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

7900シリーズ用 Cコンパイラ

NC79WA　ガイドブック

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2000.07

《安全設計に関するお願い》

三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は品質、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三菱半導体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三菱電機株式会社・三菱電機セミコンダクタシステム株式会社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、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社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、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は、予告なしに、本資料に記載した製品または仕様を変更することがあります。三菱半導体製品のご購入に当たりましては、事前に三菱電機株式会社・三菱電機セミコンダクタシステム株式会社または特約店へ最新の情報をご確認頂きますとともに、三菱電機半導体情報ホームページ（http://www.semicon.melco.co.jp/）および三菱開発ツールホームページ（http://www.tool-spt.mesc.co.jp/）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、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社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。三菱電機株式会社・三菱電機セミコンダクタシステム株式会社は、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、三菱電機株式会社・三菱電機セミコンダクタシステム株式会社また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、文書による三菱電機株式会社・三菱電機セミコンダクタシステム株式会社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三菱電機株式会社・三菱電機セミコンダクタシステム株式会社また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

電子メールの場合：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、開発ツールサポート窓口support@tool.mesc.co.jpまで送信ください。

Windows 98/95/Windows NT 4.0版：\SUPPORT\製品名\SUPPORT.TXT
EWS版：/support/製品名/toolinfo.txt

FAXの場合：各ユーザーズマニュアル及びガイドブックの最後に添付されている「技術サポート連絡書」に必要事項を記入の上、開発ツールサポート窓口まで送信ください。FAX送信先は「技術サポート連絡書」に記載してあります。

NC79WA ガイドブック目次

ガイドブックの構成

ガイドブックの構成

NC79WA へようこそ

製品の機能概要を紹介しています。
「製品に関する注意事項」は必ずお読みください。

NC79WA をインストールしよう

製品をインストールする際に必要となる情報が記載されています。
製品をインストールされる前にお読みください。

NC79WA を使ってみよう

NC79WA の基本的な使用方法を説明しています。
製品をご使用になる前にお読みください。

NC79WA のスタートアッププログラム

マイクロコンピュータ組み込みプログラムを C 言語で開発する場合の様々な設定を行うファイルについて説明しています。
NC79WA を初めてご使用になる際は必ずお読みください。

NC79WA ヒント集

NC79WA のプログラミングテクニックなどを紹介しています。
「Q&A 集」には、よくあるご質問について掲載しておりますので、ご一読ください。

付録

コマンドオプションや拡張機能などの一覧を掲載しています。

注意！

ガイドブックには、本製品をご使用いただくための基本的な情報を掲載しています。より詳細な情報は製品 CD-ROM に含まれている電子マニュアルを参照してください。

memo

NC79WA へようこそ

製品内容

NC79WA をご購入いただきありがとうございます。

NC79WA には以下のものが含まれております。これらのものが含まれていない場合には、ご購入いただいた三菱電機営業および特約店にご連絡ください。

梱包内容

- ☐ NC79WA ガイドブック（本書）
- ☐ NC79WA リリースノート
- ☐ 使用権許諾契約書
- ☐ ライセンス ID 証書（新規購入のみ）
- ☐ CD-ROM
- ☐ TM リリースノート

注意！

ライセンス ID 証書は、今後のバージョンアップ時に必要となりますので大切に保管してください。

CD-ROM の内容

CD-ROM には、以下に示す以外のコンパイラ製品も含まれておりますが、NC79WA をご購入の場合に、ご使用できる製品は以下のとおりです。

- ☐ NC79WA
- ☐ NC79 電子マニュアル（PDF 形式ファイル）
- ☐ AS79 電子マニュアル（PDF 形式ファイル）
- ☐ TM
- ☐ TM 電子マニュアル（PDF 形式ファイル）

- ☐ 7900 シリーズ関連開発支援ツールデータシート（HTML 形式ファイル）
- ☐ Adobe Acrobat Reader
- ☐ MM（マスクファイル生成ツール）
- ☐ GNU ユーティリティ

注意！

NC79、AS79 および TM の各マニュアルは、CD-ROM 内の電子マニュアルのみです。

製品の最新情報

製品をご使用になる際には、製品の最新情報を以下のホームページなどでご確認ください。

```
http://www.tool-spt.mesc.co.jp
```

こちらでは、他のツール製品に関する最新情報についても掲載されていますのでご活用ください。

NC79WA の特徴

製品の構成

NC79WA は次のようなコンポーネントから構成されています。

- ANSI 規格に準拠した C コンパイラ
- リロケータブルアセンブラ

製品の概要

NC79WA は、デバッガで読み込み可能な機械語ファイル（アブソリュートモジュールファイル）や、ROM に書き込み可能な機械語ファイル（モトローラ S フォーマット、インテル HEX フォーマット）を生成します。

関連開発支援ツール

言語系開発支援ツール（ここでは NC79WA）に関連した開発支援ツールに次のようなものがあります。詳細は「7900 関連開発支援ツールデータシート」を参照してください。

□ エミュレータデバッガ
エミュレータを PC または EWS 上で制御するソフトウェアです。

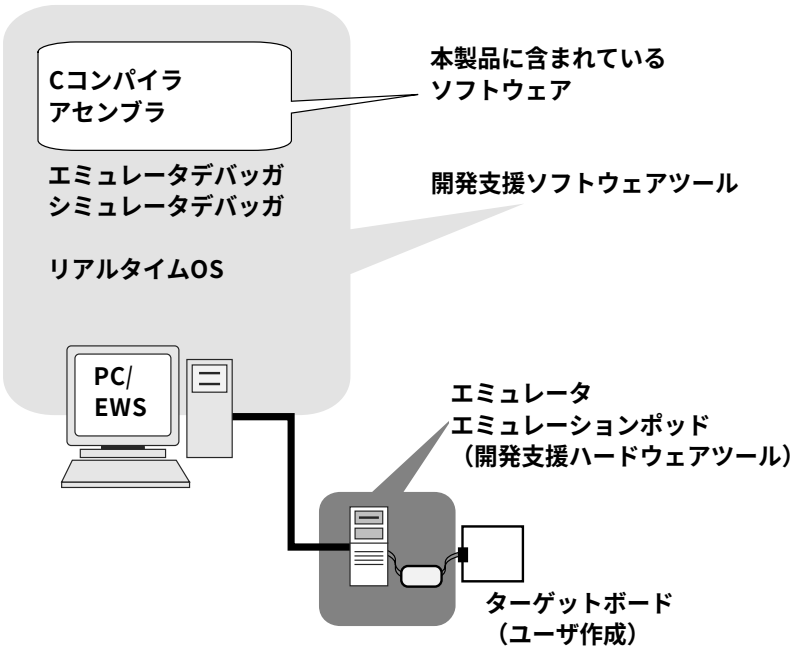
□ シミュレータデバッガ
エミュレータを使用せずアプリケーションプログラムのデバッグを可能にするソフトウェアです。

□ エミュレータ／エミュレーションポッド
エミュレータおよびエミュレーションポッドは、マイコンの機能を実現したハードウエアをいいます。

□ リアルタイム OS
アプリケーションプログラムのリアルタイム処理機能を向上できる組み込み型のリアルタイム OS です。

NC79WA を使ったプログラム開発環境例

NC79WA を使ってアプリケーションプログラムを開発するときの開発環境例を次に示します。



ANSI 規格に準拠した C コンパイラ：NC79

NC79 の特徴とコンパイル処理の流れについて説明します。

NC79 の機能概要

- ANSI 規格に準拠した C 言語で記述されたプログラムをアセンブリ言語に変換します。
- マイコンへの組み込み型プログラムを開発するための拡張機能を持っています。
- リアルタイム OS(MR79)のための拡張機能を持っています。
- C ソースレベルでデバッグするためのデバッグ情報を出力します。

コンパイル処理の流れ

1 C 言語のプリプロセス処理、コンパイル処理を順次行い、C 言語で記述されたプログラムをアセンブリ言語ファイルに変換します。

C 言語ファイル（拡張子 .c）
⇓
cpp79
⇓
ccom79
⇓
アセンブリ言語ファイル（拡張子 .a79）

2 アセンブリ言語ファイルをアセンブル処理し、リロケータブルモジュールファイルに変換します。

アセンブリ言語ファイル（拡張子 .a79）
⇓
as79
⇓
リロケータブルモジュールファイル（拡張子 .r79）

3　リロケータブルモジュールファイルをリンク処理し、アブソリュートモジュールファイルに変換します。

リロケータブルモジュールファイル（拡張子 .r79）
⇓
ln79
⇓
アブソリュートモジュールファイル（拡張子 .x79）

リロケータブルアセンブラ：AS79

AS79 の特徴とアセンブル処理の流れについて示します。

AS79 の機能概要

- アセンブリ言語で記述されたプログラムを機械語に変換します。
- マクロ機能をサポートしています。
- 構造化記述文をサポートしています。

アセンブルの処理の流れ

1　アセンブリ言語ファイルをリロケータブルモジュールファイルに変換する

アセンブリ言語ファイル（拡張子 .a79）
⇓
as79
⇓
リロケータブルモジュールファイル（拡張子 .r79）

注意！
構造化記述命令を使用しているときは、アセンブラオプション -P を指定してください。

2　リロケータブルモジュールファイルをアブソリュートモジュールファイルに変換する

リロケータブルモジュールファイル（拡張子 .r79）
⇓
ln79
⇓
アブソリュートモジュールファイル（拡張子 .x79）

その他の機能

NC79WA には、プログラム開発をサポートするユーティリティプログラムが添付されています。

Stk Viewer

C 言語でプログラム開発を行う場合に、関数が使用するスタックサイズと関数の呼び出し関係を元にスタック使用量を計算します。

utl79

使用頻度の高い変数を DPnDATA に割り当てます。

lmc79

アブソリュートモジュールファイル(.x79)からモトローラ S フォーマット(.mot)もしくは、インテル HEX フォーマット(.hex)を生成します。モトローラ S フォーマットとインテル HEX フォーマットは、PROM ライタで PROM にプログラムを書き込むためのフォーマットです。

xrf79

アセンブリ言語ファイル(.a79)からラベルやシンボルの定義および参照情報をリストにしたクロスリファレンスファイル(.xrf)を生成します。

abs79

as79 が出力するリストファイル(.lst)からアブソリュートリストファイル(.als)を生成します。リストファイルに出力されているリロケータブルなアドレス情報を絶対アドレスに変換することができます。

Map Viewer

アブソリュートモジュールファイル(.x79)のマップ情報を GUI 表示します。このプログラムは PC 版のみ対応しています。

製品に関する注意事項

製品をご使用になる際には、以下の内容に従ってくださるようお願いいたします。

PC 版に関する注意事項

動作環境についての注意事項

- PC 版は、Windows 95、Windows 98、Windows NT 4.0 以降の環境で動作します。Windows 3.1 および Windows NT 3.5x 以前のバージョンでは動作しません。
- 日本語 Windows NT 環境で DOS 窓のサイズが 80 × 25 以外に設定されている場合、NC79、AS79 等を起動すると窓のサイズが頻繁に切り替わります。窓のサイズを 80 × 25 に設定してください。

ファイル名についての注意事項

ソースプログラムファイルの名前や作業を行うディレクトリ名は、つぎの注意事項に従ってください。

- 漢字を含むディレクトリ名、ファイル名は使用できません。
- ファイル名に使用するピリオド（.）は一つのみ使用可能です。
- ネットワークパス名は使用できません。ドライブ名に割り当ててご使用ください。
- 「ショートカット」は使用できません。
- 空白文字を含むディレクトリ名、ファイル名は使用できません。
- "..." 表記を用いて 2 つ以上のディレクトリを指定することはできません。
- パス指定を含めたファイル名の長さが 128 文字以上になるものは使用できません。

ウィルスチェックプログラムに関する注意事項

ウィルスチェックプログラムが常駐した状態で NC79WA を起動すると正常に起動しない場合があります。その場合は、ウィルスチェックプログラムの常駐を解除してから NC79WA を起動しなおしてください。

PC 版 NC79WA をバージョンアップするときの注意事項

NC79WA をバージョンアップする場合は、あらかじめ、インストールされている NC79WA をアンインストールしてから、新しいバージョンをインストールしてください。

■ NC79WA のアンインストール手順
NC79WA をアンインストールするには、「コントロールパネル」-「アプリケーションの追加と削除」を選択しアンインストールを実行してください。

電子マニュアルに関する注意事項

現在、EWS 環境上では日本語の電子マニュアルを表示できません。

日本語環境に対応している Acrobat Reader が動作するパソコンでマニュアルを参照してください。Acrobat Reader が動作る環境については、Acrobat Reader のマニュアルやアドビシステムズ社のホームページなどでご確認ください（http://www.adobe.co.jp）。

NC79WA をインストールしよう

インストールを始める前に

インストールを始める前に次の内容をご確認ください。

□ 本製品の「使用権許諾契約書」、「リリースノート」などをよくお読みください。製品をインストールした場合は、契約書の記載内容に同意されたものとみなします。

□ NC79WA を快適に使用するには、32M バイト以上のメモリと 20M バイト以上の空きハードディスク領域が必要です。

□ 製品のインストールは専用のインストーラを使用してください。

□ インストールの途中でライセンス ID を入力する必要があります。インストールを始める前にライセンス ID を確認してください。

NC79WA のインストーラ

インストーラは、次に示す環境（対応ホスト、対応 OS、言語）毎に用意されています。ご購入になった製品を確認の上、該当するインストーラを使用してください。

■ 日本語環境

対応ホスト	対応 OS	インストーラ名	CD-ROM 上のディレクトリ
PC	Windows 95 Windows 98 Windows NT	SETUP.EXE	¥NC79WA¥W95J
SPARC Station	SunOS 4.x Solaris2.x	setup setup	/nc79wa/sparc /nc79wa/solaris
HP9000/700	HP/UX 10.x	setup	/nc79wa/hp700

■ 英語環境

対応ホスト	対応 OS	インストーラ名	CD-ROM 上のディレクトリ
	Windows 95 Windows 98 Windows NT	SETUP.EXE	\NC79WA\W95E

PC 版のインストール

PC 版の製品は次の手順でインストールしてください。

(1) CD-ROM 上の対象製品のインストーラが配置されているディレクトリに移動します。

(2) インストーラを起動して表示されるメッセージにしたがってインストールを完了してください。

注意！

インストールの途中で入力するデータは、ユーザー登録のためのファイルを作成するのに使用されます（ファイルを作成するのは PC 版のインストーラのみです）。

EWS 版のインストール

EWS 版のインストールは次の手順にしたがってインストールしてください。

以下の説明では、SOLARIS 版をもとに記述しています。他のホストにインストールする場合はディレクトリ名を読みかえてください。

(1) インストーラのあるディレクトリに移動
CD-ROM 上の、対象製品のインストーラが配置されているディレクトリに移動します。
ここでは、CD-ROM をマウント注）したディレクトリ名を /cdrom と記述しています。システムに設定されているディレクトリ名に読みかえてください。

注）HP9000/700 版の CD-ROM マウント方法

HP-UX 上では、CD-ROM のファイル名が大文字で見えたり、「;1」 と言った表示が付加されることがあります。この場合は製品を取り出すことができませんので、下記の例にしたがってマウントをやり直してください。

```
% pfs_mountd &
% pfsd 4 &
% pfs_mount -t iso9660 -x unix /dev/dsk/c0t2d0 /cdrom
```

注意！

・スーパーユーザが実行してください。
・CD-ROM のドライブ名（/dev/dsk/c0t2d0）はホストマシンにより異なりますので別途ご確認ください。
・マウントポイント（ /cdrom）は例です。
・例で使用している各コマンドについての詳細は、HP-UX のマニュアルを参照ください。
・% はプロンプトを表します。

(2) インストーラを実行
setup コマンドを実行してください。

(3) ライセンス ID を入力
- (ハイフン) も含めて正確に入力してください。
製品に同梱されているライセンス ID 証書に書かれている ID を入力してください。
ライセンス ID に誤りがあると、下記のメッセージを表示されます。この場合は、もう一度ライセンス ID を入力してください。

```
Illegal license ID. '1234-5678-9012-3456-7890'
```

(4) 圧縮ファイルを指定
data.0 があらかじめ指定されています。そのまま リターンキーを押してください。

(5) インストール先ディレクトリを指定
指定したディレクトリが存在する場合は、このディレクトリが書き込み可能であり、かつ十分な空き容量があることが必要です。

(6) インストール先のディレクトリを確認
指定したディレクトリが存在しない場合、ディレクトリを作成するかを尋ねられます。y(はい) または n(いいえ) を入力してを押してください。

注意！

ここで y (はい)を選んだ場合でもディレクトリが作成できないことがあります。この場合は、インストールを中断して、OS の mkdir コマンドでディレクトリを作成してから、インストールをやり直してください。

(7) 作業用ディレクトリを指定
インストーラの作業のために一時的に使用するディレクトリです。
ここで指定するディレクトリは、書き込み可能であり、かつ十分な空き容量を持っていることが必要です。
/tmp があらかじめ指定されています。/tmp に十分な空き容量がある場合は、何も入力せずにそのままリターンキーを押してください。

(8) ファイル転送の開始
転送中のファイルの名前が逐次表示されます。表示が出るまで時間がかかる場合があります。
入力したライセンス ID が対象製品と一致しない場合は、ファイル転送は行われず次のメッセージを表示して処理を中止します。この場合は、インストーラの実行からやり直してください。

```
License ID error 'data.0'
```

(9) ファイル転送の終了
メッセージに表示されるファイル（名前、ディレクトリはメッセージ参照）に、製品情報を記録したファイルが作られます。
ディレクトリ名は、インストール先ディレクトリにより変わります。

次に手順(1)～(9)にしたがってインストールを実行したときの画面表示例を示します。

```
(1) >cd /cdrom/nc79wa/solaris
(2) >./setup
    Tool Installation Utility V.1.00.00
    Copyright(c) 1998 MITSUBISHI ELECTRIC CORPORATION
    AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
    All Rights Reserved.

(3) Please type license ID.[xxxx-xxxx-xxxx-xxxx-xxxx]

(4) Program package file name[data.0]

(5) Install Dirctory[/usr/local/mtool] /usr/common/tool

(6) The directory '/usr/common/tool' does not exist.
    Do you want the directory to be created (y/n) y

(7) Installer Temporary Directory[/tmp]

(8) Setup made a file, named 'toolinfo.txt', at the directory
    "/usr/common/tool/support/nc79wa".
    And Setup recorded product information in this file.
    Please use this file if you need calling us.

(9) Installation completed.
```

インストール後に生成されるディレクトリとソフトウェア

インストールが完了するとインストーラのデフォルトのインストール先ディレクトリの下に、表にしめすディレクトリが生成され、各ディレクトリに表のファイルがコピーされます。

■ PC 版のインストール先ディレクトリ（デフォルト）

```
¥mtool
```

■ EWS 版のインストール先ディレクトリ（デフォルト）

```
/usr/local/mtool
```

注意！

EWS 対応版では実行形式のファイルに拡張子 .exe はつきません。

■ インストール先ディレクトリとインストールファイル一覧 -1-

ディレクトリ	ファイル
bin	nc79.exe（コンパイルドライバ） stkviewer.jar（スタックサイズ算出ユーティリティ） mapveiwer.exe（マップ情報ビューワー） utl79.exe（DPnDATA 宣言ユーティリティ） as79.exe（アセンブラドライバ） ln79.exe（リンケージエディタ） lmc79.exe（ロードモジュールコンバータ） lb79.exe（ライブラリアン） xrf79.exe（クロスリファレンサ） abs79.exe（アブソリュートリスタ） sc79.exe（ソースファイル変換ツール） make.exe NC79WA ヘルプファイル
lib79	cpp79.exe（C 言語プリプロセッサ） ccom79.exe（C コンパイラ本体） mac79.exe（アセンブルマクロプロセッサ） pre79.exe（構造化プリプロセッサ） asp79.exe（アセンブラプロセッサ） nc79lib.lib（ランタイムおよび標準ライブラリ） nc79m8.lib（ランタイムおよび標準ライブラリ -fDPO8 オプション用）

■ インストール先ディレクトリとインストールファイル一覧 -2-

ディレクトリ	ファイル
inc79	標準ヘッダファイル
src79\startup	ncrt0.a79（スタートアッププログラム） sect79.inc（メモリ配置設定インクルードファイル）
smp79	makefile（NC79 用サンプル make ファイル） ncrt0.a79（スタートアッププログラム） samp79.c（C 言語サンプルプログラム） sect79.inc（メモリ配置設定インクルードファイル） sample.a79（アセンブリ言語サンプルプログラム） struct.a79（構造化記述サンプルプログラム）
support	mtoolspt.htm（ツールサポートホームページアドレス）
support\nc79wa	regist.txt（ユーザー登録用紙） Support.txt（技術サポート連絡書テキストファイル）
manual	nc79uj.pdf（NC79 電子マニュアル） as79uj.pdf（AS79 電子マニュアル） sc79uj.pdf（sc79 電子マニュアル）

電子マニュアルのインストールについて

EWS 版のインストーラでは電子マニュアルはインストールされません。
EWS 版をご購入の場合、日本語環境に対応している Acrobat Reader が動作するパソコンでマニュアルを参照してください。Acrobat Reader が動作する環境については、Acrobat Reader のマニュアルやアドビシステムズ社のホームページなどでご確認ください。
`http://www.adobe.co.jp`

インストール後の環境設定を行う

インストールが完了した後、次の環境変数を設定してください。

PC の環境設定

表の中の「*自動*」は、PC 版のインストーラが AUTOEXEC.BAT を書きかえます。したがって、デフォルトでインストールを実行した場合は、AUTOEXEC.BAT を書きかえる必要はありません。

環境変数	設定例
BIN79	*自動*（SET BIN79=C:\MTOOL\BIN）
INC79	*自動*（SET INC79=C:\MTOOL\INC79）
LIB79	*自動*（SET LIB79=C:\MTOOL\LIB79）
TMP79	*自動*（SET TMP79=C:\MTOOL\TMP)
NCKIN	SET NCKIN=SJIS
NCKOUT	SET NCKOUT=SJIS
コマンドパス	*自動*（C:\MTOOL\BIN を追加）

EWS の環境設定

製品のインストール完了後、プログラムを実行する前に以下の環境変数を設定してください。

注意！

環境変数名は大文字で設定してください。

環境変数	設定例
BIN79	`>setenv BIN79 /usr/local/mtool/bin`
INC79	`>setenv INC79 /usr/local/mtool/inc79`
LIB79	`>setenv LIB79 /usr/local/mtool/lib79`
TMP79	`>setenv TMP79 /usr/local/mtool/tmp`
NCKIN	`>setenv NCKIN EUC`
NCKOUT	`>setenv NCKOUT EUC`
コマンドパス	`/usr/local/mtool/bin` を追加

NCKIN と NCKOUT 環境変数について

NCKIN と NCKOUT に、NC79 を実行した場合の入出力コードの設定をします。

■ NCKIN

入力コード系を指定する環境変数です。次の二つの環境から設定できます。

EUC

日本語文字の記述が可能です。ただし、1 文字が 3 バイトで構成される外字コードは使用できません。

SJIS

日本語文字の記述が可能です。

■ NCKOUT

出力コード系を指定する環境変数です。次の二つの環境から設定できます。内容は NCKIN と同様です。

- EUC
- SJIS

NCKIN,NCKOUT のデフォルト（環境変数を設定しなかった場合）の設定

ホスト	環境変数	デフォルト値
PC 版	NCKIN	SJIS
	NCKOUT	SJIS
EWS 版	NCKIN	EUC
	NCKOUT	EUC

注意！

日本語及び英語以外（たとえばヨーロッパ語圏）でコンパイラを使用される際は、必ず EUC と設定してください。

ユーザー登録をしましょう

バージョンアップ情報や技術サポートなどのサービスを受けるためにユーザー登録を行ってください。ユーザー登録をされていない場合は、これらのサービスを受けることができません。

また、ご購入後**30 日以内**に登録してくださるようお願い申し上げます。

PC 版のユーザー登録方法

1 PC 版をインストールすると以下のファイルが生成されます。

```
\mtool\support\nc79wa\regist.txt
```

\mtool はデフォルトでインストールした場合のディレクトリです。

2 regist.txt のファイル内容をすべてカット＆ペーストして以下の電子メールアドレス宛に送付してください。

```
regist@tool.mesc.co.jp
```

電子メールをご使用になれない場合は、regist.txt のファイル内容をプリントアウトしファクシミリで送付してください。送付先は、製品に添付されているリリースノートで確認してください。

EWS 版のユーザー登録方法

1 ライセンス ID 証書に添付されているフォームに必要事項を記入しファクシミリで送付してください。
送付先は、製品に添付されているリリースノートで確認してください。

Acrobat Reader のインストール

電子マニュアルを参照するためには Acrobat Reader が必要になります。CD-ROM に Acrobat Reader を添付していますので必要に応じてインストールしてください。

- インストーラは製品 CD-ROM 上で起動してください。
- 添付されている Readme ファイルの内容にしたがってインストールしてください。

また、Acrobat Reader は、アドビシステムズ社のホームページからダウンロードすることができます。Acrobat Reader の最新情報などはつぎの URL を参照してください。

```
http://www.adobe.co.jp
http://www.adobe.com
```

NC79WA を使ってみよう

NC79WA をご使用になる前に

NC79WA をご使用になる前に次の内容をご確認ください。

- □ 「使用権許諾契約書」、「リリースノート」をよくお読みください。
- □ NC79WA を快適に使用するには、32M バイト以上のメモリと 20M バイト以上の空きハードディスク領域が必要です。
- □ 以下の URL に開設していますホームページなどで、製品に関する最新情報をご確認ください。

```
http://www/tool-spt.mesc.co.jp
```

- □ 製品に添付されているスタートアップファイル（ncrt0.a79, sect79.inc）をご使用になる場合は、ご使用のマイコン機種、お客様のシステムにあうように内容を変更する必要があります。
  マイコン機種による変更点についてはマイクロコンピュータデータブックなどをご参照ください。
- □ ここでは、製品に添付されているインストーラの標準で、パソコン上に製品をインストールしたものとして説明を行っています。インストールの際にディレクトリ名などを変更している場合は、ディレクトリ名を読み替えてください。

NC79 を使ってみよう

製品に添付されているサンプルファイルを使用してコンパイル手順を説明します。
インストーラのデフォルトで製品をインストールした場合、次の C 言語のサンプルファイルが \mtool\smp79 ディレクトリにインストールされます。

- □ ncrt0.a79
  スタートアッププログラムのサンプルファイルです。
- □ sect79.inc
  スタートアッププログラムのサンプルファイルです。
- □ samp79.c
  C 言語プログラムのサンプルファイルです。

(1) サンプルファイルをワークディレクトリにコピーします

```
>mkdir C:¥local¥work
>copy C:¥mtool¥smp79¥*.* C:¥local¥work
>cd C:¥local¥work
```

(2) ncrt0.a79 / sect79.inc を修正します

ここでは、サンプルファイル ncrt0.a79 / sect79.inc をそのまま使用します。

注意！

プログラム開発の際は、開発するシステムの構成にあうようにこれらのプログラムを書きかえてください。サンプルファイルの詳細については「NC79WA のスタートアッププログラム」を参照してください。

(3) C 言語でプログラムを作成する

ここでは、サンプルファイル samp79.c をそのまま使用します。

注意！

C 言語でプログラムを記述する際は次の内容に注意してください。
NC79 は最適化オプションの有無に関わらず、メモリへの転送や演算を実行する場合、一時的にそのメモリの内容を書きかえた後で、転送や演算を行なう場合があります。すなわち、割り込みなどで非同期にそのメモリを参照すると期待した結果が参照できない場合があります。
したがって、割り込み処理プログラムやリアルタイム OS のタスク間で共有する変数（メモリ）に対して、その変数を volatile 修飾子で宣言する、割り込みの一時禁止を行なうなどの排他制御を必ず行なってください。

(4)　コンパイルを実行します

1　スタートアッププログラム（ncrt0.a79）をアセンブルします

はじめに、スタートアッププログラムをアセンブルします。nc79 コマンドを実行してリロケータブルモジュールファイル ncrt0.r79 を生成します。

```
>nc79 -c ncrt0.a79
```

2　C 言語サンプルプログラム（samp79.c）をコンパイルします

nc79 コマンドを実行してリロケータブルモジュールファイル samp79.r79 を生成します。

```
>nc79 -c samp79.c
```

3　ncrt0.r79 と samp79.r79 をリンクします

ln79 コマンドを実行してアブソリュートモジュールファイル samp79.x79 を生成します。

```
>ln79 -o samp79 ncrt0.r79 samp79.r79 -l nc79lib.lib
```

注意！

エラーが発生した場合は、エラーメッセージにしたがってファイルを修正してください。

ライブラリファイルをリンクする場合の注意事項

NC79WA のライブラリファイルは nc79lib.lib と nc79m8.lib の 2 種類があります。NC79WA のコンパイルオプション -fDPO8 の指定の有無により使用するライブラリファイルが異なります。

-fDPO8 の指定	使用するライブラリファイル
なし	nc79lib.lib
あり	nc79m8.lib

コンパイルドライバ nc79 でリンクを行う場合は、nc79 がコマンドオプション -fDPO8 の指定の有無に従ってライブラリファイルを選択します。ただし、ln79 コマンドでリンクを行う場合は、コンパイルオプションの指定とあうように、使用するライブラリファイル名をコマンドライン（コマンドファイル）で指定してください。

AS79 を使ってみよう

製品に添付されているサンプルファイルを使用してアセンブル手順を説明します。

インストーラのデフォルトで製品をインストールした場合、次のアセンブリ言語のサンプルファイルが \mtool\smp79 ディレクトリにインストールされます。

- □ sample.a79
  アセンブリ言語を使ったサンプルプログラムです。
- □ struct.a79
  構造化記述言語を使ったサンプルプログラムです。

sample.a79 をアセンブルしよう

1 sample.a79 をアセンブルする

as79 コマンドを実行してリロケータブルモジュールファイルとリストファイルを生成します。

```
>as79 sample -lm
```

- -lm
  アセンブラリストファイルを生成するコマンドオプションです。'-lm' はマクロ命令の展開行をリストファイルに出力することを指示しています。

2 sample.a79 をリンクする

ln79 コマンドを実行してアブソリュートモジュールファイルを生成します。

```
>ln79 sample -order ram1=40,prog1=0e0000
```

- -order
  メモリの配置を指定します。ram1 セクションを 40H 番地から、prog1 セクションを 0e0000H 番地から配置します。

注意！

メモリの配置を行うには、コマンドオプションで指定する方法と、アセンブリ言語ソースファイルに .ORG 指示命令を用いて配置する方法があります。サイズの大きいプログラムを作成する場合は、.ORG 指示命令を用いて配置する方法をおすすめします。

struct.a79 をアセンブルしよう

1　struct.a79 をアセンブルする

as79 コマンドを実行してリロケータブルモジュールファイルとリストファイルを生成します。

```
>as79 struct -ls -p
```

- -ls
  リストファイルを生成するコマンドオプションです。'-ls' は構造化記述命令文の変換行をリストファイルに出力することを指示しています。
- -p
  構造化記述文を処理することを指示するコマンドオプションです。構造化記述文を使ったソースファイルをアセンブルするときには必ずこのオプションを指定してください。指定しない場合は構造化記述を処理できずエラーが発生します。

2　struct.a79 をリンクする

ln79 コマンドを実行してアブソリュートモジュールファイルを生成します。

```
>ln79 struct
```

電子マニュアルを使ってみよう

本製品の電子マニュアルは PDF（Portable Document Format）ファイルで提供しています。マニュアルを参照される場合は、製品に添付の Acrobat Reader などの PDF ファイル表示プログラムを使用してください。

電子マニュアルは、インストーラによって mtool（デフォルトでインストールした場合）の下の次のディレクトリにインストールされます。

■　電子マニュアルのインストールディレクトリ

ディレクトリ	PDF ファイル	内容
manual	nc79uj.pdf	NC79 電子マニュアル日本語版
	as79uj.pdf	AS79 電子マニュアル日本語版
	sc79j.pdf	sc79 電子マニュアル日本語版
manual	nc79ue.pdf	NC79 電子マニュアル英語版
	as79ue.pdf	AS79 電子マニュアル英語版
	sc79e.pdf	sc79 電子マニュアル英語版

注意！

EWS 対応版のインストーラは電子マニュアルをインストールしません。製品 CD-ROM から各電子マニュアルの PDF ファイルを Acrobat Reader がインストールされている PC 上にコピーして参照してください。電子マニュアルは CD-ROM の \NC79WA\MANUAL に格納されています。

各マニュアルは、目次または「しおり」にリンクがはられています。ここから目的のページを開くことができます。

電子マニュアルを表示するには次の方法があります。

■　スタートメニューからオープンする
スタートメニューに登録されるので、[スタート]→[プログラム (P)]→[MITSUBISHI-TOOL]の目的のツール名からマニュアルを起動することもできます。

■　PDF ファイルをダブルクリックする
参照したい電子マニュアルの PDF ファイルをダブルクリックすると、Acrobat Reader がマニュアルのデータを読み込んで起動します。

■　Acrobat Reader を起動して開く
または、Acrobat Reader を起動してから「ファイル - 開く ...」メニューで電子マニュアルを指定してください。

オンラインヘルプを使ってみよう

本製品には、NC79WA のオンラインヘルプが用意されています。オンラインヘルプは、Windows のヘルプ環境に対応しています。

オンラインヘルプは、インストーラによって mtool（デフォルトでインストールした場合）の下の次のディレクトリにインストールされます。

ディレクトリ	日本語ヘルプファイル	英語ヘルプファイル
bin	nc79waj.cnt	nc79wae.cnt
	nc79waj.hlp	nc79wae.hlp

オンラインヘルプを利用するには次の方法があります。

■ スタートメニュー から起動する
スタートメニューに登録されるので、[スタート]→[プログラム (P)]→[MITSUBISHI-TOOL]から起動することもできます。

■ オンラインヘルプを直接起動する
オンラインヘルプを起動するには、ヘルプファイル（拡張子 .hlp）をダブルクリックしてください。

memo

NC79WA のスタートアッププログラム

NC79WA には、スタートアッププログラムのサンプルとして以下のファイルが含まれています。

- ☐ ncrt0.a79
- ☐ sect79.inc

スタートアップサンプルプログラムの概要

ncrt0.a79

このプログラムはC言語とマイコンの初期化などを行います。アセンブリ言語で記述されています。

sect79.inc

プログラムやデータなどをメモリのどの領域に配置するかを設定します。配置指定はセクション単位で行います。このファイルは、ncrt0.a79 からインクルードされます。

注意！

これらのサンプルプログラムを使用される場合は、ご使用のマイコン機種、お客様のシステムにあうように内容を変更する必要があります。
マイコン機種による変更点についてはマイクロコンピュータデータブックなどをご参照ください。

スタートアップサンプルプログラム ncrt0.a79 の内容

ncrt0.a79 は C 言語とマイコンの初期化などを行うスタートアッププログラムのサンプルです。アセンブリ言語で記述されています。

開発するシステムの構成にあうように ncrt0.a79 を書きかえます。

sect79.inc のインクルード部

メモリ配置を設定するファイル sect79.inc をインクルードします。

注意！

この部分は書きかえないでください。

```
;*******************************************************
        .include        sect79.inc
;=======================================
```

RAM 初期化マクロ定義部

RAM 領域を初期化するマクロ命令の定義です。

注意！

この部分は書きかえないでください。

BZERO

RAM をゼロクリアします。

BCOPY

ROM にある初期値を RAM にコピーします。

```
;=======================================
;       initialize MACRO definition
;=======================================
BZERO   .macro  _TOP,_SECT
        ldad    E,#SIZEOF _SECT
        phb                         ; push Eregister(high)
        pha                         ; push Eregister(low)
        ldad    E,#STARTOF _TOP
        phb
        pha
        .glb    _bzero
        jsrl    _bzero
        .endm

BCOPY   .macro  _FROM,_TO,_SECT
        ldad    E,#SIZEOF _SECT
        phb
        pha
        ldad    E,#STARTOF _TO
        phb
        pha
        ldad    E,#STARTOF _FROM
        phb
        pha
        .glb    _bcopy
        jsrl    _bcopy
        .endm
```

プログラムのスタートアドレス定義部

リセット直後のプログラム開始ラベルの定義です。

プログラム開始ラベル

'start:' の次の行に書かれたラベルからプログラムが始まります。

注意！

この部分は書きかえないでください。
sect79.inc の「割り込みベクタアドレスの設定」のリセットベクトルにはこのラベルを設定します。

```
    .glb    start
    .section    interrupt
start:
```

データバンクレジスタの設定

near 型データを配置する領域のデータバンクレジスタ値を設定します。

データバンクレジスタ

CPU アドレスの上位 8 ビットを設定します。データバンクレジスタで設定された範囲を near 領域として扱います。

注意！

コマンドオプション -bank=n で設定する値と同じ値を設定してください。
コマンドオプション -fDP_offset_8 を使用する場合は、”.GLB __DP1 ” から ”.GLB __DP3 ” までの 3 行を削除してください。

```
__DT    .equ    00H
    .GLB    __DP1
    .GLB    __DP2
    .GLB    __DP3
        :
    ldt #__DT           ; set DTregister
```

プロセッサモードレジスタを設定します

開発するシステムの構成にあうようにプロセッサモードレジスタを設定してください。

プロセッサモードレジスタ

スペシャルファンクションレジスタ領域にある、マイコンの動作モード（シングルチップモードなど）を設定するレジスタです。

```
    movmb   DT+:005eH,#24H   ; set PM0register
    movmb   DT+:005fH,#02H   ; set PM1register(DPR0,1,2,3)
```

nc79 の起動オプション -fDP_offset_8(-fDPO8)でダイレクトページレジスタを 8 ビットオフセットモードで使用するように選択する場合は、次に示すように設定してください。

```
    movmb   DT+:005eH,#24H     ; set PM0register
    movmb   DT+:005fH,#00H     ; set PM1register(DPR0 only)
```

NC79 の起動オプション -bank= で near 領域のバンクを 0 以外に設定したときは、次に示す方法で設定してください。

```
    ldab        A,#24H          ; set PM0register
    stab        A,LG:5eH
    ldab        A,#02H          ; set PM1register(DPR0,1,2,3)
    stab        A,LG:5fH
```

スタックポインタ設定部

スタックポインタ（S）の値を設定します。

注意！

この部分は書きかえないでください。

```
        lda.W   A,#OFFSET stack_top - 1 ; set stack pointer
        tas
```

near 領域の bss セクションをゼロクリアします

near 領域の bss セクションをゼロクリアします。

- ANSI-C 規格では初期値を持たない RAM 上の変数はゼロクリアすることになっています。bss セクションのゼロクリアはこの規格に従って行っています。開発するシステムの構成にあわせてゼロクリアをしないようにコメントにしてください。
- また、一括して RAM 領域を一度にゼロクリアするように変更することも可能です。

bss セクション
: bss セクションは C 言語で初期値を持たない RAM 上の変数の領域（セクション）です。

セクション名

bss_NE、bss_NO などはセクション名です。アンダーバー（_）の後の文字は以下の内容を示します。

N: near 領域にある変数

F: far 領域にある変数

E: 偶数サイズの変数

O: 奇数サイズの変数

I: データの初期値を保持するセクション

```
;----------------------------------------
;       Zero clear for data area
;----------------------------------------
BZERO   bss_NE,bss_NE
BZERO   bss_NO,bss_NO
```

以下の部分では、ダイレクトページレジスタを 6 ビットオフセットモードで使用する場合の各セクションをゼロクリアしています。

注意！

コマンドオプション -fDP_offset_8 を使用する場合は全て削除してください。

```
;----------------------------------------
;   Zero clear for data area
;----------------------------------------
ldd (1,2,3),#__DP1,#__DP2,#__DP3
BZERO   bss_DP1E,bss_DP1E
BZERO   bss_DP1O,bss_DP1O
BZERO   bss_DP2E,bss_DP2E
BZERO   bss_DP2O,bss_DP2O
BZERO   bss_DP3E,bss_DP3E
BZERO   bss_DP3O,bss_DP3O
lda.W   A,#OFFSET stack_top - 1 ; reset stack pointer
tas
```

near 領域の data セクションを初期化します

ROM 上に格納されている変数の初期値を RAM に転送する処理を行います。

初期値を持つ RAM 上の変数が無い場合はこの部分をコメントにしてください。また、各セクションが連続している場合は初期値を一度に転送するように変更することも可能です。

data セクション

C 言語で初期値をもつ RAM 上の変数の領域（セクション）です。

```
        ;       Initialize for data area
        ;---------------------------------------
        BCOPY   data_INE,data_NE,data_NE
        BCOPY   data_INO,data_NO,data_NO
```

以下の部分では、ダイレクトページレジスタを 6 ビットオフセットモードで使用する場合の各セクションを初期化しています。

注意！

コマンドオプション -fDP_offset_8 を使用する場合は全て削除してください。

```
    ;---------------------------------------
    ;   Initialize for data area
    ;---------------------------------------
    BCOPY   data_IDP1E,data_DP1E,data_DP1E
    BCOPY   data_IDP1O,data_DP1O,data_DP1O
    BCOPY   data_IDP2E,data_DP2E,data_DP2E
    BCOPY   data_IDP2O,data_DP2O,data_DP2O
    BCOPY   data_IDP3E,data_DP3E,data_DP3E
    BCOPY   data_IDP3O,data_DP3O,data_DP3O
    lda.W   A,#OFFSET stack_top - 1 ; reset stack pointer
    tas
```

far 領域の bss セクションをゼロクリアします

far 領域の bss セクションをゼロクリアします。

注意！

far 領域の RAM データを使用しない場合やシングルチップモードで使用する場合はコメントにしてください。

```
        ;---------------------------------------
        ;       Zero clear for data area
        ;---------------------------------------
        BZERO   bss_FE,bss_FE
        BZERO   bss_FO,bss_FO
```

far 領域の data セクションを初期化します

far 領域の data セクションの初期値を RAM 領域に転送します。

注意！

far 領域の RAM データを使用しない場合やシングルチップモードで使用する場合はコメントにしてください。

```
        ;       Initialize for data area
        ;---------------------------------------
        BCOPY   data_IFE,data_FE,data_FE
        BCOPY   data_IFO,data_FO,data_FO
```

スタックポインタの再設定部

BZERO マクロ等で使用したスタックを元に補正します。

注意！

この部分は書きかえないでください。

```
        lda.W   A,#OFFSET stack_top - 1 ; reset stack pointer
        tas
```

heap 領域を初期化します

メモリ管理関数で使用する heap 領域の初期化を行います。

注意！

メモリ管理関数を使用しない場合はコメントにしてください。

- heap

  メモリ管理関数（malloc など）によって、プログラム実行中に動的に確保されるメモリ領域です。

```
;       Initialize for HEAP area
;---------------------------------------
.glb    __mbase, __mnext, __msize
ldad    E, heap_top
stad    E, __mbase
stad    E, __mnext
ldad    E, HEAPSIZE
stad    E, __msize
```

標準入出力の初期化を行います

標準入出力の初期化を行う init 関数を呼び出します。

sprintf、sscanf 関数のみを使用する場合は、init 関数を呼び出す必要はありません。

注意！

入出力関数を使用しない場合は必ずコメントにしてください。

```
;---------------------------------------
;       Call init( initialize standard I/O )
;---------------------------------------
.glb    _init
jsrl    _init
```

main 関数の呼び出し部

main 関数の呼び出しを行います。

注意！

この部分は書きかえないでください。
main 関数呼び出し時には M、X フラグともにクリアされた状態で呼び出してください。

```
        ;---------------------------------------
        ;       Call main
        ;---------------------------------------
        .glb    _main
        jsrl    _main
```

exit 関数部

プログラムを終了させる exit 関数部です。

```
        ;---------------------------------------
        ;       Call exit
        ;---------------------------------------
        .glb    _exit
_exit:
        bra     _exit
```

ダミー割り込み処理関数部

ダミーの割り込み処理関数部です。

```
        ;---------------------------------------
        ;       Call dummy_int( dummy interrupt function )
        ;---------------------------------------
dummy_int:
        rti
        .end
```

スタートアップサンプルプログラム sect79.inc の内容

プログラムやデータ等をメモリのどの領域に配置するかを設定します。配置はセクション単位で行います。ncrt0.a79 ファイルからインクルードされます。

スタックサイズを設定します

使用するスタックサイズを定義してください。

スタック領域は以下のデータを格納する領域です。

- 記憶クラス auto の変数
- 複雑な演算を実行する場合のテンポラリ
- 関数呼び出し元への戻りアドレスと旧フレームポインタアドレス
- 関数への引数
- 64 ビットの浮動小数点を格納する内部レジスタを配置

スタックサイズは、スタックサイズ算出ユーティリティ Stk Viewer を用いて求めることができます。

注意！

Stk Viewer の使用方法は「NC79WA ヒント集」の「Stk Viewer を使ってスタック使用量を計算しよう」を参照してください。

```
;======================================
;       NEAR
;======================================
STACKSIZE       .equ    300H
HEAPSIZE        .equ    300H
```

heap サイズを設定します

使用する heap サイズを定義してください。メモリ管理関数を使用しないときは heap サイズを 0 にしてください。

```
;======================================
;       NEAR
;======================================
STACKSIZE       .equ    300H
HEAPSIZE        .equ    300H
```

near 領域の RAM 配置

サンプルでは、RAM データ領域の配置を行っています。配置を変更する場合はアセンブリ言語の .ORG 指示命令で番地を指定してください。番地が指定されていないセクションはその前のセクションの直後に配置されます。

```
;---------------------------------------
;       RAM DATA
;---------------------------------------
        .section        data_NE ,DATA
        .org            000800H
        .section        data_NO ,DATA
        .section        bss_NE  ,DATA
        .section        bss_NO  ,DATA
```

スタックセクション配置

スタックセクションを配置します。

```
        .section        stack   ,DATA
        .blkb           STACKSIZE
stack_top:
```

heap セクション配置

メモリ操作関数を使用しない場合はコメントにしてください。

サンプルでは、セクションの番地を指定していないため、その前のセクションの直後、すなわち near 領域に配置されますが、heap セクションは far 領域の RAM に配置することもできます。その場合は、次の部分を far 領域に配置されるように移動してください。

```
        .section        heap    ,DATA
        .blkb           HEAPSIZE
heap_top:
```

外部変数格納セクションを設定

拡張機能 ” #pragma DP[n]DATA ” を使用して割り当てられた外部変数を格納するセクションを設定します。開始アドレスはアセンブラ指示命令 .ORG を用いて設定してください。

注意！

拡張機能 ” #pragma DP[n]DATA ” を使用しない場合は以下の部分すべてを削除してください。また、コマンドオプション -fDP_offset_8 を使用する場合も削除してください。

```
    .section    data_DP1E,DATA
    __DP1:
    .section    data_DP1O,DATA
    .section    bss_DP1E,DATA
    .section    bss_DP1O,DATA

    .section    data_DP2E,DATA
    __DP2:
    .section    data_DP2O,DATA
    .section    bss_DP2E,DATA
    .section    bss_DP2O,DATA

    .section    data_DP3E,DATA
    __DP3:
    .section    data_DP3O,DATA
    .section    bss_DP3E,DATA
    .section    bss_DP3O,DATA
```

割り込み処理関数配置

割り込み処理関数を配置します。

```
;----------------------------------------
;       PROGRAM
;----------------------------------------
        .section        interrupt
        .org            004000H
```

near 関数配置

near 修飾された関数を配置します。

- シングルチップモードで使用する場合や関数の一部をバンク 0 に配置する場合に設定します。

```
        .section        program_N
```

near 領域の ROM 配置

near 領域に ROM データを配置しない場合は、以下の部分をコメントにしてください。

```
;---------------------------------------
;       ROM DATA
;---------------------------------------
        .section        rom_NE  ,ROMDATA
        .section        rom_NO  ,ROMDATA
        .section        data_INE,ROMDATA
        .section        data_INO,ROMDATA
```

外部変数格納セクションの配置

拡張機能 ” #pragma DP[n]DATA ” を使用して割り当てられた外部変数の ROM 上のセクションを配置します。

注意！

拡張機能 ” #pragma DP[n]DATA ” を使用しない場合は以下の部分すべてを削除してください。また、コマンドオプション -fDP_offset_8 を使用する場合も削除してください。

```
    .section    data_IDP1E,ROMDATA
    .section    data_IDP1O,ROMDATA
    .section    data_IDP2E,ROMDATA
    .section    data_IDP2O,ROMDATA
    .section    data_IDP3E,ROMDATA
    .section    data_IDP3O,ROMDATA
```

割り込みベクタアドレスの設定

vector セクションでは割り込みベクタアドレスを設定します。

- 割り込みベクタの内容は、7900 シリーズの機種により異なります。使用するマイコンにあわせて修正してください。
- 割り込み処理関数を設定するには ” dummy_int ” の部分を割り込み処理関数の先頭ラベル名（関数名に _ を付加した名前）に書きかえてください。

```
;---------------------------------------
;  VECTOR TABLE
;---------------------------------------
   .section    vector
   .org        0ffc0H
DMA3:       .word       dummy_int
DMA2:       .word       dummy_int
DMA1:       .word       dummy_int
DMA0:       .word       dummy_int
ROMC:       .word       dummy_int
RESERVED3:  .word       dummy_int
RESERVED2:  .word       dummy_int
RESERVED1:  .word       dummy_int
INT4:       .word       dummy_int
INT3:       .word       dummy_int
AD:         .word       dummy_int
UART1S:     .word       dummy_int
UART1A:     .word       dummy_int
UART0S:     .word       dummy_int
UART0A:     .word       dummy_int
TB2I:       .word       dummy_int
TB1I:       .word       dummy_int
TB0I:       .word       dummy_int
TA4I:       .word       dummy_int
TA3I:       .word       dummy_int
TA2I:       .word       dummy_int
TA1I:       .word       dummy_int
TA0I:       .word       dummy_int
INT2:       .word       dummy_int
INT1:       .word       dummy_int
INT0:       .word       dummy_int
NMI:        .word       dummy_int
WDT:        .word       dummy_int
DBC:        .word       dummy_int
BK:         .word       dummy_int
DIV0:       .word       dummy_int
RESET:      .word       OFFSET start
```

far 領域の RAM 配置

シングルチップモードで使用する場合等、この領域を使用しない場合はコメントにしてください。

```
;       FAR
;=======================================
;---------------------------------------
;       RAM DATA
;---------------------------------------
        .section        data_FE ,DATA
        .org            12000H
        .section        data_FO ,DATA
        .section        bss_FE  ,DATA
        .section        bss_FO  ,DATA
```

far 関数配置

far 修飾された関数を配置します。デフォルトでは、関数はこのセクションに配置されます。

```
;---------------------------------------
;       PROGRAM
;---------------------------------------
        .section        program_F
        .org            18000H
```

far 領域の ROM 配置

サンプルでは、ROM データ、RAM データの初期値、プログラムを配置しています。

```
;---------------------------------------
;       ROM DATA
;---------------------------------------
        .section        rom_FE  ,ROMDATA
        .section        rom_FO  ,ROMDATA
        .section        data_IFE,ROMDATA
        .section        data_IFO,ROMDATA
```

memo

NC79WA ヒント集

Stk Viewer を使ってスタック使用量を計算しよう

Stk Viewer はプログラムが使用するスタック容量を算出するプログラムです。

Stk Viewer を使用するには NC79 実行時にコマンドオプション -finfo を指定してインスペクタ情報を付加しておく必要があります。

サンプルファイル samp79.c のスタック使用量を算出する例を示します。

1　インスペクタ情報をコンパイル時に付加する。

nc79 コマンドにオプションを付加してスタック使用情報ファイルを生成します。

```
>nc79 -c -info samp79.c
```

2　Stk Viewer を起動する。

以下のいずれかの方法により Stk Viewer を起動します。

a)　ウィンドウズのスタートメニューから
［プログラム］→［MITSUBISHI-TOOL］→［NC79WA V.x.xx Releasex］→
［Stk Viewer V.x.xx.xx］を選択する。

b)　TM からの起動
TM の[Stk Viewer の実行]ボタンをクリックする。

3　スタック使用量を計算する

ツールバーの[Open]ボタンをクリックすると、[ファイル選択ダイアログ]が開きます。ここで samp79.x79 を指定し、[開く]ボタンをクリックすると、スタック使用量が計算され、Stk Viewer の [Main Window]に関数の呼び出し関係がツリー表示されます。関数名とともにそれぞれの関数から積まれる最大のスタック使用量が表示されます。（図 1. 参照）

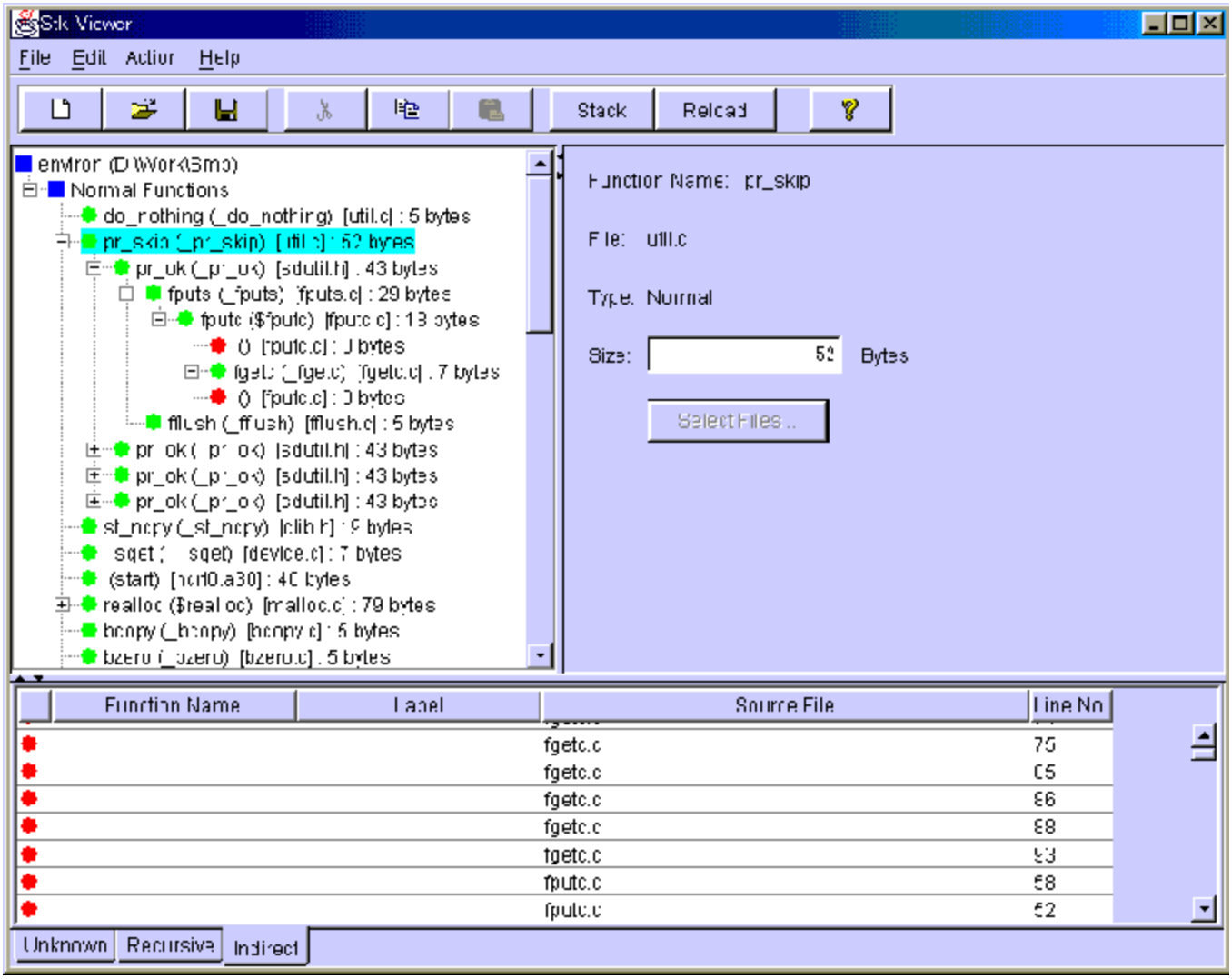


図 1：Main Window

utl79 を使って使用頻度の高い変数を DPnDATA に割り当てよう

utl79 はアブソリュートモジュールファイル(.x79)を使用して使用頻度の高い変数から順に DPnDATA に割り当てる宣言を出力します。

utl79 を使用するときは、nc79 実行時にコマンドオプション -finfo を指定してください。

次に、サンプルファイル sample.c を使って utl79 の使用例を示します。

sample.c の内容

```
int data;
main()
{
   int i;

   data=1;
   i=data;
}
```

コマンド入力例

```
>nc79 -c -finfo ncrt0.a79 sample.c            ...(1)
>ln79 -G -l  nc79lib -o sample ncrt0 sample   ...(2)
>utl79 sample.x79 -odpdata.h                  ...(3)
```

(1) コンパイラオプション -finfo を付加し、データ使用状況情報を .x79 に出力します。

(2) リンクオプション -G を付加し、アブソリュートモジュールファイルを生成します。

(3) DPnDATA 宣言を dpdata.h ファイルに出力します。utl79 はアブソリュートモジュールファイル(x79)に登録されている全てのソースファイルに対するデータ使用状況情報を読み込んで処理します。

dpdata.h の内容

```
/*
 *    #pragma DPnDATA Utility
 */
/* DP1DATA Size [63] */
#pragma  DP1DATA  data   /* size = (2) / ref = [     2] */
/*
 *    End of File
 */
```

dpdata.h の読み込み

utl79 が生成したヘッダファイルを C ソースプログラム中でインクルードします。

```
#include   "dpdata.h"

func()
{
    :
}
```

Map Viewer を使ってマップ情報を確認しよう

Map Viewer はアブソリュートモジュールファイル(.x79 ファイル)のマップ情報を GUI 表示するプログラムです。このプログラムは PC 版のみの対応です。
Map Viewer の使用例を示します。

1　Map Viewer を起動する。

以下のいずれかの方法により Map Viewer を起動します。

a)　ウィンドウズのスタートメニューから、[プログラム]→[MITSUBISHI-TOOL]→[NC79WA V.x.xx Release x]→[Map Viewer V.x.xx.xx]を選択する。

b)　コマンドラインから以下のコマンドを実行する。

```
>C:\Mtool\bin\MapViewer.exe
```

c)　TM から起動する。
TM の[Map Viewer]ボタンをクリックする。

2　各種情報を確認する。

ツールバーの[Open]ボタンまたはファイルメニューの[Open]によりアブソリュートモジュールファイルを選択すると、図 1 のマップ情報が表示されます。
図 1 では、左側ウィンドウにセクション配置の概要、右側ウィンドウにセクション配置の詳細が表示されます。セクション配置の詳細には以下の項目が表示されます。

- セクション開始アドレス
- セクションサイズ
- セクション属性（[D]:DATA、[R]:ROMDATA、[C]:CODE）
- セクション名
- ラベル情報（アドレスおよびラベル名）
- .EQU 情報（[Map View]ダイアログの[.EQU List]により値、名前を表示）
- 関数情報（[Map View]ダイアログの[C Function List]により関数開始アドレス、サイズ、関数名を表示）

右側ウィンドウでは、[MapView]ダイアログにより表示内容を選択できます。
また、左側ウィンドウでは[Memory Size Image]ボタンにより、セクションの配置位置をメモリ配置イメージで表示できます。[Section List Image]ボタンによりセクションリストイメージを表示します。

図 2 に[Memory size Image]ボタンおよび[Map View]ダイアログで[C Function List]を選択した例を示します。

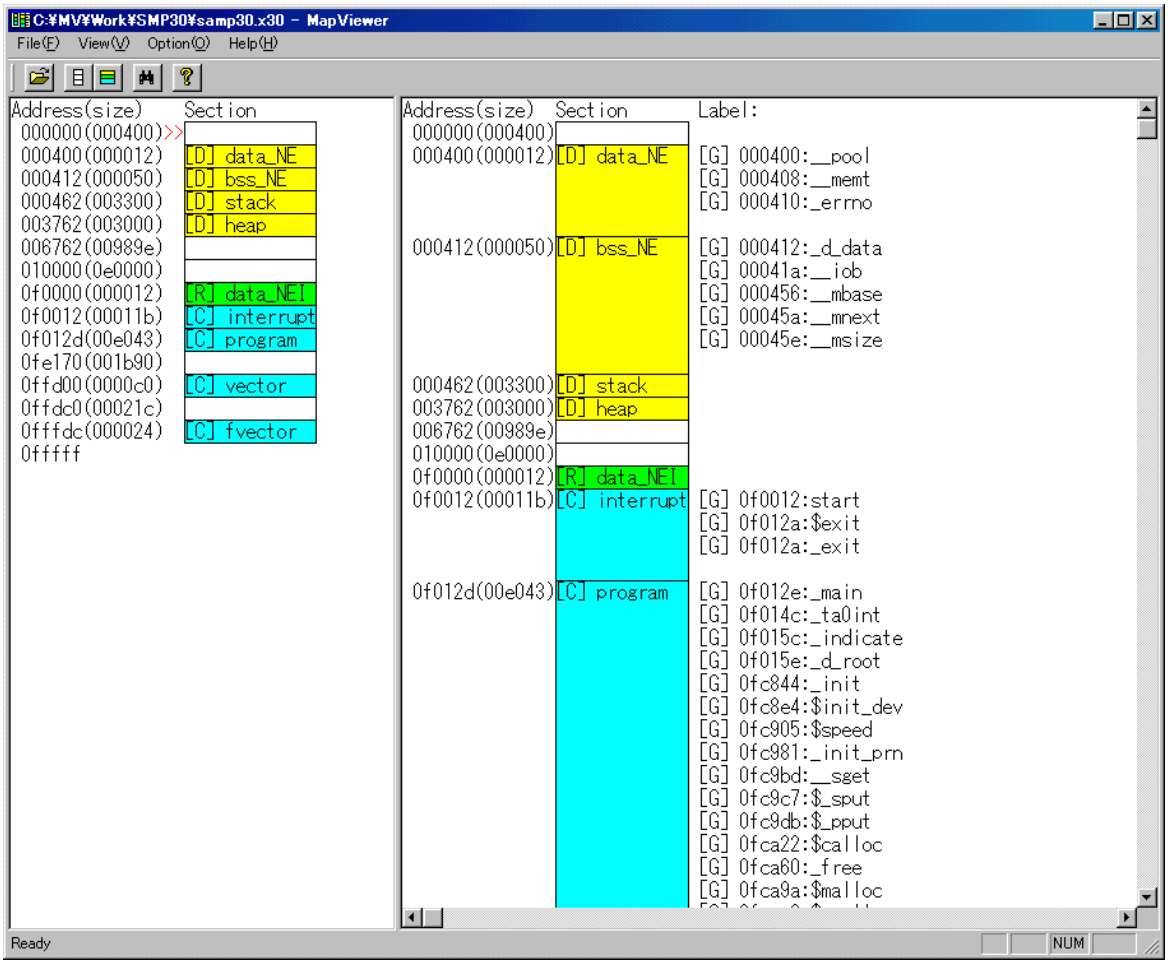


図 1：Map Viewer

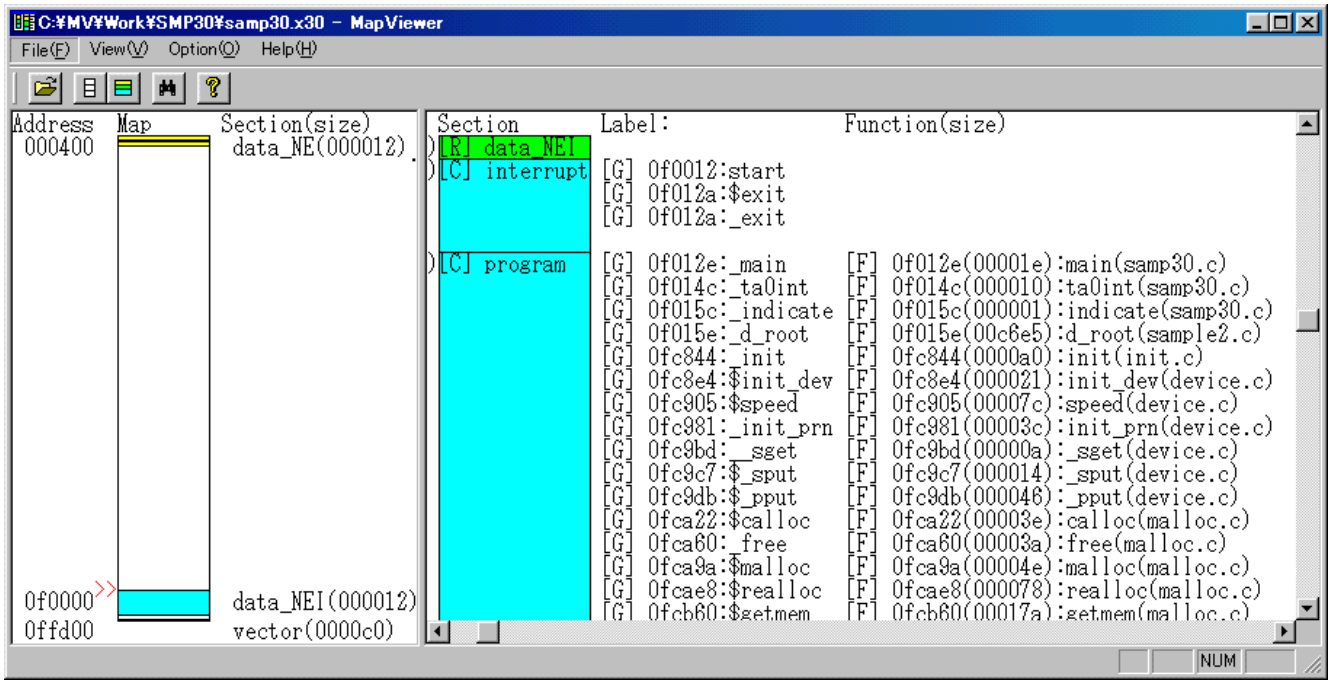


図 2：[Memory Size Image]ボタン、[Map View]ダイアログで[C Function List]を選択した表示例

xrf79 を使ってアセンブララベルの参照関係を調べよう

xrf79 はアセンブリ言語ファイルを読み込み、ラベルの定義、参照関係を行情報で表示したクロスリファレンスファイル(拡張子 .xrf)を生成します。

サンプルプログラム sample.a79 のクロスリファレンスファイルを生成してみましょう。sample.xrf ファイルが生成されます。

```
>xrf79 sample.a79
```

注意！

ファイル拡張子は（.a79）必ず指定してください。

abs79 を使ってリストファイルに絶対番地情報を付加しよう

abs79 はリストファイル（拡張子 .lst）とアブソリュートモジュールファイル（拡張子 .x79）からアブソリュートアドレス情報を持つリストファイル(拡張子 .als)を生成します。

例 1 サンプルプログラム sample.a79 のアブソリュートリストファイルを生成してみましょう。sample.als が生成されます。

注意！

マクロ命令、条件アセンブル命令を使用しているファイルからアブソリュートリストファイルを生成する場合は必ず as79 のコマンドオプション "-lmi" を指定してください。

```
>as79 -lmi sample
>ln79 sample -order ram1=80,prog1=0e0000
>abs79 sample
```

例 2 構造化記述文を含むサンプルファイル struct.a79 をもとにアブソリュートリストファイルを生成してみましょう。struct.als が生成されます。

注意！

構造化記述文を使用しているファイルからアブソリュートリストファイルを生成する場合は必ず as79 のコマンドオプション "-ls" を指定してください。

```
>as79 -p -ls struct
>ln79 struct
>abs79 struct
```

プログラムを ROM に書き込む準備をする

デバッグが終了したアプリケーションプログラムは、実際の応用製品により近い環境で動作評価をする必要があります。このとき使用する EPROM 内蔵マイコンやワンタイム PROM 内蔵マイコンにプログラムを書き込むためには次の準備が必要です。

1　ROM ライタ対応フォーマットファイルを生成

2　書き込みアダプタなどのツールを用意

ROM ライタ対応フォーマットファイルを生成

lmc79 コマンドでアブソリュートモジュールファイル（拡張子 .x79）を ROM ライタ対応フォーマットファイルに変換します。

注意！

プログラムサイズが 1Mbytes のアドレス空間を超える場合は、モトローラ S フォーマットファイルを使用してください。

モトローラ S フォーマットファイルを生成

lmc79 コマンドで次のように入力してください。sample はアブソリュートモジュールファイル名（拡張子は省略）です。sample.mot ファイルが生成されます。

```
>lmc79 sample
```

インテル HEX フォーマットファイルを生成

lmc79 コマンドで次のように入力してください。sample はアブソリュートモジュールファイル名（拡張子は省略）です。sample.hex ファイルが生成されます。

```
>lmc79 -H sample
```

注意！

どちらのフォーマットが必要になるかは、使用される ROM ライタの仕様をご確認ください。

書き込みアダプタなどのツールを用意

製品に添付されているデータシートやホームページ)に掲載されている最新の情報を参考にして必要なハードウェアを用意してください。

```
http://www.tool-spt.mesc.co.jp
```

Q&A 集

コマンドオプションが指定できない場合がありますがなぜですか？

NC79 では、コマンドオプションの大文字と小文字を区別しています。コマンドオプションを指定する場合には、大文字小文字が一致していることを確認してください。

リンクするファイル数が多くてコマンド行の制限を越えてしまいます。リンクする方法はありますか？

コマンド行の内容をファイルに書き込んで ln79 を実行させることができます。このファイルをコマンドファイルと呼びます。コマンドファイルについての詳しい情報は次の項を参照してください。

「コマンドファイル」はどのように使うのですか？

ソフトウェアの実行時にコマンドファイルを指定するときは、通常のファイルとの区別のために、ファイル名の先頭に '@' を付加して指定します。

ln79 のコマンドファイル指定例）

```
>ln79 @cmdfile
```

コマンドファイル記述例）

```
sample
-order ram1,prog1=0f0000
-m -t
```

このコマンドファイル例は、次のように ln79 コマンドを入力した場合と同じ結果になります。

```
>ln79 sample -order ram1,pog1=0f0000 -m -t
```

#include 指定でファイル名にパスを指定できますか？

相対パス名を指定できます。コンパイラは、#include で指定されているファイルを #include が記述されているファイルからの相対ディレクトリで検索します。絶対パス指定は、プログラムの移植性をそこなう可能性があるため禁止しています。

src_1.c ファイル内で src_1.h を #include で指定する場合の記述は次のようになります。

src_1.c の記述

```
#include   "..¥header¥src_1.h"
    :
```

ファイル構成

```
C:¥work¥src_1.c
C:¥work¥src_2.c
    :
C:¥header¥src_1.h
C:¥header¥src_2.h
    :
```

注意！

このとき、パスを含めたファイル名の指定を "\header\src_1.h" と記述するとエラーとなります。

レジスタ（register）記憶クラスを指定して、それをレジスタに割り当てるには？

ローカル変数にレジスタ記憶クラスを指定し、その変数を確実にレジスタに割り当てるには、コンパイル時にコマンドオプション "-fenable_register(-fER)" を指定してください。ただし、一部の型を持つ変数はレジスタに割り当てることができません。

レジスタ変数の宣言例）

```
register int   i
```

また、レジスタ変数は関数の中で、関数呼び出しをまたいで使用しないように記述してください。

register修飾子を付けていない変数がレジスタ変数として扱われていますがなぜですか？

最適化コマンドオプション '-O'、'-OR' および '-OS' を指定してコンパイルした場合、最適化が行われた結果、レジスタ変数として扱われる場合があります。

C言語で記述した関数名はアセンブリソース上ではどのような名前になるのですか？

関数名はアセンブリソース上ではラベル名として現れます。たとえば、関数名が "func_proto" のとき、アセンブリソース上では "$func_proto:" または "_func_proto:" として表されます。

"$func_proto:" と "_func_proto:" で、どのような場合にどちらのラベルが選択されるかの規則は次のように決まっています。

$func_proto

関数の引数のうち、一つ以上の引数がレジスタで渡されている関数

_func_proto

上記の条件にあてはまらないすべての関数

関数名に相当するラベルにたいしてリンク時に未定義シンボルであるというエラーが出力されますが原因は？

関数のプロトタイプが宣言されているものとされていないものが混在している可能性があります。

プロトタイプ宣言がされていない関数を見つけるには、コマンドオプション "-Wnon_prototype(-WNP)" を指定してコンパイルを実行すると、該当する関数に対してワーニングが出力されます。見つけだした関数をプロトタイプ宣言してください。

関数のプロトタイプ宣言は必要ですか？

関数をプロトタイプ宣言することで、関数の引数の型拡張を抑止したり、引数をレジスタに割り当てるなど、効率の良い関数呼び出しが可能になります。

最適化オプションをつけるとコードが生成されない行がありますが問題ありませんか？

コンパイラは、最適化オプションが指定されると文法上無意味であると思われるコードを生成しません。

最適化を抑止したい場合は、"volatile" 修飾子を指定して最適化されないようにしてください。

最適化される例

- port を読み出すだけの関数

```
extern int port;
func()
{
   port;
}
```

最適化されない例

```
volatile extern int    port;
func()
{
   port;
}
```

near/far 属性はどのように区別されますか？

NC79 は near/far が指定されていない場合は、関数を far、変数を near として扱います。

変数は通常 near として扱われますが、コマンドオプションを指定するとデフォルトの属性を変更できます。

-ffar_ROM_data(-fFROM)
: ROM データのデフォルト属性を far にします。

-ffar_RAM_data(-fFRAM)
: RAM データのデフォルト属性を far にします。

ROM 容量を圧縮するには？

far 領域に配列データがあって、そのサイズが 64K バイトを越えないとき有効な方法として、要素数を指定していない配列が far 領域にある場合に、-fsmall_array (-fSA)オプションを付加してコンパイルを実行してください。ROM 容量を圧縮し、処理速度を高速にできる可能性があります。

スタートアッププログラムの処理時間を短くするには？

スタートアッププログラム ncrt0.a79 に記述されているセクションの初期化およびゼロクリアのうち使用しない部分を削除することにより処理時間を短縮できます。また、ゼロクリアを一括して行うことにより処理時間を短縮できる場合があります。

高級言語ソースレベルでデバッグを行うには？

コマンド実行時に、C のソース行や構造化記述文のソース行でデバッグを行うための情報を機械語ファイルに出力するようにコマンドオプションを指定してください。

```
>nc79 -g -S -c samp79.c
>as79 -s ncrt0.a79 samp79.a79
>ln79 -g -o samp79 ncrt0 samp79 -l nc79lib.lib
```

NC77 用に記述したプログラムを NC79 で使用するには？

1　C コンパイラの言語仕様レベルでは互換性がありますが、アセンブラレベルでの関数名の扱いが異なっています。アセンブラ記述で関数名と同じになるようなラベルを使用していないか確認してください。

	NC79	NC77
レジスタ引数を持つ関数	$function	?function
レジスタ引数を持たない関数	_function	_function

2　NC79 コンパイラを実行する際に、オプション -fDP_offset_8(-fDPO8)を必ず指定してください。

RASM77 用アセンブリプログラムとの互換性について

7700 ファミリ用アセンブラ RASM77 と 7900 シリーズ用アセンブラ AS79 には、以降に示すような仕様の相違があります。

7700 ファミリのマイクロコンピュータ用に開発されたアプリケーションを 7900 シリーズのマイクロコンピュータのアプリケーションプログラムに流用される場合は、以降の内容をご参照のうえプログラムの必要な個所を変更してください。

注意！

それぞれの機能の詳細については、各アセンブラのユーザーズマニュアルを参照してください。

算術演算子

演算の優先順位が異なっています。したがって、式の記述内容によっては、演算結果が異なります。

AS79

二項演算子に一般の四則演算規則と同様の演算優先順位を設定しています。

RASM77

二項演算子に演算優先順位の設定はありません。複数の二項演算子が記述されている式の演算は左から順に行われます。

次に演算結果の異なる式の記述例を示します。

AS79	RASM77
3|4+5*6	3|4+5*6
=3|4+30	=7+5*6
=3|34	=12*6
=35	=72

アドレッシング指定子と EQU シンボル（オフセット計算）

ダイレクトアドレッシングモードまたは、アブソリュートアドレッシングモードが指定されている場合のオペランドの計算方法が異なります。

AS79

特別に指定をしない限り、オペランドの種類に関わらず、オペランドとダイレクトページレジスタまたは、データページレジスタとのオフセット値を計算し、その結果をオペランドコードとして生成します。詳細はユーザーズマニュアルを参照してください。

RASM77

オペランドが疑似命令'.EQU'で定義されたシンボルの場合、オペランドとダイレクトページレジスタ又は、データページレジスタとのオフセット計算を行いません。

次に、ダイレクトアドレッシングモードでオペランドが'.EQU'で定義されたシンボルを記述した場合のオペランド値の相違を示します。

AS79

```
sym.EQU   34h
   .DP    12h
   LDA    A,DP:sym
   ;CODE is 1A22
```

RASM77

```
sym .EQU   34h
    .DP    12h
    LDA    A,DP:sym
    ;CODE is A534
```

ロケーションシンボル

ロケーションシンボルが異なっています。次にロケーションシンボルを使用したプログラムの記述例を示します。

AS79

```
BCC   $+3   ;If C flag is 0, branch to NOP
INX
NOP
```

RASM77

```
BCC   *+3   ;If C flag is 0, branch to NOP
INX
NOP
```

アセンブル指示命令

AS79 では一部の指示命令が RASM77 とは異なっています。

RASM77 から削除された指示命令

```
.ERROR      .LIB        .OBJ        .LISTM      .NLISTM
.PROGNAME   .ENDIO      .ENDRAM     .IO         .RAM
.PROCINT    .PROCMAIN   .PROCSUB
```

RASM77 とは機能が異なる指示命令

```
.EVEN       .PUB        .EXT        .DPEXT      .DTEXT
.COL        .LINE       .LIST       .NLIST
```

次に、指示命令毎に相違内容を説明します。

.EVEN

アドレス補正命令 '.EVEN' は、'.ALIGN' に置き換えてください。

AS79

```
.ALIGN 2
```

RASM77

```
.EVEN
```

.PUB

パブリック指定命令 '.PUB' は、'.GLB' に置き換えてください。

AS79

```
.GLB    esym
```

RASM77

```
.PUB    esym
```

.EXT

外部参照指定命令 '.EXT' は、'.GLB' に置き換えてください。

AS79

```
.GLB    esym
```

RASM77

```
.EXT    esym
```

.DPEXT

ダイレクトアドレッシング外部参照指定命令'.DPEXT'は、'.GLB'及び'.DPSYM'に置き換えてください。

注意！

AS79では、「ダイレクトページ名ラベル(directpage_name)」に相当する機能はありません。

AS79	RASM77
.GLB esym .DPSYM esym	.DPEXT esym

.DTEXT

アブソリュートアドレッシング外部参照指定'.DTEXT'は、'.GLB'及び'.DTSYM'に置き換えてください。

注意！

AS79では、「バンク名ラベル(databank_name)」に相当する機能はありません。

AS79	RASM77
.GLB esym .DTSYM esym	.DTEXT esym

.COL

リストファイルカラム数指定命令'.COL'は、'.FORM'に置き換えてください。.FORM命令でリストファイルのカラム数を指定する場合は、第2オペランドにカラム数を指定してください。

AS79	RASM77
.FORM ,100	.COL 100

.LINE

リストファイル行数指定命令'.LINE'は、'.FORM'に置き換えてください。.FORM命令でリストファイルの行数を指定する場合は、第1オペランドに行数を指定してください。

AS79	RASM77
.FORM 60	.LINE 60

.LIST/.NLIST

リスト出力制御 '.LIST' 及び '.NLIST' は、それぞれ ".LIST　ON"、".LIST　OFF" に置き換えてください。

AS79

```
.LIST  OFF
   :
.LIST  ON
```

RASM77

```
.NLIST
   :
.LIST
```

マクロ命令に関する相違点

マクロ名

マクロ名の直後のコロン(:)を削除してください。

AS79

```
Fclr   .MACRO
       CLP C,Z,I
       .ENDM
```

RASM77

```
Fclr:   .MACRO
        CLP C,Z,I
        .ENDM
```

繰り返しマクロ命令

繰り返しマクロ命令 '.REPEAT ～ .ENDM' は、'.MREPEAT ～ .ENDR' に置き換えてください。

RASM77 の繰り返しマクロ命令 '.REPEATC' 及び '.REPEATI' は、AS79 では対応していません。

AS79

```
.MREPEAT   3
NOP
.ENDM
```

RASM77

```
.REPEAT 3
NOP
.ENDM
```

文字列連結演算子に関する相違点

文字列連結演算子 '$' は、'@' に置き換えてください。

AS79

```
Obyte  .MACRO  Oval
       .BYTE   Oval@o
       .ENDM
```

RASM77

```
Obyte:  .MACRO  Oval
        .BYTE   Oval$o
        .ENDM
```

付録 A NC79WA コマンドオプション一覧

■ nc79コマンドオプション

● nc79の起動オプション

-c

-c
リロケータブルファイル生成

-D

-Ddata1=0x2F
識別子を定義(#define と同様)

-I

-Idirname
#include ファイルの検索ディレクトリを指定

-E

-E
プリプロセスコマンドのみ起動(結果は標準出力)

-P

-P
プリプロセスコマンドのみ起動(中間ファイル生成)

-S

-S
アセンブリ言語ソースファイル生成

-U

-Umacroname
指定したプリデファインドマクロ名を無効化

-silent

-silent
起動時のコピーライトメッセージ非表示

-time

-time
プロセスの処理時間を表示（UNIX 版のみ）

●出力ファイル指定オプション

-o

-ofilename
リンカの生成ファイル名称を指定

-dir

-dir/usr/disk
出力ファイルの出力先ディレクトリを指定します。

●バージョン情報表示オプション

-v

-v
実行中のコマンドプログラム名とコマンドラインを表示

-V

-V
コンパイラの各プログラムの起動時メッセージを表示し処理終了

●デバッグ用オプション

-g

-g
シンボルファイル(属性 .sym)を生成

-genter

-genter
関数呼び出し時にスタックフレームを生成

-gno_reg

-gno_reg
レジスタ変数に関するデバッグ情報の出力停止

●ライブラリ指定オプション

-l

-llib_name
ライブラリファイル名を指定

●最適化オプション

-O[1-5]

-O
-O1
-O4
速度、ROM効率ともに良くなる（影響を与えない）最適化を行います。
レベルが高いほど最適化が強くなります。
レベルを省略した場合はレベル3で処理します。
レベル5を指定する場合は必ずマニュアルを参照してください。

-OR

-OR
ROM 容量を重視した最適化を実行

-OS

-OS
速度を重視した最適化を実行

-Oconst/-OC

-Oconst/-OC
const 修飾子にたいする最適化を実行

-Ono_bit/-ONB

-Ono_bit/-ONB
ビット操作をまとめる最適化を抑止

-Ono_break_source_debug/-ONBSD

-Ono_break_source_debug/-ONBSD
ソース行情報に影響のある最適化を抑止

-Ono_float_const_fold/-ONFCF

-Ono_float_const_fold/-ONFCF
浮動小数点の定数畳み込み処理を抑止

-Ono_stdlib/-ONS

-Ono_stdlib/-ONS
標準ライブラリ関数のインライン埋め込みや変更などを抑止

-Osp_adjust/-OSA

-Osp_adjust/-OSA
スタック補正コードを取り除く最適化を実行

-Ostack_frame_align/-OSFA

-Ostack_frame_align/-OSFA
スタックフレームの偶数アライメントを実行

-Ocompare_byte_to_word/-OCBTW

-Ocompare_byte_to_word/-OCBTW
連続した領域のバイト単位の比較をワードで行います。

-Ono_logical_or_combine/-ONLOC

-Ono_logical_or_combine/-ONLOC
論理 OR をまとめる最適化を抑止します。

●生成コード変更オプション

-fansi

-fansi
-fnot_reserve_far_and_near、-fnot_reserve_asm、
-fnot_reserve_inline、及び -fextend_to_int を有効化

-fnot_reserve_asm/-fNRA

-fnot_reserve_asm/-fNRA
asm を予約語にしない(_asm のみ有効)

-fnot_reserve_far_and_near/-fNRFAN

-fnot_reserve_far_and_near/-fNRFAN
far、near を予約語にしない（_far、_near のみ有効）

-fextend_to_int/-fETI

-fextend_to_int/-fETI
char 型データを int 型に拡張

-fchar_enumerator/-fCE

-fchar_enumerator/-fCE
enumerator(列挙子)の型を unsigned char 型で処理

-fall_far/-fAF

-fall_far/-fAF
デフォルトをすべて far 型に設定

-fnear_function/fNF

-fnear_function/fNF
関数のデフォルトを near に設定

-fno_even/-fNE

-fno_even/-fNE
データをすべて odd 属性のセクションに配置

-ffar_program_section/-fFPS

-ffar_program_section/-fFAA
near/far 関数を program_F セクションに配置

-fnot_use_MVN/-fNUM

-fnot_use_MVN/-fNUM
MVN 命令でのブロック転送を抑止

-fswitch_table/-fST

-fswitch_table/-fST
switch 文に対してテーブルジャンプ方式コードを生成

-fnot_reserve_inline/-fNRI

-fnot_reserve_inline/-fNRI
inline を予約語にしない（_inline のみ有効）

-ffar_RAM_data/-fFRAM

-ffar_RAM_data/-fFRAM
RAM データのデフォルト属性を far にする

-ffar_ROM_data/-fFROM

-ffar_ROM_data/-fNROM
ROM データのデフォルト属性を far にする

-fconst_not_ROM/-fCNR

-fconst_not_ROM/-fCNR
const で指定した型を ROM データとして扱わない

-fnot_address_volatile/-fNAV

-fnot_address_volatile/-fNAV
#pragma ADDRESS(#pragma EQU)で指定した変数をvolatileとしない

-fsmall_array/-fSA

-fsmall_array/-fSA
far 型の配列の添字を 16 ビットとする

-fenable_register/-fER

-fenable_register/-fER
レジスタ記憶クラスを有効にする

-fDP_offset_8 / -fDPO8

-fDP_offset_8 / -fDPO8
8 ビットオフセットモードでコードを生成

-fuse_DIV/-fUD

-fuse_DIV/-fUD
オーバーフローを考慮せずに除算を行います。

-fauto_128/-fA1

-fauto_128/-fA1
スタックフレームサイズの上限を64バイトから128バイトに変更します。

-finfo

-finfo
インスペクタ、stk viewer、map viewer、utl30 に必要な情報を出力します。

-fno_align

-fno_align
関数の先頭アドレスのアライメントを行いません。

-bank

-bank=1
データバンクレジスタ値を指定

●警告オプション

-Wnon_prototype/-WNP

-Wnon_prototype/-WNP

プロトタイプ宣言されていない関数にたいして警告出力

-Wunknown_pragma/-WUP

-Wunknown_pragma/-WUP
サポートしていない #pragma にたいして警告出力

-Wno_stop/-WNS

-Wno_stop/-WNS
エラーが発生してもコンパイル作業を継続

-Wstdout

-Wstdout
エラーメッセージを標準出力（stdout）に出力

-Werror_file/-WEF

-Werror_file/-WEF
タグファイルを出力

-Wstop_at_warning/-WSAW

-Wstop_at_warning/-WSAW
ワーニング発生時にコンパイル処理を停止

-Wnesting_comment/-WNC

-Wnesting_comment/-WNC
コメント中の /* にたいして警告を出力

-Wccom_max_warnings/-WCMW

-Wccom_max_warnings/WCMW
ccom79 のワーニング出力数の上限を指定

-Wall

-Wall
検出可能な警告を全て表示

-Wmake_tagfile/-WMT

-Wmake_tagfile/-WMT
ファイル単位でエラーメッセージを出力します。

-Wlarge_to_small/-WLTS

-Wlarge_to_small/-WLTS
大きいサイズから小さいサイズへの暗黙の転送に対して警告を出します。

-Wuninitialize_variable/-WUV

-Wuninitialize_variable/-WUV
初期化されていない auto 変数に対して警告を出します。

-Wno_warning_stdlib/-WNWS

-Wno_warning_stdlib/-WNWS
プロトタイプ宣言されていない標準ライブラリに対する警告を抑止します。

●その他のオプション

-dsource/-dS

-dsource/-dS
アセンブリ言語ソースリストにC言語ソースをコメントとして出力します。生成された .a30 および .r30 ファイルを削除しません。

■ utl79 コマンドオプション

-all

-all
全てのデータを SBDATA 宣言します。

-o

-ofilename
出力ファイル名を指定します。

-DP1

-DP1
DP1 レジスタに変数を割り当てます。

-DP2

-DP2
DP2 レジスタに変数を割り当てます。

-DP3

-DP3
DP3 レジスタに変数を割り当てます。

-Tpointer

-Tpointer
near ポインタ変数を対象にします。

-Wstdout

-Wstdout
エラーおよびワーニングメッセージを標準出力へ出力します。

■ as79コマンドオプション

-.

-.
画面へのメッセージ出力を停止

-A

-A
アブソリュートアドレッシングを選択

-B

-B
64K バイトモードで分岐命令を最適化

-C

-C
as79 が mac79、pre79 および asp79 に渡したコマンド行を表示

-D

-Dsymbol=0
シンボルを定義

-F

-F
..FILE が示すファイル名をソースファイル名に固定

-finfo

-finfo
NC の '-finfo' オプションで生成された各情報、またはアセンブラ指示命令で記述されたインスペクタ情報をリロケータブルモジュールファイル出力する

-G1

-G1
指示命令によるサイズ指定を無視

-G2

-G2
代入文（構造化記述命令）に対する生成命令を制御

-H

-H
リストファイルへのヘッダメッセージの出力を停止

-I

-I
インクルードファイルの検索ディレクトリを指定

-JF

-JF
外部シンボルを FAR 属性で処理

-JN

-JN
外部シンボルを NEAR 属性で処理

-L[C¦D¦I¦M¦S]

-L
アセンブラリストファイルを生成
-LC
行連結をそのままリストファイルに出力
-LD
.DEFINE指示命令を置き換える前の文字列をリストファイルに出力
-LI
条件アセンブルの判断結果が偽の部分をリストファイルに出力
-LM
マクロ展開行をリストファイルに出力
-LS
構造化記述命令の展開行をリストファイルに出力

-M6

-M6
6 ビットオフセットモードでダイレクトアドレッシングのコードを生成

-M8

-M8
8 ビットオフセットモードでダイレクトアドレッシングのコードを生成

-N

-N
リストファイルへのシンボルリスト出力を抑止

-O

```
-OC:¥work
```
ファイルの生成ディレクトリを指定

-P

```
-P
```
pre79 を起動し構造化記述命令を処理

-Q

```
-Q
```
ソース記述のアドレッシングモードの整合性をチェック

-S

```
-S
```
ローカルシンボル情報をオブジェクトファイルに出力

-SM

```
-SM
```
システムラベルを含むローカルシンボル情報をオブジェクトファイルに出力

-T

```
-T
```
エラータグファイルを生成

-V

```
-V
```
as79 のバージョン番号を表示

-W

```
-W
```
構造化記述命令の展開に対してダブルワード命令の出力を抑止

-X

```
-Xvi
```
エラータグファイルを生成し指定したプログラムを起動

■ ln79 コマンドオプション

-.

```
-.
```
画面へのメッセージ出力を停止

-C

```
-C
```
特定の分岐命令がバンク境界にある場合ワーニングを出力

-E

```
-E 0f000
```
エントリーアドレスを指定

-G

```
-G
```
ソース行情報をアブソリュートモジュールファイルに出力

-L

```
-L lib1
```
ライブラリファイル名を指定

-LD

```
-LD ¥work¥tmp
```
ライブラリファイルを検索するディレクトリを指定

-M

```
-M
```
マップファイルを生成

-MS

```
-MS
```
シンボル情報を含むマップファイルを生成

-MSL

```
-MSL
```
16 文字以上のシンボル名をそのままマップファイルに出力

-NOSTOP

```
-NOSTOP
```
発生したリンクエラー全てを表示

-O

```
-O absfile
```
アブソリュートモジュールファイル名を指定

-ORDER

```
-ORDER main=0f0000,sub1
```
セクション配置を指定

-T

-T
エラータグファイルを生成

-V

-V
ln79 のバージョン番号を表示

@

@cmdfile
指定したファイルの内容をコマンドパラメータとしてln79を起動

■ lmc79コマンドオプション

-.

-.
画面へのメッセージ出力を停止

-A

-A startadd:endadd
-A startadd
出力データのアドレス範囲を指定する

-E

-E 0f000
実行開始アドレスを設定

-F

-F FF
空き領域データの指定

-H

-H
拡張インテル HEX フォーマットファイルを生成

-L

-L
データレコード長を 32 バイトでファイルを生成

-O

-O outfile
出力ファイル名を指定

-V

-V
lmc79 のバージョン番号を表示

■ lb79コマンドオプション

-.

-.
画面へのメッセージ出力を停止

-A

-A
ライブラリファイルにリロケータブルファイル追加

-C

-C
ライブラリファイルを新しく作成

-D

-D
ライブラリファイル内のリロケータブルファイルを削除

-L

-L
ライブラリリストファイルを生成

-R

-R
ライブラリファイルのリロケータブルファイルを更新

-U

-U
ライブラリファイルのリロケータブルファイルより新しいファイルのみ更新

-V

-V
lb79 のバージョン番号を表示

-X

-X
ライブラリファイル内のリロケータブルモジュールファイルを抽出

@

@cmdfile
指定したファイルの内容をコマンドパラメータとしてlb79を起動

■ xrf79 コマンドオプション

-.

-.
画面へのメッセージ出力を停止

-N

-N
システムラベル情報をクロスリファレンスファイルに出力

-O

-OC:¥work
ファイルの生成ディレクトリを指定

-V

-V
xrf79 のバージョン番号を表示

@

@cmdfile
指定したファイルの内容をコマンドパラメータとしてxrf79を起動

■ abs79コマンドオプション

-.

-.
画面へのメッセージ出力を停止

-D

-DC:¥work
アセンブラリストファイルの検索ディレクトリを指定

-O

-OC:¥tmp
アブソリュートリストファイルの出力ディレクトリを指定

-V

-V
abs79 のバージョン番号を表示

■ sc79コマンドオプション

-E

-E
変換結果を入力ファイルに出力

-T

-T
タグファイルを生成

-V

-V
sc79 のバージョン番号を表示

付録 B NC79 拡張機能一覧

near･far修飾子

```
int far    i;          ...1
int far    func();     ...2
```

1. データをアクセスするアドレッシングモードを指定
 near... 同一バンク（64K バイト以内）のアクセス
 far... バンク外（64K バイトを越える領域の）のアクセス
2. 関数の呼び出しに使用する命令を指定
 near...JSR 命令で呼び出し
 far...JSRL 命令で呼び出し

asm関数

```
asm(" LDA  A,DP:1");       ...1
asm(0,0);  /* CLP m,x */   ...2
asm(" LDA  A,DP:$$”,i);    ...3
asm();                     ...4
asm(" LDA  A,$@",i);       ...5
```

1. C 言語プログラム中にアセンブリ言語を直接記述（関数外でも記述可能）
2. プロセッサステータスレジスタ中の m、x フラグの切り替えを指示
3. 変数名を指定可能
4. 最適化を部分的に抑止するダミーの asm 関数が記述可能
5.auto 変数名および外部変数名を指定可能

関数のデフォルト引数宣言

```
extern     int   func(int=1, char=0);
```

関数の引数にデフォルト値を定義
※関数を宣言するよりも前にデフォルト値として使用する変数の宣言を行ってください。
※デフォルト値は引数の後ろから順に埋めてください。

inline 記憶クラスのサポート

```
inline     func(int);
```

inline記憶クラス指定子により関数をインライン展開することが可能
※インライン関数を使用する前に必ずインライン関数の実体定義を行ってください。

日本語文字のサポート

```
L" 文字列 "; ...1
L' 文 ';     ...2
/* 漢字 */ ...3
```

1. 文字列中に日本語文字を使用可能
2. 日本語文字の文字定数を使用可能
3. コメント中に日本語文字を記述可能
※シフト JIS コード及び EUC コードをサポートしています。

コメント記述の拡張

```
// comment
```

C++ 言語ライクなコメント "// " を記述可能

#pragma SECTION

```
#pragma SECTION bss   nonval_data
```

NC79 が生成するセクション名を変更

#pragma ROM

```
#pragma ROM     i
```

指定した変数を rom セクションに配置

#pragma STRUCT

```
#pragma STRUCT  TAG1 unpack     ...1
#pragma STRUCT  TAG1 arrange    ...2
```

1. 指定したタグを持つ構造体のパックを禁止
2. 指定したタグを持つ構造体のメンバを並べ替え、偶数サイズのメンバを先に配置

#pragma INTERRUPT [#pragma INTF]

```
#pragma INTERRUPT    int_func
```

C 言語で記述した割り込み処理関数を宣言

#pragma ADDRESS[#pragma EQU]

```
#pragma ADDRESS port0 2H
```

変数の絶対アドレスを指定
※ near 指定された変数はバンク内アドレスを示します。

#pragma PARAMETER

```
#pragma PARAMETER    asm_func(X, Y)
```
アセンブラ関数を呼び出す際に、その引数をレジスタを介して渡すことを宣言
※本宣言を行う前に、必ず関数のプロトタイプ宣言を行ってください。

#pragma INTHANDLER[#pragma HANDLER]

```
#pragma INTHANDLER   int_func
#pragma HANDLER int_func
```
MR79 の割り込みハンドラ関数名を宣言

#pragma ALMHANDLER

```
#pragma ALHANDLER    alm_func
```
MR79 のアラームハンドラ関数名を宣言

#pragma CYCHANDLER

```
#pragma CYCHANDLER   cyc_func
```
MR79 の周期起動ハンドラ関数名を宣言

#pragma TASK

```
#pragma TASK    task1
```
MR79 のタスクの開始関数名を宣言

#pragma LOADDT

```
#pragma LOADDT  func
```
コンパイル時のデータバンクレジスタ（DT）の値を関数の先頭でロードする関数名を指定

#pragma DP[n]DATA

```
#pragma DP1DATA GLDATA1   ...1
#pragma DP2DATA GLDATA2   ...2
#pragma DP3DATA GLDATA3   ...3
```
1. 外部変数を DP1 レジスタ領域にわりあてます。
2. 外部変数を DP2 レジスタ領域にわりあてます。
3. 外部変数を DP3 レジスタ領域にわりあてます。
※コマンドオプション -fDP_offset_8(-fDPO8)指定時には本機能は無効になります。

#pragma M1FUNCTION

```
#pragma M1FUNCTION
```
M フラグが 1 の状態のまま呼び出す関数を指定

#pragma ASM - #pragma ENDASM

```
#pragma ASM
     LDA.W     A,#0000H
     TAX
#pragma ENDASM
```
アセンブリ言語で記述を行う領域を指定

#pragma PAGE

```
#pragma PAGE
```
アセンブラリスティングファイルの改ページの指定を行います。

#pragma MX1FUNCTION

```
#pragma MX1FUNCTION   func
```
MX フラグが 1 の状態のまま呼び出す関数を指定

付録 C NC79 標準関数一覧

■文字列操作関数

これらの関数を使用する場合は次の記述が必要です。
`#include <string.h>`

strcpy

```
char_far * _far strcpy(s1,s2);
```
文字列の複写

strncpy

```
char_far * _far strncpy(s1,s2,n);
```
文字列の複写(n 文字の複写)

strcat

```
char_far * _far strcat(s1,s2);
```
文字列の連結

strncat

```
char_far * _far strncar(s1,s2,n);
```
文字列の連結(n 文字の連結)

strcmp

```
int_far strcmp(s1,a2);
```
文字列の比較

strcoll

```
int_far strcoll(s1,s2);
```
文字列の比較(ロケール情報を使用)

stricmp

```
int_far stricmp(s1,s2);
```
文字列の比較(すべての英字は英大文字として扱う)

strncmp

```
int_far strncmp(s1,s2,n);
```
文字列の比較(n 文字の比較)

strnicmp

```
int_far strnicmp(s1,s2,n);
```
文字列(n 文字)の比較(英字は英大文字として扱う)

strchr

```
char_far * _far strchr(s,c);
```
文字列の先頭より指定文字を検索

strcspn

```
size_t_far strcspn(s1,s2);
```
文字列より指定以外の文字列の長さを計算

strpbrk

```
char_far * _far strpbrk(s1,s2);
```
文字列より指定文字の検索

strrchr

```
char_far * _far strrchr(s,c);
```
文字列の末尾より指定文字を検索

strspn

```
size_t_far strspn(s1,s2);
```
文字列より指定文字列の長さを計算

strstr

```
char_far * _far strstr(s1,s2);
```
文字列より指定文字列の検索

strtok

```
char_far * _far strtok(s1,s2);
```
文字列より文字列を切り出す

strlen

```
size_t_far strlen(s);
```
文字列中の文字数を計算

strerror

```
char *_far strerror(errcode);
```
エラー番号を文字列に変換

strxfrm

```
size_t_far strxfrm(s1,s2,n);
```
文字列を変換(ロケール情報を使用)

■文字判定関数

これらの関数を使用するには次の記述が必要です。
```
#include   <ctype.h>
```

isalnum

```
int isalnum(c);
```
英数字の判定

isalpha

```
int isalph(c);
```
英字の判定

iscntrl

```
int iscntrl(c);
```
コントロール文字の判定

isdigit

```
int isdigit(c);
```
数字の判定

isgraph

```
int isgraph(c);
```
英数字、ブランク以外の文字判定

islower

```
int islower(c);
```
英小文字の判定

isprint

```
int isprint(c);
```
ブランク文字を含む印字可能文字の判定

ispunct

```
int ispunct(c);
```
区切り文字の判定

isspace

```
int isspace(c);
```
ブランク、タブ、改行の判定

isupper

```
int isupper(c);
```
英大文字の判定

isxdigit

```
int isxdigit(c);
```
16 進数字の判定

tolower

```
int tolower(c);
```
大文字から小文字への変換

toupper

```
int toupper(c);
```
小文字から大文字への変換

■入出力関数

これらの関数を使用する場合は次の記述が必要です。
```
#include <stdio.h>
```

init

```
void_far init(void);
```
7700 ファミリの入出力の初期化

clearerror

```
void_far clearerr(stream);
```
エラー状態指示子を初期化(クリア)

fgetc

```
int_far fgetc(stream);
```
一文字入力

getc

```
int_far getc(stream);
```
一文字入力

getchar

```
int_far getchar(void);
```
stdin からの一文字入力

fgets

```
char *_far fgets(buffer, n,stream);
```
一行入力

gets

```
char *_far gets(buffer);
```
stdin からの一行入力

fread

```
size_t_far fread(buffer,size,count,stream);
```
指定データ数入力

scanf

```
int_far scanf(format,argument...);
```
stdin からの書式付き入力

fscanf

```
int_far fscanf(stream,format,argument...);
```
書式付き入力

sscanf

```
int_far sscanf(string,format,argument...);
```
文字からの書式付きデータ入力

fputc

```
int_far fputc(c,stream);
```
一文字出力

putc

```
int_far putc(c,stream);
```
一文字出力

putchar

```
int_far putchar(c);
```
stdout への一文字出力

fputs

```
int_far fputs(str,stream);
```
一行出力

puts

```
int_far puts(str);
```
stdout への一行出力

fwrite

```
size_t_far fwrite(buffer,size,count,stream);
```
指定データ数出力

perror

```
void_far perror(s);
```
stdout へのエラーメッセージ出力

printf

```
int_far printf(format,argument);
```
stdout への書式付き出力

fflush

```
int_far fflush(stream);
```
出力バッファのストリームをフラッシュ

fprintf

```
int_far fprintf(stream,format,argument);
```
書式付き出力

sprintf

```
inr_far sprintf(pointer,format,argument);
```
書式付き文字列設定

vfprintf

```
int_far vfprintf(stream,format,ap);
```
ストリームへの書式付き出力

vprintf

```
int_far vprintf(format,ap);
```
stdout への書式付き出力

vsprintf

```
int_far vfprintf(s,format,ap);
```
バッファへの書式付き出力

ungetc

```
int_far ungetc(c,stream);
```
一文字入力の返還

ferror

```
int_far ferror(stream);
```
入出力エラーの判定

feof

```
int_far feof(strem);
```
EOF(End Of File)の判定

■メモリ管理関数

これらの関数を使用する場合は次の記述が必要です。
```
#include <stdlib.h>
```

calloc

```
void_far * _far calloc(n,size);
```
メモリの確保と 0(ゼロ)による初期化

free

```
void_far free(cp);
```
メモリの解放

malloc

```
void_far * _far malloc(nbytes);
```
メモリの確保

realloc

```
void_far * _far realloc(cp,nbytes);
```
確保済み領域の大きさを変更

■メモリ操作関数

これらの関数を使用する場合は次の記述が必要です。
```
#include <string.h>
```

bzero

```
void_far bzero(top.size);
```
メモリ領域の初期化(ゼロクリア)

bcopy

```
void_far bcopy(src,dtop,size);
```
メモリ領域の複写

memcpy

```
void_far * _far memcpy(s1,s2,n);
```
メモリ領域の複写(n 文字の複写)

memset

```
char_far* _far memset(s,c,n);
```
メモリ領域の設定

memcmp

```
int_far memcmp(s1,s2,n);
```
メモリ量域の比較(n バイトの比較)

memicmp

```
int_far memicmp(st,s2,n);
```
メモリ領域の比較(英文字は大文字として扱う)

memmove

```
void_far *_far memmove(s1,s2,n);
```
文字列の領域を移動

memchr

```
void_far *_far memchr(s,c,n);
```
メモリ領域より文字を検索

■実行制御関数

これらの関数を使用する場合は次の記述が必要です。
```
#include <stdlib.h>
#include <setjmp.h>
```

abort

```
void_far abort(void);
```
プログラムの実行を終了

longjmp

```
void_far longjmp(env,val);
```
大域ジャンプ

setjmp

```
int_far setjmp(env);
```
大域ジャンプのためのスタック環境の設定

■数学関数

これらの関数を使用する場合は次の記述が必要です。
```
#include <math.h>
```

acos

```
double_far acos(x);
```
逆コサインを計算

asin

```
double_far asin(x);
```

逆サインを計算

atan

```
double_far atan(x);
```
逆タンジェントを計算

atan2

```
doubel_far atan2(x,y);
```
逆タンジェントを計算

ceil

```
double_far ceil(x);
```
整数繰り上げ値を計算

cos

```
double_far cos(x);
```
コサインを計算

cosh

```
double_far cosh(x);
```
双曲線コサインを計算

exp

```
double_far exp(x);
```
指数関数を計算

fabs

```
double_far fabs(x);
```
倍精度浮動小数の絶対値を計算

floor

```
double_far floor(x);
```
整数繰り下げ値を計算

fmod

```
double_far fmod(x,y);
```
剰余計算

frexp

```
double_far frexp(x,exp);
```
浮動小数を仮数部と指数部に分割

labs

```
long_far labs(n);
```
long 型整数の絶対値を計算

ldexp

```
double_far ldexp(x,exp);
```
浮動小数の巾を計算

log

```
double_far log(x);
```
自然対数を計算

log10

```
double_far log10(x);
```
常用対数を計算

modf

```
double_far modf(val,pd);
```
実数を仮数部と指数部に分割

pow

```
double_far pow(x,y);
```
巾乗計算

sin

```
double_far sin(x);
```
サインを計算

sinh

```
double_far sinh(x);
```
双曲線サインを計算

sqrt

```
double_far sqrt(x);
```
数値の平方根を計算

tan

```
double_far tan(x);
```
タンジェントを計算

tanh

```
double_far tanh(x);
```
双曲線タンジェントを計算

■整数算術関数

これらの関数を使用する場合は次の記述が必要です。
`#include <stdlib.h>`

abs

`int_far abs(n);`
整数の絶対値を計算

bsearch

`void_far bsearch(key,base,nelem,cmp);`
配列内のバイナリサーチを行う

div

`dif_t_far div(number,denom);`
int 型整数の除算と剰余

labs

`long_far labs(n);`
long 型整数の絶対値を計算

ldiv

`ldiv_t_far ldiv(number,denom);`
long 型整数の除算と剰余

qsort

`void_far qsort(base,nelen,size,cmp(e1,e2));`
配列をソート

rand

`int_far rand(void);`
疑似乱数を発生

srand

`void_far srand(seed);`
疑似乱数にシードを与える

■文字列数値変換関数

これらの関数を使用する場合は次の記述が必要です。
`#include <stdlib.h>`

atof

`double_far atof(s);`
文字列を double 型に変換

atoi

`int_far atoi(s);`
文字列を int 型に変換

atol

`long_far atol(s);`
文字列を long 型に変換

strtod

double_far strtod(s,endprt);
文字列を double 型に変換

strtol

`long_far strtol(s,endptr,base);`
文字列を long 型に変換

strtoul

`unsignedlong_far strtoul(s,endptr,base);`
文字列を unsigned long 型に変換

■多バイト文字・多バイト文字列操作関数

これらの関数を使用する場合は次の記述が必要です。
`#include <stdlib.h>`

mblen

`int_far mblen(s,n);`
マルチバイト文字列の長さを計算

mbstowcs

`size_t_far mbstowcs(wcs,s,n);`
マルチバイト文字列をワイド文字列に変換

mbtowc

`int_far mbtowc(wcs,s,n);`
マルチバイト文字をワイド文字に変換

wcstombs

`size_t_far wcstombs(s,wcs,n);`
ワイド文字列をマルチバイト文字列に変換

wctomb

`int_far wctomb(s,wchar);`
ワイド文字をマルチバイト文字に変換

■地域化関数

これらの関数を使用する場合は次の記述が必要です。
`#include <local.h>`

localeconv

```
struct lconv_far *localeconv(void);
```
構造体 lconv を初期化

setlocale

```
chat_far *setlocale(category,locale);
```
プログラムのロケール情報の設定と検索

memo

付録 D AS79 アセンブラ指示命令一覧

..FILE

```
..FILE
```
アセンブラが処理しているファイル名を保持

.ADDR

```
[str:].ADDR     exp[,exp,...]
```
3 バイト長のデータ領域を ROM に確保し値を格納

.ALIGN

```
.ALIGN    [2|4]
```
セクションの配置アドレスをワードまたはダブルワードアライメントに補正

.ASSERT

```
.ASSERT   "str"
```
オペランドの文字列を標準エラー出力に表示

.BLKA

```
[str:].BLKA     exp
```
3 バイト(24 ビット)単位で RAM に領域を確保

.BLKB

```
[str:].BLKB     exp
```
1 バイト(8 ビット)単位で RAM に領域を確保

.BLKD

```
[str:].BLKB     exp
```
4 バイト単位で RAM に領域を確保

.BLKDF

```
[str:].BLKD     exp
```
8バイト単位でRAMに領域確を確保（倍精度浮動小数点数を使用する場合）

.BLKF

```
[str:].BLKD     exp
```
4バイト単位でRAMに領域確を確保（単精度浮動小数点数を使用する場合）

.BLKW

```
[str:].BLKW     exp
```
2 バイト(16 ビット長)単位で RAM に領域を確保

.BTEQU

```
bitsym    .BTEQU 0,0
```
ビットシンボルを定義

.BYTE

```
str: .BYTE exp[,exp,...]
```
1 バイト長のデータ領域を ROM に確保し値を格納

.CALL

```
.CALL  str,str
```
インスペクタ情報の関数呼び出し先情報を生成

.DATA [8|16]

```
.DATA 8
SEM
```
データ長を宣言（本命令以降のデータを宣言したデータ長で処理する）
※本命令でデータ長を宣言する場合は、必ずCLMまたはSEMで CPU 内のフラグも操作してください。

.DEFINE

```
FLAG1 .DEFINE   "#01,DATA1"
```
シンボルに文字列を定義

.DOUBLE

```
label: .DOUBLE 5e2
```
浮動小数点数データを ROM に格納

.DP

```
.DP  1000H
```
ダイレクトページレジスタ値（0000H～0FFFFH）を宣言（8ビットオフセットモード）

.DP[0|1|2|3]

```
.DP0 20H
```
ダイレクトページレジスタ値（0000H～0FFFFH）を宣言（6ビットオフセットモード）

.DPSYM

```
.DPSYM    symbol
```
ダイレクトアドレッシングモード（8ビットオフセットモード）でコード生成するシンボルを宣言

.DP[0|1|2|3]SYM

```
.DP0SYM   symbol
```
ダイレクトアドレッシングモード（6ビットオフセットモード）でコード生成するシンボルを宣言

.DT

```
.DT  01H
```
データバンクレジスタ値宣言

.DTSYM

```
.DTSYM    symbol
```
アブソリュートアドレッシングモードでコード生成するシンボルを宣言

.DWORD

```
DWMEM:    .DWORD    0E1000H
```
4バイト長のデータ領域をROMに確保し値を格納

.EINSF

```
.EINSF
```
インスペクタ情報の関数終了情報を生成

.END

```
.END
```
アセンブルソースの終了を宣言

.EQU

```
str  .EQU exp
```
数値シンボルに値を定義

.FLOAT

```
label:    .FLOAT    5e2
```
単精度浮動小数点データをROMに格納

.FORM

```
.FORM 20,80
```
リストファイルの行数と桁数を指定

.GLB

```
.GLB symbol
```
グローバルシンボルを宣言

.IF (- .ELIF - .ELSE -) .ENDIF

```
.IF  CDATA < 0
     LDA  WORK1
.ELIFCDATA >  0
     LDA  WORK2
.ELSE
     LDA  WORK3
.ENDIF
```
条件付きアセンブル（.ELSEとそれに続く命令は省略可能）

.INCLUDE

```
.INCLUDE  filename
```
指定したファイルの記述内容を読み込む

.INDEX

```
.INDEX    8
SEP  X
```
インデックスレジスタ長を宣言

.INSF

```
.INSF  str,str,exp
```
インスペクタ情報の関数開始情報を生成

.LENGTH

```
.LENGTH   8,8
```
データ長およびインデックスレジスタ長を宣言

.LGSYM

```
.LGSYM    symbol
```
アブソリュートロングアドレッシングでコード生成するシンボルを宣言

.LIST

```
.LIST ON
```
リストファイルへの行の出力を開始
```
.LIST OFF
```
リストファイルへの行の出力を停止

.ORG

```
.ORG 0C000H
```
この行以降のアドレスを宣言

.PAGE

```
.PAGE 'Start main routine'
```
リストファイルを改ページし、指定した文字列をリストファイルに出力

.SECTION

```
.SECTION   DATA
```
セクション名を宣言

.STK

```
.STK  exp
```
インスペクタ情報のスタックサイズ情報を生成

.VER

```
.VER 'str'
```
指定した文字列をマップファイルに出力します。

.WORD

```
str: .WORD 0E00H
```
2 バイト長のデータ領域を ROM に確保し値を格納

?

```
?:
JMP  ?-
```
テンポラリラベルを宣言または参照します。

@

```
.ASSERT    "sample" > ..FILE@.dat
```
文字列を連結します。

●マクロ命令

..MACPARA

```
..MACPARA
```
マクロ命令の実引数の個数を保持

..MACREP

```
..MACREP
```
繰り返しマクロが展開された回数を保持

..MACREPZ

```
..MACREPZ
```
繰り返しマクロが展開された回数から 1 引いた値を保持

.INSTR

```
.INSTR{'source','dest',top}
```
指定文字列の開始位置を保持

.LEN

```
.LEN{'string'}
```
指定した文字列の文字数を保持

.LOCAL

```
.LOCAL     LOOP1,LOOP2
```
マクロローカルラベルを宣言(マクロ定義内でのみ使用可能)

.MACRO - (.EXITM -) .ENDM

```
DATA1:     .MACRO     VAL
      .IF   LEVEL
      .BYTE VAL
      .EXITM
      .ENDIF
      .WORD VAL
      .ENDM
```
マクロ定義命令 (.EXITM はマクロ展開を中止し .ENDM に分岐する)

.MREPEAT - .ENDR

```
.REPEAT    5
      NOP
.ENDM
```
.MREPEAT と .ENDR に囲まれた部分を指定回数だけ展開

.SUBSTR

```
.SUBSTR{'data',0,1}
```
指定した文字列から指定した文字を取り出す

■構造化記述命令

=（代入文）

```
[work] = 0
C = 0
```
右辺を左辺へ代入

IF -(ELIF - ELSE -) ENDIF

```
IF [sym1] == 10
    :
ELIF [sym2] != 10
    :
ELSE
    IF [sym3] == 10
        :
    ENDIF
ENDIF
```
条件を判断し分岐処理を制御

FOR - NEXT

```
FOR A <.S 10
     :
NEXT
```
条件を判断し繰り返し処理を制御

FOR - STEP

```
FOR X = 0 TO 10 STEP 1
     :
NEXT
```
条件を判断し繰り返し処理を制御

DO - WHILE

```
DO
 :
WHILE C==1
```
条件を判断し繰り返し処理を制御

SWITCH - CASE - (DEFAULT -)ENDS

```
SWITCH     [mem]
     CASE  mode1
           JSR   sub1
     CASE  mode2
           JSR   sub2
     DEFAULT
           JSR   sub3
ENDS
```
条件式の値によっていずれかの文(CASE)に分岐

BREAK

```
BREAK
```
FOR、DO、SWITCH の実行を中止し各ブロックの次に分岐

CONTINUE

```
CONTINUE
```
FOR、DO ブロック内から繰り返しの条件を判断するところへ分岐

FOREVER

```
FOREVER
```
繰り返しを無限ループ処理

GOTO

```
GOTO  LAB1
```
任意のアドレスに無条件で分岐

CALL

```
CALL  subroutine
```
指定したラベルにサブルーチン分岐

RETURN

```
RETURN
```
サブルーチンから復帰

■構造化記述内でのみ有効な指示命令

.FAR

```
.FAR symbol
```
指定したシンボルがバンク外にあることを宣言

.GLBB

```
.GLBB      symbol
```
指定したシンボルがグローバルであることを宣言。シンボルはバイトサイズで処理

.GLBD

```
.GLBBD     symbol
```
指定したシンボルがグローバルであることを宣言。シンボルはダブルワードサイズで処理

.GLBW

```
.GLBBW     symbol
```
指定したシンボルがグローバルであることを宣言。シンボルはワードサイズで処理

.NEAR

```
.NEAR      symbol
```
指定したシンボルが同一バンク内にあることを宣言

memo

技術サポート連絡書

年　月　日（合計　　枚）

三菱電機セミコンダクタシステム株式会社
マイコンソフトツール部
開発ツールサポート窓口行

［電子メール］ support@tool.mesc.co.jp
［大阪地区］ FAX： 06-6398-6191
［東京地区］ FAX： 03-5783-7339
［中部地区］ FAX： 052-221-7318
［九州地区］ FAX： 092-452-1427

※インストーラが生成する以下のテキストファイルもサポート連絡書としてご利用できます。
Windows 98/95/Windows NT 4.0版の場合：\SUPPORT\製品名\SUPPORT.TXT
EWS版の場合：/support/製品名/toolinfo.txt

ご連絡先	製品情報
会社名：	ソフトウェア：
部署名：	バージョン番号：V.
担当者名：	ライセンスID：
電話番号：	– – – –
FAX番号：	ホストマシン：
電子メール：	OS： V.
通信欄：	

MEMO

NC79WAガイドブック

第1版：2000年7月1日発行
資料番号：MSD-NC79WA-UG-000701

三菱電機株式会社
三菱電機セミコンダクタシステム株式会社

7900 シリーズ用　C コンパイラ
NC79WA　ガイドブック

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