常アセンブラレベルで記述する場合、ラベルシンボルを記述すれば、ディスプレースメントは自動的に計算されます。また、式を記述すれば、その値を分岐先アドレスとして、現在の PC から式の表すアドレスまでのディスプレースメントが自動的に計算されます。
- 1 つの命令で 2 つのディスプレースメントサイズが選択可能な場合、式の後に :8 または :24 を付記することで、そのどちらかを指定できます。その指定が無い場合はアセンブラが自動的に選択します。

例

```
BL  DstSymbol:8
BEQ R1,R2, 1000
```

4.8 M32Rx 並列命令の記述について

本アセンブラは、M32Rx の並列命令に対応しています。M32Rx の並列命令についての詳細は、「M32Rx ソフトウェアマニュアル」を参照してください。

■ アセンブラで並列命令を記述するには

アセンブラで、並列命令を記述するには、並列処理を行いたい命令の間に並列指定記号 '||' を指定します。(ラベルは行の先頭に記述できます。 '||' と命令Ｂの間には、ラベルを記述することはできません)

(例 1)　ラベル:　命令 A　||　命令 B

並列指定記号は、アセンブラに対する並列処理の指示であって、M32Rx の パイプラインを特に指定するものではありません。どちらの命令を O パイプ、S パイプで実行させるのかは、本アセンブラが自動的に決定します。(例 2) の記述は、(例 1) の命令記述を左右逆にしたものですが、(例 1) と同じ動作になります。

(例 2)　ラベル:　命令B　||　命令A

並列に記述できる命令は、次の４通りの命令カテゴリの組合せに限定されています。(「M32Rx ソフトウェアマニュアル」を参照)

○ 左側命令と右側命令　(O-,-S)
○ 左側命令と両側命令　(O-,OS)
○ 両側命令と右側命令　(OS,-S)
○ 両側命令と両側命令　(OS,OS)

これら以外の記述をした場合、アセンブラは、以下のエラーメッセージを出力して、以降の処理を行いません。

(エラーメッセージ)

```
a132R: "xxx", line 1: error: invalid parallel category
```

図 4.1　に並列命令の記述例（アセンブルソース）とそのアセンブル結果（アセンブルリスト）を示します。

```
$ type sample.ms

        .section P,code,align=4
label:  MACHI R0,R3,A0  || LD R2,@R4
        LDI R9,#10 || OR R1,R2
        .end
```

アセンブルソース

```
$ as32R -m32rx -l sample.lis -o sample.mo sample.ms
$typesample.lis
```

アセンブルリスト

```
* ASSEMBLER * SOURCE LIST *

 LST#   SRC# LOCATION OBJ_CODE                SOURCE_STATEMENT

[sample.ms]
   1    1                  .section P,code,align=4
   2     2                 label: MACHI R0,R3,A0  || LD R2,@R4
   3     2 00000000 22C4
   4     2 00000002 B043
   5     3                    LDI R9,#10 || OR R1,R2
   6     3 00000004 690A
   7     3 00000006 81E2
   8    4                  .end
```

2 行目の LD 命令が、コード 22C4 となってアドレス 00000000 に配置されます。
さらに、MACHI 命令がコード B043 となってアドレス 00000002 に配置されます。
MACHI 命令の最上位ビット(MSB)がセットされているので LD 命令と並列実行されます。
3 行目の LDI 命令が、コード 690A となってアドレス 00000004 に配置されます。
さらに、OR 命令がコード 81E2 となってアドレス 00000006 に配置されます。
OR 命令の最上位ビット(MSB)がセットされているので LDI 命令と並列実行されます。

図4.1 並列命令の記述例

第５章
擬似命令行の記述方法

本章では、アセンブラの擬似命令とそのオペランドの記述方法について説明します。

5.1 擬似命令の種類

擬似命令はアセンブラに指示を与えるための命令です。アセンブラには以下の擬似命令が用意されています。

表5.1 アセンブラの擬似命令の種類

グループ	擬似命令	機能
アドレス制御命令	.ALIGN	境界の調整
プログラム構造定義命令	.PROGRAM	モジュール名の指定
	.SECTION	セクションの宣言
	.END	ソースプログラムの終了
シンボル外部定義/外部参照命令	.EXPORT	外部定義シンボルの宣言
	.IMPORT	外部参照シンボルの宣言
	.GLOBAL	外部定義/外部参照シンボルの宣言
シンボル定義命令	.EQU	値シンボルの宣言
	.ASSIGN	値シンボルの宣言(再設定可能)
データ設定命令	.DATA	整数データの設定
	.DATAB	整数データの設定（データブロック）
	.SDATA	文字列データの設定
	.SDATAB	文字列データの設定（データブロック）
領域確保命令	.RES	整数データ領域の確保

各擬似命令の詳細は付録Ｂ「擬似命令リファレンス」を参照してください。

5.2 擬似命令行の構成

擬似命令行では、オペレーションフィールドにアセンブラ擬似命令ニーモニックを、オペランドフィールドにそのオペランドを記述します。

擬似命令行の構成と記述例を以下に示します。

■ 擬似命令行の構成

△*擬似命令ニーモニック* ▲[*オペランド*[,*オペランド*...]] <CR>
△*シンボル*:▲*擬似命令ニーモニック* ▲[*オペランド*[,*オペランド*...]] <CR>

※ [ ]で囲まれた項目　　省略可能な記述
※ ▲　　省略不可能な空白
※ △　　省略可能な空白
※ <CR>　　改行文字入力

■ 記述例

```
        .SECTION    P,CODE,ALIGN=4      ;Section P
        .END
          ↑           ↑                   ↑
      擬似命令      オペランド          コメント
      ニーモニック
```

注意

シンボルを指定できない擬似命令の場合は、シンボルを記述できません。

5.3 擬似命令オペランドに記述できるもの

擬似命令のオペランドには以下のものが記述できます。各命令におけるオペランドの記述形式は付録Ｂ「擬似命令リファレンス」で確認してください。

■ [*式* [,*式* | ,*文字列*]]
■ [*式* [,*式*]...]
■ *シンボル名* [, *シンボル名*]
■ *シンボル名* [, *シンボル名*]...
■ *モジュール名*
■ *セクション名* [,*属性ａ*] [,*属性ｂ*]
■ *文字列* [, *文字列*]

式は式の記述規則（3.8参照）に従って記述します。モジュール名、セクション名およびシンボル名は、名前の記述規則に従って命名します。

文字列の記述形式（図5.3）および規則を以下に示します。

形式　"*ASCII文字列*" | <*式*>

例
```
"MOJI"
"MOJI"<3>
<2><5>"MOJI"
"MOJI""RETSU"
```

図5.3　文字列の記述形式

■　文字列は、ASCII文字列、文字コード、およびそれらの組み合わせ によって表現できます。

■ ASCII文字列は、文字列をダブルクォーテーション（"）で囲んで記述します。

■ 文字コードは、式を< と > で囲んで記述します。

■ 文字コードとして記述する式に、前方参照シンボルは記述できません。また、式の値は-128～255までで、定数式でなければなりません。

■ 文字列は最大255文字まで記述できます。ただし.SDATA擬似命令では242文字までです。

5.4 演算サイズ指定

命令によっては、演算サイズ（オペランドで指定する処理対象データのサイズ）の指定ができます。擬似命令で指定できるサイズを以下に示します。

表5.2　擬似命令のサイズ指定

サイズ指定	解説
.B	バイト（8ビット）
.H	ハーフワード（16ビット）
.W	ワード（32ビット） デフォルト（演算サイズ指定がなかった場合はワードとなります）。

演算サイズ指定はオペレーションフィールドで行い、ニーモニックの後にピリオド（.）を付けて記述します。記述形式を以下に示します。

形式　ニーモニック[.B|.H|.W]

例

```
WORK:    .RES.B    20
```

図5.4　擬似命令演算サイズの記述形式

演算サイズ指定に大文字／小文字の区別はありません。ニーモニックと演算サイズ指定の間には空白文字を入れずに、全指定を続けて記述してください。

第６章
マクロ命令行の記述方法

6.1 マクロ命令の種類

アセンブラ as32R ではマクロ処理用に、以下のマクロ命令（表 6.1）と、マクロ処理用の文字列操作関数（表 6.2）をサポートしています。

表6.1 アセンブラのマクロ命令

マクロ命令（構文を構成する命令）	機能概要
.MACRO構文（.MACRO～.ENDM）	1行以上にわたる命令、擬似命令等を1つのマクロボディ[注1)]として記憶します（マクロ定義）。記憶されたマクロボディは、マクロコール[注2)]によってソースプログラム中に展開されます。
.ASSIGNA	算術式で指定した値を算術変数として定義します。
.ASSIGNC	文字式で指定した値を文字変数として定義します。
.INCLUDE	ソースプログラム中にオペランドで指定されたファイルを読み込みます。
.AIF構文（.AIF～.AELSE～.AENDI）	マクロ展開を条件により選択します。
.AWHILE構文（.AWHILE～.AENDW）	マクロ展開を条件により繰り返します。
.AREPEAT構文（.AREPEAT～.AENDR）	マクロ展開を指定された回数繰り返します。
.EXITM	マクロ展開を終了させます。

注 1） マクロボディ ：.MACRO ～ .ENDM 構文に内包された部分。.MACRO 命令で指定したマクロ名で呼び出すことができるマクロ定義。6.4節参照。

注 2） マクロコール ：定義されたマクロを使用した記述。as32R によってその部分がマクロ展開されます（.MACRO 命令で定義されたマクロボディがソースプログラム中に展開 される）。6.4節参照。

表6.2　アセンブラのマクロ処理用文字列操作関数

文字列操作関数	機能概要
.LEN	文字列の文字数を数えます。
.INSTR	文字列の位置を算出します。
.SUBSTR	文字列を取り出します。

各命令および関数の詳細については、付録C「マクロ命令リファレンス」を参照してください。

6.2 マクロ命令行の構成

マクロ命令行では、シンボルフィールドにシンボルまたはマクロ定義シンボル（プリプロセッサ変数）を記述します。また、オペレーションフィールドにマクロ命令ニーモニックを、オペランドフィールドにそのオペランドを記述します。文字列操作関数はマクロ命令のオペランドとしてオペランドフィールドに記述します。

マクロ命令行の構成と記述例を以下に示します。

■マクロ命令行の構成

△*マクロ命令ニーモニック*▲[*オペランド*]△<CR>
△*マクロ定義シンボル*：▲*マクロ命令ニーモニック*▲[*オペランド*]△<CR>

・マクロコール時
△*マクロ名*▲[*引数*]△<CR>
△*シンボル*：*マクロ名*▲[*引数*]△<CR>

※ [ ]で囲まれた項目　　省略可能な記述
※ ▲　　省略不可能な空白
※ △　　省略可能な空白
※ <CR>　　改行文字入力

■ 記述例

```
            .INCLUDE    "DATAB.H"      ;instruction line sample
AVAR_1:     .ASSIGNA    10
```

↑ マクロ定義シンボル　↑ マクロ命令ニーモニック　↑ オペランド　↑ コメント

```
MCR         R7,R7
```

↑ マクロ名　↑ 引数

マクロ命令行の記述方法は、6.3「マクロ命令行の記述方法」を参照してください。

6.3 マクロ命令行の記述方法

マクロ命令行には次のものが記述できます。

■ プリプロセッサ変数（仮引数、算術変数、文字変数）
■ 式（算術式、文字式、論理式。一般命令の式とは取り扱いが異なります）

以下より、プリプロセッサ変数（6.3.1参照）およびマクロ命令の式（6.3.2参照）の詳細を示します。

6.3.1 プリプロセッサ変数

プリプロセッサ変数とは、マクロボディ内のパラメータで、それにはマクロ展開時に引数を与えることができます。プリプロセッサ変数は、定義の方法によって以下の３つに分類されます。

■ 仮引数　　.MACRO 文のオペランドフィールドで定義されます。
■ 算術変数　　.ASSIGNA 命令で定義されます。
■ 文字変数　　.ASSIGNC 命令で定義されます。

以下に、各プリプロセッサ変数について説明します。

6.3.1.1 仮引数

形式

\ *仮引数名*

解説

仮引数は .MACRO 文で定義されます。
仮引数名の頭に\(バックスラッシュ)を付けることにより、マクロ定義の仮引数を参照できます。
仮引数の値はマクロコールで設定されますが、.MACRO 文においても仮引数の初期値を設定できます。
仮引数の参照は、その仮引数名を定義したマクロボディ内で行うことができます。
仮引数名は名前の規則に従って記述されなければなりません。

例

```
.MACRO      MCR   ARG_1          ;  MACRO 文
            MV    R5,\ARG_1      ;マクロボディ
            ADD   R6,\ARG_1      ;
.ENDM                            ;  ENDM 文
```

```
MCR    R7                              ;マクロコール
```

＜マクロ展開結果＞

```
           MV    R5,R7
           ADD   R6,R7
```

6.3.1.2 算術変数

形式

\&算術変数名

解説

.ASSIGNA 命令により算術変数の定義を行い、その値を設定します。
算術変数名の頭に \(バックスラッシュ)と、& を付けることにより、算術変数の値を参照できます。
算術変数の参照はすべてのマクロボディ内、およびマクロボディ外の算術式、文字式、論理式で行うことができます。

例

```
.MACRO     MCR                          ;MACRO 文
           LDI     R5,#\&AVAR_1         ;マクロボディ内
           ENDM                         ;ENDM 文


AVAR_1:    .ASSIGNA    10
           MCR                          ;マクロコール


AVAR_2:    .ASSIGNA    \&AVAR_1         ;マクロボディ外
```

＜マクロ展開結果＞

```
           LDI     R5,#10
```

6.3.1.3 文字変数

形式

\& 文字変数名

解説

.ASSIGNC 命令により文字変数の定義を行い、その値を設定します。
文字変数名の頭に \(バックスラッシュ)と、& を付けることにより、文字変数の値を参照できます。
文字変数の参照はすべてのマクロボディ内、およびマクロボディ外の算術式、文字式、論理式で行うことができます。

例

```
.MACRO      MCR                         ;   MACRO 文
                \&CVAR_1    R5,#1       ; マクロボディ内
                \&CVAR_2    R5,#2       ;
            .ENDM                       ;  ENDM 文


CVAR_1: .ASSIGNC    "SLLI"
CVAR_2: .ASSIGNC    "\&CVAR_1"          ; マクロボディ外
            MCR                         ; マクロコール
```

＜マクロ展開結果＞

```
            SLLI    R5,#1
            SLLI    R5,#2
```

6.3.2 式（マクロ命令用）

マクロ命令の式には以下の３種類があります。

■ 算術式　１つ以上の項を、一般的な代数規則に基づき演算子とカッコを用いて結合したものです。

■ 文字式　１つ以上の項を結合したものです。

■ 論理式　１つ以上の項を比較演算子、論理演算子およびカッコを用いて結合したものです。

以下に、これらの内容について説明します。

6.3.2.1 算術式

■算術式の記述規則

算術式は１つ以上の項を、一般的な代数規則に基づき演算子とカッコを用いて結合したものです。
算術式は32ビット符号付きです。
演算結果のオーバーフローは無視されます。
除算の結果は商のみ有効とします。
除数が０であればエラーとなります。
算術式の項として使用できるものは以下の３種類です。

・ 定数
・ プリプロセッサ変数
・ 文字列操作関数(.LEN / .INSTR)

■定数の記述規則

算術式の項として使用できる定数は次の4つです(( )内は各種定数に付ける接頭子)。

・ ２進数　(B'､b')
・ ８進数　(Q'､q')
・ 10進数　(D'､d')
・ 16進数　(H'､h')

接頭子（B'、Q'、D'、H'など）を省略すると10進数とみなされます。
定数の記述例を以下に示します。

```
B'01000101
Q'741
D'3209
H'B5F6
87905
```

■ プリプロセッサ変数の記述規則

算術式の項として次のプリプロセッサ変数が使用できます。

- 仮引数
- 算術変数
- 文字変数

仮引数の値は定数でなければなりません。
文字変数は、定数を表記した文字列（例： B'0101）を表す文字変数でなければなりません。それ以外はエラーになります。

■ 文字列操作関数(.LEN / .INSTR)の記述規則

算術式の項として使用できる文字列操作関数は次の２つです。

- .LEN 関数　　文字列の文字数を値とします。
- .INSTR 関数　　文字列中の文字の位置を値とします。

文字列操作関数の記述例を以下に示します。

```
.LEN("ABC")+3
.INSTR("DEF","E")*4
```

■ 演算子と優先順位演算子

算術式に使用できる演算子を表6.3に示します。

表6.3　算術演算子の種類

演算子	名 称
+	単項正
－	単項負
+	二項加算
－	二項減算
＊	二項乗算
/	二項除算

演算子の優先順位を表6.4に示します。

表6.4　算術演算子の優先順位

優先順位	演算子	名 称
1	()	カッコ
2	+－	単項正 単項負
3	＊/	二項乗算 二項除算
4	+－	二項加算 二項減算

■ 算術式の構成

算術式の構成を図6.1に示します。

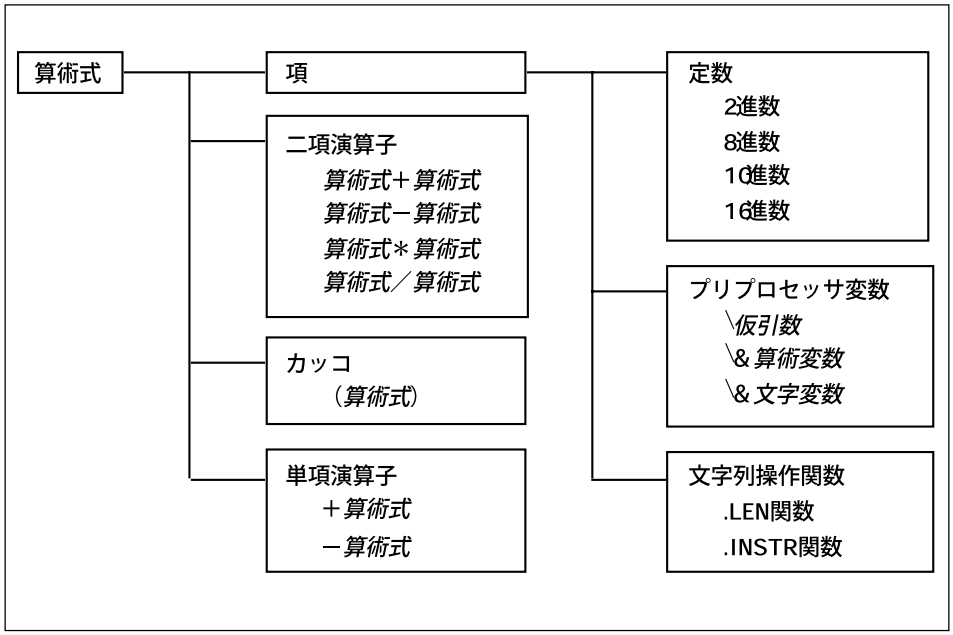


図6.1　算術式の構成図

■ 算術式の記述例

以下に算術式の記述例を示します。

```
\ARG_1 + 1                      ..... 仮引数
\&AVAR_1 - \&AVAR_2             ..... 算術変数
(-\&CVAR * 2)                   ..... 文字変数
```

6.3.2.2 文字式

■ 文字式の記述規則

文字式は、1つ以上の項を結合したものです。
項の結合は1つ以上のスペースをあけて、項を続けて記述することにより行います。
文字式の項として使用できるものは以下の3種類です。

- 文字列
- プリプロセッサ変数
- 文字列操作関数(.SUBSTR)

■ 文字列の記述規則

文字式の項として、文字列が使用できます。
文字列はダブルクォーテーション(")で囲まなければなりません。
文字列として文字コードを使用することはできません。
文字列中に " を含める場合は、" " と2回重ねて記述します。
文字列の文字数は0～255の範囲になければなりません。
以下に文字列の記述例を示します。

```
"MOJI"
"MO""JI"                        ..... MO"JIを表します
"M" "O" "J" "I"                 ..... MOJIを表します
```

■ プリプロセッサ変数の記述規則

文字式の項としてプリプロセッサ変数が使用できます。
プリプロセッサ変数はダブルクォーテーション(")で囲まなければなりません。
プリプロセッサ変数には、仮引数、算術変数、文字変数があります。
仮引数、文字変数を参照した場合、その値は文字列に置き換えられます。

算術変数を参照した場合、その値は10進整数の文字列（たとえば"1234"等）に置き換えられます。

■ 文字列操作関数(.SUBSTR)の記述規則

文字列の項として、.SUBSTR関数が使用できます。
.SUBSTR関数によって、文字列中から取り出した文字列の値を参照します。
以下に.SUBSTR関数の記述例を示します。

```
.SUBSTR("AABCCD",1,3) "DEF"
```

■ 文字式の構成

文字式の構成を図6.2に示します。

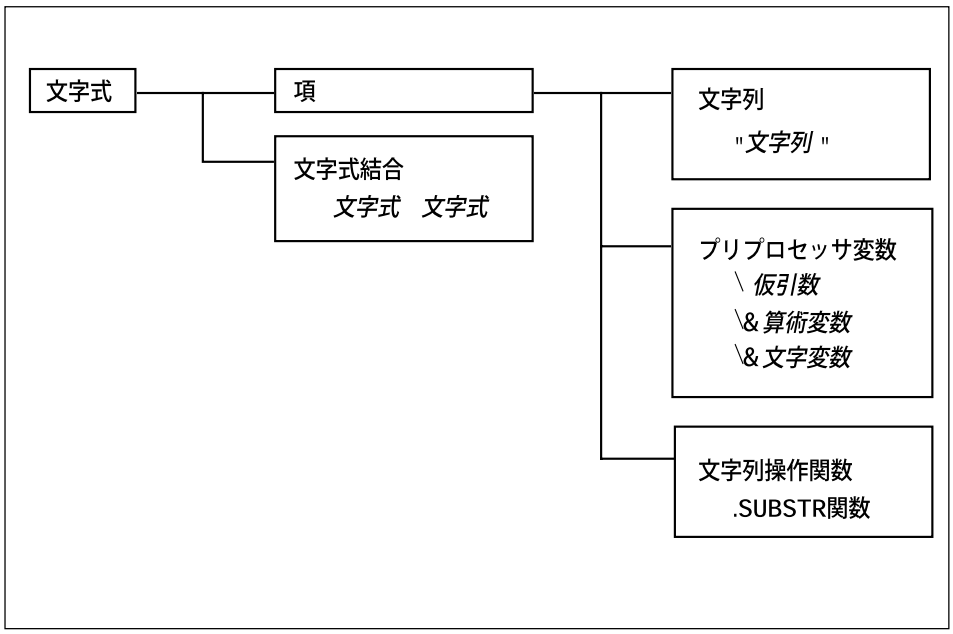


図6.2　文字式の構成図

■ 文字式の記述例

以下に文字式の記述例を示します。

```
"\ARG_1" "ABCD"
"\&AVAR_1"\ARG_1"
"\&CVAR_1" "\&CVAR_2" "EFG"
```

6.3.2.3 論理式

■ 論理式の規則

論理式は１つ以上の項を比較演算子、論理演算子およびカッコを用いて結合したものです。
論理式の結果は、真か偽のどちらかです。

■ 論理式の項

論理式の項として使用できるものは以下の３種類です。

- 算術関係式
- 文字関係式
- 算術式

■ 算術関係式の記述規則

論理式の項として、算術関係式が使用できます。
算術関係式は、算術式と算術式の間に比較演算子をおくことにより、真 / 偽を評価します。
算術関係式の記述例を以下に示します。

```
\&AVAR_1  GT  5
\&AVAR_2  NE  \&AVAR_3
```

■ 文字関係式の記述規則

論理式の項として文字関係式が使用できます。
文字関係式は、文字式と文字式の間に比較演算子をおくことにより、真 / 偽を評価します。
文字関係式の記述例を以下に示します。

```
"\&CVAR_1"  EQ  "MOJI"
"MOJI" "\&CVAR_2"  NE  "MOJISHIKI"
```

■ 算術式の記述規則

論理式の項として算術式が使用できます。
0 以外の数値を直接記述した場合、または参照した値が 0 以外の数値であった場合、そ

れは真とみなされます。
0 を直接記述した場合、または参照した値が 0 であった場合、それは偽とみなされます。

算術式の記述例を以下に示します。

```
\&AVAR_1
5          …… 真
0          …… 偽
```

■ 演算子と優先順位

論理式に使用できる演算子を表6.5、表6.6、表6.7に示します。

表6.5　論理式で用いられる算術演算子

演算子	名 称
+	単項正
－	単項負
+	二項加算
－	二項減算
＊	二項乗算
/	二項除算

表6.6　論理式で用いられる比較演算子

演算子	名 称
ＥＱ	等しい(＝)
ＮＥ	異なる(≠)
ＬＴ	より小(＜)
ＬＥ	以下(≦)
ＧＴ	より大(＞)
ＧＥ	以上(≧)

表6.7　論理式で用いられる論理演算子

演算子	名 称
ＮＯＴ	単項論理否定
ＡＮＤ	二項論理積
ＯＲ	二項論理和

演算子の優先順位を表6.8に示します。

表6.8　論理式で用いられる演算子の優先順位

優先順位	演算子	名 称
1	()	カッコ
2	+ −	単項正 単項負
3	＊/	二項乗算 二項除算
4	+ −	二項加算 二項減算
5	EQ NE LT LE GT GE	比較演算子
6	ＮＯＴ	単項論理否定
7	ＡＮＤ	二項論理積
8	OR	二項論理和

■ 論理式の構成

論理式の構成を図6.3に示します。

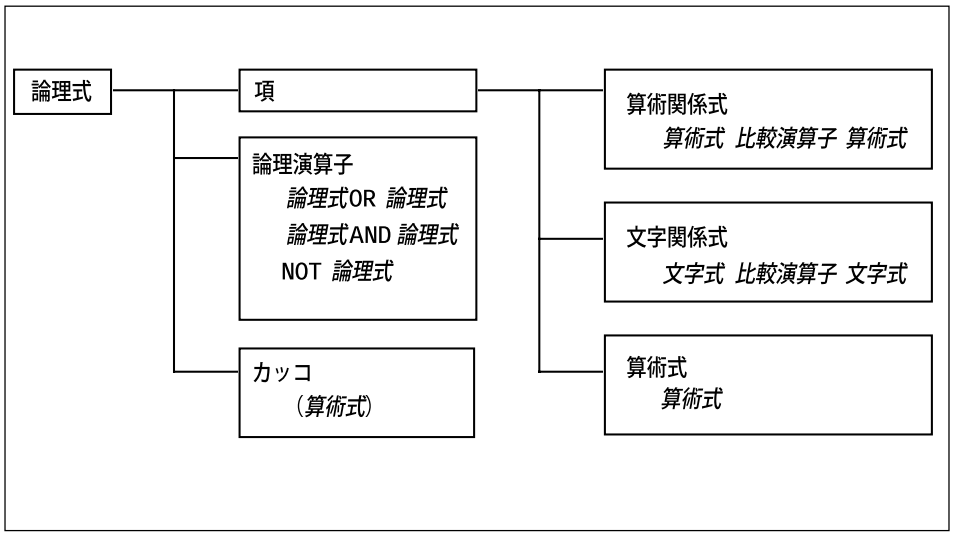


図6.3　論理式の構成図

6.4 マクロ処理の記述方法

6.4.1 マクロ処理の概要

マクロ処理とは、ソースプログラム中の一連の命令や擬似命令等を１つの集まりとして名前（マクロ名）を定義しておき、同じプログラム中のマクロ命令によって、マクロ名を定義された内容に置き換える処理です。マクロ処理には、マクロ定義、マクロコール、マクロ展開等があります。

■ マクロ定義　　１行以上にわたる命令、擬似命令等を１つのブロックとして記憶することをマクロ定義といいます。また、このブロックをマクロボディと言います。

■ マクロコール　　マクロ定義で記憶したマクロボディをソースプログラム中へ展開する指定をマクロコールと言います。マクロコールは、マクロ定義で定義されたマクロ名をオペレーションフィールドに記述することにより行われます。

■ マクロ展開　　マクロコールによりマクロボディをソースプログラム中に展開することをマクロ展開と言います。

図6.4にマクロ定義とマクロコールの記述形式、図6.5にマクロ展開例を示します。

```
.MACRO   マクロ名   [仮引数指定 , 仮引数指定 ]...      ; マクロ定義
.........................................          ;
.........................................          ;   ― マクロボディ
.............................              ;
.ENDM                                              ;


マクロ名 [引数]                                    ; マクロコール
```

図6.4　マクロ定義とマクロコールの記述形式

```
.MACRO  MCR   ARG_1,ARG_2                              ;マクロ定義
        MV    R5,\ARG_1             ;マクロボディ      ;
        ADD   R6,\ARG_2             ;                  ;
.ENDM                               ;


        MCR    R7,R7                ;マクロコール
```

＜マクロ展開結果＞

```
        MV    R5,R7
        ADD   R6,R7
```

図6.5　マクロ展開例

なお、マクロ展開時には、マクロコール文のオペランドから引数を与えることができます。

次節6.4.2より、マクロ定義の記述方法、マクロボディの記述方法と展開内容、および、マクロコールの記述方法について示します(6.4.2～6.4.4)。

6.4.2 マクロ定義の記述方法

マクロ定義とは、１行以上にわたる命令、擬似命令等を１つのブロック(マクロボディ)として記憶することです。以下に、マクロ定義の記述方法と機能について説明します。

形式

.MACRO　　マクロ名　[*仮引数指定*[, *仮引数指定*]ê]
.ENDM

仮引数指定：　*仮引数名*[= *初期値*]

解説

.MACRO 命令により、マクロ定義の開始を宣言します。
.MACRO 命令のオペランドでマクロ名、仮引数名の定義、および、仮引数の初期値の設定ができます。
.MACRO 命令で定義したマクロ名は、マクロコールとして使用できます。

マクロ定義は、.MACRO 命令、マクロボディおよび .ENDM 文からなります。
マクロ名と仮引数指定の間には、１つ以上のスペースまたはタブが必要です。
マクロ定義内に他のマクロ定義を含めることはできません。
.MACRO 命令で定義した仮引数名は、そのマクロ定義内に限り有効です。
１つの .MACRO 命令内で、仮引数名に同じ名前は使用できません。
マクロ名、仮引数名は名前の規則に従います。

初期値の設定は、仮引数名の後に等号(=)をつけて行います。
マクロコールで引数を省略すれば、.MACRO 命令で定義した初期値が仮引数となります。
マクロコールで引数を省略し、.MACRO 命令においても初期値を定義していなかった場合、空の文字列が仮引数となります。
スペース（)、コンマ（,)、等号（=)、不等号（より小(<)）を含めたものを初期値として定義する場合、初期値は<と > で囲むかまたはダブルクォーテーション(")で囲まなければなりません。
この場合 < と > は初期値には含まれませんが、ダブルクォーテーション(")は初期値として含まれます。

.ENDM 命令により、マクロ定義の終了を示します。マクロ定義の終りには、必ず .ENDM 命令がなければなりません。
以下にマクロ展開例を２つ示します。

マクロ展開例１

```
.MACRO     MCR   ARG_1,ARG_2          ;  MACRO 文
           MV    \ARG_1,\ARG_2        ; マクロボディ
           ADD   \ARG_1,R7            ;
.ENDM                                 ;  ENDM 文

MCR    R5,R6                          ; マクロコール
```

＜マクロ展開結果＞

```
MV          R5,R6
ADD         R5,R7
```

マクロ展開例２

```
.MACRO   MCR   ARG_1=STR_DATA,ARG_2   ;  MACRO 文
.SECTION    \ARG_1                    ;マクロボディ
.SDATA      "\ARG_2"                  ;
.ENDM                                 ;  ENDM 文

MCR    ,123                           ; マクロコール
```

＜マクロ展開結果＞

```
.SECTION    STR_DATA
.SDATA      "123"
```

6.4.3　マクロボディの記述方法と展開内容

マクロボディとは、.MACRO 文と .ENDM 文の間にある文のことです。なお、ここでいう文とは、１つ以上の命令および擬似命令より構成されたものです（ただし、他のマクロ定義を除きます)。

マクロボディではマクロ展開時、以下に示す処理を行います。

- ■ プリプロセッサ変数の置換
- ■ 置換除外
- ■ 順序番号処理
- ■ コメント判断

以下より、これらの処理の内容について説明します。

6.4.3.1 プリプロセッサ変数の置換

マクロボディ内のプリプロセッサ変数は、マクロ展開時、その変数に対応する値に置き換えられます。プリプロセッサ変数をその後に記述される文字列と区別したい場合は、間をシングルクォーテーション(')で区切ります。

シングルクォーテーション(')は仮引数名とみなされません(' は名前ではありません)。プリプロセッサ変数の後に文字として ' を使用する場合は、' を重ねて記述し '' とします。区切りとしての ' は、展開後には現われません。

```
.MACRO   MCR        ARG_1,ARG_2              ;   MACRO文
         MV         \ARG_1'1,\ARG_2'2        ; マクロボディ
         .SDATA     "\ARG_2''PROG"           ;
.ENDM                                        ;   ENDM文


MCR      R,ASM                               ; マクロコール

＜マクロ展開結果＞
         MV      R1,R2
         .SDATA     "ASM'PROG"
```

図6.6 プリプロセッサ変数の置換例

6.4.3.2 置換除外

形式

\(文字列)

解説

\(バックスラッシュ)の後にカッコ( ) で囲んだ文字列を記述すれば、その文字列は置き換えの対象となりません。
置換除外指定の\(バックスラッシュ)、カッコ( ) は展開後には現われません。
置換除外としての文字列は、１行を越えて指定できません。
1行の終わりにきても右カッコ) が現われなければエラーとし、行の終わりまでを置換除外とみなします。

例

```
.MACRO   MCR      ARG_1                    ;   MACRO 文
           .SDATA     "\ARG_1"             ; マクロボディ
           .SDATA     "\(\ARG_1)"          ;
.ENDM                                       ;   ENDM 文

 MCR    TEST                                ; マクロコール
```

＜マクロ展開結果＞

```
.SDATA     "TEST"
.SDATA     "\ARG_1"
```

6.4.3.3 順序番号処理

形式

```
\@
```

解説

順序番号(\@)は、マクロコールされるごとにカウントアップします。
順序番号は、00001 ～ 99999 の 5 桁の 10 進数です。
最初に展開される順序番号には 00001 が現われます。

例

```
            .MACRO    MCR             ;   MACRO 文
A\@:        MV        R5,R6           ;   マクロボディ
B\@:        ADD       R1,R5     ;
            .ENDM                     ;   ENDM 文

            MCR                 ;   マクロコール
```

＜マクロ展開結果＞

```
A00001:  MV    R5,R6
B00001:  ADD   R1,R5
```

6.4.3.4 コメント判断

形式

```
\;
```

解説

\(バックスラッシュ)とセミコロン(;)の後ろにコメントを記述すれば展開されません。
行の途中に　\;　があれば、それ以降は展開されません。

例

```
.MACRO      MCR                     ;MACRO 分
            ADD   R5,R6 ;TEST_1     ;マクロボディ
            SLLI  R5,#1\;TEST_2     ;
.ENDM                               ;ENDM 文

MCR                                 ;マクロコール
```

＜マクロ展開結果＞

```
ADD      R5,R6;TEST_1
SLLI     R5,#1
```

6.4.4　マクロコール

マクロコールとは、マクロ定義で記憶したマクロボディをソースプログラム中へ展開する(マクロ展開)指定のことです。以下にマクロコールについて説明します。

形式

[*シンボル*]	マクロ名	[*引数指定*[,*引数指定*]…]

引数指定:　*引数*　----------位置指定
仮引数名 = *引数*　----------キーワード指定
※引数指定方法には、位置指定、キーワード指定、および
それらの混合指定の３つがあります。

解説

処理を実行させたいマクロ名をオペレーションとして記述し、そのマクロボディを展開します。引数の指定はオペランドで行います。その指定方法には、位置指定、キーワード指定、および混合指定があります(詳細は本解説後半で説明しています)。

引数の指定を2つ以上行う場合、引数指定と引数指定の間はコンマ(,)で区切らなければなりません。また、コンマを連続して記述すれば、その位置に対応している引数は省略したとみなされます。

マクロコール時の引数指定は、マクロ定義時の仮引数指定の初期値より優先されます。引数を省略した場合は、マクロ定義で設定された初期値が用いられます。引数を省略した場合、マクロ定義で初期値が設定されていなければ空の文字列が引数として用いられます。

空白文字（)、コンマ（,）等号（=)、不等号（より小(<)）を含めたものを引数とする場合、その引数は < と > で囲むか、またはダブルクォーテーション(")で囲まなければなりません（例：A<Bという引数を<A<B>と記述する)。この場合、 < と > は初期値には含まれませんが、ダブルクォーテーション(")は初期値として含まれます。

■ 位置指定規則

マクロ定義で定義された仮引数名順に引数を指定します。
マクロ定義で定義された仮引数名の数より引数の数のほうが少ない場合は、不足分の引数は省略したとみなします。
マクロ定義で定義された仮引数名の数より引数の数のほうが多い場合はエラーとし、余分な引数は無視されます。
以下に位置指定の記述例を示します。

```
.MACRO  MCR   A,B=R2,C,D=R10,E,F    ;  MACRO 文
              MV    \A,\B           ; \C
              ADD   \D,\E           ; \F
.ENDM                               ;  ENDM 文

MCR     R1,,<*TEST<1>>*,R5,R6       ;マクロコール

＜マクロ展開結果＞
        MV    R1,R2 ;*TEST<1>*
        ADD   R5,R6 ;
```

■ キーワード指定規則

マクロ定義で定義された仮引数名の順番とは無関係に引数を指定します。
引数の指定は、マクロ定義で定義された仮引数名を記述し、その後に等号(=)をつけて行います。
同じ仮引数を指定した場合、引数は最後に指定した仮引数名のものが採用されます。
マクロ定義で定義された仮引数名の内、マクロコールで引数の指定がされていないものがあれば、それは省略したとみなします。
マクロ定義で定義された仮引数名にないものをマクロコールで指定すればエラーとし、その指定は無視されます。
以下に記述例を示します。

```
.MACRO MCR   A=R1,B=R6,C,D                  ;  MACRO 文
             MV    \A,\B                    ; \C
             SLLI  \D
.ENDM                                       ;  ENDM 文

MCR     C=<*TEST<2>>*,A=R2,D=<R7,#2>,A=R5   ;マクロコール

＜マクロ展開結果＞
        MV    R5,R6 ;*TEST<2>*
        SLLI  R7,#2
```

■ 混合指定規則

位置指定とキーワード指定はあわせて指定できます。
位置指定とキーワード指定で同じ仮引数を指定した場合、引数は最後に指定した仮引数名のものが用いられます。
以下に記述例を示します。

```
.MACRO  MCR     A,B=R6,C,D=<R7,#1>       ;  MACRO 文
                ADD   \A,\B                      ; \C
                SLLI  \D
.ENDM                                    ;  ENDM 文

 MCR    R5,D=<R7,#2>,,C=TEST,<*TEST_3*>  ; マクロコール
```

＜マクロ展開結果＞

```
        ADD   R5,R6 ;*TEST_3*
        SLLI  R7,#2
```

6.5 マクロ処理のネスト構造

マクロ定義、マクロコール、マクロ命令等はネスト構造にできます。つまり、マクロボディ内やマクロ命令の構文内に、他のマクロ定義やマクロ命令の構文を含むことができます。

マクロ処理におけるネスト構造を表6.9に示します。表中の記号（○、×、／）は次の意味を持ちます。

○：　記述できる
×：　記述できない
／：　記述できる場合とできない場合がある

表6.9　マクロの処理のネスト構造

記述場所 / 記述するもの	マクロ定義外	マクロボディ内	.INCLUDEファイル内	.AIF構文内	.AWHILE構文内	.AREPEAT構文内	算術式、文字式、論理式[注)]
マクロ定義	○	×	○	○	×	×	×
マクロコール	○	○	○	○	○	○	×
.ASSIGNA命令	○	○	○	○	○	○	×
.ASSIGNC命令	○	○	○	○	○	○	×
.INCLUDE命令	○	○	○	○	○	○	×
.AIF構文	○	○	○	○	○	○	×
.AWHILE構文	○	○	○	○	○	○	×
.AREPEAT構文	○	○	○	○	○	○	×
.EXITM命令	×	○	×	／	○	○	×
.LEN、.INSTR、.SUBSTR関数	×	×	×	／	／	／	○
プリプロセッサ変数	×	○	×	×	×	×	○

注)算術式、文字式、論理式　：　.ASSIGNA、.ASSIGNC、.AIF、.AREPEAT、.AWHILE文

.EXITM 命令は、.AIF 構文が、マクロボディ内、.AWHILE 構文内、および、.AREPEAT 構文内のいずれかに記述されている場合のみ、.AIF 構文内に記述することができます（それ以外は記述できません）。

プリプロセッサ変数は、.AIF 構文、.AWHILE 構文、.AREPEAT 構文がマクロボディ内に記述されていれば、それらの構文内でも記述することができます（マクロ定義外では記述できません）。また、マクロ定義内外を問わず、算術式、文字式、論理式でも記述できます。

6.6 マクロ命令によるプログラミング例

マクロ命令を用いると、複数行の命令を一つの命令に置き換えたり、複数行の命令を繰り返し展開したりすることができます。以下では .MACRO、.AREPEAT を例に挙げて、マクロ命令の記述方法について説明します。図6.7にマクロ命令行の記述例を示します。

```
 1            .SECTION   PROGRAM
 2  ;
 3  SYMBOL:   .ASSIGNA   10
 4  ;
 5            .MACRO     ABC      ARG1, ARG2
 6                       LD24     \ARG1,#10
 7                       ADD3     \ARG2,\ARG1,#\&SYMBOL
 8            .ENDM
 9  ;
10  ;
11            ABC              R6, R7
12            .AREPEAT         \&SYMBOL
13               NOP
14            .AENDR
15  ;
16            .END
```

図6.7　マクロ命令行の記述例

マクロの置き換えの対象となる部分をマクロボディと呼びます。マクロボディは、.MACRO、.ENDM で宣言します（.MACRO ～ .ENDM で囲まれた命令行がマクロボディです）。.MACRO 命令の行では、命令の後にマクロ名と仮引数を指定します。図 6.7 の 5 行目では、ABC がマクロ名で、ARG1、ARG2 が仮引数です。仮引数をマクロボディ内で参照する場合は､仮引数名にバックスラッシュ' \ 'を付けて記述します。

マクロ名は、マクロコールの際に使用します（図6.7の11行目）。マクロボディは、マクロコールの行位置に展開されます。マクロコールの引数として記述した文字列は、そのままマクロボディの仮引数として渡されます。

マクロ命令では、擬似命令 .ASSIGNA で宣言した算術変数が使用できます。算術変数とは、任意の名前に算術式を割り当てたものです。算術変数を参照する場合は、変数名の前にバックスラッシュ（\）とアンパサンド（&）を付けて記述します。算術変数は、マクロボディ内、およびマクロ命令のオペランド（算術式が記述できる命令に限る）に記述できます。算術式について詳しくは、6.3.2.1「算術式」を参照してください。

||||注意||||

マクロ命令では、アセンブラの擬似命令 .EQU および .ASSIGN で宣言したシンボルは使用できないので注意してください。

6.7 マクロ命令の制限事項

以下にマクロプログラミングにおける制限事項を示します。

■ .INCLUDE 命令のネスト

.INCLUDE 命令を使用する場合、ネストは 8 レベルまで可能です。

■ マクロ定義の個数

マクロ定義は、1024 個まで使用できます。

■ マクロボディのサイズ

マクロボディは、合わせて 128K バイトのサイズまで使用できます。

■ マクロコールのネスト

マクロコールを使用する場合、ネストは 32 レベルまで可能です。

■ .AREPEAT、.AWHILE 構文のネスト

.AREPEAT 構文、および .AWHILE 構文を使用する場合、ネストは 32 レベルまで可能です。

■ .AREPEAT、.AWHILE 構文での文のサイズ

.AREPEAT 構文、および .AWHILE 構文での文は、1 回につき 16K バイトのサイズまで使用できます。

第７章
アセンブラのメッセージ

7.1 アセンブラの実行結果を知るには

アセンブラの実行結果は、メッセージおよび終了コードから判断できます。

7.1.1 メッセージの出力形式

アセンブラは動作中にエラーを検出すると、実行状況を示すメッセージを標準エラー出力（通常は画面）に表示します。メッセージの出力形式を以下に示します。

■ 基本形式

ツール名 : *入力情報* : *メッセージ種別* : *メッセージ*

※「入力情報: 」は必要時のみ出力

■ パターン

a132R: *ファイル名*: *メッセージ種別*: *メッセージ*
a132R: *ファイル名*,*行番号*: *メッセージ種別*: *メッセージ*
a132R: <*コマンド行*>: *メッセージ種別* : *メッセージ*
a132R: *メッセージ種別* : *メッセージ*
※下線部分は入力情報（下線は出力されません）。

■ 出力例

```
a132R: "abc.ms",line5:error: invalid character &
```

ツール名　ファイル名　行番号　メッセージ種別　メッセージ

7.1.2 メッセージ種別

メッセージは警告程度によって、表7.1のように分類されています。

表7.1 メッセージ種別

メッセージ種別（表示）	メッセージ発生時の動作
ワーニング（warning）	ワーニングメッセージを出力し、処理を続行します。
エラー（error）	エラーメッセージを出力し、処理を中止します。
重大エラー（fatal）	エラーメッセージを出力し、処理を中止します。

メッセージの詳細は、7.2「アセンブラのメッセージ一覧」を参照してください。

7.1.3 終了コード

アセンブラは実行後、以下の終了コード（実行結果を示す値）を返します（表7.2）。

表7.2 終了コード

終了コード	実行結果
0	正常終了した。または、ワーニングが発生した。
1	エラーが発生した。

7.2 アセンブラのメッセージ一覧

7.2.1 ワーニング

表7.3 アセンブラのワーニングメッセージ（1/2）

メッセージ	解説
constant overflow, regard as *value*	指定された定数項がオーバーフローしています。数値*value* とみなします。
entry point address is not in a code section	エントリポイントに指定されたアドレスがコードセクションの中ではありません。
entry point is not word aligned address	エントリーポイントに指定されたアドレス はワードアライメントがとれていません。
ignore allocation attribute of a dummy section	ダミーセクションの配置属性(ALIGN= 境界調整値 または LOCATE= 先頭アドレス)は指定できませんので無視します。
ignore sign bit at *n*-bit immediate data	0 拡張される *n* ビット即値が指定された値と一致しないおそれがあります。 ［例］OR3 R0,R0,#-1→ OR3 R0,R0,#0x0000ffff のコードを生成
instruction out of a code section	命令が、セクション属性が CODE 以外のセクションに記述されています。セクション属性が CODE 以外の場合、そのセクション中のラベルは必ずしもワードアライメントが取られませんので、不正な動作をする可能性があります。
instruction's behavior is undefined	指定されたオペランドの組み合わせによる命令の動作結果は保証されていません。 ［例］LD R0,@R0+
"*label*": ignore label declaration	ラベル指定 *label* がありますが、ラベルを定義できない行ですのでラベルを無視します。次の擬似命令のある行ではラベル定義できません。 .ALIGN, .PROGRAM, .SECTION, .END, .EXPORT, .IMPORT, .GLOBAL
"*label*": not referenced import symbol	ラベル *label* は、アセンブリソース中では参照されていないが、外部参照シンボルと宣言されています。アセンブラは、この場合には参照情報をオブジェクトファイルに出力しません。

表7.3　アセンブラのワーニングメッセージ（2/2）

メッセージ	解説
no section directive, generate section P	擬似命令 .SECTION によるセクション宣言が出現していませんので、セクション P を自動生成します。このデフォルトのセクションは、.SECTION P,code,align=4 の指定とみなします。
sign extension at *n*-bit displacement	*n*ビットディスプレースメント値が指定された値と一致しない おそれ があります。 [例]LD　R0,@(65535,R1) → LD R0,@(-1,R1) のコードを生成
sign extention at *n*-bit immediate data	符号拡張される*n*ビット即値が指定された値と一致しない おそれ があります。 [例] ADDI R0,#255→ ADDI R0,#-1 のコードを生成
too long symbol, truncated	シンボル名が長すぎます。一部切り捨てました。
caution! there are some data in code section	コードセクションに命令以外のデータ（.data疑似命令等）を記述しています。ワーニングメッセージは出力されますが、データは正しく配置されます。

7.2.2 エラー

表 7.4 アセンブラのエラーメッセージ（1/9）

メッセージ	解説
addressing mode error in operand *n*	第 *n* オペランドの指定が正しくありません。
.AENDI directive is missing	.AIF 構文の終わりに .AENDI 文がありません。
.AENDR directive is missing	.AREPEAT 構文の終わりに .AENDR 文がありません。
.AENDW directive is missing	.AWHILE 構文の終わりに .AENDW 文がありません。
.AREPEAT buffer overflow	.AREPEAT 構文の繰り返し展開用テキストが多すぎます（繰り返し展開用のテキスト保存領域は 16K バイトです）。
argument buffer overflow	マクロの引数情報のセーブ領域が不足しています。
argument specification error	マクロコールの引数指定にエラーがあります。
argument specification is too long	マクロコールの引数指定が長すぎます。
.AWHILE buffer overflow	.AWHILE 構文の繰り返し展開用テキストが多すぎます（繰り返し展開用のテキスト保存領域は 16K バイトです）。
branch to invalid address	分岐先に指定されたアドレスが不正です。分岐先はワードアライメントがとれていなければなりません。

表7.4 アセンブラのエラーメッセージ（2/9）

メッセージ	解説
can't evaluate expression value	以下に示す疑似命令において、関連にある行が処理されるとき、式は評価可能でなければなりません。 .EQU および .ASSIGN でラベルに割り当てる値 … 指定可／不可：定数 可、相対アドレス 可 .END 式　で指定するエントリポイントアドレス … 定数 可、相対アドレス 可 .DATAB[.{B|H|W}] *式1*,*式2*　の*式1*で指定する繰り返し回数 … 定数 可、相対アドレス 不可 .RES[.{B|H|W}] 式　で指定する確保領域 … 定数 可、相対アドレス 不可
constant overflow	値が許される範囲を超えています。
division by zero in operand *n*	第*n*オペランドを0で割る演算となりました。
duplicate section attribute	擬似命令 .SECTION のセクションの属性の指定が重複しています。セクション属性はCODE、DATA、COMMON、STACK、またはDUMMY のうちいずれか1つを、配置属性は「ALIGN=境界調整値」または「LOCATE= 先頭アドレス」のどちらか一方を、それぞれ指定する必要があります。
.ENDM directive is missing	マクロ定義の最後に .ENDM 文がありません。
entry point should be in a code section	エントリポイントアドレスの指定が不正です。指定されたアドレスは、アセンブルした全てのセクションのどこにも入っていません。
expression syntax error in operand *n*	第*n*オペランドの式の記述に文法エラーがあります。
illegal label location	シンボルフィールドにシンボルは記述できません。
illegal location for .AELSE	.AELSE 文に対応する .AIF 文がありません。
illegal location for .AENDI	.AENDI 文に対応する .AIF 文がありません。

表7.4 アセンブラのエラーメッセージ (3/9)

メッセージ	解説
illegal location for .AENDR	.AENDR 文に対応する .AREPEAT 文がありません。
illegal location for .AENDW	.AENDW 文に対応する .AWHILE 文がありません。
illegal location for .ENDM	.ENDM 文に対応する .MACRO 文がありません。
illegal location for .EXITM	マクロ外で .EXITM 擬似命令が使用されています。
illegal name	プリプロセッサ変数、マクロ名等の文字列が名前の規則に反しています。
illegal number	数値の指定に誤りがあります。
illegal operand	マクロ命令のオペランドに誤りがあります。
illegal placed source file	ソースファイル名の指定位置に誤りがあります。
illegal redefinition	算術変数を文字変数として、または文字変数を算術変数として定義しています。
illegal suffix	マクロ命令に余分な指定がされています。
include nest over 8	インクルードの最大ネストレベル(8レベル)を超えています。
invalid address *addr*	開始アドレスの指定が不正な値*addr* です。
invalid alignment value *value*	境界調整値の指定が不正な値*value* です。
invalid character *ch*	文字 *ch* は、文法上許されていない文字です。
invalid repeat times -*num*	繰り返し数の指定値 -*num* が不正です。繰り返し数は 0 以上でなければなりません。

表7.4 アセンブラのエラーメッセージ (4/9)

メッセージ	解説
invalid reserve area size -*num*	領域確保サイズの指定値 -*num* が不正です。領域確保サイズは0以上でなければなりません。
"*label*": can't assign	ラベル *label* に対して、.ASSIGN 擬似命令で値を代入できません。セクション名であるか、既にラベルとして定義されている、もしくは外部参照または外部定義のラベルと宣言している場合が考えられます。
"*label*": symbol declared inconsistently	ラベル *label* の宣言が、以前の宣言と矛盾しています。擬似命令 .IMPORT で外部参照としたラベルを、内部で定義しています。
"*label*": symbol redeclared	同じ ラベル *label* を再度宣言しています。.ASSIGN で定義するラベルのみ再定義可能です。
"*label*": undefined symbol	ラベル*label*は 未定義のシンボルです。
line too long	一行のサイズが長すぎます。
loop nest is too deep	繰り返し構文（.AWHILE、.AREPEAT 構文）のネストが最大値（32）を超えています。
macro buffer overflow	マクロボディのセーブ領域が不足しています。
macro call nest is too deep	マクロコールのネストが最大値(32)を超えています。
macro name is missing	マクロ定義文にマクロ名がありません。
macro table overflow	マクロ定義の最大値(1024個)を超えています。
"*macro-instruction*" : too many operands	マクロ命令 *macro-instruction* で、オペランドが多すぎます。
missing ,	, が必要です。

表7.4 アセンブラのエラーメッセージ（5/9）

メッセージ	解説
missing =	= が必要です。
missing >	> が必要です。
missing) in operand *n*	第*n*オペランドに) が必要です。
multiple definition of macro	同一マクロ名でマクロが二重定義されています。
multiple definition of parameter	マクロ定義文で同一名の仮引数を使用しています。
n-bit displacement overflow in operand *m*	第 *m* オペランドの *n* ビットのディスプレースメントがオーバーフローしました。
n-bit immediate data overflow in operand *m*	第 *m* オペランドの *n* ビットの即値データがオーバーフローしました。
name is too long	プリプロセッサ変数名が長すぎます。
negative loop counter	.AREPEAT 構文のオペランドに、負の値が記述されています。
nesting of macro definition	マクロボディ中にマクロ定義があります。
non terminate string	文字列の終端が指定されていません。文字列はダブルクォートでくくる必要があります。
operator is not a permitted relative address operation	演算子*operator* は相対アドレスに対する演算として許されていません。
"*option*" : illegal placed option	コマンドオプション *option* の指定位置に誤りがあります。
"*option*" : invalid option	コマンドオプション *option* の記述に誤りがあります。

表7.4 アセンブラのエラーメッセージ (6/9)

メッセージ	解説
"*option*" : missing opition argument	オプション *option* にオプションパラメータの指定がありません。
.PROGRAM module-name redefined	擬似命令 .PROGRAM でモジュール名を指定できるのは1回だけです。
.PROGRAM module-name required	擬似命令 .PROGRAM ではモジュール名の指定が必要です。
required a section name in operand *n*	第*n*オペランドの式で、sizeofに対してはセクション名の指定が必要です。
required label declaration	擬似命令 .EQU または .ASSIGN の行には、値を割り当てるラベルを宣言する必要があります。
required operand *n*	第 *n* オペランドの指定が必要です。
"*section_name*": inconsistent section attribute	*section_name*: セクションの属性の指定が以前の宣言と矛盾しています。全ての属性の指定を省略すると以前の宣言と同じになります。
section name required	擬似命令 .SECTION では、セクション名の指定が必要です。
shift amount should be a constant	シフト量は、定数式で指定しなければなりません。
sizeof(*section_name*): not a constant	sizeof(*section_name*) は、定数値ではありません。定数式として使用できません。
string : constant syntax error	文字列 *string* が定数の文法エラーです。式の項としての定数と解釈を始めましたが、途中で文法エラーを検出しました。
:*string* : invalid displacement size in operand *n*	第 *n* オペランドのディスプレースメントサイズ指定「:*string*」が不正です。
:*string*: invalid immediate size in operand *n*	第 *n* オペランドの即値サイズ「:*string*」が不正です。
.*string* : invalid size in operand *n*	第*n*オペランドのオペランドサイズ指定「.*string*」 が不正です。

表7.4 アセンブラのエラーメッセージ（7/9）

メッセージ	解説
`string is too long`	文字列が長すぎます。
`"`*symbol*`": can't import/export`	シンボル *symbol* を外部参照または外部定義シンボルとすることはできません。*symbol* がセクション名であるか、.ASSIGN で定義されたラベルのいずれかであることが考えられます。
`"`*symbol*`": inconsistent import/export declare`	シンボル *symbol* に対する外部参照または外部定義の指定が以前の定義と矛盾しています。
`"`*symbol*`": not a section name in operand` *n*	第 *n* オペランドの式の `sizeof(`*symbol*`)` で使用しているシンボル *symbol* はセクション名ではありません。
`"`*symbol*`": not has constant value`	*symbol* は定数値が定義されたシンボルではありません。定数式として使用できません。
`syntax error at or near token in operand` *n*	文法エラーです。第 *n* オペランドの文字列 *token* 付近で発見しました。
`syntax error in constant expression`	定数式の記述にエラーがあります。アドレスを割り付けたラベルを使用している場合などが考えられます。
`syntax error in directive operand`	擬似命令のオペランド指定に誤りがあります。
`syntax error in expression`	式に文法エラーがあります。
`syntax error in macro body`	マクロ置換除外の記述に誤りがあります。
`syntax error in macro operand`	マクロ定義行の仮引数、初期値の表記に誤りがあります。
`too complex expression in operand` *n*	あまりに複雑な式（第 *n* オペランド内）なので処理できません。
`too large reserve area`	領域確保サイズの指定値が大きすぎます。32 ビットの論理アドレス空間を越えています。

表7.4 アセンブラのエラーメッセージ (8/9)

メッセージ	解説
too long command line	コマンド行の文字数が長すぎます。
too long string	文字列が長過ぎます。式で文字定数では 4 文字以内、擬似命令 .SDATAB では 242 文字以内でなければなりません。
too many arguments	マクロコールでの引数指定が多すぎます。
too many source files	ソースファイル数が多すぎます。
trap number should be a constant	トラップ番号は、定数式で指定しなければなりません。
unexpected end-of-file	文法エラー。予期しないファイル終端が出現しました。
unexpected end-of-line	文法エラー。予期しない行末が出現しました。
unexpected *token*, required symbol	シンボルの指定が必要なところに 文字列 *token* が出現しました。.EXPORT, .IMPORT, .GLOBAL のオペランドは、シンボルを , でつないだものです。
unknown directive .*token*	文字列 .*token* は存在しない擬似命令です。
unknown instruction *token*	文字列 *token* は存在しない命令です。
unknown size .*token*	.*token* は存在しないオペランドサイズです。
value : overflow for byte	バイトサイズデータに対するオーバーフローが発生しました。式の値は *value* となりました。
value : overflow for halfword	ハーフワードサイズデータに対するオーバーフローが発生しました。式の値は *value* となりました。

表7.4 アセンブラのエラーメッセージ (9/9)

メッセージ	解説
value : overflow for word	ワードサイズデータに対するオーバーフローが発生しました。式の値は *value* となりました。
"*variable*" : preprocess value is not defined	マクロボディ中、または式中に定義されてないプリプロセッサ変数 *variable* があります。
zero division in the expression	0 除算が行われています。
instructionplacedonoddlocation	命令のアドレス（ロケーションカウンター）が奇数になる命令を記述しています。
floatingpointnumberoverflow	浮動小数点数の値がオーバーフローしました。表現できる最大値を設定しました。
floatingpointnumberunderflow	浮動小数点数の値がアンダーフローしました。値をゼロに設定しました。
toomanydigitsinfloatingpointnumber;extradigitsignored	浮動小数点数の指数部の値がオーバーフローしました。表現できる最大値を設定しました。
toomanydigitsinfloatingpointnumber;extradigitsignored	浮動小数点数の指数部の値がオーバーフローしました。表現できる最大値を設定しました。

7.2.3 重大エラー

表7.5 アセンブラの重大エラーメッセージ

メッセージ	解説
`can't create default section`	デフォルトのセクションを生成できません。アセンブル処理を中止します。
`"`*file_name*`" : can't close file`	ファイル *file_name* をクローズできません。
`"`*file_name*`" : can't open file`	ファイル *file_name* をオープンできません。
`"`*file_name*`" : cannot delete file`	テンポラリファイル *file_name* を削除できません。
`out of heap space`	メモリが不足しています。
`out of memory`	メモリ不足になりました。使用できるメモリを増やす、ファイルを分割するなどの対策をして下さい。
`too many errors ! Good bye !`	多くのエラーを検出しました。アセンブル処理を中止します。
`"`*tool*`" : can't execute file`	ツール *tool* が起動できません。

付録A
一般命令一覧

本付録では、一般命令（M32R 命令セット）の概要を、機能グループ別に示します。グループは次の 6 つに大別されます。

- ロード／ストア命令
- 転送命令
- 演算命令（比較、算術演算、論理演算、シフト）
- 分岐命令
- EIT 関連命令　　※ EIT :Exception,Interrupt,Trap
- DSP 機能用命令

オペランド表記は、以下の記述規則に従っています。

表記	意味
R*n*	任意の汎用レジスタ　（*n*=0～15）
CR*n*	任意の制御レジスタ。
An	任意のアキュムレータ。（*n*=0, 1）
@R*n*	任意の汎用レジスタの内容（アドレス）が示すメモリの内容。
@R*n*+	汎用レジスタR*n*参照（レジスタ間接）後、Rnの内容が＋4される（レジスタ更新）ことを示す。
@+R*n*	汎用レジスタR*n*参照（レジスタ間接）前に、Rnの内容が＋4される（レジスタ更新）ことを示す。
@-R*n*	汎用レジスタR*n*参照（レジスタ間接）前に、Rnの内容が－4される（レジスタ更新）ことを示す。
Rsrc,Rsrc*n*	参照する汎用レジスタ（処理対象のアドレスや値をもつ）
CRsrc	参照する制御レジスタ
Rdst,CRdest	ディスティネーションレジスタ
disp_*n*	*n*ビットディスプレースメント値
imm_*n*	*n*ビット符号付き整数即値
label_*n*	分岐先に置いたラベル（nはディスプレースメントサイズ）

各命令の詳細については、M32R のソフトウェアマニュアルを参照してください。

A.1 M32R命令一覧

■ ロード／ストア命令

分類	ニーモニック	オペランド	機能概要
ロード／ストア命令	LD	Rdest, @Rsrc	メモリ→レジスタ（32ビットデータ転送）
	LD	Rdest, @(disp16, Rsrc)	
	LD	Rdest, @(LOW(disp), Rsrc)	
	LD	Rdest, @Rsrc+	
	LDB	Rdest, @Rsrc	メモリ→レジスタ（8ビットデータ符号拡張転送）
	LDB	Rdest, @(disp16, Rsrc)	
	LDB	Rdest, @(LOW(disp), Rsrc)	
	LDH	Rdest, @Rsrc	メモリ→レジスタ（16ビットデータ符号拡張転送）
	LDH	Rdest, @(disp16, Rsrc)	
	LDH	Rdest, @(LOW(disp), Rsrc)	
	LDUB	Rdest, @Rsrc	メモリ→レジスタ（8ビットデータゼロ拡張転送）
	LDUB	Rdest, @(disp16, Rsrc)	
	LDUB	Rdest, @(LOW(disp), Rsrc)	
	LDUH	Rdest, @Rsrc	メモリ→レジスタ（16ビットデータゼロ拡張転）
	LDUH	Rdest, @(disp16, Rsrc)	
	LDUH	Rdest, @(LOW(disp), Rsrc)	
	LOCK	Rdest, @Rsrc	メモリ→レジスタ転送（バスロック付き）
	ST	Rsrc1, @Rsrc2	レジスタ→メモリ（32ビットデータ転送）
	ST	Rsrc1, @(disp16, Rsrc2)	
	ST	Rsrc1, @(LOW(disp), Rsrc2)	
	ST	Rsrc1, @+Rsrc2	
	ST	Rsrc1, @-Rsrc2	
	STB	Rsrc1, @Rsrc2	レジスタ→メモリ（8ビットデータ転送）
	STB	Rsrc1, @(disp16, Rsrc2)	
	STB	Rsrc1, @(LOW(disp), Rsrc2)	
	STH	Rsrc1, @Rsrc2	レジスタ→メモリ（16ビットデータ転送）
	STH	Rsrc1, @(disp16, Rsrc2)	
	STH	Rsrc1, @(LOW(disp), Rsrc2)	
	UNLOCK	Rsrc1, @Rsrc2	レジスタ→メモリ転送（バスのアンロック付き）

■ 転送命令

分類	ニーモニック	オペランド	機能概要
転送命令	LD24	Rdest, #imm24	即値→レジスタ（24ビットデータ0拡張転送）
	LDI	Rdest, #imm8	即値→レジスタ（8ビットデータ符号拡張転送）
	LDI	Rdest, #imm16	即値→レジスタ（16ビットデータ符号拡張転送）
	MV	Rdest, Rsrc	レジスタ間のデータ転送
	MVFC	Rdest, CRsrc	制御レジスタ→レジスタ転送
	MVTC	Rdest, CRsrc	レジスタ→制御レジスタ転送
	SETH	Rdest, #imm16	即値→レジスタの上位16ビット
	SETH	Rdest, #HIGH(imm)	
	SETH	Rdest, #SHIGH(imm)	

■ 演算命令

分類	ニーモニック	オペランド	機能概要
比較命令	CMP	Rsrc1, Rsrc2	比較(レジスタ間)
	CMPI	Rsrc, #imm16	比較(即値とレジスタ)
	CMPU	Rsrc1, Rsrc2	比較(レジスタ間)
	CMPUI	Rsrc, #imm16	比較(即値とレジスタ)
算術演算命令	ADD	Rdest, Rsrc	加算(レジスタ間)
	ADD3	Rdest, Rsrc, #imm16	加算(16ビット即値と レジスタ)
	ADDI	Rdest, #imm8	加算(8ビット即値とレジスタ)
	ADDV	Rdest, Rsrc	加算(レジスタ間)
	ADDV3	Rdest, Rsrc, #imm16	加算(16ビット即値と レジスタ)
	ADDX	Rdest, Rsrc	キャリー付き加算
	NEG	Rdest, Rsrc	0の補数の算出
	SUB	Rdest, Rsrc	減算
	SUBV	Rdest, Rsrc	減算(オーバーフロー付き)
	SUBX	Rdest, Rsrc	減算(ボロー付き)
	DIV	Rdest, Rsrc	除算(32ビット符号付き整数の除算)
	DIVU	Rdest, Rsrc	除算(32ビット符号なし整数の除算)
	MUL	Rdest, Rsrc	乗算(16ビット符号つき整数の乗算)
	REM	Rdest, Rsrc	符号付き除算の剰余
	REMU	Rdest, Rsrc	符号なし除算の剰余
論理演算命令	AND	Rdest, Rsrc	論理積(レジスタ間)
	AND3	Rdest, Rsrc, #imm16	論理積(即値とレジスタ)
	NOT	Rdest, Rsrc	ビット反転
	OR	Rdest, Rsrc	論理和(レジスタ間)
	OR3 OR3	Rdest, Rsrc, #imm16 Rdest, Rsrc, #LOW(imm)	論理和(即値とレジスタ)
	XOR	Rdest, Rsrc	排他的論理和(レジスタ間)
	XOR3	Rdest, Rsrc, #imm16	排他的論理和(即値とレジスタ)
シフト命令	SLL	Rdest, Rsrc	左シフト(シフト数はRsrcの下位5ビット)
	SLL3	Rdest, Rsrc, #imm16	左シフト(シフト数は即値の下位5ビット)
	SLLI	Rdest, #imm5	左シフト(シフト数は即値の下位5ビット)
	SRA	Rdest, Rsrc	算術右シフト(シフト数はRsrcの下位5ビット)
	SRA3	Rdest, Rsrc, #imm16	算術右シフト(シフト数は即値の下位5ビット)
	SRAI	Rdest, #imm5	算術右シフト(シフト数は即値の下位5ビット)
	SRL	Rdest, Rsrc	論理右シフト(シフト数はRsrcの下位5ビット)
	SRL3	Rdest, Rsrc, #imm16	論理右シフト(シフト数は即値の下位5ビット)
	SRLI	Rdest, #imm5	論理右シフト(シフト数は即値の下位5ビット)

■ 分岐命令

分類	ニーモニック	オペランド	機能概要
分岐命令	BC	label_8	コンディションフラグの値が1 のとき分岐
	BC	label_24	
	BEQ	Rdst, Rsrc,label_16	RsrcとRdest が等しいとき分岐
	BEQZ	Rsrc, label_16	Rsrcが0のとき分岐
	BGEZ	Rsrc, label_16	Rsrcが0以上のとき分岐
	BGTZ	Rsrc, label_16	Rsrcが0より大きい分岐
	BL	label_8	リターンアドレス設定後分岐
	BL	label_24	
	BLEZ	Rsrc, label_16	Rsrcが0以下のとき分岐
	BLTZ	Rsrc, label_16	Rsrcが0より小さいとき分岐
	BNC	label_8	コンディションフラグが1でないとき分岐
	BNC	label_24	
	BNE	Rdest, Rsrc, label_16	Rdest とRsrc が等しくないとき分岐
	BNEZ	Rsrc, label_16	Rsrcが0でないとき分岐
	BRA	label_8	無条件相対分岐
	BRA	label_24	
	JL	Rsrc	リターンアドレス設定後、絶対アドレスへ分岐
	JMP	Rsrc	指定絶対アドレスへ分岐
	NOP		何もしない

■ EIT関連命令

分類	ニーモニック	オペランド	機能概要
EIT関連命令	RTE		EITからの復帰
	TRAP	#imm_4	トラップの発生

■ DSP機能用命令

分類	ニーモニック	オペランド	機能概要
DSP機能用命令	MACHI	Rsrc1, Rsrc2	積和演算(レジスタ×レジスタ→アキュムレータ)
	MACLO	Rsrc1, Rsrc2	
	MACWHI	Rsrc1, Rsrc2	
	MACWLO	Rsrc1, Rsrc2	
	MULHI	Rsrc1, Rsrc2	乗算(レジスタ×レジスタ→アキュムレータ)
	MULLO	Rsrc1, Rsrc2	
	MULWHI	Rsrc1, Rsrc2	
	MULWLO	Rsrc1, Rsrc2	
	MVFACHI	Rdest	アキュムレータ→レジスタ間転送
	MVFACLO	Rdest	
	MVFACMI	Rdest	
	MVTACHI	Rsrc	レジスタ→アキュムレータ間転送
	MVTACLO	Rsrc	
	RAC		アキュムレータの丸め
	RACH		

A.2 M32Rx/Dシリーズ拡張命令一覧

A.2.1 M32Rx新規拡張命令

M32Rx/D シリーズで、M32R ファミリ命令セットから新規に追加された命令の一覧を以下に示します。

■ M32Rx新規拡張命令一覧

分類	ニーモニック	オペランド	機能概要
比較命令	CMPEQ	Rsrc1, Rsrc2	比較(レジスタ間)
	CMPZ	Rsrc	比較(レジスタと即値0(ゼロ))
算術演算命令	DIVH	Rdest, Rsrc	除算(16ビット符号付き整数の除算)
分岐命令	BCL	pcdisp8 or pcdisp24	条件ビット(C)が 1 のとき分岐し、戻り先をR14に格納
	BNCL	pcdisp8 or pcdisp24	条件ビット(C)が 0 のとき分岐し、戻り先をR14に格納
DSP機能用命令	MACLH1	Rsrc1, Rsrc2	積和演算(レジスタ × レジスタ + アキュムレータA1 → アキュムレータA1)
	MACWU1	Rsrc1, Rsrc2	積和演算(レジスタ × レジスタ + アキュムレータA1 → アキュムレータA1)
	MSBLO	Rsrc1, Rsrc2	積和演算(レジスタ × レジスタ - アキュムレータA1 → アキュムレータA1)
	MULWU1	Rsrc1, Rsrc2	乗算(レジスタ × レジスタ → アキュムレータA1)
	SADD		加算(アキュムレータA0 + アキュムレータA1 → アキュムレータA0)
	SATB	Rdest, Rsrc	レジスタのデータに対するバイトサイズの丸め
	SATH	Rdest, Rsrc	レジスタのデータに対するハーフワードサイズの丸め

注 . アキュムレータ ACC0、ACC1 のニーモニックは A0、A1 で指定するため、上記の表中では A0、A1 で表記しています。

A.2.2 M32Rx仕様拡張命令

M32Rx/D シリーズで、M32R ファミリ命令セットから仕様が拡張された命令の一覧を以下に示します。

■ M32Rx仕様拡張命令一覧

分類	ニーモニック	オペランド	機能概要
DSP機能用命令	MACHI	Rsrc1, Rsrc2, Adest	アキュムレータを 2本に拡張したことに伴い、オペランド表記でアキュムレータA0、A1 を指定可能
	MACLO	Rsrc1, Rsrc2, Adest	
	MULHI	Rsrc1, Rsrc2, Adest	
	MULLO	Rsrc1, Rsrc2, Adest	
	MVFACHI	Rdest, Asrc	
	MVFACLO	Rdest, Asrc	
	MVFACMI	Rdest, Asrc	
	MVTACHI	Rsrc, Adest	
	MVTACLO	Rsrc, Adest	
	RAC	Adest,Asrc,Ó#imm1	アキュムレータを 2本に拡張したことに伴い、オペランド表記でアキュムレータA0、A1を指定可能 また、アキュムレータをイミディエート(imm1)で指定したビットの左シフト実行後の値が、丸め操作に反映される
	RACH	Adest,Asrc,Ó#imm1	

注 . アキュムレータ ACC0、ACC1 のニーモニックは A0、A1 で指定するため、上記の表中では A0、A1 で表記しています。

付録B
擬似命令リファレンス

本付録では、アセンブラの擬似命令をアルファベット順に説明しています。記述形式は以下のとおりです（図B.1）。

ニーモニック **分類**

機能概要

書式 擬似命令のソースプログラム中への記述方法を示します。

シンボル	擬似命令	オペランド
シンボルフィールド	オペレーションフィールド	オペランドフィールド

解説 擬似命令の機能を説明します。

記述例 擬似命令の記述例を示します。

図B.1 擬似命令リファレンスの記述形式

シンボルフィールドに何も記述がなければ、シンボルフィールド内にシンボルは記述できません。各フィールドの間は1つ以上のスペース（空白文字）を入れて区切る必要があります。

本章では表記上、以下に示す記号を使用しています（表B.1）。

表B.1 擬似命令リファレンスの記述規則

記号	意味
[]	[ ]内の内容が省略できます。
.size	サイズ指定 をします。

.ALIGN

アドレス制御命令

境界を調整します。

書式

	.ALIGN	式

式　　：ロケーションカウンタ調整値
　　式2^n (n=0,1,2,...,31)

解説

.ALIGN 擬似命令は、現在のロケーションカウンタ値が .SECTION 擬似命令で指定した境界にない場合、式で示す境界値になるまでロケーションカウンタ値を進めます。現在のロケーションカウンタ値が指定の境界にある場合は何も行いません。

本擬似命令における、式の指定規則を以下に示します。

- 式には以下の条件を満たすロケーションカウンタ調整値を指定します。
  1～2^{31} の範囲内の絶対値、かつ、2 を n 乗した値
  .SECTION 擬似命令で宣言したロケーションカウンタ調整値以下の値
- 式には定数式を記述します。
- 式の中の項に使用するシンボルは、本擬似命令以前に定義済みでなければなりません。
- .ALIGN 擬似命令で宣言したロケーションカウンタ調整値の大きさが、.SECTION 擬似命令の ALIGN= 式で宣言したロケーションカウンタ調整値の大きさ以下でなければエラーとなります。

記述例

```
        .ALIGN     4
```

.ASSIGN

シンボル定義命令

変更可能値シンボルを定義します。

書式

シンボル	.ASSIGN	式

式　　：シンボル値

解説

.ASSIGN擬似命令は、シンボルフィールドで指定したシンボルに、式の値を割り当てます。

本擬似命令で定義した値シンボルを「変更可能値シンボル」と呼びます。取り扱いは以下のとおりです。

- ASSIGN擬似命令で値を変更できる。
- 外部定義シンボルとしては使用できない。
- デバッグ情報は出力されない。

本擬似命令における、式の指定規則を以下に示します。

- 式には以下の値を指定できます。
  絶対値
  正の相対値（ただし、相対値はただ1つしか記述できません）
- 式の中の項に使用するシンボルは、本擬似命令以前に定義済みでなければなりません。

記述例

```
COUNT:    .ASSIGN    h'1084
```

.DATA

データ設定命令

定数（整数）を確保します。

書式

[シンボル]	.DATA[.size]	式[,式]…

.size ：サイズ指定　　.B（バイト　= 8 ビット）
　　.H（ハーフワード = 16ビット）
　　.W（ワード = 32 ビット。デフォルト）
式　　：整数設定値

解説

.DATA 擬似命令は、size で指定した大きさのデータ領域を確保し、その領域内に式の値（整数）を設定します。サイズ指定は、.B（バイト =8 ビット）、.H（ハーフワード =16 ビット）、.W（ワード =32 ビット）で行います。サイズ指定がない場合、.W（ワード =32 ビット）指定となります。

本擬似命令における、式の指定規則を以下に示します。

- 式の値は絶対値でも相対値でもかまいません。また、符号付き整数、および、符号なし整数が指定できます。
- 式の値はサイズ指定の範囲内で表現できる値でなければなりません。

記述例

```
TABLE:    .DATA.H   h'12, h'35A8
```

.DATAB

データ設定命令

定数（整数）ブロックを確保します。

書式

[シンボル]	.DATAB[.size]	式a,式b

.size ：サイズ指定 .B(バイト = 8ビット)
.H(ハーフワード =16ビット)
.W(ワード =32ビット。デフォルト)

式a ：ブロック確保値

式b ：整数設定値

解説

.DATAB擬似命令は、.sizeで指定した大きさのデータ領域を、式aで示す数の分だけ確保し、それぞれの領域内に、式bで示す整数を設定します。サイズの指定は、.B（バイト=8ビット)、.H（ハーフワード=16ビット)、.W（ワード=32ビット）で行います。サイズの指定がない場合は.W(ワード=32ビット)指定となります。

本擬似命令における、式の指定規則を以下に示します。

- 式aの値は定数式で、その値は、絶対値、かつ、0以上の整数でなければなりません。また、式aの項として使用するシンボルは、本擬似命令以前で定義されていなければなりません。
- 式bには、符号付き整数または符号なし整数が指定できます。式bの値はサイズ指定の範囲内でなければなりません。

記述例

```
TABLE:    .DATAB.W   10, h'48153CD
```

.END

プログラム構造定義命令

ソースプログラムの終了を示します。

書式

	.END	[式]

式　　：プログラム開始アドレス（エントリポイント）の指定

解説

.END 擬似命令は、ソースプログラムの終了を示します。本擬似命令以降にソースプログラムがある場合、その部分は無視されますが、エラーとはなりません。

オペランドフィールドに式を記述することによって、エントリポイントが指定できます。エントリポイントとは、プログラムの開始アドレスを指示するための情報です。

本擬似命令における、式の指定規則を以下に示します。

- 式の値はソースプログラム中のアドレスでなければなりません。
- 式には以下のものが記述できます。
  - 絶対値
  - 正の相対値（ただし、相対値は1つしか記述できません）
- 式を省略するとエントリポイントは設定されません。

エントリポイントは、CODE セクション内のアドレスでなければなりません。

注意

複数のモジュールがエントリポイント情報を持つ場合、リンカ時にエラーとなります。

記述例

```
.END  LABEL
```

.EQU

シンボル定義命令

値シンボルを定義します。

書式

シンボル	.EQU	式

式　　：　シンボル値

解説

.EQU 擬似命令は、シンボルフィールドで指定したシンボルに式の値を割り当てます。同一シンボルは二回以上定義できません。

本擬似命令における、式の指定規則を以下に示します。

- 式には以下のものが記述できます。
  - 定数
  - 正の相対値（ただし、相対値は1つしか記述できません）
- 式の中の項に使用するシンボルは、本擬似命令以前に定義済みでなければなりません。

記述例

```
SYMBOL:    .EQU h'D51
```

.EXPORT

シンボル外部定義/外部参照命令

外部定義シンボルを宣言します。

書式

	.EXPORT	シンボル[,シンボル]...

解説

.EXPORT 擬似命令は、モジュール内で定義したシンボルを、他のモジュールで参照できるシンボルとして宣言します。

シンボルは、以下の条件をすべて満たすもののみ指定できます。

- 絶対値、あるいは、ソースプログラム中のアドレス値を持つ。
- 当該モジュール中で定義されている。
- .ASSIGN 擬似命令で値を定義したシンボルでない。

本擬似命令は、ソースプログラム中の何行目に記述してもかまいません。

記述例

```
        .EXPORT    LABEL0, SYMBOL0
```

.GLOBAL

シンボル外部定義/外部参照命令

外部定義および外部参照シンボルを宣言します。

書式

	.GLOBAL	シンボル[,シンボル]...

解説

.GLOBAL 擬似命令には、以下の二つの機能があります。

- 外部定義シンボルの宣言　　モジュール内で定義されたシンボルを、他のモジュールで参照できるシンボルとして宣言します。
- 外部参照シンボルの宣言　　他のモジュールで定義されているシンボルを参照する際、そのシンボルが外部参照であることを宣言します。

本擬似命令における、シンボルの指定規則を以下に示します。

- 絶対値、あるいは、ソースプログラム中のアドレス値を持つシンボルのみ指定できます。
- .ASSIGN 擬似命令で値を定義したシンボルは外部定義シンボルとして宣言できません。
- 指定したシンボルは、当該モジュール中で定義されていれば外部定義シンボル、定義されていなければ外部参照シンボルとみなされます。

本擬似命令は、ソースプログラム中の何行目に記述してもかまいません。
本擬似命令は、擬似命令 .IMPORT、 .EXPORT の代わりに使用できます（同等の機能を持ちます）。

記述例

```
        .GLOBAL    EXTLAB, IMPSYM
```

.IMPORT

シンボル外部定義/外部参照命令

外部参照シンボルを宣言します。

書式

	.IMPORT	シンボル[,シンボル]

解説

.IMPORT 擬似命令は、他のモジュール中で定義されているシンボルを参照する際、そのシンボルが外部参照であることを宣言します。

本擬似命令における、シンボルの指定規則を以下に示します。

- 宣言するシンボルは、他のモジュール内で .EXPORT 擬似命令、あるいは .GLOBAL 擬似命令で定義されていなければなりません。
- 同じモジュール内で定義済みのシンボルは、本擬似命令で宣言できません。

本擬似命令は、ソースプログラム中の何行目に記述してもかまいません。

記述例

```
        .IMPORT    EXTLAB, EXTSYM
```

.PROGRAM

プログラム構造定義命令

モジュール名を指定します。

書式

	.PROGRAM	モジュール名

解説

.PROGRAM 擬似命令は、モジュール名を指定します。本擬似命令で指定したモジュール名は、そのままロードモジュールまで受け継がれます。デバッガは、デバッグ対象のロードモジュールを指定するときに、本擬似命令で指定したモジュール名を用います。

本擬似命令を省略した（宣言しなかった）場合、アセンブラが生成するオブジェクトモジュール名から拡張子（.mo）部分を削除したものがモジュール名となります。ただし、オブジェクトモジュール名が名前の規則に従っていない場合、最初のピリオド（.）までがモジュール名となります。
（例： A.B.C.D の場合、モジュール名はAとなる）。

本擬似命令における、モジュール名の指定規則を以下に示します。

- モジュール名は名前の規則に従って記述します。名前の記述規則は、3.5「名前の記述規則」を参照してください。
- モジュール名に使った名前は、プログラム中でモジュール名以外の名前としても使用できます。

本擬似命令は、1つのソースファイル中で1回限り有効です。
本擬似命令はソースプログラム中の何行目にあってもかまいません。

記述例

```
.PROGRAM   MAIN
```

.RES

領域確保命令

定数（整数）領域を確保します。

書式

[シンボル]	.RES[.size]	式

.size ： サイズ指定 .B(バイト = 8ビット)
.H(ハーフワード =16ビット)
.W(ワード =32ビット。デフォルト)

式 ： ブロック確保値

解説

.RES 擬似命令は、サイズ指定した大きさのデータ領域を、式で示す数の分だけ確保します。サイズの指定は、.B(バイト =8 ビット)、.H(ハーフワード =16 ビット)、.W(ワード =32 ビット)で行います。サイズ指定を省略した場合は .W(ワード =32 ビット)指定となります。

本擬似命令における、式の指定規則を以下に示します。

・ 式の値は次の条件をすべて満たすものでなければなりません。
　絶対値
　0 以上の整数

・ 式の中の項で使用するシンボルは、本擬似命令以前で定義済みでなければなりません。

記述例

```
WORK:.RES.B     20
```

.SDATA

データ設定命令

文字列を確保します。

書式

[シンボル]	.SDATA	文字列[,文字列]

解説

.SDATA 擬似命令は、データ領域を確保し、その領域内に文字列データを設定します。

本擬似命令における、オペランドの文字列の指定規則を以下に示します。

- 文字列は、ダブルクォーテーション(")で囲まれたASCIIコードの文字列、および、< と > で囲まれた文字コードで構成されます。
- 文字列中にダブルクォーテーション(")を含める場合は、" "と2回続けて記述します。

記述例

```
TABLE:    .SDATA    "HELLO", "WORLD"
```

.SDATAB

データ設定命令

文字列ブロックを確保します。

書式

[シンボル]	.SDATAB	式 , 文字列

式　　：　ブロック確保数

解説

.SDATAB 擬似命令は、文字列データ領域を式で示す数の分だけ確保し、それぞれの領域内に文字列データを設定します。

本擬似命令における、式の指定規則を以下に示します。

- 式の値は次の条件をすべて満たすものでなければなりません。
  絶対値
  0以上の整数
- 式の中の項に使用するシンボルは、本擬似命令以前に定義済みでなければなりません。

本擬似命令における、オペランドの文字列の指定規則を以下に示します。

- 文字列は、ダブルクォーテーション(")で囲まれたASCIIコードの文字列、および、< と > で囲まれた文字コードで構成されます。
- 文字列中にダブルクォーテーション(")を含める場合は、" "と2回続けて記述します。

記述例

```
TABLE:    .SDATAB   20, "HELLO"
```

.SECTION

プログラム構造定義命令

セクションを指定します。

書式

	.SECTION	セクション名[,属性a][,属性b]

属性a ： CODE | COMMON | DATA | DUMMY | STACK　セクション属性（デフォルト：CODE）
属性b ： ALIGN=式 | LOCATE=式　配置属性 （デフォルト：ALIGN=4）

解説

.SECTION擬似命令は、以下の指定によってセクション宣言を行います。

- セクション名の指定。
- セクションの実行可能／不可能の宣言、および、リンカに対する結合方法の指示（セクション属性の指定）。
- セクションの配置方法の指定（配置属性の指定）。

セクションのリンク（結合、配置）について詳しくは、「CC32Rユーザーズマニュアル3 <<アセンブラ編>>」のリンカの章を参照してください。

■ デフォルトのセクション指定

デフォルトのセクション指定は、.SECTION　P,CODE,ALIGN=4（セクション名P、セクション属性CODE、配置属性ALIGN=4）です。

ソースプログラムの先頭から、最初の.SECTION擬似命令までの間に、以下のような命令が記述されている場合、アセンブラはデフォルトのセクションを生成します。

- 一般命令や領域確保命令などのオブジェクトコードを生成する命令
- ロケーションカウンタを更新する擬似命令

■ セクションの継続

同一ソースプログラム内に、セクション名指定が同一である.SECTION擬似命令が複数ある場合、それら同じ名前のセクションは連続した1つのセクションとみなされます。この場合、最初の.SECTION擬似命令がセクションの始まりを表し、それ以外はセクションの継続を表します。セクションが継続する場合のロケーションカウンタは、

直前の同一名セクションの終わりのロケーションカウンタ＋1

を示します。

■ セクションの大きさ

セクションの大きさは、同一名セクション内の最大ロケーション値となります。
同一セクション名が指定された複数の .SECTION 擬似命令どうしでは、異なる属性を指定するなどの矛盾があってはなりません。
本擬似命令における、セクション名の指定規則を以下に示します。

・ セクション名は名前の規則に従って記述します。

■ セクションの指定規則

本擬似命令における、属性の指定規則を以下に示します。

・ セクション属性（属性a）と配置属性（属性b）の指定順序にとくに規定はありません。
・ セクション属性（属性a）と配置属性（属性b）には、それぞれ1つだけ属性を指定できます。

各属性指定の詳細を以下に示します。

・ セクション属性（属性a）について

セクション属性では、セクションの実行可能／不可能の宣言、および、リンカに対するセクションの結合方法の指示を行います。セクション属性を省略した場合のデフォルト属性は CODE です。
下表に、指定できるセクション属性とその内容を示します。

セクション属性（意味）	指定内容
CODE（コードセクション）	実行可能属性であることを宣言します（実行が可能なセクションは、CODE セクションのみです）。 リンカに単純結合を指示します。
DATA（データセクション）	実行不可能属性であることを宣言します。 リンカに単純結合を指示します。
STACK（スタックセクション）	実行不可能属性であることを宣言します。 リンカに単純結合を指示します。
COMMON（コモンセクション）	実行不可能属性であることを宣言します。 リンカに共有結合を指示します。
DUMMY（ダミーセクション）	実行不可能属性であることを宣言します。 配置属性（属性b）は指定できません。 ダミーセクションは、アセンブル処理は行われますが、オブジェクトコードは出力されません。 ダミーセクション内で定義したシンボルは、セクション先頭を0としたロケーションカウンタ値が設定され、絶対値をもつシンボルとして扱われます。

セクションの結合形式には単純結合と共有結合の2種類があります。下表に各結合形式について示します。

結合形式	リンカによる結合および取り扱い
単純結合	同一名セクションをすべて結合し、1つの連続したセクションと見なします。セクション配置時のアライメント（境界調整値）は、.SECTION 擬似命令で指定したそれぞれの配置属性に従います。
共有結合	他のモジュール内の同じ名前を持つ、セクションおよびメモリを共有します。リンクする同一名セクションは、属性もも同じでなければなりません。結合されたセクション内で最大のもののサイズを、結合後のセクションの大きさと見なします。

※補足　ダミーセクションについて

ダミーセクションは、構造体データを用いる際に使用する特殊なセクションです。構造体データのメンバを表すシンボルを宣言する際に利用します。以下に宣言例を示します。

```
        .SECTION  ABC, DUMMY
DT0:  .RES.W    1
DT1:  .RES.H    1
DT2:  .RES.H    1
```

上記のようにダミーセクション内で定義したラベルシンボルには、セクションの先頭からのオフセット値（絶対値）が割り当てられます。したがって上記のプログラムは、擬似命令 .EQU で下記ように定義した場合と同じ意味になります。

```
DT0:  .EQU     0
DT1:  .EQU     4
DT2:  .EQU     6
```

ダミーセクションを用いると、容易に構造体メンバの参照、設定、追加、および削除ができます。例えば、上記の構造体のメンバを参照し、そこへ整数を設定する場合、以下のように記述します（構造体データの先頭ラベルが STRU であると仮定します）。

```
LD24  R0,#STRU
LDI   R1,#10
ST    R1,@( DT0, R0 )
LDI   R1,#20
STH   R1,@( DT1, R0 )
SDL   R1,#30
STH   R1,@( DT2, R0 )
```

・ 配置属性（属性b）について

配置属性では、セクションの相対形式／絶対形式の宣言、および、リンカに対するセクションの配置方法の指示を行います。配置属性を省略した場合のデフォルト属性はALIGN=4（バイト）で、相対形式セクションとなります。
以下に、指定できる配置属性とその内容を示します。

配置属性の指定形式	指定内容
ALIGN=式	相対形式セクションであること、および、メモリ上での配置方法（境界調整方法）をリンカに指示します。 式は境界を調整する位置を示します。式の指定規則を以下に示します。 ・ この値は次の条件を満たすものでなければなりません。 絶対値 1～2^{31}の範囲で、2をn乗した値 ただし、CODEセクションの場合、次の条件になります。 絶対値 4～2^{31}の範囲で、2をn乗した値 ・ 式の中の項に使用するシンボルは、本擬似命令以前で定義済みでなければなりません。
LOCATE=式	絶対形式セクションであることをリンカに指示します。また、アセンブルの結果、式で指定したアドレスにセクションを配置します。絶対形式セクションは、リンカで再配置することはできません。 リンク時、他のソースプログラム内に同一セクション名のセクションがあってはなりません。 式は絶対アドレスを示します。式の指定規則を以下に示します。 ・ 式の値は絶対値でなければなりません。 ・ 式の中の項に使用するシンボルは、本擬似命令以前で定義済みでなければなりません。

記述例

```
.SECTION   ABC, CODE, ALIGN=4
.SECTION   ABC, COMMON,ALIGN=4
.SECTION   P,CODE,ALIGN=4
.SECTION   D,DATA,ALIGN=4
```

付録C
マクロ命令リファレンス

本付録では、アセンブラのマクロ処理用の各マクロ命令および文字列処理関数をアルファベット順に説明しています。記述形式は以下のとおりです（図C.1）。

ニーモニックまたは関数名 **分類**
機能概要

書式 マクロ命令のソースプログラム中への記述方法を示します。

シンボル	マクロ命令	オペランド
シンボルフィールド	オペレーションフィールド	オペランドフィールド

文字列操作関数の記述にはこのようなフィールド別は関係ありません。

解説 マクロ命令および文字列操作関数の機能を説明します。ここで用いる「文」とは、1つ以上の命令または擬似命令で構成されるものを指します。

記述例 マクロ命令および文字列操作関数の記述例を示します。

図C.1 マクロ命令リファレンスの記述形式

シンボルフィールドに何も記述がなければ、シンボルフィールド内にシンボルは記述できません。各フィールドの間は1つ以上のスペース（空白文字）を入れて区切る必要があります。

本章では表記上、以下に示す記号を使用しています（表C.1）。

表C.1 マクロ命令リファレンスの記述規則

記号	意味
[]	[ ]内の内容が省略できます。

.AIF .AELSE .AENDI

マクロ命令

展開を選択します。

書式

	.AIF	論理式
	[.AELSE]	
	.AENDI	

解説

論理式の評価により展開対象を選択します。本構文は以下のように記述し、マクロ定義を行います。

```
.AIF 論理式
        [文1]
[.AELSE]
        [文2]
.AENDI              ;.AENDI 命令により、.AIF 構文の終了を示します（省略不可）
```

as32Rは、論理式の評価を行って、真であれば 文1 を、偽であれば文2 を展開し、その後本構文を抜けます。文1、文2はそれぞれ省略可能であり、省略した場合は何も展開しません。.AELSE 命令を省略した場合、論理式が偽であれば、何も展開せずに本構文を抜けます。

論理式に誤りがあればエラーとなり、論理式は偽と評価されます。論理式の記述は6.3.2.3「論理式」に従ってください。

記述例

```
        .MACRO  MCRIF   ARG_1
        .AIF  .LEN("\ARG_1") EQ \&AVAR_1
              ADDI  R\&AVAR_1,#1
        .AELSE
              ADDI  R\&AVAR_1,#2
        .AENDI
        .ENDM


AVAR_1: .ASSIGNA     5
      MCRIF ABCDE
```

＜マクロ展開結果＞

```
      ADDI  R5,#1
```

.AREPEAT .AENDR

マクロ命令

回数指定により展開を繰り返します。

書式

	.AREPEAT	算術式

	.AENDR	

解説

算術式の評価により展開を繰り返します。本構文は以下のように記述し、マクロ定義を行います。

```
    .AREPEAT   算術式
         [文]
    .AENDR          ;.AENDR 命令により、.AREPEAT 構文の終了を示します（省略不可）
```

as32R は、算術式の値を計算し、算術式で表す回数だけ文を繰り返し展開し、本構文を抜けます。文は省略可能であり、省略した場合は何も展開しません。算術式が 0 の場合、何も展開せずに本構文を抜けます。

算術式が負の場合、エラーとなり、何も展開されません。算術式に誤りがある場合もエラーとなり、何も展開されません。算術式の記述は6.3.2.1「算術式」に従ってください。

記述例

```
            .MACRO     MCRRE
            .AREPEAT   \&AVAR_1
                 ADDI  R5,#\&AVAR_2
AVAR_2:     .ASSIGNA   \&AVAR_2 + 1
            .AENDR
            .ENDM


AVAR_1:     .ASSIGNA   3
AVAR_2:     .ASSIGNA   5
            MCRRE
```

＜マクロ展開結果＞

```
            ADDI  R5,#5
            ADDI  R5,#6
            ADDI  R5,#7
```

.ASSIGNA

マクロ命令

算術変数を定義します。

書式

算術変数名	.ASSIGNA	算術式

解説

算術変数とは、算術式の値を割り当てたものです。算術変数はマクロボディ内、および、マクロ命令のオペランドの算術式でのみ参照できます。算術変数は .ASSIGNA 命令により再定義できます。本 .ASSIGNA 命令で定義した算術変数の、.ASSIGNC 命令による再定義はできません。

算術式には、10 進数符号付きの整数値を設定します。算術式に誤りがあればエラーとなり、算術変数には0が設定されます。算術式の記述は6.3.2.1「算術式」に従ってください。

記述例

```
            .MACRO     MCRAA
                 ADDI  R5,#\&AVAR_1
                 ADDI  R6,#\&AVAR_2
            .ENDM


AVAR_1:     .ASSIGNA   10
AVAR_2:     .ASSIGNA   \&AVAR_1 + 5
            MCRAA
```

＜マクロ展開結果＞

```
     ADDI  R5,#10
     ADDI  R6,#15
```

.ASSIGNC

マクロ命令

文字変数を定義します。

書式

文字変数名	.ASSIGNC	文字式

解説

文字式で指定した文字式を文字変数の値として定義します。文字変数はマクロ命令でのみ使用できます。文字変数は .ASSIGNC 命令で再定義できます。本 .ASSIGNC 命令で定義した文字変数の、.ASSIGNA 命令による再定義はできません。

as32R は、指定した文字変数に、0～255 文字の文字列を値として設定します。

文字式に誤りがあればエラーとなり、文字変数の値として空の文字列が設定されます。文字式の記述は6.3.2.2「文字式」に従ってください。

記述例

```
            .MACRO     MCRAC
                 \&CVAR_1   R5,R6
            .AIF  "\&CVAR_1" NE "\&CVAR_2"
                 \&CVAR_2   R5,R7
            .AENDI
            .ENDM


CVAR_1:     .ASSIGNC   "ADD"
CVAR_2:     .ASSIGNC   "MV"
            MCRAC
```

＜マクロ展開結果＞

```
      ADD   R5,R6
      MV    R5,R7
```

.AWHILE .AENDW

マクロ命令

条件により展開を繰り返します。

書式

	.AWHILE	論理式

	.AENDW	

解説

論理式の評価により展開を繰り返します。本構文は以下のように記述し、マクロ定義を行います。

```
    .AWHILE    論理式
         [文]
    .AENDW          ;.AENDW 命令により、.AWHILE 構文の終了を示します（省略不可）
```

as32R は、論理式の評価を行って、真であれば文を展開し、再び論理式を評価します。つまり、論理式が偽となるまで文の展開を繰り返し、偽になった時点で本構文を抜けます。もし、最初の評価で偽であれば、何も展開せずに本構文を抜けます。文は省略可能であり、省略した場合は何も展開しません。

論理式に誤りがあればエラーとなります。論理式の記述は6.3.2.3「論理式」に従ってください。

記述例

```
            .MACRO     MCRWHARG
            .AWHILE    \&AVAR GE \ARG
                 ADDI  R\ARG,#\&AVAR
AVAR:       .ASSIGNA   \&AVAR / 2
            .AENDW
            .ENDM


AVAR:       .ASSIGNA   20
            MCRWH5
```

＜マクロ展開結果＞

```
     ADDI  R5,#20
     ADDI  R5,#10
     ADDI  R5,#5
```

.EXITM

マクロ命令

展開を終了します。

書式

	.EXITM	

解説

本命令によってマクロ展開を終了できます。本命令が .AWIHLE 構文、.AREPEAT 構文中にある場合は、それらの構文から抜け出します。構文がネスト構造になっている場合は、本命令を囲む最小のループから抜け出すことになります。

本命令は .AIF 構文内に記述できます。ただし、.AIF 構文が、マクロボディ内、.AWHILE 構文内、または、.AREPEAT 構文内に記述されている場合のみで、それ以外では記述できません（関連：6.5「マクロ処理のネスト構造」）。

記述例

```
            .MACRO     MCREX
            .AWHILE    1
                 LDI   R\&AVAR,#1
                 .AIF  \&AVAR EQ 5
            .EXITM
            .AENDI
AVAR:       .ASSIGNA   \&AVAR-1
            .AENDW
            .ENDM


AVAR:       .ASSIGNA   8
            MCREX
```

＜マクロ展開結果＞

```
     LDI   R8,#1
     LDI   R7,#1
     LDI   R6,#1
     LDI   R5,#1
```

.INCLUDE

マクロ命令

ファイルを読み込みます。

書式

	.INCLUDE	"ファイル名"

解説

ファイル名で指定されたファイルを読み込みます。ファイル名として、相対パス指定、絶対パス指定ともに使用できます。

本命令はネストにすることができます。すなわち、.INCLUDE 命令で読み込んだファイルの中で、.INCLUDE 命令によるファイルの読み込みが可能となります。ネストは 8 レベルまで可能です。

相対パスによるファイル指定の場合、以下の順序でファイルを検索します。

(1) 本命令の書かれているファイルと同一ディレクトリ
(2) -Iオプションで指定されたディレクトリ
(3) 環境変数 M32RINC で指定されたディレクトリ。M32RINC が定義されていない場合は、/usr/local/M32R/include。

ファイル名で指定されたファイルが存在しなければエラーとなります。

記述例

```
.INCLUDE   "DATAB.H"  ;  DATAB.Hの読み込み
```

.INSTR

文字列操作関数

文字列の位置を算出します。

書式

```
.INSTR(文字式a,文字式b[,算術式])
```

文字式a　：　検索する文字列の検索対象となる文字列
文字式b　：　検索する文字列
算術式　：　検索開始位置

解説

.INSTR関数は、文字式aで表す文字列の中から文字式bで表す文字列を探し、その位置を算出します。位置は、文字列の先頭を0として表します。

文字式aの中に文字式bが存在しなかった場合、および、検索開始位置の指定を誤った場合、関数の値は-1となります。

算術式には文字列検索の開始位置を指定します。算術式を省略した場合、検索開始位置は0となります。算術式の値は、0以上の整数でなければなりません。

.INSTR関数は算術式、論理式以外では用いることができません。算術式、文字式、論理式については、6.3.2「式（マクロ命令用）」を参照してください。

記述例

```
.MACRO     MCR   ARG_1,ARG_2
.AIF       .INSTR("\ARG_1","\ARG_2",1) EQ 3
     ADD   R5,R6
.AELSE
     MV    R5,R6
.AENDI
.ENDM


     MCR   FUNCTION,CT
```

＜マクロ展開結果＞

```
ADD   R5,R6
```

.LEN

文字列操作関数

文字列の文字数を数えます。

書式

```
.LEN(文字式)
```

解説

.LEN 関数は、文字式で表される文字列の中の文字数を数えます。文字列の文字は 0 ～ 255 の範囲になければなりません。

.LEN 関数は算術式、論理式以外では用いることができません。算術式、文字式、論理式については、6.3.2「式（マクロ命令用）」を参照してください。

記述例

```
        .MACRO    MCR
            LDI   R5,#\&AVAR
        .ENDM


CVAR:  .ASSIGNC  "FUNCTION"
AVAR:  .ASSIGNA  .LEN("\&CVAR")
       MCR
```

＜マクロ展開結果＞

```
       LDI  R5,#8
```

.SUBSTR

文字列操作関数

文字列を取り出します。

書式

```
.SUBSTR(文字式,算術式a,算術式b)
```

文字式　　：　取り出される文字列
算術式a　　：　取り出しの開始位置
算術式b　　：　取り出す文字数

解説

.SUBSTR関数は、文字式で表す文字列の中から、算術式bの数だけ文字列を取り出します。取り出し開始位置の指定は算術式aで表します。位置は、文字列の先頭を0として表します。

算術式a、bの値は0以上の整数でなければなりません。算術式bに0を指定した場合、取り出される文字列は空の文字列となります。算術式bで指定した数の文字列を取り出せなかった場合、取り出される文字列は空の文字列となります。

.SUBSTR関数は文字式、論理式以外では用いることができません。算術式、文字式、論理式については、6.3.2「式（マクロ命令用）」に記載しています。

記述例

```
.MACRO     MCR   ARG_1,ARG_2
.AIF       .SUBSTR("\ARG_1",0,3) NE "\ARG_2"
           MV    R5,R6
.AELSE
           \ARG_2 R5,R6
.AENDI
.ENDM


           MCR   ADDX,ADD
```

＜マクロ展開結果＞

```
ADD   R5,R6
```

.MACRO .ENDM

マクロ定義

1行以上にわたる命令、擬似命令等を1つのマクロボディとして記憶します。

書式

	.MACRO	マクロ名D[仮引数指定[,仮引数指定]...]
	.ENDM	

仮引数指定　：　仮引数名[=初期値]

解説

マクロ定義は、.MACRO命令、マクロボディ、および.ENDM命令から構成されます。

.MACRO命令ではマクロ定義の開始を宣言します。オペランドでマクロ名、仮引数名の定義、および、仮引数の初期値が設定できます。.

.MACRO命令で定義したマクロ名は、以降のソースプログラム中でマクロコールとして処理されます。マクロコールで引数を省略すれば、.MACRO命令で定義した初期値が仮引数となります。マクロコールで引数を省略し、.MACRO命令においても初期値を定義していなかった場合、空の文字列が仮引数となります。

マクロ名、仮引数名、初期値の記述は、名前の規則、および、以下の規則に従います。

- マクロ名と仮引数指定に間には、1つ以上のスペースまたはタブが必要です。
- マクロ定義内に他のマクロ定義を含めることはできません。
- .MACRO命令で定義した仮引数名は、そのマクロ定義内に限り有効です。1つの.MACRO命令内で、仮引数名に同じ名前は使用できません。
- 仮引数初期値の設定は、仮引数名の後に等号(=)をつけて行います。初期値には、任意の文字列を指定できます。ただし、スペース( )、コンマ(,)、等号(=)、不等号(より小(<))を含めたものを初期値として定義する場合、初期値は < と > で囲むかまたはダブルクォーテーション(")で囲まなければなりません。この場合 < と > は初期値には含まれませんが、ダブルクォーテーション(")は初期値として含まれます。

.ENDM命令ではマクロ定義の終了を宣言します。マクロ定義の終わりには、必ず.ENDM命令がなければなりません。

（続く）

記述例　**例 1**

```
.MACRO     MCR   ARG_1,ARG_2      ;  MACRO文
     MV    \ARG_1,\ARG_2          ; マクロボディ
     ADD   \ARG_1,R7              ;
.ENDM                             ;  ENDM文



     MCR   R5,R6                  ;マクロコール
```

＜マクロ展開結果＞

```
MV    R5,R6
ADD   R5,R7
```

例 2

```
.MACRO     MCR   ARG_1=STR_SEC,ARG_2  ;  MACRO文
     .SECTION   \ARG_1                          ; マクロボディ
     .SDATA     "123\ARG_2"                     ;
.ENDM                                           ;  ENDM文



 MCR                                            ;マクロコール
```

＜マクロ展開結果＞

```
.SECTION   STR_SEC
.SDATA     "123"
```

付録D
アセンブルリストファイル

本付録では、入力ソースファイルと生成されるアセンブルリストファイル（以下、リストファイル）の実例を示し、リストファイルの構成や表示内容について説明します。

アセンブラas32Rの起動時に-lオプションとリストファイル名を指定すると、リストファイルが生成されます。リストファイルには、アセンブルソースリストが出力されます。アセンブルソースリストの内容は、ソースプログラム、命令コード、ロケーション、エラーメッセージ等です。

入力ソースファイルのサンプルとして、リストD.1にアセンブリ言語ソースファイルj.ms（マクロ命令.INCLUDEを含む)、リストD.2にヘッダファイルj.h（j.msが.INCLUDE命令で読み込むファイル）を示します。これらがアセンブラドライバas32Rによりマクロプロセッサa032R、アセンブルプロセッサa132Rによって処理された場合、リストファイルj.lis(リストD.3)が生成されます。

リストD.1 入力ソースファイルサンプルj.ms

```
        .SECTION P,CODE,ALIGN=4
        .EXPORT  $main
        .macro jj arg1, arg2
                ldi \arg1, \arg2
                add \arg1, r0
        .endm
$main:
        .aif 1
        ST      r2,@-R15
        .aelse
        ST      r1,@-R15
        .aendi
        MV      r2,R15
        .include "j.h"
        bl      $main0
$main0: ADDI    R5,#-4
        ST      R14,@-R15
        jj      r3, #5
        LDI     R1,#20
        ST      R1,@(-4,r2)
        bl      _main0
        LD      R1,@(-4,r2)
        ST      R1,@-R15
        .GLOBAL _bb
        .SECTION S,DATA,LOCATE=0x200000
        .datab.w5,8
data0:
        .data.h 0xF000
        .datab.w 1, 0xF000
        .SECTION        T,DATA,ALIGN=4
rel_data:
        .datab.w 20, 0x66668888
        .sdata "This Line is SDATA"<0>
        .sdata  "F000"
        .END
```

リスト D.2 入力ソースファイルサンプル j.h（ヘッダファイル）

```
_main1:
         .aif 1
         ST      r3,@-R15
         .aelse
         ST      r1,@-R15
         .aendi
         MV      r3,R15
         ADDI    R5,#-4
         ST      R14,@-R15
         LDI     R1,#20
         ST      R1,@(-4,r3)
         LD24    R1,#-4
         ST      R1,@-R15
         .GLOBAL _refs
```

リスト D.3 リストファイルサンプル j.lis

```
* ASSEMBLER * SOURCE LIST *

 LST#  SRC# LOCATION OBJ_CODE              SOURCE_STATEMENT

[j.ms] •¦1
    1     1                               .SECTION        P,CODE,ALIGN=4
    2     2                               .EXPORT $main
    3     3                               .macro jj arg1, arg2
    4     4                                       ldi \arg1, \arg2
    5     5                                       add \arg1, r0
    6     6                               .endm
    7     7                          $main:
    8     8                                       .aif 1
    9     9 00000000 227F                         ST      r2,@-R15
   10    10                                       .aelse
   11    11                        X              ST      r1,@-R15
   12    12                                       .aendi
   13    13 00000002 128F                         MV      r2,R15
   14    14                                       .include "j.h"
[j.h]
   15     1                        1   $main1:
   16     2                        1              .aif 1
   17     3 00000004 237F          1              ST      r3,@-R15
   18     4                        1              .aelse
   19     5                        1X             ST      r1,@-R15
   20     6                        1              .aendi
   21     7 00000006 138F          1              MV      r3,R15
   22     8 00000008 45FC          1              ADDI    R5,#-4
   23     9 0000000A 2E7F          1              ST      R14,@-R15
   24    10 0000000C 6114          1              LDI     R1,#20
                    F000 •¦2
   25    11 00000010 A143FFFC      1              ST      R1,@(-4,r3)
   26    12 00000014 E1FFFFFC      1              LD24    R1,#-4
# a132R: "j.h", line 12: warning: ignore sign bit at 24-bit immediate data •¦4
   27    13 00000018 217F          1              ST      R1,@-R15
   28    14                        1              .GLOBAL _refs
[j.ms]
   29    15 0000001A 7E01                         bl      $main0
   30    16 0000001C 45FC             $main0:     ADDI    R5,#-4
   31    17 0000001E 2E7F                         ST      R14,@-R15
   32    18                                       jj      r3, #5
   33    18 00000020 6305           &             ldi r3, #5
   34    18 00000022 03A0           &             add r3, r0
   35    19 00000024 6114                         LDI     R1,#20
                    F000
   36    20 00000028 A142FFFC                     ST      R1,@(-4,r2)
   37    21 0000002C 7EFC                         bl      $main0
                    F000
```

```
38    22 00000030 A1C2FFFC          LD      R1,@(-4,r2)
39    23 00000034 217F              ST      R1,@-R15
                  F000
40    24                            .GLOBAL _bb
41    25                            .SECTION        S,DATA,LOCATE=0x200000
42    26 00200000 [5]  ※3           .datab.w 5,8
                  00000008
43    27                      data0:
44    28 00200014 F000              .data.h 0xF000
45    29 00200016 0000F000          .datab.w 1, 0xF000
46    30                            .SECTION        T,DATA,ALIGN=4
47    31                      rel_data:
48    32 00000000 [20]              .datab.w 20, 0x66668888
                  66668888
49    33 00000050 54686973 ●5      .sdata "This Line is SDATA"<0>
                  204C696E
                  65206973
                  20534441
                  544100
50    34 00000063 46303030          .sdata  "F000"
51    35                            .END
```

以下より、リストファイルに出力されるアセンブルソースリストの構成について説明します（リストD.3中の※1～※5部分についても説明しています）。

アセンブルソースリストの各行は表D.1に示す各欄から構成されています。行によっては情報が出力されない欄もあります。欄タイトル（LST#など）はリストの始めに出力されます。

表D.1 アセンブルソースリストの各行の構成

欄	内容
LST#	リスト行番号（10進数）。番号を持たない行も存在するため、リスト総行数とは一致しません。
SRC#	ソース行番号（10進数）。入力ソースファイル内での行番号を表示します。
LOCATION	コードを配置するアドレス（8桁の16進数）。8桁に満たない場合、上位桁には0が表示されます。相対セクションの場合は配置のオフセットを示します。
OBJ_CODE	おもに生成コードを示します。
SOURCE_STATEMENT	対応ソースファイル行の内容。

コード（OBJ_CODE欄）とソース（SOURCE_LIST欄）の間に、行の種類（表D.2参照）が表示される場合があります。

表D.2 行の種類

表示	意味
X	マクロ命令.AIF構文で、処理の対象にならなかった行。
&	マクロコール、.AREPEAT構文、.AWHILE構文により展開された行。
番号	マクロ命令.INCLUDEのネスト数（10進数）。
空白	上記以外の行。

その他、各種表示について以下に示します。リストファイルサンプル（リストD.3）中のアセンブルソースリスト内の各部分（※1～※5）を例にとって説明します。

■ ファイル名表示（※1）

```
------------------------------------------------------------------------
[j.ms]
------------------------------------------------------------------------
```

入力ソースファイル名を示します。ファイル名は[*ファイル名*] の形で表示されます。リストファイルの先頭、および、 .INCLUDEマクロ命令によりソースファイルがネストしている場合など、以降の行を含んでいるファイルが直前までの行を含んでいるファイルと変わる場合に表示します。

■ NOPコード（※2）

```
------------------------------------------------------------------------
   F000
------------------------------------------------------------------------
```

アセンブラがアライメント調整のためNOP命令を生成する場合があります。その場合、OBJ_CODE欄に NOP 命令のコードF000 が出力されます。LOCATION欄にアドレスは表示されません。また、SOURCE_STATEMENT欄に"NOP"は表示されません。

■ 繰り返し数表示（※3）

```
------------------------------------------------------------------------
   42    26 00200000 [5]     .datab.w 5,8
                     00000008
------------------------------------------------------------------------
```

.DATAB擬似命令のような繰り返してデータを生成する擬似命令の場合、OBJ_CODE欄に繰り返し数表示が行われます。繰り返し回数は10進数で表示され、[と]で囲まれます。データの値は次の行以降に継続して表示されます。LOCATION欄にアドレスは表示されません。繰り返し回数が1回の場合、繰り返し回数は表示されず、アドレスの後にデータがそのまま表示されます。LOCATION欄にはデータの先頭アドレスのみ表示されます。

■ エラー／ワーニングメッセージ（※4）

```
-----------------------------------------------------------------
   # a132R: "j.ms", line 12: warning: ignore sign bit at 24-bit immediate data
-----------------------------------------------------------------
```

エラーおよびワーニングメッセージは発生行以降の行に表示されます。エラーによりアセンブル処理ができなかった場合、すべての行のLOCATIONおよびCODE欄は出力されません。先頭にはアセンブルドライバas32Rが起動するコマンドツールの名称（「a032R:」もしくは「a132R:」）が出力され、次のことを意味します。

- a032R: ：マクロプロセッサa032Rで発生したエラー
- a132R: ：アセンブルプロセッサa132Rで発生したエラー

行頭には「#」が付加されます。

■ データの表示（※5）

```
-----------------------------------------------------------------
   49   33 00000050 54686973 .sdata "This Line is SDATA"<0>
                    204C696E
                    65206973
                    20534441
                    544100
-----------------------------------------------------------------
```

同一擬似命令によるデータコードは、1行に2ワード（16バイト）ずつ、16進数で表示されます。2ワードを超える場合、その次の行以降に継続して出力されます。その場合、LOCATION欄にアドレスは表示されません。

付録E
M32R/ECU#5拡張命令対応

M32R/ECU シリーズ 32180 および 32182 グループ(以下、M32R/ECU#5 と略します)の拡張命令(FPU 命令など)に対応しました。

・M32R/ECU#5 拡張命令を用いたアセンブリ言語を記述できます
・アセンブリ言語において浮動小数点定数を記述できます

FPU 命令などの M32R/ECU#5 で拡張された命令を含むプログラムをアセンブルできます。
この機能を有効にするには、次のオプションを使用します。

E.1 オプション指定

M32RE/#5 拡張命令をアセンブルする場合は、次のオプションを指定する必要があります。

-m32re5	M32R/ECU#5拡張命令を有効にします。

浮動小数点定数で非正規化数になるものは0.0に切り詰めます。

E.2 M32R/ECU#5拡張命令

表E.1に示す命令に対応します。

詳細は、M32R/ECU#5 のソフトウェアマニュアルを参照してください。

表E.1 M32R/ECU#5拡張命令一覧

分類	ニーモニック	オペランド	機能概要
ストア命令	STH	Rsrc1,@Rsrc2+	レジスタからメモリにハーフワード値をストア(ポストインクリメント付き)
ビット操作命令	BSET	#bitpos,@(disp16,Rsrc)	指定ビットに1をセット
	BCLR	#bitpos,@(disp16,Rsrc)	指定ビットに0をセット
	BTST	#bitpos,Rsrc	レジスタの指定ビットを取り出しCフラグへ
	SETPSW	#imm8	PSW のSM,IE,C の任意のビットに1をセット
	CLRPSW	#imm8	PSW のSM,IE,C の任意のビットに0をセット
浮動小数点命令	FADD	Rdest,Rsrc1,Rsrc2	浮動小数点加算 (Rdest=Rsrc1+Rsrc2)
(FPU命令)	FSUB	Rdest,Rsrc1,Rsrc2	浮動小数点減算 (Rdest=Rsrc1-Rsrc2)
	FMUL	Rdest,Rsrc1,Rsrc2	浮動小数点乗算 (Rdest=Rsrc1*Rsrc2)
	FDIV	Rdest,Rsrc1,Rsrc2	浮動小数点除算 (Rdest=Rsrc1/Rsrc2)
	FMADD	Rdest,Rsrc1,Rsrc2	浮動小数点積和演算 (Rdest=Rdest+Rsrc1*Rsrc2)
	FMSUB	Rdest,Rsrc1,Rsrc2	浮動小数点積差演算
			(Rdest=Rdest-Rsrc1*Rsrc2)
	ITOF	Rdest,Rsrc	整数から単精度浮動小数点数への変換
	UTOF	Rdest,Rsrc	符号なし整数から単精度浮動小数点数への変換
	FTOI	Rdest,Rsrc	単精度浮動小数点数から32ビット整数への変換
	FTOS	Rdest,Rsrc	単精度浮動小数点数から16ビット整数への変換
	FCMP	Rdest,Rsrc1,Rsrc2	浮動小数点比較 (Rdest=(Rsrc1とRsrc2の比較結果))
	FCMPE	Rdest,Rsrc1,Rsrc2	浮動小数点比較 (Rdest=(Rsrc1とRsrc2の比較結果))

表記の意味

Rn	任意の汎用レジスタ(n=0～15)
@Rn+	汎用レジスタRn参照(レジスタ間接)後、Rnの内容が+4される(レジスタ更新)ことを示す。
@(disp_nn,Rn)	汎用レジスタRn参照(disp_nn付きレジスタ間接)であることを示す。
bitpos	ビット位置(0～7)
disp_nn	ビットディスプレースメント値
imm_nn	ビット符号付き整数即値

付録 F
浮動小数点対応機能

F.1 浮動小数点定数

F.1.1 記述形式

２通りの形式があります。

[a] 通常表記

```
[(+|-)]<(f'|F')>< 浮動小数値 >[精度指定][(+|-)指数]
```

・符号

先頭の +,- は符号です。省略した場合は正を指定したことになります。

・精度指定

精度指定を省略すると、単精度となります。

ただし、サイズ指定のある浮動小数点記述用の擬似命令(→「F.2 拡張擬似命令」参照)に、精度指定を省略してこの形式を記述すると、擬似命令のサイズが精度として使われます。

s または S ... 単精度

d または D ... 倍精度

・指数

符号付きで 10 の何乗であるかを示します。

例)　f'1.0s+10　　(単精度) 1.0x10の10乗

-f'3.14159s+10　　(単精度) -3.14159×10の10乗

+F'3.14159D-10　　(倍精度) 3.14159 × 10 の -10 乗

[b] C 言語互換表記

```
[(+|-)]< 浮動小数値 >[(e|E)(+|-)指数][f|F]
```

・符号

先頭の +,- は符号です。省略した場合は正を指定したことになります。

・ 精度指定

精度指定を省略すると、倍精度となります。

ただし、単精度のサイズ指定のある浮動小数点記述用の擬似命令

(→「F.2 拡張擬似命令」参照)に記述した場合は単精度になります。

f または F ... 単精度

(指定無し) ... 倍精度(ただし、単精度指定(.S)のある擬似命令に記述した場合を除く)

例)　1.0e+10f　(単精度)　1.0x10 の 10 乗
-3.14159E+10　(倍精度) -3.14159 × 10 の 10 乗
+3.14159-10　(倍精度)　3.14159 × 10 の -10 乗

・指数
e または E の後に、符号付きで 10 の何乗であるかを示します。

F.1.2　利用可能箇所

次の場所で浮動小数点定数を記述すると、IEEE-754 に準拠した単精度(4 バイト)あるいは倍精度(8 バイト)のフォーマットに置換されたものに置き換わります。

・擬似命令
倍精度、単精度とも、.FDATA, .FDATAB 擬似命令のパラメータに記述することができます。

・一般命令
単精度に限り、補正オプション(HIGH,LOW,SHIGH)の中に記述することができます。

F.1.3　互換性

浮動小数点定数の単精度と倍精度は、C コンパイラで用いている float 型、double 型の内部表現とそれぞれ互換性があります。

F.1.4　非正規化数の扱い

アセンブラに -m32re5 オプションを指定すると、非正規化数は 0.0 に切り詰めます。

F.2 拡張擬似命令

浮動小数点に対応した３つの擬似命令が利用できます。

F.2.1 形式

.FDATA[.size]　定数a[,定数a]	浮動小数点定数を配置します。
.FDATAB[.size]　式b, 定数a	連続した浮動小数点定数を配置します。
.FRES[.size]　式b	浮動小数点定数領域を確保します。

※ 記号の意味

..size　：サイズ指定
.s または .S ：単精度
.d または .D ：倍精度
定数a　：浮動小数点定数
式b　：個数

F.2.2 各擬似命令の機能

・.FDATA 擬似命令

sizeで指定した精度のデータ領域を確保し、その領域内に定数a(浮動小数点数)の内部表現を格納します。

例) .FDATA　F'1.0S+2, F'2.0S+2 ; 1.0e+2f と 2.0e+2f を配置します。

・.FDATAB 擬似命令

sizeで指定した精度のデータ領域を式bの個数分確保し、その領域内に定数aの値(浮動小数点数)の内部表現を式bの個数だけ連続して格納します。

例) .FDATAB 8, F'1.0S+2　; 1.0e+2f を 8つ続けて配置します。

・.FRES 擬似命令

sizeで指定した精度のデータ領域を式bの個数分確保します。

例) .FRES.S 4　; 単精度の浮動小数点数値の領域を4つ分確保します。

F.2.3 共通事項

・サイズ指定は、.sまたは.S(単精度)、.dまたは.D(倍精度)、で行います。
・定数a (浮動小数点定数)の条件:
　サイズ指定の範囲内でなければなりません。
　定数は1つだけ記述できます (定数式 (F'1.0 + F'2.0 等)は記述できません)。
・式b (個数)の条件
　定数式かつ正の値で、0以上の整数てなければなりません。
　シンボルを含む場合、本擬似命令以前で定義されていなければなりません。

F.3 一般命令行における浮動小数点の利用

補正オプション(HIGH,LOW,SHIGH)が有効な命令に対しては、単精度の浮動小数点定数を記述することができます。(補正オプションには倍精度は適用できません。)

[通常表記を用いる場合]

```
SETH  R0,#HIGH(f'1.0s+2)
OR3   R0,R0,#LOW(f'1.0s+2)
```

[C言語互換表記を用いる場合]

```
SETH  R0,#HIGH(1.0e+2f)
OR3   R0,R0,#LOW(1.0e+2f)
```

1.0×10の2乗の内部表現は 42C80000(16進数)のため、上記は次の記述と等価です。

```
SETH  R0,#HIGH(0x42C80000)
OR3   R0,R0,#LOW(0x42C80000)
```

付録G
制限事項

本付録では、現在確認されている CC32R の制限事項を示します。

■ デバッグ情報のないファイルを得るには

Cコンパイラ cc32R、アセンブラ as32R、リンカ lnk32R において、常にデバッグ情報が有効になるように変更になっているのに伴い、cc32R，as32R，lnk32Rは、常に従来の-gオプションが付いた状態で処理を行います。

なお、これらのデバッグ情報の出力は、オプション等では抑止できません。

従来と同様のデバッグ情報が無い出力を得るためには、デバッグ情報除去ツール strip32Rをご利用ください。strip32Rは、リンカが出力するロードモジュールファイルだけでなく、コンパイラやアセンブラが出力するオブジェクトモジュールファイルも処理することができます。すなわち、cc32R，as32R，lnk32Rの実行のすぐ後に、それぞれの出力ファイルを直接strip32Rで処理するようにすれば、従来のV.4.10以前のCC32Rと等価な動作にすることができます。

[strip32R の使用例　（% はプロンプトを表します）]

通常使用

cc32R，as32R，lnk32Rの出力ファイルに対して、その都度適用できます。
リンク前のオブジェクトモジュールファイルでも、リンク後のロードモジュールファイルでも同様に処理できます。

```
% cc32R -c -o sample1.mo sample1.c
% strip32R sample1.mo
% as32R sample2.ms
% strip32R sample2.mo
% lnk32R -o sample.abs sample1.mo sample2.mo
% strip32R sample.abs
```

複数指定

コンパイルとアセンブルを行った後に、まとめて処理できます。

```
% cc32R -c sample1.c sample2.c sample3.c
% cc32R -c sample4.c
% as32R -c sample5.ms
% strip32R sample1.mo sample2.mo sample3.mo sample4.mo sample5.mo
```

※ ワイルドカードも使用可能です。

```
% strip32R *.mo
```

■ ベースレジスタ機能と標準ライブラリを併用する場合の注意事項

[ベースレジスタ機能使用上の注意事項の補足]

次のいずれかのオブジェクトファイルの組み合わせでリンクを行うことは推奨しておりません。(ユーザーズマニュアル<<C コンパイラ編>>「A.1.6 ベースレジスタ機能の制限事項」参照)

(1) ベースレジスタ機能を有効にして作成したオブジェクトファイルと、無効にして作成したオブジェクトファイル

(2) それぞれ、異なるアクセス制御ファイルを用いて作成した複数のオブジェクト

[ベースレジスタ機能と標準ライブラリ併用時の注意事項]

製品添付の標準ライブラリは、ベースレジスタ機能を無効にして作成しています。このため、ベースレジスタ機能を用いたユーザプログラムと標準ライブラリは、上記(1)の組み合わせになります。

このような場合、標準ライブラリ関数処理中は、ベースレジスタがベースアドレスを保持していません。関数処理後はベースレジスタはベースアドレスに戻りますが、次のような場合はベースレジスタの値が不正になります。

(1) 割込み処理ルーチン
標準ライブラリ関数実行中に割り込みがかかることがあるため、割り込み処理ルーチン中ではベースレジスタの値は不定と考える必要があります。

(2) 特定の標準ライブラリ関数(qsort,bsearch等)から呼び出されるユーザ関数

[対策]

ベースレジスタ機能と標準ライブラリを併用する場合は、次の(1),(2)のうちいずれかの方法を用いて対策を行ってください。

(1) ユーザプログラムと同じアクセス制御ファイルを用い、ベースレジスタ機能を有効にした標準ライブラリを再構築し、現在の標準ライブラリと差し替えてください。

(2) 割込み処理ルーチンおよび、特定の標準ライブラリ関数(qsort,bsearch等)から呼び出されるユーザ関数を、ベースレジスタ機能を無効にして再コンパイルしてください。

■ M32R/ECU シリーズでの整数剰除算の問題に対する対応方法について

M32R/ECU シリーズにおいて、整数剰除算命令（DIV,DIVU,REM,REMU の各命令。以下 DIV 系命令と略します）でゼロ除算を行った場合、その直後に実行した命令の実行結果が不正になる問題があります。詳しくは テクニカルニュース No.M32R-50-0301「M32R/ECU シリーズ 0 除算実行時の注意事項」を参照ください。
以下に、CC32R での対応方法とゼロ除算問題抑止オプション -zdiv について説明します。

[対応方法]

C 言語プログラム、アセンブリ言語プログラムの場合

(1) 論理的にゼロ除算が行われないないように、プログラミングしてくださ
CC32Rは、C言語の整数除算(/ および /=) と 整数剰余(% および %=)に対して DIV 系命令を生成しますので、これらの演算子の除数が 0 にならないようにしてください。
アセンブリ言語では、DIV 系命令の第 2 レジスタが 0 にならないようにしてください。

(2) (1)が確実にできない場合は、-zdivオプションを指定して再コンパイルあるいはアセンブルしてください。

標準ライブラリを用いている場合

標準ライブラリは本問題に対応済みですので、DIV 系命令で 0 除算を行っても問題は発生しません。
なお、CC32R V.4.10 Release 1で用意しておりましたゼロ除算対策版のライブラリ(m32RcRZ.lib, m32RcRZM.lib, m32RcRZL.lib)は、本バージョンでは通常版(m32RcR.lib, m32RcRM.lib, m32RcRL.lib)に統合しております。
このため、CC32R V.4.10 Release 1 を使用されていた方で、リンクする標準ライブラリにゼロ除算統合版を指定なさっていた場合は、それらは通常版に戻してご使用ください。

標準ライブラリ以外のライブラリを用いている場合

お客様で作成されたライブラリまたは、添付の標準ライブラリソースを再構築して作成したライブラリをご使用の場合は、-zdivオプションを付けてライブラリを再構築してください。

[-zdiv オプションの解説]

cc32R でコンパイルする場合

-zdiv オプション付きでコンパイルすると、コード生成で DIV 系命令を出力するときは、その直後に NOP 命令を挿入します。
また、asm 関数内に DIV 系命令がある場合もその直後に NOP 命令を挿入します。
なお、-S および -CS オプションと本オプションを同時に指定した場合、asm 関数内のコメントは除去し、記述も全て大文字に変換したうえでアセンブリソースを出力しますのでご注意ください。
cc32R にアセンブリソース(拡張子 .ms)を入力する場合は、as32R でアセンブルする場合と同様です。

as32R でアセンブルする場合

DIV系命令を持つアセンブリコードを -zdiv オプション付きでアセンブルすると、その DIV 系命令の直後に NOP 命令を挿入します。
ただし、すでに DIV 系命令の直後に NOP 命令が存在する場合を除きます。
これは、この DIV 系命令と NOP 命令のあいだに次のものが存在していない場合です。すなわち、これらのものが DIV 系命令と NOP 命令の間に存在しているときは、NOP 命令を DIV 系命令の直後に追加します。

(1) ラベル
(2) NOP以外の一般命令
(3) 領域に影響を及ぼす擬似命令(下記)

```
.ALIGN    .DATA    .DATAB    .END      .FDATA
.FDATAB   .FRES    .RES      .SDATA    .SDATAB
.SECTION
```

■ 可変引数関数をポインタ間接で呼び出す場合の問題

プロトタイプ宣言がない関数へのポインタ変数を用いて、間接的に可変引数の関数を呼び出すような場合、生成されるコードが正しい動作をしません。

【記述例】

```
#include <stdio.h>
   int (*funcptr)() = printf;
   int main(void) {
      (*funcptr)("calling printf with %d\n", 1);
   }
```

【回避方法】

関数へのポインタ変数にプロトタイプ宣言をつけてください。

```
   #include <stdio.h>
   int (*funcptr)(const char *,...) = printf;
   int main(void) {
      (*funcptr)("calling printf with %d\n", 1);

   }
```

■ コードセクション中のデータ定義について

アセンブラは、コードセクション中にデータ（もしくは空き領域）がある場合に注意を促すため、ワーニング(warning: caution! there are some data in code section)を出力します。
データは、データセクションに記述して頂くことを推奨いたします。
なおこのワーニングは、オプション(-warn_suppress_code_data)で抑止することができます。

■ マクロボディ内におけるプリプロセッサ変数の使用

マクロボディ内において、マクロコール直後の行の第1カラムからプリプロセッサ変数を記述した場合、マクロコールにおいてプリプロセッサ変数が正しく展開されない場合があります。

【記述例】

```
.macro INST_MACRO
    MOV         #0,R0
    .endm
    .macro LABEL_MACRO label
    INST_MACRO
\label:                 •© ƒvƒŠƒvƒ•ƒ‰ƒbƒT•Ï•",ðˆæ1 ƒJƒ‰ƒ•,©,ç‹L•ç•B
    .endm
    .section P,code,align=2
    LABEL_MACRO L1      •© •³,µ–WŠJ,³,ê,È,ç•B
     LABEL_MACRO L2      •©•³,µ–WŠJ,³,ê,È,ç•B
    .end
```

【回避方法】

マクロボディ内におけるプリプロセッサ変数は、第2カラム以降から記述してください。

■ 500行以上の関数を持つプログラムをコンパイルする場合

500行以上の大きな関数を持つCソースプログラムをコンパイルする場合、”Out of memory”のエラーが発生する場合があります。
このような場合は、この関数を分割して小さくしてください。

■ 関数引数の設定規則変更に伴う注意事項

V.3.00 Release 1 以降、関数の引数はレジスタ渡しになっています(従来の-RBPPオプション相当)。従って、V.2.10 Release 1 以前のCC32R用に作成された次のようなプログラムと組み合わせて使用することはできません。それぞれ次のように対応してください。

(1)　引数をスタック渡しするC言語プログラム

V.2.10 Release 1以前のコンパイラを用いて、-RBPPオプション指定なしでコンパイルされたオブジェクト及びライブラリが対象となります。

【対応】

V.3.00 Release 1で再コンパイルしてください。

(2)　引数をスタック渡しにするアセンブリプログラム

C言語の関数を呼び出すプログラム、および、C言語の関数から呼び出されるアセンブリ言語プログラムのうち、引数をスタック渡しするアセンブリ言語プログラムが対象になります（スタートアッププログラム、低水準ライブラリ関数を含む)。

【対応】

* V.3.00 Release 1の関数引数の設定規則(ユーザーズマニュアル《Cコンパイラ編》の「C呼び出し規則」の章を参照ください) に従って、アセンブリ言語プログラムを変更してください。

* レジスタ渡しで呼び出しを行う関数名は、オブジェクトモジュール内では、Cプログラムにおける関数名の先頭に、アンダースコア (_) ではなく ドルマーク ($) が付加されます。アセンブリ言語中の関数名はこれに従った名前に変更してください。

上記 (1) や (2)に該当するプログラムを、【対応】を行わずにCC32R V.3.00 Release 1用に作成されたプログラムとリンクすると、"external symbol not defined"のエラーとなります。

第２部

リンカ　lnk32R

第１章
リンカlnk32Rの概要

1.1 概要

lnk32R はM3T-CC32R クロスツールキットに含まれているリンカで、以下のような機能があります。

■ ロードモジュールファイル（アブソリュートロードモジュールファイル、リロケータブルロードモジュールファイル）を生成する。

オブジェクトモジュールファイル（以下 オブジェクトモジュール）、リロケータブルロードモジュールファイル（以下 リロケータブルロードモジュール）、およびライブラリファイル（以下 ライブラリ）をリンク（結合）して、アブソリュートロードモジュールファイル（M32R システム上で実行可能なロードモジュールファイル）を生成します。

■ リンクマップを生成する。

起動オプション -M 指定によって、リンクマップをファイルに出力します。リンクマップとは、各セクションの配置情報を示す「マップリスト」と外部定義シンボル情報を示す「外部定義シンボルリスト」から構成されるリストです。-M オプションが指定されると、リンカはマップジェネレータを起動してリンクマップを生成します。マップジェネレータの詳細は、第３部「マップジェネレータ map32R」で説明しています。

1.2 機能

■ ２種類のロードモジュールが生成可能

ロードモジュールの形式は、アブソリュート形式またはリロケータブル形式のいずれかを選択できます。

・ アブソリュートロードモジュール
全セクションの先頭アドレス（絶対アドレス）が決定しており、未定義シンボルがない、実行可能なロードモジュール。

・ リロケータブルロードモジュール
セクション先頭の絶対アドレスが決定していないロードモジュールで、起動オプション -r を指定したときに作成されます。リロケータブルロードモジュールは、リンカに再入力することによってアブソリュート

ロードモジュールに変換できます。

■ ROM 化機能をサポート

プログラムを ROM 化するために、以下の機能をサポートしています。

- セクションの初期値データ出力抑止機能
- セクションの初期値データ抽出機能
- 予約ラベルの自動生成機能

これらは、ターゲットシステムへの組み込み用アプリケーションを作成する場合に有効です。

■ コマンドファイルによる起動が可能

起動コマンド lnk32R のパラメータ（オプション指定、入力ファイル指定）を、コマンドファイルから与えることができます(2.1.3.2参照)。

■ オーバーレイ機能をサポート

異なるセクションを同じアドレスに配置させるため、オーバーレイ機能をサポートしています。この機能により、全てのセクションの重なりをチェックしないようになりますので、使用する際には、注意してください。

1.3 前バージョンとの互換性

1.3.1 CC32R V.2.10 Release 1 以前のオブジェクトをリンクする場合

■ CC32R V.2.10 Release 1 以前のオブジェクトをリンクする場合

V.2.10 Release 1 以前のCC32R で生成したオブジェクトファイル(ライブラリファイル含む)を V.3.00 Release 1 以降(本バージョン含む)のリンカに入力すると、以下のようなワーニングを表示します。

lnk32R:"ファイル名":warning:old interface module:"revision:01"

この場合は、該当するオブジェクトファイルを最新の CC32R で生成し直してください。

■ CC32R V.1.00 Release 3 以前のオブジェクトをリンクする場合

V.1.00 Release 3 以前のCC32R で生成したオブジェクトファイル(ライブラリファイル含む)を、V.1.00 Release 4 以降(本バージョン含む)のリンカに入力すると、"relocation out of range"のエラーが発生する場合があります。この場合は、該当するオブジェクトファイルを最新の CC32R で生成し直してください。

1.3.2 lnk32R のエラー処理の仕様（CC32R V.4.20 Release 1 以降）

■ -SECオプションのエラー処理の緩和

-SEC オプションに指定したセクションが、入力ファイルやライブラリに存在しない場合、ワーニングとして処理するようにしました。

従来と同様にエラーとして処理したい場合は、次のオプションを指定してください。

◆オプション

-Werrsec

-SEC オプションのセクションに、入力されたオブジェクトモジュールファイルに含まれないセクションを指定した場合、エラーとして処理します。
本オプションがないデフォルトの処理では、このような場合はワーニングとし、処理を続行することができます。

■ リロケーションサイズオーバーフロー時の詳細表示

リンカにおいて､"relocation size overflow (xx-bit)" の際に表示するメッセージは､セクション名と､モジュール内オフセット および 参照シンボルしか表示しませんが、次のオプションを指定することで、詳細な情報を表示するようになりました。

◆オプション

-Wreloc

relocation size overflow エラーが発生した場合に､そのメッセージの直後に位置情報(Position)と､設定値情報(Setting)を表示します。

◆表示形式

(1)位置情報 (Position):

[Position: sect"セクション名"(アドレス)+オフセット in module"モジュール名"]

リロケーションアドレスの、所属セクションおよびモジュール名と、実際のアドレスを示します。実際のリロケーションアドレスの絶対値は、表示されるアドレスにオフセットを加えたものです。
(※ アドレス、オフセットは 0xから始まる16進数で表示します。)

(2)設定値情報 (Setting):

[Setting: 設定値(参照情報)]]

設定値に関する情報です。設定値は16進数で表示します。通常は、参照情報に、この設定値の計算に用いたセクションやシンボルなどの情報を出力します。

sect"セクション名"(top=アドレス), module"モジュール名"

参照しているセクションとそれが所属するモジュール名の情報です。アドレスは、このセクションを全て結合した後の、全体の先頭アドレスを示します。
（※ アドレスは 0xか ら始まる 16 進数で表示します。）

"シンボル名"

参照しているシンボル名です。

in sect"セクション名" module"モジュール名"

設定値をアドレスと仮定した場合の、そのアドレスが所属するセクションとモジュール名を表示します。

◆-Wreloc 指定時の表示例

```
lnk32R:"c:\mtool\lib32R\m32RcR.lib":error:relocationsizeoverflow(24-bit):"P",0x00000011,""
[Position:sect"P"(0x2FCC)+0x11inmodule"stdio_pw"]
[Setting:0x800001E4(sect"C"(top=0x80000198),module"stdio_pw")]

lnk32R:"c:\mtool\lib32R\m32RcR.lib":error:relocationsizeoverflow(24-bit):"P",0x000001ED,"__100_ctype_tab"
[Position:sect"P"(0x388C)+0x1EDinmodule"locale"]
[Setting:0x800001EE("__100_ctype_tab",insect"C"module"_C_ctype")]
```

■ メッセージ数の上限設定

メッセージの数が 20 を越えると、21 個目以降のメッセージは表示を抑止するようにしました。オプション指定により、この制限値を変更したり、制限を無くして全てを表示させることもできます。

◆オプション

-Wlimit=message_max

メッセージ表示数の上限(message_max)を設定します。この数値に指定された個数だけメッセージを表示し、それ以降のメッセージは抑止します。数値に 0 を指定した場合は制限はなくなり、全てのメッセージを表示します。
デフォルトでは、-Wlimit=20 が指定されたのと同じ動作となります。

-Wnolimit

全てのメッセージを表示します。

1.3.3 lnk32R のエラー処理の仕様（CC32R V.4.30 Release 1 以降）

■ -M オプションのエラー処理の緩和

lnk32R のリンクマップファイル出力オプション、-M オプション（または、cc32R の -MAP オプション）を指定していた場合、リンク処理中にエラーが発生してもリンクマップを出力するようにしました。
エラー発生時のリンクマップには不完全な情報を含まれていますが、モジュールごとのセクション配置やシンボルアドレスなどがわかりますので、リンクエラーの原因の確認に利用することができます。

第２章
リンカの起動

2.1 リンカを起動するには

2.1.1 リンカの起動手順

リンカを起動するためには、環境変数を設定（2.1.2参照）した後、規則に従ってコマンド「lnk32R」を入力し、それを実行します（2.1.3参照）。

2.1.2 環境変数の設定

環境変数M32RBIN、M32RINC、M32RLIB、M32RTMPに正しいディレクトリを設定します（通常はインストール時に設定します）。設定方法は「M3T-CC32Rクロスツールキット V.XX Release Xリリースノート」を参照してください。環境変数の設定を省略すると、デフォルトのディレクトリを設定したことになります。

表2.1 環境変数のデフォルト

環境変数	デフォルトのディレクトリ
M32RBIN	/usr/local/M32R/bin
M32RINC	/usr/local/M32R/include
M32RLIB	/usr/local/M32R/lib
M32RTMP	/tmp

2.1.3　コマンド行の記述規則

リンカ起動コマンド「lnk32R」の、コマンド行の入力書式および規則を以下に示します。オプションおよび入力ファイルの指定方法には、コマンド行で直接入力する方法と、コマンドファイルを使って指定する方法があります（2.1.3.1、2.1.3.2参照）。起動オプションの詳細は2.2節、入出力ファイルについての詳細は2.1.4～2.1.6節を参照してください。

(1)コマンド行で各種起動指定を行う場合

```
lnk32R  [-o output_filename] [-r] [-g] [-V] [-w] [-e entrypoint]
        [-L dir] [-l lib] [-M map_filename ]
        [-SEC name[=addr],name[=addr]...] [-LOC addr1,addr2]
        [-overlap]
        [-Werrsec] [-Wreloc] [-Wlimit=message_max]
        [-Wnolimit] [-Wmangle]
        object_filenames <RET>
```

(2)コマンドファイルを使って各種起動指定を行う場合

```
lnk32R  command_filename <RET>
```

※[ ]で囲まれていない項目　：必ず入力しなければならない項目
※[ ]で囲まれた項目　：必要に応じて記述する項目
※　－の付いた記号　：起動オプション（詳細は1.3節参照）
※<RET>　：リターンキー入力

図2.1　lnk32Rコマンドの入力書式

2.1.3.1　コマンド行で各種起動指定を行う場合

コマンド行に指定した情報によってリンカを起動するには、以下の規則に従ってコマンドを入力および実行します。

■ コマンド行は、図2.1(1)の書式に従って記述します。各項目（コマンド名、オプション、入力ファイル名）の間は、1文字以上の空白文字で区切ります。最後にリターンキーを入力すると、リンカが実行を開始します。

■ オプションとそのパラメータとの間には空白文字が必要です。矛盾するオプションの組み合わせがあれば、後から指定したオプションが有効となります。

■ アドレスや数値の指定には16進数による指定のみ有効です。

■ *object_filenames*には、1つまたは複数の入力ファイル名を指定します。ファイル名とファイル名の間は1文字以上の空白で区切ります。ファイル数の制限はありません。

2.1.3.2 コマンドファイルを使って各種起動指定を行う場合

リンカを起動するためのオプションや入力ファイル名の指定は、コマンドファイルを使って指定することができます。コマンドファイルとは、あらかじめ指定内容を記述したテキストファイルです。この方法は、入力するファイル名が多い場合や、リンカに対する処理内容がほぼ決まっている場合に便利です。

コマンド行では、以下のようにパラメータとしてコマンドファイル名を指定します。

```
lnk32R command_filename <RET>
```

コマンドファイルの記述方法を以下に示します。

- 各パラメータ（オプション指定、入力ファイル名指定）の記述形式は、コマンド行で各種起動指定を行う場合の各パラメータの記述形式（図2.1(1)参照）に従う。
- パラメータ間を改行（リターンキー入力）で区切ることが可能。
- コマンドファイルの１行の文字数は255文字まで（改行文字除く）。

例えば、sin.mo、cos.mo、tan.mo、asin.mo、acos.mo、atan.mo、hsin.mo、hcos.mo、htan.mo、log.mo、およびlog10.moの11個のファイルを入力し、アブソリュート形式ロードモジュールfunc.absを出力したい場合は、以下のようなコマンドファイルを作成しておきます。

```
-o func.abs
sin.mo  cos.mo   tan.mo   asin.mo acos.mo  atan.mo
hsin.mo hcos.mo  htan.mo  log.mo  log10.mo
```

図2.2 コマンドファイルの内容（例）

2.1.4 入力ファイルの条件

リンカで処理できる入力ファイルの条件を以下に示します。これらの条件を満たさないファイルは入力しないでください。

表2.2 入力ファイルの条件

項目	条件
入力可能なファイル	オブジェクトモジュールファイル リロケータブルロードモジュールファイル ライブラリ
使用できる名前の数	セクション名　　：1ファイルにつき65535個まで シンボル名　　：1ファイルにつき65535個まで モジュール名　　：1ファイルにつき65535個まで ただし、開発環境のシステムのメモリ容量によっては制限を受けます。

2.1.5 出力ファイルの条件

リンカが生成するロードモジュールファイルには、各種名前[注]をそれぞれ 65535 個まで含めることができます。したがって、ロードモジュール内の各種名前の数が 65535 個を超えるようなリンクは実行しないでください。ただし、名前の数は開発環境のシステムのメモリ容量によっては制限を受けます。

2.1.6 出力ファイル名の命名

出力ファイル名は -o オプションで指定した名前となります。指定しなかった場合、リンカによって自動的に以下のように命名されます。

表2.3 出力ファイル名（デフォルト）

ファイル名	内容
`am.out`	リンクの結果出力される、ロードモジュールファイル。
`a.mout`	（EWS版での）リンクの結果出力される、ロードモジュールファイル。

注）各種名前　　セクション名、シンボル名、およびモジュール名

2.2 リンカの起動オプション

表2.4に、リンカの各起動オプションの機能について示します。

表2.4 リンカ起動オプション（1/3）

オプション	機能
-e *entrypoint*	ロードモジュールのエントリポイントを、*entrypoint*（シンボル）に設定します。エントリポイントは、デバッガにおいて、プログラムカウンタの自動初期設定機能等に利用される情報です。
-g	デバッグ時に必要な情報（デバッグ情報）をロードモジュールファイルに出力します。
-l *lib*	*lib*という名前のライブラリを指定します。ライブラリの検索順序は以下のとおりです。 ① -Lオプションで指定したディレクトリ ② 環境変数M32RLIBに設定されているディレクトリ （環境変数が設定されていない場合、 /usr/local/M32R/lib）。
-L *dir*	ライブラリの検索ディレクトリを指定します。

次ページ →

表2.4　リンカ起動オプション（2/3）

オプション	機能

-LOC *addr1,addr2*	CプログラムをROM化するために、各セクションを適するメモリ領域に配置するように制御するオプションです。このオプションは、-SECオプションの簡易版で、セクションがP、D、B、Cの4種類から構成される場合に有効です（ユーザーが独自に用意したセクションがある場合は使えません）。
	
	アドレスは 16進数で指定します。ただし、英文字で始まる16進数の場合は先頭に0を付加してください。
	
	*addr1*には、RAM領域（D,Bセクション配置領域）の先頭アドレスを指定します。セクションのリンク順序はD→Bの順になります。*addr1*番地からのRAM領域には、領域が確保されるだけで、その領域に初期値データは出力されません。
	
	*addr2*には、ROM領域（P,CセクションおよびDセクションの初期値データの配置領域）の先頭アドレスを指定します。セクションのリンク順序はP→C→Dの順になります。ここでのDセクション（初期値データ）は、セクション名ROM_D として出力されます。
	
	この-LOCオプションは、-SECオプションおよび-rオプションとは併用できません。
	
	以下の2つの指定は、どちらも同じ処理を意味します。
	`-LOC 1000,8000`
	`-SEC @D=1000,B,P=8000,C,D`

-M *map_filename*	*map_filename*という名前のリンクマップファイルを出力します。
-o *output_filename*	出力ファイル名を*output_filename*にします。このオプションを省略した場合、出力ファイル名はam.out となります。(EWS版の場合はa.mout)。
-r	ロードモジュールファイルをリロケータブル形式で作成します。このオプションを省略した場合、ロードモジュールファイルはアブソリュート形式となります。 本オプションは、-LOCオプションおよび-SECオプションとは併用できません。

次ページ →

表2.4　リンカ起動オプション（3/3）

オプション	機能
-SEC *name* -SEC *name=addr* -SEC *name=addr,name=addr...*	セクションのリンク順序と各セクションの先頭アドレスを指定します。*name*にはセクション名、*addr*にはそのセクションを配置するアドレスを指定します。 先頭アドレスは = の後に16進数で指定します。ただし、英文字で始まる16進数の場合は先頭に0（ゼロ）を付加してください。先頭アドレスの指定がない場合は、直前のセクションの後に続けて配置されます。 @をセクション名の頭に付けると、そのセクションについてはメモリ領域確保の情報だけを出力し、データは出力しないように制御できます（リンカの初期値データ出力抑止機能）。 @によって出力しないようにした初期値データは、別の領域に出力できます（リンカの初期値データ抽出機能）。コマンド行で、セクション名を再度@なしで指定してください。出力されるセクション名は ROM_*name* となります。 例：　-SEC　@D=1000,B,P=0c000,C,D 上記例では、Dセクションの領域を1000 番地以降に配置し、Dセクションの初期値データをCセクション[16]の後に配置します。初期値データはROM_D というセクション名で、ロードモジュールファイルに出力されます。 この-SECオプションは、-LOCオプションおよび-rオプションとは併用できません。
-V	起動メッセージを標準エラー出力に出力します。他のオプション指定はすべて無視され、実際の処理は行われません。
-w	ワーニングメッセージの表示を抑止します。
-overlap	異なるセクションを同じアドレスに配置させます（オーバーレイ機能）。このオプションを指定すると全てのセクションの重なりをチェックしないようになりますので、注意して使用してください。

オプション	機能
-Werrsec	-SECオプションのセクションに、入力されたオブジェクトモジュールファイルに含まれないセクションを指定した場合、エラーとして処理します。 本オプションがないデフォルトの処理では、このような場合はワーニングとし、処理を続行することができます。
-Wreloc	relocation size overflow エラーが発生した場合に、そのメッセージの直後に位置情報(Position)と、設定値情報(Setting)を表示します。
-Wlimit=message_max	メッセージ表示数の上限(message_max)を設定します。この数値に指定された個数だけメッセージを表示し、それ以降のメッセージは抑止します。数値に0を指定した場合は制限はなくなり、全てのメッセージを表示します。 デフォルトでは、-Wlimit=20 が指定されたのと同じ動作となります。
-Wnolimit	全てのメッセージを表示します。
-Wmangle	メッセージのデマングル表記の後にマングル名を表示します。

2.3 リンカの起動例

以下にリンカの起動例を示します（% はプロンプト、<RET> はリターンキー入力を示します)。

■例1： `% lnk32R -o asmd.abs -e _main add.mo sub.mo mul.mo div.mo <RET>`

add.mo、sub.mo、mul.mo、および、div.mo の４ファイルから、アブソリュートロードモジュールファイルasmd.absが生成されます。ロードモジュールのエントリポイントは、外部シンボル _main のアドレスとなります。エントリポイントは、デバッガでasmd.absをデバッグする際のプログラムカウンタの自動初期設定機能などに利用されます。

■例2： `% lnk32R -M map -r -g add.mo sub.mo mul.mo div.mo <RET>`

add.mo、sub.mo、mul.mo、およびdiv.mo の４ファイルから、リロケータブルロードモジュールファイルam.out(EWS版の場合はa.mout)が生成されます。ロードモジュールにはデバッグの情報が組み込みまれます。リンクマップはファイル map へ出力されます。

■例3： `% lnk32R link.cmd <RET>`

コマンドファイルlink.cmdに書いておいたパラメータが展開され、実行されます。

第３章
ロードモジュールの生成

3.1 アブソリュートロードモジュールの生成

リンカは、オブジェクトモジュールをセクション単位（第４章 セクション　参照）で取り扱い、結合および配置することによって、アブソリュートロードモジュールを生成します。

3.1.1 オブジェクトモジュールの結合（セクションの結合）

1つのアプリケーションを複数のソースファイルに分割してプログラミングすると、同一セクションが複数のオブジェクトモジュールに分散します。それらのオブジェクトモジュールからロードモジュールを生成する場合、リンカは、同一セクションが連続して並ぶようにオブジェクトモジュールを結合します。結合後、1セクション内では、コマンド行で指定された入力ファイルの順番に従って、オブジェクトコードが配置されています。

3.1.2 配置アドレスの指定

リンカでは、起動オプション -SEC または -LOC で、各セクションごとの配置アドレスが指定できるようになっています（配置属性が絶対形式のセクションの場合は、セクション情報で定義されたアドレスに配置します）。

システム組み込み用のアプリケーションを生成する場合は、RAM 上に配置すべきセクションと ROM 上に配置すべきセクションがあります。例えば、プログラムの実行により値を書き換える必要のあるデータをもつセクションは RAM 上に、プログラムセクションや固定データのセクションは ROM 上に配置します（図 3.1 参照）。

次ページ →

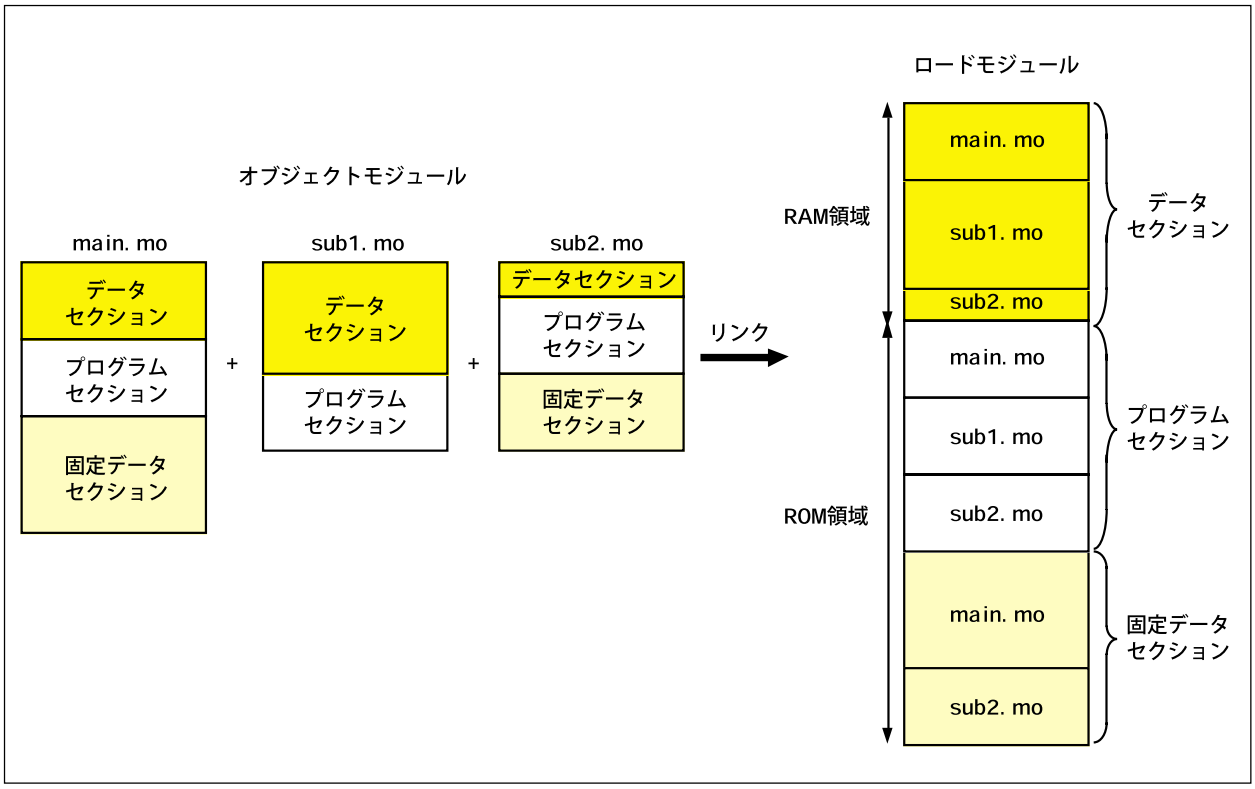


図 3.1　オブジェクトモジュールの結合（セクションの結合）とセクションの配置

3.2 リロケータブルロードモジュールの生成

リンクの結果、絶対アドレスを割り当てない、再配置可能なロードモジュール（リロケータブルロードモジュール）を生成することができます。リロケータブルロードモジュールでは、各セクションごとに、セクションの先頭からの相対アドレスが割り付けられます。リロケータブルロードモジュールを生成するには、起動オプション -r を指定します。例えば、

```
% lnk32R -r start.mo file1.mo file2.mo
```

と指定した場合、リロケータブルロードモジュール am.out（EWS版の場合 a.mout）が出力されます。

3.3 ライブラリとの結合

リンカは、指定された全入力ファイルについて外部参照シンボルの解決処理を行います。どの入力ファイルにも定義されていない外部参照シンボルがあれば、指定されたライブラリの中から、未解決の外部参照シンボルを定義しているモジュール検索し、その外部参照シンボルを含むモジュールのみを抽出した後、結合処理します。もし、指定ライブラリ中に未解決外部参照シンボルを定義しているモジュールが存在しなければ未定義エラーとなります。

ライブラリは起動オプション-lおよび-Lによって指定できます。-lオプションで指定したライブラリは、次の①～②の順に検索されます。

① -Lオプションによる指定ディレクトリ
② M32RLIBで設定されたディレクトリ

ただし、-Lオプションによる検索指定は、-Lオプションより後に指定されたライブラリにのみ有効です。-Lオプションを複数回指定した場合は、指定した順に検索されます。例えば、

```
% lnk32R -l a.lib -L /usr -L /usr/lib -l b.lib file.mo
```

と指定した場合、ライブラリa.libとb.libに含まれるモジュールが検索対象となります。ただし、a.libは「M32RLIBで設定されたディレクトリ」で検索されます。b.libは、 /usr → /usr/lib →「M32RLIBで設定されたディレクトリ」の順に検索されます。

またライブラリは、以下のように入力ファイルとしてコマンド行で直接指定することもできます。

```
% lnk32R test.mo c.lib d.lib
```

ただし、上記c.libや d.libのように、コマンド行で入力ファイルとして指定したライブラリに対しては、各検索指定（「-Lオプションによる指定ディレクトリ」および「M32RLIBで設定されたディレクトリ」）は無効です。また、-lオプションで指定したライブラリよりも先に検索されます。外部参照シンボルを含むモジュールのみの抽出および結合処理の扱いは、-lオプションで指定したライブラリの場合と同じです。

第４章
セクション

4.1 セクションの種類

プログラムの内容は１つまたは複数のセクションに分類されます。リンカがサポートするセクションには表3.1に示す５タイプがあり、プログラムのコードやデータは、これらのうちのいずれかに属します。

表4.1　セクションのタイプ

属性	プログラムの内容
CODE（コードセクション）	プログラムのコード
DATA（データセクション）	データ（固定データ、プログラム実行中に値が変更されるデータなど）
STACK（スタックセクション）	スタック領域
COMMON（コモンセクション）	別々のモジュールが共通で使用する変数
DUMMY（ダミーセクション）	構造体データのメンバ定義

4.2 セクションの定義（セクション情報）

セクションは、セクション名、セクション属性、および配置属性の３要素によって定義され、それらをセクション情報といいます（表4.2）。

表4.2　セクション情報

セクション情報	内容
セクション名	任意の名前
セクション属性	CODE、DATA、STACK、COMMON、DUMMYのいずれか
配置属性	絶対形式（先頭アドレス指定）、相対形式（アライメント指定）のいずれか

C言語ソースプログラムの内容は、Cコンパイラcc32Rでコンパイルすることによって、自動的にセクション定義されます。アセンブリ言語ソースプログラムの場合は、擬似命令 .SECTIONによってセクション定義をしながらソースコードを記述します。セクション定義の方法について詳しくは「M3T-CC32R ユーザーズマニュアル <<C コンパイラ編 >> 第７章 組み込み用アプリケーションの作成」を参照してください。リンカは、セクション情報が一致するセクションは同一セクションであると判断します。

4.3 セクション結合機能

リンカは、セクションを結合するために以下のような機能をサポートしています。

- セクションの自動結合
- セクションの結合順序指定
- セクションの配置アドレス指定

以下より、各機能の詳細を示します。

||||| 注意 |||||

同じセクション名でセクション属性または配置属性が異なるセクションが存在する場合、リンカはエラーメッセージを出力し、処理を中止します。

4.3.1 セクションの自動結合

セクションの結合は、セクション情報（セクション名、セクション属性、配置属性）によってリンカが自動的に行います。リンカは、複数の入力ファイルに分散している同一セクションを集めて結合します。結合形式はセクション属性によって決まります（4.4「セクションの結合形式」参照）。

結合が行われるのは配置属性が相対形式のセクションのみです。配置属性が絶対形式のセクションは、すでに絶対アドレスが割り付けられているため結合処理されません。

4.3.2 セクションの結合順序指定

セクションの結合順序は、起動オプション -SEC のパラメータで指定できます。結合順序はセクション名の記述順となります。例えば、

```
-SEC A,C,B
```

と指定した場合、セクションはA→C→Bの順で結合されます。

起動時に指定されなかったセクションは、結合指定されたセクションをすべて結合した後、それに続いて入力ファイル中のセクション出現順に結合されます。

セクションの結合順序の指定は、生成するロードモジュールがアブソリュート形式の場合のみ可能です。

次ページ →

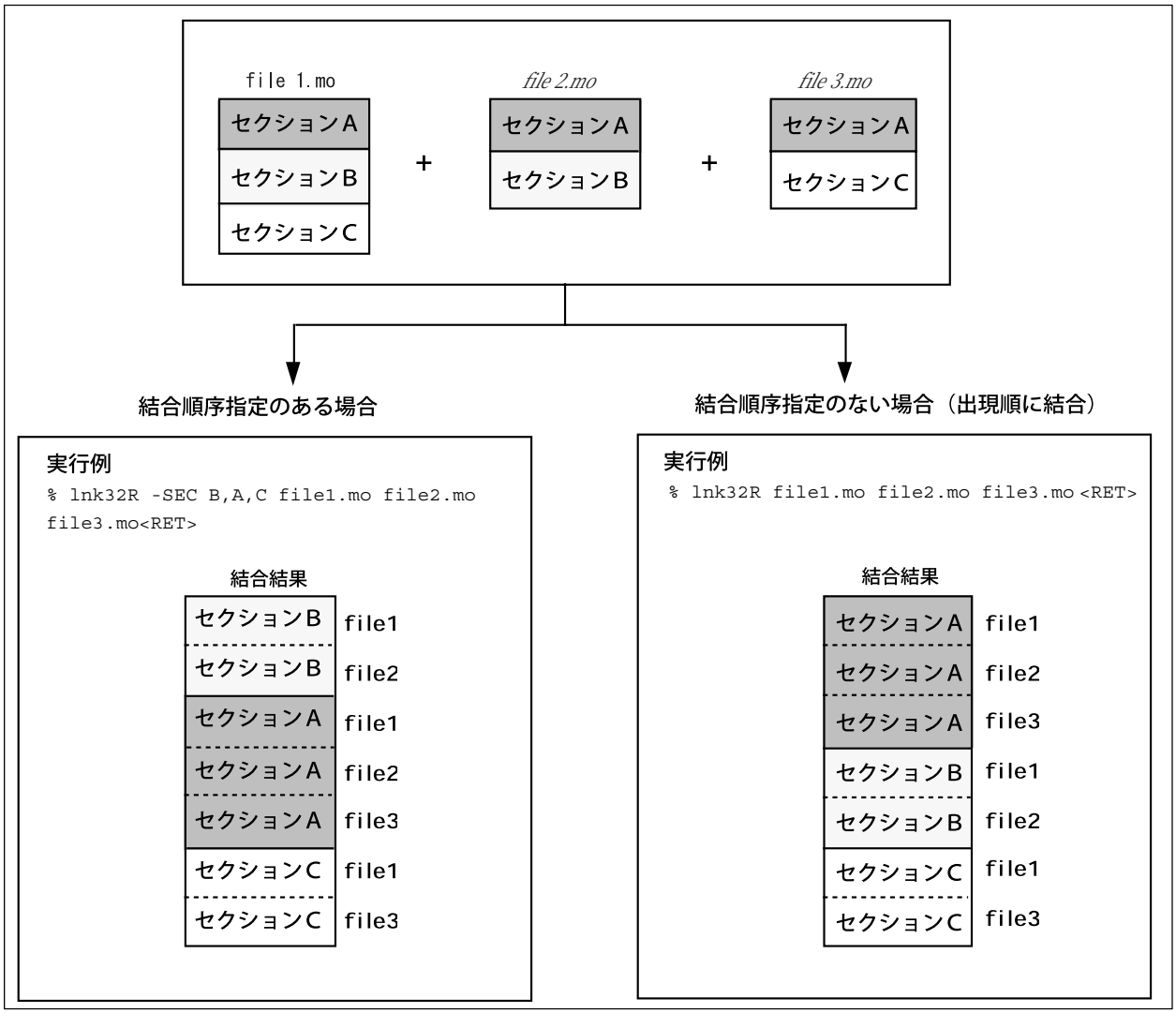


図4.1　セクションの結合

4.3.3　セクションの配置アドレス指定

セクションの配置アドレス（絶対アドレス）は、起動オプション -SEC のパラメータで指定できます。セクションの先頭アドレス（16 進数）をセクション名の後に =XXXXXX の形式で指定します。例えば、

```
-SEC A=1000,C,B
```

と指定した場合、A セクションの先頭が 1000_{16} 番地となります。

配置アドレスが指定されなかったセクションは、先頭に結合されるセクションであれば 0_{16} 番地から、そうでないセクションは結合の結果によって自動的に、絶対アドレスが割り付けられます。

同じ絶対アドレスに対してセクションを重複して割り付けた場合はエラーとなります。

次ページ →

||||| 注意 |||||

絶対アドレス指定オプションには、-SEC のほかに -LOC オプション（ROM 化用）がありますが、-LOC オプションは -SEC オプションとは併用できません。

4.4 セクションの結合形式（セクション属性による）

リンカによるセクションの結合形式は、そのセクションのセクション属性によって決まります（表 4.3）。なお、DUMMY 属性のセクションは、コードの実体が存在しないため、リンク時における結合処理の対象とはなりません。

表4.3 セクション属性と結合形式

セクション属性	結合形式
CODE、DATA、STACK	単純結合
COMMON	共有結合

以下に、各結合形式について示します。

■ 単純結合（セクション属性 CODE、DATA、STACK）

同一セクションが連続したアドレスに配置されます。配置順序はリンカへの入力ファイル指定順に従います。

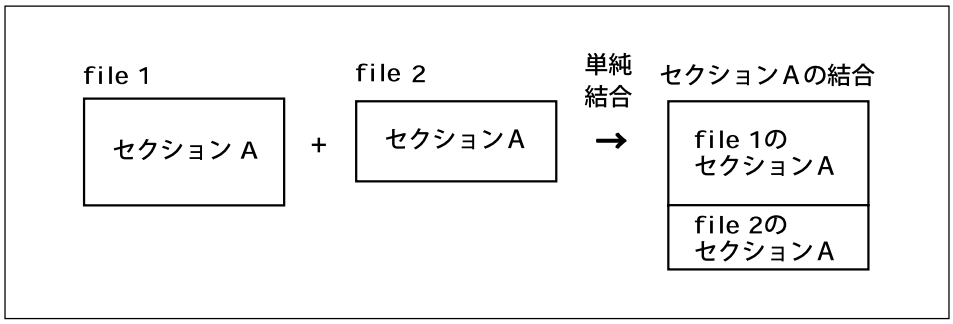


図4.2 単純結合

■ 共有結合（セクション属性 COMMON）

同一セクションが同じアドレスに配置されます。したがって、セクション属性 COMMON のセクションの大きさは、同一セクションのうちで最大サイズのセクションの大きさとなります。

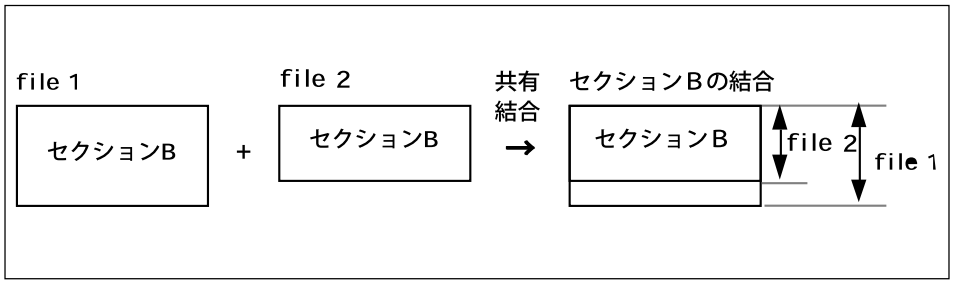


図4.3 共有結合

4.5 セクションの配置形式（配置属性による）

リンカによるセクションの配置形式には絶対形式と相対形式があり、そのセクションの配置属性によって決まります（表4.4）。

表4.4　配置属性と配置形式

配置属性	配置方法
LOCATE=*絶対アドレス*	絶対形式
ALIGN=*アライメント*	相対形式

以下に、各配置形式について示します。

■ 絶対形式（ LOCATE=*絶対アドレス*）

オブジェクトモジュール内において、配置属性が絶対アドレス指定（LOCATE=*絶対アドレス*）であるセクションは、配置アドレスが決定しています。リンカはこのセクションの配置アドレスを変更できません。

■ 相対形式（ALIGN=*アライメント*）

オブジェクトモジュール内において、配置属性がアライメント指定（ALIGN=*アライメント*）であるセクションには、再配置可能な相対アドレスが割り当てられています。実際の配置アドレスは、リンカ起動時のオプションによって指定できます。

セクションの先頭アドレスは、配置属性で指定されたアライメント値の倍数になるよう調整されます。アライメントとは、データなどをメモリに割り付けるときのアドレス調整値です。例えば、ALIGN=4と定義されたセクションは、必ず4の倍数アドレスから割り付けられます。

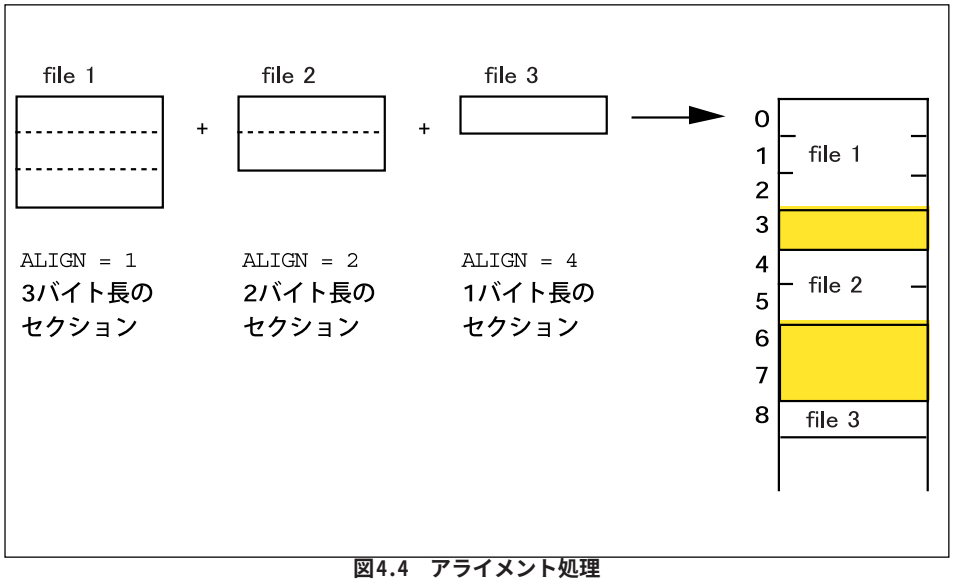


図4.4　アライメント処理

第５章
ＲＯＭ化

5.1 ROM化のためのセクション処理

ここでは、C プログラムを ROM 化するために必要な、セクションの処理について説明します。

C コンパイラが自動的に定義するセクションには以下の４種類があります。

表5.1　Ｃコンパイラが出力するセクション

セクション名	セクション属性	配置属性	概要
P	CODE	ALIGN=4	プログラムコードの領域。
C	DATA	ALIGN=4	定数データ（const宣言された変数）の領域
D	DATA	ALIGN=4	初期値ありデータ領域（初期値を持つ大域変数領域）
B	DATA	ALIGN=4	初期値なしデータ領域（初期値を持たない大域変数領域）

この C プログラムを ROM 化する場合は、以下の指定および処理が必要となります。

- ■ 各セクションの配置先の指定（リンク時）
- ■ D セクションの内容の出力先の指定（リンク時）
- ■ D、B セクションの初期化（スタートアップファイル実行時）

なお、実際に ROM 化するためには、リンクによって生成したロードモジュールファイルを、ロードモジュールコンバータによってモトローラ S フォーマットに変換してください（第５部「ロードモジュールコンバータ lmc32R」参照）。

5.1.1　セクションの配置先の指定（リンク時）

ロードモジュールを生成する際、各セクションの領域が、ROM 領域（メモリ読み出しのみ可能）または RAM 領域（読み書き可能）のいずれか適する方に配置されるように配置先を指定します。配置先の指定は、-SEC または -LOC オプションで行います。

次ページ →

■ P、C セクション　　プログラム実行中に書き換える必要がないため、読み出しのみ可能な ROM 領域に配置する。

■ D、B セクション　　プログラム実行中に書き換える可能性あるため、読み書き可能な RAM 領域に配置する。

5.1.2　セクションの内容の出力先の指定（リンク時）

ロードモジュールを生成する際、通常、各セクションの内容（プログラムコードやデータ）は配置先に出力されます。ただし、D セクションの内容は D セクションを初期化するための初期値であるため、D セクションの配置領域でなく、ROM 領域に出力する必要があります。そのためには、リンカの「初期値データ抑止機能」および「初期値データ抽出機能」を利用します（5.2　ROM 化のための機能　参照）。

5.1.3　データセクションの初期化（スタートアップファイル実行時）

RAM 領域に配置するデータセクション、すなわち D、B セクションの内容は、D、B セクションのデータを使用する C プログラムが実行される前に初期化する必要があります。各セクションの初期化方法は以下のとおりです。通常、以下の処理はスタートアッププログラム（一番最初に実行される初期設定を行うプログラム）で実施します。

■ D セクション　　初期値データ抽出機能によって ROM 領域に書き込んでおいた D セクションの初期値を、RAM 領域に確保しておいた D セクションの領域へ転送する。

■ B セクション　　RAM 領域に確保しておいた B セクションの領域を、ゼロクリア（両域内の全バイトを値 0 により初期化）する。

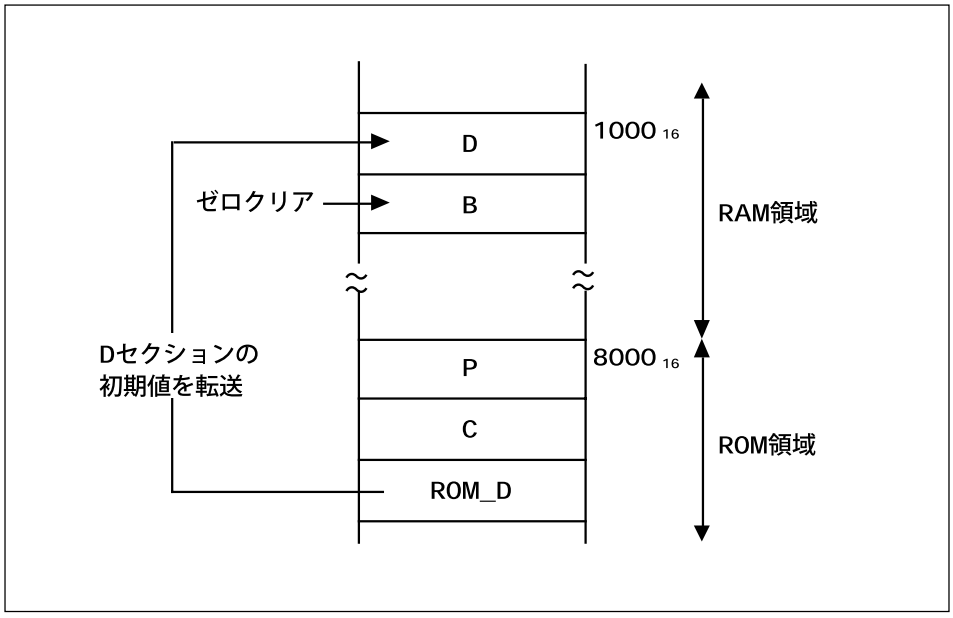


図5.1　データセクションの初期化

次ページ

初期化のためのデータ転送処理には、リンカが生成する予約ラベル（5.2.3　予約ラベルの自動生成機能　参照）を使用すると便利です。

||||| 注意 |||||

実際の組み込み用アプリケーションのスタートアッププログラムでは、この他に、プロセッサの設定やライブラリの初期化などが必要となります。スタートアッププログラムについて詳しくは「M3T-CC32R ユーザーズマニュアル <<C コンパイラ編 >>7.3　スタートアッププログラムの作成」を参照してください。

5.2　ROM化のための機能

5.2.1　初期値データ出力抑止機能

「初期値データ出力抑止機能」とは、メモリ上に確保されたセクションの領域に、そのセクションの内容を出力しない機能です。

-SEC オプションでセクションの配置先を指定した場合、リンカのデフォルト処理では、そのセクションの領域を指定配置先に確保し、そのセクションの内容（コードやデータ）を確保した領域に出力します。例えば、

```
-SEC P=8000
```

と指定すると、P セクションの領域が 8000_{16} 番地から P セクションのサイズ分確保され、その領域内に自動的に P セクションの内容が出力されます。

しかし、D および B セクションのように、RAM 上に配置されプログラム実行前に初期化されるセクションについては、セクションの領域を確保するだけでよく、確保した領域へ初期値データを出力する必要はありません。そのようなセクションに対しては「初期値データ出力抑止機能」を適用します。

「初期値データ出力抑止機能」によってセクションの内容が出力されないようにするには、-SEC オプションでセクションの配置先を指定する際、セクション名の前に @（アットマーク）を付けます。すると、そのセクションがデータを持っていてもどこにも出力されません。そのセクションの領域の内容は不定となります。例えば、

```
-SEC @D=1000,B,P=8000
```

と指定すると、D セクションの領域が 1000_{16} 番地から D セクションのサイズ分確保され、その領域内には何も出力されません（内容不定）。C コンパイラの出力する B セクションは、初期値データを持たないためデータ出力を抑止する必要はありません（-SEC オプションによる指定時に @ を付ける必要がない）。

なお、出力を抑止した初期値データを、改めて別の場所（通常、ROM 領域）へ出力したい場合は「初期値データ抽出機能」を使用します（次項　参照）。

5.2.2　初期値データ抽出機能

「初期値データ抽出機能」とは「初期値データ抑止機能」と併せて用いる機能で、データ出力を抑止したセクションの内容を、指定する領域（通常、ROM 領域）に出力する機能です。

通常、D セクションのような「セクション領域を RAM 上にとり、初期化用データを ROM 上に置く」という配置が必要なセクションを含むプログラムを ROM 化する場合、ロードモジュールを ROM に書き込む時点で、D セクション内の初期値データを ROM 領域へ移動させる作業が必要となります。しかし、リンカの初期値データ抽出機能を利用すれば、その作業をロードモジュールを作成する時点で行うことができます。これによって、ROM 化作業の簡略化が図れます。

「初期値データ抽出機能」によって、セクションの内容を、指定する領域に出力させるには、以下の 2 とおりの方法があります。

■ -SEC オプションで指定する方法

@を付けて初期値データ出力抑止機能を適用したセクションについて、もう一度@なしで配置先を指定します。

例：　`-SEC @D=1000,B,P=8000,C,D`

ただし、C コンパイラが定義する B セクションは、初期値データを持たないので初期値データ抽出機能を使う必要はありません。

■ -LOC オプションで指定する方法

RAM 領域の先頭アドレスと ROM 領域の先頭アドレスを指定します。

例：　`-LOC 1000,8000`

-LOC オプションは、C コンパイラが自動的に定義するセクションで構成されたプログラムにのみ有効で、初期値データ出力抑止機能・初期値データ抽出機能ともに自動的に適用されます。

初期値データ抽出機能によって抽出されたデータの領域には、自動的に「ROM_ セクション名」という名前が付きます。これは通常のセクションと同様に扱うことができます。例えば、D セクションの初期値データを抽出した場合、初期値データが配置された領域は ROM_D セクションとなります。

初期値データ抽出機能は、D セクションに限らずリンカが扱うすべてのセクションに対して適用できます。例えば、セクション P および C を低速な ROM 上から高速な RAM 上に転送して実行するアプリケーションにも有効です。

次ページ →

初期値データ抽出機能の適用例を以下に示します。

例：　% lnk32R -SEC @D=1000,B,P=8000,C,D file1.mo file2.mo
あるいは、
% lnk32R -LOC 1000,8000 file1.mo file2.mo

上記の２例はいずれも同じ出力結果となります。

1000_{16} 番地以降（RAM 領域）には、初期値データが出力されていない D セクションと B セクションが配置されます。8000_{16} 番地以降（ROM 領域）には、データが出力された P セクション、C セクション、および、ROM_D セクションが配置されます。ROM_D セクションには D セクション用の初期値データが出力されます。なお、ここでは、RAM 領域の先頭が 1000_{16} 番地、ROM 領域の先頭が 8000_{16} 番地と想定しています。

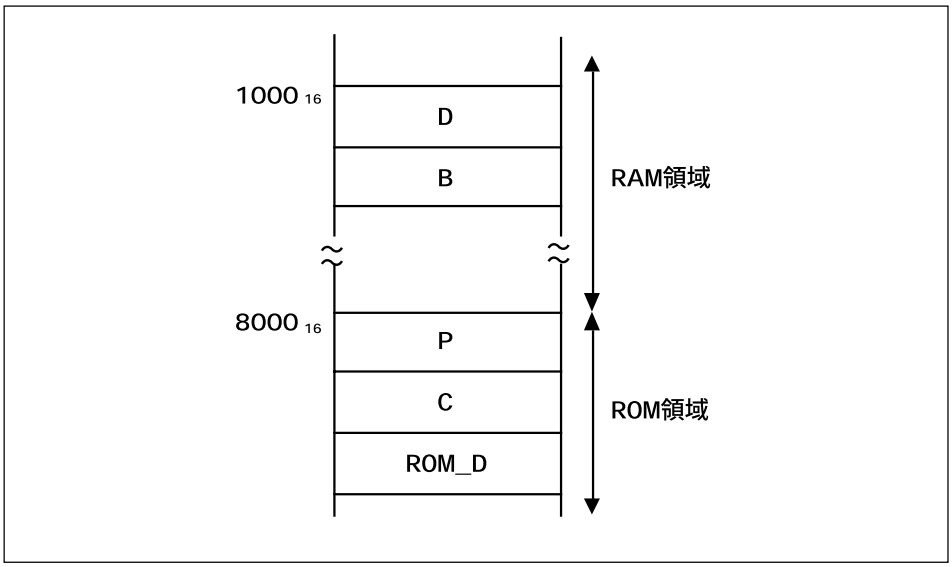


図5.2　初期値データ抽出機能の適用結果

5.2.3　予約ラベルの自動生成機能

リンカは、各セクションの先頭アドレスおよび最終アドレスを示すラベルを自動生成します。セクションの先頭を表すラベルはセクション名の頭に _ _TOP_ が付いた名前、セクションの終端（セクション内の最後のバイトの次のアドレス）を表すラベルはセクション名の頭に _ _END_ が付いた名前となります。例えば、 Dセクションの場合、先頭アドレスは _ _TOP_D 、最終アドレスは _ _END_D で表されます（図5.3）。

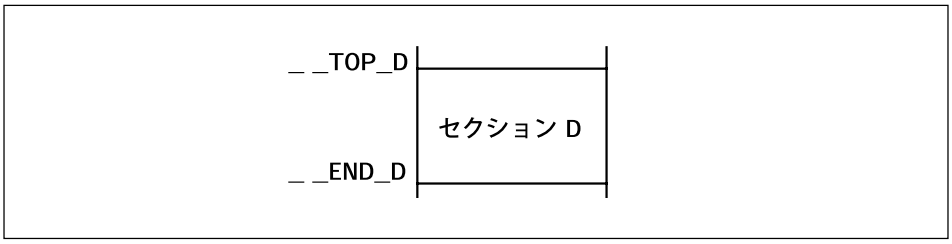


図5.3　予約ラベルの生成（D セクションの例）

予約ラベルを用いると、初期値データの転送処理（D セクションの初期化に相当）や、ゼロクリア処理（B セクションの初期化に相当）を C 言語で記述することができます。なお、予約ラベルはリンカが扱うすべてのセクションに対して、自動的に出力されます。

第６章
リンカのメッセージ

6.1 リンカの実行結果を知るには

リンカの実行結果は、メッセージおよび終了コードから判断できます。

6.1.1 メッセージの出力形式

リンカは動作中にエラーを検出すると、実行状況を示すメッセージを標準エラー出力（通常は画面）に出力します。メッセージの出力形式を以下に示します。

■ 基本形式

ツール名：*入力情報*：*メッセージ種別*：*メッセージ*

※「入力情報:」は必要時のみ出力

■ パターン

```
lnk32R: ファイル名: メッセージ種別: メッセージ
lnk32R: <コマンド行>: メッセージ種別: メッセージ
lnk32R: メッセージ種別: メッセージ
```

※下線部分は入力情報（下線は出力されません）。

■ 出力例

```
lnk32R: error: cannot open file "abc.mo"
```

ツール名　メッセージ種別　　メッセージ

6.1.2 メッセージ種別

メッセージは警告程度によって、表6.1のように分類されています。

表6.1　メッセージ種別

メッセージ種別（表示）	エラー発生時の動作
ワーニング（warning）	ワーニングメッセージを出力し、処理を続行します。
エラー（error）	エラーメッセージを出力し、処理を中止します。
重大エラー（fatal）	エラーメッセージを出力し、処理を中止します。

メッセージの詳細は、「6.2　リンカのメッセージ一覧」を参照してください。

6.1.3 終了コード

リンカは実行後、以下の終了コード（実行結果を示す値）を返します（表6.2）。

表6.2 終了コード

終了コード	実行結果
0	正常終了した。または、ワーニングが発生した。
1	エラーが発生した。

6.2 リンカのメッセージ一覧

6.2.1 ワーニング

表6.3 リンカのワーニングメッセージ

メッセージ	意味 → リンカの動作
external symbol not defined: "*symbol*"	未定義の外部シンボルを参照しています（-rオプション指定時）。 → 外部参照は無効になり、0を仮定します。
option *option* specified more than once, last setting taken	同じオプション（パラメータを伴う）を重複して指定しています。 → 後のオプションのパラメータが有効となります。

6.2.2 エラー

表6.4 リンカのエラーメッセージ（1/3）

メッセージ	意味 → 対策
cannot close file "*filename*"	該当ファイルがクローズできません。 → ディスク容量を確認してください。
cannot create file "*filename*"	該当ファイルが生成できません。 → ファイル名およびディスク容量を確認してください。
cannot execute *command_name*	コマンドが見つかりません。 → コマンドパスの確認、および環境変数M32RBINを設定してください。
cannot open file "*filename*"	該当ファイルが見つかりません。 → ファイル名を確認してください。
duplicate symbol "*symbol*"	外部定義シンボルが重複しています。 → 外部定義シンボルをチェックし、重複しないようにしてください。

表6.4 リンカのエラーメッセージ (2/3)

メッセージ	意味 → 対策
external symbol not defined: "*symbol*"	未定義の外部シンボルを参照しています。 → 未定義の外部シンボルを定義してください。
illegal file format: ID=*number*	指定ファイルのフォーマットが不正です。 → ファイル内容を確認してください。
illegal option "*option*"	不正なオプションを指定しています。 → 正しいオプションを指定してください。
illegal option parameter "*parameter*"	不正なオプションパラメータを指定しています。 → 正しいオプションパラメータを指定してください。
illegal section name "*section_name*"	不正なセクション名を指定しています。 → 正しいセクション名を指定してください。
illegal section start address: *address*	不正なセクション開始アドレスを指定しています。 → 正しいセクション開始アドレスを指定してください。
illegal symbol name "*symbol*"	不正なシンボル名を指定しています。 → 正しいシンボル名を指定してください。
multiple entry points exist: *address*	実行開始アドレス指定のあるモジュールを複数指定しています。 → 実行開始指定のあるモジュールを１つにしてください。
option *option* and *option* cannot be specified together	同時に指定できないオプションを指定しています。 → どちらか一方のオプションのみ指定してください。

表6.4 リンカのエラーメッセージ（3/3）

メッセージ	意味 → 対策
option *option* requires parameter	パラメータ指定が必要なオプションにパラメータが指定されていません。 → パラメータを指定してください。
relocation size overflow: "*section_name*", *offset*, "*symbol*"	未定義シンボルの演算結果が、ユーザーの定義したサイズを超えています。 ・セクション名 ：オーバーフローの発生したセクションの名前 ・オフセット値 ：オーバーフローの発生した位置（セクションの先頭からのオフセット値） ・シンボル名 ：シンボルの名前（シンボルがある場合のみ） → サイズ定義を大きくしてください。"
section address misaligned: "*section_name*"	セクションの開始アドレスが境界調整数と矛盾しました。 → アドレス指定値および境界調整数を確認してください。
section address overflow: "*section_name*"	セクションの割り付けアドレスが許容範囲（0000000016-FFFFFFFF16）を超えました。 → プログラム構成を変更してください。
section attribute inconsistent: "*section_name*"	属性の違う同名セクションを入力しています。 → 属性を同じにするか、セクション名を変更してください。
section form inconsistent: "*section_name*"	形式の違う同名セクションを入力しています。 → 形式を同じにするか、セクション名を変更してください。
section not found: "*section_name*"	指定されたセクションが見つかりません。 → セクション名を確認してください。
section overlap: "*section_name*"	セクションがオーバーラップしています。 → セクションがオーバーラップしないように配置アドレスを変更してください。
unsupported module type: version *number*	サポートしていないタイプのモジュールです。 → モジュールが正しく生成できているか確認してください。

6.2.3 重大エラー

表6.5 リンカの重大エラーメッセージ

メッセージ	意味 → 対策
file seek error	該当ファイルがシークできません。 → ディスク内容を確認してください。
internal error	内部エラーが発生しています。 → 当エラー発生時は至急ご連絡ください。
out of disk space	該当ファイルへ出力できません。 → ディスク容量を確認してください。
out of memory	リンカが動作するためのメモリが不足しています。 → メモリの拡張を行うか、プログラムを変更してください。

第３部

マップジェネレータ　map32R

第１章
マップジェネレータmap32Rの概要

1.1 概要

map32R はM3T-CC32R クロスツールキットに含まれているマップジェネレータです。C コンパイラ、リンカ、およびライブラリアンによって生成された、オブジェクトモジュール、アブソリュートロードモジュール、およびリロケータブルロードモジュールを読み込んで、リンクマップを生成します。
また、ロードモジュールファイルのシンボル情報から、アクセス制御ファイル、csv シンボルマップファイルを生成します。

リンクマップとは、各セクションの配置情報を示す「マップリスト」と外部定義シンボル情報を示す「外部定義シンボルリスト」から構成されるリストです。外部定義シンボル情報の出力順は、アルファベット順またはアドレス順のいずれかを指定できます。

アクセス制御ファイルとは、ベースレジスタ機能（M3T-CC32R ユーザーズマニュアル <<C コンパイラ編>> 「A.1　ベースレジスタ機能」 参照）を使用するために、

(1) 16 ビットレジスタ相対間接の基点（ベース）となるアドレス
(2) 基点（ベース）となるアドレスを保持するレジスタ
(3) ベースレジスタ機能を適応するオブジェクト（変数や構造体）。

を記述したファイルです。

csv シンボルマップファイルとは、アドレスとシンボル(変数やオブジェクト、および関数)名のマップ(csvシンボルマップ)をcsv形式で記述したファイルです。
csv シンボルマップは表形式で、アドレスを第 1 カラムとして、第 2 カラム以降にシンボルを表示します。csv形式になっており、Microsoft Excel等の表計算ソフトに入力することができます。

第２章
マップジェネレータの起動

2.1 マップジェネレータを起動するには

2.1.1 マップジェネレータの起動手順

マップジェネレータを起動するためには、環境変数を設定（2.1.2参照）した後、規則に従ってコマンド「map32R」を入力し、それを実行します（2.1.3参照）。

2.1.2 環境変数の設定

環境変数M32RBIN、M32RINC、M32RLIB、M32RTMPに正しいディレクトリを設定します（通常はインストール時に設定します）。設定方法は「M3T-CC32Rクロスツールキット V.XX Release Xリリースノート」を参照してください。環境変数の設定を省略すると、デフォルトのディレクトリを設定したことになります。

表2.1 環境変数のデフォルト

環境変数	デフォルトのディレクトリ
M32RBIN	/usr/local/M32R/bin
M32RINC	/usr/local/M32R/include
M32RLIB	/usr/local/M32R/lib
M32RTMP	/tmp

2.1.3　コマンド行の記述規則

マップジェネレータ起動コマンド「map32R」の、コマンド行の入力書式および規則を以下に示します。起動オプションおよび入出力ファイルについての詳細は、2.1.4節以降を参照してください。

```
map32R  [-o  output_filename]  [-V]  [-s]  [-n]
        [-Rn=Address]  [-Pn[=filename]]  [-Pd]
        [-Ps[=filename]]
        [-c]  [-c16]
        [-debug_no]  [-debug_sort_name]
        [-debug_sort_addr]  [-debug_sort_attr]
        [-debug_no_func]  [-debug_no_label]
        [-debug_no_var]  [-debug_no_global]
        [-debug_no_local]
        [object_filename]  <RET>
```

※[ ]で囲まれていない項目　　：必ず入力しなければならない項目
※[ ]で囲まれた項目　　：必要に応じて記述する項目
※　－の付いた記号　　：起動オプション（詳細は2.3節参照）
※<RET>　　：リターンキー入力

図2.1　map32Rコマンドの入力書式

■ コマンド行は、上記（図2.1）の書式に従って記述します。各項目（コマンド名、オプション、入力ファイル名）の間は、1文字以上の空白文字で区切ります。最後にリターンキーを入力すると、マップジェネレータが実行を開始します。

■ オプションとそのパラメータとの間には空白文字が必要です。矛盾するオプションの組み合わせがあれば、後から指定したオプションが有効となります。

■ *object_filename*には、入力ファイル名を1つだけ指定します。入力ファイル名を省略した場合、自動的にam.out(EWS版の場合はa.mout）が入力ファイルとなります。

■ 入力ファイルは、ファイル名の拡張子に関わらず、すべてオブジェクト形式のファイル（オブジェクトモジュール、ロードモジュール、ライブラリ）と判断されます。

||||注意||||

名前（モジュール名、セクション名、シンボル名）の長さが20文字を超えるとマップリスト形式が乱れる場合があります。注意してください。

2.1.4 入力ファイルの条件

マップジェネレータで処理できる入力ファイルの条件を以下に示します。これらの条件を満たさないファイルは入力しないでください。

表2.2 入力ファイルの条件

項目	条件
入力可能なファイル	オブジェクトモジュールファイル リロケータブルロードモジュールファイル アブソリュートロードモジュールファイル ライブラリ
使用できる名前の数	セクション名　：1ファイルにつき65535個まで シンボル名　：1ファイルにつき65535個まで モジュール名　：1ファイルにつき65535個まで ただし、開発環境のシステムのメモリ容量によっては制限を受けます。

2.1.5 出力ファイル名の命名

出力ファイル名は -o オプションで指定した名前となります。指定しなかった場合、標準出力へ出力されます。

2.2 マップジェネレータの起動オプション

表２.３に、マップジェネレータの各起動オプションの機能について示します。

表２.３(1/2)　マップジェネレータ起動オプション

オプション	機能
-n	マップリストの後ろに外部定義シンボルリストをアドレス順に出力します。
-o *output_filename*	マップリストおよび外部定義シンボルリストを、*output_filename*という名前のファイル（リンクマップファイル）に出力します。このオプションを省略すると標準出力に出力します。
-s	マップリストの後ろに外部定義シンボルリストをアルファベット順 に出力します。
-V	起動メッセージを標準エラー出力に出力します。他のオプション指定はすべて無視され、実際の処理は行われません。
-R*n*=*Address*	ベースレジスタ(n=11～13)とベースアドレス(16進数)を指定して、アクセス制御ファイルを生成します。（リンクマップは生成しません。）
-Pd	データシンボルの総数とヒットした(ベースレジスタを適用できると判定した)シンボルの数を表示しません。
-Pn[=file_name]	ベースレジスタにヒットしなかったデータシンボルのリストを表示(-nオプションと同様の形式)します。オプションに続けて =ファイル名を指定すればそのファイルに、ファイル名指定がない場合には標準出力に出力します。
-Ps[=file_name]	指定したベースレジスタ構成に基づき、スタートアッププログラムのサンプルを出力します。オプションに続けて =ファイル名を指定すればそのファイルに、ファイル名指定がない場合には標準出力に出力します。
-c	1アドレス形式（1行にシンボルをひとつを表示）で、csvシンボルマップを出力します。 -o オプションがある場合はそのファイルに、-o がない場合は標準出力に出力します。-c 指定時はリンクマップファイルは生成しません。
-c16	16アドレス形式（1行に16バイト分の領域に属するシンボルを表示）で、csvシンボルマップを出力します。-o オプションがある場合はそのファイルに、-o がない場合は標準出力に出力します。-c16 指定時はリンクマップファイルは出力しません。
-debug_no	デバッグソースリストとデバッグシンボルリストを出力しません。
-debug_sort_name	名前（SYMBOL）順にデバッグシンボルリストを出力します。

表 2.3(2/2)　マップジェネレータ起動オプション

オプション	機能
-debug_sort_addr	アドレス（ADDR.）順にデバッグシンボルリストを出力します。
-debug_sort_attr	属性（ATTR.）順にデバッグシンボルリストを出力します。
-debug_no_func	関数をデバッグシンボルに出力しません。
-debug_no_label	ラベル名をデバッグシンボルに出力しません。
-debug_no_var	変数をデバッグシンボルに出力しません。
-debug_no_global	大域変数をデバッグシンボルに出力しません。
-debug_no_local	ローカル変数をデバッグシンボルに出力しません。
-mangle	デマングル表記の前にマングル名を表示します。
-mangle_only	デマングル表記のかわりにマングル名を表示します。
-old_index	情報の並びを、V.4.xx以前と互換の形式で表示します。
-c_demangle	-csvシンボルマップのひとつである、マングル名インデックス形式を選択します。

2.3 マップジェネレータの起動例

マップジェネレータの起動例を示します（% はプロンプト、<RET> はリターンキー入力を示します)。

■例１：　% map32R -o sample.lst sample.abs <RET>

アブソリュートロードモジュールsample.absのマップリストおよび外部定義シンボルストをsample.lstに出力します。

■例２：　% map32R <RET>

ロードモジュールam.out(EWS版の場合はa.mout)のマップリストおよび外部定義シンボルリストを標準出力に出力します。

■例3：　% map32R -R13=F78000 -R12=F88000 -R11=FC8000 -o sample.acc -Ps=startsmp.ms sample.abs <RET>

ロードモジュールファイル sample.abs からベースアドレス機能を使用するためのファイルを作成します。
ベースアドレスおよび、ベースレジスタ機能の対象アドレス を、

0x00F78000 を R13
0x00F88000 を R12
0x00FC8000 を R11

とし、アクセス制御ファイルを、”sample.acc ”、ベースレジスタ設定のサンプルファイルを、”startsmp.ms という名前で作成します。また、標準出力にベースレジスタで対応できなかったシンボル名を表示します。

第３章
リンクマップファイル

3.1 リンクマップファイルの内容

リンクマップファイルには、マップリスト（MAP LIST）および外部定義シンボルリスト（GLOBAL SYMBOL LIST）が出力されます。以下にその例を示します。

```
Input file:  filename
Module type: relocatable load module

MAP LIST
SECTION     TYPE JOINT  MODULE      ATR. START     LENGTH    ALIGN.
P           CODE NOSHR  test1       ABS  00000000  00000518       4
                        test3       ABS  00000518  00000718       4
D           DATA SHR    test2       REL  00000718  0000003f       4
B           DATA DUMMY  test3       REL  00000758  00000128       4

GLOBAL SYMBOL LIST
SYMBOL      ADDR.       TYPE    SEC.
$func1      00000012    DAT     P
LABEL1      00000d02    DAT     D
LABEL2      00000e12    DAT     D

SYMBOL1     000000ff    EQU     B
SYMBOL2     00000001    EQU     B
```

図3.1　リンクマップファイルの例

3.2、3.3に、マップリストおよび外部定義シンボルリストの意味を示します。

3.2 マップリストの内容

マップリストは、入力ファイルにおけるセクションの配置情報を示すものです。マップリスト内の記述は以下のようになっています。

```
Input file: filename ——(1)
Module type: relocatable load module ——(2)

MAP LIST
SECTION   TYPE   JOINT   MODULE    ATR   START      LENGTH     ALIGN.
P         CODE   NOSHR   test1     ABS   00000000   00000518   4
                         test3     ABS   00000518   00000718   4
D         DATA   SHR     test2     REL   00000718   0000003f   4
B         DATA   DUMMY   test3     REL   00000758   00000128   4
(3)       (4)    (5)     (6)       (7)   (8)        (9)        (10)
```

(1) Input file（入力ファイル名）

(2) Module type（入力ファイルの属性）

objectmodule	：オブジェクトモジュール
relocatableloadmodule	：リロケータブルロードモジュール
absoluteloadmodule	：アブソリュートロードモジュール

(3) SECTION（セクション名）

(4) TYPE（セクション属性）

DATA	：データ
CODE	：コード
DUMMY	：ダミー
STACK	：スタック
文字列"XXXXX"	：属性不明

(5) JOINT（結合方法）

SHR	：共有結合
NOSHR	：単純結合
DUMMY	：ダミー結合
文字列"XXXXX"	：結合方法不明

(6) MODULE（モジュール名）

(7) ATR.（配置属性）

REL	：相対形式
ABS	：絶対形式
文字列"XXX"	：属性不明

次ページ →

(8) START（オブジェクトの開始アドレス（16 進数））

(9) LENGTH（オブジェクトのアドレス長（16 進数））

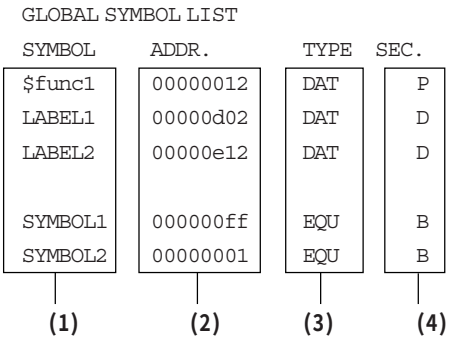
(10) ALIGN.（リンク時のロケーションカウンタ調整値（アライメント））

3.3 外部定義シンボルリストの内容

外部定義シンボルリスト（GLOBAL SYMBOL LIST）は外部定義シンボル名とその値を示すもので､起動オプション-s または -nが指定された場合のみ出力されるものです。リストの前半部にラベル、関数名、変数名（TYPE=DAT）の情報が出力され、後半部にシンボル（TYPE=EQU）が出力されます。外部定義シンボルが存在しない場合、文字列"no symbol"が出力されます。

シンボルの出力順は、-sオプション指定時はシンボル名のアルファベット順、-nオプション指定時はシンボル値順となります。ただし、シンボル（TYPE=EQU）部は常にアルファベット順です。

外部定義シンボルリスト内の記述は以下のようになっています。



(1) SYMBOL（外部定義シンボル名）

(2) ADDR.（シンボル値）

入力ファイルの属性が相対形式の場合、シンボル値は保証されません。

(3) TYPE（シンボル種別）

DAT ：関数名、変数名、ラベル

EQU ：シンボル

(4) SEC.（シンボルが定義されているセクション名）

3.4 map32R の拡張出力項目について

デバッグ情報が有効なロードモジュールを入力した場合、デバッグソースリスト（ソースファイル名のリスト）と、デバッグシンボルリスト（関数名、静的変数名、ラベル名など）をマップファイルに出力することができます。

◆使用例

[ソースファイル sample1.c]

```
1
2  int global;
3
4  void foo1 (int arg)
5  {
6   static int static_local = 10;
7
8   for (global = 0; global < 10; global++)
9      static_local += glboal;
10 }
```

[ソースファイル sample2.c]

```
1
2  static int static_global;
3
4  void foo2( int arg )
5  {
6   int local;
7
8   for (static_global=0; static_global < 10; static_global++)
9       local += static_global;
10 }
```

[コマンドライン]

```
% cc32R -o sample.abs -SEC P=0FC0000,C,D=8000,B sample1.c sample2.c
% map32R -o sample.map sample.abs
```

[生成されるマップファイル sample.abs]

```
Input file:  filename
Module type: relocatable load module

Input file : sample.abs
Module type : absolute load module

MAP LIST

SECTION  TYPE  JOINT MODULE    ATR. START     LENGTH    ALIGN.
D        DATA  NOSHR sample1   ABS  00008000  00000004       4
B        DATA  NOSHR sample1   ABS  00008004  00000004       4
                     sample2   ABS  00008008  00000004       4
P        CODE  NOSHR sample1   ABS  00fc0000  00000044       4
                     sample2   ABS  00fc0044  00000040       4

DEBUG SOURCE LIST

[1] sample1.c
[2] sample2.c

DEBUG SYMBOL LIST

SYMBOL              ATTR.       ADDR.          SIZE     SOURCE
static_local        VAR|LOCAL   0x00008000        4     [1] 6
global              VAR|GLOBAL  0x00008004        4     [1] 2
static_global       VAR|LOCAL   0x00008008        4     [2] 2
foo1                FUN|GLOBAL  0x00FC0000        0     [1] 4
foo2                FUN|GLOBAL  0x00FC0044        0     [2] 4
```

図3.2　リンクマップファイル拡張出力の例

[各項目の意味]

(1) DEBUG SOURCE LIST（デバッグソースリスト）

ロードモジュールに含まれる、ソースファイル名のリストです。
名前の前の[数字]は、ソースファイル番号を表し、次のデバッグシンボルリストでソースファイルの特定に用います。

(2) DEBUG SYMBOL LIST（デバッグシンボルリスト）

ロードモジュールに含まれる、デバッグシンボル名のリストです。
このリストには、次の属性のシンボルを表示します。

属性	ATTR の表記
関数名	FUN
アセンブララベル	LAB
Ｃソース変数	VAR
外部シンボル	GLOBAL
内部シンボル	LOCAL

デバッグシンボルリストでは、シンボルごとに次のフィールドに各シンボルの情報を表示します。

SYMBOL	名前
ATTR.	属性
ADDR.	アドレス
SIZE	変数のサイズ
SOURCE	[] に囲まれたソースファイル名番号(デバッグソースリストの番号)、および行番号。

[関連オプション]

-debug_no	デバッグソースリストとデバッグシンボルリストを出力しません
-debug_sort_name	名前（SYMBOL）順にデバッグシンボルリストを出力します
-debug_sort_addr	アドレス（ADDR.）順にデバッグシンボルリストを出力します
-debug_sort_attr	属性（ATTR.）順にデバッグシンボルリストを出力します
-debug_no_func	関数をデバッグシンボルに出力しません
-debug_no_label	ラベル名をデバッグシンボルに出力しません
-debug_no_var	変数をデバッグシンボルに出力しません
-debug_no_global	大域変数をデバッグシンボルに出力しません
-debug_no_local	ローカル変数をデバッグシンボルに出力しません

第４章
アクセス制御ファイル生成機能

map32Rは、ロードモジュールファイルのシンボル情報から、アクセス制御ファイルを生成することができます。アクセス制御ファイルについては、(M3T-CC32R ユーザーズマニュアル <<Cコンパイラ編>>「A.1.7 アクセス制御ファイルについて」)を参照してください。
作成されるアクセス制御ファイルは、コンパイラcc32Rの -access オプションに指定できます。

4.1 アクセス制御ファイル生成機能の詳細

●”-Rn=Address ”オプションを指定すると、ロードモジュールファイルから cc32Rの -access オプションで入力可能なアクセス制御ファイルを作成します(マップ出力は行いません)。

●アクセス制御ファイルは “ -o ファイル名 ”で指定します(省略すると標準出力に出力します)。

●ベースレジスタは R11～R13の3種類を指定できます。同じレジスタを2回以上指定した場合は、最後の指定を採用します。

●ベースレジスタの指す領域が重複している場合は、次のようなワーニングを表示します。

```
Base Register Area is Overlapped: R13 and R12.
```

●map32Rは、ベースレジスタの定義と、そのベースレジスタがカバー可能なデータシンボルを、先頭の _ (アンダーバー) 文字を削除した上で変数名としてアクセス制御ファイルに並べていきます。

【データシンボルの条件】

次のいずれかであればデータシンボルと判定します。

(1) どのセクションにも属していない。
(2) .EQUで定義された定数ラベル。
(3) セクションに属していて、そのセクションはコード属性でなく'C'という名前ではない。

●デフォルトでは、処理後にデータシンボルの総数と、ヒットしたシンボルの数を表示します。

次ページ →

●-Pd，-Ps，-Pn オプションは、-Rn=... と併用しない場合は無視します。
●-Psで出力される設定プログラムは、$_Set_Regbase というアセンブラのサブルーチンの形式です。スタートアップルーチンから BL あるいは JL 命令で呼び出せば必要なベースレジスタの設定を行うことができます。
● -Ps で出力されるプログラムは、Must とコメントの打たれている行のみを切り出して利用することもできます。

4.2 アクセス制御ファイル生成機能の使用例

ロードモジュールファイル sample.abs からベースアドレス機能を使用するためのファイルを作成します。
ベースアドレスおよび、ベースレジスタ機能の対象アドレスとして、

0x00F78000 を R13
0x00F88000 を R12
0x00FC8000 を R11

とし、アクセス制御ファイルを、”sample.acc ”、ベースレジスタ設定のサンプルファイルを、”startsmp.ms という名前で作成します。また、標準出力にベースレジスタで対応できなかったシンボル名を表示します。

[コマンドラインでの指定]

```
map32R -R13=F78000 -R12=F88000 -R11=FC8000 -o sample.acc -Ps=startsmp.ms sample.abs
```

[画面表示例]

```
Count of Data Symbol(s):        10
Data Symbol(s) that hit:        6
```

（この表示はデータシンボルが 10 個あり、うち 6 つが R13 ～ R11 のいずれかを使ったレジスタ相対の範囲内であるので、アクセス制御ファイルに出力したことを示します。）

[アクセス制御ファイル sample.acc の出力例]

```
@R13  0xF78000
var1
var2
var3

@R12  0xF88000
var4
var5

@R11  0xFC8000
var6
```

次ページ →

[ベースレジスタ設定プログラムのサンプル startsmp.ms の出力例]

```
      .SECTION   P,CODE,ALIGN=4
      .EXPORT    $_Set_Regbase
$_Set_Regbase:
      SETH       R13,#HIGH(__REL_BASE13)       ; Must
      OR3        R13,R13,#LOW(__REL_BASE13)    ; Must
      SETH       R12,#HIGH(__REL_BASE12)       ; Must
      OR3        R12,R12,#LOW(__REL_BASE12)    ; Must
      SETH       R11,#HIGH(__REL_BASE11)       ; Must
      OR3        R11,R11,#LOW(__REL_BASE11)    ; Must
      JMP        R14

      .EXPORT    __REL_BASE13                  ; Must
      .EXPORT    __REL_BASE12                  ; Must
      .EXPORT    __REL_BASE11                  ; Must
__REL_BASE13:   .EQU  0x00F78000               ; Must
__REL_BASE12:   .EQU  0x00F88000               ; Must
__REL_BASE11:   .EQU  0x00FC8000               ; Must
```

4.3 注意事項

●アクセス制御ファイルは、デバッグ情報を使って作成します。
デバッグ情報のないロードモジュールからアクセス制御ファイルを作成しても、シンボル名は出力されません。

●ベースシンボル(__REL_BASExx 等)はデータシンボルには含みません。

第５章
csvシンボルマップファイル生成機能

map32Rは、ロードモジュールファイルのシンボル情報から、アドレスとシンボル(変数やオブジェクト、および関数)名のマップをcsv形式で生成することができます。
作成されるcsvシンボルマップは表形式で、アドレスを第1カラムとして、第2カラム以降にシンボルを表示します。csv形式になっており、Microsoft Excel等の表計算ソフトに入力することができます。

5.1 csvシンボルマップファイルの詳細

5.1.1 csvシンボルマップファイルの生成

map32Rにオプション -c または -c16 を指定すると、アドレスとシンボル(変数やオブジェクト、および関数)名のマップをcsv形式で生成します。

-cオプション指定時は、1行にシンボルをひとつを表示する「1アドレス形式」を選択します。
-o オプションがある場合はそのファイルに、-o がない場合は標準出力に出力します。
-c指定時はリンクマップファイルは出力しません。

-c16オプション指定時は、1行に16バイト分の領域に属するシンボルを表示する「16アドレス形式」を選択します。
-o オプションがある場合はそのファイルに、-o がない場合は標準出力に出力します。
-c16指定時はリンクマップファイルは出力しません。

5.1.2 csvシンボルマップファイルの形式

[1アドレス形式の表示内容]

* 最初の行は Address,Symbol という見出しを表示します。
* シンボルが割付けられている下位(小さい)のアドレスから表示します。
* 1行につき1シンボルを表示します。
* アドレスを第1カラムに、シンボルを第2カラムにそれぞれ表示します。
* 表示したシンボルのサイズ分進めたアドレスを次の行のアドレスとします。
* シンボルが割付けられていないアドレスが連続する場合は、まとめて : を1行だけ出力します。

[16アドレス形式の表示内容]

* 最初にAddress,0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F という見出し行を表示します。
* 各行は、先頭アドレスを16の倍数に整合して表示します。
* シンボルが割付けられている下位(小さい)のアドレスから表示します。
* アドレスを第１カラムに、このアドレスから +0x0 ～ +0xF 進んだ場所にあるシンボルを第２カラム～第17カラムにそれぞれ表示します。
* シンボルの末尾には、サイズ表記(括弧つきの10進数)を付けます。
* 次の行は16バイトだけ進んだアドレスとなります。ただし、複数行にまたがるサイズのシンボルを表示する場合、: を1行だけ出力し、次の行はシンボルのサイズ分進んだアドレスになります。
* シンボルが割付けられていない行が連続する場合は、まとめて ： を1行だけ出力します。

[シンボル表記]

csv シンボルマップでは、次のような形式でシンボルを表示します。

表5.1 csvシンボルマップのシンボル表記

グローバル変数 static宣言されていない変数	symbol
ファイル内static変数 static関数	ファイル名@symbol
関数内static変数	ファイル名@関数名@symbol

5.2 csvシンボルマップファイルの出力例

5.2.1 “-c”の出力例

[コマンドライン]

```
map32R -o sample.csv -c sample.abs
```

[sample.csvの内容]

```
Address,Symbol
0x00008000,sample1.c@foo1@static_local
0x00008004,global
0x00008008,sample2.c@static_global
0x0000800C,
:
0x00FC0000,foo1
0x00FC0044,foo2
0x00FC0084,
```

5.2.2 “-c16”の 出力例

[コマンドライン]

```
map32R -o sample.csv -c16 sample.abs
```

[sample.csvの内容]

```
Address,0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
0x00008000,sample1.c@foo1@static_local(4),,,,global(4),,,,sample2.c@static_global(4),,,,,,,
:
0x00FC0000,foo1(68),,,,,,,,,,,,,,,,
:
0x00FC0040,,,,,foo2(64),,,,,,,,,,,
:
0x00FC0080,,,,,,,,,,,,,,,,
:
```

5.3 注意事項

● csvシンボルマップファイルは、デバッグ情報使って作成します。
デバッグ情報のないロードモジュールで、シンボルマップを作成しても中身のないファイルが作成されます。

第６章
マップジェネレータのメッセージ

6.1 マップジェネレータの実行結果を知るには

マップジェネレータの実行結果は、メッセージおよび終了コードから判断できます。

6.1.1 メッセージの出力形式

マップジェネレータは動作中にエラーを検出すると、実行状況を示すメッセージを標準エラー出力（通常は画面）に出力します。メッセージの出力形式を以下に示します。

■ 基本形式

ツール名：*入力情報*：*メッセージ種別*：*メッセージ*

※「入力情報:」は必要時のみ出力

■ パターン

```
map32R: ファイル名: メッセージ種別: メッセージ
map32R: <コマンド行>: メッセージ種別: メッセージ
map32R: メッセージ種別: メッセージ
```

※下線部分は入力情報（下線は出力されません）。

■ 出力例

```
map32R: error: cannot open file "abc.mo"
```

ツール名　メッセージ種別　メッセージ

6.1.2 メッセージ種別

メッセージは警告程度によって、表6.1のように分類されています。

表6.1　メッセージ種別

メッセージ種別（表示）	エラー発生時の動作
エラー（error）	エラーメッセージを出力し、処理を中止します。
重大エラー（fatal）	エラーメッセージを出力し、処理を中止します。

メッセージの詳細は、6.2「マップジェネレータのメッセージ一覧」を参照してください。

6.1.3 終了コード

マップジェネレータは実行後、以下の終了コード（実行結果を示す値）を返します（表6.2）。

表6.2 終了コード

終了コード	実行結果
0	正常終了した。
1	エラーが発生した。

6.2 マップジェネレータのメッセージ一覧

6.2.1 エラー

表6.3 マップジェネレータのエラーメッセージ

メッセージ	意味 → 対策
`cannot close file "`*filename*`"`	該当ファイルがクローズできません。 → ディスク容量を確認してください。
`cannot create file "`*filename*`"`	該当ファイルが生成できません。 → ディスク容量、およびディレクトリのパーミッションを確認してください。
`cannot open file "`*filename*`"`	該当ファイルが見つかりません。 → ファイル名を確認してください。
`illegal file format`	指定されたファイルのフォーマットが不正です。 → ファイル内容を確認してください。
`illegal option "`*option*`"`	不正なオプションを指定しています。 → 正しいオプションを指定してください。
`too many file names given`	複数の入力ファイルが指定されています。 → 入力ファイルは１つのみ指定してください。

6.2.2 重大エラー

表6.4 マップジェネレータの重大エラーメッセージ

メッセージ	意味 → 対策
`internal error`	内部エラーが発生しています。 → 当エラー発生時は至急ご連絡ください。
`out of memory`	メモリが不足しています。 → メモリ容量を確認してください。

第４部

ライブラリアン　lib32R

第１章
ライブラリアンlib32Rの概要

1.1 概要

lib32R はM3T-CC32R クロスツールキットに含まれているライブラリアンで、以下のような機能があります。

■ ライブラリを生成する

複数のオブジェクトモジュール、または、複数のリロケータブルロードモジュールから、ライブラリを生成します。1つのライブラリに登録可能なモジュール[注1]数は32767 個までです。

■ ライブラリを編集できる

すでに存在するライブラリに対して、モジュールの追加、削除、置換ができます。ライブラリ内の各モジュールを、ライブラリから抽出してライブラリに登録する前のファイル（オブジェクトモジュール）に戻せます。

■ ライブラリの情報（ライブラリアンリスト）を出力する

ライブラリの情報（モジュール情報および外部定義シンボル情報など）をライブラリアンリストとして出力します（-l（エル）オプション指定時）。また、指定したモジュールの情報を標準出力へ出力します（-t オプション指定時）。

1.2 機能

■ 1ライブラリにつき最大32767 個のモジュールが登録可能

■ モジュール管理の効率化に有効

複数のモジュールを1つのライブラリにまとめられるため、モジュール管理が容易に行えます。

■ リンク処理の効率化に有効

リンク時に指定する複数のオブジェクトモジュールをライブラリとして指定すると、ファイル操作を削減できます。これによってリンク処理の効率が向上します。

注1)「モジュール」とはライブラリを構成する要素を指します。1つのオブジェクトモジュールまたはロードモジュールをライブラリに登録すると、それがライブラリの1モジュールとなります。

■ コマンドファイルによる起動が可能

起動コマンドlib32Rのパラメータ（オプション指定、入力ファイル指定）を、コマンドファイルから与えることができます(2.1.3.2参照)。

第２章
ライブラリアンの起動

2.1 ライブラリアンを起動するには

2.1.1 ライブラリアンの起動手順

ライブラリアンを起動するためには、環境変数を設定（2.1.2参照）した後、規則に従ってコマンド「lib32R」を入力し、それを実行します（2.1.3参照）。

2.1.2 環境変数の設定

環境変数M32RBIN、M32RINC、M32RLIB、M32RTMPに正しいディレクトリを設定します（通常はインストール時に設定します）。設定方法は「M3T-CC32Rクロスツールキット V.XX Release Xリリースノート」を参照してください。環境変数の設定を省略すると、デフォルトのディレクトリを設定したことになります。

表2.1 環境変数のデフォルト

環境変数	デフォルトのディレクトリ
M32RBIN	/usr/local/M32R/bin
M32RINC	/usr/local/M32R/include
M32RLIB	/usr/local/M32R/lib
M32RTMP	/tmp

2.1.3　コマンド行の記述規則

ライブラリアン起動コマンド「lib32R」の、コマンド行の入力書式および規則を以下に示します。オプションおよび入力ファイルの指定方法には、コマンド行で直接入力する方法と、コマンドファイルを使って指定する方法があります（2.1.3.1、2.1.3.2参照）。起動オプションの詳細は2.2節、入出力ファイルについての詳細は2.1.4～2.1.6節を参照してください。

(1)コマンド行で各種起動指定を行う場合

```
lib32R { -c | -m | -q | -r | -d | -x [-s suffix] } [-t]
       [-l list_name] [-v] [-V]
       lib_name [object_filenames | module_names] <RET>
```

(2)コマンドファイルを使って各種起動指定を行う場合

```
lib32R command_filename <RET>
```

※[ ]で囲まれていない項目　　：必ず入力しなければならない項目
※[ ]で囲まれた項目　　：必要に応じて記述する項目
※{ }で囲まれた項目　　：{ }内のどれか1つを選択して入力（必須）
※　－の付いた記号　　：起動オプション（詳細は3.3節参照）
※<RET>　　：リターンキー入力

図2.1　lib32Rコマンドの入力書式

2.1.3.1　コマンド行で各種起動指定を行う場合

コマンド行に指定した情報によってライブラリアンを起動するには、以下の規則に従ってコマンドを入力および実行します。

■ コマンド行は、図2.1(1)の書式に従って記述します。各項目（コマンド名、オプション、入力ファイル名、コマンドファイル名）の間は、１文字以上の空白文字で区切ります。最後にリターンキーを入力すると、ライブラリアンが実行を開始します。

■ オプションとそのパラメータとの間には空白文字が必要です。矛盾するオプションの組み合わせがあれば、後から指定したオプションが有効となります。

■ *lib_name*には、新規に出力するライブラリ、または、編集対象となるライブラリの名前を１つ指定します。

■ *object_filenames*には、1つまたは複数の入力ファイル名（オブジェクトモジュールまたはリロケータブルロードモジュール）を指定します。ファイル名とファイル名の間は１文字以上の空白で区切ります。ファイル数は、コマンド行に記述可能な文字数を超えなければ256個まで指定できます。

■ *module_names* には、１つまたは複数の処理対象モジュール名を指定します。モジュール名とモジュール名の間は１文字以上の空白で区切ります。モジュール数は、コマンド行に記述可能な文字数を超えなければ最大 256 個まで指定できます。

■ オプションのすぐ後ろに指定した名前はライブラリ名（*lib_name*）とみなされます。それ以降に指定された名前はオブジェクトモジュール名(*object_filenames*)またはモジュール名（*module_names*）とみなされます。

注意

ライブラリアンの起動コマンド行では、指定ファイル名のうち、一番先頭に記述されたファイル名がライブラリ名とみなされます。起動オプションに続いて始めに指定するファイル名は必ずライブラリ名としてください。

2.1.3.2 コマンドファイルを使って各種起動指定を行う場合

ライブラリアンを起動するためのオプションや入力ファイル名の指定は、コマンドファイルを使って指定することができます。コマンドファイルとは、あらかじめ指定内容を記述したテキストファイルです。この方法は、入力するファイル名が多い場合や、ライブラリアンに指示する処理内容がほぼ決まっている場合に便利です。

コマンド行では、以下のようにパラメータとしてコマンドファイル名を指定します。

```
lib32R command_filename <RET>
```

コマンドファイルの記述方法を以下に示します。

■ 各パラメータ（オプション指定、入力ファイル名、処理対象モジュール名）の記述形式は、コマンド行で各種起動指定を行う場合の各パラメータの記述形式（図2.1(1)参照）に従う。

■ コマンドファイル内に限り、パラメータ間を改行（リターンキー入力）で区切ることが可能。

■ コマンドファイルの１行の文字数は 255 文字まで（改行文字除く）。

例えば、ライブラリfunction.libを作成し、そこへ3つのオブジェクトモジュールsin.mo、cos.mo、tan.moを登録し、ライブラリアンリストfunction.lisを出力したい場合、次のようなコマンドファイルを作成しておきます。

```
-l function.lis
-c function.lib
sin.mo cos.mo tan.mo
```

図2.2 コマンドファイルの内容（例）

2.1.4 入力ファイルの条件

ライブラリアンで処理できる入力ファイルの条件を以下に示します。これらの条件を満たさないファイルは入力しないでください。

表2.2 入力ファイルの条件

項目	条件
入力可能なファイル	オブジェクトモジュールファイル リロケータブルロードモジュールファイル ライブラリ
使用できる名前の数	セクション名　　：1ファイルにつき65535個まで シンボル名　　：1ファイルにつき65535個まで モジュール名　　：1ファイルにつき65535個まで ただし、開発環境のシステムのメモリ容量によっては制限を受けます。

2.1.5 生成ライブラリの条件

ライブラリアンが生成するライブラリには以下の条件があります。

- ライブラリに登録可能なモジュール数　　最大 32767 個まで
- ライブラリに登録可能なシンボル数　　最大 65535 個まで

2.1.6 出力ファイル名の命名

出力ファイルおよびその名前は、オプション指定によって以下のようになります。

表2.3 オプション指定と出力ファイル

オプション	出力ファイル	出力ファイル名
`-c,-m,-q,-r,-d`	ライブラリ	入力したライブラリと同じ名前
`-x`	オブジェクトモジュール	指定したモジュール名と同名、または、モジュール名に-sオプションで指定した拡張子が付いた名前
`-l` *list_name*	リストファイル	*list_name*

2.2 ライブラリアンの起動オプション

表2.4に、ライブラリアンの各起動オプションの機能について示します。

表2.4 ライブラリアン起動オプション(1/3)

オプション	機能
-c	**ライブラリの新規作成：** 指定したオブジェクトモジュールからライブラリを新規に作成し ます。
-d	**モジュールの削除：** 指定したモジュールをライブラリから削除します。
-l *list_name*	**ライブラリアンリストの作成：** *list_name*という名前のライブラリアンリストを作成します。ライブラリアンリストには モジュール名、モジュールの登録日時、モジュール内の外部定義シンボルを出力します。ライブラリアンリストはライブラリ中に含まれるモジュールの確認に有効です。
-m	**モジュールの追加／置換：** 指定したオブジェクトモジュールがライブラリ内に存在しなければ、そのオブジェクトモジュールをライブラリに追加します（-qオプションの処理に相当）。指定したオブジェクトモジュールがライブラリ内にあれば、そのオブジェクトモジュールと置換します（-rオプションの処理に相当）。 複数のオブジェクトモジュールを指定した場合は、指定順に追加または置換します。
-q	**モジュールの追加：** 指定したオブジェクトモジュールをライブラリに追加します。指定したモジュールがすでにライブラリ中に存在しているときは、ワーニングメッセージを出力し、追加は行われません。
-r	**モジュールの置換：** 指定したモジュールと同名のモジュールがライブラリ内に存在する場合、ライブラリ内のモジュールを指定したモジュールに置き換えます。指定したモジュールがライブラリ内になければワーニングメッセージを出力し、置換は行われません。
-s *suffix*	-xオプション指定時のみ有効。出力ファイル（内容は抽出したモジュール）名の拡張子を*suffix*にします。本オプション指定は-xオプション指定がなければ無効となります。

表2.4　ライブラリアン起動オプション（2/3）

オプション	機能
-t	**ライブラリ情報の出力：** モジュール名を指定した場合、ライブラリに登録されている指定モジュールの情報を以下の形式で標準出力へ出力します。 *モジュール名　　登録年月日　　時刻* モジュール名の指定がない場合、この情報を、ライブラリに登録されているすべてのモジュールについて出力します。
-v	-c、-d、-m、-q、-r、-t、-x オプションの処理内容詳細を標準出力に出力します。これらのオプション指定がなければ本オプション指定は無効となります。出力形式は以下のとおりです。 ■ -c、-d、-m、-q、-r、-xオプションの場合 実際に処理されたモジュールの名前、および、生成されたライブラリの名前を出力します。

処理	表示形式
モジュールを追加した	a：*モジュール名*
モジュールを置換した	r：*モジュール名*
モジュールを削除した	d：*モジュール名*
モジュールを抽出した	x：*モジュール名*
新規ライブラリを作成した	c：*ライブラリ名*

■ -tオプションの場合

モジュール指定の有無によって、ライブラリに関する以下の情報を出力します。

モジュール指定	該当ライブラリに関する出力
あり	各モジュール内の外部定義シンボル名
なし	・ライブラリ名 ・作成年月日および時刻 ・最新の更新年月日 ・登録されているモジュールの総数 ・登録されている外部定義シンボルの総数 ・各モジュール内の外部定義シンボル名

表2.4　ライブラリアン起動オプション（3/3）

オプション	機能
-V	起動メッセージを標準エラー出力に出力します。他のオプション指定はすべて無視され、実際の処理は行われません。
-x	**モジュールの抽出：** 指定したモジュールをライブラリから抽出します。抽出したモジュールはオブジェクトモジュールとして出力します。オブジェクトモジュール名は、モジュール名に-sオプションで指定する拡張子が付いた名前になります。-sオプション指定がなければ、モジュールと同じ名前になります。モジュールを指定しない場合はすべてのモジュールを抽出します。いずれの場合もライブラリの内容は変化しません。

2.3 ライブラリアンの起動例

ライブラリアンの起動例を以下に示します（% はプロンプト、<RET> はリターンキー入力を示します）。

■ 例１： 新規作成

```
% lib32R -c syslib.lib prog1.mo prog2.mo <RET>
```

新しいライブラリファイル syslib.lib を作成します。syslib.libにオブジェクトモジュールprog1.moとprog2.moを登録します。

■ 例２： 追加

```
% lib32R -l syslib.lis -q syslib.lib prog3.mo prog4.mo
<RET>
```

既存のライブラリファイルsyslib.libに、モジュールprog3.moとprog4.moを追加登録します。モジュール追加後のsyslib.libのライブラリアンリストsyslib.lisを作成します。

■ 例３： 抽出

```
% lib32R -x -s .mo syslib.lib prog1 prog3 <RET>
```

既存のライブラリsyslib.libから登録されているモジュールprog1、prog3を抽出し、登録前のモジュールファイルに再生します。-s オプションで拡張子の指定が .mo に指定されているので、出力ファイル名はそれぞれprog1.mo、prog3.mo になります。ライブラリファイルの内容は変化しません。

■ 例４： 追加、置換

```
% lib32R -c syslib.lib prog1.mo prog2.mo <RET> … ①
% lib32R -m syslib.lib prog3.mo prog1.mo <RET> … ②
```

新しいライブラリファイルsyslib.libを作成し、prog1.moとprog2.moを登録します（①）。次に、syslib.libにprog3.moを追加、prog1.moを置換します（②）。

■ 例５：

```
% lib32R -m -v syslib.lib prog1.mo prog2.mo <RET> …①
a: prog1
a: prog2
c: syslib.lib
% lib32R -t syslib.lib  …②
prog1              22-Jun-1995 14:59:23
prog2              22-Jun-1995 14:59:23
```

新しいライブラリファイルsyslib.libを作成し、prog1.moとprog2.moを登録します。このとき、-v オプションが指定されているので、詳細情報を表示します（①）。続いて、ライブラリの情報を表示します（②）。

第３章
ライブラリアンの出力

ライブラリアンが出力するライブラリ、ライブラリアンリスト、ライブラリ情報の内容を以下に示します。

3.1 ライブラリ

ライブラリとは、複数のオブジェクトモジュール、または、リロケータブルロードモジュールをまとめ、外部定義シンボルのインデックス部を付けたファイルです。

3.2 ライブラリアンリスト

ライブラリアンの起動オプション-l *list_name*を指定した場合、ライブラリの内容表示（ライブラリアンリスト）をファイル *list_name* に出力します。このライブラリアンリストの形式を以下に示します。

```
M32R FAMILY Librarian V.1.02.00 * LIBRARIAN LIST *

Library file name:  sample.lib
Creation date:      18-May-2000  9:45:38
Revision date:      18-May-2000  9:47:16
Number of modules:  2
Number of symbols:  8

Module name:                              Entry date:
gettoken                                  17-May-2000  9:45:38
     _gettoken

getvalue                            (a)   17-May-2000  9:47:16
     _chgbin
     _chgdigit
     _chghex
     _chgoct
     _getvalue
     _one
     _two
```

図3.1　ライブラリアンリスト出力例

次ページ →

図3.1の記述内容について以下に説明します（表3.1）。

表3.1　ライブラリアンリストの内容

項目	内容
Library file name:	ライブラリ名を示します。
Creation date:	ライブラリの作成年月日と時刻を示します。
Revision date:	ライブラリの最新の更新年月日と時刻を示します。
Number of modules:	ライブラリに登録されているモジュールの総数を10進数で示します。
Number of symbols:	ライブラリに登録されている外部定義シンボル総数を10進数で示します。
Module name:	ライブラリに登録されているモジュールの名前をアルファベット順に示します。また、( )内はモジュールに対する編集操作状態を示します。これは、-l オプションを -c、-r、-q、-m オプションのいずれかと共に指定した場合にのみ出力されます。編集操作状態を以下に示します。 ・　空白　　既存ライブラリに登録されているモジュール ・ (a)　　追加モジュール ・ (r)　　置換モジュール
Entry date:	モジュールがライブラリに登録された年月日、時刻を示します。
その他	各モジュール名の下に、そのモジュールで定義されている外部定義シンボルを、アルファベット順で示します。

3.3 ライブラリ情報

ライブラリアンの起動オプション-t オプションを指定した場合、ライブラリ情報を出力します。その出力形式を以下に示します（モジュール名指定および -v オプション指定の有無により４つのケースがあります）。

■ ケース１：モジュール名指定なし、-v オプション指定なしの場合

ライブラリに登録されているすべてのモジュールについて名前、登録年月日、および時刻を以下の形式で出力します。

モジュール名　　登録年月日　時刻

例：
```
gettoken        17-May-20009:45:38
prog1           22-Jun-200013:17:50
```

■ ケース2：モジュール名指定なし、-v オプション指定ありの場合

ライブラリアンリストと同じ内容を出力します。

次ページ →

■ ケース3：モジュール名指定あり、-v オプション指定なしの場合

ライブラリに登録されている指定モジュールの名前、登録年月日、および時刻を以下の形式で出力します。

```
    モジュール名    登録年月日   時刻
```

例：　gettoken　　17-May-2000　9:45:38

■ ケース4：モジュール指定あり、-v オプション指定ありの場合

ライブラリに登録されている指定モジュールの名前、登録年月日、時刻、および、そのモジュールで定義されているシンボルをアルファベット順で出力します。出力形式は以下のとおりです。

```
モジュール名　　登録年月日　時刻
シンボル名
シンボル名
```

例：　module1　　22-Jun-2000　　13:17:50
　　_symbol1
　　_symbol2
　　_symbol3

第４章
ライブラリアンのメッセージ

4.1 ライブラリアンの実行結果を知るには

ライブラリアンの実行結果は、メッセージおよび終了コードから判断できます。

4.1.1 メッセージの出力形式

ライブラリアンは動作中にエラーを検出すると、実行状況を示すメッセージを標準エラー出力（通常は画面）に出力します。メッセージの出力形式を以下に示します。

■ 基本形式

ツール名：*入力情報*：*メッセージ種別*：*メッセージ*

※「入力情報:」は必要時のみ出力

■ パターン

```
lib32R: ファイル名: メッセージ種別: メッセージ
lib32R: <コマンド行>: メッセージ種別: メッセージ
lib32R: メッセージ種別: メッセージ
```

※下線部分は入力情報（下線は出力されません）。

■ 出力例

```
lib32R: "sample.mo": error: invalid file format
```

ツール名　入力情報（ファイル名）　メッセージ種別　メッセージ

4.1.2 メッセージ種別

メッセージは警告程度によって、表4.1のように分類されています。

表4.1 メッセージ種別

メッセージ種別（表示）	エラー発生時の動作
ワーニング（warning）	ワーニングメッセージを出力し、処理を続行します。
エラー（error）	エラーメッセージを出力し、処理を中止します。
重大エラー（fatal）	エラーメッセージを出力し、処理を中止します。

メッセージの詳細は、4.2「ライブラリアンのメッセージ一覧」を参照してください。

4.1.3 終了コード

ライブラリアンは実行後、以下の終了コード（実行結果を示す値）を返します（表4.2）。

表4.2 終了コード

終了コード	実行結果
0	正常終了した。または、ワーニングが発生した。
1	エラーが発生した。

4.2 ライブラリアンのメッセージ一覧

4.2.1 ワーニング

表4.3 ライブラリアンのワーニングメッセージ

メッセージ　　意味 → ライブラリアンの動作
`duplicate module "`*module_name*`"` モジュールは既に登録済みです。 → 該当モジュールに対する操作をスキップします。
`duplicate symbol "`*symbol*`"` シンボルは既に登録済みです。 → 該当モジュールに対する処理をスキップします。
`module not found "`*module_name*`"` 指定されたモジュールが見つかりません。 → 該当モジュール名に対する操作をスキップします。

4.2.2 エラー

表4.4 ライブラリアンのエラーメッセージ（1/3）

メッセージ　　意味 → 対策
`cannot close file "`*filename*`"` 該当ファイルがクローズできません。 → ディスク容量を確認してください。

次ページ →

表4.4 ライブラリアンのエラーメッセージ (2/3)

メッセージ	意味 → 対策
cannot create file "*module_name*"	該当ファイルが生成できません。 → ファイル名、およびディスク容量を確認してください。
cannot open file "*module_name*"	該当ファイルが見つかりません。 → ファイル名を確認してください。
illegal file format	ファイルのフォーマットが不正です。 → ファイル内容を確認してください。
illegal option "*option*"	不正なオプション名を指定しています。 → 正しいオプションを指定してください。
library file not specified	ライブラリファイルが指定されていません。 → ライブラリファイルを指定してください。
option *option* requires parameter	パラメータ指定が必要なオプションにパラメータが指定されていません。 → パラメータを指定してください。
only one of [-c,-r,-d,-q,-x,-m,-t] must be specified	オプション-c、-r、-d、-q、-x、-m、-tのうち、どれも指定されていない、または、複数同時に指定されています。 → オプション-c、-r、-d、-q、-x、-m、-tのうち、1つのみ指定してください。
too many modules given	パラメータに指定されたモジュール数が許容範囲を超えました。 → 複数回に分けて処理してください。
too many modules in library file	登録するモジュール数が、ライブラリの許容範囲を超えました。 → 複数のライブラリに分割してください。
too many symbols in library file	登録するモジュールに含まれているシンボル数が、ライブラリの許容範囲を超えました。 → 複数のライブラリに分割してください。

次ページ →

表4.4　ライブラリアンのエラーメッセージ（3/3）

メッセージ	意味　→　対策
`unsupported module type: version` *number*	サポートしていないタイプのモジュールが指定されています。 →　モジュールが正しく生成できているか確認してください。
`duplicate module "`*module_name*`"`	モジュールは既に登録済みです。該当モジュールに対する操作をスキップします。 →　重複するモジュール名のどちらかの名前を変更してください。
`duplicate symbol "`*symbol_name*`"`	シンボルは既に登録済みです。該当モジュールに対する処理をスキップします。 →　重複するシンボル名のどちらかの名前を変更してください。
`module not found "`*module_name*`"`	指定されたモジュールが見つかりません。該当モジュール名に対する操作をスキップします。 →　指定したモジュールが存在するか、確認してください。

4.2.3　重大エラー

表4.5　ライブラリアンの重大エラーメッセージ

メッセージ	意味　→　対策
`internal error`	内部エラーが発生しています。 →　当エラー発生時は至急ご連絡ください。
`out of memory`	ライブラリアンが動作するためのメモリが不足しています。 →　メモリを拡張する、または、ライブラリの構成を小さくしてください。

第５部

ロードモジュールコンバータ
lmc32R

第１章
ロードモジュールコンバータ lmc32Rの概要

1.1 概要

lmc32R はM3T-CC32R クロスツールキットに含まれているロードモジュールコンバータです。リンカが生成したアブソリュートロードモジュールを、汎用ROM ライタで読み込み可能なモトローラＳフォーマット（以下、Ｓフォーマット）のロードモジュールに変換します。

1.2 機能

■ オブジェクト分割機能

起動時にコマンド行で指定した規則に従って、ロードモジュールを複数のオブジェクトデータに分割し、分割数分のＳフォーマットファイルを作成します。通常ターゲットシステムでは、データバス幅に応じて複数個のROM を用いるため、オブジェクトデータを分割して各ROM にロードします。この機能を使えば、各ROM にロードするファイルを個別に生成できます。

■ アドレス範囲の指定機能

指定したアドレス範囲のオブジェクトデータだけを、Ｓフォーマットに変換し出力できます。

■ ロードアドレス変更指定機能

オブジェクトデータのロードアドレス値をオフセット値指定によって変更できます。例えば、8000_{16} 番地から始まるオブジェクトデータ列を 0_{16} 番地から始まるオブジェクトデータ列に変更する、という処理が可能です。

第２章
ロードモジュールコンバータの起動

2.1 ロードモジュールコンバータを起動するには

2.1.1 ロードモジュールコンバータの起動手順

ロードモジュールコンバータを起動するためには、環境変数を設定（2.1.2参照）した後、規則に従ってコマンド「lmc32R」を入力し、それを実行します（2.1.3参照）。

2.1.2 環境変数の設定

環境変数M32RBIN、M32RINC、M32RLIB、M32RTMPに正しいディレクトリを設定します（通常はインストール時に設定します）。設定方法は「M3T-CC32Rクロスツールキット V.XX Release Xリリースノート」を参照してください。環境変数の設定を省略すると、デフォルトのディレクトリを設定したことになります。

表2.1 環境変数のデフォルト

環境変数	デフォルトのディレクトリ
M32RBIN	/usr/local/M32R/bin
M32RINC	/usr/local/M32R/include
M32RLIB	/usr/local/M32R/lib
M32RTMP	/tmp

2.1.3 コマンド行の記述規則

ロードモジュールコンバータの起動コマンド「lmc32R」の、コマンド行の入力書式および規則を以下に示します。起動オプションの詳細は2.2節、入出力ファイルについての詳細は2.1.4～2.1.5節を参照してください。

```
lmc32R [-o output_filename ] [-d{1|2|4|8}] [-W{1|2}]
       [-r baddr[,eaddr]] [-c[{+|-}]naddr] [-w] [-V]
       filename <RET>
```

※[ ]で囲まれていない項目　　：必ず入力しなければならない項目
※[ ]で囲まれた項目　　：必要に応じて記述する項目
※　－の付いた記号　　：起動オプション（詳細は4.3節参照）
※<RET>　　：リターンキー入力

図 2.1　lmc32R コマンドの入力書式

- コマンド行は、上記（図2.1）の書式に従って記述します。各項目（コマンド名、オプション、入力ファイル名）の間は、1文字以上の空白文字で区切ります。最後にリターンキーを入力すると、ロードモジュールコンバータが実行を開始します。
- オプションとそのパラメータとの間には空白文字が必要です。矛盾するオプションの組み合わせがあれば、後から指定したオプションが有効となります。
- アドレスや数値の指定には16進数による指定のみ有効です。
- *filename*には入力ファイル名を1つだけ指定します。入力ファイル名は省略できません。

2.1.4 入力ファイルの条件

ロードモジュールコンバータで処理できる入力ファイルの条件を以下に示します。これらの条件を満たさないファイルは入力しないでください。

表2.2　入力ファイルの条件

項目	条件
入力可能なファイル	アブソリュートロードモジュール

2.1.5　出力ファイル名の命名

出力ファイル名は、コマンド行で指定した名前によって決まります。その規則については2.2「ロードモジュールコンバータの起動オプション」の -o オプションの説明を参照してください。コマンド行での指定がなかった場合、デフォルトで以下のように命名されます。

表2.3　出力ファイル名（デフォルト）

ファイル名	内容
filename.m*AB*	分割処理をした場合の各Sフォーマットロードモジュール。 *filename*　：入力ファイル名 *A*　：分割数（1桁の数字） *B*　：ファイルが何番目であるかを示す数 （0を含む1桁の数字。0、1、～）
filename.mot	分割処理をしない場合のSフォーマットロードモジュール。 *filename*　：入力ファイル名

2.2 ロードモジュールコンバータの起動オプション

表2.4に、ロードモジュールコンバータの各起動オプションの機能について示します。

表2.4 ロードモジュールコンバータ起動オプション(1/2)

オプション	機能
-c [{+|-}]*naddr*	ロードアドレスを*naddr*に変更します。先頭に + または - を付けることによってオフセット値とすることができます。*naddr*は16進数で指定してください。
-d {1|2|4|8}	オブジェクトデータの分割数を指定します。デフォルトは1です。
-o *output_filename*	*output_filename*で指定するファイル名をもとに出力ファイル名を付けます。名前の付け方は、ロードモジュールを複数のオブジェクトデータに分割する（-dオプション指定）かしないかによって以下のようになります。なお、PC版の出力ファイル名の名前の付け方は、表2.5 を参照してください。 ■ 分割する場合（-dオプションで1以外を指定） 出力ファイル名の拡張子を.m*AB* とします。*A*は分割数（1桁の数字）、*B*はファイルが何番目であるか（1桁の数字。先頭は0）をそれぞれ示す数字です。 各出力ファイル名は、*output_filename*に拡張子 .m*AB* が付いたものとなります。例えば、-o a.mout と指定すると、各出力ファイル名は a.mout.m*AB* となります。 本オプションを省略すると出力ファイル名は、入力ファイル名*filename*に .m*AB*を付加したものになります。 ■ 分割しない場合（-dオプションで1を指定、または-dオプションなし） 出力ファイル名の拡張子を.motとします。 出力ファイル名は、*output_filename*に拡張子 .mot が付いたものとなります。例えば、-o a.mout とすると出力ファイル名はa.mout.mot となります。 本オプションを省略すると、出力ファイル名は、入力ファイル名*filename*に .mot を付加したものになります。

次ページ →

表2.4　ロードモジュールコンバータ起動オプション(2/2)

オプション	機能
-r *baddr*[,*eaddr*]	変換を行うデータの範囲を指定します。変換開始アドレス*baddr*から変換終了アドレス*eaddr*までのデータの範囲を出力します。*eaddr*省略時は、変換開始アドレス*baddr*以降のデータをすべて出力します。*baddr*、*eaddr*とも16進数で指定してください。
-V	起動メッセージを標準エラー出力に出力します。他のオプション指定はすべて無視され、実際の処理は行われません。
-w	ワーニングメッセージの表示を抑止します。
-W {1|2}	オブジェクトデータを分割する場合のデータ幅（バイト数）を指定し ます。デフォルトは1です。

表 2.5　lmc32R の出力ファイル名命名規則（PC 版の場合）

出力オブジェクトの分割（-d オプション） 出力ファイル名の指定（-o オプション）	分割しない （-d オプションで 1 を指定、または、-d オプション指定なし）	分割する （-d オプションで 1 以外を指定） 指定例：-d 2
指定あり	出力ファイル名は、 指定出力ファイル名と同じ （拡張子を新たに付加しない）。	出力ファイル名は、 指定出力ファイル名（拡張子付き）の拡張子を .m*AB*[注]に置換したもの。 または、 指定出力ファイル名（拡張子なし）に拡張子 .m*AB* を付加したもの。
出力ファイル名指定例： 例 1）-o SMP.MOT 例 2）-o SMP	命名例： 例 1 の場合）SMP.MOT 例 2 の場合）SMP	命名例： 例 1 の場合）SMP.m20， SMP.m21 例 2 の場合）SMP.m20， SMP.m21
指定なし	出力ファイル名は、 指定入力ファイル名（拡張子付き）の拡張子を .mot に置換したもの。 または、 指定入力ファイル名（拡張子なし）に拡張子 .mot を付加したもの。	出力ファイル名は、 指定入力ファイル名（拡張子付き）の拡張子を .m*AB*[注)] に置換したもの。 または、 指定入力ファイル名（拡張子なし）に拡張子 .m*AB* を付加したもの。
入力ファイル名指定例： 例 3）am.out 例 4）am	命名例： 例 3 の場合）am.mot 例 4 の場合）am.mot	命名例： 例 3 の場合）am.m20，am.m21 例 4 の場合）am.m20，am.m21

注）　拡張子「.m*AB*」
実際には、*A*は分割数（-dオプションで指定されたもの）を示す数字、*B*は分割されたオブジェクトのうちの何番目のオブジェクトであるかを示す0から始まる番号となります。例えば「.m20」ならば、2分割されたオブジェクトのうちの1つ目であることを表しています。

第３章
ロードモジュールコンバータの使用方法

ここでは、ロードモジュールコンバータの起動例を使用方法とともに説明します（起動例のコマンド行で、%はプロンプト、<RET>はリターンキー入力を示します）。

3.1 出力ファイルを分割するには（オブジェクト分割機能）

「オブジェクト分割機能」とは、Sフォーマットに変換したロードモジュールを複数ファイルに分割して出力する機能です。この機能は、ロードモジュールを複数のROMにロードする場合に有効です。以下に、出力ファイルを分割するための、コマンド行における指定方法を示します。

① 分割数を指定します。

-dオプションで、変換後のデータをいくつのファイルに振り分けるかを指定します。分割ファイル数は、1、2、4、8の中から選択します。-dオプションを省略した場合、または、分割数=1の場合、ファイルを分割しないという指定になります。

② データの振り分け幅（バイト）を指定します。

-Wオプションで、各出力ファイルへのオブジェクトデータの振り分け単位を指定します。データの振り分け単位は、1バイト幅または2バイト幅のどちらかを選択します。-Wオプションを省略した場合、1バイト幅指定となります。

③ 出力ファイル名を指定します。

-oオプションで、各出力ファイル名を指定します。実際には、ここで指定するファイル名にSフォーマットの分割ファイルであることを示す拡張子.m*AB*（*A*には分割数、*B*には分割番号0～が入ります）が、ロードモジュールコンバータによって自動的に、付加されます。例えば、

```
% lmc32R -d4 -o file a.mout  <RET>
```

と実行した場合、出力ファイル名は次のようになります。

```
f i l e . m 4 0
f i l e . m 4 1
f i l e . m 4 2
f i l e . m 4 3
```

分割番号（4分割の場合　0、1、2、3）

分割数（4分割）

次ページ →

このため、出力ファイル名には拡張子を取り除いた部分のみ指定してください。もし、拡張子付きの名前を指定すると、それにさらに .m*AB*が付きます。

分割例を以下に示します。

例： `% lmc32R -d4 -W1 -o file a.mout <RET>`

このコマンドを実行した場合、出力ファイルは以下のように分割されます（図4.2)。なお、分割後の絶対アドレスは、変換対象ロードモジュール（この場合a.mout）の絶対アドレスを分割数で割ったアドレス（この場合H'400～）となります。

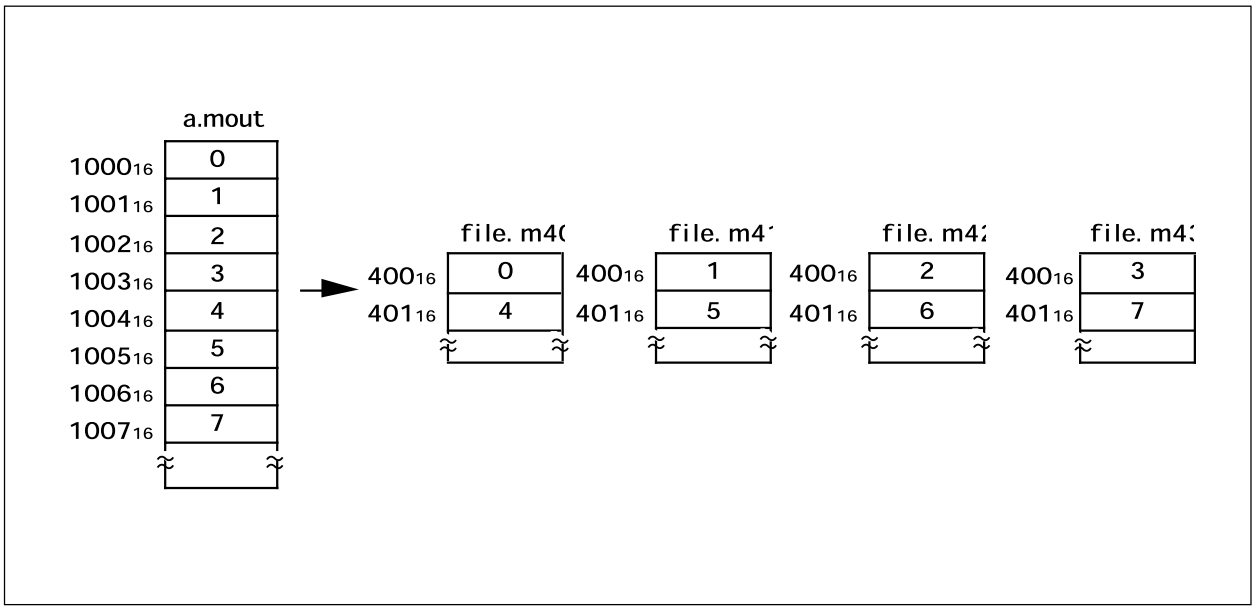


図3.1　出力オブジェクトの分割例

3.2 ロードモジュールの一部分のみSフォーマットに変換するには（アドレス範囲の指定機能）

「アドレス範囲の指定機能」とは、変換対象のロードモジュールのうち、指定する一部分のオブジェクトデータだけを変換する機能です。変換指定は -r オプションで行い、変換対象範囲を開始アドレスと終了アドレスで指定します。

例： lmc32R -o test -r 2000,3000 test.abs <RET>

上記コマンドを実行した場合、以下のように入力ファイルの一部のみを変換し結果が出力されます（図3.2）。

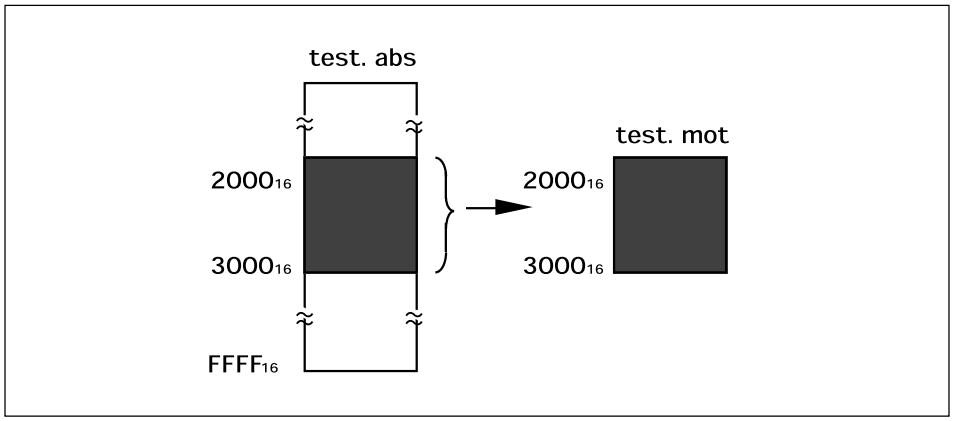


図3.2 一部分のみ変換・出力する例

3.3 ロードモジュールのロードアドレスを変更するには（ロードアドレス変更指定機能）

変換対象のロードモジュールには絶対アドレスが設定されています。そのアドレス設定がROMなどへロードするための実際のアドレスと異なる場合、変換時にオフセット値を指定して、出力ロードモジュールのアドレスを調整できます（ロードアドレス変更指定機能）。オフセット指定は-cオプションで行い、 -c {+|-}*naddr*と指定します。

例： % lmc32R -c -8000 test.abs <RET>

上記コマンドを実行した場合、以下ようにアドレスを調整できます（図3.3）。

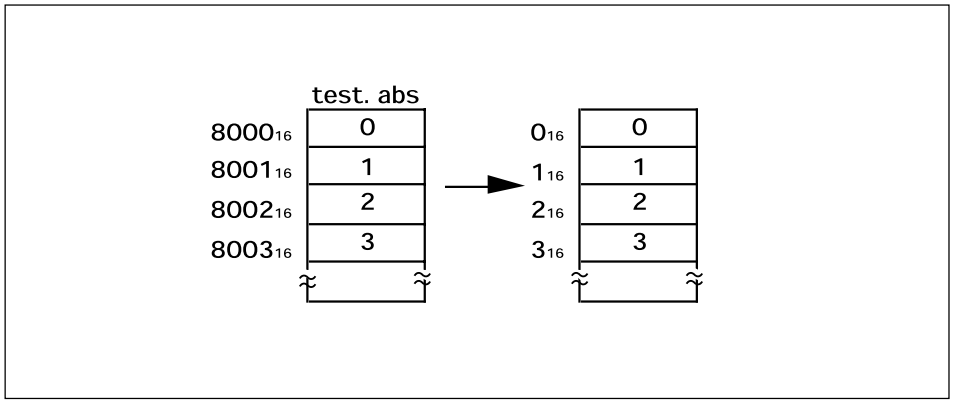


図3.3 アドレスのオフセット例

第４章
Sフォーマット

4.1 Sフォーマットの構成

S フォーマットオブジェクトは次の 3 レコードから構成されています。

■ ヘッダレコード
■ データレコード
■ エンドレコード

データレコードは、ロードアドレスの値により、それぞれ表4.1 に示す3つのレコードタイプがあります。

表4.1 データレコードタイプ

アドレス範囲	データレコードのタイプ
0_{16}〜 $FFFF_{16}$	S1
0_{16}〜 $FFFFFF_{16}$	S2
0_{16}〜 $FFFFFFFF_{16}$	S3

エンドレコードは、ロードモジュール中に含まれるデータレコードのタイプにより、次のように決定されます。

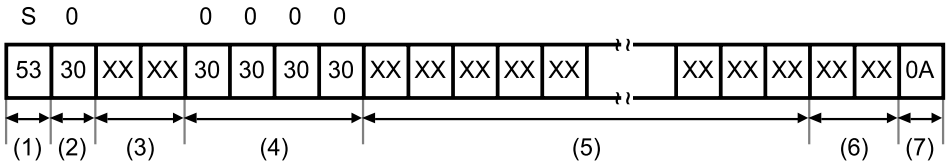
表4.2 エンドレコードタイプ

データレコードの構成	エンドレコードのタイプ
S1だけで構成されている	S9
S2が含まれている	S8
S3が含まれている	S7

以下より、各レコードの構成について示します。

4.2 レコードの内容

4.2.1 ヘッダレコード

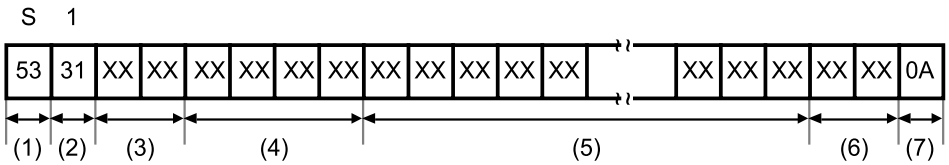


(1)レコードマーク
(2)レコードタイプ
(3)レコード長（(4)，(5)，(6) のバイト数）
(4)未使用
(5)コメント
(6)チェックサム（レコード長からチェックサムの前までのデータ値をバイト単位に加算した結果の、1 の補数）
(7)改行コード

4.2.2 データレコード

ロードアドレスによって、以下のような構成になっています。

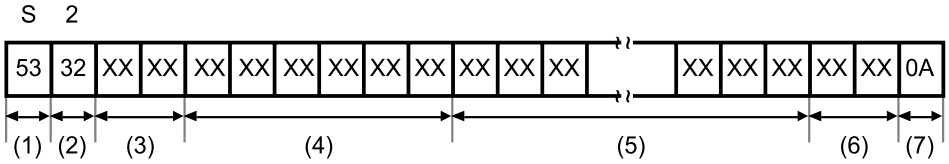
■ ロードアドレスが 0_{16} ～ $FFFF_{16}$ の場合



(1)レコードマーク
(2)レコードタイプ
(3)レコード長（(4)，(5)，(6) のバイト数）
(4)ロードアドレス（2 バイト分）
(5)オブジェクトデータ（1 バイトのオブジェクトデータを 16 進数 2 文字で表現したもの。最大 16 バイトまでのデータが格納される。）
(6)チェックサム（レコード長からチェックサムの前までのデータ値をバイト単位に加算した結果の、1 の補数）
(7)改行コード

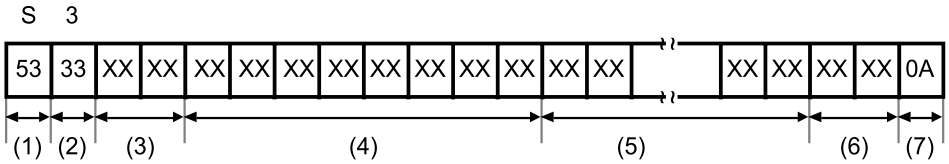
次ページ →

■ ロードアドレスが 0_{16} ～ $FFFFFF_{16}$ の場合



(1)レコードマーク
(2)レコードタイプ
(3)レコード長（(4)，(5)，(6) のバイト数）
(4)ロードアドレス（3 バイト分）
(5)オブジェクトデータ（1 バイトのオブジェクトデータを 16 進数 2 文字で表現したもの。最大 16 バイトまでのデータが格納される。）
(6)チェックサム（レコード長からチェックサムの前までのデータ値をバイト単位に加算した結果の、1 の補数）
(7)改行コード

■ ロードアドレスが 0_{16} ～ $FFFFFFFF_{16}$ の場合

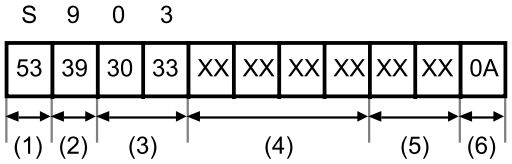


(1)レコードマーク
(2)レコードタイプ
(3)レコード長（(4)，(5)，(6) のバイト数）
(4)ロードアドレス（4 バイト分）
(5)オブジェクトデータ（1 バイトのオブジェクトデータを 16 進数 2 文字で表現したもの。最大 16 バイトまでのデータが格納される。）
(6)チェックサム（レコード長からチェックサムの前までのデータ値をバイト単位に加算した結果の、1 の補数）
(7)改行コード

4.2.3 エンドレコード

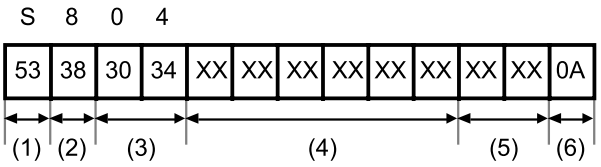
データレコードにS1、S2、S3がどのように含まれているかによって、以下のような構成になっています。

■ データレコードがS1だけで構成される場合



(1)レコードマーク
(2)レコードタイプ
(3)レコード長（(4)，(5) のバイト数）
(4)実行開始アドレス（2バイト分）
(5)チェックサム（レコード長からチェックサムの前までのデータ値をバイト単位に加算した結果の、1の補数）
(6)改行コード

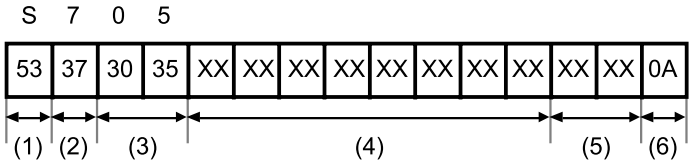
■ データレコードにS2が含まれる場合



(1)レコードマーク
(2)レコードタイプ
(3)レコード長（(4)，(5) のバイト数）
(4)実行開始アドレス（3バイト分）
(5)チェックサム（レコード長からチェックサムの前までのデータ値をバイト単位に加算した結果の、1の補数）
(6)改行コード

次ページ →

■ データレコードにS3が含まれる場合



(1)レコードマーク
(2)レコードタイプ
(3)レコード長（(4)，(5) のバイト数）
(4)実行開始アドレス（4バイト分）
(5)チェックサム（レコード長からチェックサムの前までのデータ値をバイト単位に加算した結果の、1の補数）
(6)改行コード

第５章
ロードモジュールコンバータのメッセージ

5.1 ロードモジュールコンバータの実行結果を知るには

ロードモジュールコンバータの実行結果は、メッセージおよび終了コードから判断できます。

5.1.1 メッセージの出力形式

ロードモジュールコンバータは動作中にエラーを検出すると、実行状況を示すメッセージを標準エラー出力（通常は画面）に出力します。メッセージの出力形式を以下に示します。

■ 基本形式

ツール名：*入力情報*：*メッセージ種別*：*メッセージ*

※「入力情報：」は必要時のみ出力

■ パターン

lmc32R：*ファイル名*：*メッセージ種別*：*メッセージ*

lmc32R：*<コマンド行>*：*メッセージ種別*：*メッセージ*

lmc32R：*メッセージ種別*：*メッセージ*

※下線部分は入力情報（下線は出力されません）。

■ 出力例

```
lmc32R: inputfile: error: input file is not absolute load module
```

ツール名　入力情報（ファイル名）　メッセージ種別　メッセージ

5.1.2 メッセージ種別

メッセージは警告程度によって、表5.1のように分類されています。

表5.1 メッセージ種別

メッセージ種別（表示）	エラー発生時の動作
ワーニング（warning）	ワーニングメッセージを出力し、処理を続行します。
エラー（error）	エラーメッセージを出力し、処理を中止します。
重大エラー（fatal）	エラーメッセージを出力し、処理を中止します。

メッセージの詳細は、5.2「ロードモジュールコンバータのメッセージ一覧」を参照してください。

5.1.3 終了コード

ロードモジュールコンバータは実行後、以下の終了コード（実行結果を示す値）を返します（表5.2）。

表5.2 終了コード

終了コード	実行結果
0	正常終了した。または、ワーニングが発生した。
1	エラーが発生した。

5.2 ロードモジュールコンバータのメッセージ一覧

5.2.1 ワーニング

表5.3 ロードモジュールコンバータのワーニングメッセージ

メッセージ　　意味 → ロードモジュールコンバータの動作
address overflow 出力ファイルに設定するロードアドレスが許容範囲以上になりました。 → 許容範囲内に丸め込みます。
address underflow 出力ファイルに設定するロードアドレスが許容範囲以下になりました。 → 許容範囲内に丸め込みます。
invalid data width for dividing, option -w ignored 分割数が１のときに、不正な分割幅が指定されています。 → -W オプションを無視します。
option *option* specified more than once, last setting taken 同じオプション（パラメータを伴う）を重複して指定しています。 → 後のオプションのパラメータが有効となります。

5.2.2 エラー

表5.4 ロードモジュールコンバータのエラーメッセージ (1/3)

メッセージ　　意味 → 対策
cannot close file "*filename*" 該当ファイルがクローズできません。 → ディスク容量を確認してください。
cannot create file "*filename*" 該当ファイルが生成できません。 → ファイル名、およびディスク容量を確認してください。
cannot open file "*filename*" 該当ファイルが見つかりません。 → ファイル名を確認してください。

表5.4　ロードモジュールコンバータのエラーメッセージ (2/3)

メッセージ　　意味　→　対策

input file is not absolute load module

入力ファイルがリンカの出力したアブソリュート形式のロードモジュールではありません。

→　リンカの出力したアブソリュート形式のロードモジュールを指定してください。

input file not specified

入力ファイルが指定されていません。

→　入力ファイルを指定してください。

illegal address　"*address*"

オプション指定アドレスに誤りがあります。

→　アドレスを指定しなおしてください。

illegal file format

ファイルフォーマットに誤りがあります。

→　ロードモジュールが正しく生成できているか確認してください。

illegal option　"*option*"

不正なオプションを指定しています。

→　正しいオプションを指定してください。

illegal option parameter "*parameter*"

既定値以外の値が、オプションのパラメータに指定されています。

→　既定値を指定してください。

no data in input file

入力ファイルにオブジェクトデータが含まれていません。

→　オブジェクトデータのあるファイルを指定してください。

no data in specified address range

変換を行うアドレスの範囲にオブジェクトデータが含まれていません。

→　オブジェクトデータのあるアドレス範囲を指定するか、アドレス範囲を確認してください。

option parameter not specified : *option*

パラメータを省略できないオプションのパラメータが省略されています。

→　パラメータを指定してください。

same file name as input

分割によって出力される出力ファイル名と、入力ファイル名が同じです。

→　出力ファイル名を入力ファイル名と異なるものにしてください。

too many file names given

複数の入力ファイルが指定されています。

→　指定する入力ファイルを一つにしてください。

5.2.3 重大エラー

表5.5 ロードモジュールコンバータの重大エラーメッセージ

メッセージ	意味 → 対策
`internal error`	内部エラーが発生しています。 → 当エラー発生時は至急ご連絡ください。
`out of memory`	メモリが不足しています。 → メモリ容量を確認してください。

MEMO

M32Rファミリ用
C/C++**コンパイラパッケージ** V.5.00
アセンブラユーザーズマニュアル

発行年月日 2005**年**07**月**15**日** Rev.1.00

発行 **株式会社 ルネサス テクノロジ 営業企画統括部**
〒100-0004 **東京都千代田区大手町**2-6-2

編集 **株式会社 ルネサス ソリューションズ ツール開発部**

M32R ファミリ用
C/C++ コンパイラパッケージ V.5.00
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

RJJ10J0905-0100