お客様各位

カタログ等資料中の旧社名の扱いについて

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

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

2010年4月1日
ルネサスエレクトロニクス株式会社

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

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

ご注意書き

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

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

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

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

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

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

お客様各位

資料中の「三菱電機」、「三菱XX」等名称の株式会社ルネサス テクノロジへの変更について

2003年4月1日を以って株式会社日立製作所及び三菱電機株式会社のマイコン、ロジック、アナログ、ディスクリート半導体、及びDRAMを除くメモリ(フラッシュメモリ・SRAM等)を含む半導体事業は株式会社ルネサス テクノロジに承継されました。
従いまして、本資料中には「三菱電機」、「三菱電機株式会社」、「三菱半導体」、「三菱XX」といった表記が残っておりますが、これらの表記は全て「株式会社ルネサス　テクノロジ」に変更されておりますのでご理解の程お願い致します。尚、会社商標・ロゴ・コーポレートステートメント以外の内容については一切変更しておりませんので資料としての内容更新ではありません。

注:「高周波・光素子事業、パワーデバイス事業については三菱電機にて引き続き事業運営を行います。」

2003年4月1日
株式会社ルネサス テクノロジ
カスタマサポート部

User's Manual

RENESAS

M3T-SRA74 V.4.10

ユーザーズマニュアル

740 ファミリ用リロケータブルアセンブラ

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2003.08

- 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 、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、X/Open Company Limited が独占的にライセンスしている米国ならびに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 マイコンツール部
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

はじめに

SRA74 は 740 ファミリ 専用の構造化記述リロケータブルマクロアセンブラです。SRA74 は 740 ファミリ 構造化記述言語、アセンブリ言語、又はそれらの混在したソースプログラムから、740 ファミリ の機械語データファイル、デバッグ情報ファイル等を生成します。本書は、SRA74 に含まれている以下の 5 つのソフトウェアの機能と操作方法について記述しています。

1. 構造化記述リロケータブルマクロアセンブラ SRA74
2. リンケージエディタ LINK74
3. ライブラリアン LIB74
4. クロスリファレンサ CRF74
5. M37280 用コンバータ CV74

本書の構成

本書は、以下に示す 5 部から構成されています。マニュアルの各部は可能な限り同じ順序で説明を行っています。例えば、環境変数については、操作説明の章の最後の節を見ることにより、各ソフトウェアの情報がわかるようになっています。

- 第 1 部 : SRA74 操作マニュアル
  構造化記述リロケータブルマクロアセンブラ SRA74 の操作方法とソースプログラム記述方法について説明します。
- 第 2 部 : LINK74 操作マニュアル
  リンカ LINK74 の操作方法とセクションの機能について説明します。
- 第 3 部 : LIB74 操作マニュアル
  ライブラリアン LIB74 の操作方法について説明します。
- 第 4 部 : CRF74 操作マニュアル
  クロスリファレンサ CRF74 の操作方法について説明します。
- 第 5 部 : CV74 操作マニュアル
  M37280 用コンバータ CV74 の操作方法について説明します。

第 1 部

740 ファミリ 用
構造化リロケータブルマクロアセンブラ

SRA74 操作マニュアル

目 次

図 目 次

表 目 次

第 1 章

マニュアルの構成

SRA74 操作マニュアルは、以下の章から構成されています。

- 第 2章 概要
  SRA74 の基本的な機能と SRA74 が生成するファイルについて説明します。
- 第 3章 ソースプログラムの記述方法
  SRA74 で処理するソースプログラムの記述方法について説明します。
- 第 4章 アセンブリ言語
  SRA74 で使用可能な 740 ファミリ のアセンブリ言語について説明します。
- 第 5章 構造化記述言語
  SRA74 で使用可能な構造化記述言語について説明します。
- 第 6章 疑似命令
  SRA74 で使用可能な疑似命令について説明します。
- 第 7章 マクロ命令
  SRA74 で使用可能なマクロ命令について説明します。
- 第 8章 操作方法
  SRA74 の操作方法について説明します。
- 付録 A エラーメッセージ一覧表
  SRA74 が表示するエラーメッセージについて、その内容と対策を一覧表で示します。
- 付録 B 命令一覧表
  SRA74 で使用可能な 740 ファミリ のアセンブリ言語の全命令を示します。
- 付録 C アドレッシングモード別命令一覧表
  SRA74 で使用可能な 740 ファミリ のアセンブリ言語をアドレッシングモード別に示します。
- 付録 D 疑似命令一覧
  SRA74 で使用可能な全疑似命令について、疑似命令ごとの内容を示します。

- 付録 E マクロ命令一覧
  SRA74 で使用可能な全マクロ命令について、マクロ命令ごとの内容を示します。
- 付録 F 構造化命令一覧
  SRA74 で使用可能な構造化命令について、構造化命令ごとの内容を示します。
- 予約語一覧　SRA74 における予約語をまとめています。

注意事項

本マニュアルに掲載されているプログラム例など、一部においてスペシャルページアドレッシングモードを表す ‘ ¥ ’ 記号を ‘ \ ’ で表していることがあります。これは、オペレーティングシステムによって表記が異なるためで、コードは同一ですのでいずれも使用できます。

第 2 章

概要

SRA74 は 740 ファミリ 用の構造化リロケータブルマクロアセンブラです。アセンブリ言語、構造化記述言語で記述されたソースプログラム (以下ソースファイルと呼びます) を LINK74[1]、及び LIB74[2]で処理可能なリロケータブルファイルに変換します。以下、この作業をアセンブルと呼びます。リロケータブルファイルは LINK74 により機械語データに変換されます。

2.1 機能

大規摸なソフトウェアの開発では、複数エンジニア間でのデータ交換や既存ソフトウェアの再利用など、プログラムの共有を容易に行う機能が必要になります。

SRA74 では、以下の機能によりこれらの作業を効率良く行うことができます。

1. 疑似命令.SECTION により、分割された領域に任意の名前（セクション名）を指定できます。リンク時には、ここで指定したセクション名を使用してアドレスを指定することが可能です。

2. 一つのファイル内で記述できるセクション数に制限がないので、多くの領域に分割された ROM 、RAM を持つユーザーシステムに対応できます。

3. 疑似命令.LIB 、及び.OBJ により、リンク対象ファイル名をソースプログラム中で指定できます (リンク時のリンク対象ファイル名指定が不要になります) 。

4. 疑似命令.VER によってバージョンを宣言することにより、リンク時にリロケータブルファイル間のバージョン確認を行うことが可能です。

この他の機能として以下のような特長があります。

1. アセンブリ言語、構造化記述言語を混在記述しているソースファイルのアセンブルが可能です (構造化記述言語命令をアセンブリ言語命令に変換しアセンブルを行います) 。

[1] 740 ファミリ 用リンケージエディタ
[2] 740 ファミリ 用ライブラリアン

2. 構造化記述言語をアセンブリ言語に変換したファイルが生成できます (このファイルを使用して、ユーザー側で最適化ができます) 。

3. エラー内容を格納したタグファイル [3] を生成できます (アセンブルエラーの修正を効率的に行うことができます) 。

4. 1 つのファイル中に RAM 領域、ROM 領域を混在できるため、アブソリュートアセンブラとして使用できます (但し、リンクは必要です) 。

5. マクロ機能 (命令) により、プログラミング環境を整えることができます。

6. アセンブル時のコマンドパラメータの指定により、エディタ、クロスリファレンサを起動することができます。

7. アセンブル時のコマンドパラメータ及び疑似命令の指定により、デバッグ情報を出力することができます。

2.2 生成ファイル

SRA74 では、以下の 5 種類のファイルを生成します。なお、コマンドパラメータ等の指定がない場合、ソースファイルと同じディレクトリに生成します。

1. リロケータブルファイル (以下 R74 ファイルと呼びます)

   - 機械語データと、その再配置情報を格納したファイルです。
   - コマンドパラメータの内容に関わらず生成します。但し、アセンブルエラーの発生など、SRA74 が異常終了した場合には生成しません。
   - コマンドパラメータ “–O” が指定されている場合には、その指定されたディレクトリに出力します。
   - シンボリックデバッグのためのシンボル情報を含んでいます。但し、規定値ではローカルシンボル情報は出力されません。
   - ローカルシンボル情報は、コマンドパラメータ “–S” を指定したときに R74 ファイル中に出力します。
   - ソースラインデバッグ情報は、コマンドパラメータ “–C” 又は疑似命令 “.FUNC” を指定したときに R74 ファイル中に出力します。
   - LINK74 で処理することにより、インテル HEX 形式の機械語ファイルを生成します。
   - ファイル属性は、.R74 です。

2. アセンブリ言語ファイル (以下 I ファイルと呼びます)

   - ソースファイル中のマクロを展開、及び構造化命令をニーモニックに変換したファイルです。

[3] タグの名称は、エラーやワーニングの場所を示す荷札 (タグ) に由来しています。

- コマンドパラメータ ”–I” を指定した時に R74 ファイルと同じディレクトリに生成されます。但し、同時にコマンドパラメータ ”–A” が指定された場合は生成されません。
- コマンドパラメータ ”–I” の指定のない場合はアセンブル作業用一時ファイルとして TMP ファイルと同じディレクトリに生成され、アセンブル終了時に自動的に消去されます。
- SRA74 のパス 2 処理の入力ファイル (ソースファイル) として使用します。なお、マクロ定義部、マクロ呼び出し部、及び構造化命令は全てコメント行として出力します。
- ソースファイル中で、疑似命令.INCLUDE により読み込むファイルも同様に.In(n は 00 〜 99) の属性を持つファイル名で出力します。
- I ファイルは、アセンブリ言語レベルでソースファイルの最適化を行う場合にご使用ください。
- ファイル属性は、.I です。

3. プリントファイル (以下 PRN ファイルと呼びます)

- 処理対象のソースファイルと、その配置アドレス及び生成データを示したファイルです。
- PRN ファイルは、コマンドパラメータ “–L” を指定したときに生成します。
- コマンドパラメータ “–O” が同時に指定されている場合には、その指定されたディレクトリに出力します。
- コマンドパラメータ “–I” が同時に指定されている場合には、構造化命令をニーモニックに変換し、ソース行の下に出力します。
- PRN ファイルは、プリント出力してデバッグなどにお使いください。
- ファイル属性は、.PRN です。

4. タグファイル (以下 TAG ファイルと呼びます)

- アセンブル中に発生したアセンブルエラーメッセージ及びワーニングメッセージを格納したファイルです。
- TAG ファイルは、コマンドパラメータ “–E” を指定したときに出力します。
- タグジャンプ機能を持つエディタを使用することで効率よくエラーの修正ができます。
- TAG ファイルは、エディタによるエラー修正時に参照用としてお使いください。
- ファイル属性は、.TAG です。

5. 一時ファイル (以下 TMP ファイルと呼びます)

- アセンブル作業用一時ファイルです。

- このファイルは、環境変数 TMP が設定されている場合 TMP により指定しているディレクトリ内に生成されます。
- このファイルは、アセンブル終了時に自動的に消去されます。
- ファイル属性は、.$$n(n は 1 〜 5) です。

注意事項

R74 ファイルはバイナリ形式ですので、プリンタ及び画面への出力は、行わないでください。

2.2.1 I ファイルの構成

図 2.1に示すソースファイルをアセンブルした場合に生成される I ファイルの出力例を図 2.2、及び図 2.3に示します。I ファイルは、以下の情報から構成されています。

- 構造化記述言語命令で記述された部分は、アセンブリ言語命令に変換され、その構造化記述言語命令の下に示されます。なお、構造化記述言語命令、及びマクロ命令は全てコメント行として出力します。そのため、I ファイルはソースファイルとして SRA74 でアセンブルすることができます。

```
        .INCLUDE        ZERO.H                  ;section z
        .PAGE
        .SECTION        AD_CONVERT
        .ORG            $E000
        .SEXT           GET_AD_DATA             ;sub routine
        .PUB            START
        .INCLUDE        MACRO.A74
START:
        D = 0
        T = 0
        S = STACK
        RAM_CL  "WORK0, WORK1"                  ;ram clear
        [ fAD_INT_OK ] = OFF
        X = 2                                   ;A/D convert 2 times
        do
                JSR     \GET_AD_DATA
                [ WORK0 ] = [ WORK0 ] + [ AD_RESULT ]
                [ WORK1 ] = [ WORK1 ] + 0 with_c
                X = --X
        while  X
        C = 0                                   ;A/D DATA average
        [ WORK1 ] = [ WORK1 ] >> 1
        [ WORK0 ] = [ WORK0 ] >>1 with_c
        [ AD_DATA ] = [ WORK0 ]
        .END
```

図 2.1: ソースファイル例

```
        .INCLUDE        ZERO.I00                ;section z
                              ↑インクルード命令行は、SRA74 が生成した I ファイルへ
                                名前が変更されます。
        .PAGE
        .SECTION        AD_CONVERT
        .ORG            $E000
        .SEXT           GET_AD_DATA             ;sub routine
        .PUB            START
        .INCLUDE        MACRO.I01
START:
;       D = 0           ←構造化命令はコメントとして出力し、アセンブリ言語に変換します。
                CLD
;       T = 0
                CLT
;       S = STACK
        LDX     #STACK
                TXS
;       RAM_CL  "WORK0, WORK1"                  ;ram clear
        LDA     #0
;       .REPEATI        ram, WORK0, WORK1       ←マクロ命令はコメントとして出力し
;       STA     ram                               アセンブリ言語に変換します。
;       .ENDM
        STA     WORK0
        STA     WORK1
;       .ENDM
;       [ fAD_INT_OK ] = OFF
                CLB     fAD_INT_OK
;        X = 2                                  ;A/D convert 2 times
                LDX     #2
```

図 2.2: I ファイル例 (ソースファイル前半部)

```
;       do
        .DO:
                JSR     \GET_AD_DATA
;               [ WORK0 ] = [ WORK0 ] + [ AD_RESULT ]
                        LDA     WORK0
                        CLC
                        ADC     AD_RESULT
                        STA     WORK0
;               [ WORK1 ] = [ WORK1 ] + 0 with_c
                        LDA     WORK1
                        ADC     #0
                        STA     WORK1
;               X = --X
                        DEX
;       while  X
                CPX     #0
                BNE     .DO
;       C = 0                                   ;A/D DATA average
                CLC
;       [ WORK1 ] = [ WORK1 ] >> 1
                LSR     WORK1
;       [ WORK0 ] = [ WORK0 ] >>1 with_c
                ROR     WORK0
;       [ AD_DATA ] = [ WORK0 ]
                LDA     WORK0
                STA     AD_DATA
        .END
```

図 2.3: I ファイル例 (ソースファイル後半部)

2.2.2 PRN ファイルの構成

図 2.4から図 2.11に、PRN ファイルの出力例を示します。PRN ファイルは、以下の情報を示しています。

1. ソースファイルの内容とこれに対応するアドレス、生成データを示す情報 (図 2.4、図 2.5 の前半部分、図 2.7、図 2.8、及び図 2.9の前半部分) 。
   - 外部ラベル [4]を参照している行は、ソースの横に 'E' を表示します。
   - パブリックラベル [5]を参照している行は、ソースの横に 'P' を表示します。
   - ローカルラベル [6]を参照している行は、ソースの横に 'L' を表示します。
   - シンボル [7]を参照している行は、ソースの横に 'S' を表示します。
   - ビットシンボル [8]を参照している行は、ソースの横に 'B' を表示します。
2. アセンブル結果 (図 2.5の後半部分と図 2.9の後半部分)
   エラー数、ワーニング数、全行数、コメント行数、セクションごとのメモリ容量を示します。
3. シンボルリスト (図 2.6の前半部分と図 2.10)
   プログラム中のシンボルとその値を、以下の 5 種類にわけて表示します。
   なお、シンボルリスト付きプリントファイルは、コマンドパラメータ "–LS" で生成できます。
   - `USED ( -d OPTION )`
     コマンド行からコマンドパラメータ "–D" により定義され、プログラム中で参照されているシンボルを示します。
   - `USED ( EQUATE )`
     疑似命令.EQU により定義され、プログラム中で参照されているシンボルを示します。
   - `USED ( BIT EQUATE )`
     疑似命令.EQU により定義され、プログラム中で参照されているビットシンボルを示します。
   - `UNUSED ( -d or EQUATE )`
     上記の方法により定義されており、プログラム中で参照されていないシンボルを示します。
   - `UNUSED ( BIT EQUATE )`
     疑似命令.EQU により定義されており、プログラム中で参照されていないビットシンボルを示します。

[4]他のファイル中で定義しているラベルを指します。なお、外部ラベルとパブリックラベルを総称してグローバルラベルと呼びます。
[5]このファイル中で定義しているラベルで、他のファイルから参照可能なラベルを指します。
[6]このファイル中で定義しているラベルで、ファイル中でのみ参照可能なラベルを指します。
[7]コマンドパラメータ "–D" 、及び.EQU 疑似命令で定義しているシンボルを指します。
[8].EQU 疑似命令で定義しているビットシンボルを指します。

4. ラベルリスト (図 2.6の後半部分と図 2.11)
   プログラム中のラベルとその値を、以下の 3 種類にわけて表示します。

   - USED (USER DEFINED)
     定義されており、プログラム中で参照されているラベルをセクション単位に示します。
   - USED (SYSTEM DEFINED)
     SRA74 で定義されたラベルで、プログラム中で参照されているラベルをセクション単位に示します (コマンドパラメータ “–I” が指定されている場合のみ出力します) 。
   - UNUSED (USER DEFINED)
     定義されているが、プログラム中で参照されていないラベルをセクション単位に示します。

5. 疑似命令.COL によるカラム数指定が 132 文字の場合、リストのヘッダ部にアセンブルを実行した時刻を次の形式で表示します。

```
Sat Jan 16 15:06:42 1993


 1                0         .INCLUDE        ZERO.H                ;section z
                 ↑インクルードのネストを示します。
 2                1         .SECTION        Z
 3                1         .ORG            0
 4 0000 (0001)    1 :WORK0:          .BLKB   1
 5 0001 (0001)    1 :WORK1:          .BLKB   1
 6 0002 (0001)    1 :AD_RESULT:      .BLKB   1
 7 0003 (0001)    1 :AD_DATA:        .BLKB   1
          ↑確保された領域のサイズを示します。
 8                1         .ORG            $FE
 9 00FE (0001)    1 :ICON:           .BLKB   1
                ↓パブリックラベルの参照を示します。
10 4,00FE        P 1 :fAD_INT_OK    .EQU    4, ICON
   ↑ビット番号を示します。
11                1         .INCLUDE        DEFINE.H
12 00BF           2 :STACK           .EQU    $BF
13 0000           2 :OFF             .EQU    0
14 0001           2 :ON              .EQU    1
```

図 2.4: PRN ファイル例 (ソースファイル前半部:コマンドパラメータ “–I” なし)

```
   15                0         .PAGE
   16                0         .SECTION        AD_CONVERT
   17                0         .ORG            $E000
   18                0         .SEXT           GET_AD_DATA             ;sub routine
   19                0         .PUB            START
   20                0         .INCLUDE        MACRO.A74
   21                1 RAM_CL: .MACRO  RAMS
   22                1         LDA     #0
   23                1         .REPEATI        ram, RAMS
   24                1         STA     ram
   25                1         .ENDM
   26                1         .ENDM
   27 E000           0 START:
   28 E000           0         D = 0
   29 E001           0         T = 0
   30 E002           0         S = STACK
   31 E005           0         RAM_CL  "WORK0, WORK1"                  ;ram clear
   32 E00B           0         [ fAD_INT_OK ] = OFF
   33 E00D           0         X = 2                                   ;A/D convert 2 times
   34 E00F           0         do
   35 E00F 2200     E 0                 JSR     \GET_AD_DATA
                    ↑外部ラベルの参照を示します。
   36 E011           0                 [ WORK0 ] = [ WORK0 ] + [ AD_RESULT ]
   37 E018           0                 [ WORK1 ] = [ WORK1 ] + 0 with_c
   38 E01E           0                 X = --X
   39 E01F           0         while  X
   40 E023           0         C = 0                                   ;A/D DATA average
   41 E024           0         [ WORK1 ] = [ WORK1 ] >> 1
   42 E026           0         [ WORK0 ] = [ WORK0 ] >>1 with_c
   43 E028           0         [ AD_DATA ] = [ WORK0 ]
   44                0         .END

ERROR   COUNT           00000
WARNING COUNT           00000
STRUCTURED STATEMENT    00014 LINES
TOTAL   LINE ( SOURCE ) 00044 LINES
TOTAL   LINE ( OBJECT ) 00044 LINES
COMMENT LINE ( SOURCE ) 00000 LINES
COMMENT LINE ( OBJECT ) 00000 LINES
OBJECT  SIZE ( Z          ) 00005 (0005) BYTES
               ↑セクション名 (最大 32 文字まで) を示します。
OBJECT  SIZE ( AD_CONVERT ) 00044 (002C) BYTES
```

図 2.5: PRN ファイル例 (ソースファイル後半部: コマンドパラメータ “–I” なし)

```
*** USED   SYMBOLS ( TYPE = -d OPTION      ) ***


*** USED   SYMBOLS ( TYPE = EQUATE         ) ***
↓シンボル名は最大 32 文字まで出力します。
OFF           0000p      STACK         00BFp
                  ↑パブリックラベルを示します。
*** USED   SYMBOLS ( TYPE = BIT EQUATE     ) ***

fAD_INT_OK    4,00FEp
              ↑該当ビットを 0 〜 7 の値で示します。
*** UNUSED SYMBOLS ( TYPE = -d or EQUATE   ) ***

ON            0001p

*** UNUSED SYMBOLS ( TYPE = BIT EQUATE     ) ***


*** USED   LABELS  ( TYPE = USER   DEFINED ) ***

EXTERNAL
    ↑外部参照を示します。
GET_AD_DATA   0000e
                  ↑外部ラベルを示します。
SECTION : Z
          ↑セクション名 (最大 122 文字まで) を示します。
AD_DATA       0003p      AD_RESULT     0002p      ICON          00FEp
WORK0         0000p      WORK1         0001p

SECTION : AD_CONVERT


*** UNUSED LABELS  ( TYPE = USER   DEFINED ) ***

EXTERNAL


SECTION : Z


SECTION : AD_CONVERT

START         E000p
```

図 2.6: PRN ファイル例 (シンボル及びラベルリスト: コマンドパラメータ “–I” なし)

```
 1                0          .INCLUDE        ZERO.H                  ;section z
 2                1          .SECTION        Z
 3                1          .ORG            0
 4 0000 (0001)    1 :WORK0:          .BLKB   1
 5 0001 (0001)    1 :WORK1:          .BLKB   1
 6 0002 (0001)    1 :AD_RESULT:      .BLKB   1
 7 0003 (0001)    1 :AD_DATA:        .BLKB   1
 8                1          .ORG            $FE
 9 00FE (0001)    1 :ICON:           .BLKB   1
10 4,00FE       P 1 :fAD_INT_OK     .EQU    4, ICON
11                1          .INCLUDE        DEFINE.H
12 00BF           2 :STACK           .EQU    $BF
13 0000           2 :OFF             .EQU    0
14 0001           2 :ON              .EQU    1
```

図 2.7: PRN ファイル例 (ソースファイル最前部: コマンドパラメータ “–I” あり)

```
15                  0          .PAGE
16                  0          .SECTION        AD_CONVERT
17                  0          .ORG            $E000
18                  0          .SEXT           GET_AD_DATA             ;sub routine
19                  0          .PUB            START
20                  0          .INCLUDE        MACRO.A74
21                  1 RAM_CL:  .MACRO  RAMS
22                  1          LDA     #0
23                  1          .REPEATI        ram, RAMS
24                  1          STA     ram
25                  1          .ENDM
26                  1          .ENDM
27 E000             0 START:
28                  0 ;         D = 0       ←構造化命令はコメントとなります。
29 E000 D8          0                  CLD  ←構造化命令をアセンブリ言語に変換します。
30                  0 ;         T = 0
31 E001 12          0                  CLT
32                  0 ;         S = STACK
33 E002 A2BF      S 0                  LDX     #STACK
                  ↑シンボルの参照を示します。
34 E004 9A          0                  TXS
35 E005             0          RAM_CL  "WORK0, WORK1"                  ;ram clear
36 E005 A900        0+         LDA     #0
                    ↑マクロ展開を示します。
37                  0+         .REPEATI        ram, WORK0, WORK1
38                  0+         STA     ram
39                  0+         .ENDM
40 E007 8500      P 0+         STA     WORK0
41 E009 8501      P 0+         STA     WORK1
42                  0+         .ENDM
43                  0 ;        [ fAD_INT_OK ] = OFF
44 E00B 9FFE      B 0                  CLB     fAD_INT_OK
                  ↑ビットシンボルの参照を示します。
45                  0 ;        X = 2                                   ;A/D convert 2 times
46 E00D A202        0                  LDX     #2
```

図 2.8: PRN ファイル例 (ソースファイル中間部: コマンドパラメータ “–I” あり)

```
   47                0 ;       do
   48 E00F           0         .DO:
   49 E00F 2200    E 0                 JSR     \GET_AD_DATA
   50                0 ;               [ WORK0 ] = [ WORK0 ] + [ AD_RESULT ]
   51 E011 A500    P 0                         LDA     WORK0
   52 E013 18        0                         CLC
   53 E014 6502    P 0                         ADC     AD_RESULT
   54 E016 8500    P 0                         STA     WORK0
   55                0 ;               [ WORK1 ] = [ WORK1 ] + 0 with_c
   56 E018 A501    P 0                         LDA     WORK1
   57 E01A 6900      0                         ADC     #0
   58 E01C 8501    P 0                         STA     WORK1
   59                0 ;               X = --X
   60 E01E CA        0                         DEX
   61                0 ;       while  X
   62 E01F E000      0                 CPX     #0
   63 E021 D0EC    L 0                 BNE     .DO
   64                0 ;       C = 0                                   ;A/D DATA average
   65 E023 18        0                 CLC
   66                0 ;       [ WORK1 ] = [ WORK1 ] >> 1
   67 E024 4601    P 0                 LSR     WORK1
   68                0 ;       [ WORK0 ] = [ WORK0 ] >>1 with_c
   69 E026 6600    P 0                 ROR     WORK0
   70                0 ;       [ AD_DATA ] = [ WORK0 ]
   71 E028 A500    P 0                 LDA     WORK0
   72 E02A 8503    P 0                 STA     AD_DATA
   73                0         .END

ERROR   COUNT           00000
WARNING COUNT           00000
STRUCTURED STATEMENT    00014 LINES
TOTAL   LINE ( SOURCE ) 00043 LINES
TOTAL   LINE ( OBJECT ) 00073 LINES
COMMENT LINE ( SOURCE ) 00000 LINES
COMMENT LINE ( OBJECT ) 00014 LINES
OBJECT  SIZE ( Z          ) 00005 (0005) BYTES
OBJECT  SIZE ( AD_CONVERT ) 00044 (002C) BYTES
```

図 2.9: PRN ファイル例 (ソースファイル最後部: コマンドパラメータ “–I” あり)

```
*** USED   SYMBOLS ( TYPE = -d OPTION      ) ***

*** USED   SYMBOLS ( TYPE = EQUATE         ) ***
OFF            0000p      STACK        00BFp
*** USED   SYMBOLS ( TYPE = BIT EQUATE     ) ***
fAD_INT_OK    4,00FEp
*** UNUSED SYMBOLS ( TYPE = -d or EQUATE   ) ***
ON             0001p
*** UNUSED SYMBOLS ( TYPE = BIT EQUATE     ) ***
```

図 2.10: PRN ファイル例 (シンボルリスト: コマンドパラメータ “–I” あり)

```
*** USED   LABELS  ( TYPE = USER   DEFINED ) ***

EXTERNAL

GET_AD_DATA   0000e

SECTION : Z

AD_DATA       0003p      AD_RESULT     0002p      ICON          00FEp
WORK0         0000p      WORK1         0001p

SECTION : AD_CONVERT


*** USED   LABELS  ( TYPE = SYSTEM DEFINED ) ***

SECTION : Z


SECTION : AD_CONVERT

.DO           E00F'

*** UNUSED LABELS  ( TYPE = USER   DEFINED ) ***

EXTERNAL


SECTION : Z


SECTION : AD_CONVERT

START         E000p
```

図 2.11: PRN ファイル例 (ラベルリスト: コマンドパラメータ “–I” あり)

2.2.3 TAG ファイルの構成

図 2.12に、TAG ファイルの出力例を示します。TAG ファイルは、以下の情報を示しています。

- エラー、又はワーニングが発生した場所のファイル名、そのファイル中の行番号、通し行番号、エラー番号、エラーメッセージを示します。

TAG ファイルはプリンタに出力してエディタでのエラー修正時に参照用としてお使いください。

```
TEST.A74 115 ( TOTAL LINE 115 ) Error 18: Relative jump is out of range
TEST.A74 127 ( TOTAL LINE 127 ) Error 22: Value is out of range "data"
TEST.A74 593 ( TOTAL LINE 593 ) Error 23: "()" format error ";"
```

図 2.12: TAG ファイル例

第 3 章

ソースプログラムの記述方法

3.1　ソースプログラムの構成

ソースプログラムは、行を単位として構成しています。以下に、行記述の規則を示します。

1. 各行は 1 行ごとに完結しています。従って 1 命令を、2 行以上にわたって記述することはできません。
2. 1 行の文字数は、改行コードを含めて最大 256 文字です。SRA74 は、256 文字を越えた部分を無視します。
3. 行は内容別に以下の 5 種類の行に分類できます。
    - アセンブリ言語命令行
      740 ファミリ のアセンブリ言語命令を記述した行です。
      この行は対応する機械語データを生成します。
    - 構造化記述言語命令行
      740 ファミリ の構造化記述言語命令を記述した行です。
      この行は対応する機械語データを生成します。
    - 疑似命令行
      740 ファミリ の疑似命令を記述した行です。
    - マクロ命令行
      740 ファミリ のマクロ命令を記述した行です。
    - コメント行
      この行は SRA74 で処理しませんのでユーザーが自由に使用できます。

3.2 行の構成

本節では、それぞれの行の構成について説明します。なお、以下に表記上用いている記号の説明及び規則を示します。

1. △、▲はスペース又はタブコードを示しています。
   △の部分は必須、▲の部分は省略可能です。

2. ラベルを記述する場合 ':'(コロン) は必須ではありませんが、':' を省略した場合には各命令との間にスペース、又はタブコードが必要になります。但し、アセンブル時にはコマンドパラメータ "–U" の指定が必要です。従って通常 ':' を付加して記述することを推奨します。

3.2.1 アセンブリ言語命令行

アセンブリ言語命令行の構成を以下に示します。この行の詳細については、第 4章、付録 B、及び付録 Cを参照してください。

▲ | ラベル | :▲ | オペコード | △ | オペランド | ▲; | コメント | <RET>

1. ラベル欄
   行を他の場所から参照するためのラベルを記述します。

2. オペコード欄
   740 ファミリ のアセンブリ言語ニーモニック (以下オペコードと呼びます) を記述します。オペコードは、英大文字/小文字を区別しません。従って NOP 、nop 、Nop いずれも有効です。
   SRA74 はオペコードを予約語として識別しますので、ラベルを記述しない場合は行の先頭から記述することができます。

3. オペランド欄
   オペコードの処理対象を記述します。

   - オペランドにデータが 2 つ以上ある場合、データ間を ','(カンマ) で区切ってください。
   - カンマの両側には、スペース又はタブコードが記述できます。

4. コメント欄
   この欄は、SRA74 で処理しませんのでユーザーが自由に使用できます。

3.2.2 構造化記述言語命令行

構造化記述言語命令 (以下構造化命令と呼びます) 行の構成を以下に示します。この行の詳細については、第 5章、及び付録 Fを参照してください。

▲ | ラベル | :▲ | 構造化命令 | △ | 条件式 | ▲; | コメント | <RET>

1. ラベル欄
   行を他の場所から参照するためのラベルを記述します。

2. 構造化命令欄
   740 ファミリ の構造化命令を記述します。構造化命令は、英大文字/小文字を区別しません。従って IF 、if 、If いずれも有効です。
   SRA74 は構造化命令を予約語として識別しますので、ラベルを記述しない場合は行の先頭から記述することができます。

3. 条件式欄
   740 ファミリ の構造化命令の処理対象を記述します。

4. コメント欄
   この欄は、SRA74 で処理しませんのでユーザーが自由に使用できます。

3.2.3 疑似命令行

疑似命令行の構成を以下に示します。この行の詳細については、第 6章、及び付録 Dを参照してください。

▲ [シンボル/ビットシンボル] △ [.EQU] △ [オペランド] ▲; [コメント] <RET>

▲ [ラベル] :▲ [疑似命令] △ [オペランド] ▲; [コメント] <RET>

1. シンボル/ビットシンボル欄
   疑似命令.EQU により値を割り当てるシンボル、ビットシンボルを記述します。

2. ラベル欄
   行を他の場所から参照するためのラベルを記述します。

3. 疑似命令欄
   740 ファミリ の疑似命令を記述します。疑似命令は、英大文字/小文字を区別しません。従って.END 、.end 、.End いずれも有効です。
   SRA74 は疑似命令を予約語として識別しますので、ラベルを記述しない場合は行の先頭から記述することができます。

4. オペランド欄
   疑似命令の処理対象を記述します。

   - オペランドにデータが 2 つ以上ある場合、データ間を ‘,’ (カンマ) で区切ってください。
   - カンマの両側には、スペース又はタブコードが記述できます。

5. コメント欄
   この欄は、SRA74 で処理しませんのでユーザーが自由に使用できます。

3.2.4 マクロ命令行

マクロ命令行の構成を以下に示します。この行の詳細については、第 7章、及び付録 Eを参照してください。

▲ | マクロ名 | :▲ | .MACRO | △ | オペランド | ▲; | コメント | <RET>

▲ | ラベル | :▲ | マクロ命令 | △ | オペランド | ▲; | コメント | <RET>

1. マクロ名欄
   マクロ定義を呼び出すための名前です。

2. ラベル欄
   マクロ命令行を他の場所から参照するためのラベル (名前) を記述します。

3. マクロ命令欄
   740 ファミリ のマクロ命令を記述します。マクロ命令は、英大文字/小文字を区別しません。従って.REPEATI 、.repeati 、.RepeatI いずれも有効です。
   SRA74 はマクロ命令を予約語として識別しますので、ラベルを記述しない場合は行の先頭から記述することができます。

4. オペランド欄
   マクロ命令の処理対象を記述します。

   - オペランドにデータが 2 つ以上ある場合、データ間を ‘,’ (カンマ) で区切ってください。
   - カンマの両側には、スペース又はタブコードが記述できます。

5. コメント欄
   この欄は、SRA74 で処理しませんのでユーザーが自由に使用できます。

3.2.5 コメント行

コメント行は、行の最初の文字 (▲を除く) を ‘;’(セミコロン) で始めてください。
コメント行の構成を以下に示します。

▲; | コメント | <RET>

注意事項

コメント行の最初の文字が ‘;’ 以外である場合、 SRA74 はその行をコメント行とは認識せずアセンブル処理を行います。その結果、アセンブルエラーが発生したり予期しないコードを生成する可能性がありますので注意してください。

3.3 欄の記述方法

本節では、各命令行において記述形式が共通な欄についてのみ説明します。記述形式が異なる欄については、それぞれ第 4章、第 5章、第 6章、第 7章を参照してください。

3.3.1 シンボル/ビットシンボル/ラベル欄

この欄は、各命令行とも記述形式は同じです。但し SRA74 は、シンボル、ビットシンボル、ラベル[1]を区別して管理します。以下に記述上の規則を示します。

1. 名前には、英数字、'_'(アンダーライン) 、'.'(ピリオド) 、及び '?'(クエッションマーク) が使用できます。但し、1 文字目に使用できる文字は英字と '_'(アンダーライン) だけです。また、文字数は、':' を含めて最大 255 文字まで使用できます。

2. 大文字/小文字を区別します。従って BIG と Big は異なった名前として判断します。

3. 名前に、予約語は使用できません。SRA74 では、レジスタ名、フラグ名、オペコード、構造化命令、疑似命令、及びマクロ命令を予約語として処理します。

4. 特殊文字'..'（ピリオド 2 個）で始まるラベルは SRA74 で予約されていますので使用しないでください。また、ピリオド 1 個およびピリオド 3 個以上で始まる名前も使用できません。

5. "??"(クエッションマーク 2 個) で始まるラベルは、そのセクション内だけで有効なセクション内ローカルラベルとして扱われます。従って、異なるセクションで同一名のラベルが使用できます。参照した場合は、常にそのセクション内のラベルのみが参照されます。

[1]疑似命令.EQU 、又はコマンドパラメータ "–D" により定義したものをシンボル又はビットシンボルとし、これ以外で定義したものをラベルとして扱います。

```
例）          .SECTION PROG1
    ??MAIN:   NOP
               :
              BRA      ??MAIN     セクション名‘‘PROG1’’ の‘‘??MAIN’’
                                  へジャンプする。
              .SECTION PROG2
    ??MAIN:   NOP
               :
              BRA      ??MAIN     セクション名‘‘PROG2’’ の‘‘??MAIN’’
                                  へジャンプする。
```

6. シンボル、ビットシンボル、及びラベルの直前に ‘:’ をつけることにより.PUB 宣言を省略することができます。

```
例）:SYMBOL   .EQU     10
               :
              .SECTION PROG2
    :LABEL:   NOP
               :
```

ラベルを記述する場合、名前の直後に ‘:’(コロン) をつけることができます。シンボルとの区別を容易にするため、又エディタでのラベル検索を効率的に行うため ‘:’ を付加した記述を推奨します。但し、シンボル、ビットシンボルを記述する場合は ‘:’ を付加するとエラーとなりますのでご注意ください。

3.3.2 コメント欄

ユーザーの任意の情報を記述することができます。以下に記述形式を示します。

1. コメントの先頭には必ず ‘;’(セミコロン) を付けて記述してください。
2. コメント欄にはどのような文字も記述できます。

第 4 章

アセンブリ言語

4.1 アドレッシングモード

アドレッシングモードは、命令が処理対象データを指定する基本的な方式 (モード) のことです。740 ファミリ では 19 通りのアドレッシングモードがあり、アドレッシングモードごとにオペランド形式が決まっています。以下に、オペランド記述に関係するアドレッシングモードについて示します。

1. インプライド
   オペコードのみの命令です。オペランドには何も指定しません。
   例)　　`BRK`
2. アキュムレータ
   処理対象データをアキュムレータの内容とする方法です。
   例)　　`ASL　　A`
3. イミディエイト
   処理対象データをオペランドで直接指定する方法です。オペランドに記述する値は、'#' で始めてください。
   例)　　`ADC　　#IMMDATA`
4. ゼロページ
   処理対象データとして、ゼロページ領域 (00_{16}〜FF_{16}) を指定する方法です。
   例)　　`ADC　　ZWORK`
5. ゼロページ X
   処理対象データとして、ゼロページアドレスをレジスタ X で修飾する方法です。','(カンマ) の後にレジスタ名 ' X ' を記述してください。
   例)　　`ADC　　ZWORK,X`
6. ゼロページ Y
   処理対象データとして、ゼロページアドレスをレジスタ Y で修飾する方法です。','(カンマ) の後にレジスタ名 ' Y ' を記述してください。
   例)　　`LDX　　ZWORK,Y`

7. ゼロページインダイレクト
   処理対象データをメモリ間接で示す方法です。オペランドには、対象アドレスを格納したゼロページアドレスを記述し、メモリに 2 バイトの対象アドレスを格納します。オペランドに記述する値は、() で示してください。
   例)　　JMP　　(ZWORK)

8. ゼロページインダイレクト X
   処理対象データをメモリ間接とレジスタ X で修飾して示す方法です。オペランドには、対象アドレスを格納したゼロページアドレスを記述し、メモリに 2 バイトの対象アドレスを格納します。‘ ,’(カンマ) の後にレジスタ名 ‘ X ’ を記述し、オペランドに記述する値は、() で示してください。
   例)　　ADC　　(ZWORK,X)

9. ゼロページインダイレクト Y
   処理対象データをメモリ間接とレジスタ Y で修飾して示す方法です。オペランドには、対象アドレスを格納したゼロページアドレスを記述し、メモリに 2 バイトの対象アドレスを格納します。オペランドに記述する値は、() で示し、‘,’(カンマ) の後にレジスタ名 ‘ Y ’ を記述してください。
   例)　　ADC　　(ZWORK),Y

10. アブソリュート
    処理対象データとして、一般ページ領域 (0100_{16}〜 $FFFF_{16}$) を指定する方法です。
    例)　　ADC　　WORK

11. アブソリュート X
    処理対象データとして、一般ページアドレスをレジスタ X で修飾する方法です。‘,’(カンマ) の後にレジスタ名 ‘ X ’ を記述してください。
    例)　　ADC　　WORK,X

12. アブソリュート Y
    処理対象データとして、一般ページアドレスをレジスタ Y で修飾する方法です。‘,’(カンマ) の後にレジスタ名 ‘ Y ’ を記述してください。
    例)　　ADC　　WORK,Y

13. アブソリュートインダイレクト
    処理対象データをメモリ間接で示す方法です。オペランドには、対象アドレスを格納した一般ページアドレスを記述し、メモリに 2 バイトの対象アドレスを格納します。オペランドに記述する値は、() で示してください。
    例)　　JMP　　(WORK)

14. スペシャルページ
    処理対象データとして、スペシャルページ領域を ($FF00_{16}$〜 $FFFF_{16}$) を指定する方法です。オペランドに記述する値は、‘¥’ 又は ‘ \ ’ で示してください。
    例)　　JSR　　¥WORK

15. ゼロページビット

処理対象データとして、ゼロページ領域 (00_{16}〜FF_{16}) の特定ビットを指定する方法です。また、オペランドにビットシンボルを記述すると、そのビットシンボルのビット値とアドレスが参照されます。

例)　　CLB　　0,ZWORK　⇒ 'ZWORK' のビット 0 を指定します。

例)　　CLB　　BITSYMBOL

16. アキュムレータビット

処理対象データとして、アキュムレータの特定ビットを指定する方法です。また、第 1 オペランドにビットシンボルを記述すると、そのビットシンボルのビット値だけが参照されます。

例)　　CLB　　1,A　⇒ アキュムレータのビット 1 を指定します。

例)　　CLB　　BITSYMBOL,A

17. レラティブ

本命令の先頭アドレスに、オペランドの内容を加えた番地にジャンプします。但し、オペランドには相対値そのものを記述することはできません。オペランドに、ラベル又は対象アドレスを記述すると、SRA74 が相対値を計算します。

例)　　BRA　　*-12

例)　　BRA　　NEXT

18. ゼロページビットレラティブ

処理対象データとして、ゼロページ領域 (00_{16}〜FF_{16}) の特定ビットを指定します。また第 1 オペランドにビットシンボルを記述すると、そのビットシンボルのビット値とアドレスが参照され、そのビットの状態により本命令の先頭アドレスに、最終オペランドの内容を加えた番地にジャンプする方法です。但し、最終オペランドには相対値そのものを記述することはできません。ラベル、又は対象アドレスを記述すると、SRA74 が相対値を計算します。

例)　　BBC　　2,ZWORK,NEXT　⇒ 'ZWORK' のビット 2 を指定します。

例)　　BBC　　BITSYMBOL,NEXT

19. アキュムレータビットレラティブ

処理対象データとして、アキュムレータの特定ビットを指定します。また第 1 オペランドにビットシンボルを記述すると、そのビットシンボルのビット値だけが参照され、そのビットの状態により本命令の先頭アドレスに、最終オペランドの内容を加えた番地にジャンプする方法です。但し、最終オペランドには相対値そのものを記述することはできません。ラベル、又は対象アドレスを記述すると、SRA74 が相対値を計算します。

例)　　BBC　　3,A,NEXT　⇒ アキュムレータのビット 3 を指定します。

例)　　BBC　　BITSYMBOL,A,NEXT

各命令のアドレッシングモードごとの記述形式は、付録 Cを参照してください。

4.2 オペランドデータ形式

オペランドには、以下の 4 種類のデータ形式が記述できます。

1. 数値定数

   - 数値定数の前に、'+' 又は '−' の演算子をつけて正、又は負の値の表現ができます。'+' も '−' もない場合は、正の値として処理します。
   - 数値の種類を表す記号と数値との間には、スペース又はタブをいれることはできません。
     例)　　.BYTE　$ 64 ⇒ エラーとなります。
   - 数値定数には、2 進、8 進、10 進、16 進数が使用できます。
   - 2 進数 ⇒ 2 進数で構成され、先頭に '%' を付けるか、最後に 'B' 、又は 'b' を付けて記述してください。
     例)　　.BYTE　%100110
     例)　　.BYTE　100110B
   - 8 進数 ⇒ 8 進数で構成され、先頭に '@' を付けるか、最後に 'O' 、'o' 、又は 'Q' 、'q' を付けて記述してください。
     例)　　.BYTE　@70
     例)　　.BYTE　70o
     例)　　.BYTE　70Q
   - 10 進数 ⇒ 10 進数で構成され、特に何も指定しません。23 、256 のように整数のみで記述してください。
     例)　　.BYTE　100
   - 16 進数 ⇒ 16 進数で構成され、先頭に '$' を付けるか、最後に 'H' 、又は 'h' を付けて記述してください。先頭が英文字 (A ～ F) で始まる場合は先頭に 0 を付加してください。
     例)　　.BYTE　$64
     例)　　.BYTE　64H
     例)　　.BYTE　0ABH

2. 文字定数

   - 文字には、ASCII コードで定義されている文字が記述できます。
   - 文字定数は、'(シングルクォート) 又は "(ダブルクォート) で囲んで記述してください。各文字は、7 ビット ASCII コード (再上位ビットは 0) に対応しています。
     例)　　.BYTE　'A' ⇒ 41_{16}を設定します。

3. 記号定数

   - 記号はシンボル、ビットシンボル、ラベル、及び現在のステートメントの先頭を表す '*' の 4 種類があります。ビットシンボルは、割り当てられたビットの値と、そ

のビットの属するアドレスを、シンボルは絶対値を、ラベル又は '*' は相対又は絶対値をもちます。

例) `.WORD SUB` ⇒ ラベル SUB のアドレスを設定します。

例) `BRA *+2` ⇒ 現在のアドレスに 2 を加えたアドレスへジャンプします。

4. 式

- 式は、数値定数、文字定数、記号定数、演算子の組み合わせで構成します。演算子と各項の間には、必要に応じてスペース又はタブを入れることができます。

  例) `TBL + 1`

- 式は左から右に計算します (演算子の優先順位はありません) 。

  例) `2*3` ⇒ 結果は 6 になります。

  例) `2+6/2` ⇒ 結果は 4 になります。

4.3 演算子

表 4.1に、オペランドデータ記述で使用可能な演算子の一覧表を示します。

表 4.1: 演算子一覧表

分類	演算子 [1]	内容
単項演算子	+	正の数を表す
	-	負の数を表す
	!	1 の補数をとる
	<	ラベル、又はシンボルの上位 8 ビットを切り出す
	>	ラベル、又はシンボルの下位 8 ビットを切り出す
	SIZEOF [2]	セクションの大きさを求める
	BK [3]	ラベルのバンク値を取得する
	BL [3]	ラベルのエクストラエリア値 [4] を取得する
2項演算子	+	加算
	-	減算
	*	乗算
	/	除算
	&	ビットごとの AND
	|	ビットごとの OR

注意事項

1. SRA74 では、V.4.00.00 以降では、式の評価は 32 ビット符号付き整数値として処理されます。ただし、式の結果も 32 ビット符号付き整数値で扱われるのは次の 2 つの値のみです。

   - .ORG 指示命令のオペランド
   - -BANK オプションが指定されたときのラベルの値

   次の例に示すような式を記述した場合、V.2.00.10 以前の SRA74 の演算結果と異なる結果となる場合があります。
   例)　　`2223h + 0FFFFh / 2`

   - V.2.00.10 以前の演算結果　`1111H`
   - V.4.00.00 以降の演算結果　`9111H`

2. SIZEOF の値は、参照セクションがリロケータブルかアブソリュートかに関係なくリンク時に決まります。SIZEOF 演算子は、セクション名にしか使用できません。又、SIZEOF 演算子を用いる場合は、セクション名との間には必ずスペースを入れてください。
   例)　　`.WORD　　SIZEOF DATA`

3. 演算子 BK,BL のオペランドには、アセンブル実行時に値が確定するラベルを記述してください。

4. エクストラエリア値は、オペランド（ラベル）の値の下位 12 ビット値に 1000H を加算した値です。

5. 演算は左から右へ行います (演算子の優先順位はありません) 。
   例)　　`2+6/2`　⇒ 結果は 4 になります。

6. ビットシンボルに対する演算、“.ZEXT”と “.SEXT”で参照されているラベル間の演算はできません。

第 5 章

構造化記述言語

5.1 構造化命令の機能

プログラムの記述において分岐命令とラベルの多用はプログラムの可読性を低下させる大きな原因となります。構造化記述言語では、条件分岐構造、繰り返し構造などをラベル及び分岐命令を用いずに記述できるため、プログラムの流れの読み取り易い記述ができます。また、レジスタ及びフラグを直接操作する命令により、効率を意識した記述も可能です。

5.2 文の種類

構造化記述言語には、以下の 7 種類の文があります。l(エル) を付加した場合それぞれ分岐命令に “JMP” を生成するロング分岐となります。以下に、記述例と右側に同一の処理を行うアセンブラのプログラムを示します。各文の詳細は付録 Fを参照してください。

1. 代入文
   右辺を左辺に代入します。

```
[ MEM ] = 10              ; LDM   #10,MEM
[ MEM ] = [ MEM1 ]        ; LDA   MEM1
                          ; STA   MEM
```

2. (l)if 〜 (l)else 〜 endif 文
   if 文は、制御の流れを 2 方向に変える命令で、分岐する方向は条件式によって決定されます。

```
if [ MEM ]                ;         LDA   MEM
        [ WORK ] = 1      ;         BEQ   I0
else                      ;         LDM   #1,WORK
        [ WORK ] = 2      ;         BRA   I1
endif                     ; I0:
                          ;         LDM   #2,WORK
                          ; I1:
```

3. (l)for ～ next 文

for 文は、繰り返しを制御する命令で、指定した条件式が真である間、文を繰り返し実行します。

```
for [ MEM ]             ; F0:
        jsr   output    ;         LDA   MEM
next                    ;         BEQ   F1
                        ;         JSR   OUTPUT
                        ;         BRA   F0
                        ; F1:
```

4. (l)do ～ while 文

do 文は、条件式が満たされている (真である) 間、文を繰り返し実行します。

```
do                      ; D0:
        jsr   output    ;         JSR   OUTPUT
while  [ MEM ]          ;         LDA   MEM
                        ;         BNE   D0
```

5. (l)switch ～ case ～ ends 文

switch 文は、条件式の値によって制御をいずれかの文に渡すものです。

```
switch  [ MEM ]         ;         LDA   MEM
    case  1             ;         CMP   #1
        jsr  output1    ;         BNE   S0
        break           ;         JSR   OUTPUT1
    case  2             ;         BRA   S3
        jsr  output2    ; S0:
        break           ;         CMP   #2
    case  3             ;         BNE   S1
        jsr  output3    ;         JSR   OUTPUT2
        break           ;         BRA   S3
    default             ; S1:
        jsr  output4    ;         CMP   #3
ends                    ;         BNE   S2
                        ;         JSR   OUTPUT3
                        ;         BRA   S3
                        ; S2:
                        ;         JSR   OUTPUT4
                        ; S3:
```

6. (l)break 文
   break 文は、該当する for 文、do 文、又は switch 文の実行を中止して、その次に実行する文に制御を渡します。

```
for [ MEM1 ]            ; F0:
        if  [ MEM2 ]    ;         LDA   MEM1
                break   ;         BEQ   F1
        endif           ;         LDA   MEM2
        jsr  output     ;         BEQ   I0
next                    ;         BRA   F1
                        ; I0:
                        ;         JSR   OUTPUT
                        ;         BRA   F0
                        ; F1:
```

7. (l)continue 文
   continue 文は、それを含む最小の繰り返しの for 文、do 文中の最後の文の後に仮想的に空文を置き、その空文に制御を渡します。

```
for [ MEM1 ]            ; F0:
        if  [ MEM2 ]    ;         LDA   MEM1
              continue  ;         BEQ   F1
        endif           ;         LDA   MEM2
        jsr  output     ;         BEQ   I0
next                    ;         BRA   F0
                        ; I0:
                        ;         JSR   OUTPUT
                        ;         BRA   F0
                        ; F1:
```

5.3 記述上の注意事項

構造化記述言語でプログラミングを行う場合の記述上の注意事項について説明します。

1. 代入文、又は各制御文 (if 文など) の条件式欄に 740 ファミリ の各アドレッシングモードによって参照されるメモリを記述する場合は"[ ]"、又は"{ }"で囲んで記述します。詳細は F.3.1の代入文の展開例を参照してください。

2. 代入文、又は各制御文 (if 文など) の条件式欄にビットシンボルで参照できる任意のビットを記述する場合は"[ ]"、又は"{ }"で囲んで記述します。但し、アキュムレータの任意のビットを参照するために以下の予約語が用意されています。この場合は"[ ]"、又は"{ }"で囲む必要はありません。また、大文字/小文字を区別しませんので BIT_A0 、bit_a0 いずれでも有効です。

BIT_A0　アキュムレータのビット 0　BIT_A1　アキュムレータのビット 1
BIT_A2　アキュムレータのビット 2　BIT_A3　アキュムレータのビット 3
BIT_A4　アキュムレータのビット 4　BIT_A5　アキュムレータのビット 5
BIT_A6　アキュムレータのビット 6　BIT_A7　アキュムレータのビット 7

詳細は F.3.1の A） のレジスタ、フラグ代入文の展開例を参照してください。

3. 代入文、又は各制御文 (if 文など) の条件式欄に 740 ファミリ の各種レジスタを記述する場合にはそのまま記述してください。これは SRA74 では予約語として以下のような名前で用意されています。また、大文字/小文字を区別しませんので A 、a いずれでも有効です。

   A　アキュムレータ　　X　インデックスレジスタ X
   Y　インデックスレジスタ Y　　S　スタックポインタ
   P　プロセッサステータスレジスタ

   詳細は F.3.1の A） のレジスタ、フラグ代入文の展開例を参照してください。

4. 代入文、又は各制御文 (if 文など) の条件式欄に 740 ファミリ のステータスレジスタ内のフラグを記述する場合にはそのまま記述してください。これは SRA74 では予約語として以下のような名前で用意されています。また、大文字/小文字を区別しませんので C 、c いずれでも有効です。

   C　キャリーフラグ　　Z　ゼロフラグ
   I　割り込み禁止フラグ　　D　10 進モードフラグ
   T　X 修飾演算モードフラグ　V　オーバフローフラグ
   N　ネガティブフラグ

   詳細は F.3.1の A） のレジスタ、フラグ代入文の展開例、及び F.3.2のフラグ条件式の展開例を参照してください。

5. SRA74 では、以下のような記述 (前方参照) を行った場合ラベルとして処理します (LDA 命令などによるコード展開が行われます) 。従って、“BITSYM” が後でビットシンボルとして定義されると展開コードが合わなくなります (このような場合 SRA74 ではエラーとなります) 。ビットシンボル (BITSYM) を記述する場合は、まず定義してから参照するようにプログラムを書き換えてください。

   例)

```
            ;if [ BITSYM ]
                LDA     BITSYM          展開コード
                BEQ     .I0             展開コード
                   :
            ;endif
            .I0:
                   :
BITSYM      .equ        1,80h           ビットシンボルの定義
```

   詳細は付録 Fの構文図を参照してください。

6. 生成コードサイズの小さい記述方法

- IF 文、FOR 文の場合、＞より＞＝の方が効率がよく、＜＝より＜の方が効率がよくなります。DO 文の場合は、＞＝より＞の方が効率がよく、＜より＜＝の方が効率がよくなります。
  詳細は F.3.2のメモリ条件式の展開例を参照してください。
- 最低１回以上処理を行う場合は、FOR 文より DO 文の方が効率がよくなります。
- SWITCH 文の場合、lswitch にするより、必要なところだけに lbreak を使った方が効率がよくなります。

LSWITCH 文	アセンブリ言語	LBREAK 文	アセンブリ言語
lswitch ［MEM］	CMP # 1	switch ［MEM］	CMP # 1
case 1	BEQ .Z0	case 1	BNE .S2
NOP	JMP .S2	NOP	NOP
break	.Z0:	lbreak	JMP .S0
case 2	NOP	case 2	.S2:
:	JMP .S0	:	:
case 16	.S2:	case 16	.S16:
NOP	:	NOP	CMP # 2
break	.S16:	break	BNE .S17
default	CMP # 2	default	NOP
NOP	BEQ .Z15	NOP	BRA .S0
ends	JMP .S17	ends	.S17:
	.Z15:		NOP
	NOP		.S0:
	JMP .S0		
	.S17:		
	NOP		
	.S0:		

7. アキュムレータの値について
   構造化記述言語の演算では、ほとんどの場合アキュムレータを介します。このため、次に示すように構造化命令実行後はアキュムレータの値が変化しますので、ご注意ください。

```
;[ WORK ] = [ DATA1 ] + [ DATA2 ]
        LDA     DATA1
        CLC
        ADC     DATA2
        STA     WORK
;IF [ WORK ] & 0FH > 4
        LDA     WORK
        AND     #0FH
        CMP     #4
        BEQ     .I0
        BCC     .I0
;   X = ++X
            INX
;ENDIF
.I0:
```

   但し、演算なしのレジスタへの転送及び比較ではアキュムレータを用いずに効率良くアセンブリ言語に展開します。

```
;X = [ WORK ]
        LDX     WORK
;IF X > [ WORK ]
        CPX     WORK
        BEQ     .I0
        BCC     .I0
;   Y = ++Y
            INY
;ENDIF
.I0:
```

   詳細は F.3.1の D） の 2 項演算代入文の展開例を参照してください。

5.4 構造化演算子

表 5.1に、構造化記述言語で使用可能な演算子の一覧表を示します。

表 5.1: 構造化記述言語の演算子一覧表

分類	演算子 [1]	内容	分類	演算子	内容
単項演算子	+	正の数を表す	2項演算子	+ [1]	加算
	-	負の数を表す		- [1]	減算
	~	1 の補数をとる		* [2]	乗算
	++	インクリメント		/ [2]	除算
	--	デクリメント		% [2]	除算の余り
比較演算子	<	より小さい		&	ビットごとの AND
	>	より大きい		|	ビットごとの OR
	==	等しい		^	ビットごとの排他的な OR
	!=	等しくない		&& [3]	論理 AND
	<=	小さいか等しい		|| [3]	論理 OR
	>=	大きいか等しい		<<	左シフト
				>>	右シフト

注意事項

1. SRA74 は、構造化記述演算は 8 ビット符号なしの数値として処理します。演算 (+, − のみ) 結果についてはオーバーフローは考慮しません。従って、以下のような大小比較を行う場合にはオーバーフローが発生しないようにご注意ください (以下の例では work の内容が FF_{16} とすると演算結果は 00_{16} となり条件は成立しません) 。

   例)
   ```
   if [ work ] + 2 > 10
               :
   else
               :
   endif
   ```

2. 2 項演算子において *, /, %を使用する場合は付属のサブルーチンライブラリ (SRA74.A74) を **アセンブル後**リンクしてください。手順は次のとおりです。

   (a) SRA74.A74 をアセンブルし、リロケータブルファイル SRA74.R74 を生成します。

   (b) 演算子に対応したラベルを外部参照指定します。

   SRA74 は、乗除算 2 項演算子 *, /, %から SRA74.R74 に含まれるサブルーチンをコールする機械語を生成するので、次の表に示す演算子に対応したラベルの外部参照指定が必要となります。

演算子	ラベル	
*	.mult_8	乗算ルーチン
/	.div_8	除算ルーチン
%	.mod_8	余り算ルーチン

(指定例)

```
.ext    .mult_8, .div_8
```

(c) ユーザープログラムと SRA74.R74 をリンクします。

なお、スタック領域を 1 ページ (0100_{16}番地以降) にとるシステムにおいても使用可能ですが、この場合以下の定義、及びワーク領域の確保が必要となります。

- スタックページの設定 (.SPPAGE ラベルの定義) を行う。設定する値が 0 で 0 ページ、1 で 1 ページとなります (但し、これはアセンブラに対する指示を行うのみであり、これによりスタックが設定されるわけではありませんのでご注意ください) 。
  例) `.SPPAGE    .EQU    1`
  SRA74.A74 ではこのシンボルにより条件アセンブルを行っています。
- 演算用ワーク領域 (.syswk) を 0 ページ内 RAM 上に 3 バイト確保する。演算結果は乗算の場合.syswk と.syswk+1 に下位、上位の順で設定されます (演算結果の上位を参照したい場合は、ユーザー自身で参照してください) 。
  例) `.syswk:    .blkb    3`
- 演算用ワーク領域 (.syswk) を、割り込み処理ルーチンで使用する場合はご注意ください。この場合は、付属のライブラリソース (SRA74.A74) をユーザー自身で変更するか、演算ワーク領域のスタックへの退避などが必要となります。

3. 2 項論理演算子 `&&, ||` を使用することにより構造化命令 (`if, for, while`) の条件式を 6 つまで記述することができます。
   但し、論理演算子に優先順位はなく、先頭から順番に評価するように展開します。すなわち、次のように展開されます。

```
;IF BIT_A1 == 1 || BIT_A2 == 1 && BIT_A3 == 1 || BIT_A4 == 1
        BBS     BIT_A1,A,.I0
        BBC     BIT_A2,A,.I1
        BBS     BIT_A3,A,.I0
        BBC     BIT_A4,A,.I1
.I0:
;    X = ++X
            INX
;ENDIF
.I1:
```

第 6 章

疑似命令

6.1 疑似命令の機能

疑似命令は、命令が目的とする機械語データを生成するように SRA74 に対する指示[1] を行います。SRA74 は、48 個の疑似命令を用意していますが、これらは機能的に以下の 6 つのグループに分類できます。

1. アセンブル制御
   - 疑似命令自身はデータを生成しませんが、アセンブル処理の流れを制御します。
   - アドレスの更新には影響しません。
   - このグループは以下の 7 個の疑似命令を含みます。

.ASSERT	アセンブルアサーション宣言
.END	プログラム終了宣言
.ERROR	アセンブルエラー宣言
.IF (.ELSE) .ENDIF	条件付きアセンブル
.INCLUDE	ファイル読み込み

2. アドレス制御
   - データ設定疑似命令は定数データの生成を行います。
   - アドレスの更新を行います。
   - このグループは以下の 7 個の疑似命令を含みます。

.EQU (=)	同義定義
.ORG (*=)	アドレス指定
.BLKB	RAM 領域確保
.BYTE .WORD	データ設定

[1] 疑似命令の名称は、SRA74 に対する指示を行うものを “宣言” 、出力ファイルに影響するものを “指定” と呼んでいます。

3. リンク制御

   - リンク処理に関係する制御を行います。
   - このグループは以下の 18 個の疑似命令を含みます。

.BEXT .ZBEXT	外部参照指定 (ビットシンボル)
.EXT .SEXT .ZEXT	外部参照指定 (シンボル、ラベル)
.PUB	パブリック指定 (ビット、シンボル、ラベル)
.SECTION	ROM、及び RAM 領域指定
.SECTION P(.PMOD)	ROM 領域指定 (一般ページ)
.SECTION R(.RMOD)	RAM 領域指定 (一般ページ)
.SECTION S(.SMOD)	ROM 領域指定 (スペシャルページ)
.SECTION Z(.ZMOD)	RAM 領域指定 (ゼロページ)
.OBJ .LIB	リンクファイル名指定
.VER	バージョン指定

4. リスト制御

   - PRN ファイル出力に関係する制御を行います。
   - このグループは以下の 7 個の疑似命令を含みます。

.COL .LINE	リスト形式 (カラム数/行数) 指定
.LIST .NLIST	リスト出力/抑止指定
.LISTM .NLISTM	マクロ展開部のリスト出力/抑止指定
.PAGE	改ページ及びタイトル指定

5. デバッグサポート

   - ソースラインデバッグに関係する制御を行います。
   - このグループは以下の 2 個の疑似命令を含みます。

.FUNC .ENDFUNC	ファンクション指定

注意事項

(a) SRA74 において、“–c”を指定してアセンブルした場合は、“.FUNC”及び “.ENDFUNC”擬似命令行については評価を行いません。したがって、これらの擬似命令についてのエラーは検出されません。

6. 予約疑似命令

   - 将来の拡張のための予約疑似命令です。これらの疑似命令はアセンブル処理には影響を与えません。
   - このグループは以下の 9 個の疑似命令を含みます。

.PROGNAME	プログラム名宣言
.IO .ENDIO .RAM .ENDRAM	領域名宣言
.PROCMAIN .PROCSUB .PROCINT	モジュール名宣言
.ENDPROC	モジュール終了宣言

以下に各グループ別に疑似命令の機能を説明します。

6.2 アセンブル制御

1. アセンブルアサーション宣言

   .ASSERT
   アセンブル作業中にオペランドに記述した文字列を画面に表示します。

2. アセンブル終了宣言

   .END
   ソースプログラムの終りを宣言します。SRA74 は、この行以降の内容を処理しません。

3. アセンブルエラー宣言

   .ERROR
   この疑似命令の、オペランドに記述した文字列を画面に表示しアセンブル作業を終了します。

4. 条件付きアセンブル

   .IF (.ELSE) .ENDIF
   シンボル値の内容により、アセンブルを行う場所を指定します。複数仕様に対応するプログラムを 1 つのソースプログラムで管理する場合、テストルーチンのアセンブルを制御する場合などに使用できます。

5. ファイル読み込み

   .INCLUDE
   この命令を記述した場所に、他のファイル内容を読み込みます。大きなソースプログラムを分割して編集する場合に使用できます。

6.3 アドレス制御

1. 同義定義

   .EQU 、又は =
   シンボルに絶対値を定義します。この疑似命令は、シンボルに 0 〜 7 のビット値と 0000_{16}〜 $FFFF_{16}$の値を定義します。

2. アドレス宣言

   .ORG 、又は *=
   以降の行のアドレスを宣言します。この疑似命令を記述したセクションは絶対属性となり、リンク時にアドレス指定を行うことはできません。割り込みベクタなど、アドレスが固定している領域に使用できます。

3. 領域確保

.BLKB
オペランドで指定した容量のメモリ領域を RAM 領域に確保します。

4. データ設定

.BYTE .WORD
オペランドで指定したデータを ROM 領域に生成します。

6.4 リンク制御

1. グローバルラベル指定

.BEXT .ZBEXT
外部参照するビットシンボル名を指定します。なお、ここで指定したビットシンボル名は必ず他のファイルでパブリック指定されていなければなりません。

.EXT .SEXT .ZEXT
外部参照するラベル、及びシンボル名を指定します。なお、ここで指定したラベル名は必ず他のファイルでパブリック指定されていなければなりません。

.PUB
このファイル中で定義しているラベル、シンボル、又はビットシンボルを他のファイルから参照できるよう指定します。

2. 領域指定

.SECTION
この行以降が、オペランドで指定した名前を持つ領域であることを指定します。又、領域属性はこの命令以降の命令により SRA74 が自動的に決定します。この指定は、他の領域指定命令が現れるまで有効です。

.SECTION P 、又は.PMOD
この行以降が、一般ページの ROM 領域であることを指定します。この指定は、他の領域指定命令が現れるまで有効です。

.SECTION R 、又は.RMOD
この行以降が、一般ページの RAM 領域であることを指定します。この指定は、他の領域指定命令が現れるまで有効です。

.SECTION S 、又は.SMOD
この行以降が、スペシャルページの ROM 領域であることを指定します。この指定は、他の領域指定命令が現れるまで有効です。

.SECTION Z 、又は.ZMOD

この行以降が、ゼロページの RAM 領域であることを指定します。この指定は、他の領域指定命令が現れるまで有効です。

3. リンクファイル名指定

.OBJ .LIB

リンク対象の R74 ファイル名、及びライブラリファイル名を指定します。ここで宣言したファイルは、LINK74(リンカ) が自動的に参照するため、リンク時のコマンド入力を簡略化できます。

4. バージョン指定

.VER

R74 ファイルのバージョンを指定します。LINK74 のコマンドパラメータ “–V” を指定すると R74 ファイル間のバージョンの一致を確認できます。

6.5 リスト制御

1. 改ページ及びタイトル指定

.PAGE

リストの改ページ及びタイトルを指定します。

2. リスト形式指定

.COL .LINE

リストのカラム数、行数を指定します。これらの疑似命令は、ソースファイル中に 1 回だけ記述できます。

3. リスト出力/抑止指定

.LIST .NLIST

PRN ファイルへのリスト出力制御を行います。プログラムの一部をデバッグする場合などで、リストの一部のみが必要なときに使用してください。

4. マクロ展開部のリスト出力/抑止指定

.LISTM .NLISTM

PRN ファイルへのマクロ展開部のリスト出力制御を行います。

6.6 デバッグサポート

SRA74 では、ソースレベルデバッグ情報はコマンドパラメータ “–C” を指定した場合に生成されますが、.FUNC 〜 .ENDFUNC で囲まれた部分のみソースラインデバッグ情報を出力させることも可能です。必要な部分のデバッグ情報のみに限定することにより、デバッグ時のホストマシンのメモリの消費を減らすことができます。また、 .FUNC 疑似命令を使用したソースファイルに対してもコマンドパラメータ “–C” を指定することにより、ファイル全体に対するソースラインデバッグ情報を生成することができます。

1. ファンクション開始指定

   .FUNC
   ファンクションの始まりを指定します。

2. ファンクション終了指定

   .ENDFUNC
   ファンクションの終わりを指定します。

注意事項

1. SRA74 では、“–C” を指定してアセンブルを行った場合、 “.FUNC” 及び、“.ENDFUNC” 擬似命令行については評価を行いません。したがって、これらの擬似命令行に関するエラーは検出されませんのでご注意ください。

6.7 予約疑似命令

予約疑似命令は、SRA74 が将来の拡張のために予約している疑似命令です。これらの疑似命令は、ソースファイルに記述してもエラーにはなりません。又、アセンブル結果にも影響しません。一般的には、市販の文字列検索プログラムと組み合わせてソースファイルの内容を確認する場合などに使用できます。

ソースファイル中に以下のように疑似命令.PROCMAIN を記述しておけば、文字列検索プログラムで “.PROCMAIN” を検索することにより、メインプログラムの検索が行えます。

例） `.PROCMAIN  KEY_SCAN  ; キースキャンプログラム部エントリ`

第 7 章

マクロ命令

7.1　マクロ命令の機能

740 ファミリ のアセンブリ言語、及び構造化記述言語を使用したプログラムをマクロとして定義しておくことにより、ユーザーはその際につけた名前 (マクロ名) をソースプログラムのオペランド欄にオペコードや構造化命令と同様に記述することができます。従って、いろいろなマクロ定義を用意しておけば、プログラミングの際に 740 ファミリを拡張した新しい CPU として利用することができます。このようにマクロ機能は、ユーザーが自分自身のプログラミング環境を整えることができる機能を提供します。

7.2　マクロ命令の種類

マクロ命令は、SRA74 に組み込まれているマクロとユーザーが定義するマクロの 2 種類に分類できます。

1. システムマクロ命令

    - .REPEATI 〜.ENDM
      オペランドに記述された引数の個数回、繰り返し処理を行います。
    - .REPEATC 〜.ENDM
      オペランドで引数として与えた文字数回、繰り返し処理を行います。
    - .REPEAT 〜.ENDM
      指定した回数だけ繰り返し処理を行います。
    - .EXITM
      マクロ展開を強制的に打ち切ります。

2. ユーザーマクロ命令

    - .MACRO 〜.ENDM
      マクロ命令に対する定義を行います。

- .EXITM
  マクロ展開を強制的に打ち切ります。
- .LOCAL
  マクロ定義のなかで使用されているラベルをマクロ内ローカルラベルにします。

詳細は付録 Eを参照してください。

注意事項

1. ユーザーマクロを使用する場合は、前もってマクロ定義をしておかなければなりません。このためマクロ定義は、通常プログラムの最初に定義するか、先頭でマクロ定義のファイルを.INCLUDE 疑似命令で取り込んでおきます。なお、ユーザーマクロ定義のネストは 20 レベルまでです（但し、これはホストマシンのメモリ容量に依存します)。
2. マクロ定義のファイルを、別のファイル (マクロライブラリ) としておけば、プログラムの初めにインクルードするだけでマクロが使用できるようになり、各プログラムごとにマクロ定義を記述する必要がなくなります。
3. マクロ展開された部分は、プリントファイル中ソースの横に'+' で示されます。
4. LOCAL 宣言されたラベルは、その順に..n(n は 10 進数で..1 〜..65535) というラベルを割り当ててアセンブルします。
5. ピリオド'.' で始まる名前をマクロ名やラベル名などに使用しないでください。
6. マクロ呼び出し時のマクロ引数に構造化記述のメモリ参照を示す [ ] 、および{ }を用いる場合は、必ず"(ダブルクォーテーション) で囲ってください。

[例]

```
mac:  .macro  para1,para2
      para1 = para2
      .endm
mac "[work]",10h
```

7.3 マクロ演算子

表 7.1に、マクロ命令で使用可能な演算子の一覧表を示します。

表 7.1: マクロ演算子一覧表

演算子	内容
¥ [1]	マクロの引数として使用できない特殊文字の前において‘¥’の後の文字を引数として認識させます。 [書式] ¥文字
;; [2]	マクロ定義において展開を行わないコメントを定義します。 [書式] ;; コメント
" [3]	マクロの呼び出し時に、引数中にスペース、タブ、カンマ (,) 、及び予約語などが含まれる場合に用います。 [書式] ” 文字列”
$ [4]	引数の前後に文字列を連結する場合に使用します。 [書式](1) 引数 $ 文字列　(2) 文字列 $ 引数

注意事項

1. エスケープキャラクタの働きをして次の文字の特別な意味を打ち消します。

例）

[マクロ定義]

```
DATA:   .MACRO   VAL
        .BYTE    VAL
        .ENDM
```

[呼び出し例]

```
        DATA    ”¥”HELLO !¥””
```

[マクロ展開]

```
        .BYTE    "HELLO !"
        .ENDM
```

2. マクロの展開結果をプリントファイルに出力する場合、セミコロン 1 個 (';') で開始するコメントについてはマクロ展開されるたびに出力されますが、セミコロン 2 個 (";;") で始まるコメントは出力されません。

例)

[マクロ定義]

```
LOOP:   .MACRO
        .LOCAL  LOOP1
        LDA     #20             ;  comment
LOOP1:  DEC     A               ;; comment
        BNE     LOOP1           ;; comment
        .ENDM
```

[呼び出し例]

```
        LOOP
```

[マクロ展開]

```
        LDA     #20             ;  comment
..0:    DEC     A
        BNE     ..0
        .ENDM
```

3. オペランドで引数として与えた中にスペース、タブ、カンマ (,) 、予約語などが含まれる場合は全体をダブルクォート (") で囲んで表します。

例)

[ソース記述]

```
SUB:    .REPEATI INST,"NOP","LDA #1","JSR SUB1","RTS"
       INST
        .ENDM
```

[マクロ展開]

```
SUB:
        NOP
        LDA #1
        JSR SUB1
        RTS
        .ENDM
```

4. 引数の前後に文字列を連結し、引数で与える名前を変更する場合に使用します。'$' と文字列の間には、スペース又はタブを挿入しないでください。

例)

[ソース記述]

```
ADDWI:  .MACRO  MEM,IMM
        CLC
        LDA     MEM$_2
        ADC     #>IMM
        STA     MEM$_1
        LDA     #<IMM
        ADC     MEM$_2+1
        STA     MEM$_1+1
        .ENDM
```

[呼び出し例]

```
        ADDWI   RAM,1000H
```

[マクロ展開]

```
        CLC
        LDA     RAM_2
        ADC     #>1000H
        STA     RAM_1
        LDA     #<1000H
        ADC     RAM_2+1
        STA     RAM_1+1
```

5. '(　)' で囲まれた文字列は、1つの引数として扱われます。

例)

[マクロ定義]

```
ADDI:   .MACRO  MEM,IMM
        LDA     MEM
        CLC
        ADC     #IMM
        STA     MEM
        .ENDM
```

[呼び出し例]

```
        ADDI    ( RAM,X ),5
```

[マクロ展開]

```
        LDA     ( RAM,X )
        CLC
        ADC     #5
        STA     ( RAM,X )
```

第 8 章

操作方法

8.1 起動方法

SRA74 を実行するするためには、以下の情報 (入力パラメータ) を入力する必要があります。

1. ソースファイル名 (必須項目)
2. コマンドパラメータ

SRA74 では、これらの情報をオペレーティングシステムのコマンド行から指定します。また、環境変数 SRA74 で定義することもできます。

8.2節で入力パラメータについて説明します。また、例を示しながら、 8.3節でコマンド行入力について、 8.5節で環境変数 SRA74 の定義について説明します。

8.2 入力パラメータ

8.2.1 ソースファイル名

1. アセンブル対象のソースファイル名を指定します。指定できる個数は 1 個です。
2. ファイル属性 (.A74) を省略した場合、既定値として属性.A74 を選びます。
3. ファイル名をフルネームで指定することにより、.A74 以外の属性 (例 .ASM) のファイルもアセンブル可能です。
4. ファイル名にはディレクトリパス指定が可能です。ファイル名のみを指定した場合は、カレントドライブのカレントディレクトリ中のファイルを処理します。
   例) `A>SRA74 C:¥WORK¥TEST<RET>`

8.2.2 コマンドパラメータ

1. コマンドパラメータはマイナス記号 (–) 及びそれに続く文字により 構成されます。
2. 文字は大文字/小文字どちらでも有効です。

3. 各パラメータは同時に複数指定することができます。この場合は、各パラメータをスペースで区切って入力してください。

4. マイナス記号を 2 つ続けることによりそのコマンドパラメータを無効にすることができます。例えば --L とすると PRN ファイルの生成が無効となります。

5. コマンドパラメータは環境変数 SRA74 から先に処理されます。

表 8.1に、コマンドパラメータの内容を示します。

表 8.1: コマンドパラメータ一覧表

コマンドパラメータ	内容
−.	画面へのエラーメッセージ以外のメッセージ出力を抑止します。バッチファイルなどで SRA74 を実行する場合に、画面にエラーメッセージのみ表示したいときにお使いください。
−!8	!演算子の演算に対して、!演算を行った後、下位 8 ビットのみを切り出した値を演算結果とします。
−A	プログラムがアセンブリ言語のみで記述されていることを指定します。[1]
−BANK	アドレス空間の上限を FFFFH から 1FFFFH に拡張します。演算子 BK および BL が有効になります。セクション E の情報をリロケータブルファイル、リストファイルに出力しません。
−C	ファイル中の全ての行に対してソースラインデバッグ情報を出力します。
−D	シンボルに数値を設定します。コマンドの機能は、疑似命令.EQU と同等です。指定の書式は、次のようになります (複数のシンボルを同時に定義する場合、':' で区切ってください) 。 –D シンボル=数値 [:シンボル=数値....:シンボル=数値] 例) A>SRA74 SRCFILE -DS1=10:S2=20<RET>
−E	TAG ファイルの生成及びエディタの起動を行います。エディタのプログラム名指定の書式は、次のようになります。[2] −E[エディタ名] 例) A>SRA74 SRCFILE -EMI<RET> [ ] の部分は省略可能です。省略した場合、TAG ファイルの生成のみを行います。エディタ名が指定された場合はアセンブル終了後、TAG ファイルを引き数としてエディタを起動します。但し、エディタ名を指定してもエラーが発生しなかった場合は、エディタの起動は行いません。

コマンドパラメータ	内容
–I	構造化記述言語命令をアセンブリ言語命令に展開したソースファイル (.I) を生成します。また、このパラメータが–L と同時に指定された場合、生成されるプリントファイル中にも展開部分が出力されます。
–K	SRA74 の生成したラベル (".I0" 等) の情報を R74 ファイルへ出力しないようにします。これにより –S オプションで出力されるローカルラベルはユーザーの定義したもののみとなります。
–L	PRN ファイルを生成します。この指定がない場合 PRN ファイルは生成しません。
–LS	シンボルリストの付いた PRN ファイルを生成します。
–M	マクロ展開の結果を PRN ファイル中に出力します。この指定がない場合、マクロ展開の結果はリスト中には出力されません。
–O	生成ファイルの出力先パスを指定します。パスにはディレクトリ又はドライブ名が指定できます。この指定がない場合、ソースファイルと同じパスに出力します。指定の書式は、次のようになります。 –O パス名 例) `A>SRA74 SRCFILE -OB:¥USR<RET>`
–S	R74 ファイルにローカルなビットシンボル、シンボル、及びラベルを出力します。
–U	ラベル記述の ':'(コロン) を無視します。この指定がない場合、プログラムでラベルを記述した場合に ':' がないとエラーとなります。
–X	アセンブル終了後、クロスリファレンサ CRF74 を起動します。[3] 例) `A>SRA74 SRCFILE -X<RET>`

注意事項

1. この指定を行った場合 SRA74 はアセンブル時、一時ファイルを作成しません。従って、アセンブル処理時間が短縮されます。
   また、I ファイルをソースファイルとして入力する場合にも、このコマンドパラメータを指定してください。

2. カレントディレクトリ又はコマンドパス中に CRF74 が存在しない場合は、システムエラーになります。

8.3 入力方法

SRA74 は、オペレーティングシステムのプロンプト状態でコマンド行を入力することにより起動します。図 8.1に、起動コマンドの入力例を示します。

コマンド行入力に誤りを検出すると、図 8.2のようにヘルプ画面を表示しアセンブルを中止します。マンド行入力が正常に行われるとアセンブルを開始します。アセンブルが終了するとエラー数、ワーニング数、全行数、コメント行数、セクション単位のメモリ容量を画面に出力します。アセンブルが正常に終了したときの画面表示例を図 8.3に示します。

```
A>SRA74 FILENAME -L -E <RET>
          ↓        ↓
アセンブル対象の   コマンドパラメータ
ソースファイル名
```

図 8.1: 起動コマンド入力例

```
A>SRA74<RET>
740 Family SRA74 V.4.00.00
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Usage: sra74 <filename> [ options ]
       -.    : all messages suppressed
       -a    : assemble assembly language source file
       -c    : source line data output to .R74 file
       -d    : define symbol ( syntax -ds1=1:s2=2 )
       -e    : make tag file
       -i    : make assembler source file
       -k    : suppress system label information to R74 file
       -l[s] : make list file or symbol list file
       -m    : macro listing
       -o    : select drive and directory for output ( syntax -otmp )
       -q    : .EQU symbol multi define warning message output
       -s    : local symbol data output to .R74 file
       -u    : don't care ':' at end of label
       -x    : execute crf74
A>
```

図 8.2: コマンドエラー時のヘルプ画面

```
A>SRA74 TEST<RET>
740 Family SRA74 V.4.00.00
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

now processing pass 1  ( TEST.A74 )
----*----
now processing pass 2  ( TEST.A74 )
----*----

ERROR   COUNT           00000
WARNING COUNT           00000
STRUCTURED STATEMENT    00702 LINES
TOTAL   LINE ( SOURCE ) 00994 LINES
TOTAL   LINE ( OBJECT ) 00994 LINES
COMMENT LINE ( SOURCE ) 00247 LINES
COMMENT LINE ( OBJECT ) 00147 LINES
OBJECT  SIZE ( Z      ) 00010 (000A) BYTES
OBJECT  SIZE ( R      ) 00053 (0035) BYTES
OBJECT  SIZE ( P      ) 01938 (0792) BYTES

A>
```

図 8.3: 正常終了時の画面表示

8.4 エラー

8.4.1 エラーの種類

SRA74 実行時に発生するエラーは、以下の原因によるものがあります。

1. オペレーティングシステムに関するエラー
   ディスクやメモリ容量の不足など、SRA74 を実行するオペレーティングシステム環境に関わるエラーです。付録 A エラーメッセージ一覧表を参照の上、オペレーティングシステムのコマンドにより対応してください。

2. SRA74 のコマンド行入力に関するエラー
   SRA74 起動時のコマンド行入力に関わるエラーです。本章の内容を確認の上コマンドを再入力してください。

3. アセンブル対象のソースファイルの内容に関するエラー
   ラベルの 2 重定義、未定義シンボルの参照などのソースファイルの内容に関わるエラーです。該当箇所のソースファイル内容を修正して再度アセンブルを行ってください。
   アセンブルエラーを検出した場合 R74 ファイルは生成しません。

SRA74 はエラー及びワーニングを検出すると、図 8.4の形式でエラー内容 (ファイル名、ファイル中の行番号、通し行番号、エラー番号及びエラーメッセージ) を画面と PRN ファイルに出力します。付録 A のエラー番号順のエラー一覧表を参照の上対応してください。

```
A>SRA74 TEST<RET>
740 Family SRA74 V.4.00.00
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

now processing pass 1  ( TEST.A74 )
----*----
now processing pass 2  ( TEST.A74 )
-
  115 E025  EAEA        BCC     LOOP2
TEST.A74 115 ( TOTAL LINE 115 ) Error 18: Relative jump is out of range
  127 E031  EAEA        LDA     #data
TEST.A74 127 ( TOTAL LINE 127 ) Error 22: Value is out of range "data"
---*
  551 E42B  EAEA        BRA     TEST2
TEST.A74 551 ( TOTAL LINE 551 ) Error 18: Relative jump is out of range
  593 E4FC  EAEAEA      LDA     (work,x                 ; data set
TEST.A74 593 ( TOTAL LINE 593 ) Error 23: "()" format error ";"
----

ERROR   COUNT           00004
WARNING COUNT           00000
STRUCTURED STATEMENT    00702 LINES
TOTAL   LINE ( SOURCE ) 00994 LINES
TOTAL   LINE ( OBJECT ) 00994 LINES
COMMENT LINE ( SOURCE ) 00247 LINES
COMMENT LINE ( OBJECT ) 00147 LINES
OBJECT  SIZE ( Z      ) 00010 (000A) BYTES
OBJECT  SIZE ( R      ) 00053 (0035) BYTES
OBJECT  SIZE ( P      ) 01938 (0792) BYTES

A>
```

図 8.4: エラー表示例

8.4.2 オペレーティングシステムへの戻り値

オペレーティングシステムのバッチファイルなどに実行コマンドを記述する場合、実行結果に応じて処理の内容を変えたい場合があります。SRA74 では、実行結果を表 8.2の 4 つのエラーレベルに分けてオペレーティングシステムに返すようにしています。

表 8.2: エラーレベル一覧表

エラーレベル	実行結果の内容
0	正常終了
1	アセンブル対象のソースファイルの内容に関するエラー
2	SRA74 のコマンド入力に関するエラー
3	オペレーティングシステムに関するエラー
4	^C（コントロールC）入力による強制終了

8.5 環境変数

SRA74 は、以下の環境変数を使用しています。

- TMP
  アセンブルするときに生成するテンポラリファイルの作成ディレクトリ名を指定します。この環境変数が設定されていない場合、テンポラリファイルはソースファイルと同じディレクトリ内に生成されます。環境変数の設定例を以下に示します。

  例) `set TMP=A:¥TMP`

- INC
  アセンブルするときにインクルードするファイルのディレクトリ名を指定します。この環境変数が設定されていない場合、.INCLUDE 疑似命令により読み込むファイルはカレントディレクトリより検索されます。環境変数の設定例を以下に示します。

  例) `set INC=A:¥INC74`

- SRA74
  環境変数 SRA74 を用いて、ファイル名及びコマンドパラメータをコマンド行以外に指定することができます。環境変数の設定例を以下に示します。

  例) `SET SRA74=-L -I`

  したがって、頻繁に使用するコマンドパラメータを定義しておくことにより、アセンブルごとのコマンド行への入力の必要がなくなります。また、環境変数 SRA74 に定義されているコマンドパラメータを使用したくない場合は、コマンド行からそれを無効にすることができます。
  環境変数 SRA74 に定義した情報は、コマンド行による指定よりも先に処理されます。例えば、以下の書式により指定された場合、(PC)

```
SET  SRA74=-L  -I
SRA74  FILE  --L  -S
```

以下の指定をコマンド行より入力した場合と同じになります。

```
SRA74  -L  -I  FILE  --L  -S
```

すなわち、コマンド行からの指定によりコマンドパラメータ “–L” は無効となり、PRN ファイルは生成されないことになります。

付録 A

エラーメッセージ一覧表

A.1 システムエラー一覧表

アセンブル中にシステムエラーを検出すると、エラーメッセージを画面に表示しアセンブルを中止します。表 A.1にシステムエラー一覧表を示します。

表 A.1: システムエラー一覧表

エラーメッセージ	エラー内容と対策
Usage: sra74 <filename> [options]	コマンド入力が誤っています。 $\Longrightarrow$ ヘルプ画面を参照して、コマンドを再入力してください。
Can't open xxx [1]	該当ファイルが見つかりません。 $\Longrightarrow$ ソースファイル名を確認して、再入力してください。
Can't create xxx	該当ファイルが生成できません。 $\Longrightarrow$ コマンドパラメータ "–O" の指定を確認して、再入力してください。
Out of disk space	ファイルを出力するためのディスク容量が不足しています。 $\Longrightarrow$ ディスク上に空き領域を作ってください。
Out of heap space	アセンブラが動作するために必要なメモリが不足しています。[2] $\Longrightarrow$ シンボル又はラベルの数を減らすか、ファイルを分割してください。
Can't find crf74.exe	CRF74 が見つかりません。 $\Longrightarrow$ CRF74 をカレントディレクトリ又はオペレーティングシステムのコマンドパス内のディレクトリにコピーしてください。

エラーメッセージ	エラー内容と対策
Can't find command.com for execute xxx	コマンドパラメータ "–E" で指定されたエディタを起動するために必要なファイル COMMAND.COM が見つかりません。 ⟹ オペレーティングシステムのコマンドパス指定を確認してください。

注意事項

1. SRA74 で扱えるシンボル及びラベルの総数は、SRA74 を実行するシステムでの使用可能メモリ容量に依存しています。

A.2 アセンブルエラー一覧表

アセンブルエラーを検出すると、エラーメッセージを画面と PRN ファイルに出力します。表 A.2にアセンブルエラーとその内容を示します。

表 A.2: アセンブルエラー一覧表

エラー番号	エラーメッセージ	エラー内容と対策

1	Already had same statement	ソースファイル中に 1 回しか使用できない疑似命令を、2 回以上使用しています。
		例) `.LINE 60`
		`:`
		`.LINE 80`
		⟹ 宣言を 1 回にしてください。

2	Reference to forward label or symbol	疑似命令が前方のラベル又はシンボルを参照しています。 例) `.ORG TOP` `TOP:` ⟹ ラベル又はシンボルの定義を参照以前に行ってください。
3	Division by 0	数式中に 0 による割り算を含んでいます。 ⟹ 数式の記述を確認してください。

エラー番号	エラーメッセージ	エラー内容と対策
4	Illegal operand	オペランドに使用できない文字を含んでいます。 例) `LDA #&10` $\Longrightarrow$ オペランドの記述を確認してください。
5	Improper operand type	ニーモニックとオペランドの組み合わせが誤っています。 例) `ASL work,Y` $\Longrightarrow$ 命令記述形式を確認してください。
6	Invalid label definition	ラベル定義ができない場所でラベル定義を行っています。 例 1) `LABEL1: .LINE 60` $\Longrightarrow$ ラベルを削除してください。 例 2) `LABEL2: .EQU 100` $\Longrightarrow$ ラベルをシンボルに変更してください。
7	Invalid symbol definition	シンボル定義ができない場所でシンボル定義を行っています。 例) `SYMBOL .LINE 60` $\Longrightarrow$ シンボルを削除してください。
8	Out of maximum program size	アドレスが $0FFFF_{16}$ を超えています。 例) `.ORG 0FFF0H` `.WORD 1,2,3,4,5,6,7,8,9` $\Longrightarrow$ アドレスが範囲内になるようにプログラムを変更してください。
9	Label or symbol is multiple defined	同一のラベル又はシンボルが 2 回以上定義されています。 例) `MAIN: NOP` `MAIN: NOP` $\Longrightarrow$ ラベル名又はシンボル名を確認してください。
10	Nesting error	ネスティングレベルの限界を越えて記述されています。 $\Longrightarrow$ ネスティングレベルの限界を越えないように記述してください。

エラー番号	エラーメッセージ	エラー内容と対策
11	No .END statement	ソースファイル内に.END 文がありません。 ⟹ プログラムの最後に.END 文を記述してください。
12	No symbol definition	シンボルが記述されていません。 例)　　.EQU　　60 ⟹ シンボルを記述してください。
13	No ';' at the top of comment	コメント欄の先頭に ';'(セミコロン) がありません。 例)　　LDA　　#CNT counter set ⟹ コメント欄の先頭に ';' を付加してください。
14	Not in conditional block	対応する.IF 文がないのに、.ELSE 又は.ENDIF が記述されています。 (対応する.IF 文がエラーになっている場合もこのエラーが発生します。) 例)　　.IF DATA1 : .ENDIF : .ELSE : .ENDIF ⟹.IF 文を確認してください。
15	Operand is expected	命令に必要なオペランドが不足しています。 例)　　.BYTE ⟹ オペランドの記述を確認してください。
16	Questionable syntax	ニーモニックのスペルに誤りがあります。 例)　　ADD　　#DATA ⟹ ニーモニックのスペルを確認してください。
17	Reference to multi defined label or symbol	重複定義されているラベル又はシンボルを参照しています。 例)　MAIN:　NOP MAIN:　NOP BRA　　MAIN ⟹ ラベル名又はシンボル名を確認してください。

エラー番号	エラーメッセージ	エラー内容と対策
18	Relative jump is out of range	相対ジャンプ命令のジャンプ先アドレスが範囲内にありません。 $\Longrightarrow$ プログラムの再配置を行うか、命令を変更してください。
19	Label or symbol is reserved word	ラベル又はシンボルに予約語と同一名称が使われています。 例) `A .EQU 1FFH` $\Longrightarrow$ ラベル名又はシンボル名を変更してください。
20	Reference to undefined label or symbol	未定義のラベル又はシンボルを参照しています。 $\Longrightarrow$ ラベル又はシンボルを確認してください。
21	Value error	データ記述形式に誤りがあります。 例) `ADC #'A` $\Longrightarrow$ データ記述形式を確認してください。
22	Value is out of range	データの範囲が許容値を越えています。 例) `ADC #100H` $\Longrightarrow$ オペランド記述形式を確認してください。
23	"()" format error	左カッコ '(' と右カッコ ')' の数があっていません。 例) `ADC (WORK` $\Longrightarrow$ オペランド記述形式を確認してください。

24	Relocatable error	リロケータブルセクションの中で疑似命令 .ORG を記述しています。
		例)
		`.SECTION PROG`
		`LDA WORK`
		`:`
		`.ORG 1000H`
		`:`
		$\Longrightarrow$ セクションを分けてください。

25	No .SECTION statement	疑似命令.SECTION を記述していません。 $\Longrightarrow$ プログラム記述の前に、疑似命令.SECTION を記述してください。

エラー番号	エラーメッセージ	エラー内容と対策
26	Reference to undefined section	未定義のセクション名を参照しています。 例) `LDA #SIZEOF UNDEF` ⟹ 該当セクションを確認してください。
27	Section type mismatch	命令又はデータ設定疑似命令 (.BYTE など) が、領域確保命令 (.BLKB など) と混在しています。 例) `LDA #WORK` `.BLKB 1` ⟹ セクションを分けてください。
28	Constant value is required	相対属性のラベル及び外部参照シンボルを用いることはできません。 ⟹ 絶対値を指定してください。
30	else not associated with if	else に対応する if がありません。 ⟹ プログラムを確認してください。
31	endif not associated with if	endif に対応する if がありません。 ⟹ プログラムを確認してください。
32	next not associated with for	next に対応する for がありません。 ⟹ プログラムを確認してください。
33	while not associated with do	while に対応する do がありません。 ⟹ プログラムを確認してください。
34	break not inside for, do or switch	break の使用箇所が不適当です。 ⟹ プログラムを確認してください。
35	case not inside switch	case が switch の範囲外です。 ⟹ プログラムを確認してください。
36	deplicate case value	case の値として同一値が重複して使用されています。 ⟹ プログラムを確認してください。
37	more than one default	同一 switch 文において default が 2 つ以上あります。 ⟹ プログラムを確認してください。
38	default not inside switch	default が switch 文の範囲外です。 ⟹ プログラムを確認してください。
39	ends not associated with switch	ends に対応する switch がありません。 ⟹ プログラムを確認してください。
40	continue not inside for or do	continue の使用箇所が不適当です。 ⟹ プログラムを確認してください。

A.3 ワーニング一覧表

ワーニングを検出すると、ワーニングメッセージを画面とPRNファイルに出力します。表A.2にワーニングとその内容を示します。

表 A.3: ワーニング一覧表

ワーニング番号	ワーニングメッセージ	ワーニング内容と対策

1	Phase warning	1) 疑似命令.ORGで指定したアドレスが、それ以前のアドレスよりも前になっています。
		例)
		`.ORG 0E000H`
		`MAIN: LDA WORK`
		`:`
		`.ORG 0C000H`

		2) 命令がこの行より前方で定義されているラベル又はシンボルを参照しています。
		例)
		`LDA WK,X`
		`:`
		`WK .EQU 80H`
		$\Longrightarrow$ ラベル又はシンボルを参照する行より前で定義してください。

2	.END statement in include file	疑似命令.ENDをインクルードファイル中に記述しています。 $\Longrightarrow$ .ENDをソースファイル中に記述してください。
3	statement has no effect	ステートメントとして意味を持ちません。 $\Longrightarrow$ プログラムを確認してください。
4	not case values for switch statement	switch文中にcaseが一つもありません。 $\Longrightarrow$ プログラムを確認してください。
5	statement not preceded by case or default	switch文において、case, defaultよりもステートメントが先にきています。 $\Longrightarrow$ プログラムを確認してください。
6	.EQU symbol is multiple defined	疑似命令 .EQUで同一シンボルを2回以上定義しています。
7	'.SECTION E' requires command option '-BANK'	セクションEが記述されています。コマンドオプション-BANKを指定してアセンブルしてください。

付録 B

命令一覧表

B.1　記号表

命令一覧表で使用している記号の意味を、表 B.1に示します。

表 B.1: 記号表

記号	内容	記号	内容
A	アキュムレータ	X	インデックスレジスタ X
Y	インデックスレジスタ Y	imm	イミディエイトデータ
zz	ゼロページアドレス	hhll	一般ページアドレス
i	ビット値 (0 〜 7)	bitsym	ビットシンボル
#	イミディエイトモード	¥	スペシャルページモード

B.2　命令一覧表

SRA74 で使用可能な全命令を表 B.2に示します。各命令の横に、アドレッシングモード名、記述形式を示します。

表 B.2: 命令一覧表

命令名	アドレッシングモード名	記述形式
ADC	イミディエイト	ADC #imm
	ゼロページ	ADC zz
	ゼロページ X	ADC zz,X
	ゼロページ・インダイレクト X	ADC (zz,X)
	ゼロページ・インダイレクト Y	ADC (zz),Y
	アブソリュート	ADC hhll
	アブソリュート X	ADC hhll,X
	アブソリュート Y	ADC hhll,Y
AND	イミディエイト	AND #imm
	ゼロページ	AND zz
	ゼロページ X	AND zz,X
	ゼロページ・インダイレクト X	AND (zz,X)
	ゼロページ・インダイレクト Y	AND (zz),Y
	アブソリュート	AND hhll
	アブソリュート X	AND hhll,X
	アブソリュート Y	AND hhll,Y
ASL	アキュムレータ	ASL A
	ゼロページ	ASL zz
	ゼロページ X	ASL zz,X
	アブソリュート	ASL hhll
	アブソリュート X	ASL hhll,X
BBC	アキュムレータ・ビット・レラティブ	BBC i,A,hhll BBC bitsym,A,hhll
	ゼロページ・ビット・レラティブ	BBC i,zz,hhll BBC bitsym,hhll
BBS	アキュムレータ・ビット・レラティブ	BBS i,A,hhll BBS bitsym,A,hhll
	ゼロページ・ビット・レラティブ	BBS i,zz,hhll BBS bitsym,hhll

命令名	アドレッシングモード名	記述形式
BCC	レラティブ	BCC hhll
BCS	レラティブ	BCS hhll
BEQ	レラティブ	BEQ hhll
BIT	ゼロページ	BIT zz
	アブソリュート	BIT hhll
BMI	レラティブ	BMI hhll
BNE	レラティブ	BNE hhll
BPL	レラティブ	BPL hhll
BRA	レラティブ	BRA hhll
BRK	インプライド	BRK
BVC	レラティブ	BVC hhll
BVS	レラティブ	BVS hhll
CLB	アキュムレータ・ビット	CLB i,A CLB bitsym,A
	ゼロページ・ビット	CLB i,zz CLB bitsym
CLC	インプライド	CLC
CLD	インプライド	CLD
CLI	インプライド	CLI
CLT	インプライド	CLT
CLV	インプライド	CLV
CMP	イミディエイト	CMP #imm
	ゼロページ	CMP zz
	ゼロページ X	CMP zz,X
	ゼロページ・インダイレクト X	CMP (zz,X)
	ゼロページ・インダイレクト Y	CMP (zz),Y
	アブソリュート	CMP hhll
	アブソリュート X	CMP hhll,X
	アブソリュート Y	CMP hhll,Y
COM	ゼロページ	COM zz

命令名	アドレッシングモード名	記述形式
CPX	イミディエイト	CPX #imm
	ゼロページ	CPX zz
	アブソリュート	CPX hhll
CPY	イミディエイト	CPY #imm
	ゼロページ	CPY zz
	アブソリュート	CPY hhll
DEC	アキュムレータ	DEC A
	ゼロページ	DEC zz
	ゼロページ X	DEC zz,X
	アブソリュート	DEC hhll
	アブソリュート X	DEC hhll,X
DEX	インプライド	DEX
DEY	インプライド	DEY
DIV	ゼロページ X	DIV zz,X
EOR	イミディエイト	EOR #imm
	ゼロページ	EOR zz
	ゼロページ X	EOR zz,X
	ゼロページ・インダイレクト X	EOR (zz,X)
	ゼロページ・インダイレクト Y	EOR (zz),Y
	アブソリュート	EOR hhll
	アブソリュート X	EOR hhll,X
	アブソリュート Y	EOR hhll,Y
FST	インプライド	FST
INC	アキュムレータ	INC A
	ゼロページ	INC zz
	ゼロページ X	INC zz,X
	アブソリュート	INC hhll
	アブソリュート X	INC hhll,X
INX	インプライド	INX
INY	インプライド	INY
JMP	アブソリュート	JMP hhll
	ゼロページ・インダイレクト	JMP (zz)
	アブソリュート・インダイレクト	JMP (hhll)

命令名	アドレッシングモード名	記述形式
JSR	アブソリュート	JSR hhll
	スペシャルページ	JSR ¥hhll
	ゼロページ・インダイレクト	JSR (zz)
LDA [1]	イミディエイト	LDA #imm
	ゼロページ	LDA zz LDA bitsym
	ゼロページ X	LDA zz,X
	ゼロページ・インダイレクト X	LDA (zz,X)
	ゼロページ・インダイレクト Y	LDA (zz),Y
	アブソリュート	LDA hhll LDA bitsym
	アブソリュート X	LDA hhll,X
	アブソリュート Y	LDA hhll,Y
LDM	ゼロページ	LDM #imm,zz
LDX	イミディエイト	LDX #imm
	ゼロページ	LDX zz
	ゼロページ Y	LDX zz,Y
	アブソリュート	LDX hhll
	アブソリュート Y	LDX hhll,Y
LDY	イミディエイト	LDY #imm
	ゼロページ	LDY zz
	ゼロページ X	LDY zz,X
	アブソリュート	LDY hhll
	アブソリュート X	LDY hhll,X
LSR	アキュムレータ	LSR A
	ゼロページ	LSR zz
	ゼロページ X	LSR zz,X
	アブソリュート	LSR hhll
	アブソリュート X	LSR hhll,X
MUL	ゼロページ X	MUL zz,X
NOP	インプライド	NOP

注意事項

1. LDA 命令のオペランドにビットシンボルを記述すると、そのビットシンボルのアドレスのみが有効となります。

命令名	アドレッシングモード名	記述形式
ORA	イミディエイト	ORA #imm
	ゼロページ	ORA zz
	ゼロページ X	ORA zz,X
	ゼロページ・インダイレクト X	ORA (zz,X)
	ゼロページ・インダイレクト Y	ORA (zz),Y
	アブソリュート	ORA hhll
	アブソリュート X	ORA hhll,X
	アブソリュート Y	ORA hhll,Y
PHA	インプライド	PHA
PHP	インプライド	PHP
PLA	インプライド	PLA
PLP	インプライド	PLP
ROL	アキュムレータ	ROL A
	ゼロページ	ROL zz
	ゼロページ X	ROL zz,X
	アブソリュート	ROL hhll
	アブソリュート X	ROL hhll,X
ROR	アキュムレータ	ROR A
	ゼロページ	ROR zz
	ゼロページ X	ROR zz,X
	アブソリュート	ROR hhll
	アブソリュート X	ROR hhll,X
RRF	ゼロページ	RRF zz
RTI	インプライド	RTI
RTS	インプライド	RTS
SBC	イミディエイト	SBC #imm
	ゼロページ	SBC zz
	ゼロページ X	SBC zz,X
	ゼロページ・インダイレクト X	SBC (zz,X)
	ゼロページ・インダイレクト Y	SBC (zz),Y
	アブソリュート	SBC hhll
	アブソリュート X	SBC hhll,X
	アブソリュート Y	SBC hhll,Y
SEB	アキュムレータ・ビット	SEB i,A SEB bitsym,A

命令名	アドレッシングモード名	記述形式
SEB	ゼロページ・ビット	SEB i,zz SEB bitsym
SEC	インプライド	SEC
SED	インプライド	SED
SEI	インプライド	SEI
SET	インプライド	SET
SLW	インプライド	SLW
STA [1]	ゼロページ	STA zz STA bitsym
	ゼロページ X	STA zz,X
	ゼロページ・インダイレクト X	STA (zz,X)
	ゼロページ・インダイレクト Y	STA (zz),Y
	アブソリュート	STA hhll STA bitsym
	アブソリュート X	STA hhll,X
	アブソリュート Y	STA hhll,Y
STP	インプライド	STP
STX	ゼロページ	STX zz
	ゼロページ Y	STX zz,Y
	アブソリュート	STX hhll
STY	ゼロページ	STY zz
	ゼロページ X	STY zz,X
	アブソリュート	STY hhll
TAX	インプライド	TAX
TAY	インプライド	TAY
TST	ゼロページ	TST zz
TSX	インプライド	TSX
TXA	インプライド	TXA
TXS	インプライド	TXS
TYA	インプライド	TYA
WIT	インプライド	WIT

注意事項

1. STA 命令のオペランドにビットシンボルを記述すると、そのビットシンボルのアドレスのみが有効となります。

付録 C

アドレッシングモード別命令一覧表

C.1 アドレッシングモード別命令一覧表

アドレッシングモードごとの命令記述形式と該当命令を示します。説明で使用している記号は、付録 B.1の記号表に従います。

1. インプライド

該当命令	記述形式
BRK, CLC, CLD, CLI, CLT, CLV, DEX, DEY INX, INY, NOP, PHA, PHP, PLA, PLP, RTI RTS, SEC, SED, SEI, SET, SLW, STP, TAX TAY, TSX, TXA, TXS, TYA, WIT	BRK

2. イミディエイト

該当命令	記述形式
ADC, AND, CMP, CPX, CPY, EOR, LDA LDX, LDY, ORA, SBC	ADC #imm

3. アキュムレータ

該当命令	記述形式
ASL, DEC, INC, LSR, ROL, ROR	ASL A

4. ゼロページ

該当命令	記述形式
ADC, AND, ASL, BIT, CMP, COM, CPX, CPY DEC, EOR, INC, LDA, LDX, LDY, LSR, ORA ROL, ROR, RRF, SBC, STA, STX, STY, TST	ADC zz
LDM	LDM #imm,zz

5. ゼロページ・インデックス X

該当命令	記述形式
ADC, AND, ASL, CMP, DEC, DIV, EOR, INC LDA, LDY, LSR, MUL, ORA, ROL, ROR, SBC STA, STY	ADC　zz,X

6. ゼロページ・インデックス Y

該当命令	記述形式
LDX, STX	LDX　zz,Y

7. ゼロページ・インダイレクト

該当命令	記述形式
JMP, JSR	JMP　(zz)

8. ゼロページ・インダイレクト・インデックス X

該当命令	記述形式
ADC, AND, CMP, EOR, LDA, ORA, SBC, STA	ADC　(zz,X)

9. ゼロページ・インダイレクト・インデックス Y

該当命令	記述形式
ADC, AND, CMP, EOR, LDA, ORA, SBC, STA	ADC　(zz),Y

10. アブソリュート

該当命令	記述形式
ADC, AND, ASL, BIT, CMP, CPX, CPY, DEC EOR, INC, JMP, JSR, LDA, LDX, LDY, LSR ORA, ROL, ROR, SBC, STA, STX, STY	ADC　hhll

11. アブソリュート・インデックス X

該当命令	記述形式
ADC, AND, ASL, CMP, DEC, EOR, INC, LDA LDY, LSR, ORA, ROL, ROR, SBC, STA	ADC　hhll,X

12. アブソリュート・インデックス Y

該当命令	記述形式
ADC, AND, CMP, EOR, LDA, LDX, ORA, SBC STA	ADC hhll,Y

13. アブソリュート・インダイレクト

該当命令	記述形式
JMP	JMP (hhll)

14. スペシャルページ

該当命令	記述形式
JSR	JSR ¥hhhll

15. ゼロページ・ビット

該当命令	記述形式
CLB, SEB	CLB i,zz CLB bitsym

16. アキュムレータ・ビット

該当命令	記述形式
CLB, SEB	CLB i,A CLB bitsym,A

17. リラティブ

該当命令	記述形式
BCC, BCS, BEQ, BMI, BNE, BPL, BRA, BVC BVS	BCC hhll

18. ゼロページ・ビット・リラティブ

該当命令	記述形式
BBC, BBS	BBC i,zz,hhll BBC bitsym,hhll

19. アキュムレータ・ビット・リラティブ

該当命令	記述形式
BBC, BBS	BBC　i,A,hhll BBC　bitsym,hhll

付録 D

疑似命令一覧

D.1 疑似命令一覧の見方

SRA74 で使用可能な疑似命令をアルファベット順に掲載します。表記上の規則を以下に示します。

1. 説明中の [ ] 内の部分は省略可能部分を示しています。
2. △、▲はスペース又はタブコードを示しています。△の部分は必須、▲の部分は省略可能を示します。
3. 以下の説明ではラベルと疑似命令間は▲で示しています。ラベルを記述する場合 ':'(コロン) は必須ではありませんが、':' を省略した場合には疑似命令との間にスペース、又はタブコードが必要になります。

D.2 疑似命令一覧

.ASSERT　アセンブルアサーション宣言

書式

▲.ASSERT △’文字列’

内容

- アセンブル作業中に、オペランドに記述した文字列を画面に表示します。

記述例

```
.IF     MODE
 :
.ELSE
.ASSERT 'MODE is FALSE'
.ENDIF
```

.BEXT　外部参照指定 (一般ページビットシンボル)

書式

▲.BEXT △ビットシンボル [, ビットシンボル, . . , ビットシンボル]

内容

- オペランドに指定したビットシンボルを外部参照することを指定します。
- 本疑似命令により指定されたビットシンボルは一般ページ内にあるものと解釈します。
- 本疑似命令は、ラベルを参照する行より前に記述してください。

記述例

```
.BEXT   BIT0,BIT1,BIT2
 :
```

.BLKB　RAM 領域確保 (単位：バイト)

書式

▲ [ラベル:] ▲.BLKB △数式

内容

- 数式で指定した大きさの領域を確保します (単位:1 バイト) 。
- 数式中に使用するラベル又はシンボルは、この行より前に定義してください。
- オペランドに、リロケータブルな値を持つラベルは記述できません。

記述例

```
label:  .BLKB   10      ; 10 バイトの領域を確保します。
        :
```

.BYTE　バイトデータ設定

書式

▲ [ラベル:] ▲.BYTE △数式

内容

- 1 バイトの定数データを設定します。
- 2 個以上のデータを設定する場合は、各データを ',' で区切って指定してください。指定できる個数は 255 個までです。
- オペランドに、グローバルラベルを記述できます。

記述例

```
label1: .BYTE   10      ; 0AH を設定します。
label2: .BYTE   'A','B' ; 41H 、42H を設定します。
        :
```

.COL　カラム数指定 (既定値は 512)

書式

▲.COL △数式

内容

- リストの 1 行の文字数 (80 ～ 512) を指定します。
- 79 以下を指定した場合は 80 に、513 以上を指定した場合は 512 文字になります。
- 本疑似命令は、プログラム中に 1 回だけ記述できます。

記述例

```
.COL    80      ; カラム数を 80 に設定します。
 :
```

.END　プログラム終了宣言

書式

▲.END

内容

- ソースプログラムの終了を指定します。
- 本疑似命令以降の行はアセンブルしません。

記述例

```
 :
.END                ; プログラムの終了を宣言します。
```

.ENDFUNC　　ファンクション終了指定

書式

▲.ENDFUNC △ラベル

内容

- ファンクション (サブルーチン) の終了を指定します。
- この命令により、ソースラインデバッグが可能となります。

記述例

```
          .FUNC    SUB
SUB:      LDA      #1
           :
           :
          RTS
          .ENDFUNC SUB          ; ファンクションの終了を指定します。
```

.EQU(又は‘=’)　　同義定義

書式 1

シンボル△.EQU(又は ‘=’) △数式

書式 2

ビットシンボル△.EQU(又は ‘=’) △数式, 数式

内容

- 左辺のシンボルに数値を割り当てます。
- 書式 1 はシンボルに符号付き 16 ビットの整数値を割り当てます。書式 2 はシンボルに 0 〜 7 のビット値と符号付き 16 ビットの整数値 (アドレス) を割り当てます。
- 数式中に使用するラベル又はシンボルは、この行より前に定義してください。
- オペランドにリロケータブルな値を持つラベルは記述できません。
- シンボルについては再設定を行うことができます。
- コマンドパラメータ “–Q”を指定することにより、シンボルの再設定時にワーニングを出力します。

記述例

```
symbol   .EQU    1         ; symbol に 1 を設定
  :
symbol   .EQU    2         ; symbol に 2 を設定
  :
symbol   .EQU    3         ; symbol に 3 を設定
  :
bitsym   .EQU    1,23H     ; bitsym に 23H のビット 1 を設定
  :
```

.ERROR　アセンブルエラー宣言

書式

▲.ERROR △’文字列’

内容

- 条件アセンブル命令 (.IF) と組み合わせて使用します。
- 条件として有り得ないものが指定された場合に、オペランドに記述した文字列を画面へ表示しアセンブル作業を終了します。

記述例

```
.IF      MODE
      :
.ELSE
.ERROR   'undefined assemble mode'
.ENDIF
```

.EXT　外部参照指定 (一般ページ)

書式

▲.EXT △ラベル又はシンボル [, ラベル又はシンボル,.., ラベル又はシンボル]

内容

- オペランドに指定したラベルをアブソリュートアドレッシングモードで外部参照することを指定します。ゼロページアドレッシングモードでアセンブルしたい場合には.ZEXT で、スペシャルページアドレッシングモードでアセンブルしたい場合には.SEXT で指定してください。
- 本疑似命令は、ラベルを参照する行より前に記述してください。

記述例

```
.EXT    WORK1,WORK2,WORK3
 :
```

.FUNC　ファンクション開始指定

書式

▲.FUNC △ラベル

内容

- ファンクション (サブルーチン) の開始を指定します。
- この命令により、ソースラインデバッグが可能となります。
- “ .FUNC ” 疑似命令に用いたラベル名を、別の “ .FUNC ” 疑似命令で使用することはできません。

記述例

```
        .FUNC    SUB        ; ファンクションの開始を指定します。
SUB:    LDA      #1
         :
         :
        RTS
        .ENDFUNC SUB
```

.IF (.ELSE) .ENDIF　　条件付きアセンブル

書式

▲.IF △式
　　<文 1>
▲.ELSE
　　<文 2>
▲.ENDIF

内容

- オペランドの式には数式または文字列式を記述することができます。
- オペランドの数式が真 (0 でない) なら文 1 をアセンブルし、偽 (0) なら文 2 をアセンブルします。
- オペランドの文字列式が真（文字列データがある）なら文 1 をアセンブルし、偽（文字列データがない）なら文 2 をアセンブルします。
- ネスティングは 20 レベルまで可能です（但し、これはホストマシンのメモリ容量に依存します)。
- 文 1 、文 2 には複数行が記述できます。
- オペランドに、リロケータブルな値を持つラベルは記述できません。
- オペランドに条件式判定の比較演算子が記述可能です。記述可能な条件判定式を表に示します。

<	より小さい
>	より大きい
==	等しい
!=	等しくない
< =	小さいか等しい
> =	大きいか等しい

- 未定義シンボルを数式中に使用した場合は 0 の値を持つシンボルとして処理されます。また、文字列式中に記述した場合は文字列として処理されます。
- 論理演算子 (‘ || ’, ‘ && ’) によりオペランドに条件式を 6 つまで組み合わせることができます。但し論理演算子の間に優先順位はなく、条件式は先頭から順番に評価されます。

記述例

(1)

```
.IF     FLAG    ; FLAG の値が真なら.ELSE までをアセンブルする
 :
 :
.ELSE           ; 偽なら.ENDIF までをアセンブルする
 :
 :
.ENDIF
```

(2)

```
ADD:    .MACRO  OP1,OP2,OP3
        .IF     "OP3"       ;引数 OP3 が存在した場合
                            ;.ELSE までをアセンブルする。
        ADC     OP1,OP2,OP3
        .ELSE
        ADC     OP1,OP2
        .ENDIF
```

(3)

```
.IF  FLAG1 && FLAG2 || FLAG3 && FLAG4
 :     この部分は次に示す表のとおりにアセンブルの
 :     対象となるかどうかが決定されます。
.ENDIF
```

FLAG1	FLAG2	FLAG3	FLAG4	結果
真	真	–	–	真
真	偽	真	真	真
真	偽	真	偽	偽
真	偽	偽	–	偽
偽	–	–	–	偽

‘ – ’は真偽どちらの場合でも同じ結果を示します。

.INCLUDE　ファイル読み込み

書式

▲.INCLUDE △ファイル名

内容

- 本疑似命令を記述した場所に、オペランドで指定したファイルの内容を読み込みます。
- ファイル名はフルネームで指定してください。
- ネスティングは 4 レベルまで可能です。
- ネスティングレベルをプリントファイルに出力します。

記述例

```
.INCLUDE    TEST.INC    ; TEST.INC の内容を読み込む
 :
```

.LIB　ライブラリファイル名指定

書式

▲.LIB △ファイル名 [, ファイル名, . . , ファイル名]

内容

- リンク対象のライブラリファイル名を指定します。
- 指定可能なファイルは、.LIB 属性のファイルだけです。これ以外のファイルを指定した場合は、リンク時にエラーになります。
- 本疑似命令は入れ子にできません。
- ファイル名にディレクトリパス、ファイル属性 (.LIB) は記述できません。

記述例

```
.LIB     LIB1,LIB2,LIB3
 :
```

.LINE　1 ページ当りの行数指定 (既定値は 54)

書式

▲.LINE △数式

内容

- リストの 1 ページ当たりの行数 (5 〜 255) を指定します。
- 本疑似命令は、プログラム中に 1 回だけ記述できます。

記述例

```
.LINE   60      ; 行数を 60 に設定します。
 :
```

.LIST　リスト出力開始 (既定値)

書式

▲.LIST

内容

- PRN ファイルへのリスト出力を行います。
- 疑似命令.NLIST により PRN ファイルへの出力を中止後、リスト出力を再開する場合に使用します。

記述例

```
.NLIST          ; リスト出力の抑止を行います。
 :              ; ‘‘.LIST’’ までの間を PRN ファイルへ出力しません。
 :
.LIST           ; リスト出力の開始を行います。
 :              ; この疑似命令以降を PRN ファイルへ出力します。
 :
```

.LISTM　マクロ展開部のリスト出力開始 (既定値)

書式

▲.LISTM

内容

- PRN ファイルへ、マクロ命令の展開部分を出力します。
- 疑似命令.NLISTM によりマクロ展開部の出力を中止後、リスト出力を再開する場合に使用します。
- 疑似命令.NLIST によりリスト出力全体が抑止されている場合は、.LISTM 命令は無効となります。

記述例

```
.NLISTM          ; マクロ展開部のリスト出力の抑止を行います。
 :               ; ‘‘.LISTM’’ までの間のマクロ展開部分を出力しません。

 :
.LISTM           ; マクロ展開部のリスト出力の開始を行います。
 :               ; この疑似命令以降のマクロ展開部分は出力します。
 :
```

.NLIST　リスト出力抑止

書式

▲.NLIST

内容

- PRN ファイルへの出力を抑止します。
- この状態は疑似命令.LIST により解除できます。

記述例

```
.NLIST           ; リスト出力の抑止を行います。
 :               ; ‘‘.LIST’’ までの間を PRN ファイルへ出力しません。
 :
.LIST            ; リスト出力の開始を行います。
 :               ; この疑似命令以降を PRN ファイルへ出力します。
 :
```

.NLISTM　マクロ展開部のリスト出力抑止

書式

▲.NLISTM

内容

- PRN ファイルへ、マクロ命令の展開部分を出力しません。
- この状態は疑似命令.LISTM により解除できます。

記述例

```
.NLISTM         ; マクロ展開部のリスト出力の抑止を行います。
 :              ; ‘‘.LISTM’’までの間のマクロ展開部分を出力しません。

 :
.LISTM          ; マクロ展開部のリスト出力の開始を行います。
 :              ; この疑似命令以降のマクロ展開部分は出力します。
 :
```

.OBJ　リロケータブルファイル名指定

書式

▲.OBJ △ファイル名 [, ファイル名, . . , ファイル名]

内容

- リンク対象のリロケータブルファイル名を指定します。
- 指定可能なファイルは、.R74 属性のファイルだけです。これ以外のファイルを指定した場合、リンク時にエラーになります。
- 本疑似命令は入れ子にできません。
- ファイル名にディレクトリパス、ファイル属性 (.R74) は記述できません。

記述例

```
.OBJ     OBJ1,OBJ2,OBJ3
 :
```

.ORG(又は “∗=”)　　アドレス宣言 (既定値は 0000H)

書式

▲.ORG(又は ‘‘∗=’’) △数式

内容

- この行以降の開始アドレスを宣言します。
- 指定がない場合、開始アドレスは 0000_{16} として処理します。
- 本疑似命令を記述したセクションは、絶対属性になります。本疑似命令を含まないセクションは、相対属性になります。
- 数式中に使用するラベル又はシンボルは、この行より前に定義してください。
- オペランドに、リロケータブルな値を持つラベルは記述できません。

記述例

```
.ORG    0C000H  ; ロケーションを C000H に設定します。
 :
*=      0E000H  ; ロケーションを E000H に設定します。
```

.PAGE　　リスト改ページおよびタイトル指定

書式

▲.PAGE △ [’タイトル’]

内容

- この命令の直前でリストの改ページを行い、オペランドで指定したタイトルをリストのヘッダ部に出力します。タイトルは、’ (シングルクォート) 又は ”(ダブルクォート) で囲んで記述してください。
- タイトルの文字数は、カラム指定が 80 のとき最大 20 文字、105 から 512 のとき最大 45 文字まで、81 から 104 まではカラム数から 60 を引いた文字数 まで、各々使用できます。なおタイトル省略時には改ページのみを行います。

記述例

```
.PAGE   ’PROG1’ ; PROG1 を PRN ファイルのヘッダ部に出力します。
 :
```

.PMOD　ROM 領域指定 (一般ページ)

書式

▲.PMOD

内容

- この命令以降が、一般ページの ROM 領域であることを指定します。
- この命令は、疑似命令.SECTION のオペランドに 'P' を指定した場合と同等です。
- この指定は、他の領域指定命令が現れるまで有効です。
- ファイルの先頭に領域指定命令がない場合は、.PMOD が規定値として選択されます。従って、ファイルの最初の ROM 領域に限り領域指定命令は省略できます。

記述例

```
        .PMOD
        .EXT    MAIN,SUB
        .PUB    ENZAN
ENZAN:
        CLT
         :
```

.PUB　パブリック指定

書式

▲.PUB △ビットシンボル又はシンボル又はラベル,....

内容

- オペランドに指定したビットシンボル、シンボル又はラベルを、他のソースファイルから参照可能にします。
- Z セクション及び R セクションのみで構成される RAM 領域のみのファイルでは、総てのラベルはグローバル属性として処理されます。従って、ラベルに対する.PUB 指定は省略することができます。
- 本疑似命令は、ラベル又はシンボルを定義する行より前に記述してください。

記述例

```
          .RMOD
          .PUB    WORK1,WORK2,WORK3
WORK1:    .BLKB   1
WORK2:    .BLKB   1
WORK3:    .BLKB   1
           :
```

.RMOD　　RAM 領域指定 (一般ページ)

書式

▲.RMOD

内容

- この命令以降が、一般ページの RAM 領域であることを指定します。
- この命令は、疑似命令.SECTION のオペランドに 'R' を指定した場合と同等です。
- この指定は、他の領域指定命令が現れるまで有効です。
- Z セクション及び R セクションのみで構成される RAM 領域のみのファイルでは、総てのラベルはグローバル属性として処理されます。従って、ラベルに対する.PUB 指定は省略することができます。

記述例

```
          .RMOD
WORK1:    .BLKB   1
WORK2:    .BLKB   1
WORK3:    .BLKB   1
           :
```

.SECTION 領域指定

書式

▲.SECTION △セクション名

内容

- この行以降が、オペランドで指定した名前を持つ領域であることを指定します。
- SRA74 では、オペランドに予約セクション名 (P 、R 、S 、Z) を記述した場合、本疑似命令以降はそれぞれの領域属性として認識され処理されますが、任意のセクション名を記述した場合は、この命令以降の命令により領域属性を決定します。但し、この場合一般ページの ROM 領域、又は一般ページの RAM 領域としてしか認識されません。
- この指定は、他の領域指定命令が現れるまで有効です。
- 同じ名前をもつセクションがファイル内に複数個存在してもかまいません。
- ファイルの先頭に領域指定命令がない場合は、.SECTION P が規定値として選択されます。従って、ファイルの最初の ROM 領域に限り、領域指定命令は省略できます。

記述例

```
        .SECTION    DATA    ; DATA セクションを開始します。
datatop:
nulldt: .BLKB    8
         :
        .SECTION     STACK  ; STACK セクションを開始します。
        .BLKB        16
stacktop:
         :
        .SECTION     PROG   ; PROG セクションを開始します。
_init:
        LDX          #stacktop
        TXS
        LDA          #SIZEOF DATA
        LDX          #datatop
         :
```

.SEXT 外部参照指定 (スペシャルページ)

書式

▲.SEXT △ラベル又はシンボル [, ラベル又はシンボル,.., ラベル又はシンボル]

内容

- オペランドに指定したラベルを、スペシャルページアドレッシングモード (JSR ¥のみ) 、又はアブソリュートアドレッシングモードで外部参照することを指定します。ゼロページアドレッシングモードでアセンブルしたい場合には.ZEXT で指定してください。
- 本疑似命令は、ラベルを参照する行より前に記述してください。

記述例

```
        .SECTION P
        .EXT    WORK1,WORK2
        .SEXT   SUB
START:
        LDA     WORK1
        JSR     \SUB
         :
```

.SMOD ROM 領域指定 (スペシャルページ)

書式

▲.SMOD

内容

- この命令以降が、スペシャルページの ROM 領域であることを指定します。
- この命令は、疑似命令.SECTION のオペランドに 'S' を指定した場合と同等です。
- この指定は、他の領域指定命令が現れるまで有効です。

記述例

```
        .SMOD
        .EXT    MAIN,SUB
        .PUB    ENZAN
ENZAN:
        CLT
         :
```

.VER　プログラムバージョン指定

書式

▲.VER △’文字列’

内容

- リロケータブルファイルのバージョン指定を行います。
- LINK74 はコマンドパラメータ “–V” が指定されると、各リロケータブルファイル中のバージョン指定の一致を確認します。これによりリロケータブルファイル間のバージョン整合性の確認が行えます。コマンドパラメータ “–V” の詳細については、LINK74 操作マニュアルを参照してください。
- バージョンの一致は、文字列の比較により確認します。大文字/小文字も区別しますので、注意して記述してください。
- 本疑似命令は、プログラム中に 1 回だけ記述できます。

記述例

```
        .VER    'V.1.0'       ; バージョン‘‘V.1.0’’を指定します。
         :
```

.WORD　ワードデータ設定

書式

▲ [ラベル:] ▲.WORD △数式

内容

- 数式のもつ値を設定します (単位:ワード) 。
- 2 つ以上のデータを設定する場合は、各データを ‘,’ で区切って指定してください。1 行で指定できる個数は 16 個までです。
- データは下位バイトより設定します。
- オペランドにグローバルラベルを記述できます。

記述例

```
label:  .WORD   0E000H  ; 00H,E0H を設定します。
        .WORD   symbol  ; symbol のもつ値を下位バイトより設定します。

         :
```

.ZBEXT　外部参照指定 (ゼロページビットシンボル)

書式

▲.ZBEXT △ビットシンボル [, ビットシンボル,.., ビットシンボル]

内容

- オペランドに指定したビットシンボルを外部参照することを指定します。
- 本疑似命令により指定されたビットシンボルはゼロページ内にあるものと解釈します。
- 本疑似命令は、ラベルを参照する行より前に記述してください。

記述例

```
.ZBEXT   BIT0,BIT1,BIT2
 :
```

.ZEXT　外部参照指定 (ゼロページ)

書式

▲.ZEXT △ラベル又はシンボル [, ラベル又はシンボル,.., ラベル又はシンボル]

内容

- オペランドに指定したラベルを、ゼロページアドレッシングモードで外部参照することを指定します。アブソリュートアドレッシングモードでアセンブルしたい場合には.EXT、スペシャルページアドレッシングモードでアセンブルしたい場合には.SEXTで指定してください。
- 本疑似命令は、ラベルを参照する行より前に記述してください。

記述例

```
        .SECTION Z
        .ZEXT   WORK1,WORK2
        .SEXT   SUB
START:
        LDA     WORK1
        JSR     \SUB
         :
```

.ZMOD　　RAM 領域指定 (ゼロページ)

書式

▲.ZMOD

内容

- この命令以降が、ゼロページの RAM 領域であることを指定します。
- この命令は、疑似命令.SECTION のオペランドに 'Z' を指定した場合と同等です。
- この指定は、他の領域指定命令が現れるまで有効です。
- Z セクション及び R セクションのみで構成される RAM 領域のみのファイルでは、総てのラベルはグローバル属性として処理されます。従って、ラベルに対する.PUB 指定は省略することができます。

記述例

```
          .ZMOD
WORK1:    .BLKB   1
WORK2:    .BLKB   1
WORK3:    .BLKB   1
           :
```

D.3　予約疑似命令一覧

以下の疑似命令は、将来の拡張のために予約しています。これらの疑似命令を記述してもアセンブルには影響を与えません。

.ENDIO　I/O 領域終了宣言 (予約)

書式

▲.ENDIO

内容

- I/O 領域の終了宣言を行います。
- .IO より.ENDIO までに記述したラベル及びシンボルを I/O 領域として解釈します。

記述例

```
        .IO
port0   .EQU    00H
port1   .EQU    01H
        .ENDIO
```

.ENDPROC　プログラムモジュール終了宣言 (予約)

書式

▲.ENDPROC

内容

- メインプログラム、サブプログラム、割り込み処理プログラムの各モジュールの終了を宣言します。
- .PROCINT 、.PROCMAIN 、.PROCSUB より.ENDPROC で囲んだ範囲を一つのプログラムモジュールとして解釈します。

記述例

```
        .PROCMAIN
MAIN:
         :
        JMP     MAIN
        .ENDPROC
```

.ENDRAM　RAM 領域終了宣言 (予約)

書式

▲.ENDRAM

内容

- RAM 領域の終了宣言を行います。
- .RAM より.ENDRAM までに記述したラベル及びシンボルを RAM 領域として解釈します。

記述例

```
        .RAM
work0:  .BLKB   1
work1:  .BLKB   1
        .ENDRAM
```

.IO　I/O 領域開始宣言 (予約)

書式

▲.IO

内容

- I/O 領域の開始宣言を行います。
- .IO より.ENDIO までに記述したラベル及びシンボルを I/O 領域として解釈します。

記述例

```
        .IO
port0   .EQU    00H
port1   .EQU    01H
        .ENDIO
```

.PROCINT　割り込み処理プログラム開始宣言 (予約)

書式

▲.PROCINT △ [ラベル]

内容

- 割り込み処理プログラムの開始宣言を行います。
- .PROCINT より.ENDPROC で囲んだ範囲を割り込み処理プログラムとして解釈します。
- SRA74 は、オペランドに記述したラベルをこの行のラベルとして処理します。

記述例

```
.PROCINT INT
 :
 :
.ENDPROC
```

.PROCMAIN　メインプログラム開始宣言 (予約)

書式

▲.PROCMAIN △ [ラベル]

内容

- メインプログラムの開始宣言を行います。
- .PROCMAIN より.ENDPROC で囲まれた範囲をメインプログラムとして解釈されます。
- SRA74 は、オペランドに記述したラベルをこの行のラベルとして処理します。

記述例

```
.PROCMAIN MAIN
 :
.ENDPROC
```

.PROCSUB　サブプログラム開始宣言 (予約)

書式

▲.PROCSUB △ [ラベル]

内容

- サブプログラムの開始宣言を行います。
- .PROCSUB より.ENDPROC で囲んだ範囲をサブプログラムとして解釈します。
- SRA74 は、オペランドに記述したラベルをこの行のラベルとして処理します。

記述例

```
.PROCSUB SUB
 :
 :
.ENDPROC
```

.PROGNAME　プログラム名宣言 (予約)

書式

▲.PROGNAME △プログラム名

内容

- プログラム名の宣言を行います。
- オペランドの内容をプログラムタイトルとして解釈します。

記述例

```
.PROGNAME プリンタ制御プログラム
```

.RAM　　RAM 領域開始宣言 (予約)

書式

▲.RAM

内容

- RAM 領域の開始宣言を行います。
- .RAM より.ENDRAM までに記述したラベル及びシンボルを RAM 領域として解釈します。

記述例

```
        .RAM
work0:  .BLKB    1
work1:  .BLKB    1
        .ENDRAM
```

付録 E

マクロ命令一覧

E.1 マクロ命令一覧の見方

SRA74 で使用可能なマクロ命令をアルファベット順に掲載します。表記上の規則を以下に示します。

1. 説明中の [ ] 内の部分は省略可能部分を示しています。
2. △、▲はスペース又はタブコードを示しています。△の部分は必須、▲の部分は省略可能を示します。
3. 以下の説明ではラベルとマクロ命令間は▲で示しています。ラベルを記述する場合 ':'(コロン) は必須ではありませんが、':' を省略した場合にはマクロ命令との間にスペース、又はタブコードが必要になります。

E.2 マクロ命令一覧

.ENDM　ENDM マクロ命令

書式

▲.ENDM

内容

- この命令は、すべてのマクロ定義部の終了を指定します。

記述例

```
[マクロ定義]
    ADD:    .MACRO   VAL
            CLC
            ADC      VAL
            .ENDM
[呼び出し例]
            ADC      #10
[マクロ展開]
            CLC
            ADC      #10
            .ENDM
```

.EXITM　　EXITM マクロ命令

書式

▲.EXITM

内容

- この命令は、全てのマクロ展開を中止し一番近い.ENDM に制御を渡します。

記述例

```
[マクロ定義]
    DATA1:  .MACRO   VAL
            .IF     LABEL
            .BYTE   VAL
            .EXITM
            .ENDIF
            .WORD   VAL
            .ENDM
[呼び出し例]
    LABEL   .EQU    1
            DATA1   10
[マクロ展開]
            .IF     LABEL
            .BYTE   10
            .EXITM
            .ENDIF
            .ENDM
```

.LOCAL　　LOCAL マクロ命令

書式

▲.LOCAL △ラベル,[ラベル,..,ラベル]

内容

- この命令は、マクロ定義のなかで定義されているラベルをマクロ内ローカルラベルとします。
- .LOCAL 宣言されたラベルを、その順に..n(n は 10 進数で..1 〜..65535) というラベルを割り当ててアセンブルします。従ってユーザーは、..n というラベルは使用できませんので注意してください。

記述例

```
[マクロ定義]
    LOOP:   .MACRO
            .LOCAL   LOOP1
            LDA     #20
    LOOP1:  DEC     A
            BNE     LOOP1
            .ENDM
[呼び出し例]
            LOOP
[マクロ展開]
            LDA     #20
    ..0:    DEC     A
            BNE     ..0
            .ENDM
```

.MACRO ～.ENDM　　MACRO マクロ命令

書式

▲ [マクロ名:] ▲.MACRO △ [引数 1, 引数 2, . . , 引数 n]

内容

- .MACRO と記述された行からマクロ定義が始まり、.ENDM が記述された行でそのマクロ定義が終了します。
- .MACRO 命令の行に付けたラベルがこのマクロ定義に付けられた名前となります。
- マクロ定義には引数を持つマクロと持たないマクロがあり、引数を持つマクロを使用する場合には、マクロの呼び出し文で必要な引数を渡す必要があります。
- マクロ呼び出しが行われると引数は展開される際、マクロ定義で記述された仮の引数の順序に従って引数を渡します。
- .MACRO ～.ENDM の間には、アセンブリ言語命令、構造化記述命令、マクロ命令、及び .INCLUDE 以外の疑似命令が記述できます。但し、マクロ定義のネスティングは 20 レベルまでです。
- マクロの呼び出しはマクロ定義の後であれば、プログラムの任意の場所で行えます。マクロ定義がマクロ呼び出しの後でしか出てこない場合はエラーとなります。
- マクロ呼び出しでオペランドに指定した引数は、マクロ定義の仮の引数と左から順に対応する引数と置き換えられます。なお、引数の数と定義したときの仮の引数の数は一致していなくてもかまいませんが、引数の数が仮の引数より多いときは余分な引数は無視され、少ない場合は、不足する仮の引数は空文字 (長さ 0 の文字列) とみなされます。
- マクロ定義に用いた仮の引数に関係なく、引数は任意の個数指定できますが、その全てが一行以内に収まらなければなりません。引数と引数の間は',' で区切るため、カンマやスペースを引数として渡す場合は ‘ ” ’ でくくらなければなりません。但し、'()' 内のカンマは引数の区切りとして扱いません。
- マクロ展開された部分は、リストファイル中に‘+’ で示されます。
- マクロ定義は同一のマクロ名で複数回行うことができます。このとき、マクロ呼び出しを行った場合、その呼び出し直前に定義されたものが呼び出し対象となります。

記述例

例 1）オペランドを持たないマクロ定義

[マクロ定義]

```
ADDa:   .MACRO
        LDA     ABC
        LDX     #DEF
        CLC
        ADC     TABLE,X
        STA     GHI
        .ENDM
```

[呼び出し例]

```
        ADDa
```

[マクロ展開]

```
        LDA     ABC
        LDX     #DEF
        CLC
        ADC     TABLE,X
        STA     GHI
        .ENDM
```

例 2）オペランドを持つマクロ定義

[マクロ定義]

```
ADDb:   .MACRO   V1,IMM,V2    ; 仮の引数（V1,IMM,V2）
        LDA     V1
        LDX     #IMM
        CLC
        ADC     TABLE,X
        STA     V2
        .ENDM
```

[呼び出し例]

```
        ADDb    WORK1,10,WORK2
```

[マクロ展開]

```
        LDA     WORK1
        LDX     #10
        CLC
        ADC     TABLE,X
        STA     WORK2
        .ENDM
```

記述例

例 3）マクロのネスティング

[マクロ定義]

```
ADD:    .MACRO  SRC
        CLC
        ADC     SRC
        .ENDM
```

[呼び出し例]

```
ADDW:   .MACRO  SRC
        ADD     SRC     ;マクロ内でのマクロ呼び出し
        ADC     SRC+1
        .ENDM
```

例 4）マクロの再帰呼び出し

[マクロ定義]

```
MAC:    .MACRO              ; 一度目の定義
DATA:   .BLKB   1
MAC:    .MACRO  VALUE       ; 二度目の定義
        LDM     #VALUE,DATA
        .ENDM
        .ENDM
```

[呼び出し例]

```
        MAC                 ; 領域の確保と新たなマクロ定義
         :
        MAC     10H         ; 新たに定義されたマクロの呼び出し
```

.REPEAT～.ENDM　　REPEATマクロ命令

書式

▲[ラベル:]▲.REPEAT △回数

内容

- .REPEATから.ENDMまでにある740ファミリ シリーズの命令をオペランドで指定した回数だけ繰り返しアセンブルします。
- .REPEAT命令の行に付けたラベルは生成された行の最初のラベルとなります。
- .REPEAT～.ENDMの間には、アセンブリ言語命令、構造化記述命令、マクロ命令、及び.INCLUDE以外の疑似命令が記述できます。但し、マクロのネスティングは20レベルまでです。
- オペランドには、数値定数、記号定数(ラベル)が記述できますが、リロケータブルな値を持つラベルは記述できません。
- オペランドに記述できる値は、0～32767の範囲です。この範囲を越える値を指定した場合はエラーとなります。

記述例

[ソース記述例]

```
TIME5:  .REPEAT  5
        NOP
        .ENDM
```

[マクロ展開後]

```
TIME5:  .REPEAT  5
        NOP
        .ENDM
TIME5:
        NOP
        NOP
        NOP
        NOP
        NOP
```

.REPEATC～.ENDM　REPEATC マクロ命令

書式

▲ [ラベル:] ▲.REPEATC △仮引数, 実引数

内容

- オペランドで実引数で与えた文字数回.ENDM までを繰り返しアセンブルします。
- 繰り返すたびに実引数から 1 文字ずつ取り出しこれを仮引数に渡します。
- .REPEATC 命令の行に付けたラベルは生成された行の最初のラベルとなります。
- .REPEATC ～.ENDM の間には、アセンブリ言語命令、構造化記述命令、マクロ命令、及び .INCLUDE 以外の疑似命令が記述できます。但し、マクロのネスティングは 20 レベルまでです。
- 文字列にスペース、タブ、,(カンマ) などの特殊文字を含む場合は、全体を‘”’でくくって文字列を指定します。このとき文字列は‘”’を取り除いた残りが用いられます。

記述例

例 1)

[ソース記述例]

```
DATA:   .REPEATC VAL,ABCDE
                  ↑    ↑
               仮引数 実引数
        .BYTE   'VAL'
        .ENDM
```

[マクロ展開後]

```
DATA:   .REPEATC VAL,ABCDE
        .BYTE   'VAL'
        .ENDM
DATA:
        .BYTE   'A'
        .BYTE   'B'
        .BYTE   'C'
        .BYTE   'D'
        .BYTE   'E'
```

記述例

例 2）

[ソース記述例]

```
DATA:   .REPEATC VAL,"ABC,;"
                  ↑    ↑
               仮引数 実引数
        .BYTE   'VAL'
        .ENDM
```

[マクロ展開後]

```
DATA:   .REPEATC VAL,"ABC,;"
        .BYTE   'VAL'
        .ENDM
DATA:
        .BYTE   'A'
        .BYTE   'B'
        .BYTE   'C'
        .BYTE   ','
        .BYTE   ';'
```

.REPEATI～.ENDM　　REPEATI マクロ命令

書式

▲ [ラベル:] ▲.REPEATI △仮引数, 実引数 [, 実引数,..., 実引数]

内容

- オペランドに記述された引数の個数回.ENDM までを繰り返しアセンブルします。
- 繰り返すたびにオペランドに記述された引数から 1 つずつ取り出しこれを仮引数に渡します。
- .REPEATI 命令の行に付けたラベルは生成された行の最初のラベルとなります。
- .REPEATI ～.ENDM の間には、アセンブリ言語命令、構造化記述命令、マクロ命令、及び .INCLUDE 以外の疑似命令が記述できます。但し、マクロのネスティングは 20 レベルまでです。
- 実引数には数値定数、文字定数、記号定数 (ラベル) 、文字列定数が記述できます。他のマクロ命令は記述できません。
- 実引数の中にスペース、タブ、,(カンマ) が含まれる場合は全体を ‘”’ で囲んで表します。

記述例

例 1)

[ソース記述例]

```
SUB:    .REPEATI INST,"NOP","LDA #1","JSR SUB1","RTS"
                   ↑           ↑
                 仮引数        実引数
        INST
        .ENDM
```

[マクロ展開後]

```
SUB:    .REPEATI INST,"NOP","LDA A,#1","JSR SUB1","RTS"
        INST
        .ENDM
SUB:
        NOP
        LDA #1
        JSR SUB1
        RTS
```

例 2)

[ソース記述例]

```
DATA:   .REPEATI VAL,0,1,2,"'HELLO !!'"
                   ↑     ↑
                 仮引数  実引数
        .BYTE   VAL
        .ENDM
```

[マクロ展開後]

```
DATA:   .REPEATI VAL,0,1,2,"'HELLO !!'"
        .BYTE   VAL
        .ENDM
DATA:
        .BYTE   0
        .BYTE   1
        .BYTE   2
        .BYTE   'HELLO !!'
```

付録 F

構造化命令一覧

F.1　構造化命令一覧の見方

SRA74 で使用可能な構造化命令をアルファベット順に掲載します。表記上の規則を以下に示します。

1. △、▲はスペース又はタブコードを示しています。△の部分は必須、▲の部分は省略可能です。
2. 説明中の [ ] の部分は省略できます。
3. ラベルを記述する場合 ':'(コロン) は必須ではありませんが、':' を省略した場合には各構造化命令との間にスペース、又はタブコードが必要になります。通常 ':' を付加して記述することを推奨します。

F.2　構造化命令一覧

BREAK　　BREAK 文

書式

▲ [ラベル:] ▲ (L)BREAK

内容

- break 文は、該当する for 文、do 文、又は switch 文の実行を中止して、その次に実行する文に制御を渡します。
- for 文、do 文、switch 文の中でのみ使用できます。

CONTINUE　　CONTINUE 文

書式

▲ [ラベル:] ▲ (L)CONTINUE

内容

- continue 文は、それを含む最小の繰り返し文 for 文、do 文中の最後の文に仮想的に空文を置き、その空文に制御を渡します。
- for 文、do 文の中でのみ使用できます。

DO ～ WHILE　　DO 文

書式

▲ [ラベル:] ▲ (L)DO
　　<文>
▲ [ラベル:] ▲ WHILE △条件式

内容

- do 文は、条件式が満たされている (真である) 間、文を繰り返し実行します。但し、繰り返すかどうかの判定を文の実行後に行います。そのため、do 文はある条件で繰り返しを終了する場合、終了前にもう一度実行して繰り返しから抜けたい場合に使用すると便利です。
- 条件式に、ever を記述すると無限ループとなります。
- 条件式の詳細は、構文図を参照してください。

FOR ～ NEXT　　FOR 文

書式

▲ [ラベル:] ▲ (L)FOR △条件式
　　<文>
▲ [ラベル:] ▲ NEXT

内容

- for 文は、繰り返しを制御する命令で、指定した条件式が真である間、文を繰り返し実行します。
- 条件式に、ever を記述すると無限ループとなります。
- 条件式の詳細は、構文図を参照してください。

IF ～ (ELSE) ～ ENDIF　IF 文

書式

▲ [ラベル:] ▲ (L)IF △条件式
　　<文>
▲ [[ラベル:] ▲ (L)ELSE]
　　<文>
▲ [ラベル:] ▲ ENDIF

内容

- if 文は、制御の流れを 2 方向に変える命令で、分岐する方向は条件式によって決定されます。条件式の演算結果がゼロ (偽) か否 (真) かによって分岐が判定されます。真なら、その直後の命令、偽で else がある場合は else の直後の命令、なければ endif 以降の命令へ制御を渡します。
- else 部は省略可能です。
- if 文の入れ子の数には制限がありません。
- if 文が複雑な入れ子になる場合、if と else の対応関係は最も近い if と else が順次ペアになります。
- 条件式の詳細は、構文図を参照してください。

SWITCH～CASE～ENDS　　SWITCH 文

書式

▲[ラベル:]▲(L)SWITCH △条件式
▲[ラベル:]▲CASE △定数
　　<文>
▲[ラベル:]▲CASE △定数
　　<文>
　　:
▲[ラベル:]▲DEFAULT
　　<文>
▲[ラベル:]▲ENDS

内容

- switch 文は、条件式の値によって制御をいずれかの文に渡すものです。式の値が文の前に付けられた case 接頭辞の定数と比較され、一致した文に制御が渡されます。式の値と case で指定した値に一致するものがない場合、default 接頭辞があれば、その文に制御が渡されます。default がなければどの文も実行されずに switch 文を抜けます。
- default 部は省略可能です。
- 一致した case 文に制御が渡された後は、それ以降に記述されている case 文も順次実行していきます。もし、次の case 文に移らずに switch 文を抜けたいときには、break 文を用いて switch 文から抜けます。
- case の値として相対値及び、外部参照値を用いることはできません。
- switch 文の入れ子の数には制限がありません。
- 条件式、定数の詳細は、構文図を参照してください。

代入　　代入文

書式

▲[ラベル:]▲左辺▲=▲右辺

内容

- 右辺を左辺に代入します。なお、メモリビット変数、レジスタビット変数、フラグ変数への代入は、0 又は 1 の値をもつ数値定数及び記号定数のみ使用できます。
- 左辺、右辺の詳細は、構文図を参照してください。

F.3 展開例

構造化記述言語をアセンブリ言語に展開した例について説明します。

F.3.1 代入文の展開例

条件分岐命令以外のアセンブリ言語と構造化記述言語との対応関係を説明します。条件分岐についてはF.3.2の条件式の展開で説明します。なお、代入文の展開例で使用している記号の意味を以下に示します。

表 F.1: 記号表

記号	内容	記号	内容
A	アキュムレータ	X	インデックスレジスタX
Y	インデックスレジスタY	S	スタックポインタ
P	プロセッサステータスレジスタ	C	キャリーフラグ
Z	ゼロフラグ	I	割り込み禁止フラグ
D	１０進モードフラグ	T	X修飾演算モードフラグ
V	オーバフローフラグ	N	ネガティブフラグ
imm	イミディエイトデータ	zz	ゼロページアドレス
hhll	一般ページアドレス	#	イミディエイトモード
MEM	メモリ	FLAG	メモリビット

A) レジスタ、フラグ代入文の展開例

表 F.2: レジスタ、フラグ代入文の展開例

分類	構造化記述言語	アセンブリ言語
bit処理	BIT_A0 = 0	CLB BIT_A0
	BIT_A0 = 1	SEB BIT_A0
フラグ設定	C = 0	CLC
	C = 1	SEC
	D = 0	CLD
	D = 1	SED
	I = 0	CLI
	I = 1	SEI
	T = 0	CLT
	T = 1	SET
	V = 0	CLV

分類	構造化記述言語	アセンブリ言語
データ転送命令	A = imm	LDA # imm
	X = imm	LDX # imm
	Y = imm	LDY # imm
	X = A	TAX
	A = X	TXA
	Y = A	TAY
	A = Y	TYA
	X = S	TSX
	S = X	TXS
	[S] = A	PHA
	[S] = P	PHP
	A = [S]	PLA
	P = [S]	PLP
演算命令	A = A + imm WITH_C	ADC # imm
	A = A − imm WITH_C	SBC # imm
	A = ++ A	INC A
	A = −− A	DEC A
	X = ++ X	INX
	X = −− X	DEX
	Y = ++ Y	INY
	Y = −− Y	DEY
	A = A & imm	AND # imm
	A = A |imm	ORA # imm
	A = A ^imm	EOR # imm
	A = A << imm	ASL A
	A = A >> imm	LSR A
	A = A << imm WITH_C	ROL A
	A = A >> imm WITH_C	ROR A

B) メモリ代入文の展開例

表 F.3: メモリ代入文の展開例

分類	構造化記述言語	アセンブリ言語
データ転送命令	A = [zz]	LDA zz
	X = [zz]	LDX zz
	Y = [zz]	LDY zz
	[zz] = imm	LDM # imm, zz
	[zz] = A	STA zz
	[zz] = X	STX zz
	[zz] = Y	STY zz
演算命令	A = A + [zz] WITH_C	ADC zz
	A = A − [zz] WITH_C	SBC zz
	[zz] = ++ [zz]	INC zz
	[zz] = −− [zz]	DEC zz
	A = A & [zz]	AND zz
	A = A | [zz]	ORA zz
	A = A ^ [zz]	EOR zz
	[zz] = [zz] << imm	ASL zz
	[zz] = [zz] >> imm	LSR zz
	[zz] = [zz] << imm WITH_C	ROL zz
	[zz] = [zz] >> imm WITH_C	ROR zz
	[zz] = ~ [zz]	COM zz
bit 処理	[FLAG] = 0	CLB FLAG
	[FLAG] = 1	SEB FLAG

C) アドレッシングモード代入文の展開例

表 F.4: アドレッシングモード代入文の展開例

分類	構造化記述言語	アセンブリ言語
ロード命令	A = [zz,X]	LDA zz,X
	A = [hhll]	LDA hhll
	A = [hhll,X]	LDA hhll,X
	A = [hhll,Y]	LDA hhll,Y
	A = [(zz,X)]	LDA (zz,X)

分類	構造化記述言語	アセンブリ言語
ロード命令	A = [(zz),Y]	LDA (zz),Y
	X = [zz,Y]	LDX zz,Y
	X = [hhll]	LDX hhll
	X = [hhll,Y]	LDX hhll,Y
	Y = [zz,X]	LDY zz,X
	Y = [hhll]	LDY hhll
	Y = [hhll,X]	LDY hhll,X
ストア命令	[zz,X] = A	STA zz,X
	[hhll] = A	STA hhll
	[hhll,X] = A	STA hhll,X
	[hhll,Y] = A	STA hhll,Y
	[(zz,X)] = A	STA (zz,X)
	[(zz),Y] = A	STA (zz),Y
	[zz,Y] = X	STX zz,Y
	[hhll] = X	STX hhll
	[zz,X] = Y	STY zz,X
	[hhll] = Y	STY hhll
加減算命令	A = A + [zz,X] WITH_C	ADC zz,X
	A = A + [hhll] WITH_C	ADC hhll
	A = A + [hhll,X] WITH_C	ADC hhll,X
	A = A + [hhll,Y] WITH_C	ADC hhll,Y
	A = A + [(zz,X)] WITH_C	ADC (zz,X)
	A = A + [(zz),Y] WITH_C	ADC (zz),Y
	A = A − [zz,X] WITH_C	SBC zz,X
	A = A − [hhll] WITH_C	SBC hhll
	A = A − [hhll,X] WITH_C	SBC hhll,X
	A = A − [hhll,Y] WITH_C	SBC hhll,Y
	A = A − [(zz,X)] WITH_C	SBC (zz,X)
	A = A − [(zz),Y] WITH_C	SBC (zz),Y
	[zz,X] = ++ [zz,X]	INC zz,X
	[hhll] = ++ [hhll]	INC hhll
	[hhll,X] = ++ [hhll,X]	INC hhll,X
	[zz,X] = −− [zz,X]	DEC zz,X
	[hhll] = −− [hhll]	DEC hhll
	[hhll,Y] = −− [hhll,Y]	DEC hhll,X

分類	構造化記述言語	アセンブリ言語
論理演算命令	A = A & [zz,X]	AND zz,X
	A = A & [hhll]	AND hhll
	A = A & [hhll,X]	AND hhll,X
	A = A & [hhll,Y]	AND hhll,Y
	A = A & [(zz,X)]	AND (zz,X)
	A = A & [(zz),Y]	AND (zz),Y
	A = A | [zz,X]	ORA zz,X
	A = A | [hhll]	ORA hhll
	A = A | [hhll,X]	ORA hhll,X
	A = A | [hhll,Y]	ORA hhll,Y
	A = A | [(zz,X)]	ORA (zz,X)
	A = A | [(zz),Y]	ORA (zz),Y
論理演算命令	A = A ^ [zz,X]	EOR zz,X
	A = A ^ [hhll]	EOR hhll
	A = A ^ [hhll,X]	EOR hhll,X
	A = A ^ [hhll,Y]	EOR hhll,Y
	A = A ^ [(zz,X)]	EOR (zz,X)
	A = A ^ [(zz),Y]	EOR (zz),Y
回転・シフト命令	[zz,X] = [zz,X] << imm	ASL zz,X
	[hhll] = [hhll] << imm	ASL hhll
	[hhll,X] = [hhll,X] << imm	ASL hhll,X
	[zz,X] = [zz,X] >> imm	LSR zz,X
	[hhll] = [hhll] >> imm	LSR hhll
	[hhll,X] = [hhll,X] >> imm	LSR hhll,X
	[zz,X] = [zz,X] << imm WITH_C	ROL zz,X
	[hhll] = [hhll] << imm WITH_C	ROL hhll
	[hhll,X] = [hhll,X] << imm WITH_C	ROL hhll,X
	[zz,X] = [zz,X] >> imm WITH_C	ROR zz,X
	[hhll] = [hhll] >> imm WITH_C	ROR hhll
	[hhll,X] = [hhll,X] >> imm WITH_C	ROR hhll,X

＊ 構造化記述言語と対応していないもの

MUL, DIV, BIT, TST, RRF, JSR, RTI, RTS, NOP, FST, SLW, WIT, STP, BRA, JMP

D) 2項演算代入文の展開例

表 F.5: 2項演算代入文の展開例

分類	構造化記述言語	アセンブリ言語
回転・シフト命令	[zz] = [zz] << 2	ASL zz ASL zz
	[zz] = [zz] << 2 WITH_C	ROL zz ROL zz
	[zz] = [zz] >> 2	LSR zz LSR zz
	[zz] = [zz] >> 2 WITH_C	ROR zz ROR zz
加減算命令	[zz] = [zz] + 4	LDA zz CLC ADC # 4 STA zz
	[zz] = [zz] + 4 WITH_C	LDA zz ADC # 4 STA zz
	[zz] = [zz] + [zz]	LDA zz CLC ADC zz STA zz
	[zz] = [zz] − 4	LDA zz SEC SBC # 4 STA zz
	[zz] = [zz] − 4 WITH_C	LDA zz SBC # 4 STA zz

分類	構造化記述言語	アセンブリ言語
減算命令	[zz] = [zz] − [zz]	LDA zz SEC SBC zz STA zz
乗除算命令	[zz] = [zz] ＊ 4	LDA zz JSR .mult_8 .BYTE 4 STA zz
	[zz] = [zz] ／ 4	LDA zz JSR .div_8 .BYTE 4 STA zz
	[zz] = [zz] % 4	LDA zz JSR .mod_8 .BYTE 4 STA zz
論理演算命令	[zz] = [zz] & 4	LDA zz AND # 4 STA zz
	[zz] = [zz] | 4	LDA zz ORA # 4 STA zz
	[zz] = [zz] ^ 4	LDA zz EOR # 4 STA zz

＊ .mult_8, .div_8, .mod_8 については、5.4注意事項の 2 を参照してください。

F.3.2 条件式の展開例

- IF 文、DO 文、FOR 文における条件式の展開例を示します。
  なお、ラベル名は次の展開例に対応しています。

  – if ～ (else) ～ endif

```
if   [ MEM ] == 2        ;       LDA    MEM
     JSR   out           ;       CMP    #2
endif                    ;       BNE    hhl1
                         ;       JSR    out
                         ; hhl1:
```

  – do ～ while

```
do                       ; hhl1:
     JSR   out           ;       JSR    out
while   [ MEM ] == 2     ;       LDA    MEM
                         ;       CMP    #2
                         ;       BEQ    hhl1
```

  – for ～ next

```
for   [ MEM ] == 2       ; hhl1:
     JSR   out           ;       LDA    MEM
next                     ;       CMP    #2
                         ;       BNE    hhl2
                         ;       JSR    out
                         ;       BRA    hhl1
                         ; hhl2:
```

表 F.6: フラグ条件式の展開例

構造化記述言語	アセンブリ言語	ロング分岐
・if ［FLAG］ == 1 ・while ［FLAG］ == 0 ・for ［FLAG］ == 1	BBC FLAG,hhll	BBS FLAG, .Z0 JMP hhll .Z0:
・if ［FLAG］ == 0 ・while ［FLAG］ == 1 ・for ［FLAG］ == 0	BBS FLAG,hhll	BBC FLAG, .Z0 JMP hhll .Z0:
・if C == 1 ・while C == 0 ・for C == 1	BCC hhll	BCS .Z0 JMP hhll .Z0:
・if C == 0 ・while C == 1 ・for C == 0	BCS hhll	BCC .Z0 JMP hhll .Z0:
・if Z == 1 ・while Z == 0 ・for Z == 1	BNE hhll	BEQ .Z0 JMP hhll .Z0:
・if Z == 0 ・while Z == 1 ・for Z == 0	BEQ hhll	BNE .Z0 JMP hhll .Z0:
・if N == 1 ・while N == 0 ・for N == 1	BPL hhll	BMI .Z0 JMP hhll .Z0:
・if N == 0 ・while N == 1 ・for N == 0	BMI hhll	BPL .Z0 JMP hhll .Z0:
・if V == 1 ・while V == 0 ・for V == 1	BVC hhll	BVS .Z0 JMP hhll .Z0:
・if V == 0 ・while V == 1 ・for V == 0	BVS hhll	BVC .Z0 JMP hhll .Z0:

表 F.7: メモリ条件式の展開例

分類	構造化記述言語	アセンブリ言語	ロング分岐
IF 文	if ［MEM］ == 2	LDA MEM CMP # 2 BNE hhll	LDA MEM CMP # 2 BEQ .Z0 JMP hhll .Z0:
	if ［MEM］ != 2	LDA MEM CMP # 2 BEQ hhll	LDA MEM CMP # 2 BNE .Z0 JMP hhll .Z0:
	if ［MEM］ > 2	LDA MEM CMP # 2 BEQ hhll BCC hhll	LDA MEM CMP # 2 BEQ .Z0 BCS .Z1 .Z0: JMP hhll .Z1:
	if ［MEM］ >= 2	LDA MEM CMP # 2 BCC hhll	LDA MEM CMP # 2 BCS .Z0 JMP hhll .Z0:
	if ［MEM］ < 2	LDA MEM CMP # 2 BCS hhll	LDA MEM CMP # 2 BCC .Z0 JMP hhll .Z0:
	if ［MEM］ <= 2	LDA MEM CMP # 2 BEQ .Z0 BCS hhll .Z0:	LDA MEM CMP # 2 BEQ .Z0 BCC .Z0 JMP hhll .Z0:

分類	構造化記述言語	アセンブリ言語	ロング分岐
DO文	while ［MEM］ == 2	LDA MEM CMP # 2 BEQ hhll	LDA MEM CMP # 2 BNE .Z0 JMP hhll .Z0:
	while ［MEM］ != 2	LDA MEM CMP # 2 BNE hhll	LDA MEM CMP # 2 BEQ .Z0 JMP hhll .Z0:
	while ［MEM］ > 2	LDA MEM CMP # 2 BEQ .Z0 BCS hhll .Z0:	LDA MEM CMP # 2 BEQ .Z0 BCC .Z0 JMP hhll .Z0:
	while ［MEM］ >= 2	LDA MEM CMP # 2 BCS hhll	LDA MEM CMP # 2 BCC .Z0 JMP hhll .Z0:
	while ［MEM］ < 2	LDA MEM CMP # 2 BCC hhll	LDA MEM CMP # 2 BCS .Z0 JMP hhll .Z0:
	while ［MEM］ <= 2	LDA MEM CMP # 2 BEQ hhll BCC hhll	LDA MEM CMP # 2 BEQ .Z0 BCS .Z1 .Z0: JMP hhll .Z1:

分類	構造化記述言語	アセンブリ言語	ロング分岐
FOR文	for ［MEM］ == 2	LDA MEM CMP # 2 BNE hhll2	LDA MEM CMP # 2 BEQ .Z0 JMP hhll2 .Z0:
	for ［MEM］ != 2	LDA MEM CMP # 2 BEQ hhll2	LDA MEM CMP # 2 BNE .Z0 JMP hhll2 .Z0:
	for ［MEM］ > 2	LDA MEM CMP # 2 BEQ hhll2 BCC hhll2	LDA MEM CMP # 2 BEQ .Z0 BCS .Z1 .Z0: JMP hhll2 .Z1:
	for ［MEM］ >= 2	LDA MEM CMP # 2 BCC hhll2	LDA MEM CMP # 2 BCS .Z0 JMP hhll2 .Z0:
	for ［MEM］ < 2	LDA MEM CMP # 2 BCS hhll2	LDA MEM CMP # 2 BCC .Z0 JMP hhll2 .Z0:
	for ［MEM］ <= 2	LDA MEM CMP # 2 BEQ .Z0 BCS hhll2 .Z0:	LDA MEM CMP # 2 BEQ .Z0 BCC .Z1 .Z0: JMP hhll2 .Z1:

表 F.8: レジスタ条件式の展開例

分類	構造化記述言語	アセンブリ言語	ロング分岐
A レジスタ	if A == [MEM]	CMP MEM BNE hhll	CMP MEM BEQ .Z0 JMP hhll .Z0:
	if A != [MEM]	CMP MEM BEQ hhll	CMP MEM BNE .Z0 JMP hhll .Z0:
	if A > [MEM]	CMP MEM BEQ hhll BCC hhll	CMP MEM BEQ .Z0 BCS .Z1 .Z0: JMP hhll .Z1:
	if A >= [MEM]	CMP MEM BCC hhll	CMP MEM BCS .Z0 JMP hhll .Z0:
	if A < [MEM]	CMP MEM BCS hhll	CMP MEM BCC .Z0 JMP hhll .Z0:
	if A <= [MEM]	CMP MEM BEQ .Z0 BCS hhll .Z0:	CMP MEM BEQ .Z0 BCC .Z0 JMP hhll .Z0:

分類	構造化記述言語	アセンブリ言語	ロング分岐
X レジスタ	if X == [MEM]	CPX MEM BNE hhll	CPX MEM BEQ .Z0 JMP hhll .Z0:
	if X != [MEM]	CPX MEM BEQ hhll	CPX MEM BNE .Z0 JMP hhll .Z0:
	if X > [MEM]	CPX MEM BEQ hhll BCC hhll	CPX MEM BEQ .Z0 BCS .Z1 .Z0: JMP hhll .Z1:
	if X >= [MEM]	CPX MEM BCC hhll	CPX MEM BCS .Z0 JMP hhll .Z0:
	if X < [MEM]	CPX MEM BCS hhll	CPX MEM BCC .Z0 JMP hhll .Z0:
	if X <= [MEM]	CPX MEM BEQ .Z0 BCS hhll .Z0:	CPX MEM BEQ .Z0 BCC .Z0 JMP hhll .Z0:

分類	構造化記述言語	アセンブリ言語	ロング分岐
Y レジスタ	if Y == [MEM]	CPY MEM BNE hhll	CPY MEM BEQ .Z0 JMP hhll .Z0:
	if Y != [MEM]	CPY MEM BEQ hhll	CPY MEM BNE .Z0 JMP hhll .Z0:
	if Y > [MEM]	CPY MEM BEQ hhll BCC hhll	CPY MEM BEQ .Z0 BCS .Z1 .Z0: JMP hhll .Z1:
	if Y >= [MEM]	CPY MEM BCC hhll	CPY MEM BCS .Z0 JMP hhll .Z0:
	if Y < [MEM]	CPY MEM BCS hhll	CPY MEM BCC .Z0 JMP hhll .Z0:
	if Y <= [MEM]	CPY MEM BEQ .Z0 BCS hhll .Z0:	CPY MEM BEQ .Z0 BCC .Z0 JMP hhll .Z0:

F.4 構造化命令の構文図

SRA74 で使用可能な構造化命令の文法を構文図により示します。
以下、説明において用いているおもな用語の意味について示します。

- 変数
  メモリ変数、メモリビット変数、レジスタ変数、レジスタビット変数、フラグ変数を総称して変数と呼んでいます。
- メモリ変数
  740 ファミリの各アドレッシングモードで参照される任意のメモリ、又はスタックのことです。記述する場合は [] 、又は{}で囲んで記述します。

[ZZ]	ゼロページ	[ZZ,X]	ゼロページ X
[ZZ,Y]	ゼロページ Y	[HHLL]	アブソリュート
[HHLL,X]	アブソリュート X	[HHLL,Y]	アブソリュート Y
[(ZZ,X)]	ゼロページインダイレクト X	[(ZZ),Y]	ゼロページインダイレクト Y
[S]	スタック		

- メモリビット変数
  740 ファミリのビットアドレッシングモードで参照される任意のビットのことです。記述する場合は [] 、又は{}で囲んで記述します。但し、この変数は.EQU 疑似命令により以下のように、参照する以前に定義しておく必要があります。

  例)

```
BITSYM    .EQU 1,1000H          ビットシンボルの定義
          if [ BITSYM ]         ビットシンボルの参照
                 :
          else
                 :
          endif
```

- レジスタ変数
  740 ファミリの各種レジスタのことです。記述する場合には [] 、又は {}で囲む必要はありません。そのまま記述してください。これは SRA74 では予約語として以下のような名前で用意されています。また、大文字/小文字を区別しませんので A 、a いずれでも有効です。

A	アキュムレータ	X	インデックスレジスタ X
Y	インデックスレジスタ Y	S	スタックポインタ
P	プロセッサステータスレジスタ		

- レジスタビット変数
  740 ファミリのアキュムレータビットアドレッシングモードで参照されるアキュムレータ内の任意のビットのことです。これは SRA74 では予約語として以下のような名前で用意されています。また、大文字/小文字を区別しませんので BIT_A0 、bit_a0 いずれでも有効です。

BIT_A0	アキュムレータのビット 0	BIT_A1	アキュムレータのビット 1
BIT_A2	アキュムレータのビット 2	BIT_A3	アキュムレータのビット 3
BIT_A4	アキュムレータのビット 4	BIT_A5	アキュムレータのビット 5
BIT_A6	アキュムレータのビット 6	BIT_A7	アキュムレータのビット 7

- フラグ変数

  740 ファミリのステータスレジスタ内のフラグのことです。これは SRA74 では予約語として以下のような名前で用意されています。また、大文字/小文字を区別しませんので C 、c いずれでも有効です。

C	キャリーフラグ	Z	ゼロフラグ
I	割り込み禁止フラグ	D	10 進モードフラグ
T	X 修飾演算モードフラグ	V	オーバフローフラグ
N	ネガティブフラグ		

- WITH_C

  キャリー付き演算指定のことです。これは SRA74 では予約語として用意されています。また、大文字/小文字を区別しませんので WITH_C 、with_c いずれでも有効です。

  例)

```
[ work ] = [ work ] << 2 with_c    work の内容をキャリーを含めて
                                   左に 2 回シフトする。
```

- EVER

  無限ループ指定のことです。これは SRA74 では予約語として用意されています。また、大文字/小文字を区別しませんので EVER 、ever いずれでも有効です。

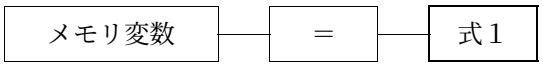
  例)

```
for ever                           無限ループの指定
   :
next
```

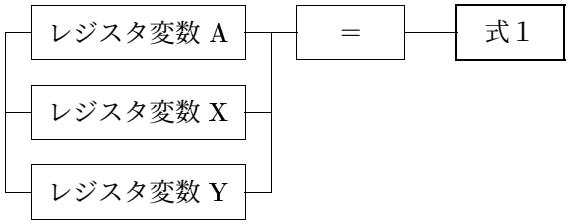
- 定数

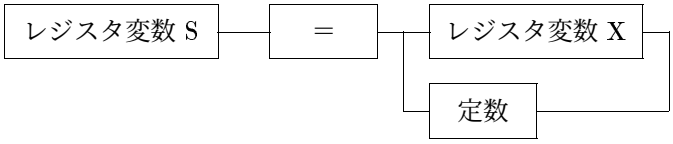
  数値定数、文字定数、記号定数、又はこれらを演算子で組み合わせものを総称して定数と呼んでいます。

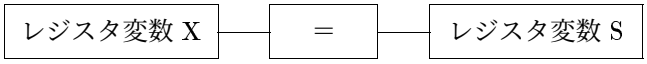
- 代入文
  - メモリ変数代入文

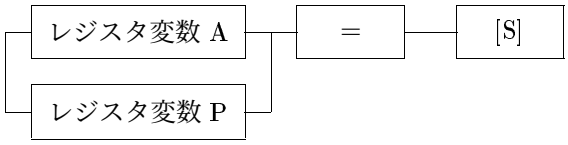


  - レジスタ変数代入文

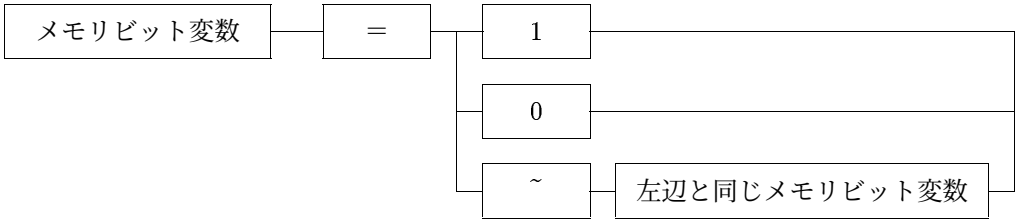




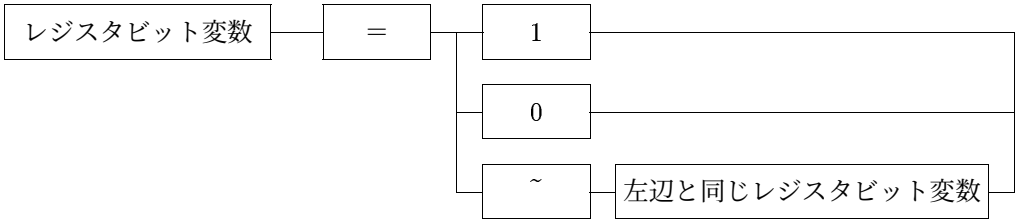




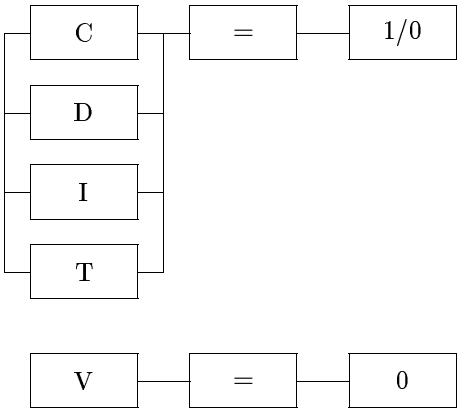
  - メモリビット変数代入文



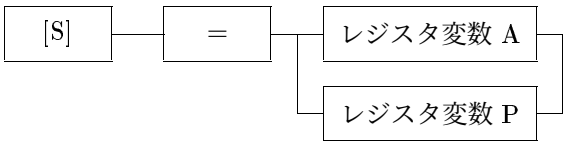
◦ レジスタビット変数代入文



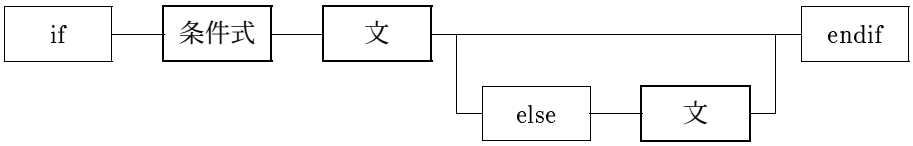
◦ フラグ変数代入文



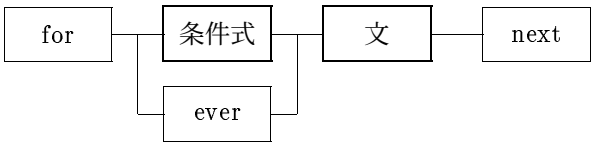
◦ スタックフレーム変数代入文



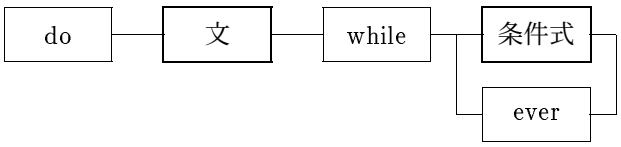
- if 文



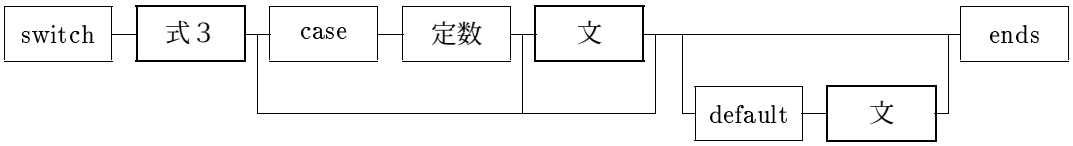
- for 文



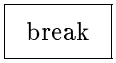
- do 文



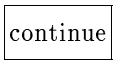
- switch 文



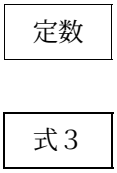
- break 文



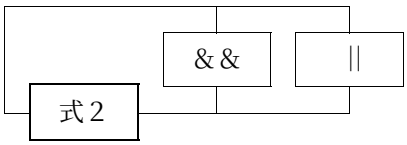
- continue 文

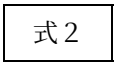


• 式 1



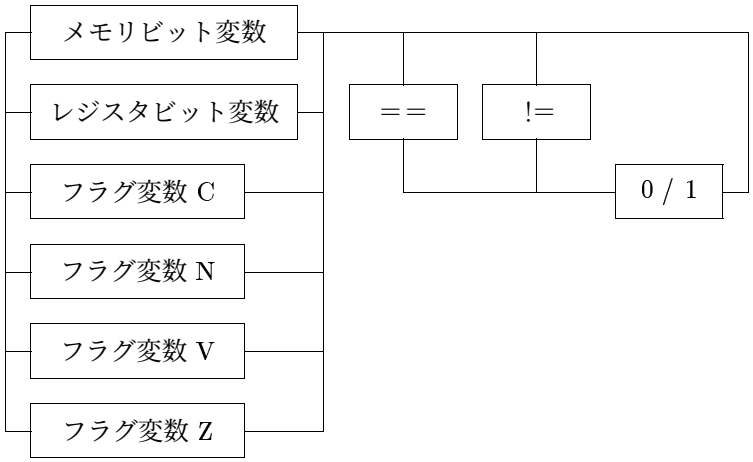
• 条件式

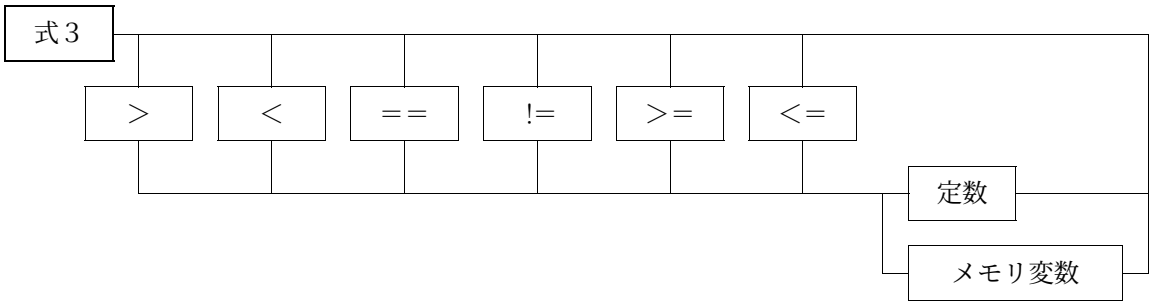




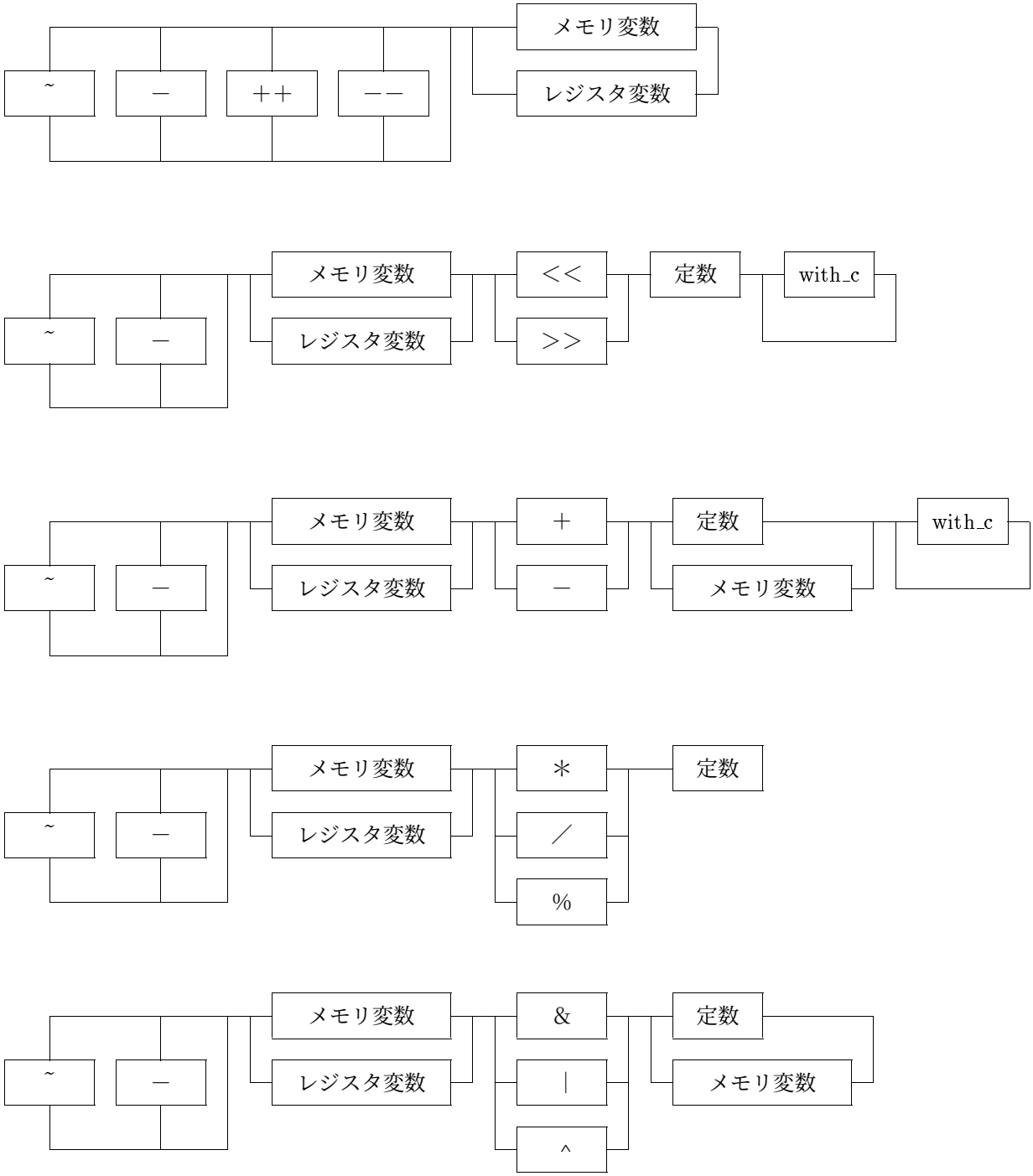
は 6 つまで連結可能

◦ 式 2



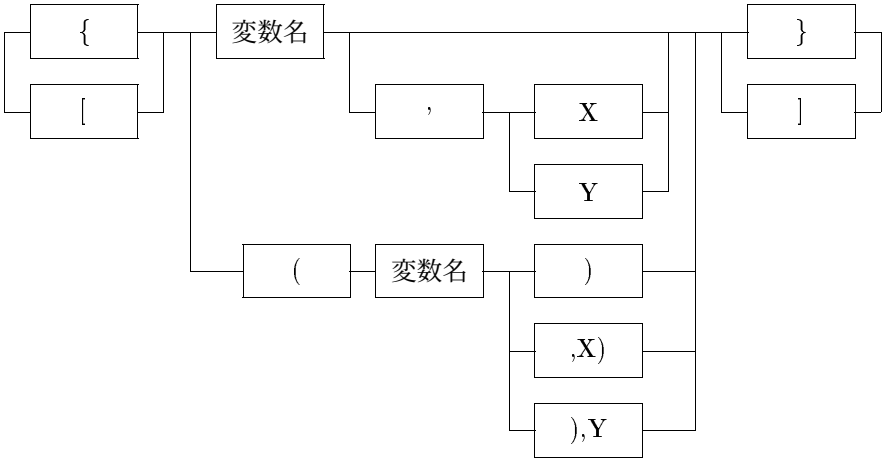


- 式 3

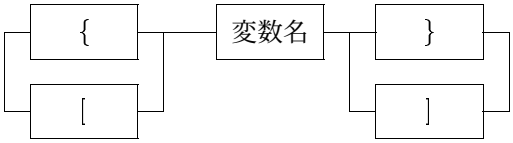

メモリ変数
~
－
＋＋
－－
レジスタ変数
メモリ変数
＜＜
定数
with_c
~
－
レジスタ変数
＞＞
メモリ変数
＋
定数
with_c
~
－
レジスタ変数
－
メモリ変数
メモリ変数
＊
定数
~
－
レジスタ変数
／
%
メモリ変数
&
定数
~
－
レジスタ変数
|
メモリ変数
^

- 変数

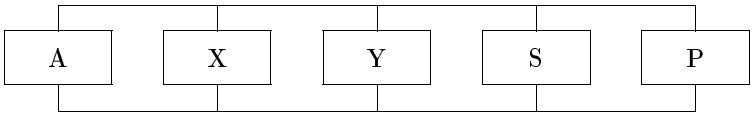
  - メモリ変数



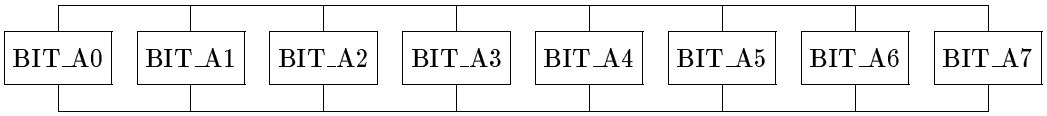
  - メモリビット変数



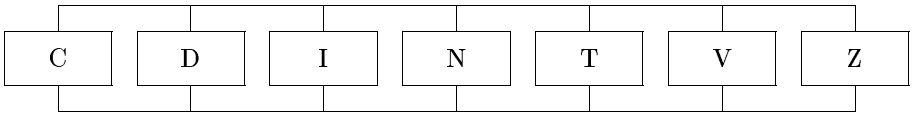
  - レジスタ変数



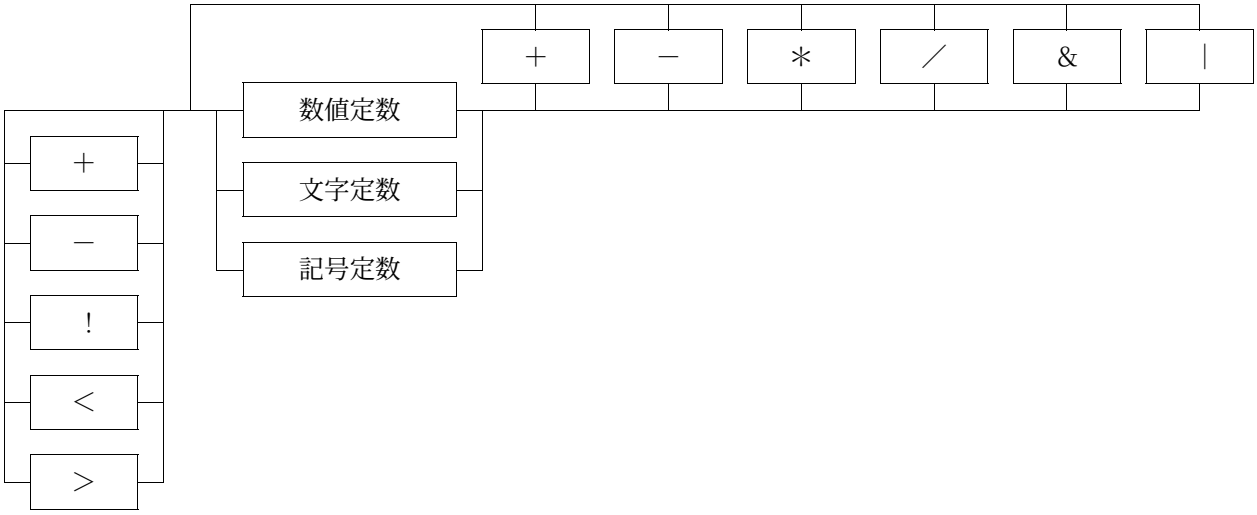
  - レジスタビット変数



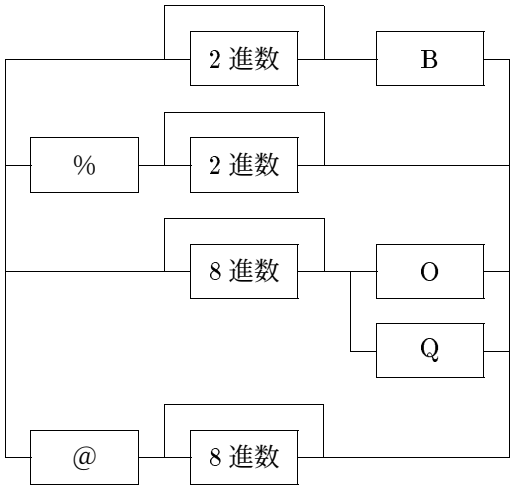
  - フラグ変数

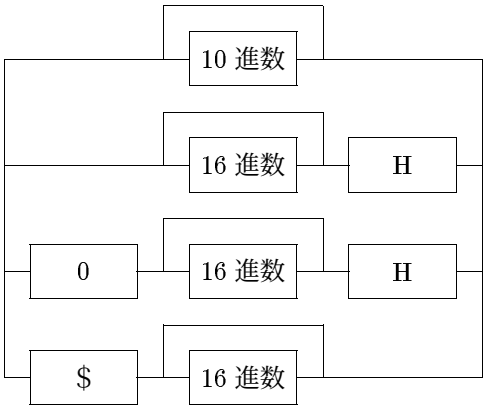


◦ 定数

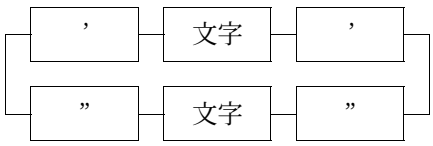

＋
－
＊
／
&
|
数値定数
文字定数
記号定数
＋
－
!
＜
＞

◦ 数値定数

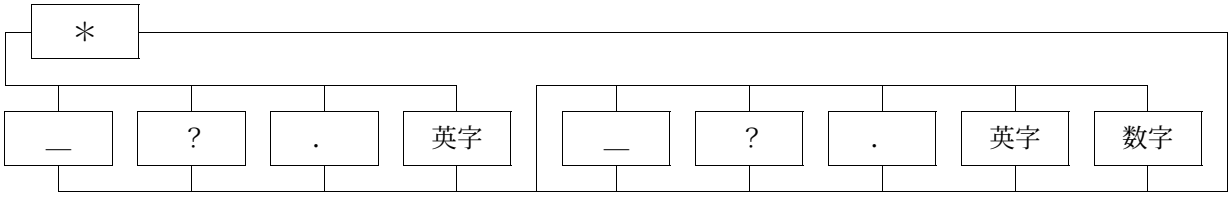

2 進数
B
%
2 進数
8 進数
O
Q
@
8 進数


10 進数
16 進数
H
0
16 進数
H
$
16 進数

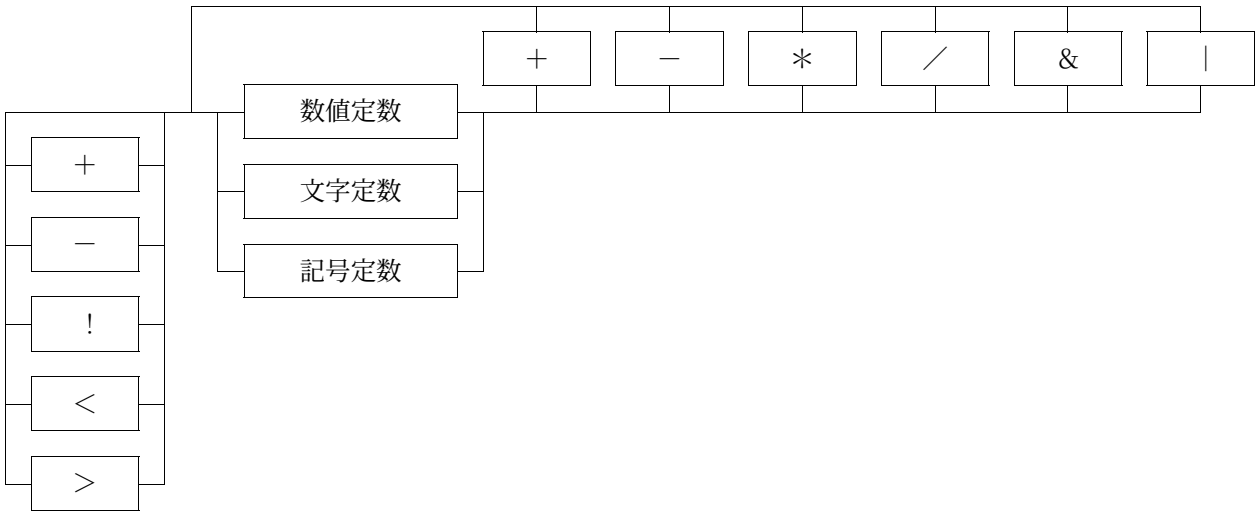
◦ 文字定数


’
文字
’
”
文字
”

◦ 記号定数



◦ 変数名



予約語一覧

記号

..0 ～..65535 ラベル
.ASSERT 疑似命令
.BEXT 疑似命令
.BLKB 疑似命令
.BYTE 疑似命令
.COL 疑似命令
.D0 ～.D65535 ラベル
.ELSE 疑似命令
.END 疑似命令
.ENDFUNC 疑似命令
.ENDIF 疑似命令
.ENDIO 予約疑似命令
.ENDM マクロ命令
.ENDPROC 予約疑似命令
.ENDRAM 予約疑似命令
.EQU 疑似命令
.ERROR 疑似命令
.EXITM マクロ命令
.EXT 疑似命令
.F0 ～.F65535 ラベル
.FUNC 疑似命令
.I0 ～.I65535 ラベル
.IF 疑似命令
.INCLUDE 疑似命令
.IO 予約疑似命令
.LIB 疑似命令
.LINE 疑似命令
.LIST 疑似命令
.LISTM 疑似命令
.LOCAL マクロ命令
.MACRO マクロ命令
.NLIST 疑似命令
.NLISTM 疑似命令
.OBJ 疑似命令
.ORG 疑似命令
.PAGE 疑似命令
.PMOD 疑似命令
.PROCINT 予約疑似命令
.PROCMAIN 予約疑似命令
.PROCSUB 予約疑似命令
.PROGNAME 予約疑似命令
.PUB 疑似命令
.RAM 予約疑似命令
.REPEAT マクロ命令
.REPEATC マクロ命令
.REPEATI マクロ命令
.RMOD 疑似命令
.S0 ～.S65535 ラベル
.SECTION 疑似命令
.SEXT 疑似命令
.SMOD 疑似命令
.VER 疑似命令
.WORD 疑似命令
.ZBEXT 疑似命令
.ZEXT 疑似命令
.ZMOD 疑似命令
??0 ～??65535 ラベル

A

A アキュムレータ
ADC ニーモニック
AND ニーモニック
ASL ニーモニック

B

BBC ニーモニック
BBS ニーモニック
BCC ニーモニック

C

D

E

F

I

J

L

M

N

O

P

R

S

T

V

W

X

Y

Z

第 2 部

740 ファミリ 用リンケージエディタ

LINK74 操作マニュアル

目 次

図 目 次

表 目 次

第 1 章

LINK74 操作マニュアルの構成

LINK74 操作マニュアルは、以下の章から構成されています。

- 第 2 章 概要
  LINK74 の基本的な機能と、LINK74 が生成するファイルについて紹介します。
- 第 3 章 セクションの機能
  LINK74 の基本的な操作単位であるセクションについて説明します。
- 第 4 章 操作方法
  LINK74 のコマンド入力方法について説明します。
- 付録 A エラーメッセージ一覧表
  LINK74 が表示するエラーメッセージについて、その内容と対策を一覧表で示します。

注意事項

本マニュアルに掲載されているプログラム例など、一部においてスペシャルページアドレッシングモードを表す ‘ ¥ ’ 記号を ‘ \ ’ で表していることがあります。これは、オペレーティングシステムによって表記が異なるためで、コードは同一ですのでいずれも使用できます。

第 2 章

概要

LINK74 は、SRA74 で生成したリロケータブルファイルをライブラリファイルとリンクし、740 ファミリ 用機械語データファイルを生成します。

2.1 機能

LINK74 は、740 ファミリ用デバッガと共に利用できます。また、これらの各機能を十分生かすため以下の機能を備えています。

1. 複数のリロケータブルファイル中の同一セクション[1]名を持つ領域を連結して配置します。
2. セクションの配置順序やセクション単位ごとの開始アドレスを指定できます。
3. LIB74 によって作成したライブラリファイルを利用できます。
4. デバッグ時に便利なマップファイルを生成します。
5. デバッグを行うために必要なシンボルファイルを生成します。

[1] セクションとは、プログラムを構成する ROM 領域や RAM 領域のような物理的に異なる性質をもった要素単位を指します。詳細については第 4 章を参照ください。

2.2 生成ファイル

LINK74 では、以下の 3 種類のファイルを生成します。

1. 機械語データファイル (以下 HEX ファイルと呼びます)
   - インテル 16 進形式で出力します。
   - ファイル属性は、.HEX です。
2. マップファイル (以下 MAP ファイルと呼びます)
   - 各セクションが最終的に配置されたアドレス情報を示します。
   - このファイルはプリンタへ出力して、デバッグ時や各セクションのメモリ容量の把握のためにお使いください。
   - MAP ファイルは、コマンドパラメータ “–M” を指定した時に出力します。
   - ファイル属性は、.MAP です。
3. シンボルファイル (以下 SYM ファイルと呼びます)
   - デバッグを行う時に必要な各種情報を含んだファイルです。
   - このファイルはリスト形式になっていないので、プリント出力は行わないでください。
   - SYM ファイルは、コマンドパラメータ “–S” を指定した時に出力します。
   - ファイル属性は、.SYM です。

2.3 MAP ファイルの構成

図 2.1に、MAP ファイルの出力例を示します。MAP ファイルは、以下の情報を示しています。

1. どのリロケータブルファイルからどれだけのデータがリンクされたかについてのセクション単位での情報。ここには、次の情報を出力しています。
   - ATR 部 : 相対属性か絶対属性[2]かを示します。REL は相対、ABS は絶対をそれぞれ示しています。
   - TYPE 部 : RAM 領域か ROM 領域かを示します。
   - START 部 : 開始アドレスを示します。
   - LENGTH 部 : 領域の大きさをバイト数で示します。
   - ライブラリファイルをリンクした場合、ライブラリファイル名とリロケータブルファイル名の両方が示されます。リロケータブルファイル名は、() 内に表示します。
2. グローバルラベルリスト
   プログラム中のグローバルラベル[3]とその絶対番地を示します。この部分は、コマンドパラメータ “–MS” を指定した時のみ出力します。
3. グローバルシンボルリスト
   プログラム中のグローバルシンボル[4]とその絶対番地を示します。この部分は、コマンドパラメータ “–MS” を指定した時のみ出力します。
4. グローバルビットシンボルリスト
   プログラム中のグローバルビットシンボル、[5]ビット値、及び絶対番地を示します。この部分は、コマンドパラメータ “–MS” を指定した時のみ出力します。

[2]アセンブリ言語ソース中に、疑似命令.ORG(または＊＝) により開始アドレス指定を行ったものが絶対属性になります。
[3]疑似命令.PUB で宣言したラベルを指します。
[4]疑似命令.PUB で宣言したシンボルを指します。
[5]疑似命令.PUB で宣言したビットシンボルを指します。

```
SECTION           FILENAME          ATR.  TYPE  START  LENGTH

WORKRAM           MAIN.R74          ABS   RAM   0000   0080
                  SUB.R74           REL   RAM   0080   0100
                  UTIL.LIB          REL   RAM   0180   0008
                  (CALC.R74)
PROM              MAIN.R74          REL   ROM   C000   1800
                  SUB.R74           REL   ROM   D800   1500
                  UTIL.LIB          REL   ROM   ED00   0820
                  (CALC.R74)
DROM              MAIN.R74          REL   ROM   F520   0023
                  SUB.R74           REL   ROM   F544   0030

GLOBAL LABEL INFORMATION

ADCNT           0030   COUNT           009C   DATA0           00A4
DATA1           00A6   MAIN            C000   TIME            00C6

GLOBAL SYMBOL INFORMATION

GLOBAL BIT SYMBOL INFORMATION
```

図 2.1: MAP ファイル出力例

第 3 章

セクションの機能

3.1 セクションの役割

アセンブリ言語で書かれたプログラムは、一般に RAM 領域、プログラム領域及び固定データ領域から構成されます。SRA74 でソースファイルをアセンブルするとリロケータブルファイルが生成されますが、リロケータブルファイルはこのような領域を 1 個以上含んでいます。セクションは、このような領域を表す単位です。以下では具体的な例を示して、セクションの内容とその使い方を説明します。

最も簡潔な例として 2 つのリロケータブルファイル MAIN.R74 と SUB.R74 を想定します。各リロケータブルファイルは図 3.1のように、それぞれ RAM 領域、プログラム領域、固定データ領域を持っています。

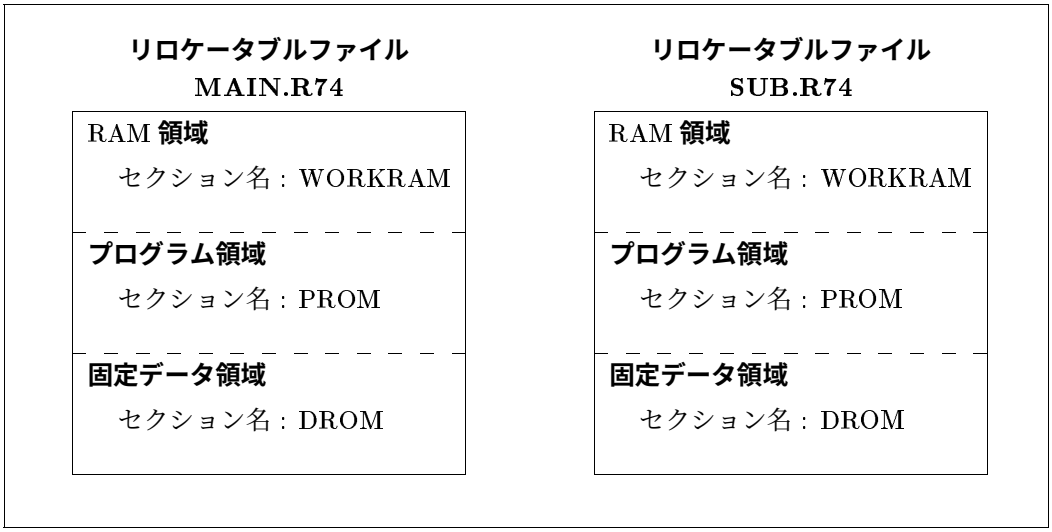


図 3.1: リロケータブルファイル構成図

これらのリロケータブルファイルを図 3.2のようなメモリ空間に配置したい場合、予めアセンブラ疑似命令.SECTION により結合したいセクションに同一の名前を指定しておきます。こうすることにより、各セクションはリンク時に連続した領域に配置されます。LINK74 では各セクションの開始アドレスをリンク時に指定できます。

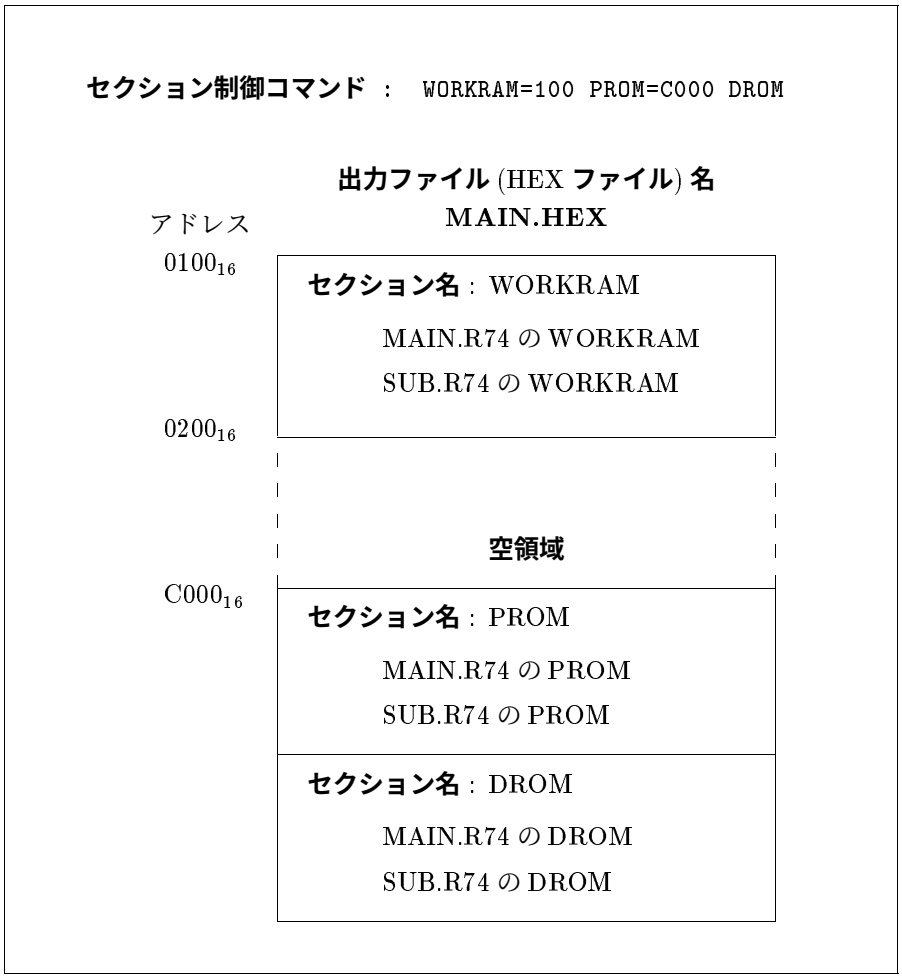


図 3.2: システムメモリマップ

以上のように、LINK74 を使用すればリンク時のコマンドによってシステムの最終的なアドレスに対応した機械語データを作成することができます。

3.2 セクションの属性

各セクションは、アドレスの配置に関する情報 (再配置可能か固定か) と、物理的な配置に関する情報 (ROM か RAM か) の 2 つの情報を持っています。前者をアドレス属性、後者を物理属性と呼びます。以下では、各属性の内容について説明します。

3.2.1 アドレス属性

アドレス属性には、相対属性と絶対属性の 2 つの属性があります。これらの属性は、ソースプログラムの該当セクション中の疑似命令.ORG (または＊＝) の有無によって決定します。各属性の特徴について以下に示します。

1. 相対属性
   - セクション内に疑似命令.ORG がない場合、そのセクションは相対属性になります。相対属性のセクションは再配置可能なセクションで、リンク時に開始アドレスを指定することができます。
2. 絶対属性
   - セクション内に疑似命令.ORG が書かれている場合、そのセクションは絶対属性になります。絶対属性のセクションは、疑似命令.ORG で指定した固定のアドレスに配置されます。
   - 絶対属性のセクションは、リンク時に開始アドレスの指定を行うことはできません。

複数のリロケータブルファイルに存在する同一名セクションは、異なるアドレス属性を持つことができます。

3.2.2 物理属性

物理属性には、ROM 属性と RAM 属性の 2 つの属性があります。これらの属性は、セクションが配置される領域の物理的性質を示しています。物理属性の特徴を以下に示します。

1. ROM 属性
   - アセンブリ言語ソース中にコードを発生するステートメント (LDA 等の命令や疑似命令.BYTE 等) が記述されているセクションが ROM 属性になります。
   - ROM 属性のセクションは、リンクの結果 HEX ファイルを生成します。
   - アドレス属性との混同を避けるためこの属性を ROM タイプと呼びます。
2. RAM 属性
   - RAM 属性のセクションは、リンクの結果 HEX ファイルを生成しません。
   - アセンブリ言語ソース中に、領域確保疑似命令.BLKB を記述しているセクションが RAM 属性になります。
   - アドレス属性との混同を避けるためこの属性を RAM タイプと呼びます。

同一名のセクションは、すべて同一の物理属性でなければいけません。

3.2.3 予約セクション

LINK74 は、セクション Z 、S 、P 、R を予約セクションとして扱います。以下で各々の予約セクションについて説明します。

1. Z セクション

   Z セクションは、740 ファミリのゼロページ (00_{16}番地から FF_{16}番地) に配置される RAM 属性のセクションです。SRA74 の疑似命令.SECTION でセクション名 'Z'(小文字でも可) を宣言した場合、または疑似命令.ZMOD を宣言した場合、そのセクションは Z セクションになります。

   Z セクションが相対属性の場合、LINK74 のコマンド入力で先頭アドレスを指定することができます。先頭アドレスの指定を省略した場合は、00_{16}番地から配置します。

2. S セクション

   S セクションは、740 ファミリのスペシャルページ ($FF00_{16}$番地から $FFFF_{16}$ 番地) に配置される ROM 属性のセクションです。SRA74 の疑似命令.SECTION でセクション名 'S'(小文字でも可) を宣言した場合、または疑似命令.SMOD を宣言した場合、そのセクションは S セクションになります。

   S セクションが相対属性の場合、LINK74 のコマンド入力で先頭アドレスを指定することができます。先頭アドレスの指定を省略した場合は、$FF00_{16}$番地から配置します。

3. P セクション

   P セクションは、プログラムまたは固定データを持つ ROM 属性のセクションです。プログラム領域内 (00_{16}番地から $FFFF_{16}$番地) のどの領域にも配置可能です。SRA74 の疑似命令.SECTION でセクション名 'P'(小文字でも可) を宣言した場合、または疑似命令.PMOD を宣言した場合、そのセクションは P セクションになります。

   P セクションが相対属性の場合、コマンドの指定を省略すると 00_{16}番地から配置します。

4. R セクション

   R セクションは、RAM 属性のセクションです。プログラム領域内 (00_{16}番地から $FFFF_{16}$番地) のどの領域にも配置可能です。SRA74 の疑似命令.SECTION でセクション名 'R'(小文字でも可) を宣言した場合、または疑似命令.RMOD を宣言した場合、そのセクションは R セクションになります。

   R セクションが相対属性の場合、コマンドの指定を省略すると 00_{16}番地から配置します。

5. E セクション

   E セクション内で定義されたシンボルおよび、セクション内のソース行デバッグ情報をリロケータブルファイルに出力されません。SRA74 の疑似命令.SECTION でセクション名'E'（小文字でも可）を宣言した場合、そのセクションは E セクションになります。E セクションは必ず疑似命令.ORG で予めセクションの配置アドレスを指定しておいてください。また、SRA74 を実行するさいに必ず-BANK オプションを指定してアセンブルしてください。コマンドオプションが指定されていない場合、E セクションの定義行でワーニングが出力されます。

3.3 セクションの基本機能

1. 同一名のセクションはリンク時に、連続して配置されます。同一名セクション間に他の名前のセクションが配置されることはありません。

2. 同一名セクション内での配置順序は、リンクコマンド中のリロケータブルファイル名の指定順序に従います。

第 4 章

操作方法

4.1 起動方法

LINK74 を実行するためには、以下の情報 (入力パラメータ) を入力する必要があります。

1. リロケータブルファイル名 (必須項目)
2. ライブラリファイル名
3. セクション制御情報
4. コマンドパラメータ

LINK74 はこれらの情報を入力する方法として、操作環境に応じた以下の 3 通りの方式を用意しています。

1. 対話式入力
2. コマンド行入力
3. コマンドファイル入力

これらの入力方式は、同一の入力パラメータを使用します。又、どの方式を用いても同一のコマンドが実行できます。4.2 節では、この入力パラメータについて説明し、4.3 節で、各入力方式について具体的な例を示しながら説明します。

4.2 入力パラメータ

4.2.1 リロケータブルファイル名

1. リロケータブルファイル名は必ず入力してください。
2. リロケータブルファイルは、属性.R74 を持ったものだけを扱います。コマンド入力では、ファイル属性を省略することができます。
3. ファイル名は、ディレクトリパスが指定可能です。ファイル名のみを記述した場合、カレントドライブのカレントディレクトリ中のファイルを処理します。
4. 最初に指定したリロケータブルファイル名が出力ファイル名になります。コマンドパラメータ “–F” で出力ファイル名を指定した場合は、コマンドパラメータ “–F” を優先します。
5. 最初に指定したリロケータブルファイルのディレクトリに出力ファイルを出力します。コマンドパラメータ “–O” でディレクトリを指定した場合は、コマンドパラメータ “–O” を優先します。

4.2.2 ライブラリファイル名

1. ライブラリファイル名は省略できます。
2. ライブラリファイルは、属性.LIB を持ったものだけを扱います。コマンド入力の時は、ファイル属性を省略することができます。
3. ファイル名には、パス指定ができます。ファイル名のみを指定した場合、カレントディレクトリのファイルを処理します。
4. ライブラリファイルは、リロケータブルファイル中に解決されないグローバルラベル又はグローバルシンボルがあった場合のみ参照します。

4.2.3 セクション制御

1. セクション情報は省略できます。この場合、読み込んだリロケータブルファイル中で出会ったセクション順に配置されます。
2. セクション情報の指定は、相対属性のセクションに対して行って下さい。絶対属性のセクションは、セクション指定の有無に関わらず疑似命令.ORG で指定した固定のアドレスから配置されます。
3. セクション配置の指定は、アドレスの下位に配置するセクションから順に、セクション名を空白で区切って記述してください。
4. 各セクションの開始アドレスは、該当セクション名の後に ‘=’ (イコール) に続けてアドレスを記述してください。アドレスは、16 進数で指定してください。この時、先頭の ‘0’ と最後の ‘H’ は不用です。

5. セクション名は、大文字/小文字を区別します。

6. 相対属性のセクションの開始アドレスを省略した場合、0000_{16} 番地から配置します。なお予約セクション S は、$FF00_{16}$から配置します。

7. セクションのアドレスオーバーラップはエラーになります。
   ただし、コマンドパラメータ “–A” を指定すると、絶対アドレスをオーバーラップさせることができます。これにより、SFR 領域などの絶対アドレスラベルを外部参照指定せず、疑似命令 “.INCLUDE” により、複数のプログラムファイルに読み込むことが可能になります。

4.2.4 コマンドパラメータ

コマンドパラメータは、リンクの出力ファイル、バージョン確認の有無等の制御を行います。表 4.1に、コマンドパラメータの内容を示します。

表 4.1: コマンドパラメータ一覧表

コマンドパラメータ	内容
–A	同名の絶対属性セクションのオーバーラップを許可します。グローバルなメモリ領域を共有結合する場合に利用できます。
–BANK	アドレス空間の上限を FFFFH から 1FFFFH に拡張します。
–F	出力ファイル名を指定します。指定の書式は次のようになります。 –Fsample
–M	MAP ファイルの出力を行います。(セクション情報のみ)
–MS	MAP ファイルを、グローバルラベル、グローバルビットシンボル、及びグローバルシンボルリスト付きで出力します。
–N	ソースファイル中の疑似命令.OBJ 、.LIB により指定された.R74 、.LIB ファイルの参照指定情報を無視します。
–O	出力ファイルのディレクトリを指定します。 指定の書式は、次のようになります。 –OC:¥USR¥WORK C ドライブの ¥USR¥WORK を出力ディレクトリに指定します。
–S	SYM ファイルの出力を行います。
–V	リロケータブルファイル間のバージョン整合性の確認を行います。ただし、確認を行うためには、アセンブリ言語ソース中で疑似命令.VER によりバージョンの指定を行っておく必要があります。

4.3 入力方法

4.3.1 対話式入力

対話式入力の特徴を以下に示します。

1. リロケータブルファイル、ライブラリファイル、セクション制御コマンド、コマンドパラメータの順に対話式に入力する方式です。
2. リロケータブルファイルやセクションの数が少ないときや、試行錯誤的にアドレス設定を試したい場合などに便利な方式です。
3. コマンド行入力やコマンドファイル入力でコマンドの数が不足している時は、自動的にこの方式に移行します。

対話式入力は、オペレーティングシステムのプロンプト状態で LINK74<RET>と入力することにより起動します。LINK74 は、起動後次のような画面を出力します。

```
A>LINK74 <RET>
740 Family LINKER V.4.00.00
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Relocatable files (.R74) >>
```

図 4.1: LINK74 起動画面

図 4.1の最後の行が、リロケータブルファイルの入力待ちを示しています。 >>に続いて、リンク対象のリロケータブルファイルを入力してください。対話式入力では、このように順次ライブラリファイル名、セクション制御、コマンドパラメータの入力待ちとなりますので、図 4.2のように入力していってください (複数のファイルを入力する場合、各ファイル名はスペース又はタブで区切ってください。ライブラリファイル名入力、セクション情報、コマンドパラメータについても同様です) 。

```
A>LINK74 <RET>
740 Family LINKER V.4.00.00
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Relocatable files (.R74) >> MAIN SUB<RET>
Libraries         (.LIB) >> UTIL1 UTIL2<RET>
Section information      >> WORKRAM=100 PROM=C000 DROM<RET>
Command parameter        >> -O\WORK -M -S<RET>
```

図 4.2: 対話式入力時の画面

4.3.2 コマンド行入力

コマンド行入力の特徴を以下に示します。

1. オペレーティングシステムのコマンド入力状態ですべてのコマンド入力を行う方式です。
2. OS によってはコマンドラインの文字数が制限されているためリロケータブルファイルやセクションの数が少ない場合に使用できます。
3. バッチファイルや、make ファイル中で実行コマンドを記述する場合に使用できます。
4. 入力パラメータの 4 種類の情報は、‘,’(カンマ) で区切って入力してください。図 4.2と同じコマンドをコマンド行で入力した例を図 4.3に示します。
5. ライブラリ名以降のパラメータが不用な場合も、カンマは必ず入力してください。図 4.3の例で、ライブラリが必要ない場合は図 4.4のようになります。
6. 特別な場合として、コマンドパラメータを省略する場合のみコマンドパラメータがないことを明確に示すために 2 つのカンマが必要になります。図 4.4でコマンドパラメータを省略した例を図 4.5に示します。
7. 入力パラメータの数が不足している時は、自動的に対話式入力状態に移行します。

```
A>LINK74 MAIN SUB, UTIL1 UTIL2, WORKRAM=100 PROM=C000 DROM, -O\WORK -M -S<RET>
```

図 4.3: コマンド行入力例 1

```
A>LINK74 MAIN SUB,, WORKRAM=100 PROM=C000 DROM, -O\WORK -M -S<RET>
```

図 4.4: コマンド行入力例 2(ライブラリ名の省略)

```
A>LINK74 MAIN SUB,, WORKRAM=100 PROM=C000 DROM,,<RET>
```

図 4.5: コマンド行入力例 3(コマンドパラメータの省略)

4.3.3 コマンドファイル入力

コマンドファイル入力の特徴を以下に示します。

1. リンクコマンドを予めエディタによりコマンドファイルとして作成し、起動時にこのファイル名を指定する方式です。
2. コマンドの文字数が多くコマンド行入力が使用できない場合に便利な方式です。
3. コマンドファイル名の指定は、オペレーティングシステムから LINK74 を起動する時に図 4.6のように、ファイル名の前に '@' をつけて指定します。図 4.6の場合、ファイル名 CMD.DAT の内容がコマンドとして実行されます。
4. コマンドファイルに記述する内容は、コマンド行入力と同じです (ただし、LINK74 の起動を指示する LINK74 の部分は不用) 。改行コードは無視しますので長いコマンドは複数行に分けて記述できます。図 4.5をコマンドファイルで記述すると図 4.7のようになります。
5. コマンドパラメータが不足している場合、LINK74 は対話式入力状態に移ります。例えば、図 4.7の最後の行の 2 つ目のカンマがない場合 LINK74 は、コマンドパラメータの対話式入力画面になります。

```
A>LINK74 @CMD.DAT<RET>
```

図 4.6: コマンドファイルの指定

```
MAIN SUB
,
,WORKRAM=100 PROM=C000 DROM
,,
```

図 4.7: コマンドファイル記述例

4.4 エラー

4.4.1 エラーの種類

LINK74 実行時に発生するエラーは、以下の原因によるものがあります。

1. オペレーティングシステムに関するエラー
   ディスクやメモリ容量の不足等、LINK74 を実行するオペレーティングシステム環境に関わるエラーです。付録 A エラーメッセージ一覧表を参照の上、オペレーティングシステムのコマンドにより対応してください。

2. LINK74 のコマンド行入力に関するエラー
   LINK74 起動時のコマンド行入力に関わるエラーです。本章の内容を確認の上コマンドを再入力してください。

3. リンク対象のリロケータブルファイルの内容に関するエラー
   グローバルラベルの 2 重定義、外部ラベルの未定義等のリロケータブルファイルの内容に関わるエラーです。該当箇所のソースプログラム等を確認して、必要があればアセンブルから再実行してください。

4. LINK74 の機能に関するエラー
   SRA74 、LIB74 とのバージョン不整合等によるエラーです。不明な点については当社までご連絡ください。

LINK74 は、図 4.8の形式でエラー内容を画面に表示します。付録 A のエラー番号順のエラー一覧表を参照の上処理してください。

```
A>LINK74 MAIN SUB,,WORKRAM=100 PROM=C000 DROM,,
740 Family LINKER V.4.00.00
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

now processing pass 1
processing "MAIN.R74"
ERROR NO.2: Out of heap space

A>
```

図 4.8: エラー表示例

4.4.2 オペレーティングシステムへの戻り値

オペレーティングシステムのバッチファイル等に実行コマンドを記述する場合、実行結果に応じて処理の内容を変えたい場合があります。LINK74 では、実行結果を表 4.2の 5 つのエラーレベルに分けてオペレーティングシステムに返すようにしています。

表 4.2: エラーレベル

エラーレベル	実行結果の内容
0	正常終了
1	リンク対象のリロケータブルファイルの内容に関するエラー
2	LINK74 のコマンド入力に関するエラー
3	オペレーティングシステムに関するエラー
4	LINK74 の機能に関するエラー

4.5 環境変数

LINK74 は、ライブラリファイルを読み込む際に環境変数 LIB74 を参照します。ライブラリファイルが存在するディレクトリパス名を環境変数 LIB74 に設定することにより、コマンド入力においてパス名の指定を省略することができます。

環境変数の設定方法の詳細はお手持ちの OS のマニュアルを参照下さい。

付録 A

エラーメッセージ一覧表

表 A.1: エラーメッセージ一覧表

エラー番号	エラーメッセージ	エラー内容と対策
0	xxx file not found	入力指定されたファイルが見つかりません。入力指定には、疑似命令.OBJ 又は.LIB によるものも含みます。 ⟹ ファイル名を正しく入力してください。
1	Invalid command input	入力コマンドのパラメータ数が 5 個以上か、コマンドの入力パラメータが 2048 文字以上です。 ⟹ 上記の範囲以内で再入力してください。
2	Out of heap space	リンカが動作するために必要なメモリが不足しています。 ⟹ パブリックシンボル数を減らしてください。
3	Invalid section information	セクション情報の指定が誤っています。 ⟹ “セクション名=アドレス” の形式で再入力してください。
4	Invalid parameter input ”xxx”	コマンドパラメータの指定が誤っています。 ⟹ 正しく入力してください。
5	Non relocatable file name	リロケータブルファイル名の入力がありません。 ⟹ ファイル名を入力してください。
6	Internal error	LINK74 の内部エラーが発生しました。 ⟹ LINK74 購入先に御連絡ください。

エラー番号	エラーメッセージ	エラー内容と対策
7	xxx relocatable format is mismatch	.R74 ファイルのリロケータブル形式バージョンが違います。 $\Longrightarrow$ このエラーは、ご使用中のアセンブラ又はライブラリアンとリンカの不整合が生じた時に発生します。リンク対象の.R74 及び.LIB ファイルを LINK74 と同じバージョンの SRA74 又は LIB74 で作成してください。
8	Program version is different	疑似命令.VER で宣言されているプログラムバージョンが一致していません。 $\Longrightarrow$ リンク対象のリロケータブルファイルの疑似命令.VER によるバージョン指定を一致させるか、コマンドパラメータ “–V” の指定を解除してください。
9	Unresolved label ”xxx” in xxx	該当セクション内で、外部参照宣言されたラベル又はシンボルが定義されていません。 $\Longrightarrow$ 該当ラベル又はシンボルがパブリック宣言されているプログラムをリンクしてください。
10	”xxx” is multiple defined in xxx. others in xxx	該当ラベル又はシンボルが二重定義です。 $\Longrightarrow$ 該当ラベル又はシンボル名を変更してください。
11	Location overlap. SECTION=xxx ADDRESS=xxx in xxx	該当セクションのアドレス空間がオーバーラップしています。 $\Longrightarrow$ 該当セクションのアドレス配置を確認して、アドレスオーバラップが発生しないようにしてください (絶対属性のセクションの場合は、ソースファイル中の疑似命令.ORG を変更してください) 。
12	SECTION xxx is an absolute	絶対属性のセクションに対して、セクション制御コマンドで先頭アドレスが指定されています。 $\Longrightarrow$ コマンド入力でのアドレス指定をやめるか、該当セクションを相対属性に変更してください。

エラー番号	エラーメッセージ	エラー内容と対策
14	Can't find SECTION xxx	該当セクションが存在しません。 $\Longrightarrow$ セクション情報を正しく指定してください (セクション名は大文字/小文字を区別しますのでご注意ください) 。
15	Can't create xxx	該当ファイルが生成できません。 $\Longrightarrow$ コマンドパラメータ "–O" の指定を確認して再入力してください。
16	File seek error xxx	該当ファイルがシークできません。 $\Longrightarrow$ このエラーは、オペレーティングシステムにかかわるエラーです。一般にディスク装置のハードウェア的な異常による場合が多いと考えられます。
17	Expression value is out of range. SECTION=xxx ADDRESS=xxx OFFSET=xxx	該当箇所の演算結果が制限値を超えています (エラー箇所の情報として、セクション名、絶対アドレス、セクションの先頭からのオフセットを表示します) 。 $\Longrightarrow$ 制限値を越えないよう、プログラムを変更してください。
18	Out of disk space	ファイルを出力するためのディスク領域が不足しています。 $\Longrightarrow$ ディスク上に空き領域を作ってください。
19	Relative jump is out of range. SECTION=xxx ADDRESS=xxx OFFSET=xxx	該当箇所の相対ジャンプ先アクセス対象に届きません。 $\Longrightarrow$ ジャンプ先のラベルが、範囲内にくるようにプログラムを変更してください (エラー箇所の情報として、セクション名、絶対アドレス、セクションの先頭からのオフセットを表示します) 。

エラー番号	エラーメッセージ	エラー内容と対策
22	Out of maximum program size	プログラム領域が最大領域 64K バイト ($FFFF_{16}$) を超えました。 $\Longrightarrow$ プログラム容量を縮小してください。
23	Section type mismatch in SECTION xxx	該当セクションで ROM タイプと RAM タイプが混在しています。 $\Longrightarrow$ 同一セクションは、ROM タイプ又は RAM タイプに統一してください。
25	Expression is out of ZERO page. SECTION=xxx ADDRESS=xxx OFFSET=xxx	ゼロページアドレッシングで処理された計算式の結果が、00_{16}から FF_{16}の範囲を越えました。 $\Longrightarrow$ 計算結果がゼロページの範囲内になるように変更して下さい。
26	Expression is out of SPECIAL page. SECTION=xxx ADDRESS=xxx OFFSET=xxx	スペシャルページアドレッシングで処理された計算式の結果が、$FF00_{16}$から $FFFF_{16}$の範囲を越えました。 $\Longrightarrow$ 計算結果がスペシャルの範囲内になるように変更して下さい。
27	label ”xxx” type is mismatch.	外部参照したラベルの種類 (ラベルまたはビットシンボル) が宣言と異なっています。 $\Longrightarrow$ 外部参照の疑似命令を正しく宣言して下さい。
28	Section ”xxx” information is out of range.	コマンド入力で指定したセクションの開始アドレスが範囲外です。 $\Longrightarrow$ 開始アドレスを正しく指定して下さい。
29	Bit value is out of range. SECTION=xxx ADDRESS=xxx OFFSET=xxx	ビット値の計算式の結果が 0 から 7 の範囲を越えました。 $\Longrightarrow$ 計算結果が範囲内になるように変更して下さい。
30	Extended area requires command option ’-BANK’	拡張領域が定義されています。 $\Longrightarrow$ コマンドオプション’-BANK’ を指定してリンクしてください。

第 3 部

740 ファミリ 用ライブラリアン

LIB74 操作マニュアル

目 次

図 目 次

表 目 次

第 1 章

LIB74 操作マニュアルの構成

LIB74 操作マニュアルは、以下の章から構成されています。

- 第 2 章 概要
  LIB74 の基本的な機能と、LIB74 が生成するファイルについて紹介します。
- 第 3 章 操作方法
  LIB74 のコマンド入力方法について説明します。
- 付録 A エラーメッセージ一覧表
  LIB74 が表示するエラーメッセージについて、その内容と対策を一覧表で示します。

第 2 章

概要

LIB74 は、SRA74 で生成したリロケータブルファイルをライブラリ形式で管理するソフトウェアです。よく利用するサブルーチンをライブラリ化することによりアセンブル時間の短縮や、プログラムの再利用を推進することができます。

2.1 機能

LIB74 は、SRA74 によって生成されたリロケータブルファイルを処理することができます。また、LIB74 が生成したファイルは LINK74[1] によって参照することができます。また、これらの各機能を十分生かすため以下の機能を備えています。

1. LINK74 で参照可能なライブラリファイルの作成及び修正を行います。
2. リロケータブルファイルを、ライブラリファイルに登録します。
3. ライブラリファイル中の不要なリロケータブルファイルを削除します。
4. ライブラリファイル中の古いリロケータブルファイルを、新しく作成したリロケータブルファイルに更新します。
5. ライブラリファイルに登録済のリロケータブルファイルを、抽出します。
6. ライブラリファイル中のリロケータブルファイルの情報を表示します。

2.2 特長

1. リンク処理の高速化。
   リロケータブルファイルをライブラリ化することにより、リンク時に必要な情報の取り出しが高速に行えるようになり、LINK74 によるリンク処理を高速化できます。
2. ライブラリファイル中のリロケータブルファイルを更新する際、ファイルの更新日比較により、最新バージョンのリロケータブルファイルのみを更新します (コマンドパラメータ “–U” 指定時) 。

[1] 740 ファミリ 用リンケージエディタのプログラム名。

2.3 生成ファイル

LIB74 では、以下の 4 種類のファイルを生成します。

1. ライブラリファイル
   - SRA74 で生成したリロケータブルファイルを編集し、ラベル及びシンボルのインデックス部を付けたファイルです。
   - ライブラリファイル中では、リロケータブルファイルを 1 つのモジュールとして管理します (以下ライブラリファイル中のリロケータブルファイルをモジュールと呼びます) 。
   - モジュール名は、ファイル属性も含むリロケータブルファイルの名前と同一です。
   - ファイル属性は、.LIB です。
2. リストファイル (以下 LST ファイルと呼びます)
   - コマンドパラメータ “–L” を指定した時に生成します。
   - ライブラリファイル内のリロケータブルファイル名、グローバルラベル及びシンボル等の一覧表を示します。
   - ファイル属性は、.LST です。
   - このファイルの構成については次節で説明します。
3. リロケータブルファイル
   - コマンドパラメータ “–X” を指定した時に生成します。
   - ライブラリファイル中のリロケータブルファイルを再生したファイルです。
   - 再生したリロケータブルファイルは、SRA74 が生成したライブラリ登録前のリロケータブルファイルと同じものです。
   - ファイル属性は、.R74 です。
4. バックアップファイル
   - コマンドパラメータの内容にかかわらず生成します。但し、LIB74 が異常終了した場合には生成しません。
   - ライブラリファイルを変更したとき、変更前のライブラリファイルをバックアップファイルとして残します。
   - ファイル属性は、.BAK です。

注意事項

リストファイル以外はバイナリ形式ですので、プリンタ及び画面への出力は、行わないでください。

2.4 LST ファイルの構成

図 2.1、図 2.2、図 2.3に、LST ファイルの出力例を示します。LST ファイルは、以下の情報を示しています。

1. モジュール名一覧表 (図 2.1)
   ライブラリファイル内に登録されている以下の情報を出力しています。

   - Module_name 部: ライブラリファイルに登録されているモジュール名を示します。出力順序は、ライブラリファイルに登録されている順番になります。
   - Offset 部: ライブラリファイルの先頭からモジュールの先頭位置までのバイト数を示します (16 進表記) 。
   - Module size 部: モジュールのメモリ容量を示します (16 進表記) 。

```
LIB74 librarian V.4.00.00                    date 1992-Dec-10 15:30  page  1

Library file name :       CALC8.LIB
Relocatable format:       VER.A
Last update time  :       1992-Dec-10 15:30
Number of module  :       3
Number of global symbol:  10

Module_name:
key_scan ..... Offset: 00000000H Module size: 00000100H
multiply ..... Offset: 00000180H Module size: 00000080H
division ..... Offset: 00000200H Module size: 000000A5H
```

図 2.1: LST ファイルの出力例 1(モジュール名一覧表)

2. グローバルラベル及びシンボル一覧表 (アルファベット順) (図 2.2)
   ここでは、以下の 2 つのテーブルを出力しています。

   - パブリックラベルリスト (PUBLIC symbol table)

     Symbol_name 部: パブリックラベル、パブリックシンボル、又はパブリックビットシンボルを示します。疑似命令 .EQU で宣言したパブリックシンボルには “(e)” を付加して出力します。パブリックビットシンボルには “(b)” を付加して出力します。

     Module_name 部: パブリックラベル、パブリックシンボル、又はパブリックビットシンボルを含むモジュール名を示します。

- 外部参照ラベルリスト (EXTERN symbol table)

  Symbol_name 部: 外部ラベル又は外部シンボルを示します。

  Module_name 部: 外部参照指定を行っているモジュール名を示します。

```
LIB74 librarian V.4.00.00                    date 1989-Jun-30 15:30  page  2

PUBLIC symbol table (symbol count = 0010)

  Symbol_name      Module_name           Symbol_name       Module_name

  _dividel........ division              _division........ division
  _key_flg1(b).... key_scan              _key_flg2(b)..... key_scan
  _key_scan....... key_scan              _key_scn1........ key_scan
  _multi1......... multiply              _multiply........ multiply
  _one(e)......... key_scan              _two(e).......... key_scan




LIB74 librarian V.4.00.00                    date 1992-Dec-10 15:30  page  3

EXTERN symbol table (symbol count = 0010)

  Symbol_name      Module_name           Symbol_name       Module_name

  _div_ansl....... division              _div_ansh........ division
  _key_buff1...... key_scan              _key_buff2....... key_scan
  _key_buff3...... key_scan              _key_buff4....... key_scan
  _mul_ansl....... multiply              _mul_ansh........ multiply
```

図 2.2: LST ファイルの出力例 2(グローバルラベル及びシンボル一覧表)

3. グローバルラベル及びシンボルのモジュール別一覧表 (図 2.3)
   ここでは、PUBLIC symbol table と EXTERN symbol table の 2 つのテーブルを出力します。どちらもモジュール名を示したあとに、そのモジュール内のグローバルラベル及びグローバルシンボルを列挙しています。

```
LIB74 librarian V.4.00.00                    date 1992-Dec-10 15:30  page   4

 PUBLIC symbol table

 Module name: key_scan (symbol count = 0006)

 _key_flg1(b)   _key_flg2(b)    _key_scan       _key_scn1
 _one(e)        _two(e)

 Module name: multiply (symbol count = 0002)

 _multi1        _multiply

 Module name: division (symbol count = 0002)

 _divide1       _division



LIB74 librarian V.4.00.00                    date 1992-Dec-10 15:30  page   5

 EXTERN symbol table

 Module name: key_scan (symbol count = 0006)

 _key_buff1     _key_buff2      _key_buff3      _key_buff4
 _linecnty      _linecntx

 Module name: multiply (symbol count = 0002)

 _mul_ansl      _mul_ansh

 Module name: division (symbol count = 0002)

 _div_ansl      _div_ansh
```

図 2.3: LST ファイルの出力例 3(グローバルラベル及びシンボルのモジュール別一覧表)

第 3 章

操作方法

3.1 起動方法

LIB74 を実行するためには、以下の情報を入力する必要があります。

1. ライブラリファイル名 (必須項目)
2. リロケータブルファイル名
3. コマンドパラメータ

LIB74 ではこれらの情報を入力する方法として、以下の 2 通りの方式を用意しています。

1. コマンド行入力
2. コマンドファイル入力

LIB74 は、いずれの入力方式においても同一の入力パラメータを使用することができ、同一のコマンドを実行することができます。次節では、各入力パラメータの機能について説明し、最後に各入力方式について具体的な例を示しながら説明します。

3.2 入力パラメータ

3.2.1 ライブラリファイル名

1. ライブラリファイル名は、必ず入力してください。
2. コマンドパラメータ “–O” の後に、スペース又はタブを置いて編集対象のライブラリファイル名を入力してください。
3. ファイル名にはディレクトリパス名が指定できます。パスの指定がない場合、まずカレントディレクトリを参照し、続いて環境変数 “LIB74” で指定されているディレクトリパスを参照します。環境変数の設定がない場合は、カレントディレクトリのファイルを処理します。
4. ファイル属性.LIB は省略可能です。

3.2.2 リロケータブルファイル名

1. リロケータブルファイル名は、スペース又はタブで区切り、複数個指定できます。
2. コマンドパラメータ “–F” の後にスペース又はタブで区切り、処理対象のリロケータブルファイル名を入力してください。
3. ファイル名にはディレクトリパス名が指定できます。パスの指定がない場合、カレントドライブのカレントディレクトリ中のファイルを処理します。
4. ファイル属性.R74 は省略可能です。

3.2.3 コマンドパラメータ

コマンドパラメータは、ライブラリファイルの操作内容の指定、出力ファイルの指定等の制御を行います。表 3.1に、コマンドパラメータ一覧表を示します。

表 3.1: コマンドパラメータ一覧表

番号 [1]	コマンドパラメータ [2]	内容
1	–O (Output)	編集対象のライブラリファイル名を指定します。
2	–F (Files)	ライブラリファイルへの追加、更新、又は抽出を行うリロケータブルファイル名、又はライブラリファイルから削除を行うモジュール名を指定します。
3	–A (Append)	ライブラリファイルへリロケータブルファイルを追加します。
	–R (Replace)	ライブラリファイル中のモジュールを、更新します。
	–D (Delete)	ライブラリファイル中から、指定したモジュールを削除します。
	–L (List)	ライブラリファイル中に登録されているモジュールに関する一覧表を出力します。
	–X (eXtract)	ライブラリファイル中から、指定したモジュールを抽出します。抽出したモジュールは、LINK74 で処理可能なリロケータブルファイルとしてカレントディレクトリに生成されます。
4	–V (Verbose)	処理中のファイル名を画面に表示します。
	–U (Update)	ライブラリファイル中のモジュールを更新する際、最新のリロケータブルファイルに対応したモジュールのみを更新します。

注意事項

1. 番号は、以下の内容を示します。

   1. 必須項目。編集対象のライブラリファイル名は必ず指定してください。

   2. コマンドパラメータ “–A” 、“–R” 、“–X” のいずれかを指定し、リロケータブルファイルの追加、更新、抽出を行う場合は、対象となるリロケータブルファイル名の指定を行ってください。モジュールを削除する場合には、対象となるモジュール名の指定を行ってください。
   同時に複数個指定するとエラーとなり、処理を中止します。

   3. 5 つの内 1 つだけを、必ず指定してください。

   4. 必要に応じて指定してください。

2. コマンドパラメータの大文字/小文字は区別しません。従って、“–A” 、“–a” のいずれも有効です。

3.2.4 コマンドパラメータの詳細説明

以下に、コマンドパラメータの詳細を説明します。

1. **−O**

   - 編集対象のライブラリファイル名を指定します。
   - ライブラリファイル名にディレクトリパス名を指定できます。
     例） `A>LIB74 -LO B:¥USR¥TEST<RET>`
   - ディレクトリパス指定を省略した場合は、まずカレントディレクトリを参照し、続いて環境変数 “LIB74” で指定されているディレクトリパスを参照します。環境変数が以下のように設定されている場合、B ドライブの ¥USR ¥LIB ディレクトリ中のファイルを処理します。
     例） `A>SET LIB74=B:¥USR¥LIB`
   - ディレクトリパス指定を省略し、環境変数の設定もしていない場合は、カレントドライブのカレントディレクトリ中のファイルを処理します。
   - ファイル属性を省略した場合、既定値として属性.LIB を選択します。
   - ファイルをフルネームで記述することにより、“.LIB” 以外の属性のファイルも指定可能です。

2. **−F**

   - ライブラリファイルに対して追加、更新、抽出を行うリロケータブルファイル名、又は削除を行うモジュール名を指定します。
   - スペース又はタブで区切って複数個のファイル名を指定できます。
   - ファイル名にディレクトリパス名が指定できます。
     例） `A>LIB74 -AO TEST.LIB -F B:¥WORK ¥SUB1 C:SUB2<RET>`
   - ディレクトリパス指定を省略した場合は、カレントドライブのカレントディレクトリ中のファイルを処理します。
   - ファイル属性を省略した場合、既定値として属性.R74 を選択します。
   - ファイルをフルネームで記述することにより、“.R74” 以外の属性のファイルも指定可能です。

3. **−A**

   - ライブラリファイルの新規作成、又はライブラリファイルにリロケータブルファイルの追加を行います。
   - ライブラリファイルを新規に作成する場合は、“–F ” で指定したリロケータブルファイルを、指定した順番にライブラリの先頭から登録します。
   - リロケータブルファイルを追加する場合には、“–F ” で指定したリロケータブルファイルを、指定した順番でライブラリファイルの最後に追加します。

- 指定したリロケータブルファイル名と、既に登録されているモジュール名との重複チェックは行いません。
- 同一モジュール名が同時に指定された場合は、ラベル又はシンボルの二重定義エラーが検出され、処理は中断されます。

4. **–R**

   - ライブラリファイル中のモジュールを更新します。
   - "–U" と共に使用すると、ライブラリファイル内のモジュールより、リロケータブルファイルの更新日付けが新しいモジュールだけを更新します。

5. **–D**

   - ライブラリファイルから、指定したモジュールを削除します。

6. **–L**

   - ライブラリファイル中に登録されているモジュールに関する一覧表 (LST ファイル) を出力します。
   - "–F" でファイル名を指定した場合、指定したリロケータブルファイルの情報を LST ファイルに出力します。
   - "–F" でファイル名を指定しなかった場合は、ライブラリファイル中のすべてのモジュールについての情報を LST ファイルに出力します。
   - ファイル属性は、.LST です。

7. **–X**

   - ライブラリファイル中から指定したモジュールを抽出し、ライブラリファイルに登録される前のリロケータブルファイル状態にします。
   - リロケータブルファイルの更新日時は、抽出した時の日時になります。
   - 抽出したリロケータブルファイルは、LINK74 で処理することができます。
   - 抽出したリロケータブルファイルは、カレントディレクトリに生成します。
   - "–F" でリロケータブルファイル名を指定しなかった場合、ライブラリファイル中のすべてのモジュールを抽出します。
   - ライブラリファイルの内容は変化しません。
   - ファイル属性は.R74 です。

8. **–V**

   - LIB74 が現在どのファイルに対して処理を行っているかを画面に表示します。

9. –U

   - ライブラリファイル中のモジュールを更新する際 (“–R” 指定時) 、最新のファイルのみを更新します。
   - ライブラリファイル中のモジュールを更新する際 “–F” で指定したリロケータブルファイルの更新日時と、ライブラリファイル内に登録されているモジュールの更新日時を比較し、“–F” で指定したリロケータブルファイルの方が新しい時、更新処理を行います。
   - リロケータブルファイルの更新日時は、オペレーティングシステムのファイル管理情報に依存します。

3.3 入力方法

3.3.1 コマンド行入力

コマンド行入力の特徴を以下に示します。

1. オペレーティングシステムのコマンド入力状態ですべてのコマンド入力を行う方式です。
2. 指定するファイルの数やコマンドパラメータの数が少ない場合に有用です。
3. バッチファイルや、make ファイルで実行コマンドを記述する場合に使用できます。

ライブラリファイル TEST.LIB からモジュール FILE1.R74 を削除する例を図 3.1に示します。 ライブラリファイル TEST.LIB にリロケータブルファイル FILE1.R74 、FILE2.R74 を

```
A>LIB74 -D -O TEST.LIB -F FILE1<RET>
```

図 3.1: コマンド行入力例 1(モジュールの削除)

追加する例を図 3.2に示します。

```
A>LIB74 -AO TEST.LIB -F FILE2 FILE3<RET>
```

図 3.2: コマンド行入力例 2(リロケータブルファイルの追加)

3.3.2 コマンドファイル入力

コマンドファイル入力の特徴を以下に示します。

1. LIB74 の処理内容を予めエディタによりコマンドファイルとして作成し、起動時にこのファイル名を指定する方式です。

2. 指定するファイル名やコマンドの文字が多くコマンド行入力が使用できない場合に便利な方式です。

3. コマンドファイル名の指定は、LIB74 を起動する時に '@' 記号を用いて図 3.3のように指定します。図 3.3の場合、ファイル CMD.DAT の内容を実行します。

4. コマンドファイルに記述する内容は、コマンド行入力と同じです (ただし、LIB74 の起動を指示する LIB74 の部分は不用) 。改行コードはスペースとみなしますので、長いコマンドは複数行に分けて記述できます。図 3.2をコマンドファイルで記述すると図 3.4のように書くことができます。

コマンドが正しく入力されると LIB74 は処理を開始します。LIB74 の各コマンド処理が終了すると、画面に終了メッセージを表示してプログラムを終了します。図 3.5に LIB74 が正常に処理を終了した場合の画面表示例を示します。

```
A>LIB74 @CMD.DAT<RET>
```

図 3.3: コマンドファイルの指定

```
-AO
TEST.LIB
-F
FILE2
FILE3
```

図 3.4: コマンドファイル記述例

```
A>LIB74 -AVO TEST.LIB -F SUB_1 SUB_100<RET>
740 Family LIBRARY MANAGER V.4.00.00
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.
< test.lib > Create
APPEND FILE = sub_1,sub_100
MODULE COUNT          000002
GLOBAL SYMBOL COUNT   000019
A>
```

図 3.5: LIB74 正常終了時の画面表示例

3.4 エラー

3.4.1 エラーの種類

LIB74 実行時に発生するエラーは、以下の原因によるものがあります。

1. オペレーティングシステムに関するエラー
   ディスクやメモリ容量の不足等、LIB74 を実行するオペレーティングシステム環境に関わるエラーです。付録 A エラーメッセージ一覧表を参照の上、オペレーティングシステムのコマンドにより対応してください。

2. LIB74 のコマンド入力に関するエラー
   LIB74 起動時のコマンド入力に関わるエラーです。コマンド入力に誤りがあった場合、図 3.6のようなヘルプ画面を画面に表示します。本章の内容を確認の上、コマンドを再入力してください。

3. 処理対象のリロケータブルファイルの内容に関するエラー
   パブリックラベルが二重定義されている等、リロケータブルファイルの内容に関わるエラーです。LST ファイルを参照して該当箇所を修正してください。

```
A>LIB74<RET>
740 Family LIBRARY MANAGER V.4.00.00
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

  Usage: A>LIB74 -[ardxluv]o <filename> [-f <filename> ...]

          -o :   library file name
          -f :   relocatable file names
          -a :   append command
          -r :   replace command
          -d :   delete command
          -x :   extract command
          -l :   listout command
          -u :   update check (option)
          -v :   verbose (option)
```

図 3.6: コマンド行エラー時のヘルプ画面

LIB74 動作中にエラーが発生すると、画面にエラーメッセージを表示します。付録 A エラーメッセージ一覧表を参照の上、処理してください。

3.4.2 オペレーティングシステムへの戻り値

バッチファイル等に実行コマンドを記述する場合、実行結果に応じて処理の内容を変えたい場合があります。LIB74 では、実行結果を表 3.2の 4 つのエラーレベルに分けてオペレーティングシステムに返すようにしています。

表 3.2: エラーレベル

エラーレベル	実行結果の内容
0	正常終了
1	処理対象のライブラリファイル又はリロケータブルファイルに関するエラー
2	LIB74 のコマンド入力に関するエラー
3	オペレーティングシステムに関するエラー

3.5 環境変数

LIB74 は、常にカレントディレクトリ内のライブラリファイルを優先して処理します。

カレントディレクトリ内にライブラリファイルが存在しない場合で、かつ環境変数 “LIB74” にディレクトリパスが設定されている場合は、環境変数に設定されたディレクトリ内のライブラリファイルを処理します。

例） `A>SET LIB74=B:¥USR¥LIB<RET>`

付録 A

エラーメッセージ一覧表

A.1 システムエラー一覧表

LIB74 動作中にシステム上のエラーを検出すると、エラーメッセージを画面に表示して処理を中止します。表 A.1にシステムエラー一覧表を示します。

A.2 ライブラリアンエラー一覧表

LIB74 動作中に処理上のエラー検出すると、エラーメッセージを画面に表示して処理を中止します。表 A.2にライブラリアンエラー一覧表を示します。

表 A.1: システムエラー一覧表

エラーメッセージ	エラー内容と対策
Usage:A>LIB74 −[adluvxz]o <filename> [−f <filename>...]	コマンド入力が誤っています。 ⟹ ヘルプ画面を参照して、コマンドを再入力してください。
Can't open xxx	該当ファイルが見つかりません。 ⟹ コマンドパラメータ "–O" や "–F" で指定したファイルが、指定したディレクトリに存在するか確認してください。
Can't create xxx	該当ファイルが生成できません。 ⟹ コマンドパラメータ "–O" の指定を確認して再入力してください。
Out of disk space[1]	ファイルを出力するためのディスク領域が不足しています。 ⟹ ディスク上に空き領域を作ってください。
Input file read error xxx	該当ファイル読み込み中にエラーが発生しました。 ⟹ このエラーは、オペレーティングシステムに関わるエラーです。一般にディスク装置のハードウェア的な異常による場合が多いと考えられます。
Internal error	LIB74 の内部エラーが発生しました。 ⟹ LIB74 購入先に御連絡ください。
File seek error xxx	該当ファイルがシークできません。 ⟹ このエラーは、オペレーティングシステムにかかわるエラーです。一般にディスク装置のハードウェア的な異常による場合が多いと考えられます。

注意事項

1. LIB74 を実行するときは、その実行過程で中間ファイルを作成するため、ライブラリファイルの約 2 〜 3 倍のディスク空き容量が必要です。

表 A.2: ライブラリアンエラー一覧表

エラーメッセージ	エラー内容と対策
xxx is a multiple defined in xxx. others in xxx	パブリックラベル又はパブリックシンボルを二重定義しています。 $\Longrightarrow$ LST ファイルでライブラリファイル内のパブリックラベル又はシンボルを確認してください。
xxx module is not in the library	ライブラリファイル中に該当モジュールが存在しません。 $\Longrightarrow$ LST ファイルでライブラリファイル中のモジュール名を確認してください。
Invalid module or library	ライブラリファイル中のモジュールと指定されたリロケータブルファイルのフォーマットが異なります。 $\Longrightarrow$ 編集対象のライブラリファイルとリロケータブルファイルを同じバージョンの SRA74 で作成してください。
xxx command file not found	該当コマンドファイルが存在しません。 $\Longrightarrow$ 指定したコマンドファイルを確認してください。
Out of heap space	ライブラリアンが動作するために必要なメモリが不足しています。 $\Longrightarrow$ グローバルラベル数を減らしてください。
Too many object module	ライブラリファイル内のモジュール数が多すぎます。 $\Longrightarrow$ ライブラリファイルを分割してください。 一つのライブラリファイル中のモジュール数は、最大 500 個です。
CPU number error	指定したライブラリファイル又はリロケータブルファイルが SRA74 で生成されたものではありません。 $\Longrightarrow$ 指定したライブラリファイル又はリロケータブルファイルを確認してください。

第 4 部

740 ファミリ 用クロスリファレンサ

CRF74 操作マニュアル

目 次

図 目 次

表 目 次

第 1 章

CRF74 操作マニュアルの構成

CRF74 操作マニュアルは、以下の章から構成されています。

- 第 2章 概要
  CRF74 の基本的な機能と、CRF74 が生成するファイルについて紹介します。
- 第 3章 操作方法
  CRF74 のコマンド入力方法について説明します。
- 付録 A エラー一覧表
  CRF74 が表示するエラーメッセージについて、その内容と対策を一覧表で示します。

第 2 章

概要

CRF74 は、ソースファイル内のラベル及びシンボルの相互参照リスト (クロスリファレンスリストと呼びます) を生成します。このリストを使用すれば、プログラム修正時にソースファイル各部間の依存関係を容易に把握することができます。

2.1 機能

CRF74 は、SRA74[1] と共に利用できます。CRF74 では、以下の機能によりソースファイルの把握を効率的に行うことができます。

1. ラベルの参照命令の種類を参照行番号に表示します。
2. 疑似命令.INCLUDE によるファイル読み込みの実行が可能です。
3. 疑似命令.PAGE によるヘッダの出力が可能です。

2.2 生成ファイル

CRF74 では、クロスリファレンスリスト (以下 CRF ファイルと呼びます) を生成します。

- ラベル及びシンボル名の相互参照リストを示します。
- 1 行当たりのカラム数は 80 、1 ページの行数は 57 行に固定です。
- このファイルは、プリンタに出力してデバッグやエディットの際にお使いください。
- ファイル属性は、.CRF です。

[1] 740 ファミリ 用リロケータブルアセンブラ名。

2.3 CRF ファイルの構成

図 2.1に、CRF ファイルの出力例を示します。CRF ファイルは、以下の情報を示しています。

1. ラベル及びシンボル名と、その参照行及び定義行。
   定義行に '#'(シャープ) 、参照行のうちサブルーチンコールによるものに '&'(アンパサンド) を付加して示します。
2. ラベル及びシンボル名は、32 文字まで表示します。なお、CRF ファイルはもっとも長い名前に合わせてフォーマッティングされています。
3. リストのヘッダには、疑似命令.PAGE で指定しているタイトルを表示します (ただし、表示は 30 文字まで有効となり、それを越える文字は表示されません) 。
4. CRF74 は、ソースプログラム中のラベル及びシンボルの値の判断は行いません。従って、条件付きアセンブルの判断はできませんのでご注意ください。

```
740 Family CROSS REFERENCE V.1.00.10                                P. 001

A0          3926   4285   8549   9079   9100
AA          3884   5545   5668
ABEND       9396&  9465&  9587#
ABEND10     9588   9593#
ABENDRT     9590#  9605
ACCHK       1201#  1408
ACCHK5      1213   1237#
ACCHKE      1235   1239   1241   1249#
ADDING      4994   5006#
ADDING0     5013   5014   5016#
ADDING1     5015   5019#
ADDRESS      300    318   1025
ADR_CHK     9154#
ADR_OUT     8302   8336   9145#
ADR_PNT     8157#
ADR_PNT2    8149#
```

図 2.1: CRF ファイル例

第 3 章

操作方法

3.1 起動方法

CRF74 を実行するためには、以下の情報 (入力パラメータ) を入力する必要があります。

1. ソースファイル名 (必須項目)
2. コマンドパラメータ

3.2 入力パラメータ

3.2.1 ソースファイル名

1. ソースファイル名は必ず入力してください。
2. ファイル属性 (.A74) を省略した場合、既定値として属性.A74 を選びます。
3. ファイル名をフルネームで指定することにより、.A74 以外の属性 (例.ASM) のファイルも処理可能です。
4. ファイル名は、ドライブ名が指定可能です。ファイル名のみを記述した場合、カレントドライブ中のファイルを処理します。
5. ソースファイル名は 16 個まで指定できます。

3.2.2 コマンドパラメータ

コマンドパラメータは、ソースファイル中の疑似命令.INCLUDE の検出の有無や、出力ファイルのドライブ名を指定します。表 3.1に、コマンドパラメータの内容を示します。

表 3.1: コマンドパラメーター一覧表

コマンドパラメータ	内容
–O	CRF ファイルを出力するドライブ名、及びディレクトリパスを指定します。 指定の書式は、次のようになります。 例) A>CRF74 SRCFILE -OC:\TMP <RET> (‘ ¥ ’ 記号は、‘ \ ’で表しています。これは、オペレーティングシステムによって表記が異なるためで、コードは同一ですのでいずれも使用できます) CRF ファイルを C ドライブの TMP というディレクトリに出力します。
–I	疑似命令.INCLUDE を無視します。

3.3 入力方法

3.3.1 コマンド行入力

CRF74 は、オペレーティングシステムのプロンプト状態でコマンド行を入力することにより起動します。図 3.1に、起動コマンドの入力例を示します。コマンド行入力中に誤りを検出

```
A>CRF74 SRCFILE1 SRCFILE2 SRCFILE3<RET>
```

図 3.1: コマンド行入力例

すると、図 3.2のようにヘルプ画面を表示し処理を中止します。

```
A>CRF74<RET>
740 Family CROSS REFERENCE V.1.00.10
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Usage: crf74 <filename> [-ifilename,..] [-opath]
       -i : not include specified files ( use -ifilename,.... )
       -o : select drive and directory for output ( use -otmp )
```

図 3.2: コマンド行エラー時のヘルプ画面

3.4 エラー

3.4.1 エラーの種類

CRF74 実行時に発生するエラーは、以下の原因によるものがあります。

1. オペレーティングシステムに関するエラー
   ディスクやメモリ容量の不足等、CRF74 を実行するオペレーティングシステム環境に関わるエラーです。付録 Aエラーメッセージ一覧表を参照の上、オペレーティングシステムのコマンドにより対応してください。

2. CRF74 のコマンド行入力に関するエラー
   CRF74 起動時のコマンド行入力に関わるエラーです。本章の内容を確認の上コマンドを再入力して下さい。

3. 処理対象のソースファイル内容に関するエラー
   疑似命令.INCLUDE によって指定されたソースファイルが存在しない場合に発生します。

CRF74 は、エラーを検出すると図 3.3の形式でエラー内容を画面に表示します。付録 Aのエラー一覧表を参照の上処理して下さい。

```
A>CRF74 SRCFILE1<RET>
740 Family CROSS REFERENCE V.1.00.10
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

now processing pass 1
now making cross reference ( SRCFILE1.A74 )
-----+
Out of heap space

A>
```

図 3.3: エラー表示例

3.4.2 オペレーティングシステムへの戻り値

オペレーティングシステムのバッチファイル等に実行コマンドを記述する場合、実行結果に応じて処理の内容を変えたい場合があります。CRF74 では、実行結果を表 3.2の 3 つのエラーレベルに分けてオペレーティングシステムに返すようにしています。

表 3.2: エラーレベル

エラーレベル	実行結果の内容
0	正常終了
1	疑似命令.INCLUDE で指定されたソースファイルが存在しない場合に発生するエラー
2	CRF74 のコマンド入力に関するエラー
3	オペレーティングシステムに関するエラー

3.5 環境変数

CRF74 は、オペレーティングシステムの環境変数は使用していません。

付録 A

エラーメッセージ一覧表

A.1 システムエラー一覧表

CRF74 処理中にシステムエラーを検出すると、エラーメッセージを画面に表示し処理を中止します。表 A.1にシステムエラー一覧表を示します。

表 A.1: システムエラー一覧表

エラーメッセージ	エラー内容と対策
Usage: crf74 <filename> [-ifilename] [-opath]	コマンド入力が誤っています。 $\Longrightarrow$ ヘルプ画面を参照して、コマンドを再入力してください。
Can't open xxx	該当ファイルが見つかりません。 $\Longrightarrow$ ソースファイル名を確認して再入力してください。
Can't create xxx	該当ファイルが生成できません。 $\Longrightarrow$ ディスク上に空き領域を作ってください。
Out of disk space	ファイルを出力するためのディスク容量が不足しています。 $\Longrightarrow$ ディスク上に空き領域を作ってください。
Out of heap space	クロスリファレンサが動作するために必要なメモリが不足しています。[1] $\Longrightarrow$ シンボル又はラベルの数を減らしてください。

注意事項

1. CRF74 で扱えるシンボル及びラベルの総数は、CRF74 を実行するシステムでの使用可能メモリ容量に依存しています。

A.2 クロスリファレンスエラー一覧表

クロスリファレス作成に関するエラーを検出すると、エラーメッセージを画面に出力し、処理はそのまま続行します。表 A.2にクロスリファレンスエラーとその内容を示します。

表 A.2: クロスリファレンスエラー一覧表

エラーメッセージ	エラー内容と対策
Can't open include file xxx	疑似命令.INCLUDE で指定されたソースファイルが存在しません。 $\Longrightarrow$ ディレクトリ内容を確認してください。

第 5 部

M37280 用ファイル変換ツール

CV74 操作マニュアル

目次

第１章 CV74 操作マニュアルの構成

CV74 操作マニュアルは、以下の章から構成されています。

- 第２章 概要
  CV74 の基本的な機能と、CV74 が生成するファイルについて説明します。

- 第３章 操作方法
  CV74 のコマンド入力方法について説明します。

- 付録 A エラーメッセージ一覧表
  CV74 が表示するエラーメッセージについて、その内容と対策を一覧表で示します。

第２章 概要

CV74 は 、M37280 対応アセンブラシステム（SRA74 V.3.00 以上）にて生成された HEX ファイルとシンボルファイルをデバッガで利用可能なデータ形式に変換します。

デバッガを使って M37280 の 64KB を越えるアドレス空間のデータが存在するプログラムをデバッグする際にご利用ください。

CV74 をご使用になる前に、あらかじめ LINK74 で HEX ファイルとシンボルファイルを生成してください。

2.1 機能

CV74 は、M37280 対応アセンブラシステム（SRA74 V.3.00 以上）にて生成された HEX ファイルとシンボルファイルをデバッガで利用可能なデータ形式に変換することができます。

1. HEX ファイルを、64KB を越えるアドレス空間のデータと 64KB 以内のアドレス空間のデータに分割します。
2. シンボルファイルをデバッガにてデバッグ可能な形式に変換します。

注）CV74 で変換する以前の HEX ファイル、シンボルファイルを用いた場合は正常にデバッグできません。

2.2 生成ファイル

CV74 では、以下の３種類のファイルを生成します。

1. 64KB 以内のアドレス空間のデータを格納した HEX ファイル

   - デバッガでデバッグ可能な HEX ファイルです。
   - LINK74 が生成した HEX ファイルから 64KB 以内のアドレス空間のデータを抽出し格納したファイルです。
   - 拡張子は、.HEX です。

2. 64KB を越えるアドレス空間のデータを格納した HEX ファイル

   - LINK74 が生成した HEX ファイルから 64KB を越えるアドレス空間のデータを抽出し格納したファイルです。
   - 拡張子は、.HXH です。

3. デバッガでデバッグ可能なシンボルファイル

   - 拡張子は、.SYM です。

第３章 操作方法

3.1 起動方法

CV74 を起動するためには、以下の情報（入力パラメータ）を入力する必要があります。

1. 変換対象ファイル名
2. コマンドパラメータ

3.2 入力パラメータ

3.2.1 変換対象ファイル名

1. 変換対象ファイル名は、必ず入力してください。
2. 変換対象の HEX ファイル、シンボルファイルは、同じディレクトリ上に準備してください。
3. ファイル名は複数指定することはできません。
4. 相対ディレクトリ指定を含めピリオドを含むファイル名は指定できません。

3.2.2 コマンドパラメータ

コマンドパラメータは、出力ファイル名や、変換対象ファイルを指定します。表 3.1 にコマンドパラメータの内容を指定します。

表 3.1： コマンドパラメータ一覧表

コマンドパラメータ	内容
－O	出力ファイル名を指定します。 このオプションは必ず指定してください。
－H	HEX ファイルだけを変換します。 （－Ｓオプションと同時に指定はできません。）
－S	シンボルファイルだけを変換します。 （－Hオプションと同時に指定はできません。）

なお、－H、－Ｓいずれも指定されなかった場合には、HEX、シンボルファイルの変換を行います。

[注意事項]

□ デバッガでデバッグする際には、CV74 が生成した HEX、シンボルファイルをダウンロードしてください。

3.3 入力方法

CV74 は、オペレーティングシステムのプロンプト状態でコマンドを入力することにより起動します。

図 3.1 に、LINK74 で生成した HEX ファイル（master.hex）とシンボルファイル（master.sym）をデバッガでデバッグ可能な HEX ファイル（change.hex）とシンボルファイル（change.sym）に変換する例を示します。

```
A> CV74 master –Ochange <RET>
HEX, SYM file converter for 7200 Series V.1.00.00
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.


== HEX file convert now == ( master.hex --> change.hex )
== SYM file convert now == ( master.sym --> change.sym )
```

図 3.1　CV74 起動例

3.4 エラー

3.4.1 エラーの種類

CV74 実行時に発生するエラーは、以下の原因によるものがあります。

1. オペレーティングシステムに関するエラー
   ディスクやメモリ容量の不足等、CV74 を実行するオペレーティングシステム環境に関わるエラーです。付録Ａエラーメッセージ一覧を参照の上、オペレーティングシステムのコマンドにより対応してください。

2. CV74 のコマンド行入力に関するエラー
   CV74 起動時のコマンド行入力に関わるエラーです。本章の内容を確認の上コマンドを再入力してください。

3. 処理対象のソースファイル内容に関するエラー
   処理対象のＨＥＸファイル、シンボルファイルの内容に関わるエラーです。ファイル内容を確認の上、再入力してください。

```
A> CV74 <RET>
   HEX, SYM file converter for 7200 Series V.1.00.00
   Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
   AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
   All Rights Reserved.

    Error : No input files specified

     Usage: cv74 Input-filename -Ooutput-filename [-S or -H]<cr>
```

図 3.2 :　コマンド行エラー時のヘルプ画面

CV74 動作中にエラーが発生すると、画面にエラーメッセージを表示します。付録Ａエラーメッセージ一覧表を参照の上、処理してください。

3.4.2 **オペレーティングシステムへの戻り値**

OS のバッチファイル等に実行コマンドを記述する場合、実行結果に応じて処理の内容を変えたい場合があります。CV74 では、実行結果を 表 3.2 の５つのエラーレベルに分けてオペレーティングシステムに返すようにしています。エラーレベルの利用方法については、市販の OS 等の説明書を参照してください。

表 3.2 : エラーレベル一覧表

エラーレベル	実行結果の内容
0	正常終了
1	^C（コントロールＣ）入力による強制終了
2	オペレーティングシステムに関するエラー
3	処理対象のソースファイル内容に関するエラー
4	CV74 のコマンド入力に関するエラー

付録 A エラーメッセージ一覧

A.1 エラーメッセージ

エラーメッセージ	内容と対策
'-H' and '-S' are used	－Ｈオプションと－Ｓオプションを同時に指定しています。 ⇒ どちらか一方のオプションを指定してください。
. is specified in the filename	ファイル名にピリオドが指定されています。 ⇒ ファイル名を確認してください。
Can't create 'xx' file	'xx'ファイルが生成できません。 ⇒ ディレクトリ容量を確認してください。
Can't create temporary file	テンポラリファイルが生成できません。 ⇒ カレントディレクトリ以外にテンポラリファイルを作成するように、環境変数'TMP'にディレクトリを指定してください。
Can't open 'xx' file	'xx'ファイルがオープンできません。 ⇒ ファイル名を確認してください。
Command line is too long	コマンド行の文字数が多すぎます。 ⇒ コマンドを入力し直してください。
File format error	入力ファイルのフォーマットが不正です。 ⇒ ファイルフォーマットを確認してください。
Hex files number exceed 1	変換対象ファイルが複数指定されています。 ⇒ 変換対象ファイルは１つだけ指定してください。
Invalid option 'xx' is used	無効なコマンドオプション 'xx' を使用しています。 ⇒ 指定したオプションは存在しません。コマンドを入力し直してください。
No input files specified	入力ファイルの指定がありません。 ⇒ 入力ファイルを指定してください。
No output files specified	出力ファイル名が指定されていません。 ⇒ 出力ファイル名を指定してください。
Not enough memory	メモリが足りません。 ⇒ ファイルを分割して実行し直してください。又はメモリを増設してください。
Option 'xx' is multiple defined	コマンドオプション'xx'が二重に定義されています。 ⇒ コマンドを入力し直してください。
Option 'xx' is not appropriate	コマンドオプション 'xx' の記述が正しくありません。 ⇒ コマンドオプションを指定し直してください。

A.2 **ワーニングメッセージ**

ワーニングメッセージ	内容と対策
Output data to the ‘xx’ file doesn't exist	指定領域のデータが空のため、ファイル’xx’は生成されません。 ⇒ 入力ファイルの内容を確認してください。

MELPS740 ファミリ用分岐命令最適化ツール
操作説明書

1．機能

このプログラムはアセンブラプログラムの分岐命令を最適な分岐命令に変換するものです。

2．生成ファイル

LOOP74 は、コマンドオプションの指定により変換結果下記のように出力します。

-ASM コマンドオプションを指定した場合：

SRA74 が生成した xxx.i ファイルに出力します（xxx.i ファイルは入力ファイルと同じディレクトリに出力されます)。

-ASM コマンドオプションを指定しない場合：

LOOP74 に指定された構造化ソースファイルに出力します。

注）-ASM コマンドオプションを付加して変換したファイルは、デバッガにてソースレベルデバッグはできなくなります。

3．起動

3.1．入力パラメータ

LOOP74 を実行するためには、以下の情報（入力パラメータ）を入力する必要があります。

・ソースファイル名

変換対象のソースファイル名を指定します（指定できる個数は１個です)。

ファイル名は拡張子も含めて入力して下さい（使用できる拡張子は、.A74 のみです)。

ファイル名にはディレクトリパス指定が可能です。

ファイル名のみを指定した場合は、カレントドライブのカレントディレクトリ中のファイルを処理します。

・LOOP74 コマンドオプション

オプション	内容
－.	実行状況を画面に出力しません（エラーメッセージはのぞく）
－ＡＳＭ	構造化命令をアセンブラニーモニックに展開した分岐命令を対象として分岐命令の変換処理を行います。（このオプションを使用した場合、ソースレベルデバッグはできなくなります。）
－Ｖ	LOOP74 のバージョンを表示して終了します
－ＣＯＵＮＴｎ	変換回数を、ｎ（10 進数値)で指定された回数までで終了します。ｎは １ ～ 100 の範囲で指定してください。デフォルト値は、100 回の変換回数となっています。
－ＣＯＭＭＥＮＴ	変換対象行に LOOP74 が変換したことを示すコメントを付加します。

上記のオプションは、大文字、小文字の判別を行いますので必ず上記の通り、アルファベット大文字で指定しなければなりません。

・SRA74 コマンドオプション

LOOP74 が SRA74 を起動する際に指定する SRA74 用のコマンドオプションを指定します。

LOOP74 は自身がサポートしているコマンドオプション以外はすべて SRA74 のコマンドオプションとして判断し SRA74 に渡します。

3.2. 起動方法

LOOP74 は、以下の形式で起動します。

```
A> LOOP74 SAMPLE.A74 -COMMENT -S -C -L<RET>
```

上記の場合変換を行うアセンブラソースファイルとして「SAMPLE.A74」を指定し、変換した行に LOOP74 がコメントを付加することを指示、また SRA74 にはコマンドオプションとして「-S -C -L」を渡すことを指定しています。

また、上記の指定は下記のように行っても同様に処理されます。

```
A> LOOP74 -L SAMPLE.A74 -S -COMMENT -C <RET>
```

すなわち、LOOP74 ではコマンドパラメータの指定順序は関係ありません。
※LOOP74 は自動的にタグファイルを生成しますので､SRA74 の``-E"オプションを記述する必要はありません（記述しても問題はありません)。

4. 入力ソースファイルフォーマット

LOOP74 の処理するソースファイルフォーマットは SRA74 の入力ソースファイルフォーマットに準じます。
SRA74 で処理できないソースファイルは LOOP74 でも処理はできません。
また、LOOP74 では以下の制限事項があります。

・ .EXT などで宣言されたラベルには対応していません。
・ ラベル定義に ':' が記述されていないソースには対応していません。
・ .if 疑似命令の条件「偽」部分など、SRA74 が処理しない部分には対応していません。
・ 分岐命令のオペランドに下記のような相対アドレスを記述した場合はエラーとなります。

また、LOOP74 では相対アドレスによる記述であるかどうかは、分岐命令オペランドに、＊の記述がある場合に相対アドレスの指定であると判断しています。
(例) :

```
BRA ＊ + 5
BRA ＊ + label
```

5. 出力ファイルフォーマット

LOOP74 は変換の対象となる分岐命令行を下記のフォーマットで出力します。
LOOP74 は通常変換前の命令行は削除しますがコマンドオプション -COMMENT を指定した場合、元の命令行はコメントとして残します。

・コマンドパラメータ -COMMENT が指定されていない場合（デフォルト）

・変換対象行①

```
BEQ          L1
```

・LOOP74 の変換結果

```
▲BNE ▲..lop1▲; This is the line which loop74 generated
▲JMP ▲L1▲; This is the line which loop74 generated
..lop1：
```

・変換対象行②

```
BEQ     L1     ;goto L1
```

・LOOP74 の変換結果

```
▲BNE ▲..lop1 ▲; This is the line which loop74 generated
▲JMP ▲L1 ▲;goto L1
..lop1：
```

上記のように、LOOP74 が生成した分岐命令行には LOOP74 が生成したことを示すコメントが付加されます。

また、変換対象行にコメント記述がされていた場合 LOOP74 は、出力した最後の分岐命令に対象行に記述されていたコメントを出力します。

・コマンドパラメータ -COMMENT が指定されている場合

・変換対象行①

```
BEQ      L1
```

・LOOP74 の変換結果

```
;  BEQ   L1
▲BNE ▲..lop1▲; This is the line which loop74 generated
▲JMP ▲L1▲; This is the line which loop74 generated
..lop1：
```

・変換対象行②

```
BEQ     L1     ;goto L1
```

・LOOP74 の変換結果

```
    ;   BEQ     L1      ;goto L1
    ▲BNE ▲..lop1 ▲; This is the line which loop74 generated
    ▲JMP ▲L1 ▲; This is the line which loop74 generated
..lop1：
```

上記のように、コマンドオプション -COMMENT を指定した場合、変換対象行はコメントとして出力します。

また、変換対象行にコメント記述がされていた場合 LOOP74 は、出力した最後の分岐命令に対象行に記述されていたコメントを出力しません。

6. 変換規則

LOOP74 は以下の規則で分岐命令を変換します。

入力行	出力行
BEQ L1	BNE ..lop JMP L1 ..lop1:
BNE L1	BEQ ..lop1 JMP L1 ..lop1:
BCC L1	BCS ..lop1 JMP L1 ..lop1:
BCS L1	BCC ..lop1 JMP L1 ..lop1:
BMI L1	BPL ..lop1 JMP L1

```
                              ..lop1:
-------------------------------------------------------------
   BPL      L1                          BMI      ..lop1
                                        JMP      L1
                              ..lop1:
-------------------------------------------------------------
   BVC      L1                          BVS      ..lop1
                                        JMP      L1
                              ..lop1:
-------------------------------------------------------------
   BVS      L1                          BVC      ..lop1
                                        JMP      L1
                              ..lop1:
-------------------------------------------------------------
   BBC      BIT,MEM,L1                  BBS      BIT,MEM,..lop1
                                        JMP      L1
                              ..lop1:
-------------------------------------------------------------
   BBS      BIT,MEM,L1                  BBC      BIT,MEM,..lop1
                                        JMP      L1
                              ..lop1:
-------------------------------------------------------------
   BRA      L1                          JMP      L1
=============================================================
```

LOOP74 が生成するラベルは、..lop1～..lop65535 の形式で出力します。

　また、同一飛び先に分岐している条件分岐命令が連続して複数行記述されていた場合は下記のように変換します。

```
=============================================================
    入力行                                出力行
=============================================================
   Bcnd_1      L1                        Bcnd_1      ..lop1
   Bcnd_2      L1                        Bcnd_2      ..lop1
          :                                   :
          :                                   :
   Bcnd_n      L1                        BcndR_n     ..lop2
```

```
                                ..lop1:
                                        JMP         L1
                                ..lop2:
============================================================
```

上記の、Bcnd_1、Bcnd_2、Bcnd_n は下記の条件分岐命令を示します。

BEQ、BNE、BCC、BCS、BMI、BPL、BVC、BVS

BcndR_n は、Bcnd_n 分岐命令の反転した条件分岐命令を示します。

上記のように、LOOP74 は同一飛び先に分岐する条件分岐命令が連続していた場合、最後に記述された条件分岐命令のみ条件を反転して展開します。

下記に具体例を示します。

```
============================================================
        入力行                          出力行
============================================================
        BEQ     L1                      BEQ     ..lop1
        BCS     L1                      BCC     ..lop2
                                ..lop1:
                                        JMP         L1
                                ..lop2:
============================================================
```

7. エラー

LOOP74 を実行中にエラーを検出すると、エラーメッセージを画面に表示し、LOOP74 を中止します。エラー一覧表を以下に示します。

エラーメッセージ	内容と対策
Can't create file 'filename'	'filename'ファイルが生成できません。 ディスク容量を確認してください。
Can't create Temporary file	テンポラリファイルが生成できません。 ディレクトリが書き込み禁止になっていないか確認してください。

Can't open file 'filename'	'filename'ファイルがオープンできません。 指定したファイル名を確認してください。
Can't translate	最大の分岐命令でもアドレスが届きません。 ソースプログラム内容を確認してください。
Command line is too long	コマンド行の文字数が多すぎます。 LOOP74 はコマンド行は最大 255 バイトまでしか受付けません。入力の文字数を減らしてください。
Error occurred in executing 'sra74'	SRA74 の実行でエラーが発生しました。 SRA74 を実行し直してください。
Illegal source file	LOOP74 が解析できないソースファイルフォーマットです。 正しいフォーマットのソースファイルを指定してください。
Illegal TAG file	LOOP74 が解析できないタグファイルフォーマットです。 正しいフォーマットのタグファイルを指定してください。
No input files specified	入力ファイルの指定がありません。 入力ファイルを指定してください。
Not enough memory	メモリが足りません。 入力ファイルを分割してください。 またはメモリを増設してください。
Option 'xx' is not appropriate	コマンドオプション xx の記述が正しくありません。 コマンドオプションを指定し直してください。
Source files number exceed 1	ソースファイルが複数指定されています。 ソースファイルは１つだけ指定して下さい。

Too many branch error	変換する分岐命令が多すぎます。 ソースファイルを分割して、１ファイル中の LOOP74 が処理する分岐命令個数を減らしてください。

8．ワーニング

LOOP74 を実行中にワーニングを検出すると、ワーニングメッセージを画面に表示します。LOOP74 の処理は続行します。ワーニング一覧表を以下に示します。

ワーニングメッセージ	内容と対策
Relative address is specified	分岐先に相対アドレスが指定されています。 この行の変換処理は行われません。

9．ＯＳへの返り値

LOOP74 は処理を終えた際にＯＳに下記のように返り値を渡します。

返り値	内容
0	正常終了
1	LOOP74 の読み込んだソースファイル、タグファイル内容に関するエラー
2	コマンド行の入力に対するエラーが発生しています。
3	オペレーティングシステムに関するエラー
4	^C（コントロールＣ）入力による強制終了

M3T-SRA74 V.4.10 ユーザーズマニュアル

Rev. 1.00
03.08.01
RJJ10J0192-0100Z

M3T-SRA74 V.4.10
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

RJJ10J0192-0100Z