EPSON
EXCEED YOUR VISION

CMOS 4-BIT SINGLE CHIP MICROCOMPUTER

S5U1C63000A Manual

S1C63 Familyアセンブラパッケージ Ver.11

セイコーエプソン株式会社

Rev.1.1

評価ボード・キット、開発ツールご使用上の注意事項

本資料の内容については、予告無く変更することがあります。

1. 本評価ボード・キット、開発ツールは、お客様での技術的評価、動作の確認および開発のみに用いられることを想定し設計されています。それらの技術評価・開発等の目的以外には使用しないで下さい。本品は、完成品に対する設計品質に適合していません。
2. 本評価ボード・キット、開発ツールは、電子エンジニア向けであり、消費者向け製品ではありません。お客様において、適切な使用と安全に配慮願います。弊社は、本品を用いることで発生する損害や火災に対し、いかなる責も負いかねます。通常の使用においても、異常がある場合は使用を中止して下さい。
3. 本評価ボード・キット、開発ツールに用いられる部品は、予告無く変更されることがあります。

本資料のご使用につきましては、次の点にご留意願います。

本資料の内容については、予告無く変更することがあります。

1. 本資料の一部、または全部を弊社に無断で転載、または、複製など他の目的に使用することは堅くお断りいたします。
2. 本資料に掲載される応用回路、プログラム、使用方法等はあくまでも参考情報であり、これらに起因する第三者の知的財産権およびその他の権利侵害あるいは損害の発生に対し、弊社はいかなる保証を行うものではありません。また、本資料によって第三者または弊社の知的財産権およびその他の権利の実施権の許諾を行うものではありません。
3. 特性値の数値の大小は、数直線上の大小関係で表しています。
4. 製品および弊社が提供する技術を輸出等するにあたっては「外国為替および外国貿易法」を遵守し、当該法令の定める手続きが必要です。大量破壊兵器の開発等およびその他の軍事用途に使用する目的をもって製品および弊社が提供する技術を費消、再販売または輸出等しないでください。
5. 本資料に掲載されている製品は、生命維持装置その他、きわめて高い信頼性が要求される用途を前提としていません。よって、弊社は本(当該)製品をこれらの用途に用いた場合のいかなる責任についても負いかねます。
6. 本資料に掲載されている会社名、商品名は、各社の商標または登録商標です。

製品型番体系

●デバイス

S1　C　63158　F　0A01　00

■ 梱包仕様
[00 : テープ&リール以外
0A : TCP BL　2方向
0B : テープ&リール BACK
0C : TCP BR　2方向
0D : TCP BT　2方向
0E : TCP BD　2方向
0F : テープ&リール FRONT
0G : TCP BT　4方向
0H : TCP BD　4方向
0J : TCP SL　2方向
0K : TCP SR　2方向
0L : テープ&リール LEFT
0M : TCP ST　2方向
0N : TCP SD　2方向
0P : TCP ST　4方向
0Q : TCP SD　4方向
0R : テープ&リール RIGHT
99 : 梱包仕様未定]

■ 仕様

■ 形状
[D: ベアチップ、F: QFP、B: BGA]

■ 機種番号

■ 機種名称
[C: マイコン、デジタル製品]

■ 製品分類
[S1: 半導体]

●開発ツール

S5U1　C　63000　A1　1　00

■ 梱包仕様
[00: 標準梱包]

■ バージョン
[1: Version 1]

■ ツール種類
[Hx : ICE
Ex : EVAボード
Px : ペリフェラルボード
Wx: FLASHマイコン用ROMライタ
Xx : ROMライタ周辺ボード
Cx : Cコンパイラパッケージ
Ax : アセンブラパッケージ
Dx : 機種別ユーティリティツール
Qx : ソフトシミュレータ]

■ 対応機種番号
[63000: S1C63ファミリ共通]

■ ツール分類
[C: マイコン用]

■ 製品分類
[S5U1: 半導体用開発ツール]

はじめに

本書はS1C63 Familyの全機種に共通するソフトウェア開発ツール「S1C63 Family Assembler Package」によるアセンブリソースプログラムのアセンブルからデバッグまでの手順および各ツールの使用方法を説明します。

注意事項

本パッケージのツールを使用せずにお客様が作成されたパックデータ、あるいはツールが生成したデータをお客様が編集して作成されたパックデータから製作した製品の不具合については、弊社は如何なる責任も負いかねます。

マニュアルの読み方

本書はプログラム開発に従事されている方を対象に編集されています。したがって、以下の内容が予備知識として必要です。
・アセンブリ言語に関する一般的な知識
・アセンブラを使用したプログラム開発全般の基礎知識
・Windowsの基本的な操作方法

■インストールの前に

1章をお読みください。パッケージの構成品や各ツールの概要が記載されています。

■インストール

1章に記載されていますように、インストーラの指示に従ってツールをインストールしてください。

■プログラム開発全体の流れと、その操作手順を理解するには

2章のプログラム開発手順をお読みください。

■ソースプログラム作成時

4章の必要個所をお読みください。ソースファイル作成上の注意事項や、アセンブリソースファイルの内容等が記載されています。
ソースファイルを作成する際には、以下のマニュアルも併せて参照してください。

S1C63xxxテクニカルマニュアル
　デバイスの仕様、周辺回路の動作と制御方法について

S1C63000コアCPUマニュアル
　コアCPUの機能と動作、命令セットの詳細について

■プログラムのデバッグについて

8章でデバッガの詳細を解説しています。8.1項～8.8項でデバッグ機能の概要を把握していただけます。各デバッグコマンドの詳細については8.9項を参照してください。
なお、デバッグ前には以下のマニュアルを参照し、インサーキットエミュレータICEおよびペリフェラルボードS5U1C63xxxPの取り扱い方法を理解しておいてください。

S5U1C63000H Manual (S1C63 Family In-Circuit Emulator)
　インサーキットエミュレータICEの取り扱い方法について

S5U1C63xxxP Manual (Peripheral Circuit Board for S1C63xxx)
　各機種のハードウェア仕様をICEに提供するペリフェラルボードの機能と取り扱い方法について

■各ツールの詳細について

3～8の各章で、それぞれのツールの詳細を解説しています。必要に応じて参照してください。

■一通りの操作に慣れたら

クイックリファレンスに掲載されているインストラクションセットやコマンドの一覧を利用されると便利でしょう。

マニュアルの表記

このマニュアルは、次の表記規則に従って記述されています。

■サンプル画面

サンプルの画面はすべてWindows英語版での表示例です。システムや搭載されているフォントにより表示が異なる場合があります。

■各部の名称

ウィンドウ名、メニューおよびメニューコマンド名、ボタン名、ダイアログボックス名、キーの名称は、[ ]で囲んで記述されています。たとえば、[Command]ウィンドウ、[File ¦ Exit]メニューコマンド([File]メニュー内の[Exit]コマンド)、[Key Break]ボタン、[q]キーなどのように記述します。

■命令などの名称

CPUの命令やコマンド名など、大文字、小文字の両方が許されるものについては小文字を使用しています。ただし、ユーザーが指定するシンボル例などは除きます。

■数値の表記

数値は次のように記述しています。
10進数: 数値の前後に何も付けない。(例: 123、1000)
16進数: 数値の前に"0x"を付ける。(例: 0x0110、0xffff)
2進数: 数値の前に"0b"を付ける。(例: 0b0001、0b10)
ただし、表示例などには、16進数、2進数に何も付かない場合があります。また、説明上16進数をxxxxh、2進数をxxxxbのように表現している場合もあります。

■マウス操作

クリック: 目的の位置にカーソル(ポインタ)を合わせ、マウスの左ボタンを一度押す操作をすべて「クリック」で表現します。右ボタンのクリック操作は「右クリック」という言葉を使います。
ダブルクリック: 目的の位置にカーソル(ポインタ)を合わせ、マウスの左ボタンを二度連続的に押す操作をすべて「ダブルクリック」で表現します。
ドラッグ: クリックにより選択したファイル(アイコン)などを、左ボタンを押したまま別の場所に移動させる操作を「ドラッグ」で表現します。
選択: メニューコマンドなどをクリックして選択する操作を、「選択」のみで表現します。

■キー操作

特定のキーを押す操作は「キーを入力」、または「キーを押す」と表現しています。
[Ctrl]+[C]キーのように+を使ったキーの組み合わせは、[Ctrl]キーを押しながら[C]キーを押す操作を表します。
キーボードからの入力例については特に[ ]で囲んでいません。また、入力例中の[Enter]キーを押す操作は↵で表しています。
なお、このマニュアルでは、マウスで可能な操作は、すべてマウス操作方法のみを記述しています。キーボードによる操作についてはWindowsのマニュアルやヘルプ等を参考にしてください。

■コマンドや起動時オプション、メッセージの一般形

[ ]で囲まれた項目はユーザー選択項目で、入力しなくても動作することを表しています。
<>で囲まれた内容は、特定の名称が用いられることを表しています。たとえば、<file name>や<ファイル名>は実際のファイル名に置き換えることを示しています。
{ }で囲まれ、¦で区切られた項目はその中から1つを選択することを示します。たとえば、{A ¦ B}はAまたはBのどちらか1つを選択することを示します。

■その他開発ツールの名称

ICE: S5U1C63000H(S1C63 Family専用ICE)を示しています。

目 次

1 概要

1.1 特長

S1C63 Family Assembler packageはS1C63 Familyの全機種に共通のソフトウェア開発ツールで、ソースプログラムのアセンブルからデバッグまでの効率良い作業環境を提供します。
主な特長を以下に示します。

・シンプルなツール構成
アセンブラからデバッガまでを最小限のツール構成で実現しています。

・統合化作業環境
Windows GUIアプリケーション《ワークベンチwb63》により、ソースファイルの編集も含めた、全ツールの統合化作業環境を実現しています。

・モジュラプログラミングが可能
リロケータブルアセンブラによりモジュール別にプログラムが開発できます。汎用モジュールとして開発したプログラムは今後の機種開発にも利用することができます。

・ソース表示が可能なデバッガ
シンプルかつ強力なデバッグ機能を持つデバッガによって､アセンブリソースを表示させてデバッグが可能です。

・S1C63 Familyの全機種に対応
本パッケージのツールは、ICEパラメータファイルや機種情報定義ファイルから機種固有の情報を読み込むことによって、S1C63 Familyの全機種に対応します。ただし、マスクデータ作成ツールについては、別途用意されている機種もあります。

・旧ツールと完全互換
本パッケージのツールは旧ツールの文法にも完全に対応しています。旧アセンブラの文法で作成されたソースファイルも変更することなく処理可能です。

1.2 ツール構成

1.2.1 ソフトウェアツールの概要

本パッケージに含まれる主要なソフトウェアツールの概要を以下に示します。

アセンブラ(as63.exe)

アセンブラas63はアセンブリソースファイルをアセンブルし、ソースファイルのニーモニックをS1C63000のオブジェクト(機械語)コードに変換します。結果はリロケータブルオブジェクトファイルとして出力されます。また、マクロや、条件アセンブル、インクルードなど、アセンブルの前処理のための付加機能も含まれています。

リンカ(lk63.exe)

アセンブラas63によって生成したリロケータブルオブジェクトのメモリロケーションを決定し、実行可能なアブソリュートオブジェクトコードを生成します。また、プログラムの分岐範囲を意識しないプログラミングを可能とするEXT命令の自動挿入/修正機能も提供します。

HEXコンバータ(hx63.exe)

リンカが出力するIEEE-695形式のアブソリュートオブジェクトファイルをモトローラS形式あるいはインテルHEX形式のROMイメージデータに変換します。この変換は、デバッグ用のプログラムROMを作成する場合や、マスクデータチェッカでマスクデータを作成する場合に必要です。

ディスアセンブラ(ds63.exe)

IEEE-695形式またはモトローラS形式のアブソリュートオブジェクトを逆アセンブルし、ソースファイルに復元します。復元したソースファイルは、アセンブラ、リンカ、HEXコンバータでオリジナルのソースと同様に処理可能で、復元前と同じアブソリュートオブジェクトまたはHEXファイルが生成されます。

デバッガ(db63.exe)

ハードウェアツールとして用意されているICEを制御してデバッグを行うソフトウェアです。ブレークやステップ実行など、頻繁に使用するコマンドはツールバーに登録されており、キーボード操作の量を抑えています。また、ソースやレジスタ内容、コマンド実行結果がマルチウィンドウ上に表示できるため、デバッグ作業が効率良く行えます。

ワークベンチ(wb63.exe)

Windows GUIベースのアプリケーションで、統合開発環境を提供します。ソースの作成や編集、ファイルの選択、各ツールの起動時オプションの選択と起動などが、Windowsの基本操作のみで可能となります。

以下のツールはマスクデータを作成するためのWindows GUIツールです。起動には機種情報定義ファイル（s1c63xxx.ini）が必要です。CD-ROMには以下のツールでサポートしている機種の機種情報定義ファイルが含まれています。未対応の機種については、それぞれの機種ごとに"Development Tool"が用意されています。

ファンクションオプションジェネレータ（winfog.exe）

winfogはS1C63xxxのマスクオプションを選択し、ICEのファンクションオプション設定ファイルとICマスクパターンを生成するためのファンクションオプションドキュメントファイルを作成するツールです。ウィンドウに表示された選択項目をチェックボックスで選択するだけでファンクションオプションデータを作成することができます。

セグメントオプションジェネレータ（winsog.exe）

winsogはS1C63xxxのLCDセグメントオプションを設定し、ICEのセグメントオプション設定ファイルとICマスクパターンを生成するためのセグメントオプションドキュメントファイルを作成するツールです。ウィンドウに表示された表示メモリマップとセグメントデコードテーブルをマウスでクリックするだけで、セグメント割り付けデータを作成することができます。
winsogはセグメントオプションを持つ機種以外では使用しません。

マスクデータチェッカ（winmdc.exe）

winmdcは開発が終了したプログラムROM/データROMファイル、オプションドキュメントファイルのデータをチェックし、セイコーエプソンへ提出するためのマスクデータファイルを作成するツールです。

1.3 動作環境

S1C63 Family Assembler packageを使用するには、次の動作環境が必要です。

●パーソナルコンピュータ

IBM PC/ATまたは完全互換機が必要です。Pentium 90MHz以上のCPUと32MB以上のRAMを搭載した機種を推奨します。
別売のインサーキットエミュレータICEを使用するには、シリアルポート（D-sub 9pin）も必要です。

●ディスプレイ

800×600ドット以上のディスプレイが必要です。

●ハードディスク

S1C63 Family Assembler packageをインストールするには、ハードディスクが必要です。

●マウス

ツールの操作にマウスが必要です。

●システムソフトウェアについて

S1C63 Family Assembler packageはWindowsに対応しています（詳細はS5U1C63000AのReadme_J.txtを参照してください）。

●その他

プログラムのデバッグには、本パッケージとは別売のインサーキットエミュレータICEおよびペリフェラルボードS5U1C63xxxPが必要です。
S5U1C63xxxPは各機種ごとに用意されています。

1.4 インストール

ツールのインストールはインストーラ(Setup.exe)によって行います。
"Setup.exe"を起動し、ダイアログボックスに表示される指示に従ってツールをインストールしてください。

1.5 インストール後のディレクトリとファイル構成

インストーラは、指定ディレクトリ(デフォルトは"C:¥EPSON¥S1C63¥")に以下のファイルをコピーします。

```
[EPSON¥S1C63]
    README_J.TXT          ... ReadMeテキストファイル(日本語)
    README_E.TXT          ... ReadMeテキストファイル(英語)

    [¥BIN]                ... S1C63 Family Assembler package Tool
        WB63.EXE          ... ワークベンチ
        AS63.EXE          ... アセンブラ
        LK63.EXE          ... リンカ
        HX63.EXE          ... HEXコンバータ
        DS63.EXE          ... ディスアセンブラ
        DB63.EXE          ... デバッガ
        . . .             ... その他関連ファイル

        [¥DRIVER]         ... ICE USBドライバファイル

    [¥DEV]
        [¥BIN]            ... S1C63 Family Development Tool for Windows
            WINFOG.EXE    ... ファンクションオプションジェネレータ
            WINSOG.EXE    ... セグメントオプションジェネレータ
            WINMDC.EXE    ... マスクデータチェッカ

        [¥63xxx]          ... 各機種用ファイル
            S1C63xxx.INI  ... 機種情報定義ファイル
            PAR63xxx.PAR  ... パラメータファイル
            C63xxx.FSA    ... ファンクションオプションHEXサンプルファイル(ICE設定用)
            C63xxx.SSA    ... セグメントオプションHEXサンプルファイル(ICE設定用)
        :

    [¥WRITER]
        [¥OBPW]
            OBPW63.EXE    … On Board Writerコントロールソフトウェア
            RW6xxxx.INI   … 機種情報設定用ファイル

        [¥DRIVER]
            FTDIBUS.INF   … USB-Serial On Board Writer USBドライバ定義ファイル
            FTDIPORT.INF  … USB-Serial On Board Writerシリアルポートドライバ定義ファイル
        :
```

＊ ROM Writer II、On Board Writer/USB-Serial On Board Writerの詳細については、テクニカルマニュアルを参照してください。

```
[¥ICE]
    [¥FPGA]
        C63xxx.MOT          ... 標準ペリフェラルボード用FPGAデータ
        :
[¥DOC]
    [¥JAPANESE]             ... ドキュメントフォルダ(日本語)
        REL_xxxx_J.TXT      ... ツールのリリースノート
        s5u1c63000a_xxJ.pdf ... 本マニュアルのPDFファイル
    [¥ENGLISH]              ... ドキュメントフォルダ(英語)
        REL_xxxx_E.TXT      ... ツールのリリースノート
        s5u1c63000a_xxE.pdf... 本マニュアルのPDFファイル
```

●PDF形式のオンラインマニュアル

添付のオンラインマニュアルはPDF形式で作成されており、参照するにはAdobe Acrobat ReaderのVer. 5.0以降が必要です。

●今後リリースされる機種のファイル

今後の機種用のファイルはリリース時に提供予定です。インストールにつきましては、それぞれのReadmeファイルを参照してください。

2　ソフトウェア開発手順

この章では基本的なソフトウェア開発手順を説明します。

2.1　ソフトウェア開発フロー

ソフトウェア開発の流れを図2.1.1に示します。

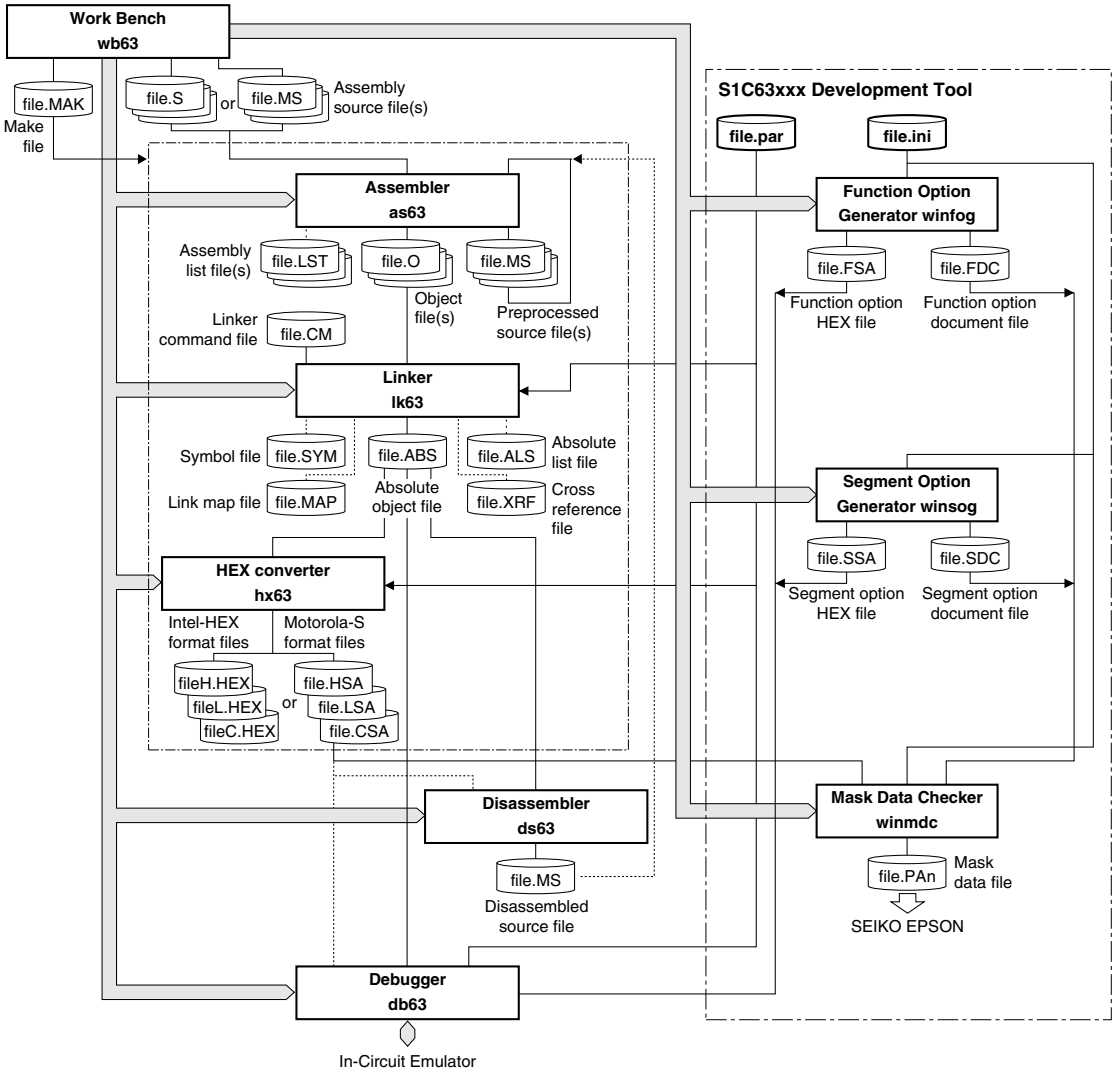


図2.1.1　ソフトウェア開発フロー

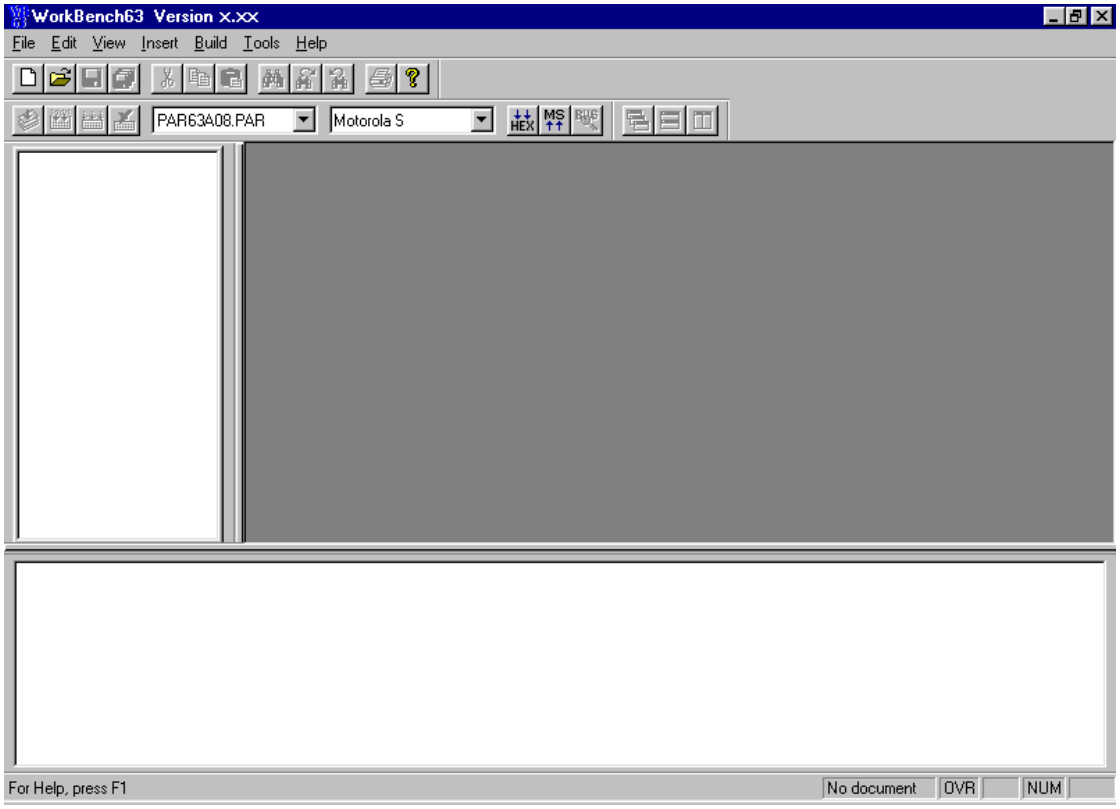
ワークベンチはソースの編集からデバッグまでの統合開発環境を提供します。アセンブラやリンカなどのツールはワークベンチ上から起動することができます。また、各ツールはコマンドプロンプトなどからも個別に起動可能です。
各ツールの詳細についてはそれぞれの章を参照してください。
機種によっては"S1C63xxx Development Tool"が別途用意されている場合があります(fog63xxx、sog63xxx等)。それらの機種依存のツールに関しては、本マニュアルには説明されていません。該当機種のツールマニュアルを参照してください。

2.2　ワークベンチによる開発

ここでは、ワークベンチwb63を使用する基本的な開発の流れを説明します。
操作の詳細については、"3 ワークベンチ"を参照してください。

2.2.1 ワークベンチの起動

[プログラムメニュー]から"WorkBench63"を選択し、ワークベンチを起動させてください。



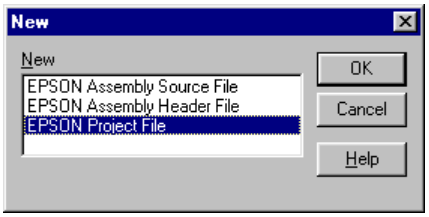
2.2.2 プロジェクトの新規作成

ワークベンチは、1つのアプリケーション開発に必要なファイルと各ツールの設定をプロジェクトとして管理します。そこで、最初にプロジェクトを新規作成します。

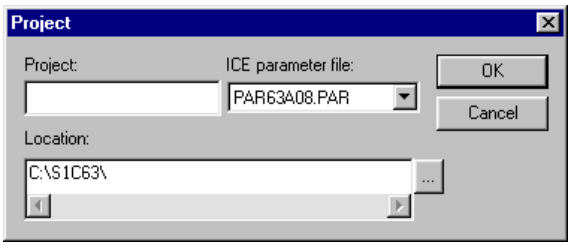
1. [File]メニューから[New]を選択します(または、[New]ボタンをクリックします)。

   [New]ボタン

   [New]ダイアログボックスが表示されます。



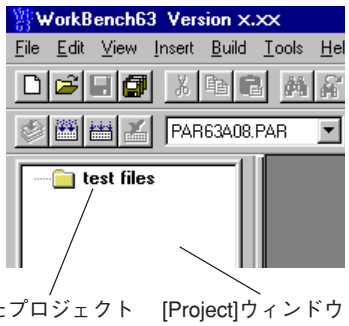
2. [EPSON Project File]を選択し、[OK]をクリックします。
   [Project]ダイアログボックスが表示されます。



3. プロジェクト名を入力し、ディレクトリとICEパラメータファイルを選択して[OK]をクリックします。

   * [ICE parameter file:]ボックスには、"dev63"ディレクトリ内に存在するパラメータファイルがリストアップされます。

ワークベンチは指定のプロジェクト名でフォルダ(ディレクトリ)を作成し、そのフォルダをワークスペースとして使用します。また、プロジェクトファイル(.epj)をその中に作成します。
ここで指定したプロジェクト名は、後で生成されるアブソリュートオブジェクトや他のファイル名にも使用されます。



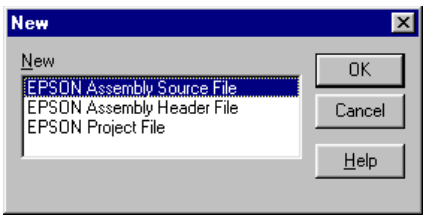
2.2.3 ソースファイルの編集

ワークベンチはエディタ機能も持っています。これにより、他のエディタを使用せずにソースの作成/編集が行えます。ソースファイルの新規作成は次のように行います。

1. [File]メニューから[New]を選択します(または、[New]ボタンをクリックします)。

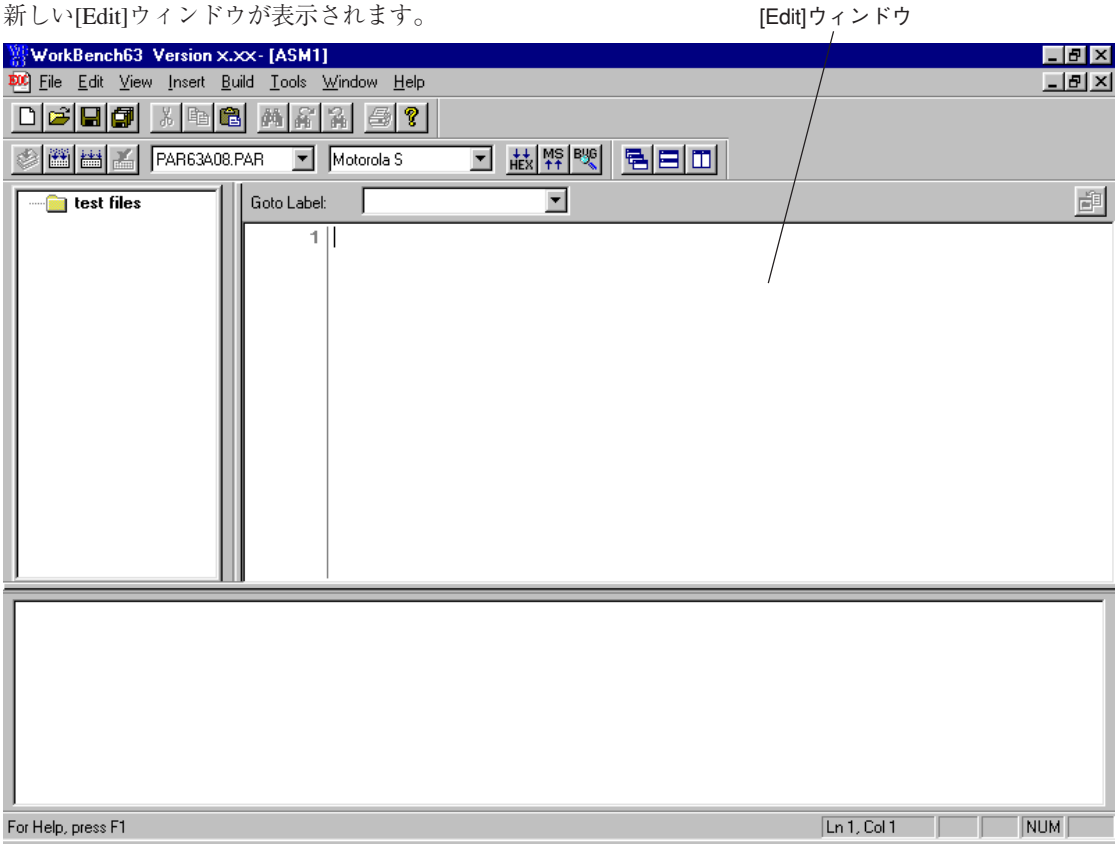
   [New]ボタン

   [New]ダイアログボックスが表示されます。

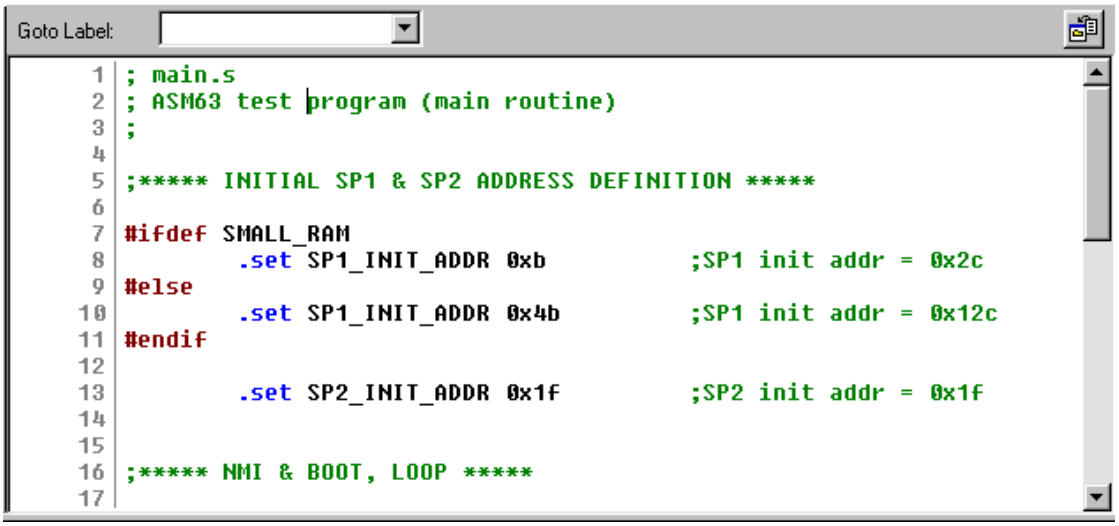


2. [EPSON Assembly Source File]を選択し、[OK]をクリックします。

新しい[Edit]ウィンドウが表示されます。



3. [Edit]ウィンドウにソースコードを入力します。



4. [File]メニューの[Save]（または[Save]ボタン）で作成したソースをファイルに保存します。


[Save]ボタン

5. [Edit]ウィンドウの[Insert into project]ボタンをクリックします。


[Insert into project]ボタン

作成したソースファイルがプロジェクトに追加されます。

すでに作成済みのソースファイルをプロジェクトに追加するには、[Insert]メニューの[Files into project...]を使用してください。また、Windowsのエクスプローラからソースファイルを[Project]ウィンドウにドラッグすることによっても追加できます。
必要なソースファイルを作成し、すべてプロジェクトに追加してください。

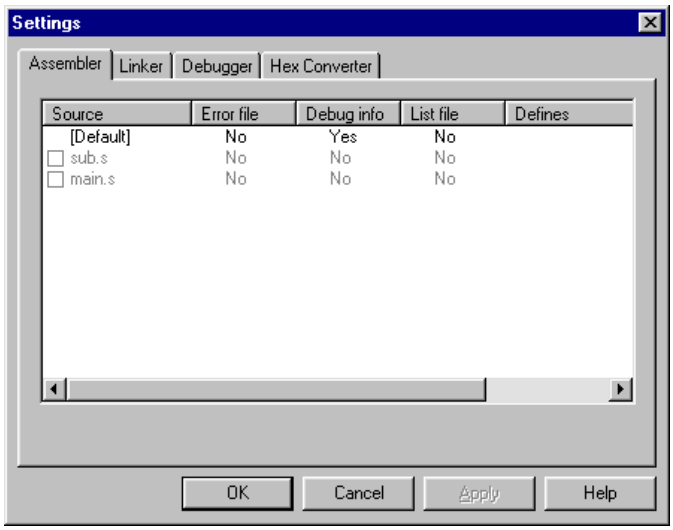
[Project]ウィンドウのリスト例

プロジェクト内のソースファイルは、[Project]ウィンドウ内にリストされます。リストされているソースファイル名をダブルクリックすると、そのファイルが開き、内容が[Edit]ウィンドウに表示されます。

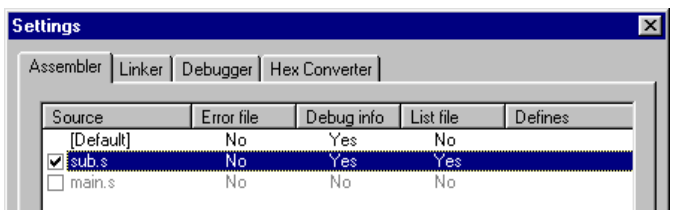
2.2.4 ツールオプションの設定

ワークベンチは各ツールのすべての起動オプションに対応しており、ダイアログボックスで選択/設定できるようになっています。実行形式オブジェクトを作成するmake処理は、この設定にしたがって生成されます。オプション選択に加え、リンカおよびデバッガ用のコマンドファイルもここで編集できます。
ツールオプションの選択は次のように行います。

1. [Build]メニューから[Setting...]を選択します。
   ダイアログボックスが表示されます。



2. 必要なオプションを選択、設定します。
   チェックボックスの項目はクリックして選択、あるいは解除できます。リスト内の項目はダブルクリックして選択を反転、あるいは入力します。



[Settings]ダイアログの詳細については、"3 ワークベンチ"を参照してください。

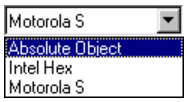
2.2.5 実行形式オブジェクトのビルド

実行形式のオブジェクトは次のように作成します。

1. [Build]メニューから[Build]を選択します(または[Build]ボタンをクリックします)。

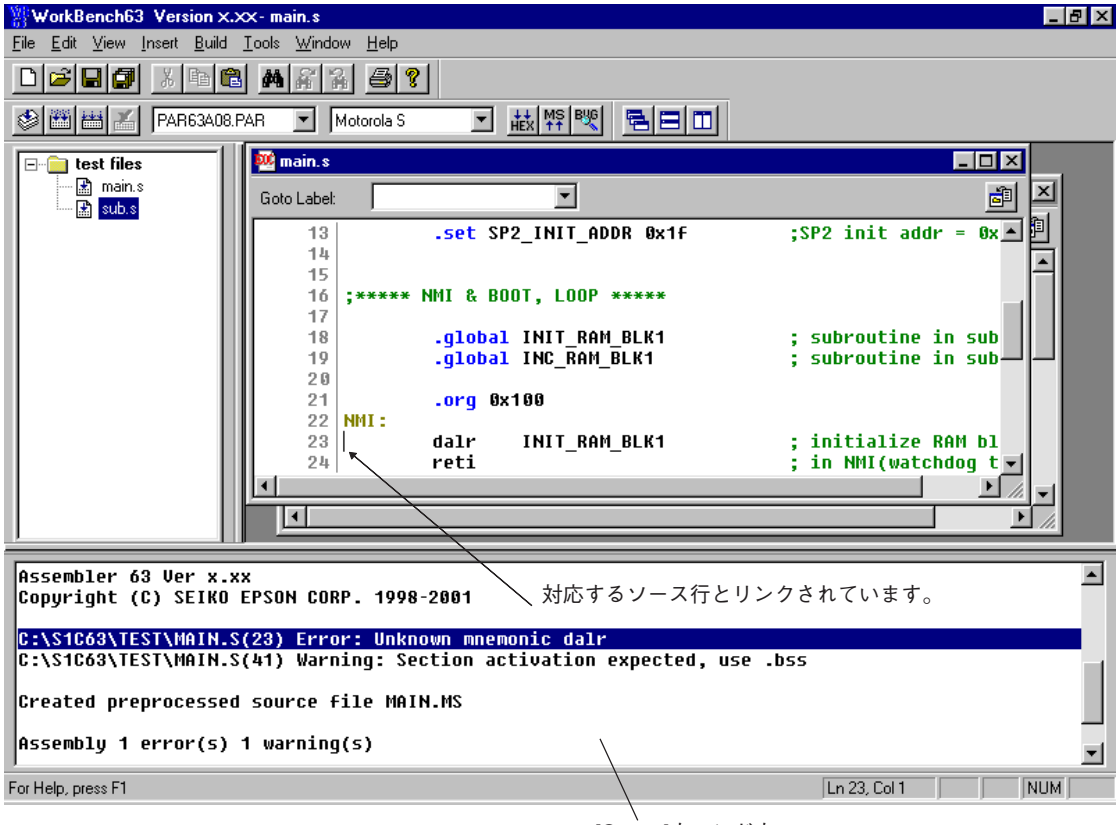

[Build]ボタン

ワークベンチは、アセンブラとリンカを起動し、プロジェクト内のソースファイルから実行形式のオブジェクトファイルを作成します。[Output format]ボックスでHEXファイル形式(インテルHEXまたはモトローラS)を選択している場合は、リンク後にHEXコンバータも実行されます。デフォルトでは、IEEE-695形式のアブソリュートオブジェクトファイルが生成されます。



[Output format]ボックス

各ツールが実行中に出力するメッセージは[Output]ウィンドウに表示されます。ワークベンチはタグジャンプ機能を持っており、[Output]ウィンドウに表示された文法エラーなどのエラーメッセージをダブルクリックすると、[Edit]ウィンドウの該当するソース行にジャンプすることができます。該当のソースファイルが開いていない場合は、自動的に開きます。



対応するソース行とリンクされています。

[Output]ウィンドウ

ビルドでは、必要なファイルのみを更新する一般的なmake処理が実行されます。このmake機能を使用せずに最初からすべてのソースファイルを対象にビルドを行うには、[Build]メニューから[Rebuild All]を選択(または[Rebuild All]ボタンをクリック)してください。


[Rebuild All]ボタン

文法エラーなどの修正のためにアセンブラのみを起動させたい場合は、[Build]メニューから[Assemble]を選択(または[Assemble]ボタンをクリック)してください。


[Assemble]ボタン

2.2.6 デバッグ

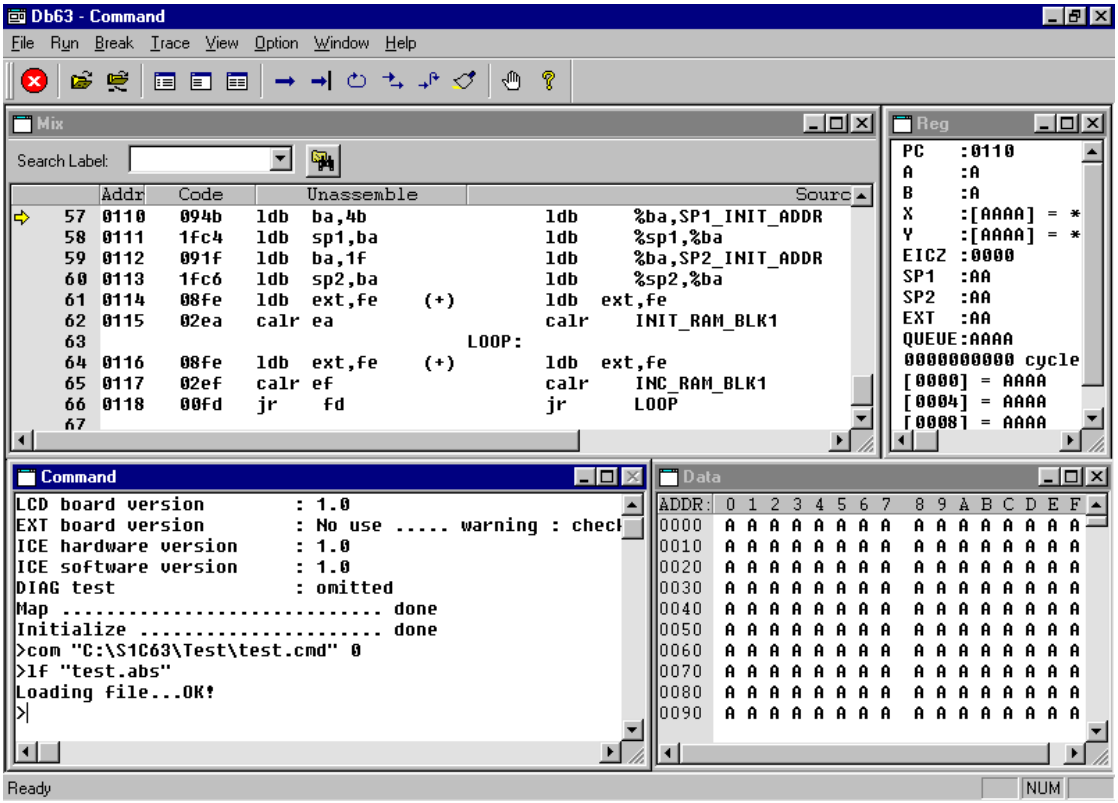
実行形式オブジェクトを作成後、プログラムのデバッグは次のように行います。

1. [Build]メニューから[Debug]を選択します(または[Debug]ボタンをクリックします)。


[Debug]ボタン

デバッガが、指定されているICEパラメータファイルに従って起動し、実行形式のオブジェクトファイルをロードしてデバッグ可能な状態になります。

注:　デバッガを起動する前に、ICEをデバッグ可能な状態にしておく必要があります。ICEの設定と起動方法については、ICEのマニュアルを参照してください。



デバッグ機能と操作方法については、"8 デバッガ"を参照してください。

3　ワークベンチ

この章では、ワークベンチwb63の機能と操作方法を説明します。

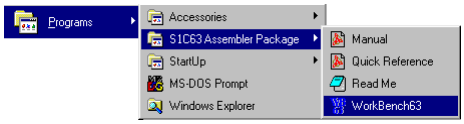
3.1　特長

ワークベンチwb63は、ソースの作成/編集からデバッガまでの統合操作環境を実現するソフトウェアです。機能と特長を次にまとめます。

- コピー/ペースト、検索/置き換え、印刷、ラベルジャンプ、エラーメッセージからのタグジャンプに対応したソース編集機能
- 必要なファイルやツールの設定などの情報をプロジェクトとして容易に管理可能
- 更新が必要なファイルのみを処理するmake処理による実行形式オブジェクトのビルド
- アセンブラ、リンカ、HEXコンバータ、ディスアセンブラ、デバッガのすべてのオプションをサポート
- 容易な操作環境を提供するWindows GUIインタフェース

3.2　起動および終了方法

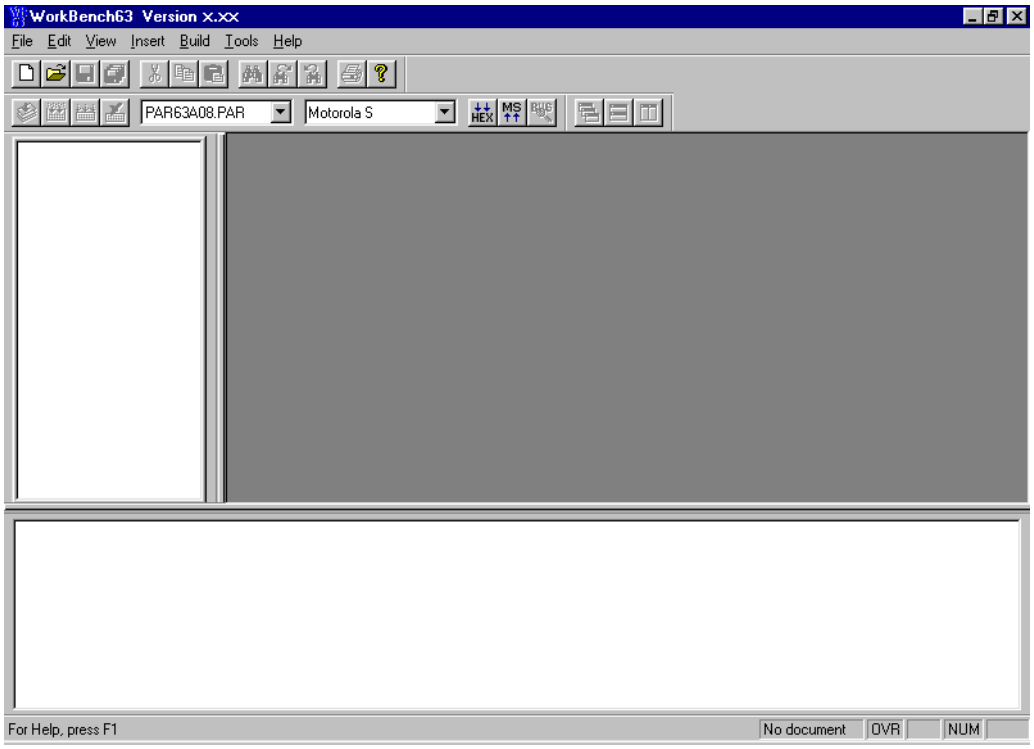
●ワークベンチを起動するには



[プログラム]メニューから"WorkBench63"を選択して起動させてください。

※ [プログラム]メニューに"WorkBench63"が登録されていない場合は、インストールが正しく行われていません。再度インストールを行ってください。

ワークベンチが起動すると次の実行ウィンドウが表示されます。

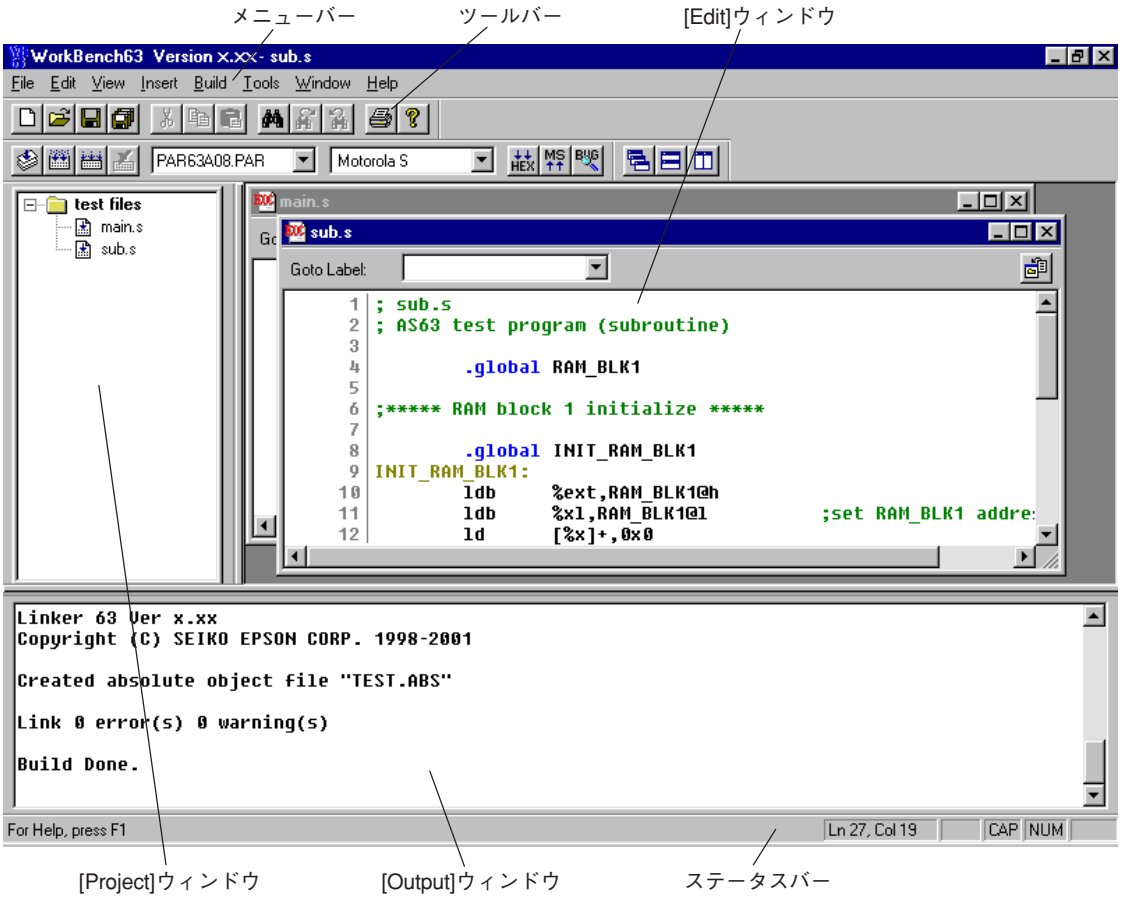


●ワークベンチを終了するには

[File]メニュから[Exit]を選択してください。

3.3 ワークベンチウィンドウ

3.3.1 ウィンドウの構成



ワークベンチは[Edit]ウィンドウ、[Project]ウィンドウおよび[Output]ウィンドウの3種類のウィンドウを持ちます。

●[Edit]ウィンドウ

ソースファイルの編集に使用します。ソース以外の標準テキストファイルも表示可能です。[Edit]ウィンドウ領域には複数の[Edit]ウィンドウを開くことができます。

アセンブリソースファイルを開いた場合は、ソースをステートメントの種類により色分けして表示します。

S1C63の命令:	黒
前処理用擬似命令(#xxx):	茶(暗)
アセンブル用擬似命令(.xxx):	青
ラベル:	茶(明)
コメント:	緑

これらの色は[Tools¦Options]メニューコマンドにより変更することができます(3.10項参照)。

●[Project]ウィンドウ

現在開いているワークスペースフォルダとプロジェクト中のすべてのソースファイル名を、Windowsエクスプローラと同様に表示します。

リストされているソースファイルのアイコンをダブルクリックするとそのソースファイルが開き、内容が[Edit]ウィンドウに表示されます。

●[Output]ウィンドウ

ビルドやアセンブルの処理で実行されるツールが出力するメッセージを表示します。
このウィンドウに表示されるソース行番号を含む文法エラーなどのエラーメッセージをダブルクリックすると、対応するソースの[Edit]ウィンドウが開き、あるいはアクティブになり、エラーが発生したソース行を表示します。

●メニューバー

3.5項を参照してください。

●ツールバー

3.4項を参照してください。

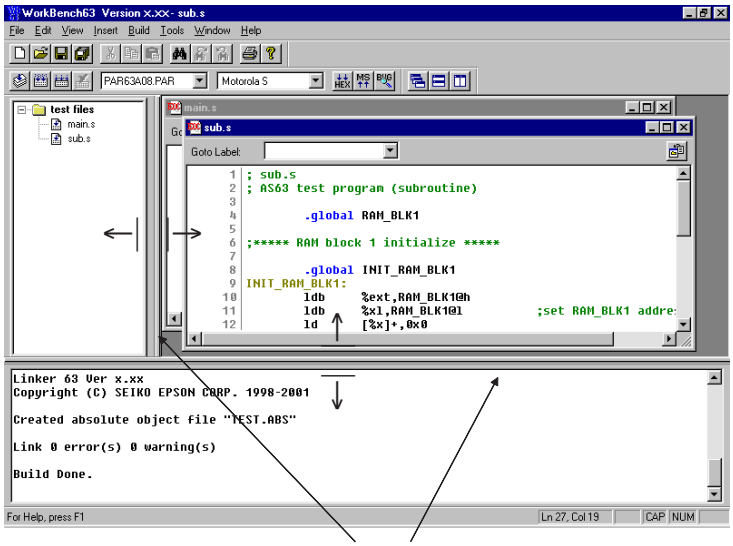
●ステータスバー

マウスカーソルをメニュー項目やボタンに重ねると、この機能を示す簡易ヘルプが表示されます。
また、[Edit]ウィンドウ中のカーソル位置やキーロック（Num lock、Caps lock、Scroll lock）の状態を表示します。

3.3.2 ウィンドウの操作

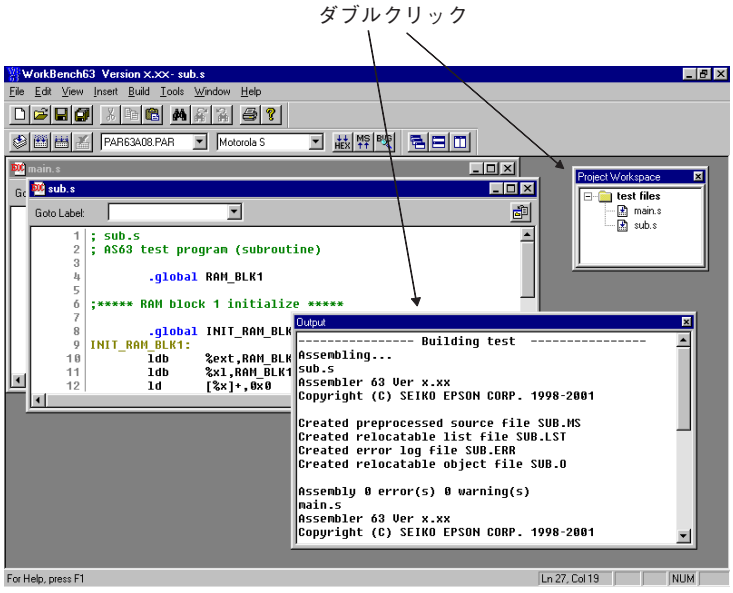
●ウィンドウのサイズ変更

各ウィンドウ領域は、ウィンドウの境界をドラッグすることによってサイズを変更することができます。ウィンドウサイズの情報はワークベンチ終了時にセーブされ、次にワークベンチを起動した際には終了時と同じレイアウトとなります。



●[Project], [Output]ウィンドウのフローティングとドッキング

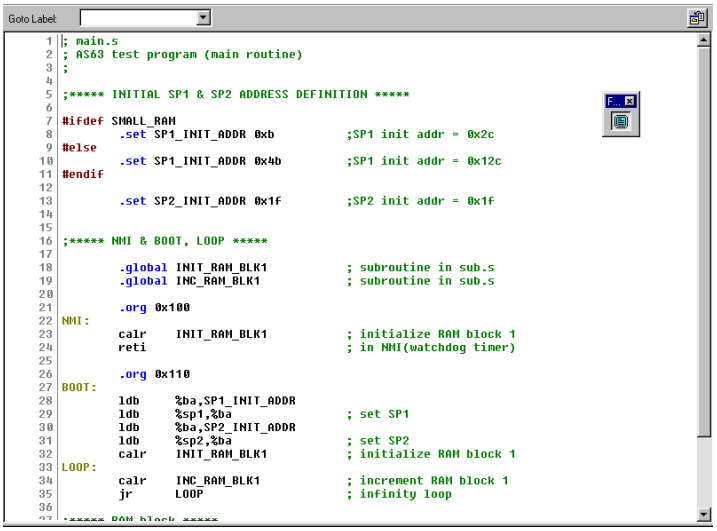
[Project]ウィンドウと[Output]ウィンドウは、ウィンドウ境界をダブルクリックすることによってフローティングウィンドウに変わります。フローティングウィンドウにすると、ワークベンチウィンドウ内で自由に移動、サイズ変更が可能になります。フローティング状態をドッキング状態に戻すには、ウィンドウのタイトルバーをダブルクリックするか、タイトルバーをワークベンチウィンドウの端にドラッグします。



●[Project], [Output]ウィンドウのクローズ

[Project]ウィンドウと[Output]ウィンドウは、それぞれ[View]メニューから[Project Window]、[Output Window]を選択することによって閉じることができます。開くには、再度メニューコマンドを選択してください。

●[Edit]ウィンドウ領域の最大化



[Edit]ウィンドウ領域は、[View]メニューの[Full Screen]を選択することによって全画面サイズに拡大できます。他のウィンドウおよびツールバーは[Edit]ウィンドウ領域の後ろに隠れます。
元の表示に戻すには、画面上に表示されるボタンをクリックしてください。このボタンは、タイトルバーをドラッグすることで画面内の任意の位置に移動可能です。また、[ESC]キーによっても、元の表示に戻ります。

●[Edit]ウィンドウのオープン/クローズ

[Edit]ウィンドウは、メニュー、ボタンまたは[Project]ウィンドウのファイルアイコンでソースファイル(テキストファイル)を開くか、ソースの新規作成を選択すると開きます。
閉じるには、各[Edit]ウィンドウのタイトルバー上にある[閉じる]ボタンをクリックするか、[File]メニューから[Close]を選択します。
プロジェクトファイルをセーブすると、[Edit]ウィンドウの情報(ファイル名、サイズ、位置)もセーブされます。次にプロジェクトを開いた際には、セーブされた状態から編集を再開できます。

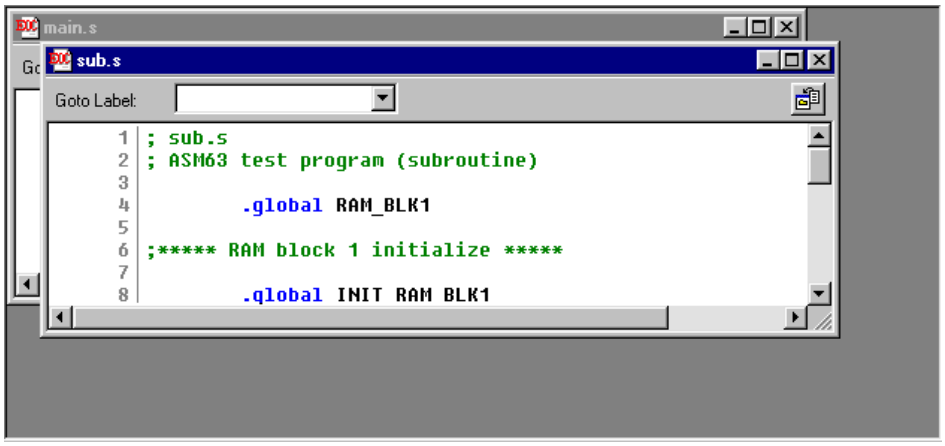
●[Edit]ウィンドウの整列

開いている[Edit]ウィンドウは一般のWindowsアプリケーションと同様に整列させることができます。

1 カスケード表示

[Window]メニューから[Cascade]を選択、または[Cascade Windows]ボタンをクリックすると、[Edit]ウィンドウを斜めに重ねて表示します。

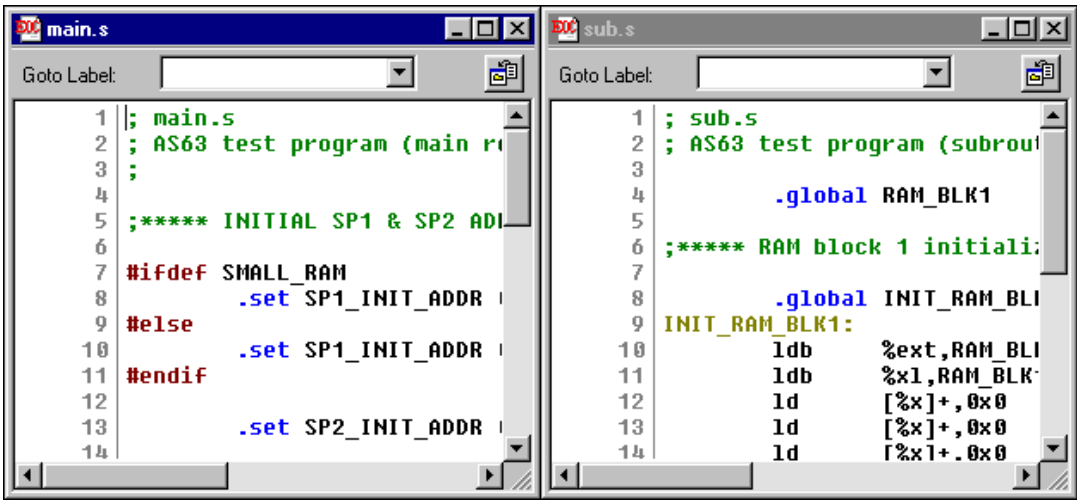

[Cascade Windows]ボタン



2 タイル表示

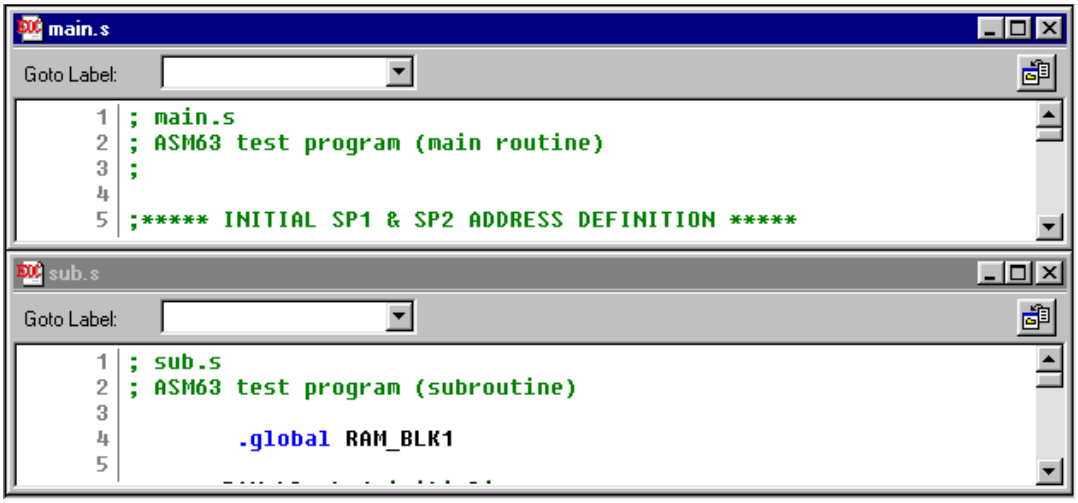
[Window]メニューから[Tile Vertically]を選択、または[Tile Vertically]ボタンをクリックすると、[Edit]ウィンドウを横に並べて表示します。

[Tile Vertically]ボタン

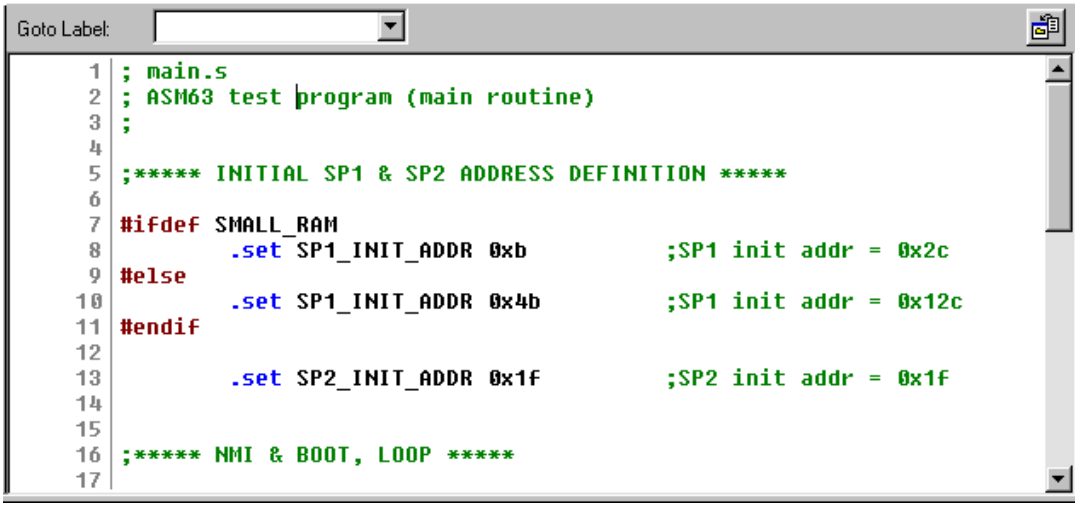


[Window]メニューから[Tile Horizontally]を選択、または[Tile Horizontally]ボタンをクリックすると、[Edit]ウィンドウを縦に並べて表示します。

[Tile Horizontally]ボタン



3 [Edit]ウィンドウの最大化
タイトルバー上の[最大化]ボタンをクリックします。その[Edit]ウィンドウが[Edit]ウィンドウ領域サイズに拡大され、他の[Edit]ウィンドウはその後ろに隠れます。



4 [Edit]ウィンドウの最小化
タイトルバー上の[最小化]ボタンをクリックします。その[Edit]ウィンドウはウィンドウアイコンに最小化されます。最小化されたウィンドウアイコンは[Window]メニューの[Arrange Icons]によって、[Edit]ウィンドウ領域の下部に整列させることができます。

5 [Edit]ウィンドウの移動とサイズ変更
最大化されていない[Edit]ウィンドウは、一般のWindowsアプリケーションと同様に位置とサイズを変更することができます。

●アクティブ[Edit]ウィンドウの切り替え

アクティブにしたい[Edit]ウィンドウをクリックします。もし、他のウィンドウに隠れている場合は、[Window]メニュー内に表示される、現在開いているウィンドウのリストからアクティブにしたいウィンドウ名(ソースファイル名)を選択してください。

●表示内容のスクロール

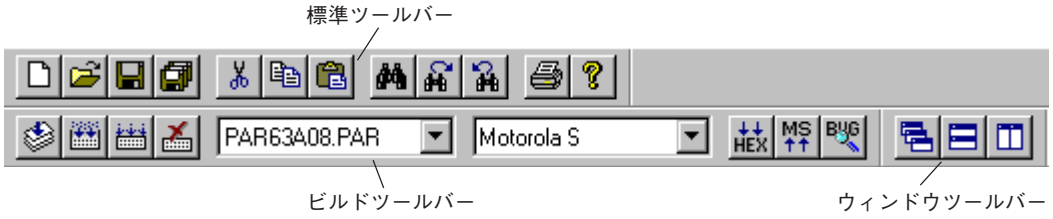
表示内容がウィンドウサイズに収まらない場合は標準のスクロールバーが表示されますので、それを使用して内容をスクロールさせてください。矢印キーも使用可能です。

●ステータスバーの表示/非表示

ステータスバーの表示/非表示は、[View]メニューの[Status Bar]で切り替えることができます。

3.4　ツールバーとボタン

ワークベンチには、標準ツールバー、ビルドツールバー、ウィンドウツールバーの3種類のツールバーが用意されています。



3.4.1 標準ツールバー

このツールバーには以下のボタンが登録されています。

[New]ボタン
ドキュメントを新規作成します。ドキュメントの種類をアセンブリソース、アセンブリヘッダ、プロジェクトの3種類から選択するダイアログボックスが表示されます。

[Open]ボタン
ドキュメントを開きます。ファイルを選択するダイアログボックスが表示されます。

[Save]ボタン
アクティブな[Edit]ウィンドウ内のテキストをファイルにセーブします。ファイルは上書きされます。このボタンは、[Edit]ウィンドウが1つも開いていない場合は無効になります。

[Save All]ボタン
開いているすべての[Edit]ウィンドウの内容とプロジェクトの情報をそれぞれのファイルにセーブします。

[Cut]ボタン
[Edit]ウィンドウ内で選択されているテキストをクリップボードにカットします。

[Copy]ボタン
[Edit]ウィンドウ内で選択されているテキストをクリップボードにコピーします。

[Paste]ボタン
クリップボードにコピーされているテキストを[Edit]ウィンドウのカーソル位置に挿入、または選択範囲をクリップボードのテキストで置き換えます。

[Find]ボタン
アクティブな[Edit]ウィンドウ内で、指定の文字列を検索します。検索する文字列と検索条件を指定するダイアログボックスが表示されます。

[Find Next]ボタン
ファイルの終端方向に次の検索を行います。

[Find Previous]ボタン
ファイルの先頭方向に次の検索を行います。

[Print]ボタン
アクティブな[Edit]ウィンドウの内容を印刷します。印刷条件を設定する標準の印刷ダイアログボックスが表示されます。

[Help]ボタン
ヘルプウィンドウを表示します。

3.4.2 ビルドツールバー

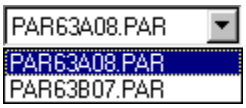
このツールバーにはプロジェクトのビルドに使用するボタンとリストボックスが登録されています。

[Assemble]ボタン
アクティブな[Edit]ウィンドウのアセンブリソースをアセンブルします。アクティブな[Edit]ウィンドウがアセンブリソース以外を表示している場合、このボタンは無効となります。

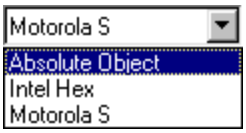
[Build]ボタン
現在開いているプロジェクトのmake処理を行い、実行形式オブジェクトをビルドします。

[Rebuild All]ボタン
現在開いているプロジェクトのビルドを行います。ファイルの更新状況にかかわらず、すべてのアセンブリソースのアセンブルから処理されます。

[Stop Build]ボタン
実行中のビルド処理を中止します。ビルド中以外、このボタンは無効となります。



[ICE Parameter]プルダウンリストボックス
開発機種のICEパラメータファイルを選択します。ここには、"dev63"ディレクトリ内に存在するICEパラメータファイルがリストされます。



[Output Format]プルダウンリストボックス
ビルドで生成する実行形式オブジェクトのファイル形式を選択します。IEEE-695形式のアブソリュートオブジェクト、インテルHEX形式、モトローラS形式の3種類から選択できます。

[HEX Convert]ボタン
HEXコンバータを起動し、アブソリュートオブジェクトをインテルHEX形式またはモトローラS形式に変換します。アブソリュートオブジェクトファイルとHEXコンバータのオプションを指定するダイアログボックスが表示されます。

[Disassemble]ボタン
ディスアセンブラを起動し、アブソリュートオブジェクトを逆アセンブルします。アブソリュートオブジェクトファイルとディスアセンブラのオプションを指定するダイアログボックスが表示されます。

[Debug]ボタン
指定のICEパラメータファイルでデバッガを起動します。

3.4.3 ウィンドウツールバー

このツールバーにはウィンドウ操作に使用するボタンが登録されています。

[Cascade]ボタン
開いている[Edit]ウィンドウを斜めに重ねて表示します。

[Tile Horizontally]ボタン
開いている[Edit]ウィンドウを縦に並べて表示します。

[Tile Vertically]ボタン
開いている[Edit]ウィンドウを横に並べて表示します。

3.4.4 ツールバーの操作

●ツールバーの表示/非表示

ツールバーが不要な場合は、[View]メニューからツールバー名を選択し、それぞれ非表示とすることができます。このメニューコマンドは選択ごとに表示と非表示を切り替えます。

●ツールバー位置の変更

各ツールバーはドラッグすることでツールバー領域内の位置を変更することができます。ツールバー領域外へドラッグすると、ウィンドウ形式に変更されます。

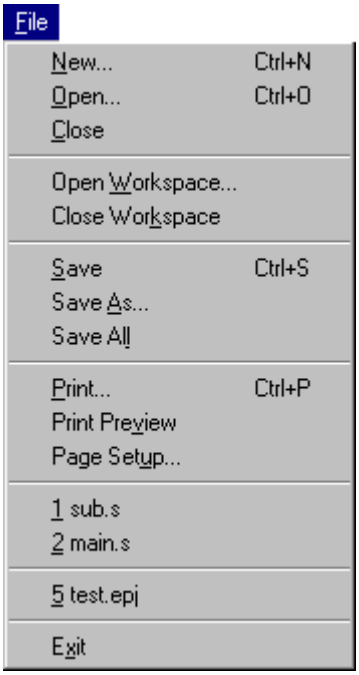
3.4.5 [Edit]ウィンドウの[Insert into project]ボタン

 [Insert into project]ボタン

ソースファイル(.s、.ms)を開くと[Edit]ウィンドウには[Insert into project]ボタンが表示されます。このボタンは、そのウィンドウで開いているソースファイルを現在のプロジェクトに追加するのに使用します。
ソース以外のテキストファイルを開いた場合、その[Edit]ウィンドウに[Insert into project]ボタンは付加されません。

3.5 メニュー

3.5.1 [File]メニュー



このメニューにリストされているファイル名は最近開いたソースとプロジェクトファイルです。ここからの選択でも、そのファイルを開くことができます。リストするファイル数は[Tools ¦ Options]メニューコマンドで設定できます。

[New...] ([Ctrl]+[N])
ドキュメントを新規作成します。ドキュメントの種類をアセンブリソース、アセンブリヘッダ、プロジェクトの3種類から選択するダイアログボックスが表示されます。

[Open...] ([Ctrl]+[O])
ドキュメントを開きます。ファイルを選択するダイアログボックスが表示されます。

[Close]
アクティブな[Edit]ウィンドウを閉じます。このメニューコマンドは[Edit]ウィンドウがアクティブになると表示されます。

[Open Workspace...]
プロジェクトを開きます。プロジェクトを選択するダイアログボックスが表示されます。

[Close Workspace]
現在開いているプロジェクトを閉じます。このメニューコマンドはプロジェクトが開かれていない場合は無効となります。

[Save] ([Ctrl]+[S])
アクティブな[Edit]ウィンドウ内のテキストをファイルにセーブします。ファイルは上書きされます。このメニューコマンドは、[Edit]ウィンドウがアクティブになると表示されます。

[Save As...]
アクティブな[Edit]ウィンドウ内のテキストを他の名前でセーブします。セーブする場所とファイル名を指定するダイアログボックスが表示されます。このメニューコマンドは、[Edit]ウィンドウがアクティブになると表示されます。

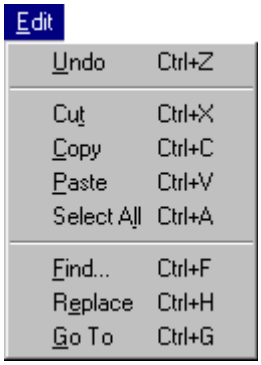
[Save All]
開いているすべての[Edit]ウィンドウの内容とプロジェクトの情報をそれぞれのファイルにセーブします。

[Print...] ([Ctrl]+[P])
アクティブな[Edit]ウィンドウの内容を印刷します。印刷条件を設定する標準の印刷ダイアログボックスが表示されます。このメニューコマンドは、[Edit]ウィンドウがアクティブになると表示されます。

[Print Preview]
アクティブな[Edit]ウィンドウ内のドキュメントの印刷イメージを表示します。このメニューコマンドは、[Edit]ウィンドウがアクティブになると表示されます。

[Page Setup...]
用紙やプリンタを選択するダイアログボックスを表示します。

3.5.2 [Edit]メニュー



[Undo] ([Ctrl]+[Z])
[Edit]ウィンドウに対する直前の操作を取り消します。

[Cut] ([Ctrl]+[X])
[Edit]ウィンドウ内で選択されているテキストを、クリップボードにカットします。

[Copy] ([Ctrl]+[C])
[Edit]ウィンドウ内で選択されているテキストを、クリップボードにコピーします。

[Paste] ([Ctrl]+[V])
クリップボードにコピーされているテキストを[Edit]ウィンドウのカーソル位置に挿入、または選択範囲をクリップボードのテキストで置き換えます。

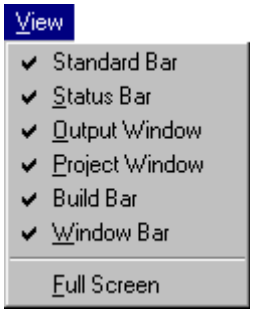
[Select All] ([Ctrl]+[A])
アクティブな[Edit]ウィンドウ内のテキストをすべて選択します。

[Find...] ([Ctrl]+[F])
アクティブな[Edit]ウィンドウ内で、指定の文字列を検索します。検索する文字列と検索条件を指定するダイアログボックスが表示されます。

[Replace] ([Ctrl]+[H])
アクティブな[Edit]ウィンドウ内で指定の文字列を検索し、他の文字列に置き換えます。検索文字列と置き換え文字列を指定するダイアログボックスが表示されます。

[Go To] ([Ctrl]+[G])
アクティブな[Edit]ウィンドウ内の指定の行番号またはラベル位置にジャンプします。行番号またはラベル名を指定するダイアログボックスが表示されます。

3.5.3 [View]メニュー



[Standard Bar]
標準ツールバーの表示/非表示を切り替えます。

[Status Bar]
ステータスバーの表示/非表示を切り替えます。

[Output Window]
[Output]ウィンドウ表示/非表示を切り替えます。

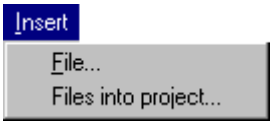
[Project Window]
[Project]ウィンドウの表示/非表示を切り替えます。

[Build Bar]
ビルドツールバーの表示/非表示を切り替えます。

[Window Bar]
ウィンドウツールバーの表示/非表示を切り替えます。

[Full Screen]
[Edit]ウィンドウ領域を全画面サイズに拡大します。

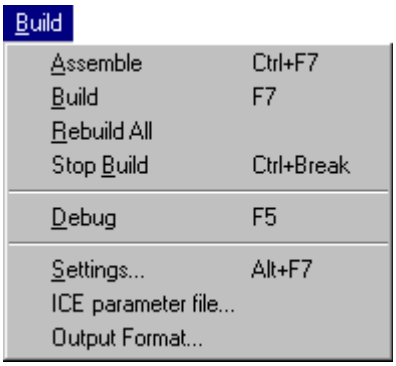
3.5.4 [Insert]メニュー



[File...]
指定のファイルを[Edit]ウィンドウのカーソル位置に挿入、または選択範囲を指定ファイルの内容で置き換えます。ファイルを選択するダイアログボックスが表示されます。

[Files into project...]
指定のソースファイルを現在開いているプロジェクトに追加します。ソースファイルを選択するダイアログボックスが表示されます。

3.5.5 [Build]メニュー



[Assemble] ([Ctrl]+[F7])
アクティブな[Edit]ウィンドウのアセンブリソースをアセンブルします。アクティブな[Edit]ウィンドウがアセンブリソース以外を表示している場合、このメニューコマンドは無効となります。

[Build] ([F7])
現在開いているプロジェクトのmake処理を行い、実行形式オブジェクトをビルドします。

[Rebuild All]
現在開いているプロジェクトのビルドを行います。ファイルの更新状況にかかわらず、すべてのアセンブリソースのアセンブルから処理されます。

[Stop Build] ([Ctrl]+[Break])
実行中のビルド処理を中止します。ビルド中以外、このメニューコマンドは無効となります。

[Debug] ([F5])
指定のICEパラメータファイルでデバッガを起動します。

[Settings...] ([Alt]+[F7])
各ツールのオプションを設定するダイアログボックスを表示します。

[ICE parameter file...]
ICEパラメータファイルを選択するダイアログボックスを表示します。

[Output Format...]
実行形式オブジェクトの出力形式を選択するダイアログボックスを表示します。IEEE-695形式のアブソリュートオブジェクト、インテルHEX形式、モトローラS形式の3種類から選択できます。ビルドを実行すると、ここで選択した形式のファイルが生成されます。

3.5.6 [Tools]メニュー

[HEX Converter...]
HEXコンバータを起動し、アブソリュートオブジェクトをインテルHEX形式またはモトローラS形式に変換します。アブソリュートオブジェクトファイルとHEXコンバータのオプションを指定するダイアログボックスが表示されます。

[Disassembler...]
ディスアセンブラを起動し、アブソリュートオブジェクトを逆アセンブルします。アブソリュートオブジェクトファイルとディスアセンブラのオプションを指定するダイアログボックスが表示されます。

[WinFOG]
ファンクションオプションジェネレータwinfogを起動します。

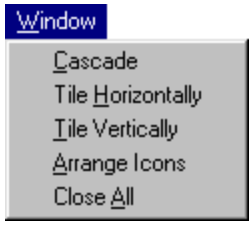
[WinSOG]
セグメントオプションジェネレータwinsogを起動します。

[WinMDC]
マスクデータチェッカwinmdcを起動します。

[OBPW63]
On Board Writerコントロールソフトウェアを起動します。

[Options...]
[Edit]ウィンドウ内のテキストの表示色や印刷用フォントなど、ワークベンチのオプションを選択するダイアログボックスが表示されます。

3.5.7 [Window]メニュー



このメニューは[Edit]ウィンドウを開くと表示されます。

[Cascade]
開いている[Edit]ウィンドウを斜めに重ねて表示します。

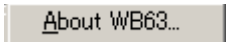
[Tile Horizontally]
開いている[Edit]ウィンドウを縦に並べて表示します。

[Tile Vertically]
開いている[Edit]ウィンドウを横に並べて表示します。

[Arrange Icons]
最小化された[Edit]ウィンドウアイコンを[Edit]ウィンドウ領域下部に整列させます。

[Close All]
開いている[Edit]ウィンドウをすべて閉じます。

3.5.8 [Help]メニュー



[About WB63...]
ワークベンチのバージョン情報を表示します。

3.6 プロジェクトとワークスペース

ワークベンチは、プログラム開発のタスクをワークスペースフォルダと、ファイル情報やツールの起動に必要な情報などを含むプロジェクトファイルで管理します。

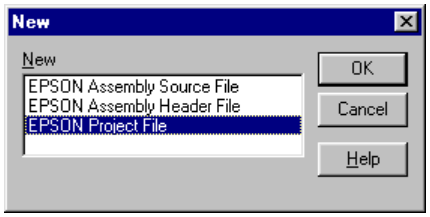
3.6.1 プロジェクトの新規作成

プロジェクトファイルは以下の手順で作成できます。

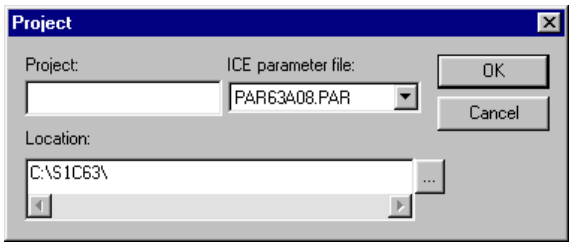
1. [File]メニューから[New]を選択するか、[New]ボタンをクリックします。

    [New]ボタン

   [New]ダイアログボックスが表示されます。



2. [EPSON Project File]を選択後、[OK]をクリックします。
   [Project]ダイアログボックスが表示されます。



3. プロジェクト名を入力し、ICEパラメータファイルとディレクトリを選択後、[OK]をクリックします。

   * [ICE parameter file:]ボックスには、"dev"ディレクトリ内に存在するICEパラメータファイルがリストされます。

ワークベンチはワークスペースとして、指定のプロジェクト名でフォルダ(ディレクトリ)を作成し、その中にプロジェクトファイル(.epj)を作成します。
指定位置にすでに同じ名前のフォルダが存在する場合は、そのフォルダをそのままワークスペースとして使用します。
ここで指定したプロジェクト名は、ビルドにより生成されるアブソリュートオブジェクトファイルやその他のファイル名としても使用されます。

3.6.2 プロジェクトへのソースファイルの登録

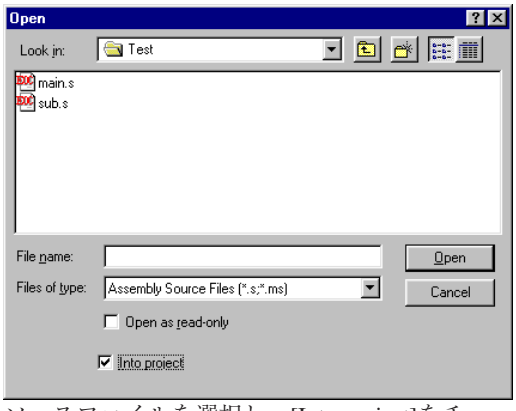
作成したソースファイルはプロジェクトに登録しておく必要があります。
これには、以下に示す4つの方法のいずれかを使用してください。

1. [Insert ¦ Files into project...]メニューコマンド
   このメニューコマンドを選択すると、ダイアログボックスが表示されます。

   ソースファイルを選択し、[Open]ボタンをクリックします。

注: 複数のソースファイルをまとめて選択する場合は、ファイル名のトータル文字数が256文字を超えないようにしてください。

2. [File ¦ Open...]メニューコマンドまたは[Open]ボタン
    [Open]ボタン
   このメニューコマンドまたはボタンを選択すると、ダイアログボックスが表示されます。



   ソースファイルを選択し、[Into project]をチェックして[Open]ボタンをクリックしてください。

3. [Edit]ウィンドウ上の[Insert into project]ボタン
    [Insert into project]ボタン
   ソースファイルが開かれている場合は、[Edit]ウィンドウ上の[Insert into project]ボタンをクリックしてください。ソースを新規作成している場合は、プロジェクトに挿入する前に一度ファイルにセーブしてください。

4. [Project]ウィンドウへのソースファイルのドラッグ
   Windowsエクスプローラから[Project]ウィンドウ内へソースファイルをドラッグしてください。ソースファイルが現在開いているプロジェクトに挿入されます。

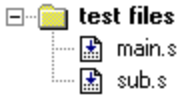
プロジェクトに登録されたソースファイルは[Project]ウィンドウにリストされます。

●プロジェクトからのソースファイルの削除

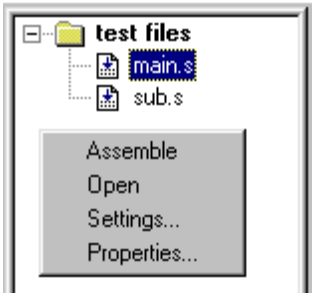
ソースファイルをプロジェクトから削除するには、[Project]ウィンドウでそのソースファイル名を選択し、[Delete]キーを押します。この操作は、プロジェクトからソース情報を削除するのみで、実際のソースファイルには影響を与えません。

3.6.3 [Project]ウィンドウ

[Project]ウィンドウはワークスペースフォルダと、現在開いているプロジェクトに含まれるソースの一覧を表示します。

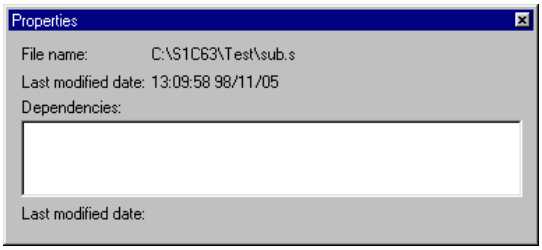


ソースファイルのアイコンをダブルクリックすると、そのファイルが開きます。すでに開かれている場合は、対応する[Edit]ウィンドウがアクティブになります。



[Project]ウィンドウのショートカットメニュー

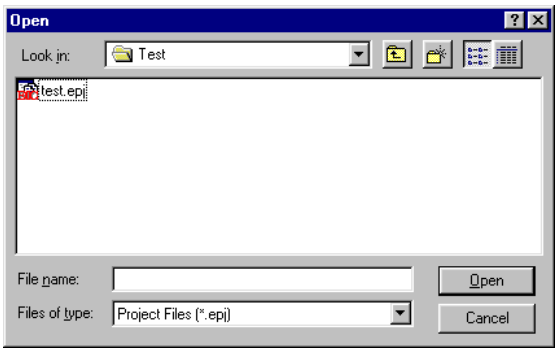
フォルダアイコンまたはソースファイルアイコンをマウスの右ボタンでクリックすると、実行可能なビルドメニューコマンドを持つショートカットメニューが表示されます。
[Properties...]を選択すると、次のようにソースファイルの情報を表示します。



注: [Project]ウィンドウのリストは、実際のディレクトリ構造を表しているわけではありません。ワークスペースフォルダ以外にセーブされているソースファイルも、ワークスペースフォルダ内に存在するように表示されますので注意してください。

3.6.4 プロジェクトのオープン/クローズ

プロジェクトを開くには、[File]メニューから[Open WorkSpace...]を選択してください。
プロジェクトファイルを選択するダイアログボックスが表示されます。



ワークベンチは、複数のプロジェクトを同時に開くことができません。プロジェクトが開いている状態で他のプロジェクトをオープンすると、前のプロジェクトは閉じられます。このとき、前のプロジェクトが修正後に保存されていない場合は、プロジェクトをセーブするかどうかを選択するダイアログボックスが表示されます。

プロジェクトファイルは[File]メニューの[Open]または[Open]ボタンによっても開くことができます。この場合は、ファイルオープンダイアログボックス内のファイル形式をProject Files(*.epj)にしてプロジェクトファイルを選択してください。

現在開いているプロジェクトを閉じるには、[File]メニューの[Close WorkSpace]を選択してください。プロジェクトが修正後に保存されていない場合は、プロジェクトをセーブするかどうかを選択するダイアログボックスが表示されます。[Yes]を選択すると、ソース、ツール設定、ウィンドウ情報を含むすべての変更項目がセーブされます。

3.6.5 ワークスペースフォルダ内のファイル

ワークベンチは以下のファイルをワークスペースフォルダ内に作成します。

<file>.epj　　プロジェクトファイル

このファイルにはプロジェクト情報が記録されます。

<file>.cm　　リンカコマンドファイル

このファイルはビルド開始時に作成され、リンカがアブソリュートオブジェクトファイルを生成するために使用します。

例:
```
; S1C WorkBench Generated
; Thursday, November 05, 1998

"C:¥EPSON¥S1C63¥DEV¥63A08¥PAR63A08.PAR"          ;ICE parameter file

-o "test.abs"         ;output file : absolute object

; linked object file(s)
"sub.o"
"main.o"
```

内容はプロジェクト内のソースファイルの構成とリンカのオプション設定によって変わります。

<file>.cmd　　デバッガ用コマンドファイル

このファイルはビルド開始時に作成され、デバッガが起動時に読み込んで記述されたコマンドを実行します。

例:
```
lf "test.abs"
```

ワークベンチは、出力形式で指定された実行形式オブジェクトをデバッガが起動時にロードするように、このファイルを作成します。

<file>.mak　　ビルド用makeファイル

このファイルはビルド開始時に作成され、ワークベンチによるビルド処理を制御します。

例:
```
# S1C WorkBench Generated
# Thursday, November 05, 1998

ASM = as63.exe
LINK = lk63.exe
HEX = hx63.exe
ASM_FLG = -g
LINK_FLG = -g
HEX_FLG =

ALL : test.abs

test.abs : test.cm sub.o main.o
     $(LINK) $(LINK_FLG) test.cm

sub.o : C:¥EPSON¥S1C63¥Test¥sub.s
     $(ASM) $(ASM_FLG) C:¥EPSON¥S1C63¥Test¥sub.s

main.o : C:¥EPSON¥S1C63¥Test¥main.s
     $(ASM) $(ASM_FLG) C:¥EPSON¥S1C63¥Test¥main.s
```

マクロ設定や依存リストが記述された、一般的なmakeファイルとして作成されます。

以下のファイルはビルドで実行されるツールが生成します。

<file>.o	リロケータブルオブジェクトファイル(アセンブラが生成)
<file>.abs	アブソリュートオブジェクトファイル(リンカが生成)
<file>.hsa, <file>.lsa, <file>.csa	モトローラSファイル(この形式を指定した場合にHEXコンバータが生成)
<file>h.hex, <file>l.hex, <file>c.hex	インテルHEXファイル(この形式を指定した場合にHEXコンバータが生成)

3.7 ソースエディタ

ワークベンチはソースエディタ機能を持っています。[Edit]ウィンドウ上でソースの作成と編集が行えます。

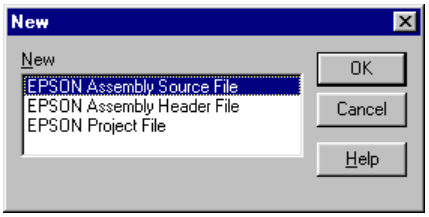
3.7.1 ソース、ヘッダファイルの新規作成

ソースファイルの新規作成するには、

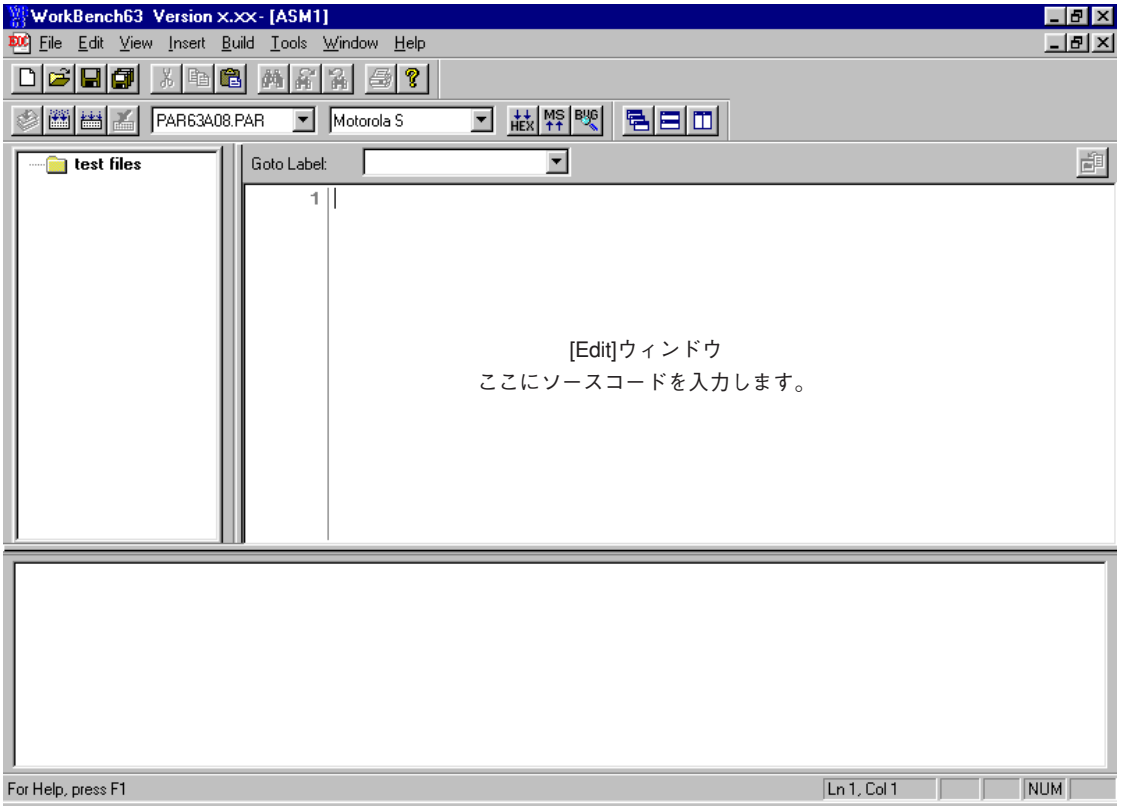
1. [File]メニューから[New]を選択するか、[New]ボタンをクリックします。

   [New]ボタン

   [New]ダイアログボックスが表示されます。



2. [EPSON Assembly Source File]を選択後、[OK]をクリックします。
   新しい[Edit]ウィンドウが開きます。



このウィンドウにソースコードを入力します。

[New]ダイアログボックスでは[EPSON Header File]も選択できます。ソースにインクルードする定数定義などのヘッダファイルを作成する場合に選択してください。

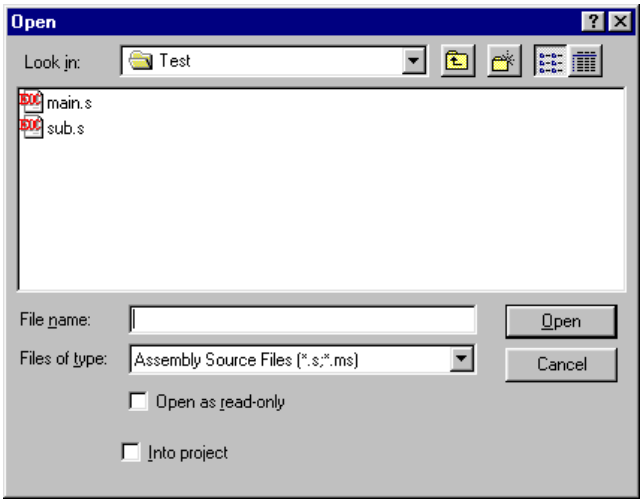
3.7.2 ファイルのロードとセーブ

ソースファイルをロードするには、

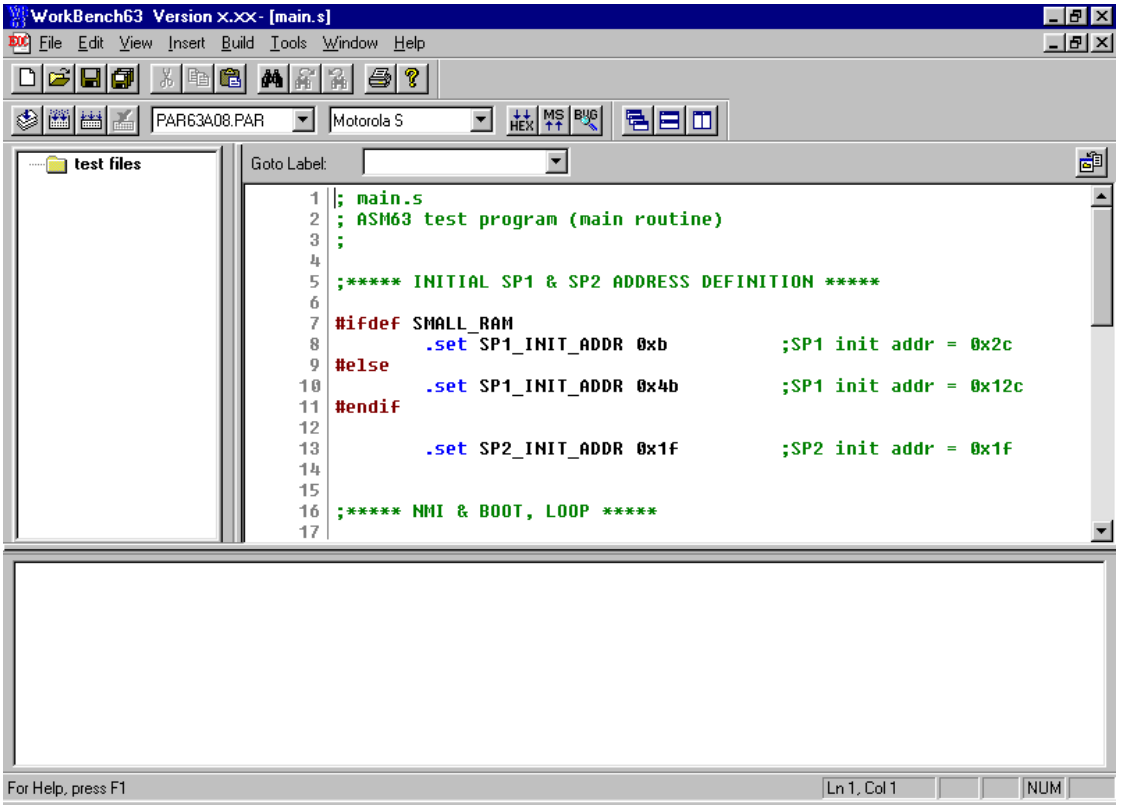
1. [File]メニューから[Open...]を選択するか、[Open]ボタンをクリックします。

   [Open]ボタン

   [Open]ダイアログボックスが表示されます。

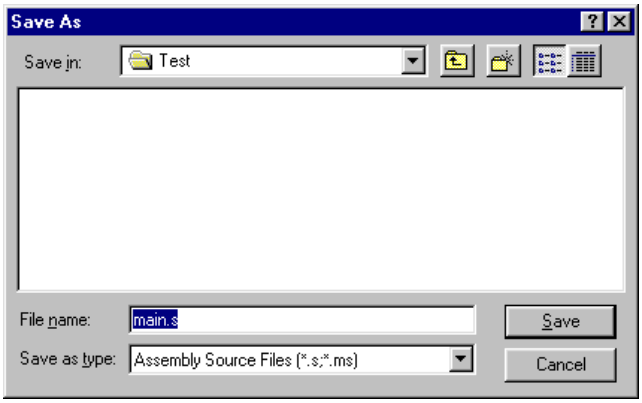


2. ファイル形式に"Assembly Source Files(*.s、*.ms)"を指定してロードするソースファイルを選択し、[OK]ボタンをクリックします。
   新しい[Edit]ウィンドウが開き、ロードしたソースを表示します。



ソースをセーブするには、

1. セーブするソースを表示している[Edit]ウィンドウをアクティブにします。
2. [File]メニューから[Save as...]を選択します。
   [Save As]ダイアログボックスが表示されます。



3. ファイル名を入力し、[OK]をクリックします。

編集中のソースファイルに上書きする場合は、[File]メニューから[Save]を選択するか、[Save]ボタンをクリックしてください。

[Save]ボタン

[File]メニューの[Save All]、または[Save All]ボタンで、開いているすべてのソースファイルとプロジェクトファイルを更新することもできます。

[Save All]ボタン

3.7.3 編集機能

ソースエディタは、一般のWindowsアプリケーションと同様の標準的なテキスト編集機能を持っています。

●テキストの編集

基本的なテキスト編集機能は、一般のWindowsアプリケーションと同様です。
カット、コピー、ペーストは[Edit]メニューとツールバーボタンで実行できます。なお、これらのコマンドは[Edit]ウィンドウに対してのみ有効です。
[Edit]ウィンドウに対する操作の取り消しは[Edit]メニューから選択できます。

タブは8文字ごとに設定されています。

●検索, 置き換え, ジャンプ

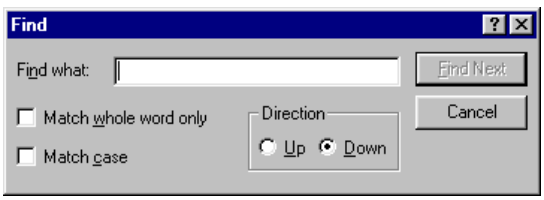
[Edit]ウィンドウでは任意の文字列を検索できます。

検　索

文字列を検索するには、[Edit]メニューから[Find...]を選択するか、[Find]ボタンをクリックします。

[Find]ボタン

[Find]ダイアログボックスが表示されます。



ダイアログボックス上の各コントロールの機能は以下のとおりです。

[Find what:]テキストボックス
検索する文字列をここに入力します。指定の文字列は、このダイアログボックスを閉じても、検索文字列として保持されます。

[Match whole word only]チェックボックス
このオプションを選択すると、ワークベンチは検索文字列に完全に一致する文字列のみを検索します。このオプションを選択しない場合は、検索文字列が部分的に含まれている場合でも検索の対象になります。

[Match case]チェックボックス
このオプションを選択すると、大文字と小文字を区別して検索が行われます。このオプションを選択しない場合は、大文字と小文字は区別されません。

[Direction]オプション
[Up]ラジオボタンを選択すると、ファイルの先頭方向に検索が行われます。[Down]を選択すると、ファイルの終わりに向かって検索が行われます。

[Find Next]ボタン
このボタンをクリックすることにより指定文字列の検索を開始します。文字列が見つかると、[Edit]ウィンドウの表示がその位置に移動し、見つかった文字列をハイライト表示します。

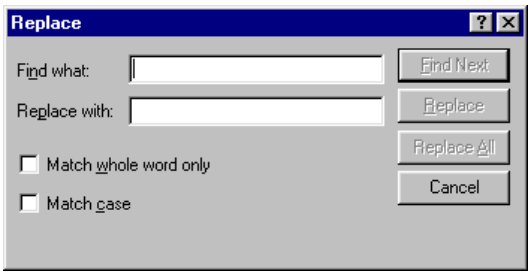
[Cancel]ボタン
このボタンをクリックすると、ダイアログボックスが閉じます。

一度このダイアログボックスで検索文字列を指定すると、ツールバーの[Find Next]と[Find Previous]ボタンが使用可能となり、引き続き前方検索、後方検索を行うことができます。

 [Find Next]ボタン　　  [Find Previous]ボタン

置き換え

文字列の置き換えを行うには、[Edit]メニューから[Replace]を選択します。
[Replace]ダイアログボックスが表示されます。



ダイアログボックス上の各コントロールの機能は以下のとおりです。

[Find what:]テキストボックス
検索する文字列を入力します。[Find]ダイアログボックスですでに検索文字列を指定している場合は、その文字列がここに表示されます。

[Replace with:]テキストボックス
置き換え文字列を入力します。

[Match whole word only]チェックボックス
このオプションを選択すると、ワークベンチは検索文字列に完全に一致する文字列のみを検索します。このオプションを選択しない場合は、検索文字列が部分的に含まれている場合でも検索の対象になります。

[Match case]チェックボックス
このオプションを選択すると、大文字と小文字を区別して検索が行われます。このオプションを選択しない場合は、大文字と小文字は区別されません。

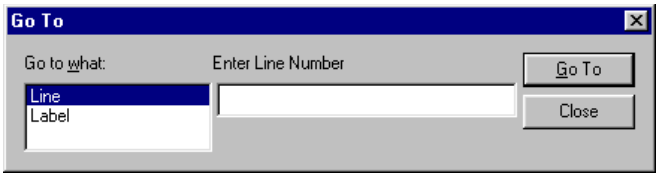
[Find Next]ボタン
このボタンをクリックすることにより指定文字列の検索を開始します。文字列が見つかると、[Edit]ウィンドウの表示がその位置に移動し、見つかった文字列をハイライト表示します。

[Replace]ボタン
検索文字列が見つかった後にこのボタンをクリックすると、その部分を置き換え文字列で置き換えます。置き換え後、ワークベンチは次の検索を開始します。

[Replace All]ボタン
ファイルの中で該当するすべての検索文字列を置き換え文字列で置き換えます。これを実行すると、最後に置き換えた文字列以外は、取り消し(Undo)できませんので注意してください。

[Cancel]ボタン
このボタンをクリックすると、ダイアログボックスが閉じます。

ジャンプ
任意の行番号またはラベルの位置に素早くジャンプできます。これには、[Edit]メニューの[Go To]を選択します。
[Go To]ダイアログボックスが表示されます。

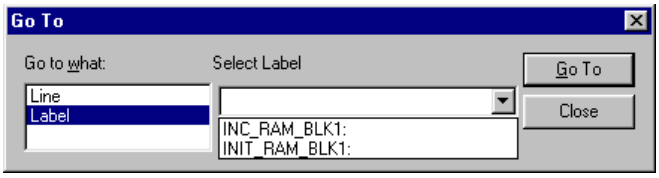


ソース行へのジャンプ

1. [Go to what:]リストボックスで"Line"を選択します。
2. [Enter Line Number]ボックスに行番号を入力し、[Go To]ボタンをクリックします。

ラベルへのジャンプ

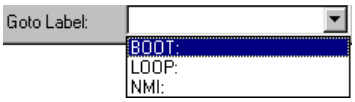
1. [Go to what:]リストボックスで"Label"を選択します。
   [Enter Line Number]ボックスが[Select Label]リストボックスに変わります。



2. [Select Label]リストボックスでラベルを選選択し、[Go To]ボタンをクリックします。

[Select Label]リストボックスのプルダウンリストには、現在編集中のソースファイル内に定義されたラベル名が表示されます。

ソースファイル(*.s、*.ms)を開いている[Edit]ウィンドウは、[Go To Label]リストボックスを持っています。[Go To]ダイアログボックスの[Select Label]と同様に、ここでもジャンプ先のラベルを選択できます。

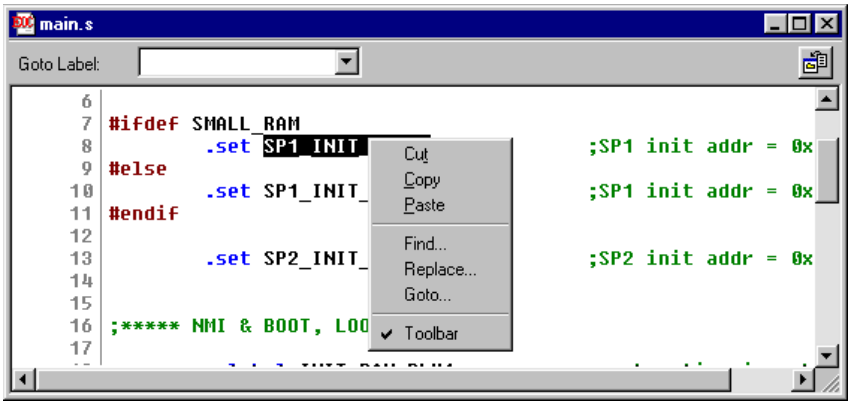


●ファイルの挿入

ヘッダファイルや他のソースファイルを現在のソースのカーソル位置に挿入するには、[Insert]メニューの[File...]を選択します。
挿入するファイルを選択するダイアログボックスが表示されます。

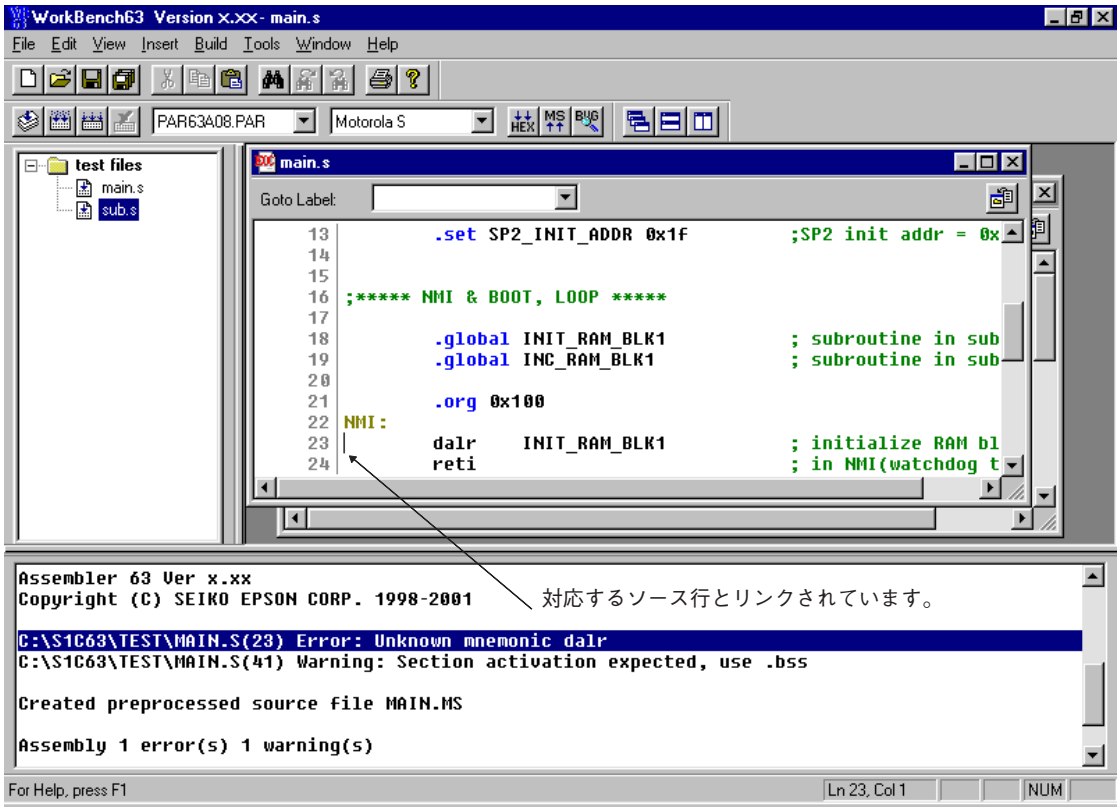
●ショートカットメニュー

[Edit]ウィンドウは、ウィンドウ上でマウスの右ボタンをクリックすることにより表示されるショートカットメニューに対応しています。ショートカットメニューキーを持つキーボードを使用している場合は、[Edit]ウィンドウをアクティブにした状態でそのキーを押すことによってもショートカットメニューを表示できます。このメニューには前述の編集メニューコマンドが登録されており、このメニューからもそれらのコマンドが実行できます。



3.7.4 タグジャンプ機能

アセンブルで文法エラーが発生すると、そのエラーメッセージが[Output]ウィンドウに表示されます。このメッセージをダブルクリックすることで、[Edit]ウィンドウ内のエラーが発生したソース行にジャンプすることができます。
ただし、この機能が有効となるのは、エラーメッセージがソース行番号を含んでいる場合のみです。



3.7.5 印刷

[Edit]ウィンドウ内のテキストを印刷することができます。
[File]メニューに[Print...]、[Print Preview]および[Page Setup...]コマンドが用意されています。また、ツールバーの[Print]ボタンも使用可能です。これらの機能は、Windowsの標準的なアプリケーションと同様です。印刷するテキストの[Edit]ウィンドウをアクティブにして、コマンドを選択してください。

3.8 ビルド

[Build]メニューまたはビルドツールバーにより、アセンブラ、リンカ、デバッガ、HEXコンバータ、ディスアセンブラをワークベンチから実行することができます。
ワークベンチでは、ソースファイルから実行形式のオブジェクトファイルを生成するプロセスをビルドと呼んでいます。

各ツールの詳細については、それぞれの章を参照してください。

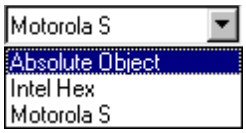
3.8.1 ビルドの準備

ビルドを実行する前に、必要なソースファイルを作成し、各ツールのオプションを設定しておきます。
1. プロジェクトを新規作成(3.6.1項参照)
2. ICEパラメータファイルを選択(3.6.1項参照)
3. ソースファイルの作成とプロジェクトへの登録(3.7項、3.6.2項参照)
4. ツールオプションの選択(3.9項参照)

3.8.2 実行形式オブジェクトのビルド

実行形式のオブジェクトファイルを作成するには、

1. プロジェクトファイルを開きます。
2. [Output Format]リストボックスから出力形式(アブソリュートオブジェクト、インテルHEXまたはモトローラS)を選択します。



3. [Build]メニューから[Build]を選択するか、[Build]ボタンをクリックします。

 [Build]ボタン

ワークベンチは、プロジェクト内のソースファイル構成とユーザが設定したツールオプションに従ってmakeファイルを作成します。このファイルは各ツールの実行を制御するために使用されます。
make処理は最初にアセンブラを起動し、各ソースファイルをアセンブルします。すでに最新のリロケータブルファイルが作成されている場合は、処理時間を短縮するため、そのソースはアセンブルされません。
次に、リンカを起動してアブソリュートオブジェクトファイルを作成します。この処理に使用するリンカコマンドファイルは、ワークベンチによって自動的に作成されます。出力形式にアブソリュートオブジェクトファイルを選択している場合は、この段階でビルドは完了します。インテルHEXまたはモトローラSを選択している場合は、さらにHEXコンバータを起動して、アブソリュートオブジェクトを指定形式のHEXファイルに変換します。

最新のリロケータブルオブジェクトファイルを含め、すべてのファイルを対象に再度ビルドを行う場合は、[Build]メニューの[Rebuild All]を選択するか、[Rebuild All]ボタンをクリックします。

 [Rebuild All]ボタン

ビルドの開始後に実行を中止するには、[Build]メニューから[Stop Build]を選択するか、[Stop Build]ボタンをクリックしてください。

 [Stop Build]ボタン

アセンブラのみを起動するには、アセンブルするソースの[Edit]ウィンドウをアクティブにしてから、[Build]メニューの[Assemble]を選択するか、[Assemble]ボタンをクリックします。

[Assemble]ボタン

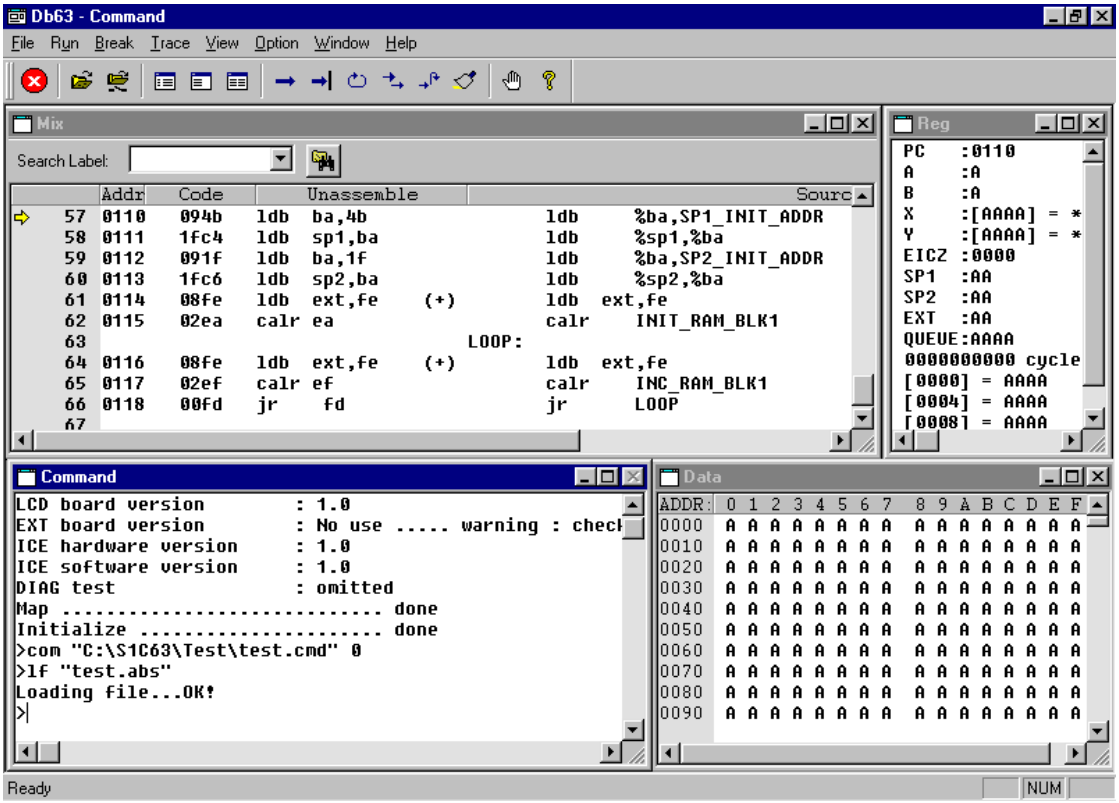
3.8.3 デバッグ

作成した実行形式のオブジェクトをデバッグするには、[Build]メニューから[Debug]を選択するか、[Debug]ボタンをクリックします。

[Debug]ボタン

デバッガが、指定されているICEパラメータファイルに従って起動し、ワークベンチが作成したコマンドファイルを実行して実行形式のオブジェクトファイルをロードします。
このコマンドファイルには指定形式のファイルをデバッガにロードするコマンドが記述されていますが、3.9項に示す[Settings]ダイアログボックスで内容を編集することもできます。

※ デバッガ起動後に再度ビルドを行った場合、デバッガウィンドウをアクティブにした時点でオブジェクトファイルを再度デバッガによってロードされます。



デバッガの操作方法については、"8 デバッガ"を参照してください。

3.8.4 その他のツールの実行

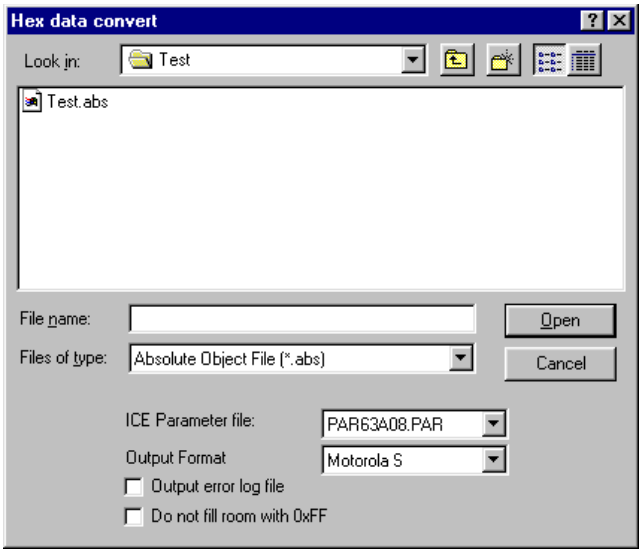
HEXコンバータとディスアセンブラは、ビルドとは独立して起動可能です。また、マスクデータ作成ツールもwb63から起動することができます。

●HEXコンバータ

HEXコンバータを起動するには、[Tools]メニューから[HEX Converter...]を選択するか、[HEX Convert]ボタンをクリックしてください。


[HEX Convert]ボタン

[Hex data convert]ダイアログボックスが表示されますので、変換するアブソリュートオブジェクトファイルを選択します。



ダイアログボックスで、HEXコンバータの以下のオプションが選択できます。

[ICE Parameter file:]リストボックス
　ICEパラメータファイルをプルダウンリストから選択します。

[Output Format:]リストボックス
　変換形式をインテルHEXとモトローラSから選択します。

[Output error log file]チェックボックス
　HEXコンバータのエラーファイルを作成する場合は、このオプションを選択します。

[Do not fill room with 0xFF]チェックボックス
　未使用領域への0xFF埋め込みを行わない場合は、このオプションを選択します。

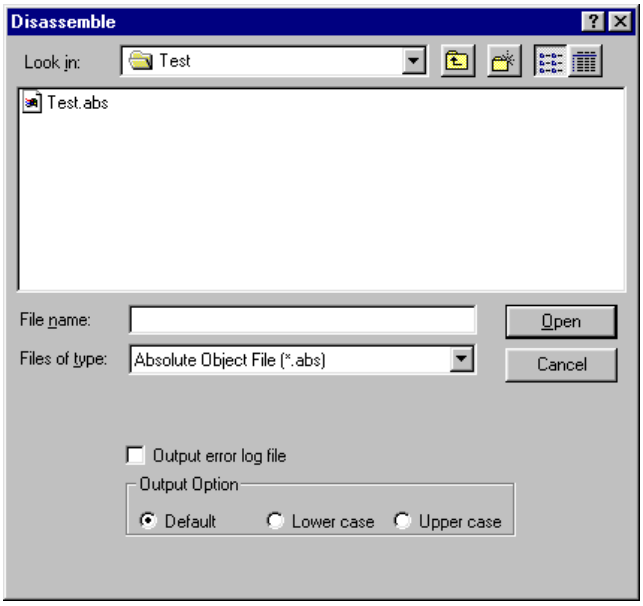
アブソリュートオブジェクトファイルとオプションを選択後、[Open]ボタンをクリックします。HEXコンバータが起動し、選択したアブソリュートオブジェクトファイルを指定形式のHEXファイルに変換します。HEXコンバータが出力するメッセージは、[Output]ウィンドウに表示されます。

●ディスアセンブラ

ディスアセンブラを起動するには、[Tools]メニューから[Disassembler...]を選択するか、[Disassemble]ボタンをクリックしてください。


[Disassemble]ボタン

[Disassemble]ダイアログボックスが表示されますので、逆アセンブルする実行形式ファイルを選択します。



ダイアログボックスで、ディスアセンブラの以下のオプションが選択できます。

[Output error log file]チェックボックス
　ディスアセンブラのエラーファイルを作成する場合は、このオプションを選択します。

[Output Option]
　文字ケースのオプションをラジオボタンで選択します。
　[Default]を選択すると、ディスアセンブラは逆アセンブルした内容のラベルには大文字を、命令には小文字を使用してソースを作成します。
　[Upper case]を選択すると、ソースはすべて大文字を使用して作成されます。
　[Lower case]を選択すると、ソースはすべて小文字を使用して作成されます。

実行形式オブジェクトファイルとオプションを選択後、[Open]ボタンをクリックします。ディスアセンブラが起動し、選択したファイルをソースファイルに変換します。ディスアセンブラが出力するメッセージは、[Output]ウィンドウに表示されます。

●ファンクションオプションジェネレータ, セグメントオプションジェネレータ, マスクデータチェッカ, On Board Writerコントロールソフトウェア

[Tools]メニューの選択により、以下のツールを起動することができます。

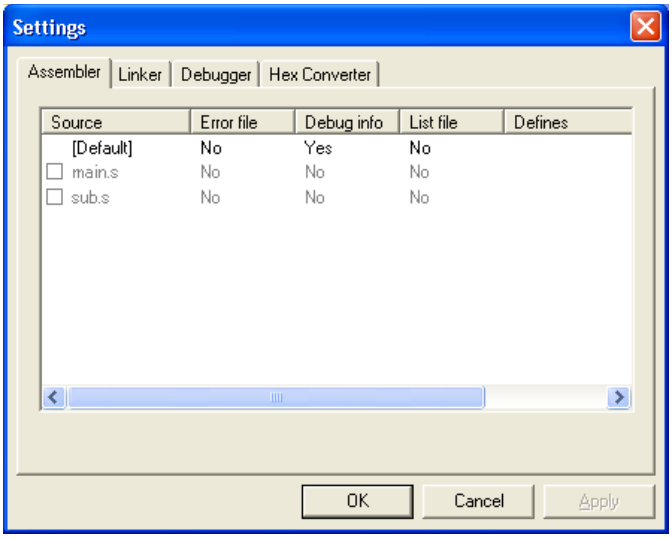
[WinFOG]	ファンクションオプションジェネレータwinfog（9章参照）
[WinSOG]	セグメントオプションジェネレータwinsog（10章参照）
[WinMDC]	マスクデータチェッカwinmdc（11章参照）
[OBPW63]	On Board Writerコントロールソフトウェアobpw63（テクニカルマニュアル参照）

各ツールの操作方法については、それぞれの章またはテクニカルマニュアルを参照してください。

注:　機種によっては、上記のツールが対応していない場合（機種情報定義ファイルが用意されていない場合）があります。それらの機種については、機種個別にツールが用意されていますが、[Tool]メニューから起動することはできません。

3.9 ツールオプションの設定

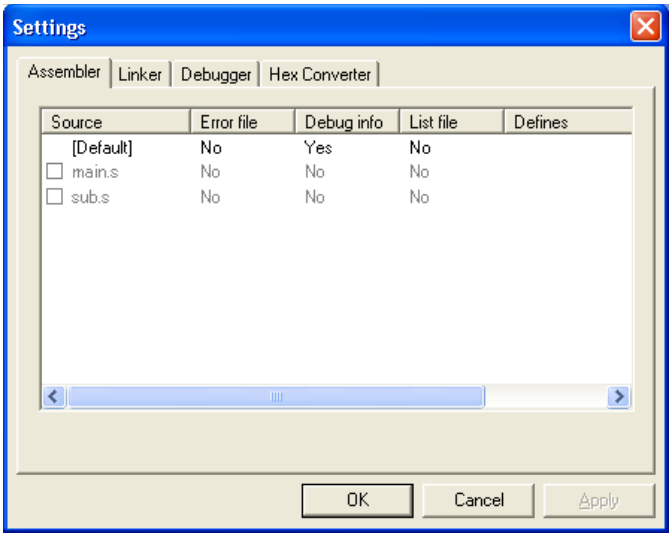
各ツールは起動時に指定可能なオプションを持っています。
ワークベンチでは、[Build]メニューから[Settings...]を選択すると表示されるダイアログボックス上で、それらのオプションが設定可能です。



各ツールのオプションは、ツール名のタブをクリックすると表示されます。

オプションを設定後、[OK]ボタンをクリックするとプロジェクト内のオプション設定情報が更新され、ダイアログボックスが閉じます。
他のツールのオプション設定を継続する場合は、[Apply]ボタンをクリックしてください。この操作では、ダイアログボックスは閉じられません。
設定を中止する場合は、[Cancel]ボタンをクリックします。

3.9.1 アセンブラオプション



このダイアログでは、以下の4つのアセンブラオプションが選択できます。

[Error file]　エラーファイルの出力(No: 出力しない、Yes: 出力)
[Debug info]　リロケータブルオブジェクトへのデバッグ情報の付加(No: 付加しない、Yes: 付加)
[List file]　リロケータブルリストファイルの出力(No: 出力しない、Yes: 出力)
[Defines]　条件アセンブル用の名前の定義(名前を入力)

編集ボックスはデフォルト設定([Default])とプロジェクト内のソースファイルを表示します。
デフォルト設定は、特に指定した以外のソースすべてに適用されます。ソース個別にオプションを選択する場合は、ソースファイル名の前にあるチェックボックスをチェックします。

ここをチェック →☐ sub.s　　No　　No　　No

[Error file]、[Debug info]、[List file]オプションは"No"または"Yes"が選択可能で、ダブルクリックにより切り替わります。たとえば、デフォルトの[List file]オプションを"No"から"Yes"に変更する場合、"No"の文字上をダブルクリックします。表示が"Yes"に切り替わります。

Source	Error file	Debug info	List file	Defines
[Default]	No	Yes	No	← ここをダブルクリックして"Yes"に変更

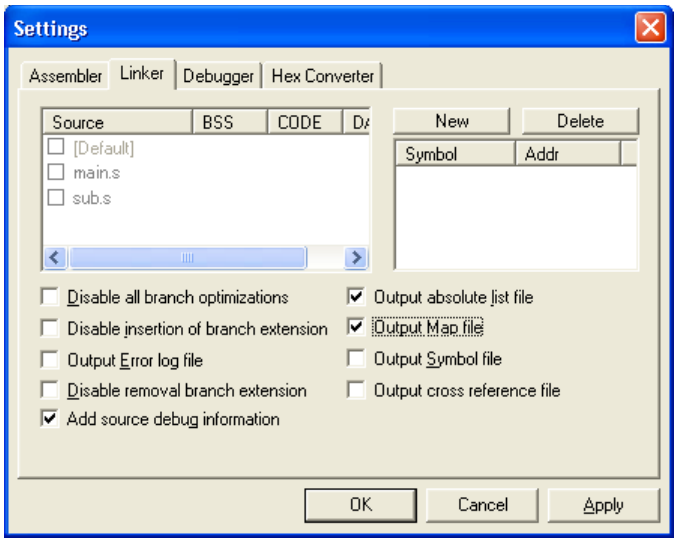
条件アセンブル用の名前を定義するには、[Defines]の部分をダブルクリックします。

Source	Error file	Debug info	List file	Defines
[Default]	No	Yes	No	← ここをダブルクリックして名前を入力

テキストボックスが現れますので、そこに名前を入力します。2個以上の名前を定義する場合は、それぞれをカンマ(,)で区切って入力してください。

アセンブラオプションの詳細については、"4 アセンブラ"を参照してください。

3.9.2 リンカオプション



このダイアログでは、セクションの配置、シンボル定義およびその他のリンカオプションを指定することができます。ワークベンチは、ここで指定されたオプションに従ってリンカコマンドファイルを作成し、リンカの起動に使用します。

●セクション配置の指定

このオプションのデフォルト設定では、すべてのリロケータブルセクションがメモリの先頭アドレスから順次配置されるようになっています。セクションの開始アドレスを指定するには、該当するセクションタイプの列をダブルクリックしてアドレスを入力します。

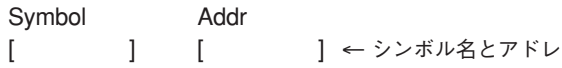
Source	BSS	CODE	DATA
☐ [Default]		← デフォルトCODEセクションの開始アドレスを変更するには、ここをクリックしてアドレスを入力	

Source	BSS	CODE	DATA
☑ [Default]		0x100	

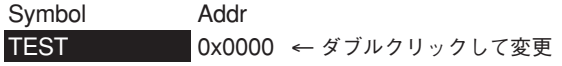
編集ボックスはデフォルト設定([Default])とプロジェクト内のソースファイルを表示します。
デフォルト設定は、特に指定したソース以外のすべてのセクションに適用されます。
ソース個別にオプションを選択する場合は、ソースファイル名の前にあるチェックボックスをチェックします。

●シンボル定義

シンボルを定義するには、[New]ボタンをクリックし、シンボル名とアドレスを編集ボックスに入力します。



入力されているシンボル名またはアドレスを変更するには、そのシンボル名またはアドレスをダブルクリックし、新しい名前またはアドレスを入力してください。



シンボルを削除するには、そのシンボルの行をクリックしてハイライト表示させ、[Delete]ボタンをクリックしてください。

●その他のオプション選択

[Disable all branch optimizations]チェックボックス
　分岐最適化機能のすべて(拡張命令の自動挿入/削除/修正)を使用しない場合に選択します。

[Disable insertion of branch extension]チェックボックス
　分岐最適化の中で拡張命令の挿入を禁止する場合に選択します。

[Output Error log file]チェックボックス
　リンカのエラーファイルを作成する場合に選択します。

[Disable removal branch optimization]チェックボックス
　分岐最適化の中で拡張命令の削除を禁止する場合に選択します。

[Add source debug information]チェックボックス
　デバッグ情報を含めたアブソリュートオブジェクトファイルを作成する場合に選択します。このオプションが選択されないと、デバッガでソースを表示することができません。

[Output absolute list file]チェックボックス
　アブソリュートリストファイルを作成する場合に選択します。

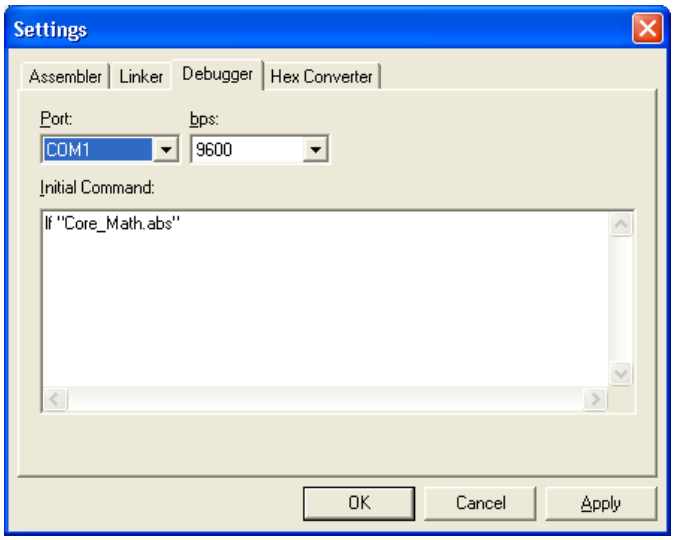
[Output Map file]チェックボックス
　リンクマップファイルを作成する場合に選択します。

[Output Symbol file]チェックボックス
　シンボルファイルを作成する場合に選択します。

[Output cross reference file]チェックボックス
　クロスリファレンスファイルを作成する場合に選択します。

リンカオプションの詳細については、"5 リンカ"を参照してください。

3.9.3 デバッガオプション



[COM Port:]リストボックス

ICEを接続するパーソナルコンピュータの設定を選択します。RS-232Cで接続する場合は、COMを選択します。USBで接続する場合は、USBを選択します。デフォルトではUSBに設定されています。

[bps:]リストボックス

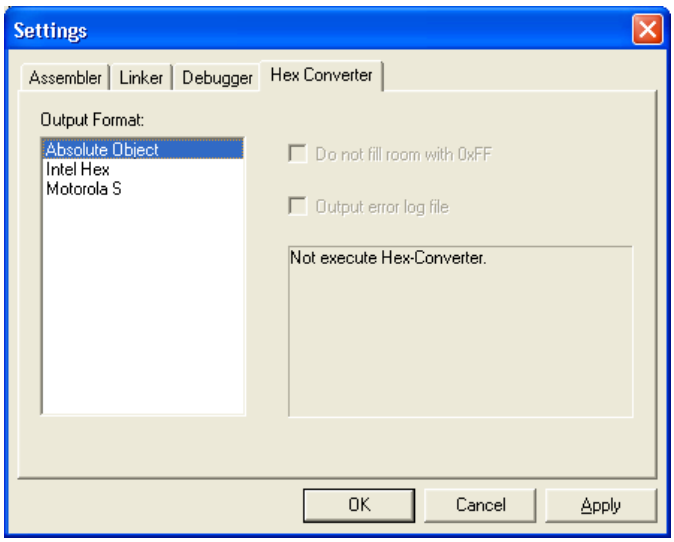
RS-232Cで接続する場合は、ICEと通信するためのボーレートを選択します。[COM Port:]リストボックスでUSBを選択した場合は、選択の必要はありません。

[Initial Command:]編集ボックス

デバッガ起動時に実行させるデバッグコマンドを編集します。ワークベンチはここに入力されたコマンドでコマンドファイルを作成し、デバッガ起動時に実行させます。デフォルトで、オブジェクトファイルをロードするコマンドが設定されています。

デバッガオプションの詳細については、"8 デバッガ"を参照してください。

3.9.4 HEXコンバータオプション



[Output Format:]リストボックス

ビルドで生成する実行形式オブジェクトの出力形式を選択します。"Absolute Object"を選択すると、ビルドはリンクが完了した時点で終了し、HEXコンバータは起動されません。"Intel Hex"または"Motorola S"を選択すると、リンク後にHEXコンバータを起動します。このどちらかのオプションを選択すると、HEXコンバータの他のオプションが選択可能になります。

[Do not fill room with 0xFF]チェックボックス

未使用領域への0xFF埋め込みを行わない場合に選択します。

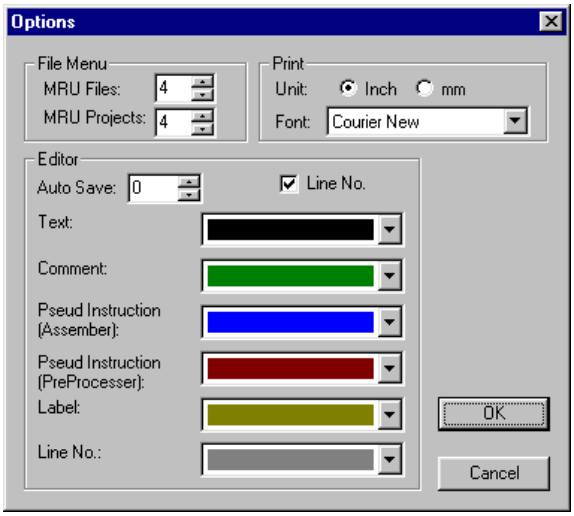
[Output error log file]チェックボックス

HEXコンバータのエラーファイルを作成する場合に選択します。

HEXコンバータオプションの詳細については、"6 HEXコンバータ"を参照してください。

3.10 ワークベンチオプション

[Tools]メニューから[Options...]を選択すると、いくつかのオプションを選択するダイアログボックスが表示され、ワークベンチをカスタマイズすることができます。



●[File]メニューオプション

[MRU Files:]ボックス

[File]メニューにリストする"最近開いたファイル"のファイル数を設定します。0～9個の範囲で選択できます。

[MRU Projects:]ボックス

[File]メニューにリストする"最近開いたプロジェクトファイル"のファイル数を設定します。0～9個の範囲で選択できます。

●印刷オプション

[Unit:]ラジオボタン

用紙の余白を指定する長さの単位をインチまたはミリメートルから選択します。この選択により、[Page Setup...]ダイアログボックスの余白設定の単位が変わります。

[Font:]リストボックス

[Edit]ウィンドウ内のテキスト印刷に使用するフォントを選択します。

●編集オプション

[Auto Save:]ボックス

[Edit]ウィンドウで編集中のドキュメントのテンポラリファイルを自動保存する間隔を、0～999秒の範囲で設定します。編集中のドキュメントはアセンブルやビルド時、またはワークベンチ終了時に自動保存します。0を選択した場合、テンポラリファイルの自動保存を行いません。

[Line No.]チェックボックス

[Edit]ウィンドウ内の行番号表示を行うか、非表示とするか選択します。チェックすると行番号が表示されます。

●カラー選択リストボックス

[Edit]ウィンドウのテキスト表示色を選択します。テキスト(ニーモニック)、コメント、アセンブル擬似命令、前処理擬似命令、ラベルおよび行番号が、ここで選択したそれぞれの色で表示されます。

3.11 ショートカットキー一覧

キー操作	機　能
Ctrl + N	ドキュメントの新規作成
Ctrl + O	ドキュメントのオープン
Ctrl + F12	ドキュメントのオープン
Ctrl + S	ファイルへのセーブ
Ctrl + P	アクティブなドキュメントの印刷
Ctrl + Shift + F12	アクティブなドキュメントの印刷
Ctrl + Z	直前の操作の取り消し
Alt + BackSpace	直前の操作の取り消し
Ctrl + X	選択範囲のカット
Shift + Delete	選択範囲のカット
Ctrl + C	選択範囲のコピー
Ctrl + Insert	選択範囲のコピー
Ctrl + V	クリップボードからのペースト
Shift + Insert	クリップボードからのペースト
Ctrl + A	ドキュメント内のテキストをすべて選択
Ctrl + F	文字列の検索
F3	次を検索
Shift + F3	前を検索
Ctrl + H	文字列の置き換え
Ctrl + G	ソース行、ラベルへのジャンプ
Ctrl + F7	ソースのアセンブル
F7	プロジェクトのビルド
Ctrl + Break	ビルドの中止
F5	デバッガの起動
Alt + F7	ツールオプションの設定
Ctrl + Tab	アクティブウィンドウの切り替え
Short-cut-key	ポップアップメニューの表示
Shift + F10	ポップアップメニューの表示

3.12 エラーメッセージ

ワークベンチのエラーメッセージを以下に示します。

エラーメッセージ	内　容
<filename> is changed by another editor. Reopen this file ?	現在開いているファイルが他のエディタで変更されました。
Cannot create file : <filename>	ファイル(リンカコマンドファイル、デバッグコマンドファイル等)が作成できません。
Cannot find file : <filename>	ソースファイルが見つかりません。
Cannot find ICE parameter file	ICEパラメータファイルが見つかりません。
Cannot open file : <filename>	ソースファイルが開けません。
You cannot close workspace while a build is in progress. Select the Stop Build command before closing.	ビルド中にプロジェクトを閉じるまたはワークベンチを終了するコマンドが指定されました。
Would you like to build it ?	ビルド前にデバッガを起動しようとしました。

3.13 注意事項

(1) ワークベンチで表示、編集可能なソースファイルの最大サイズは16Mバイトです。

(2) ワークベンチのラベル検索、およびラベルの色付け機能は、コロン(:)で終わっていないラベルには対応していません。

(3) ワークベンチはmakeファイル、リンカコマンドファイルおよびデバッグコマンドファイルを作成しますが、他のエディタで作成したこれらのファイルをワークベンチに入力することはできません。

(4) ワークベンチは日本語(2バイト文字)に対応していませんので、パスやファイル名に使用しないでください。

(5) ワークベンチを終了してプロジェクトを削除しようとした場合、「「使用中のフォルダ」別のプログラムがこのフォルダを開いているので操作を完了できません。フォルダを閉じてから再実行してください」というメッセージが出てプロジェクトフォルダが削除できないことがあります。
これは、プロジェクトをビルドした時、プロジェクトフォルダをカレントディレクトリとしてconim.exe(コマンドプロンプトで日本語入力を可能にする)が起動しビルド後も終了しないため、プロジェクトフォルダが削除できなくなります。
この場合には、タスクマネージャなどからconim.exeのプロセスを終了させるかPCを再起動させてから、プロジェクトフォルダを削除してください。

最新バージョンの制限事項やサポート機能、バグ情報についてはS5U1C63000Aのrel_workb_J.txtを参照してください。
最新バージョンのサポート機種はS5U1C63000AのReadme_J.txtを参照してください。

4 アセンブラ

この章ではアセンブラas63の機能と、アセンブリソースファイルの文法などについて解説します。

4.1 機能

アセンブラas63は本パッケージの中心となるツールで、アセンブリソースファイルをアセンブル(翻訳)し、機械語のオブジェクトファイルを生成します。アセンブラas63の機能、特長を以下に示します。

- 1つのソースに、アブソリュートセクションとリロケータブルセクションが混在可能
- リンカで再配置可能なリロケータブルオブジェクトが生成されるため、複数のソースによるモジュール別のプログラム開発が可能
- デバッガ上でソースレベルデバッグを行うためのデバッグ情報を出力可能
- 旧S1C63プリプロセッサとアセンブラと上位互換

基本的なアセンブル機能に加え、以下の付加機能も提供します。

- マクロ定義とマクロ呼び出し
- デファイン名の定義
- 数値演算子
- 他のファイルの挿入
- 条件アセンブル

アセンブラは、ソースファイルを前処理とアセンブルの2段階に処理します。前処理ステージでは、ソース内に記述された上記の付加機能部分をアセンブル可能なニーモニックに展開し、結果を中間ファイル(プリプロセスファイル)として出力します。アセンブルステージでは、プリプロセスファイルをアセンブルし、ソースコードを機械語に変換します。

4.2 入出力ファイル

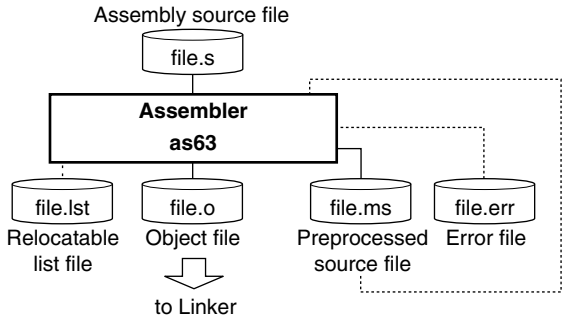


図4.2.1 フローチャート

4.2.1 入力ファイル

●アセンブリソースファイル

ファイル形式: テキストファイル

ファイル名: <ファイル名>.s
<ファイル名>.ms (アセンブラ、ディスアセンブラで生成したプリプロセスファイル)

内容: ソースプログラムを記述したファイルです。ファイルの拡張子を省略すると、アセンブラは指定のファイル名に".s"の拡張子を持つソースファイルが指定されたものとして処理します。

注意: ".s"の拡張子を持つソースファイルを指定した場合は、アセンブルの前に前処理が行われます。".ms"の拡張子を持つソースファイルを指定した場合は、アセンブルのみが行われます。したがって、".ms"ファイルに前処理用の擬似命令を記述しておくことはできません。

4.2.2 出力ファイル

●オブジェクトファイル

ファイル形式: IEEE-695リロケータブル形式のバイナリファイル
ファイル名: <ファイル名>.o (<ファイル名>は、-oで指定した場合を除き入力ファイル名と同じです。)
出力先: カレントディレクトリ
内容: リロケータブルなプログラムコード(機械語)が記録されたファイルで、リンカによって他のモジュールとリンクされ、実行形式のアブソリュートとなります。

●リロケータブルリストファイル

ファイル形式: テキストファイル
ファイル名: <ファイル名>.lst (<ファイル名>は、-oで指定した場合を除き入力ファイル名と同じです。)
出力先: カレントディレクトリ
内容: オフセットアドレス、オブジェクトコード、ソースコードが記録された、アセンブル結果を示すリストファイルです。

●プリプロセスファイル

ファイル形式: テキストファイル
ファイル名: <ファイル名>.ms (<ファイル名>は、-oで指定した場合を除き入力ファイル名と同じです。)
出力先: カレントディレクトリ
内容: 前処理用の命令(条件アセンブルやマクロ命令等)がアセンブル可能なニーモニックに展開されたファイルです。

●エラーファイル

ファイル形式: テキストファイル
ファイル名: <ファイル名>.err (<ファイル名>は、-oで指定した場合を除き入力ファイル名と同じです。)
出力先: カレントディレクトリ
内容: 起動時オプション(-e)が指定された場合に出力されるファイルで、エラーメッセージなど、アセンブラが標準出力(stdout)に対して出力する内容を記録します。

4.3 起動方法

●コマンドラインの一般形

as63 ∧ [起動時オプション] ∧ [<ソースファイル名>]

∧はスペースを示します。
[ ]は省略可能なことを示します。

●ソースファイル名

コマンドラインでは、一度に指定可能なアセンブリソースファイルは1つのみです。複数のファイルを処理するには、ファイルの回数分、アセンブラを実行してください。
Windowsの長いファイル名およびパスも指定可能です。ファイル名にスペースを挿入する場合は、ファイル名をダブルクォーテーションマーク(")で囲んで指定してください。

●起動時オプション

アセンブラには6種類の起動時オプションが用意されています。

-d <デファイン名>
機能: デファイン名の定義
説明: ソースの先頭に"#define <デファイン名>"を記述したのと同じ働きをします。条件アセンブルを起動時に制御するためのオプションです。
-dと<デファイン名>の間には1個以上のスペースが必要です。
2つ以上のデファイン名を定義する場合は、"-d <デファイン名>"の指定を繰り返してください。

-g
機能: デバッグ情報の付加
説明: シンボリック/ソースデバッグ情報を含む出力ファイルが生成されます。
シンボリック/ソースデバッグを行う場合は、必ず指定してください。
デフォルト: このオプションを省略すると、リロケータブルオブジェクトファイルにデバッグ情報は付加されません。

-o <ファイル名>
機能: 出力パス/ファイル名の指定
説明: 出力パス/ファイル名を拡張子なしで、あるいは拡張子".o"を付けて指定します。
拡張子の指定を省略した場合は、指定のパス/ファイル名に".o"が付加されます。
デフォルト: このオプションを省略すると、入力ファイル名を出力ファイルに使用します。

-c
機能: シンボルの大文字/小文字を無視
説明: 大文字と小文字のシンボルを区別したくない場合に指定します。
デフォルト: このオプションを省略すると、シンボルの大文字/小文字は区別されます。

-l
機能: リロケータブルリストファイルの出力
説明: リロケータブルリストファイルを出力します。
デフォルト: このオプションを省略すると、リロケータブルリストファイルは出力されません。

-e
機能: エラーファイルの出力
説明: アセンブラが標準出力デバイス(stdout)に出力するエラーメッセージなどの情報をエラーファイル(.err)に保存します。
デフォルト: このオプションを省略すると、エラーファイルは出力されません。

コマンドラインにオプションを入力する場合、オプションの前後には1個以上のスペースが必要です。オプションの指定順序およびソースファイル名との前後関係には、特に制限はありません。
例: `c:¥epson¥s1c63¥bin¥as63 -g -e -l -d TEST1 -d TEST2 test.s`

4.4 メッセージ

アセンブラはメッセージをすべて標準出力(stdout)に対して出力します。

●起動メッセージ

起動時は次のメッセージを出力します。

```
Assembler 63 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 1998-2001
```

●終了メッセージ

正常に終了した場合は、生成したファイル名を出力します。

```
Created preprocessed source file <FILENAME.MS>
Created relocatable object file <FILENAME.O>
Created relocatable list file <FILENAME.LST>
Created error log file <FILENAME.ERR>

Assembly 0 error(s) 0 warning(s)
```

●Usage出力

ファイル名が指定されなかったり、オプション指定が正しくない場合、次の使用方法のメッセージを出力して終了します。

```
Usage: as63 [options] <file name>
Options: -d <symbol>    Add preprocess definition
         -e             Output error log file (.ERR)
         -g             Add source debug information in object
         -l             Output relocatable list file (.LST)
         -c             Ignore character case of symbols
         -o <file name> Specify output file name
File name: Source file name (.S or .MS)
```

●エラー・ワーニング発生時

エラーが発生した場合は、終了メッセージの前にエラーメッセージが表示されます。

例:
```
TEST.S(5) Error: Illegal syntax
  Assembly 1 error(s) 0 warning(s)
```

エラーの場合、アセンブラは出力ファイルを作成せずに終了します。前処理ステージでエラーが発生した場合、アセンブラは処理を中止し前処理のエラーメッセージのみを出力します。

ワーニングが発生した場合は、終了メッセージの前にワーニングメッセージが表示されます。

例:
```
TEST.S(6) Warning: Expression out of range
  Assembly 0 error(s) 1 warning(s)
```

ワーニングの場合、アセンブラは出力ファイルを作成して終了します。

エラーおよびワーニングメッセージの先頭には、コマンドラインで指定したソースファイル名が出力されます。
エラーとワーニングの詳細は4.10項「エラー/ワーニングメッセージ」を参照してください。

4.5 アセンブリソースの文法

アセンブリソースファイルは汎用のエディタ、あるいはワークベンチのソースエディタで作成してください。ソースは標準のテキストファイルとしてセーブします。ファイル名には、Windowsの長いファイル名も指定できます。

以下、アセンブリソースファイルを作成する際の規則、文法について解説します。

4.5.1 ステートメント

アセンブリソースの個々の命令や定義をステートメントと呼びます。基本的なステートメントの構成は次のとおりです。

構文パターン

(1) ニーモニック　　オペランド　(;コメント)
(2) アセンブラ擬似命令　パラメータ　(;コメント)
(3) ラベル:　　(;コメント)
(4) ;コメント

例:

```
ステートメント                                    構文パターン
#include   "define.h"                                (2)
           .set  IO1, 0xfff1                        (2)

; TEXT SECTION (ROM, 13bit width)                   (4)

           .org  0x100                              (2)
NMI:                                                (3)
           reti                                     (1)
           nop                                      (1)
           nop                                      (1)
           jr    NMI                                (1)

           .org  0x110                              (2)
BOOT:                                               (3)
           ld    %f,0x4                             (1)
           ld    %a,0                               (1)
           ld    %a,0                               (1)
           ldb   %ext,0     ; clear memory 0 to 3   (1)
               :                        :
```

上記の例が一般的なソース記述方法です。視認性を高めるため、各ステートメントを構成する要素はタブやスペースで位置を揃えています。

●制限事項

- 各行には1つのステートメントのみ記述できます。1行に2つ以上の命令を記述するとエラーとなります。ただし、コメントやラベルは命令の行にも記述することができます。

  例: ;OK
  ```
  BOOT:     ld   %f,0x4

  ;Error
  BOOT:     ld   %f,0x4          ld   %a,0x0
  ```

- 1つのステートメントを複数行に分けて記述することはできません。1行で完結していないステートメントはエラーとなります。

  例:
  ```
  .word     0x0,0x1,0x2,0x3      ...OK
  .word     0xa,0xb,0xc,0xd      ...OK

  .word     0x0,0x1,0x2,0x3
            0xa,0xb,0xc,0xd      ...Error
  ```

- 1行に記述可能な文字数はスペースも含めて最大259文字(ASCIIキャラクタ)です。これを越えるとエラーとなります。

- コメントを除き、使用可能な文字はASCIIキャラクタ(英数記号)に限られます。また、使用できる記号にも制限があります(詳細は後述)。

- ニーモニックや擬似命令などの予約語は、大文字と小文字が区別されません。したがって、ニーモニックと擬似命令は大文字(A～Z)、小文字(a～z)のどちらで記述しても受け付けられます。たとえば、"ld"、"LD"、"Ld"はすべて"ld"命令として受け付けられます。シンボル等と区別するため、本マニュアルでは予約語にはすべて小文字を使用します。
  一方、-cオプションを省略した場合、ユーザが定義するラベルやシンボルは大文字と小文字が区別されます。

4.5.2 命令(ニーモニックと擬似命令)

アセンブラはS1C63000命令セットのすべてのニーモニックと、アセンブラ擬似命令に対応しています。
以下、命令の記述方法について説明します。

●ニーモニック

命令は「ニーモニック」+「オペランド」の構成となります。命令によっては、オペランドを持たないものもあります。

命令表記の一般形

一般形: <ニーモニック>
<ニーモニック> タブまたはスペース <オペランド>
<ニーモニック> タブまたはスペース <オペランド1>, <オペランド2>
<ニーモニック> タブまたはスペース <オペランド1>, <オペランド2>, <オペランド3>

例:
```
nop
jr     NMI
ld     %f,0x4
```

行の中で、ニーモニックの記述開始位置に制限はありません。ニーモニックの前にあるタブやスペースは無視されます。

オペランドを持つ命令の場合、ニーモニックとオペランド間は1個以上のタブまたはスペースで区切る必要があります。また、オペランドが複数の場合は、オペランド間を1個のカンマ(,)で区切ります。オペランド間のスペースは無視されます。

オペランドの要素については後述します。

ニーモニックの種類

S1C63 Familyで使用できるニーモニックは次の39種類です。

```
add adc and bit calr calz clr cmp dec ex halt inc int jp jr jrc jrnc jrnz
jrz ld ldb nop or pop push ret retd reti rets rl rr sbc set sll slp srl
sub tst xor
```

命令の詳細については"S1C63000コアCPUマニュアル"を参照してください。

注意

アセンブラはS1C63 Familyの全機種に共通のため、すべての命令が受け付けられます。開発機種がサポートしていない命令やオペランドを記述してもアセンブラではエラーになりませんので注意してください。このチェックはリンカが行います。

●アセンブラ擬似命令

アセンブラ擬似命令は、実行コードに変換される命令ではなく、アセンブルを制御したりデータを設定する命令です。
他の命令と区別するため、アセンブラ擬似命令はすべてピリオド(.)またはシャープ(#)で始まります。

擬似命令表記の一般形

一般形: <擬似命令>
<擬似命令> タブまたはスペース <パラメータ>
<擬似命令> タブまたはスペース <パラメータ1> タブ、スペースまたはカンマ <パラメータ2> ...

例:
```
#define     SW1  1
.org        0x100
.comm       BUF  4
```

行の中で、擬似命令の記述開始位置に制限はありません。

パラメータを持つ命令の場合、擬似命令とパラメータ間は1個以上のタブまたはスペースで区切る必要があります。また、パラメータが複数の場合は、パラメータ間を適切なデリミタで区切ります。

擬似命令の種類
アセンブラには、次に示す25種類の擬似命令が用意されています。

```
#include #define #macro #endm #ifdef #ifndef #else #endif #defnum
.abs .align .org .code .data .bss .codeword .word .comm .lcomm
.global .set .list .nolist .stabs .stabn
```

擬似命令の詳細については、"4.7 アセンブラ擬似命令"を参照してください。

●制限事項

ニーモニックと擬似命令は、大文字と小文字が区別されません。したがって、大文字(A～Z)、小文字(a～z)のどちらで記述しても受け付けられます。たとえば、"ld"、"LD"、"Ld"はすべて"ld"命令として受け付けられます。ただし、オペランドやパラメータに使用するユーザ定義シンボルは大文字と小文字が区別されます。これらについては、定義した文字列と同じに記述する必要があります。-cオプションを指定してアセンブルする場合、すべてのシンボルで大文字/小文字は区別されません。

4.5.3 シンボル(ラベル)

シンボル(ラベル)はプログラム中の任意のアドレスを参照するための識別子です。プログラムの分岐先アドレスやデータメモリ内のアドレスを、定義されたシンボルを使って参照することができます。

●シンボルの定義

シンボルは次の3つのいずれかの方法によって16ビットの値として定義されます。

1. <シンボル>:
   例: `LABEL1:`　　...LABEL1は記述された位置のアドレスを示すラベル

   先頭のスペースやタブは無視されます。一般的には行の先頭から記述します。

2. .set擬似命令で定義
   例: `.set  ADDR1  0xff00`　　...ADDR1は絶対アドレス0xff00を示すシンボル

3. .commまたは.lcomm擬似命令で定義
   例: `.comm  BUF1  4`　　...BUF1はRAMアドレスを示すラベル

   .commおよび.lcomm擬似命令は、BSSセクション(RAMなどのデータメモリ)内のラベルのみ定義可能です。プログラムメモリアドレスは定義できません。

●シンボルの参照

定義されたシンボルはアドレスを示します。
実際のアドレス値はアブソリュートセクションを除き、リンク時に決定します。
例: `LABEL1:`

```
            :
        jr      LABEL1          ... LABEL1の位置へジャンプ

.set  IO_M   0xfff0
.org  0x0000
.bss
.comm COUNT1 1

.code
ldb   %ext,IO_M@h
ldb   %xl,IO_M@l        ... Xレジスタに0xfff0がロードされます。(@h、@lはシンボルマスク)
inc  [COUNT1]           ... inc [0x0000]とみなされます。
```

●スコープ

スコープはシンボル(ラベル)の参照範囲のことです。定義されたファイル内でのみ参照可能なものをローカルシンボル、他のファイルからも参照可能なものをグローバルシンボルと呼びます。
シンボル定義時のスコープはローカルです。グローバルシンボルとして使用するには、定義したファイルおよび参照するファイルで.global擬似命令を使用してグローバル宣言を行う必要があります。
ローカルシンボルの多重定義はアセンブラでエラーとなります。グローバルシンボルの多重定義はリンカでエラーとなります。

例: グローバルシンボルを定義するファイル(file1)

```
        .global    SYMBOL      ...シンボルを定義するファイルでグローバル宣言
SYMBOL:
            :
LABEL:                         ...ローカルシンボル
            :                         (このファイル内でのみ参照可能)
```

グローバルシンボルを参照するファイル(file2)

```
        .global    SYMBOL      ...他のソースファイルで定義されたシンボルのグローバル宣言
        call       SYMBOL      ...シンボルを外部参照
            :
LABEL:                         ...ローカルシンボル
            :                         (file1のLABELとは別のシンボルとして扱われます。)
```

アセンブラはこれらのシンボルをアドレス未定としてアセンブルし、その情報を出力するオブジェクトファイルに含めます。それらのアドレスはリンカの処理によって最終的に決定します。

※ .comm擬似命令でシンボルを定義すると、そのシンボルはグローバルシンボルとなります。したがって、定義したファイル内では.global擬似命令によるグローバル宣言は不要です。ただし、参照するファイルでは、参照前にグローバル宣言が必要です。

●シンボルマスク

シンボルマスクは、16ビットアドレスを示すシンボルから上位8ビットアドレスおよび下位8ビットアドレスを取得するためのものです。
次に示す5種類のシンボルマスクが使用可能です。

シンボルマスク	意 味
@lまたは@L	絶対アドレスの下位8ビットを取得
@hまたは@H	絶対アドレスの上位8ビットを取得
@rlまたは@RL	相対アドレスの下位8ビットを取得
@rhまたは@RH	相対アドレスの上位8ビットを取得
@xhまたは@XH	絶対アドレスの上位8ビットを符号を反転して取得(cmp命令と組み合わせるldb命令専用)

例:

```
ldb   %ext,ADDR@h
ldb   %xl,ADDR@l          ... "ld %x, ADDR(16ビット)"として働きます。

ldb   %ext,NUM@h
add   %x,NUM@l            ..."add %x, NUM(16ビット)"として働きます。

ldb   %ext,LABEL@rh
calr  LABEL@rl            ..."calr LABEL(16ビット)"として働きます。

ldb   %ext,DATA@xh
cmp   %x,DATA@l           ..."cmp %x, DATA(16ビット)"として働きます。

.set  IO_ADDR  0xff12
ldb   %ext,IO_ADDR@l
ld    %a,[%y]             ..."ld %a, [IO_ADDR]"として働きます。
```

●制限事項

- シンボルの文字数は最大259文字です(コロンは文字数に含みません)。これを越えた場合、エラーとなります。
- 使用できる文字は以下のものに限られます。
  A～Z a～z _ 0～9 ?
- シンボルは数字で始まることはできません。
  例:
```
;OK                      ;Error
FOO:                     1lable:
L1:                      L 1:
.set   IO   0xfff0       .set    #IO       0xfff0
.comm  BUF  4            .lcomm  1st_BUF   2
```
- シンボルはデフォルトで大文字と小文字が区別されます。シンボルを参照する場合は、定義したものとまったく同じシンボルを使用してください。
  例:
```
_Abcd:
        :
    jr    _ABCD              ...._Abcdにはジャンプしない
```
  ただし、-cオプションを指定するとシンボルの大文字/小文字は区別されなくなります。
- シンボルマスクは定義されたシンボルにのみ有効です。数値にシンボルマスクを使用した場合はエラーとなります。
- シンボルマスクを省略すると、その命令に有効な下位側のビットが使用されます。ただし、指定可能な範囲を超えるとエラーまたはワーニングとなります。
- シンボルおよびシンボルマスクは、4ビット即値には使用できません。

4.5.4 コメント

コメントは一連のルーチン、あるいは各ステートメントの意味を記述しておくもので、コード化の対象とはなりません。

●コメントの定義

セミコロン(;)で始まり、改行コード(LF)で終わる文字列がコメントとみなされます。
コメントにはASCIIキャラクタのほかに漢字などの非ASCIIキャラクタも使用可能です。
例:
```
;This line is a comment line.
LABEL:        ;This is the comment for LABEL.
  ld  %a,%b   ;This is the comment for the instruction on the left.
```

●制限事項

- コメントの長さはセミコロン(;)、コメント前後あるいは内部のスペース、復帰/改行コードを含めて259文字までに制限されます。
- コメントが複数行に渡る場合は、各行ごとにセミコロンで始める必要があります。
  例:
```
;These are
  comment lines.    ...2行目はコメントとはみなされません。エラーとなります。

;These are
; comment lines.    ...2行ともコメントとみなされます。
```

4.5.5 空白行

本アセンブラでは、復帰/改行コードのみの空白行も許されます。一連のルーチンの区切りなど、セミコロンでコメント行にする必要はありません。

4.5.6 レジスタ名

レジスタ名は大文字でも、小文字でも受け付けます(区別されません)。

表4.5.6.1　レジスタ名の表記

レジスタ		表　記
A	データレジスタA	%a, %A, aまたはA
B	データレジスタB	%b, %B, bまたはB
BA	BAレジスタペア	%ba, %BA, baまたはBA
X	インデックスレジスタX	%x, %X, xまたはX
XH	Xレジスタの上位8ビット	%xh, %XH, xhまたはXH
XL	Xレジスタの下位8ビット	%xl, %XL, xlまたはXL
Y	インデックスレジスタY	%y, %Y, yまたはY
YH	Yレジスタの上位8ビット	%yh, %YH, yhまたはYH
YL	Yレジスタの下位8ビット	%yl, %YL, ylまたはYL
F	フラグレジスタF	%f, %F, fまたはF
EXT	拡張レジスタEXT	%ext, %EXT, extまたはEXT
SP1	スタックポインタSP1	%sp1, %SP1, sp1またはSP1
SP2	スタックポインタSP2	%sp2, %SP2, sp2またはSP2

注:　"%"は省略可能です。これらのシンボルは予約語ですので、ユーザ定義のシンボル名として使用することはできません。

4.5.7 数値表記

本アセンブラは10進形式、16進形式、2進形式の3種類の数値表記に対応しています。

●10進形式の数値表記

0～9の数値のみで表記されたものは10進数とみなされます。負の数を指定する場合は、マイナス(-)符号を数値の前に付けてください。

例: `1　255　-3`

0～9、および符号(-)以外の文字は使用できません。

●16進形式の数値表記

16進数を指定する場合は数値の前に"0x"を付けてください。

例: `0x1a　0xff00`

"0x"の後ろには0～9、a～f、A～F以外の文字は使用できません。

●2進形式の数値表記

2進数を指定する場合は数値の前に"0b"を付けてください。

例: `0b1001　0b01001100`

"0b"の後ろには0と1以外の文字は使用できません。

●数値の指定範囲

即値データのサイズ(指定範囲)は各命令により異なります。
各即値データの指定可能範囲は次のとおりです。

表4.5.7.1　即値データの種類と指定可能範囲

記号	種　類	10進数	16進数	2進数
imm2	2ビット即値データ	0～3	0x0～0x3	0b0～0b11
imm4	4ビット即値データ	0～15	0x0～0xf	0b0～0b1111
imm6	ソフトウェア割り込みベクタアドレス	0～64	0x0～0x3f	0b0～0b111111
imm8	8ビット即値データ	0～255	0x0～0xff	0b0～0b11111111
n4	4ビットn進指定データ	1～16	0x1～0x10	0b0～0b10000
sign8	符号付き8ビット即値データ	-128～127	0x0～0xff	0b0～0b11111111
add6	6ビットアドレス	0～64	0x0～0x3f	0b0～0b111111

●その他の数値表記

以下の数値表記も使用可能です。

nnnnB: 2進数
nnnnO: 8進数
nnnnQ: 8進数
nnnnH: 16進数

前処理段階で、"nnnnB"(2進数)と"nnnnH"(16進数)は新しい形式("0bnnnn"、"0xnnnn")に変換され、"nnnnO"と"nnnnQ"(8進数)は16進数("0xnnnn")に変換されます。

●ASCII文字の16進数への変換

1文字あるいは2文字のASCII文字を、数値に変換せずに記述(''で囲む)することができます。数値演算子も使用可能です。記述した文字は、16進数のASCIIコードに変換されてリロケータブルオブジェクトファイルに出力されます。

例:
```
retd  '1'    →   (retd  0x31)
retd  '23'   →   (retd  0x3233)
retd  '4'+1  →   (retd  0x35)
```

注: 3文字以上は記述できません。また以下の文字は使用できません。
コントロールコード(0x0～0x1f) スペース @ [ ] ; ,

4.5.8 数値演算子

数値の指定、デファイン名(数値定義の場合のみ)の定義には、数値またはラベルを含む定義済みのシンボルと数値演算子を組み合わせた式を使用することができます。
式は内部的に、すべて符号付き16ビットとして処理され、結果は16進数として出力されます。

●演算子の種類

算術演算子		例
+	加算, プラス符号	+0xff, 1+2
-	減算, マイナス符号	-1+2, 0xff-0b111
*	乗算	0xf*5
/	除算	0x123/0x56
%	剰余	0x123%0x56 (%%もサポートしています。)
>>	右シフト	1>>2
<<	左シフト	0x113<<3
^H	上位8ビットを取得	0x1234^H
^L	下位8ビットを取得	0x1234^L
( )	カッコ	1+(1+2*5)

算術演算子は指定された項の算術演算結果を返します。

論理演算子		例
&	論理積	0b1101&0b111
|	論理和	0b123|0xff
^	排他的論理和	12^35
~	ビット反転	~0x1234

論理演算子は指定された項の論理演算結果を返します。

関係演算子		例
==	左項と右項が等しい場合に真	SW==0
!=	左項と右項が等しくない場合に真	SW!=0
<	左項が右項より小さい場合に真	ABC<5
<=	左項が右項以下の場合に真	ABC<=5
>	左項が右項より大きい場合に真	ABC>5
>=	左項が右項以上の場合に真	ABC>=5
&&	AND	ABC&&0xf
||	OR	ABC||0b1010

関係演算子は式が真の場合に1、偽の場合に0を返します。

●優先順位

演算子は、以下のような優先順位となっています。同じ優先順位の場合は、式の左より計算します。

1. ( )　　優先順位が高い
2. +(プラス符号), -(マイナス符号), ~　　↑
3. ^H, ^L
4. *, /, %(%%)
5. +(加算), -(減算)
6. <<, >>
7. ==, !=, <, <=, >, >=
8. &
9. ^
10. |
11. &&　　↓
12. ||　　優先順位が低い

●使用例

```
#defnum BLK_HEADER_SIZE  4
#defnum BLK_START  0x30+BLK_HEADER_SIZE*2
#defnum BLK_END    BLK_START+4*2

#macro  ADD_X   ADDR
        ldb     %ext,(ADDR*2)^H        ...マクロの使用が可能
        add     %x,(ADDR*2)^L
#endm

        ldb     %ext,BLK_START^H  ; %x=BLK_START
        ldb     %xl, BLK_START^L
        ld      [%x],0b11&0x110
        ldb     %ext, ~BLK_END^H  ; cmp %x, BLK_END
        cmp     %x,  BLK_END^L
        ADD_X   (0x1200+0x34)*2   ; %x+=0x1234*2
```

●注意事項

- -1から-32768（負の数値）は0xffffから0x8000として扱われます。
- 式はすべて、内部的に16ビットで処理されます。計算結果を4ビット即値として使用する際にはその範囲を越えないように注意してください。特に負（マイナス）の数値を扱う際には注意してください。

```
例: ld  %a,-2+1            ...ld a,0xffffとなりNG
    ld  %a,(-2+1)&0xf      ...ld a,0xfとなりOK
```

- 式は、符号付きで計算されます（C言語でいうsigned short）。
  16進数を使った >>、/、%の計算結果には注意してください。

```
例: .set NUM1 0xfffe/2     ...-2/2 = -1（0xffff）となります。
                              /と%は+32767から-32768の範囲でしか使用できません。
    .set NUM2 0xfffe>>1    ...-2>>1 = -1（0xffff）となります。
                              （0xfffe>>1）&0x7fffのようにマスクしてください。
```

- #define文ではその中で使用している式がそのまま展開されますので、数値を定義する場合は注意してください。

```
例: #define NUM1 1+1
    ld       %a,NUM1*2     ..."ld %a, 1+1*2" (=3)として展開されます。
    #define NUM2 (1+1)
    ld       %a,NUM2*2     ..."ld %a, (1+1)*2" (=4)として展開されます。
```

- 演算子と項の間には、スペース、タブ等を入れないでください。

4.5.9 ロケーションカウンタシンボル"$"

各ステートメントのアセンブル時、各命令のアドレスは16ビットのロケーションカウンタにセットされます。このカウンタ値を、ラベルと同様にシンボル"$"を使用して参照することができます。"$"は現在の位置を示しますので、相対ジャンプに使用可能です。ラベルと同様に、演算子を組み合わせた式に"$"を使用することもできます。

```
例: jr  $                  ...現在のアドレスにジャンプ（永久ループ）
    jr  $+2                ...2命令後のアドレスにジャンプ
    jr  $-10               ...10命令前のアドレスにジャンプ
    jr  $+16+(16*(BLK>16)) ...演算子と定義済みシンボルが使用可能
```

●注意事項

他のセクション内のアドレスを"$"を使用して相対的に参照する場合、参照先までの相対距離がソースの段階で確定している必要があります。リロケータブルセクションの場合、リンクが終了するまで絶対アドレスが確定しないため、目的のアドレスが参照できるとはかぎりません。

4.5.10 旧プリプロセッサ用最適化分岐命令

旧バージョンのS1C63プリプロセッサには、分岐用の拡張命令を最適化する最適化分岐命令が用意されていました。現在のバージョンではこの機能をリンカがサポートしているため、アセンブラは最適化分岐命令をすべて拡張命令のない通常の分岐命令に展開します。ラベルまでの相対距離は、展開内容には影響しません。

最適化分岐命令			展開後のニーモニック	
xjr	LABEL	→	jr	LABEL
xjrc	LABEL	→	jrc	LABEL
xjrnc	LABEL	→	jrnc	LABEL
xjrz	LABEL	→	jrz	LABEL
xjrnz	LABEL	→	jrnz	LABEL
xcalr	LABEL	→	calr	LABEL

4.6 セクション管理

4.6.1 セクション定義

S1C63 Familyのメモリ構成はプログラムを書き込んでおくコードROM、データRAMやI/Oメモリなどのデータメモリに分けられます。また機種によっては静的データを書き込んでおくデータROMを搭載している場合もあります。
セクションはコードの書かれた(あるいは配置される)領域のことで、各メモリに対応して次の3つに分けられます。

1. CODEセクション　　コードROM内に配置する領域です。
2. DATAセクション　　データROM内に配置する領域です。
3. BSSセクション　　RAM領域を表します。

ソースファイル内でこれらのセクションを指定するために、アセンブラas63には擬似命令が用意されています。

●CODEセクション

.code擬似命令はCODEセクションを定義します。この命令から、他のセクション定義命令までのステートメントはプログラムコードとみなされ、コードROMに配置されるように処理されます。ソースファイルはデフォルトでCODEセクションとみなされます。したがって、ファイルの先頭から他のセクション定義までの間はCODEセクションとして処理されます。この領域は12ビット/ワードのため、4ビットデータの定義は行えません。

●DATAセクション

.data擬似命令はDATAセクションを定義します。この命令から、他のセクション定義命令までのステートメントは4ビットデータとみなされ、データROMに配置されるように処理されます。したがって、この領域にはデータROMのアドレスを参照するためのシンボルと、4ビットデータ定義擬似命令(.word)、コメント以外は記述できません。
また、このセクションはデータROMを持つ機種にのみ適用されます。

●BSSセクション

.bss擬似命令はBSSセクションを定義します。この命令から、他のセクション定義命令までのステートメントは4ビットデータとみなされ、データメモリ(RAM)に配置されるように処理されます。したがって、この領域ではデータメモリのアドレスを参照するためのシンボル定義と領域確保のみが行えます。
.comm擬似命令および.lcomm擬似命令はBSSセクション内のシンボルとサイズを定義する命令です。BSSセクションは基本的にRAM領域ですが、表示メモリ、I/Oメモリなどのデータメモリ領域にも使用可能です。S1C63 Familyのような組み込み型マイコンでは、この領域へのコード定義は意味を持ちませんので、上記2命令とコメント以外は記述できません。

4.6.2 アブソリュートおよびリロケータブルセクション

本アセンブラはリロケータブルアセンブラで、リンクによってアブソリュートオブジェクトに変換する必要のあるリロケータブルオブジェクトを生成します。ただし、1つのソース内には記述方法によりリロケータブルセクションとアブソリュートセクションが混在可能になっています。.org擬似命令によって絶対アドレスが指定されているセクションがアブソリュートセクション、絶対アドレスが指定されていないセクションがリロケータブルセクションです。リロケータブルセクションの絶対アドレスはリンク時に決定します。

4.6.3 セクション定義例

```
            :
            CODE1(リロケータブルプログラム)
            :
.data
            :
            DATA1(リロケータブルデータの定義)
            :
.bss
            :
            BSS1(リロケータブルRAM領域の定義)
            :
.code
.org    0x0         ...この指定を省略すると、このCODEセクションはCODE1の直後のアドレスから始まります。
            :
            CODE2(アブソリュートプログラム)
            :
.bss
.org    0x0         ...この指定を省略すると、このBSSセクションはBSS1の直後のアドレスから始まります。
            :
            BSS2(アブソリュートRAM領域の定義)
            :
.code
            :
            CODE3(リロケータブルプログラム)
            :
.data
.org    0x8000      ...この指定を省略すると、このDATAセクションはDATA1の直後のアドレスから始まります。
            :
            DATA2(アブソリュートデータの定義)
            :
```

上記のセクション定義例では、アブソリュートセクションとリロケータブルセクションが1つのソースに混在しています。.org擬似命令で絶対アドレスを指定しているCODE2、BSS2、DATA2がアブソリュートセクションで、これらのセクションは指定のアドレスから配置されます。

他のセクションはリロケータブルセクションで、これらのアドレスはアセンブル時には確定しません。リンクによる再配置後に確定します。

●注意事項

あるセクション内に他のセクション用のステートメントが現れるとワーニングとなり、そのステートメントに該当する新しいセクションがそこから始まります。

例:
```
.code
   .comm  BUF 16     ...ワーニング: 新規のBSSセクションが始まります。

   .bss
   ld     %a,%b      ...ワーニング: 新規のCODEセクションが始まります。
```

4.7 アセンブラ擬似命令

アセンブラ擬似命令は、実行コードに変換される命令ではなく、アセンブルを制御したりデータを設定する命令です。
他の命令と区別するため、アセンブラ擬似命令はシャープ(#)またはピリオド(.)で始まります。"#"で始まる命令は前処理用の擬似命令で、それらは前処理においてアセンブル可能な形に展開されます。展開結果はプリプロセスファイル(.ms)として出力されます。元の#擬似命令のステートメントは";"が付加され、コメントとして出力されます。"."で始まる擬似命令は、セクションやデータ定義などを行う擬似命令で、前処理では変換されません。
擬似命令自体は大文字と小文字は区別されません。
本アセンブラがサポートする擬似命令は以下のとおりです。

擬似命令	機能
#include	ファイルの挿入
#define	文字列の定義
#defnum	数値の定義(*1)
#macro~#endm	マクロ定義
#ifdef~#else~#endif	条件アセンブル
#ifndef~#else~#endif	条件アセンブル
.abs	逆アセンブルの指定(*1)
.align	セクションのアライメント
.org	絶対アドレスの設定
.code	CODEセクション(ROMへの配置)を宣言
.data	DATAセクション(ROMへの配置)を宣言
.bss	BSSセクション(RAMへの配置)を宣言
.codeword	CODEセクションへのデータ定義
.word	DTAセクションへのデータ定義
.comm	BSSセクション内にグローバル領域を確保
.lcomm	BSSセクション内にローカル領域を確保
.global	外部参照シンボルの定義
.set	シンボルに絶対アドレスを定義
.list	リロケータブルリスト出力を制御
.nolist	リロケータブルリスト出力を制御
.stabs	デバッグ情報（ソース名）
.stabn	デバッグ情報（ソース行番号）

*1: 旧アセンブラとの互換性をとるために残されています。

4.7.1 インクルード命令(#include)

インクルード命令は、ソースファイルの任意の場所に他のファイルの内容を挿入します。同じソースを複数のソースファイルで共通に使用する場合などに有効です。

●命令の形式

#include "<ファイル名>"

- ファイル名にはドライブ名、パス名も指定できます。
- 命令と"<ファイル名>"の間には1個以上のスペースまたはタブが必要です。
- #include命令と<ファイル名>は、大文字と小文字は区別されません。

記述例:
```
#include   "sample.def"
#include   "c:¥EPSON¥S1C63¥header¥common.h"
```

●展開規則

指定したファイルは#includeが記述された位置に挿入されます。

●注意事項

- 挿入可能なファイル形式はテキストファイルのみです。
- インクルードするファイル内でも#include命令を使用することができます。ただし、ネスティングは10レベルまでで、これを越えるとエラーとなります。

4.7.2 デファイン命令(#define)

デファイン命令(#define)によって任意の置換文字列をデファイン名として定義しておくことができ、その定義内容をプログラムの各所からデファイン名で参照することができます。

●命令の形式

#define <デファイン名> [<置換文字列>]

デファイン名:
- 1文字目はa～z、A～Z、?、_に限られます。
- 2文字目以降はa～z、A～Z、0～9、?、_が使用可能です。
- 大文字と小文字は区別されます。(#define自体は区別されません。)
- -cオプションを指定してアセンブルする場合、すべてのシンボルで大文字/小文字は区別されません。
- 命令とデファイン名の間には1個以上のスペースまたはタブが必要です。

置換文字列:
- すべての文字が使用可能です。ただし、セミコロン(;)はコメントの始まりと見なされます。
- 大文字と小文字は区別されます。
- デファイン名と置換文字列の間には1個以上のスペースまたはタブが必要です。
- 置換文字列は省略可能です。その場合は置換文字列としてNULLが定義されます。条件アセンブル命令に利用できます。

定義例:

```
#define   TYPE1
#define   L1        LABEL_01
#define   Xreg      %x

#define   CONST     (DATA1+DATA2)*2
```

●展開規則

定義されたデファイン名がソース内に現れると、プリプロセッサはそのデファイン名を定義された文字列に置き換えます。

展開例:

```
#define INT_F1     0xfff0
#define INT_F1_1   0
          :
        set   [INT_F1], INT_F1_1     ..."[0xfff0],0"に展開されます。
          :
```

●注意事項

- アセンブラはデファイン名の後方参照のみを許可しています。定義は参照する前に行われている必要があります。
- 一度定義したデファイン名を取り消すことはできません。ただし、定義されたデファイン名を使って別のデファイン名を定義することができます。

```
例: #define    XL    %xl
    #define    Xlow XL
               ldb   [Xlow],%ba        ..."ldb  [%xl],%ba"に展開されます。
```

- デファイン名を重複して定義した場合は、ワーニングメッセージが表示されます。再定義される前までは前の内容で展開され、再定義後は新たな内容で展開されます。プログラムの動作上は問題ありませんが、デファイン定義はそれぞれに異なる名称で行ってください。
- [ ]、[ ]+で囲む場合を除き、ソース中のデファイン名の前後にはデリミタ(スペース、タブ、改行やカンマ)を除く文字を付加することはできません。ただし、演算子またはシンボルマスク(@..)はデリミタなしに記述可能です。

```
例: #define    INT_F 0xfff
               tst    [INT_F1],0 ;tst [0xfff1],0?      ...このような指定は行えません。

    #define    L      LABEL
               ldb    %ext,L@h         ..."ldb %ext, LABEL @h"に置き換えられます。
               ldb    %xl,L@l          ..."ldb %xl, LABEL @l"に置き換えられます。
```

- #define文ではその中で使用している式がそのまま展開されますので、数値を定義する場合は注意してください。

```
例: #define    NUM1   1+1
               ld     %a,NUM1*2        ..."ld  %a, 1+1*2"に展開されます(=3)。

    #define    NUM2   (1+1)
               ld     %a,NUM2*2        ..."ld  %a, (1+1)*2"に展開されます(=4)。
```

- アセンブラの前処理では文字列を置き換えたことによるステートメントの妥当性はチェックしません。

4.7.3 数値デファイン命令(#defnum)

●命令の形式

#defnum　<数値デファイン名> <数値>

●機能

#defnum擬似命令は、旧アセンブラとの互換性をとるために用意されています。旧アセンブラでは数値定数の定義に#defnum、文字の定義に#define擬似命令を使用する必要がありました。新しいアセンブラには数値と文字の区別がなく、どちらも#define擬似命令で定義できます。

4.7.4 マクロ命令(#macro ... #endm)

マクロ命令(#macro)によって任意のステートメント列をマクロとして定義しておくことができ、その定義内容をプログラムの各所からマクロ名で呼び出すことができます。サブルーチンとは異なり、マクロを呼び出している箇所は定義内容と置き換えられます。

●命令の形式

```
#macro    <マクロ名>    [<仮パラメータ>] [,<仮パラメータ>] ...
          <ステートメント列>
#endm
```

マクロ名:
- 1文字目はa～z、A～Z、?、_に限られます。
- 2文字目以降はa～z、A～Z、0～9、?、_が使用可能です。
- 大文字と小文字は区別されます。(#macro自体は区別されません。)
- -cオプションを指定してアセンブルする場合、すべてのシンボルで大文字/小文字は区別されません。
- 命令とマクロ名の間には1個以上のスペースまたはタブが必要です。

仮パラメータ:
- マクロ定義用の仮パラメータシンボルで、定義するマクロにパラメータが必要な場合に記述します。
- マクロ名と最初の仮パラメータの間には1個以上のスペースまたはタブが必要です。また、複数のパラメータを記述する場合、それぞれの間にカンマ(,)が必要です。
- マクロ名と同じシンボルも使用可能です。
- パラメータの数はメモリの空き容量に依存します。

ステートメント列:
- 以下のステートメントが記述できます。
  - -基本命令(ニーモニック&オペランド)
  - -条件アセンブル命令
  - -分岐用内部ラベル*
  - -コメント
- 以下のステートメントは記述できません。
  - -アセンブラ擬似命令(条件アセンブル命令を除く)
  - -分岐用内部ラベル以外のラベル
  - -マクロ呼び出し

***分岐用内部ラベル** マクロはソース内の複数箇所に展開されます。このため、マクロ内にラベルを記述するとラベルの二重定義エラーとなります。そこで、マクロ内でのみ有効な分岐用内部ラベルを使用します。
- 分岐用内部ラベルの数はメモリの空き容量に依存します。
- マクロ名と同じシンボルも使用可能です。

定義例:

```
#define  C_RESET    0b1101
#macro   WAIT       COUNT
         ld         %a,COUNT
         and        %f,C_RESET
LOOP:
         nop
         jr         LOOP
#endm
```

●展開規則

定義されたマクロ名がソース内に現れると、アセンブラはその箇所に定義されたステートメント列を挿入します。
このとき実パラメータが記述されていれば、その順序に従って仮パラメータが実パラメータに置き換えられます。
分岐用内部ラベルはソースの先頭から現れる順序に従って、それぞれ__L0001～に置き換えられます。

展開例: 前記のマクロWAITが定義されている場合

マクロ呼び出し

```
          :
WAIT  15
          :
```

展開後

```
          :
;WAIT    15
ld    %a,15
and   %f,0b1101
__L0001:
nop
jr    __L0001
```

("__L0001"はソース内で初めて分岐用内部ラベルが展開された場合です。)

●注意事項

- アセンブラのマクロ呼び出しは後方参照のみを許可しています。定義は呼び出す前に行われている必要があります。
- 一度定義したマクロ名を取り消すことはできません。マクロ名を重複して定義した場合は、ワーニングメッセージが表示されます。再定義される前までは前の内容で展開され、再定義後は新たな内容で展開されます。プログラムの動作上は問題ありませんが、マクロ定義はそれぞれに異なる名称で行ってください。
- 定義するステートメント中の仮パラメータの前後にはデリミタ(スペース、タブ、改行やカンマ)を除く文字を付加することはできません。
- マクロ名にデファイン命令と同一の文字列を使用することはできません。
- 仮パラメータ数と実パラメータ数が異なる場合、エラーとなります。
- マクロで指定可能なパラメータ数と分岐用内部ラベル数はメモリの空き容量に依存します。
- 分岐用内部ラベルに使用される"__L####"は他のラベルやシンボルとしては使用しないでください。

4.7.5 条件アセンブル命令(#ifdef...#else...#endif, #ifndef...#else...#endif)

条件アセンブル命令は、指定する名前(デファイン名)の定義の有無によって指定範囲のアセンブルを行うかどうかを決定します。

●命令の形式

形式1)
```
#ifdef      <名前>
            <ステートメント列1>
[#else
            <ステートメント列2>]
#endif
```

名前が定義されていれば<ステートメント列1>をアセンブルの対象とします。
名前が定義されておらず、#else...<ステートメント列2>が記述されている場合は<ステートメント列2>をアセンブルの対象とします。#else...<ステートメント列2>は省略可能です。

形式2)
```
#ifndef     <名前>
            <ステートメント列1>
[#else
            <ステートメント列2>]
#endif
```

名前が定義されていなければ<ステートメント列1>をアセンブルの対象とします。
名前が定義されており、#else...<ステートメント列2>が記述されている場合は<ステートメント列2>をアセンブルの対象とします。#else...<ステートメント列2>は省略可能です。

名前:
- デファイン名の制限に従います。(#defineを参照)

ステートメント列:
- 条件アセンブル命令を除くすべてのステートメントが記述できます。

記述例:
```
#ifdef   TYPE1
         ld      %x,0x12
#else
         ld      %x,0x13
#endif

#ifndef  SMALL
#define  STACK1  0x31
#endif
```

●名前の定義

名前の定義は、条件アセンブル命令が実行される前に次の2つの方法のいずれかで行っておく必要があります。

(1) アセンブラの起動時オプション(-d)を使用して定義
例: `as63 -d TYPE1 sample.s`

(2) デファイン名定義命令(#define)を使用してソースファイル内に定義
例: `#define TYPE1`

#defineステートメントは、そのデファイン名を使用する条件アセンブル命令の前であればインクルードされるファイル内でも有効です。条件アセンブル命令より後に定義された名前は、未定義とみなされます。

条件アセンブルにのみ使用する場合は、置換文字列を指定する必要はありません。

●展開規則

アセンブル対象となったステートメント列は、他のアセンブラ擬似命令の展開規則に従って展開されます。（アセンブラ擬似命令を含んでいない場合は、そのままの状態でファイルに出力されます。）
アセンブル対象とならないステートメント列はすべてコメントとして出力されます。

●注意事項

条件に指定する名前は大文字と小文字を区別して評価されます。-cオプションを指定してアセンブルする場合、すべてのシンボルで大文字/小文字は区別されません。
文字ケースも含め同一のデファイン名が定義されている場合のみ条件成立とみなされます。

4.7.6 セクション定義擬似命令(.code, .data, .bss)

セクション定義擬似命令は、コードまたはデータを1つのまとまりとして定義し、リンク時にそれらを個々に再配置できるようにします。セクション定義擬似命令を使用しない場合でも、ステートメントによってセクションの種類は自動的に判断されます(ただし、ワーニングが発生)。現在のセクションの種類と異なるステートメントがソース内に現れると、そこから新しいセクションが始まります。

.code擬似命令

●命令の形式

.code

●機能

CODEセクションの開始を宣言します。本命令以降のステートメントは、他のセクションが宣言されるまで、コードROMに配置されるものとしてアセンブルされます。
CODEセクションがアセンブラのデフォルト設定です。したがって、ソースファイルの先頭では.code擬似命令を省略できます。他のセクションからCODEセクションに変更する場合に記述してください。

●注意事項

- CODEセクションはソースファイル中の複数箇所に分割して定義することができます(それぞれの開始位置に.code擬似命令を記述)。
- .org擬似命令で絶対アドレスが指定されるか、あるいは.align擬似命令でセクションのアライメントを指定した場合以外は、リロケータブルセクションとして定義されます。

.data擬似命令

●命令の形式

.data

●機能

DATAセクションの開始を宣言します。本命令以降のステートメントは、他のセクションが宣言されるまで、データROMに配置されるものとしてアセンブルされます。

●注意事項

- DATAセクションは静的データ領域で、データROMを搭載する機種にのみ有効です。
- DATAセクションには.word擬似命令、シンボル、コメント以外記述できません。記述した場合、エラーとなります。
- DATAセクションをソースファイル中の複数箇所に分割して定義することができます(それぞれの開始位置に.data擬似命令を記述)。ただし、アブソリュートアセンブルでは1つのソースファイル内に定義可能なすべてのセクションの合計は最大256個までです。
- .org擬似命令で絶対アドレスが指定されるか、あるいは.align擬似命令でセクションのアライメントを指定した場合以外は、リロケータブルセクションとして定義されます。

.bss擬似命令

●命令の形式

.bss

●機能

BSSセクションの開始を宣言します。本命令以降のステートメントは、他のセクションが宣言されるまで、RAMに配置されるものとしてアセンブルされます。

●注意事項

- BSSセクションには.comm、.lcommおよび.org擬似命令、シンボル、コメント以外記述できません。
- BSSセクションはソースファイル中の複数箇所に分割して定義することができます(それぞれの開始位置に.bss擬似命令を記述)。
- .org擬似命令で絶対アドレスが指定されるか、あるいは.align擬似命令でセクションのアライメントを指定した場合以外は、リロケータブルセクションとして定義されます。

4.7.7 絶対アドレス定義擬似命令(.org, .align)

絶対アドレス定義擬似命令は、セクションの開始位置を指定します。
.align擬似命令: セクションを2^nワード境界に配置
.org擬似命令: セクションを指定の絶対アドレスに配置

.org擬似命令

●命令の形式

.org <アドレス>

アドレス: 絶対アドレス指定
- 10進数、2進数、16進数の数値のみ記述可能です。
- 指定可能なアドレスは0から65,535(0xffff)までです。
- 命令とアドレスの間には1個以上のスペースまたはタブが必要です。

記述例:
```
.code
.org   0x0100
```

●機能

アセンブリソースファイル内でCODE、DATAまたはBSSセクションの絶対アドレスを指定します。.org擬似命令により始まるセクションはアブソリュートセクションとなります。

●注意事項

- .org擬似命令で既にコードが存在するアドレス、または確保されている領域内のアドレスを指定するとエラーとなります。

例:
```
.bss
.org    0x00
.comm   RAM0     4      ...RAMの領域確保(0x00～0x03)
.org    0x01
.comm   RAM1     4      ...エラー(0x01～0x03の領域がオーバーラップするため)
```

- あるセクション内に.org擬似命令が現れると、そこから新しいアブソリュートセクションを開始します。セクションの種類は直前までのものと同じです。.org擬似命令の有効範囲は、次のセクション定義擬似命令(.code、.dataまたは.bss)、または絶対アドレス定義擬似命令(.orgまたは.align)が現れるまでに限られます。

例:
```
      :
.code              ...前のリロケータブルセクション定義
      :
.org    0x100      ...アドレス0x100から新しいCODEセクションを開始します。
      :
.bss               ...このセクションはリロケータブルで、.org擬似命令の影響を受けません。
      :
.code              ...このセクションはリロケータブルで、.org擬似命令の影響を受けません。
      :
```

- .org擬似命令がセクション定義命令(.code、.dataまたは.bss)の直後に置かれている場合、そのセクション定義命令は新しいセクションを開始しません。.org擬似命令から、そのセクション擬似命令で指定される種類の新しいセクションが始まります。

例:
```
.code              ...新規セクションを開始しません。
.org    0x100      ...アブソリュートCODEセクションを開始します。
      :
```

- .org擬似命令がセクション定義命令(.code、.dataまたは.bss)の直前に置かれている場合、その.org擬似命令は無効となり新規セクションを開始しません。また、直前および直後のセクションにも影響を与えません。

例:
```
        :
.code               ...前のリロケータブルセクション定義
        :
.org    0x100       ...アブソリュートセクションを開始しません。
.bss                ...新しいリロケータブルBSSセクションを開始します。.org擬似命令の影響を受
        :             けません。
.code               ...このセクションはリロケータブルCODEセクションで、.org擬似命令の影響を
        :             受けません。
```

.align擬似命令

●命令の形式

.align　　<アライメント値>

アライメント値: 2^nで指定するワード境界
- 10進数、2進数、16進数の数値のみ記述可能です。
- アライメントは2^nの数値で指定します。
- 命令とアライメント値の間には1個以上のスペースまたはタブが必要です。

記述例:
```
.code
.align   32        ...直後の32ワード境界アドレスにアライメント
```

●機能

アセンブリソースファイル内でCODE、DATAまたはBSSセクションのワードアライメントを指定します。.align擬似命令により、そのセクションは指定ワード境界から始まる、完全にはアドレスが決定しないアブソリュートセクションとなります。

●注意事項

- あるセクション内に.align擬似命令が現れると、そこから新しいアブソリュートセクションを開始します。セクションの種類は直前までのものと同じです。.align擬似命令の有効範囲は、次のセクション定義擬似命令(.code、.dataまたは.bss)、または絶対アドレス定義擬似命令(.orgまたは.align)が現れるまでに限られます。

例:
```
        :
.code               ...前のリロケータブルセクション定義
        :
.align    32        ...32ワード境界から新しいCODEセクションを開始します。
        :
.bss                ...このセクションはリロケータブルで、.align擬似命令の影響を受けません。
        :
.code               ...このセクションはリロケータブルで、.align擬似命令の影響を受けません。
        :
```

- .align擬似命令がセクション定義命令(.code、.dataまたは.bss)の直後に置かれている場合、そのセクション定義命令は新しいセクションを開始しません。.align擬似命令から、そのセクション擬似命令で指定される種類の新しいセクションが始まります。

例:
```
.code               ...新規セクションを開始しません。
.align    32        ...アブソリュートCODEセクションを開始します。
        :
```

- .align擬似命令がセクション定義命令(.code、.dataまたは.bss)の直前に置かれている場合、その.align擬似命令は無効となり新規セクションを開始しません。また、直前および直後のセクションにも影響を与えません。

例:
```
        :
.code               ...前のリロケータブルセクション定義
        :
.align    32        ...アブソリュートセクションを開始しません。
.bss                ...新しいリロケータブルBSSセクションを開始します。.align擬似命令の影響を
        :             受けません。
.code               ...このセクションはリロケータブルCODEセクションで、.align擬似命令の影響
        :             を受けません。
```

4.7.8 アブソリュートアセンブル擬似命令(.abs)

●命令の形式

.abs

●機能

.abs擬似命令は、旧アセンブラとの互換性をとるために用意されています。旧アセンブラでは、この擬似命令がアブソリュートアセンブルの指定に必要でした。新しいアセンブラでは、1つのソースにアブソリュートセクションとリロケータブルセクションが混在可能で、アブソリュートセクションは.orgや.align擬似命令で指定可能です。したがって、この擬似命令は不要です。

4.7.9 シンボル定義擬似命令(.set)

●命令の形式

.set <シンボル>[,] <数値>

シンボル: 数値参照用のシンボル
- 1文字目はa～z、A～Z、?、_に限られます。
- 2文字目以降はa～z、A～Z、0～9、?、_が使用可能です。
- 大文字と小文字は区別されます。
- -cオプションを指定してアセンブルする場合、すべてのシンボルで大文字/小文字は区別されません。
- 命令とシンボルの間には1個以上のスペースまたはタブが必要です。

数値: 数値指定
- 10進数、2進数、16進数の数値のみ記述可能です。
- 文法上、指定可能なアドレスは0から65,535(0xffff)までです。
- シンボルとアドレスの間には1個以上のスペース、タブまたはカンマ(,)が必要です。

記述例:
```
.set   DATA1  0x20
.set   STACK1 0x100
```

●機能

定数参照用のシンボルを定義します。

●注意事項

定義されているシンボルをオペランドに使用すると、その数値がそのまま適用されます。したがって、数値がオペランドの有効範囲を超えている場合、ワーニングが発生します。

例:
```
.set  DATA1  0xff00
  ldb   %ext,DATA1@h     ...OK
  ldb   %x1,DATA1@1      ...OK
  ld    %a,DATA1         ...Warning
```

4.7.10 データ定義擬似命令(.codeword, .word)

.codeword擬似命令

●命令の形式

.codeword　　<データ>[,<データ> ... ,<データ>]

データ:　13ビットデータ
- 10進数、2進数、16進数の数値のみ記述可能です。
- 指定可能なデータは0から8,191(0x1fff)までです。
- 命令と最初のデータ間には1個以上のスペースまたはタブが必要です。
- データ間にはカンマ(,)が必要です。

記述例:
```
.code
.codeword  0xa,0xa40,0xff3
```

●機能
コードROMに書き込む13ビットデータを定義します。

●注意事項
.codeword擬似命令はCODEセクション内でのみ使用可能です。

.word擬似命令

●命令の形式

.word　　<データ>[,<データ> ... ,<データ>]

データ:　4ビットデータ
- 10進数、2進数、16進数の数値のみ記述可能です。
- 指定可能なデータは0から15(0xf)までです。
- 命令と最初のデータ間には1個以上のスペースまたはタブが必要です。
- データ間にはカンマ(,)が必要です。

記述例:
```
.data
.word    0xa,0xb,0xc,0xd
```

●機能
データROMに書き込む4ビットデータを定義します。

●注意事項
.word擬似命令はDATAセクション内でのみ使用可能です。

4.7.11 領域確保擬似命令(.comm, .lcomm)

●命令の形式

.comm　　<シンボル>[,]　　<サイズ>
.lcomm　　<シンボル>[,]　　<サイズ>

シンボル:　メモリアクセス(アドレス参照)用のシンボル
- 1文字目はa～z、A～Z、?、_に限られます。
- 2文字目以降はa～z、A～Z、0～9、_が使用可能です。
- 大文字と小文字は区別されます。
- -cオプションを指定してアセンブルする場合、すべてのシンボルで大文字/小文字は区別されません。
- 命令とシンボルの間には1個以上のスペースまたはタブが必要です。

サイズ:　確保する領域のワード数(4ビット/ワード)
- 10進数、2進数、16進数の数値のみ記述可能です。
- 文法上、指定可能なサイズは0から65,535までです。
- シンボルとサイズの間には1個以上のスペース、タブまたはカンマ(,)が必要です。

記述例:
```
.bss
.comm    RAM0  4
.lcomm   BUF,1
```

●機能

BSSセクション(RAM等のデータメモリ)内に指定サイズの領域を設定し、その先頭アドレスを示すシンボルを指定の名称で生成します。このシンボルを使用してRAMをアクセスする命令を記述することができます。

●.commと.lcommの相違点

.comm擬似命令と.lcomm擬似命令は機能的にまったく同じで、生成されたシンボルのスコープが異なります。.comm擬似命令で生成されたシンボルは他のモジュールから外部参照が可能なグローバルシンボルとなります(ただし、参照するファイルには.global擬似命令による指定が必要)。.lcomm擬似命令で生成されたシンボルはローカルシンボルで、他のモジュールからは参照することができません。

●注意事項

.comm/.lcomm擬似命令はBSSセクションにのみ記述可能です。

4.7.12 外部参照擬似命令(.global)

●命令の形式

.global <シンボル>

シンボル: 現在のファイルで定義するシンボル、または他のモジュールで定義済みのシンボル
• 命令とシンボル間には1個以上のスペースまたはタブが必要です。

記述例: `.global GENERAL_SUB1`

●機能

シンボルのグローバル宣言を行います。シンボルを定義したファイル内での宣言は、そのシンボルを他のモジュールから参照可能なグローバルシンボルに設定します。参照する側のファイルでは、グローバルシンボルを参照する前に本命令による宣言が必要です。

4.7.13 リスト制御擬似命令(.list, .nolist)

●命令の形式

.list
.nolist

●機能

リロケータブルリストファイルへの出力を制御します。
.nolist擬似命令は、それ以降のリロケータブルリストファイルへの出力を停止します。
.list擬似命令は.nolist擬似命令によって停止した出力をそこから再開します。

●注意事項

アセンブラは-lオプションを指定して起動した場合にのみ、リロケータブルリストファイルを出力します。したがって、-lオプションを指定しない場合、これらの命令は無効です。

4.7.14 ソースデバッグ情報擬似命令(.stabs, .stabn)

●命令の形式

(1).stabs "<ファイル名>", FileName
(2).stabn 0, FileEnd
(3).stabn <行番号>, LineInfo

●機能

アセンブラは、これらの命令に従ったソースデバッグ情報を含めてIEEE-695形式のオブジェクトファイルを出力します。このデバッグ情報はデバッガdb63によってアセンブリソースを表示してデバッグを行う場合に必要です。

形式(1)はファイルの開始位置情報を出力します。
形式(2)はファイルの終了位置情報を出力します。
形式(3)はソースファイル内における命令の行番号情報を出力します。

●デバッグ擬似命令の挿入

起動時オプションとして-gが指定されると、アセンブラは前処理でデバッグ擬似命令をプリプロセスファイルに挿入します。
したがって、ソースファイル作成時にこれらの擬似命令を記述する必要はありません。

4.7.15 コメント付加機能

前処理用の#で始まる擬似命令は、すべてアセンブル可能なコードに展開され、プリプロセスファイルに出力されます。その後でも最初に書かれていた命令が分かるように、それらの命令はセミコロン(;)で始まるコメントに置き換えられます。ただし、デファイン名の置き換えについてはコメントとして残りません。

コメントは展開後の最初の行の後ろに付加されます。元のステートメントにコメントが付いていた場合は、そのコメントも含めて付加します。
マクロ定義などは行の先頭にセミコロン(;)を挿入します。

例: **展開前**

```
#define    Areg       %a

#macro     ADDX2Y     VALUE
           ld         Areg, VALUE
           add        Areg, [%x]
           ld         [%y], Areg
#endm

           ADDX2Y     10h   ; MX + 10h -> MY
```

展開後(デバッグ情報なし)

```
;#define   Areg       %a

;#macro    ADDX2Y     VALUE
;          ld         Areg, VALUE
;          add        Areg, [%x]
;          ld         [%y], Areg
;#endm

           ;ADDX2Y    10h   ; MX + 10h -> MY
           ld         %a, 0x10
           add        %a, [%x]
           ld         [%y], %a
```

4.7.16 擬似命令の優先度

前処理用擬似命令の優先度に関連する事項を以下に示します。

1. 条件アセンブル命令(#ifdef、#ifndef)が最優先されます。
   条件アセンブル命令はネスティングができません。
2. 条件アセンブル命令の中に、デファイン命令(#define)、インクルード命令(#include)、マクロ命令(#macro)は記述できます。
3. マクロ定義の中にデファイン命令(#define)、インクルード命令(#include)、マクロ命令(#macro)は記述できません。
4. デファイン定義はマクロ定義に優先して展開されます。

4.8 リロケータブルリストファイル

リロケータブルリストファイルは、アセンブリソースファイルの各行の前半にアセンブル結果(オフセットアドレスやオブジェクトコード)を付加したファイルです。起動時オプション(-l)が指定された場合にのみ出力されます。
ファイル形式はテキストファイルで、ファイル名は"<ファイル名>.lst"(<ファイル名>は入力ソースファイルと同じ名称)です。

アセンブリリストファイルの各行の形式は次のとおりです。

行番号: アドレス コード ソースステートメント

例:

```
Assembler 63 ver x.xx Relocatable List File MAIN.LST Mon Jan 15 12:40:41 2001

         1:                ; main.s
         2:                ; AS63 test program (main routine)
         3:                ;
                                     :
        25:
        26:                         .org    0x110
        27:                BOOT:
        28:   0110   0900           ldb     %ba,SP1_INIT_ADDR
        29:   0111   1fc4           ldb     %sp1,%ba                ; set SP1
        30:   0112   0900           ldb     %ba,SP2_INIT_ADDR
        31:   0113   1fc6           ldb     %sp2,%ba                ; set SP2
        32:   0114   0200           calr    INIT_RAM_BLK1           ; initialize RAM block 1
        33:                LOOP:
        34:   0115   0200           calr    INC_RAM_BLK1            ; increment RAM block 1
        35:   0116   0000           jr      LOOP                    ; infinity loop
        36:
        37:
        38:                ;***** RAM block *****
        39:
        40:                         .org 0x0
        41:                         .bss
        42:   0000   00             .comm RAM_BLK0, 4
        43:   0004   00             .comm RAM_BLK1, 4
```

- 行番号の内容
  ソースファイルの先頭からの行番号が出力されます。

- アドレスの内容
  アブソリュートセクションの場合: 絶対アドレスが16進数で出力されます。
  リロケータブルセクションの場合: ファイルの先頭からの相対アドレスが16進数で出力されます。

- コードの内容
  CODEセクション: 命令(機械語)コードが16進数で出力されます。1つのアドレスに1つの命令が対応します。アドレスが未確定のシンボルを参照しているコードの即値部分は、アセンブル時にすべて0となります。これらの即値はすべてリンク時に決定します。
  DATAセクション: .word擬似命令で定義した4ビットデータが16進数で出力されます。1つのアドレスに1つのデータが対応します。
  BSSセクション: 確保した領域のサイズにかかわらず、すべて00が出力されます。BSSセクションのアドレスは、シンボルに定義されたアドレス(確保した領域の先頭アドレス)のみ出力されます。

4.9 実行例

●コマンドライン

```
C:¥EPSON¥S1C63¥bin¥as63 -g -e -l main.s
```

●アセンブリソースファイル

```
; main.s
; AS63 test program (main routine)
;

;***** INITIAL SP1 & SP2 ADDRESS DEFINITION *****

#ifdef SMALL_RAM
       .set SP1_INIT_ADDR 0xb              ;SP1 init addr = 0x2c
#else
       .set SP1_INIT_ADDR 0x4b             ;SP1 init addr = 0x12c
#endif

       .set SP2_INIT_ADDR 0x1f             ;SP2 init addr = 0x1f


;***** NMI & BOOT, LOOP *****

       .global INIT_RAM_BLK1               ; subroutine in sub.s
       .global INC_RAM_BLK1                ; subroutine in sub.s

       .org    0x100
NMI:
       calr    INIT_RAM_BLK1               ; initialize RAM block 1
       reti                                ; in NMI(watchdog timer)

       .org    0x110
BOOT:
       ldb     %ba,SP1_INIT_ADDR
       ldb     %sp1,%ba                    ; set SP1
       ldb     %ba,SP2_INIT_ADDR
       ldb     %sp2,%ba                    ; set SP2
       calr    INIT_RAM_BLK1               ; initialize RAM block 1
LOOP:
       calr    INC_RAM_BLK1                ; increment RAM block 1
       jr      LOOP                        ; infinity loop


;***** RAM block *****

       .org 0x0
       .bss
       .comm RAM_BLK0, 4
       .comm RAM_BLK1, 4
```

●プリプロセスファイル

```
.stabs "C:¥EPSON¥S1C63¥Test¥main.s", FileName
; main.s
; AS63 test program (main routine)
;

;***** INITIAL SP1 & SP2 ADDRESS DEFINITION *****

;#ifdef SMALL_RAM
;       .set SP1_INIT_ADDR 0xb              ;SP1 init addr = 0x2c
;#else
        .set SP1_INIT_ADDR 0x4b             ;SP1 init addr = 0x12c
;#endif

        .set SP2_INIT_ADDR 0x1f             ;SP2 init addr = 0x1f


;***** NMI & BOOT, LOOP *****

        .global INIT_RAM_BLK1               ; subroutine in sub.s
        .global INC_RAM_BLK1                ; subroutine in sub.s

        .org    0x100
NMI:
.stabn 23, LineInfo
        calr    INIT_RAM_BLK1               ; initialize RAM block 1
.stabn 24, LineInfo
        reti                                ; in NMI(watchdog timer)

        .org    0x110
BOOT:
.stabn 28, LineInfo
        ldb     %ba,SP1_INIT_ADDR
.stabn 29, LineInfo
        ldb     %sp1,%ba                    ; set SP1
.stabn 30, LineInfo
        ldb     %ba,SP2_INIT_ADDR
.stabn 31, LineInfo
        ldb     %sp2,%ba                    ; set SP2
.stabn 32, LineInfo
        calr    INIT_RAM_BLK1               ; initialize RAM block 1
LOOP:
.stabn 34, LineInfo
        calr    INC_RAM_BLK1                ; increment RAM block 1
.stabn 35, LineInfo
        jr      LOOP                        ; infinity loop


;***** RAM block *****

        .org    0x0
        .bss
        .comm    RAM_BLK0, 4
        .comm    RAM_BLK1, 4
.stabn 0, FileEnd
```

●アセンブリリストファイル

```
Assembler 63 ver x.xx Relocatable List File MAIN.LST Mon Jan 15 12:40:41 2001

          1:                ; main.s
          2:                ; ASM63 test program (main routine)
          3:                ;
          4:
          5:                ;***** INITIAL SP1 & SP2 ADDRESS DEFINITION *****
          6:
          7:                #ifdef SMALL_RAM
          8:                        .set SP1_INIT_ADDR 0xb      ;SP1 init addr = 0x2c
          9:                #else
         10:                        .set SP1_INIT_ADDR 0x4b     ;SP1 init addr = 0x12c
         11:                #endif
         12:
         13:                        .set SP2_INIT_ADDR 0x1f     ;SP2 init addr = 0x1f
         14:
         15:
         16:                ;***** NMI & BOOT, LOOP *****
         17:
         18:                        .global INIT_RAM_BLK1       ; subroutine in sub.s
         19:                        .global INC_RAM_BLK1        ; subroutine in sub.s
         20:
         21:                        .org    0x100
         22:                NMI:
         23:   0100   0200          calr    INIT_RAM_BLK1       ; initialize RAM block 1
         24:   0101   1ff9          reti                        ; in NMI(watchdog timer)
         25:
         26:                        .org    0x110
         27:                BOOT:
         28:   0110   0900          ldb     %ba,SP1_INIT_ADDR
         29:   0111   1fc4          ldb     %sp1,%ba            ; set SP1
         30:   0112   0900          ldb     %ba,SP2_INIT_ADDR
         31:   0113   1fc6          ldb     %sp2,%ba            ; set SP2
         32:   0114   0200          calr    INIT_RAM_BLK1       ; initialize RAM block 1
         33:                LOOP:
         34:   0115   0200          calr    INC_RAM_BLK1        ; increment RAM block 1
         35:   0116   0000          jr      LOOP                ; infinity loop
         36:
         37:
         38:                ;***** RAM block *****
         39:
         40:                        .org 0x0
         41:                        .bss
         42:   0000   00            .comm RAM_BLK0, 4
         43:   0004   00            .comm RAM_BLK1, 4
```

●エラーファイル

```
Assembler 63 Ver x.xx Error log file MAIN.ERR Mon Jan 15 12:40:41 2001

Assembler 63 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 1998-2001

Created preprocessed source file MAIN.MS
Created relocatable list file MAIN.LST
Created error log file MAIN.ERR
Created relocatable object file MAIN.O

Assembly 0 error(s) 0 warning(s)
```

4.10 エラー/ワーニングメッセージ

4.10.1 エラー

エラーが発生した場合、オブジェクトファイルは出力されません。
エラーメッセージは次の形式で出力/表示されます。

<ソースファイル名>(<行番号>) Error: <エラーメッセージ>

例: `TEST.S(431) Error: Illegal syntax`

* 一部のエラーメッセージでは、行番号が表示されません。

以下にアセンブラエラーメッセージの一覧を示します。

エラーメッセージ	説　明
Address out of range	アドレスが指定可能範囲外です。
Cannot read <file kind> file <FILE NAME>	指定ファイルが読み込めません。
Cannot read <file kind> file <FILE NAME>	指定ファイルが読み込めません。
Cannot write <file kind> file <FILE NAME>	指定ファイルに書き込めません。
Directory path length limit <directory path length limit> exceeded	パスの長さが制限を越えています。
Division by zero	0で除算が行われました。
File name length limit <file name length limit> exceeded	ファイル名の長さが制限を越えています。
Illegal macro label <label>	マクロ定義内の内部分岐ラベルの記述が不正です。
Illegal macro parameter <parameter>	マクロパラメータの記述が不正です。
Illegal syntax	不正な文法で記述されています。
Line length limit <line length limit> exceeded	1行の文字数が制限を越えています。
Macro parameter range <macro parameter range> exceeded	マクロパラメータ数が制限を越えています。
Memory mapping conflict	アドレスが重複しています。
Multiple statements on the same line	1行に2つ以上のステートメントが記述されています。
Nesting level limit <nesting level limit> exceeded	#includeのネストが制限を越えました。
Number of macro labels limit <number of macro label limit> exceeded	マクロ内の内部分岐ラベル数が制限を越えました。
Out of memory	メモリを確保できません。
Second definition of label <label>	ラベルが二重定義されました。
Second definition of symbol <symbol>	シンボルが二重定義されました。
Symbol name length limit <symbol name length limit> exceeded	シンボル名の長さが制限を越えています。
Token length limit <token length limit> exceeded	トークンの長さが制限を越えています。
Unexpected character <name>	無効な文字を使用しています。
Unknown label <label>	未定義のラベルを参照しています。
Unknown mnemonic <name>	無効な命令が記述されています。
Unknown symbol <name>	未定義のシンボルを参照しています。
Unknown register <name>	無効なレジスタ名が記述されています。
Unknown symbol mask <name>	無効なシンボルマスクが記述されています。
Unsupported directive <directive>	無効な擬似命令が記述されています。

4.10.2 ワーニング

ワーニングの場合、アセンブラは処理を継続します。他のエラーが発生しなければ、ワーニングメッセージを表示して処理を終了します。
ワーニングメッセージは次の形式で出力/表示されます。

<ソースファイル名>(<行番号>) Warning: <ワーニングメッセージ>

例: `TEST.S(41) Warning: Expression out of range`

以下にワーニングメッセージの一覧を示します。

ワーニングメッセージ	説　明
Expression out of range	式の計算結果が有効範囲を越えました。
Invalid symbol mask	シンボルマスクが正しく定義されていません。
Second definition of define symbol <symbol>	#defineでシンボルが二重定義されました。
Section activation expected, use <.code/.bss>	セクション定義がありません。
Missing Delimiter	オペランド間のカンマが省略されています。

4.11 注意事項

(1) #include擬似命令のネストは、最大10レベルに制限されています。これを越えるとエラーとなります。

(2) 個々のマクロ内で使用可能な内部分岐ラベル数は最大64個、1つのソースファイル内の内部分岐ラベル数の合計は最大9999個に制限されています。これを越えるとエラーとなります。

(3) セクション数など、他の制限はメモリの空き容量に依存します。

最新バージョンの制限事項やサポート機能、バグ情報についてはS5U1C63000Aのrel_asm_J.txtを参照してください。

5 リンカ

この章ではリンカlk63の機能を説明します。

5.1 機能

リンカlk63は実行可能なオブジェクトファイルを生成するソフトウェアで、以下の機能があります。

- 複数のオブジェクトモジュールをリンクし、1つの実行可能なオブジェクトファイルを生成
- モジュール間の外部参照を解決
- リロケータブルアドレスをアブソリュートアドレスに再配置
- リンク後のオブジェクトファイルに行番号やシンボル情報などのデバッグ情報を出力
- リンクマップファイル、シンボルファイル、アブソリュートリストファイル、クロスリファレンスファイルを出力可能
- 分岐命令の最適化機能（"ldb %ext, imm8"分岐拡張命令の自動挿入/削除/修正）

5.2 入出力ファイル

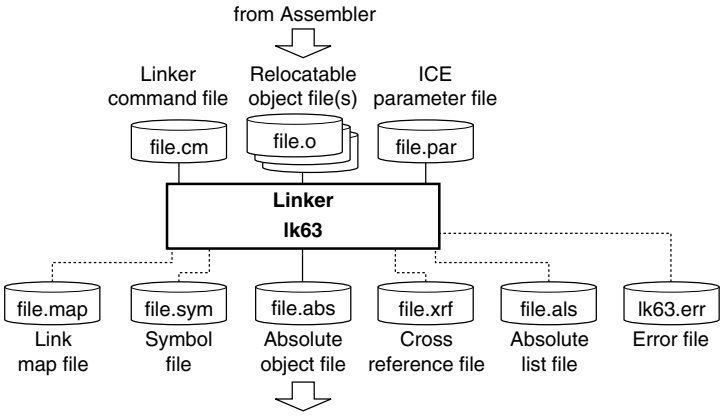


図5.2.1 フローチャート

5.2.1 入力ファイル

●リロケータブルオブジェクトファイル

＊このファイルは、コマンドラインまたはリンカコマンドファイル内のどちらかで必ず指定します。

ファイル形式: IEEE-695形式のバイナリファイル
ファイル名: <ファイル名>.o （パス指定も可能）
内容: アセンブラで生成したモジュール個々のオブジェクトファイルです。

●リンカコマンドファイル

ファイル形式: テキストファイル
ファイル名: <ファイル名>.cm （パス指定も可能）
内容: リンカのオプションを指定するファイルです。このファイルによりコマンドライン入力の手間を削減できます。
リンクは、このファイルを作成せずにコマンドラインのみでも処理可能です。

●ICEパラメータファイル

＊このファイルは、コマンドラインまたはリンカコマンドファイル内のどちらかで必ず指定します。

ファイル形式: バイナリファイル
ファイル名: <ファイル名>.par （パス指定も可能）
内容: S1C63 Family各機種の、メモリ構成や未使用命令などの情報が記録されています。このファイルは機種ごとに用意されています。デバッガ、HEXコンバータでも同じファイルを使用します。

5.2.2 出力ファイル

出力ファイル名は、コマンドラインまたはコマンドファイル内に-oオプションを入力することで指定できます。指定を省略した場合は、最初にリンクされるリロケータブルオブジェクトファイルの名称が使用されます。

●アブソリュートオブジェクトファイル

ファイル形式: IEEE-695形式のバイナリファイル
ファイル名: <ファイル名>.abs
出力先: カレントディレクトリ
内容: デバッガに入力可能な実行形式のオブジェクトファイルです。1つのプログラムを構成するすべてのモジュールがリンクされ、すべてのコードが配置される絶対アドレスが決定した内容となっています。デバッグに必要な情報もIEEE-695形式で含まれています。

●リンクマップファイル

ファイル形式: テキストファイル
ファイル名: <ファイル名>.map
出力先: カレントディレクトリ
内容: 各入力ファイルが各セクションのどのアドレスから配置されたかを示すマップ情報ファイルです。起動時オプションで-mが指定された場合に出力されます。

●シンボルファイル

ファイル形式: テキストファイル
ファイル名: <ファイル名>.sym
出力先: カレントディレクトリ
内容: すべてのモジュール内で定義されているシンボルとそのアドレス情報が出力されます。起動時オプションで-sが指定された場合に出力されます。

●クロスリファレンスファイル

ファイル形式: テキストファイル
ファイル名: <ファイル名>.xrf
出力先: カレントディレクトリ
内容: すべてのモジュール内のラベルの定義アドレスと参照アドレスが出力されます。起動時オプションで-xが指定された場合に出力されます。

●アブソリュートリストファイル

ファイル形式: テキストファイル
ファイル名: <ファイル名>.als
出力先: カレントディレクトリ
内容: コードの絶対アドレスが記録されることと、すべてのソースが1つのファイルとして参照できることを除き、アセンブラが出力するリロケータブルリストファイルと同様のリストファイルです。起動時オプションで-lが指定された場合に出力されます。

●エラーファイル

ファイル形式: テキストファイル
ファイル名: lk63.err
出力先: カレントディレクトリ
内容: 起動時オプション(-e)が指定された場合に出力されるファイルで、エラーメッセージなど、リンカが標準出力(stdout)に対して出力する内容を記録します。ファイル名はデフォルトで"lk63.err"ですが、-oオプションにより指定の名称に変わります。

5.3 起動方法

●コマンドラインの一般形

lk63 ^ [オプション] ^ [<リロケータブルオブジェクトファイル>] ^ [<リンカコマンドファイル>] ^ <ICEパラメータファイル>

^はスペースを示します。
[ ]は省略可能なことを示します。
オプションとファイル名の記述順序に制限はありません。

●ファイル名

各ファイルは拡張子によって識別されます。したがって、各ファイル名は拡張子を含めて指定してください。ただし、リロケータブルオブジェクトファイルのみは、拡張子(.o)を省略できます。

リロケータブルオブジェクトファイル: <File name.o>
リンカコマンドファイル: <File name.cm>
ICEパラメータファイル: <File name.par>

リンカコマンドファイルを使用すると、オプション、リロケータブルオブジェクトファイル名、ICEパラメータファイル名および出力ファイル名をすべてファイルに記述でき、コマンドラインではリンカコマンドファイルのみを指定するだけで済みます。すべての項目をコマンドラインから入力する場合、リンカコマンドファイルは必要ありません。

複数のリロケータブルオブジェクトをコマンドライン上で指定する場合は、各ファイル名を1個以上のスペースで区切ってください。

出力ファイル名を指定する場合は、アブソリュートオブジェクトファイル名(.abs)を指定します。その名称が他の出力ファイルにも適用されます。出力ファイル名を指定しない場合は、最初にリンクされるリロケータブルオブジェクトファイルの名称で、すべての出力ファイルが生成されます。

ICEパラメータファイル名は省略できません。

Windowsの長いファイル名およびパスも指定可能です。ファイル名にスペースを挿入する場合は、ファイル名をダブルクォーテーションマーク(")で囲んで指定してください。

●オプション

リンカには以下の起動時オプションが用意されています。

-d
機能: 分岐最適化の禁止
説明: 分岐命令(ジャンプとコール)に対する拡張命令(ldb %ext, imm8)の自動挿入/削除/修正を禁止します。
デフォルト: 本オプションを省略した場合、分岐最適化(拡張命令の自動挿入/削除/修正)を行います。

-di
機能: 分岐拡張命令挿入の禁止
説明: 分岐最適化(拡張命令の自動挿入/削除/修正)機能を有効にしている場合に、その中の自動挿入機能を無効にします。
デフォルト: 分岐最適化が許可されている場合にこのオプションを省略すると、拡張命令の自動挿入が行われます。

-dr
機能: 分岐拡張命令削除の禁止
説明: 分岐最適化(拡張命令の自動挿入/削除/修正)機能を有効にしている場合に、その中の自動削除機能を無効にします。既存の拡張命令を削除しないようにするには、この指定が必要です。
デフォルト: 分岐最適化が許可されている場合にこのオプションを省略すると、不要な拡張命令が削除されます。

-e
機能: エラーファイルの出力
説明: リンカが標準出力デバイス(stdout)に出力するエラーメッセージなどの情報をエラーファイル(.err)に保存します。
デフォルト: 本オプションを省略した場合、エラーファイルは作成されません。

-g
機能: デバッグ情報の付加
説明: デバッグ情報を含むアブソリュートオブジェクトファイルが生成されます。
ソース表示、シンボリックデバッグを行う場合は、必ず指定してください。
デフォルト: 本オプションを省略した場合、デバッグ情報は出力されません。

-l
機能: アブソリュートリストファイルの出力
説明: アブソリュートリストファイルを出力します。
デフォルト: オプションを省略した場合、アブソリュートリストファイルは出力されません。

-m
機能: リンクマップファイルの出力
説明: リンクマップファイルを出力します。
デフォルト: 本オプションを省略した場合、リンクマップファイルは出力されません。

-o <ファイル名>
機能: 出力パス/ファイル名の指定
説明: 出力パス/ファイル名を拡張子なしで、あるいは拡張子".abs"を付けて指定します。
拡張子の指定を省略した場合は、指定のパス/ファイル名に".abs"が付加されます。
デフォルト: 本オプションを省略すると、最初に入力されるリロケータブルオブジェクトファイル名を出力ファイルに使用します。

-s
機能: シンボルファイルの出力
説明: シンボルファイルを出力します。
デフォルト: 本オプションを省略した場合、シンボルファイルは出力されません。

-x
機能: クロスリファレンスファイルの出力
説明: クロスリファレンスファイルを出力します。
デフォルト: 本オプションを省略した場合、クロスリファレンスファイルは出力されません。

-code <アドレス>
機能: リロケータブルCODEセクション開始アドレスの設定
説明: リロケータブルなCODEセクションの開始アドレスを設定します。アブソリュートセクションは影響を受けません。
各モジュールのCODEセクションは、特に指定したものを除き、このアドレス以降に順次配置されます。
-codeと<アドレス>の間には1個以上のスペースまたはタブが必要です。
<アドレス>は16進数(0xnnnn)で指定してください。
デフォルト: 本オプションを省略した場合、CODEセクションはICEパラメータファイルで指定されるコードROMの開始アドレスから始まります。
指定例: `-code 0x100`

-data <アドレス>
機能: リロケータブルDATAセクション開始アドレスの設定
説明: リロケータブルなDATAセクションの開始アドレスを設定します。アブソリュートセクションは影響を受けません。
各モジュールのDATAセクションは、特に指定したものを除き、このアドレス以降に順次配置されます。
-dataと<アドレス>の間には1個以上のスペースまたはタブが必要です。
<アドレス>は16進数(0xnnnn)で指定してください。
デフォルト: 本オプションを省略した場合、DATAセクションはICEパラメータファイルで指定されるデータROMの開始アドレスから始まります。
指定例: `-data 0x8000`

-bss <アドレス>
機能: リロケータブルBSSセクション開始アドレスの設定
説明: リロケータブルなBSSセクションの開始アドレスを設定します。アブソリュートセクションは影響を受けません。
各モジュールのBSSセクションは、特に指定したものを除き、このアドレス以降に順次配置されます。
-bssと<アドレス>の間には1個以上のスペースまたはタブが必要です。
<アドレス>は16進数(0xnnnn)で指定してください。
デフォルト: 本オプションを省略した場合、BSSセクションはICEパラメータファイルで指定されるRAMの開始アドレスから始まります。
指定例: `-bss 0x000`

-rcode <ファイル名>=<アドレス>
機能: ファイル別CODEセクション開始アドレスの設定
説明: 指定モジュールのCODEセクションを配置するアドレスを設定します。割り込みベクタなど、特定のアドレスに固定するコードを持つモジュールを、このコマンドにより指定します。アブソリュートセクションは影響を受けません。
-rcodeと<ファイル名>の間には1個以上のスペースまたはタブが必要です。
<アドレス>は16進数(0xnnnn)で指定してください。
デフォルト: 本オプションを省略した場合、各モジュールのCODEセクションは-codeオプションで設定したアドレスから連続的に配置されます。
指定例: `-rcode test1.o = 0x0110`

-rdata <ファイル名>=<アドレス>
機能: ファイル別DATAセクション開始アドレスの設定
説明: 指定モジュールのDATAセクションを配置するアドレスを設定します。割り込みベクタなど、データROMの特定のアドレスに固定するデータを持つモジュールを、このコマンドにより指定します。アブソリュートセクションは影響を受けません。
-rdataと<ファイル名>の間には1個以上のスペースまたはタブが必要です。
<アドレス>は16進数(0xnnnn)で指定してください。
デフォルト: 本オプションを省略した場合、各モジュールのDATAセクションは-dataオプションで設定したアドレスから連続的に配置されます。
指定例: `-rdata test1.o = 0x8100`

-rbss <ファイル名>=<アドレス>
機能: ファイル別BSSセクション開始アドレスの設定
説明: 指定モジュールのBSSセクションを配置するアドレスを設定します。RAMの特定のアドレスに固定するシンボルを持つモジュールを、このコマンドにより指定します。アブソリュートセクションは影響を受けません
-rbssと<ファイル名>の間には1個以上のスペースまたはタブが必要です。
<アドレス>は16進数(0xnnnn)で指定してください。
デフォルト: 本コマンドが指定されない場合、各モジュールのBSSセクションは-bssオプションで設定したアドレスから連続的に配置されます。
指定例: `-rbss test1.o = 0x100`

-defsym <シンボル名>=<アドレス>
機能: グローバルシンボルのアドレス設定
説明: グローバルシンボルの絶対アドレスを、参照しているモジュールのために定義します。ここで設定するシンボルを、ソース内でグローバルシンボルとして定義しておくことはできません。
-defsymと<シンボル名>の間には1個以上のスペースまたはタブが必要です。
指定例: `-defsym BOOT = 0x100`

コマンドラインにオプションを入力する場合、オプションの前後には1個以上のスペースが必要です。
例:
```
c:¥epson¥s1c63¥bin¥lk63  -defsym INIT=0x200 test.cm par63xxx.par
c:¥epson¥s1c63¥bin¥lk63  -g  -e  -s  -m  test1.o test2.o -o test.abs par63xxx.par
```

5.4 メッセージ

リンカはメッセージをすべて標準出力(stdout)に対して出力します。

●起動メッセージ

起動時は次のメッセージを出力します。

```
Linker 63 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 1998-2001
```

●終了メッセージ

正常に終了した場合は、生成したファイル名を出力します。

```
Created absolute object file <FILENAME.ABS>
Created absolute list file <FILENAME.ALS>
Created map file <FILENAME.MAP>
Created symbol file <FILENAME.SYM>
Created cross reference file <FILENAME.XRF>
Created error log file <FILENAME.ERR>

Link 0 error(s) 0 warning(s)
```

●Usage出力

ファイル名が指定されなかったり、オプション指定が正しくない場合、次の使用方法のメッセージを出力して終了します。

```
Usage: lk63 [options] <file names>
Options: -d                          Disable full branch optimization
         -dr                         Disable removal branch optimization
         -e                          Output error log file (.ERR)
         -g                          Add source debug information
         -l                          Output absolute list file (.ALS)
         -m                          Output map file (.MAP)
         -o <file name>              Specify output file name
         -s                          Output symbol file (.SYM)
         -x                          Output cross reference file (.XRF)
         -code <address>             Specify CODE start address
         -data <address>             Specify DATA start address
         -bss <address>              Specify BSS start address
         -rcode <file name>=<address> Specify CODE start address of the file
         -rdata <file name>=<address> Specify DATA start address of the file
         -rbss <file name>=<address>  Specify BSS  start address of the file
         -defsym <symbol>=<address>   Define symbol address
File names: Relocatable object file (.O)
            Command parameter file (.CM)
            ICE parameter file (.PAR)
```

●エラー・ワーニング発生時

エラーが発生した場合は、終了メッセージの前にエラーメッセージが表示されます。

例:
```
Error: Cannot create absolute list file TEST.ABS
  Link 1 error(s) 0 warning(s)
```

エラーの場合、リンカは出力ファイルを作成せずに終了します。

ワーニングが発生した場合は、終了メッセージの前にワーニングメッセージが表示されます。

例:
```
Warning: No debug information in TEST.O
  Link 0 error(s) 1 warning(s)
```

ワーニングの場合、リンカは出力ファイルを作成して終了します。ただし、その動作は保証されません。

エラーとワーニングの詳細については"5.12 エラー/ワーニングメッセージ"を参照してください。

5.5 リンカコマンドファイル

リンカは起動時のコマンドライン指定を簡略化するため、必要な指定(オプションやファイル名)を記述したリンカコマンドファイル(.cm)を入力してリンク処理を行うことができます。

リンカコマンドファイル例

```
-e                          ; エラーファイルの作成
-g                          ; デバッグ情報の追加
-code 0x0100                ; CODEセクション開始アドレス
-rcode test2.o = 0x0110 ; test2.oのCODEセクション開始位置を固定
-data 0x8000                ; DATAセクション開始アドレス
-bss 0x00e0                 ; BSSセクション開始アドレス

-defsym IO = 0xFF00     ; グローバルシンボルの設定

-o test.abs                 ; 出力ファイル名の指定
test1.o                     ; 入力ファイル1の指定
test2.o                     ; 入力ファイル2の指定
```

リンカコマンドファイルは以下の規則に沿って作成してください。

●ファイルの形式

リンカコマンドファイルは、上記例に示したとおり一般的なテキスト形式です。
ファイル名の拡張子は".cm"としてください。

●コマンドの記述

オプションはすべてハイフン(-)で始まります。個々のコマンドは1個以上のスペース、タブ、あるいは改行で区切る必要があります。視認性の面から、個々のコマンドは行を変えて記述することを推奨します。

注: • アドレスを指定する数値は16進数形式(0xnnnn)で記述してください。10進数、2進数表記は受け付けません。

• 単一の設定のみ許されるコマンドが重複して指定された場合、最後に指定したコマンドが有効となります。

例:
```
-code 0x0000
-code 0x0100      ...-code 0x0100が有効
```

●入力ファイルの指定

リロケータブルオブジェクトファイル名は、リンカコマンドファイルの最後に記述してください。リンクによる配置は、特に指定した場合を除き、記述した順序で行われます。
ファイル名は拡張子(.o)まで記述してください。

●コメント

リンカコマンドファイルにはコメントを記述することができます。
ソースファイルと同様に、セミコロン(;)からその行の終わりまでの文字列がコメントとみなされます。

●空白行

空白文字と改行のみの空白行は無視されます。セミコロンでコメントにする必要はありません。

5.6 リンクマップファイル

リンクマップファイルは、各セクションごとのモジュールの配置情報を参照するためのファイルで、-mオプションが指定された場合に出力されます。
ファイル形式はテキストファイルで、ファイル名は"<ファイル名>.map"(<ファイル名>は出力オブジェクトファイルと同じ名称)です。

●リンクマップファイル例

```
Linker 63 ver x.xx Link map file TEST.MAP Mon Jan 15 12:40:41 2001

CODE section map of TEST.ABS
Index  Start   End     Size    Opt  Type File         SecNbr
    0: 0x0000  0x000d  0x000e   +0  Rel  SUB.S        1
    1: 0x000e  0x00ff  0x00f2  ---  ---  ------------ ---
    2: 0x0100  0x0102  0x0003   +1  Abs  MAIN.S       1
    3: 0x0103  0x010f  0x000d  ---  ---  ------------ ---
    4: 0x0110  0x0118  0x0009   +2  Abs  MAIN.S       2
    5: 0x0119  0x1fff  0x1ee7  ---  ---  ------------ ---
Total: 0x1a occupied, 0x1fe6 blank

BSS section map of TEST.ABS
Index  Start   End     Size    Type File         SecNbr
    0: 0x0000  0x0007  0x0008  Rel  MAIN.S       3
    1: 0x0008  0xf2bf  ------  ---  ------------ ---
    2: 0xf800  0xf8ff  ------  ---  ------------ ---
    3: 0xff00  0xffff  ------  ---  ------------ ---
Total: 0x8 occupied, 0xf4b8 blank
```

●リンクマップファイルの内容

Index セクションのインデックス番号を示します。

Start セクションの開始アドレスを示します。

End セクションの終了アドレスを示します。

Size セクションのサイズを示します。

Opt 最適化で挿入/削除された拡張命令数を示します。

Type セクションタイプを示します。
(Rel = リロケータブルセクション、Abs = アブソリュートセクション)

File リンクされたモジュールのファイル名を示します。

SecNbr セクション番号を示します。

Total マップされた領域と未使用領域のサイズを示します。

Size、Opt、Type、File、SecNbrの"---"はセクションが配置されていないことを示します。

5.7 シンボルファイル

シンボルファイルはすべてのモジュール内で定義されているシンボルとそのアドレス情報を参照するためのファイルで、-sオプションが指定された場合に出力されます。
ファイル形式はテキストファイルで、ファイル名は"<ファイル名>.sym"(<ファイル名>は出力オブジェクトファイルと同じ名称)です。

●シンボルファイル例

```
Linker 63 ver x.xx Symbol file TEST.SYM Mon Jan 15 12:40:41 2001

CODE section labels of TEST.ABS
Address Type   File          Symbol
0x0110  Local  "MAIN.O" .... BOOT
0x0007  Global "SUB.O" ..... INC_RAM_BLK1
0x0000  Global "SUB.O" ..... INIT_RAM_BLK1
0x0116  Local  "MAIN.O" .... LOOP
0x0100  Local  "MAIN.O" .... NMI

BSS section labels of TEST.ABS
Address Type   File          Symbol
0x0000  Global "MAIN.O" .... RAM_BLK0
0x0004  Global "MAIN.O" .... RAM_BLK1
```

●シンボルファイルの内容

Symbol　定義されているすべてのシンボルをアルファベット順に示します。

Address　シンボルに定義された絶対アドレスを示します。

Type　シンボルのスコープを示します。
(Global = グローバルシンボル、Local = ローカルシンボル)

File　シンボルが定義されているオブジェクトファイル名を示します。

5.8 アブソリュートリストファイル

リロケータブルリストファイルは、アセンブリソースファイルの各行の前半に絶対アドレスとオブジェクトコードを付加したファイルで、-lオプションが指定された場合にのみ出力されます。
ファイル形式はテキストファイルで、ファイル名は"<ファイル名>.als"(<ファイル名>は出力オブジェクトファイルと同じ名称)です。リロケータブルリストファイルが各アセンブリソースファイルごとのリストファイルであるのに対し、アブソリュートオブジェクトファイルはリンクされたすべてのオブジェクトコードと対応するすべてのソースが1つのファイルにまとめられます。

●アブソリュートリストファイル例

```
Linker 63 ver x.xx Absolute list file TEST.ALS Mon Jan 15 12:40:41 2001

         1:                     ; sub.s
         2:                     ; AS63 test program (subroutine)
         3:
         4:                             .global RAM_BLK1
         5:
         6:                     ;***** RAM block 1 initialize *****
         7:
         8:                             .global INIT_RAM_BLK1
         9:                     INIT_RAM_BLK1:
        10:   0000   0800               ldb     %ext,RAM_BLK1@h
        11:   0001   0a04               ldb     %xl,RAM_BLK1@l      ;set RAM_BLK1 address to x
        12:   0002   1e90               ld      [%x]+,0x0
         :      :     :                        :
        55:                             .org    0x110
        56:                     BOOT:
        57:   0110   094b               ldb     %ba,SP1_INIT_ADDR
        58:   0111   1fc4               ldb     %sp1,%ba            ; set SP1
        59:   0112   091f               ldb     %ba,SP2_INIT_ADDR
        60:   0113   1fc6               ldb     %sp2,%ba            ; set SP2
        61:   0114   08fe   (+)         ldb  ext,fe
        62:   0115   02ea               calr    INIT_RAM_BLK1       ; initialize RAM block 1
        63:                     LOOP:
        64:   0116   08fe   (+)         ldb  ext,fe
        65:   0117   02ef               calr    INC_RAM_BLK1        ; increment RAM block 1
        66:   0118   00fd               jr      LOOP                ; infinity loop
         :      :     :                        :
```

●アブソリュートリストファイルの内容

アセンブリリストファイルの各行の形式は次のとおりです。

行番号: 絶対アドレス　コード　ソースステートメント

行番号　ファイルの先頭からの行番号です。
アドレス　命令が配置された絶対アドレスです。
コード　各命令のオブジェクトコードです。
ソース　各コードに対応するソースコードです。

●分岐最適化(拡張命令の挿入/削除/修正)結果

分岐最適化の結果、オリジナルのソースの拡張命令(ldb %ext, imm8)が削除あるいは修正されたり、新しい拡張命令が追加されていることがあります。アブソリュートリストファイルでは、以下のシンボルを使用して、これらの結果が確認できるようにしています。

追加された拡張命令
コードの右に"(+)"が付きます。その行はオリジナルのソースにはありませんが、コードを逆アセンブルした"ldb %ext, imm8"命令がソース部に出力されます。

削除された拡張命令
オリジナルソースの拡張命令の左に"(-)"が付きます。コードは生成されていません。

オペランドを修正した拡張命令
オリジナルソースの拡張命令の左に"(*)"が付きます。

●アセンブラで前処理された命令

アセンブラで展開された命令(マクロ、ソースのインクルード)は、ソース部の先頭に"+"が付きます。

5.9 クロスリファレンスファイル

クロスリファレンスファイルには、すべてのモジュール内で定義されているラベルの情報と、そのラベルを参照しているアドレスが記録されます。-xオプションを指定した場合にのみ出力されます。
ファイル形式はテキストファイルで、ファイル名は"<ファイル名>.xrf"(<ファイル名>は出力オブジェクトファイルと同じ名称)です。

●クロスリファレンスファイル例

```
Linker 63 ver x.xx Cross reference file TEST.XRF Mon Jan 15 12:40:41 2001

Label "INIT_RAM_BLK1" at 0x0000  SUB.O  CODE, Global
   0x0101  MAIN.O  CODE
   0x0115  MAIN.O  CODE

Label "RAM_BLK0" at 0x000  MAIN.O  BSS, Global
   0x0101  MAIN.O  CODE
   0x0115  MAIN.O  CODE

Label "RAM_BLK1" at 0x004  MAIN.O  BSS, Global
   0x0000  SUB.O  CODE
   0x0001  SUB.O  CODE
   0x0007  SUB.O  CODE
   0x0008  SUB.O  CODE

Label "INC_RAM_BLK1" at 0x0007  "SUB.O"  CODE, Global
   0x0117  MAIN.O  CODE

Label "NMI" at 0x0100  MAIN.O  CODE, Local

Label "BOOT" at 0x0110  MAIN.O  CODE, Local

Label "LOOP" at 0x0116  MAIN.O  CODE, Local
   0x0118  MAIN.O  CODE
```

●クロスリファレンスファイルの内容

各ラベルの情報は次の形式で出力されます。

Label <ラベル情報>
　<アドレス>　<ファイル名>　<セクションタイプ>

ラベル情報	ラベル定義に関する以下の情報を示します。 • ラベル名 • 定義されたアドレス • ラベルが定義されているオブジェクトファイル • セクションタイプ • スコープ
アドレス	ラベルを参照しているアドレス
ファイル名	ラベルを参照しているオブジェクトファイル
セクションタイプ	ラベルを参照しているアドレスが含まれるセクションのタイプ

5.10 リンク

●リンク規則

リンク処理は以下の規則に従って行われます。

- アブソリュートセクションはアセンブル時に決定されている絶対アドレスのとおりに、リロケータブルモジュールより先に配置されます。アブソリュートセクションが使用可能なメモリ領域を越えた場合は、エラーとなります。
- オプション(-rcode、-rdata、-rbss)で開始アドレスが指定されている特定のファイルのセクションは、その指定アドレスから順次配置されます。その他のリロケータブルセクションは、リロケータブルCODE/DATA/BSSセクションの先頭(デフォルトまたは-code、-data、-bssで指定)から配置されます。
- 基本的にリロケータブルセクションは、-rcode、-rdataまたは-rbssで指定した場合を除き、処理される順序で連続的に配置されます。ただし、あるセクションを連続的に配置するとICEパラメータファイルで指定される未使用領域にかかる場合、リンカはそのセクションを別の領域に配置することを試みます。使用可能な領域が見つからない場合はエラーとなります。1つのセクションを2つ以上のブロックに分割して配置することはありません。
  これにより空いた領域には、後で配置可能なセクションがあれば、割り付けられます。

●リンクの制限

オブジェクトの全体サイズが使用可能なメモリサイズ内であっても、各セクションのサイズや指定したアドレスによっては、全セクションを配置できない場合がありますので注意してください。

●リンク実行例

"test1.o"、"test2.o"の2つのリロケータブルオブジェクトファイルを、次の条件でリンクする例を以下に示します。

開発機種のメモリ構成

コードROM: 0x0000～0x1fff
データROM: 0x8000～0x87ff
RAM: 0x0000～0x07ff
表示、I/Oメモリ: 0xf000～0xffff

リロケータブルオブジェクトファイル

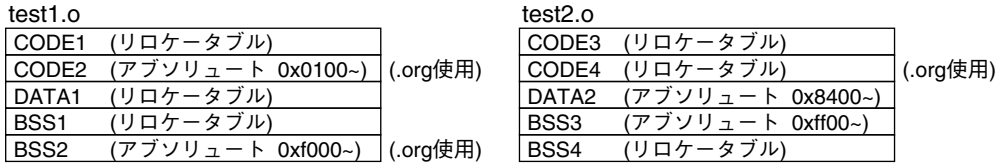


図5.10.1 リロケータブルオブジェクトファイルの構成(例)

リンカコマンドファイル例

```
-code 0x0000              ; リロケータブルCODEセクション開始アドレス
-rcode test2.o = 0x0110   ; test2.oのCODEセクション開始位置を固定
-data 0x8000              ; リロケータブルDATAセクション開始アドレス
-bss 0x0000               ; リロケータブルBSSセクション開始アドレス
-rbss test2.o = 0x0400    ; test2.oのBSSセクション開始位置を固定
-o test.abs               ; 出力ファイル名
test1.o                   ; 入力ファイル1
test2.o                   ; 入力ファイル2
```

上記のコマンドを定義してリンクを行うと、リンカは各モジュールのセクションを図5.10.2のように配置します。

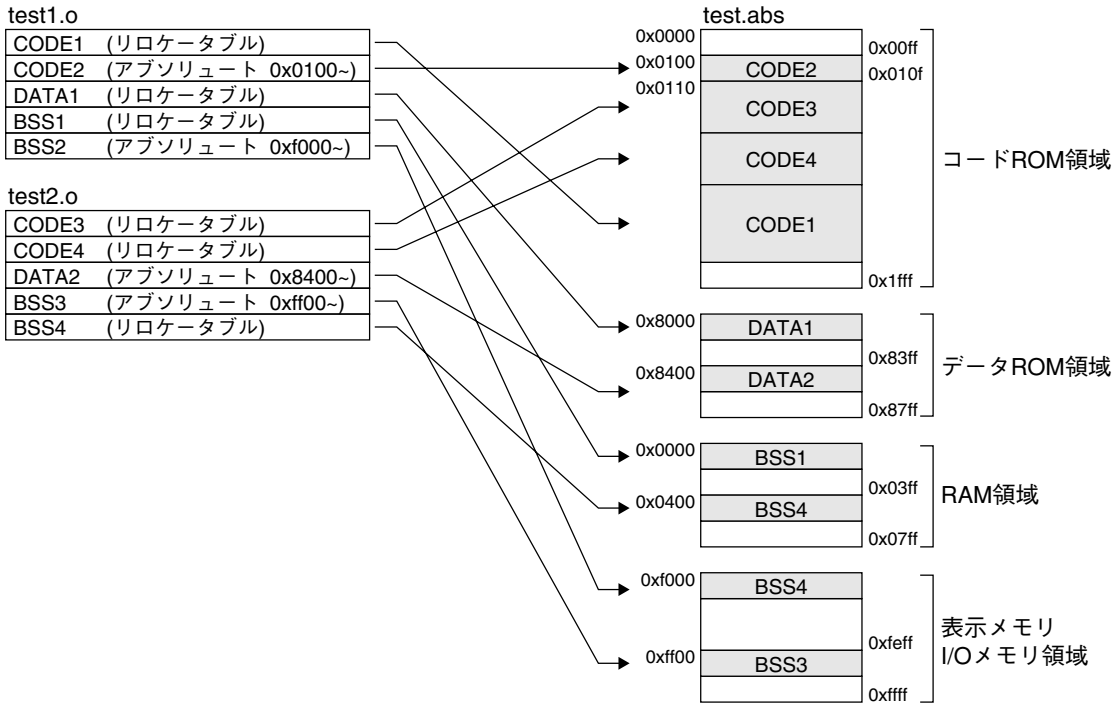


図5.10.2　リンク例

アブソリュートセクションのCODE2、BSS2、DATA2、BSS3はソースファイルで指定されている位置に配置されます。

"test2.o"のリロケータブルCODEおよびBSSセクションは-rcodeと-rbssで開始アドレスが指定されていますので、CODE3はアドレス0x0110から、CODE4はCODE3の直後に配置されます。BSS4はアドレス0x0400から配置されます。
"test1.o"のリロケータブルCODE、DATAおよびBSSセクションは開始アドレスが指定されていませんので、-code、-data、-bssで指定されている開始アドレスから配置されます。
リンカは最初にCODE1をアドレス0x0000～0x00ffの領域に配置することを試みます。もし、CODE1が0x100ワード以下であれば、CODE1は0x0000から配置されます。この例では、CODE1が0x100ワードより大きかったため、CODE4の直後に配置されます。
DATA1は0x8000から配置され、BSS1は0x0000から配置されます。
セクションが他のセクションにオーバーラップすることは許されていません。セクションサイズ以上の空き領域がない場合はエラーとなります。たとえば、CODE2が0x10ワードよりも大きい場合、エラーが発生します。

5.11 分岐最適化機能

PC相対分岐命令(jr、jrc、jrnc、jrz、jrnz、calr)から分岐先までの距離が-127～128の範囲を越える場合、その分岐命令の直前にはアドレス拡張命令(ldb %ext, imm8)が必要です。リロケータブルセクションはリンクが完了するまで位置が確定しませんので、リンカには拡張命令を自動挿入、削除、修正する機能があります。これにより、ソース内での拡張指定を省略することができます。ただし、この機能は分岐先の指定にラベルを使用している分岐命令にのみ有効です。
拡張命令の自動挿入/削除/修正機能はデフォルトで有効となり、特に指定のない場合は分岐最適化が行われます。-dオプションにより、この機能をすべて無効とすることができます。また、-diオプションにより拡張命令の挿入機能を無効に、-drオプションにより拡張命令の削除機能を無効にすることもできます(-dオプションが指定されていない場合)。

リンカはPC相対分岐命令から分岐先までの距離を調べ、その結果によって拡張命令の挿入、削除、あるいは修正を行います。

(1) 分岐先が分岐命令から-127～+128の範囲内

分岐命令に拡張命令が付いていない場合、拡張命令は挿入されません。
分岐命令に拡張命令が付いていた場合、その拡張命令は削除されます(-drオプションが指定されている場合は削除されません)。

例:
```
jr    LABEL              →    jr    LABEL

ldb   %ext,LABEL@rh
calr    LABEL@rl         →    calr  LABEL@rl
```

(2) 分岐先が分岐命令から-127～+128の範囲外

分岐命令に拡張命令が付いていない場合、拡張命令が正しく挿入されます。
分岐命令に不正な拡張命令が付いていた場合、その拡張命令は正しいものに置き換えられます。

例:
```
jr    LABEL              →    ldb   %ext,LABEL@rh
                                jr    LABEL@rl

ldb   %ext,LABEL1@rh     →    ldb   %ext,LABEL2@rh
calr  LABEL2@rl                calr  LABEL2@rl
```

分岐の最適化によってセクション間に隙間が生じた場合、その隙間をアドレスが小さい方向に詰めるようにリロケータブルセクションの配置が再調整されます。

5.12 エラー/ワーニングメッセージ

5.12.1 エラー

エラーが発生した場合、リンカはエラーメッセージを表示して、直ちに処理を終了します。オブジェクトファイルは出力されません。他のファイルはエラー発生前までに処理された分が出力されます。
リンカエラーメッセージの一覧を以下に示します。

エラーメッセージ	内　容
Branch destination too far from <address>	分岐先が分岐可能範囲外です。
CALZ for non zero page at <address>	指定アドレスが0x0000~0x00ffの範囲外です。
Cannot create absolute object file <FILE NAME>	アブソリュートオブジェクトファイルが作成できません。
Cannot open <file kind> file <FILE NAME>	ファイルが開けません。
Cannot read <file kind> file <FILE NAME>	ファイルが読み込めません。
Cannot relocate <section kind> section of <FILE NAME>	リロケータブルセクションが配置できません。
Cannot write <file kind> file <FILE NAME>	ファイルへの書き込みができません。
Illegal address range <address> for a code at <address>	TST/SET/CLRで指定されたアドレスが0x0000~0x003fまたは0xffC0~0xffffの範囲外です。
Illegal file name <FILE NAME>	ファイル名が不正です。
Illegal file name <FILE NAME> specified with option <option>	オプションで指定されているファイル名が不正です。
Illegal ICE parameter at line <line number> of <FILE NAME>	ICEパラメータファイルに不正なパラメータが記述されています。
Illegal object <FILE NAME>	入力ファイルがIEEE-695形式のアブソリュートオブジェクトファイルではありません。
Illegal option <option>	不正なオプションが指定されました。
No address specified with option <option>	オプションにアドレスの指定がありません。
No code to locate	マップする有効なコードがありません。
No ICE parameter file specified	ICEパラメータファイルが指定されていません。
No name and address specified with option <option>	オプションに名前とアドレスの指定がありません。
No object file specified	リンクするオブジェクトファイルが指定されていません。
Out of memory	メモリが確保できません。
<section kind> section <address>-<address> overlaps with <section kind> section <address>-<address>	セクションのアドレス範囲が他のセクションのアドレス範囲に重なっています。
<section kind> section <address>-<address> overlaps with the unavailable memory	セクションのアドレス範囲がメモリの未使用領域にかかっています。
Unresolved external <label> in <FILE NAME>	未定義シンボルを参照しています。
Unusable instruction code <instruction code> in <FILE NAME>	オブジェクトファイルが開発機種が未対応の命令を含んでいます。

5.12.2 ワーニング

ワーニングの場合、リンカは処理を継続します。他のエラーが発生しなければ、ワーニングメッセージを表示して処理を終了します。出力ファイルは生成されますが、プログラムの動作は保証できません。
ワーニングメッセージおよびその内容を以下に示します。

ワーニングメッセージ	内　容
Cannot create <file kind> file <FILE NAME>	ファイルが作成できません。
Cannot open <file kind> file <FILE NAME>	ファイルが開けません。
No debug information in <FILE NAME>	入力ファイルにデバッグ情報が含まれていません。
No symbols found	シンボルが見つかりません。
Second definition of label <label> in <FILE NAME>	指定のラベルは既に定義されています。
Second ICE parameter file <FILE NAME> ignored	2つ以上のICEパラメータファイルが指定されています。

5.13 注意事項

(1) リンク可能なセクション数やオブジェクト数などの制限は、メモリの空き容量に依存します。

(2) リンカで生成したアブソリュートオブジェクトファイルをデバッガにロードする場合、デバッガ起動時にリンカで使用したものと同じICEパラメータファイルを指定してください。

最新バージョンの制限事項やサポート機能、バグ情報についてはS5U1C63000Aのrel_asm_J.txtを参照してください。

6 HEXコンバータ

この章ではHEXコンバータhx63の機能を説明します。

6.1 機能

HEXコンバータhx63は、リンカが生成したIEEE-695形式のアブソリュートオブジェクトファイルをモトローラS形式またはインテルHEX形式のHEXファイルに変換します。この変換はプログラムをROMに書き込んでデバッグ/評価を行う場合や、マスクデータチェッカでマスクデータを作成する場合に必要です。ROMイメージのHEXデータを作成する際、HEXコンバータは未使用領域への0xffの埋め込みも行います。

6.2 入出力ファイル

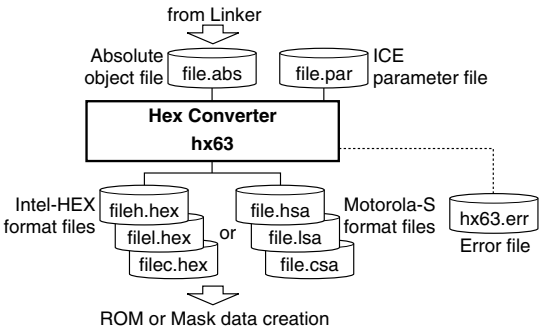


図6.2.1 フローチャート

6.2.1 入力ファイル

●アブソリュートオブジェクトファイル

ファイル形式: IEEE-695形式のバイナリファイル
ファイル名: <ファイル名>.abs （パス指定も可能）
内容: リンカで生成したアブソリュートオブジェクトファイルです。

●ICEパラメータファイル *省略不可

ファイル形式: バイナリファイル
ファイル名: <ファイル名>.par （パス指定も可能）
内容: S1C63 Family各機種の、メモリ構成や未使用命令などの情報が記録されています。このファイルは機種ごとに用意されています。リンカ、デバッガでも同じファイルを使用します。

6.2.2 出力ファイル

●HEXファイル

ファイル形式: モトローラS形式またはインテルHEX形式のテキストファイル
ファイル名: モトローラS形式 <ファイル名>.hsa、<ファイル名>.lsa および <ファイル名>.csa
インテルHEX形式 <ファイル名>h.hex、<ファイル名>l.hex および <ファイル名>c.hex
出力先: カレントディレクトリ
内容: オブジェクトコードの上位5ビットを8ビットに0拡張したデータを含むファイル(".hsa"または"h.hex")、コードの下位8ビットを含むファイル(".lsa"または"l.hex")およびデータROM用の4ビットデータを含むファイル(".csa"または"c.hex")が生成されます。デフォルトでモトローラSファイルが生成されます。インテルHEXファイルを作成する場合は-iオプションを指定します。

●エラーファイル

ファイル形式: テキストファイル
ファイル名: hex63.err
出力先: カレントディレクトリ
内容: 起動時オプション(-e)が指定された場合に出力されるファイルで、エラーメッセージなど、HEXコンバータが標準出力(stdout)に対して出力する内容を記録します。ファイル名はデフォルトで"hex63.err"ですが、-oオプションにより指定の名称に変わります。

6.3 起動方法

●コマンドラインの一般形

hx63 ^ [オプション] ^ <アブソリュートオブジェクトファイル> ^ <ICEパラメータファイル>

^はスペースを示します。
[ ]は省略可能なことを示します。
オプションとファイル名の記述順序に制限はありません。

●ファイル名

アブソリュートオブジェクトファイル: <File name.abs>
ICEパラメータファイル: <File name.par>

アブソリュートオブジェクトファイルは、拡張子(.abs)を省略できます。ICEパラメータファイルは拡張子を含めて指定してください。

Windowsの長いファイル名およびパスも指定可能です。ファイル名にスペースを挿入する場合は、ファイル名をダブルクォーテーションマーク(")で囲んで指定してください。

●オプション

HEXコンバータには4つの起動時オプションが用意されています。

-b
機能: コード有効部のみの変換
説明: アブソリュートオブジェクト内のコードが存在するアドレスのみを変換します。未使用領域は出力されません。
デフォルト: 本オプションを省略した場合、有効なプログラムメモリ領域全体が出力されます。未使用領域には0xffが埋め込まれます。

-e
機能: エラーファイルの出力
説明: HEXコンバータが標準出力デバイス(stdout)に出力するエラーメッセージなどの情報をエラーファイル(.err)に保存します。
デフォルト: 本オプションを省略した場合、エラーファイルは作成されません。

-i
機能: インテルHEX形式への変換
説明: HEXファイル("h.hex"、"l.hex"、"c.hex")をインテルHEX形式で作成します。
デフォルト: 本オプションを省略した場合、モトローラSファイル(".hsa"、".lsa"、".csa")が作成されます。

-o <ファイル名>
機能: 出力パス/ファイル名の指定
説明: 出力パス/ファイル名を拡張子なしで、あるいは拡張子".hsa"、".lsa"、".csa"、"h.hex"、"l.hex"、"c.hex"のいずれかを付けて指定します。1つのファイル名の指定のみで、3つのHEXファイルが作成されます。
拡張子の指定を省略した場合は、指定のパス/ファイル名に出力形式に従った拡張子が付加されます。この場合、インテルHEXファイルには"h.hex"、"l.hex"、"c.hex"が付加されるため、DOSのファイル名(最大8文字)がWindowsの長いファイル名に変わる場合があります。
デフォルト: 本オプションを省略すると、入力ファイル名を出力ファイルに使用します。

コマンドラインにオプションを入力する場合、オプションの前後には1個以上のスペースが必要です。
例: `c:¥epson¥s1c63¥bin¥hx63 -e test.abs par63xxx.par`

6.4 メッセージ

HEXコンバータはメッセージをすべて標準出力(stdout)に対して出力します。

●起動メッセージ

起動時は次のメッセージを出力します。

```
HEX converter 63 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 1998-2001
```

●終了メッセージ

正常に終了した場合は、生成したファイル名を出力します。

```
Created hex file <FILE NAME>.HSA
Created hex file <FILE NAME>.LSA
Created error log file HX63.ERR

Hex conversion 0 error(s) 0 warning(s)
```

●Usage出力

ファイル名が指定されなかったり、オプション指定が正しくない場合、次の使用方法のメッセージを出力して終了します。

```
Usage: hx63 [options] <file names>
Options: -b              Do not fill unused memory with 0xff
         -e              Output error log file (HX63.ERR)
         -i              Use Intel Hex format
         -o <file name>  Specify output file name
File names: Absolute object file (.ABS)
            ICE parameter file (.PAR)
```

●エラー・ワーニング発生時

エラーが発生した場合は、終了メッセージの前にエラーメッセージが表示されます。

例:
```
Error : No ICE parameter file specified
  Hex conversion 1 error(s) 0 warning(s)
```

エラーの場合、HEXコンバータは出力ファイルを作成せずに終了します。

ワーニングが発生した場合は、終了メッセージの前にワーニングメッセージが表示されます。

例:
```
Warning : Output file name conflict
  Hex conversion 0 error(s) 1 warning(s)
```

ワーニングの場合、HEXコンバータは出力ファイルを作成して終了します。ただし、出力結果は保証されません。

エラーとワーニングの詳細については"6.6 エラー/ワーニングメッセージ"を参照してください。

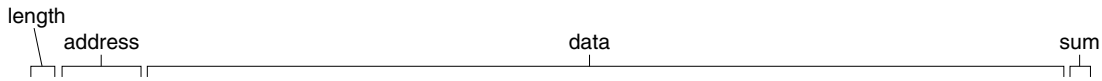
6.5 HEXファイルの出力

6.5.1 HEXファイルの構成

S1C63000の各命令は13ビットで構成されるため、HEXコンバータも常に上位データと下位データ用の2つのプログラム命令のHEXファイルを生成します。
下位データ用のHEXファイル(".lsa"、"l.hex")は、オブジェクトコードの下位バイト(ビット7～0)を含みます。上位データ用のHEXファイル(".hsa"、"h.hex")は、オブジェクトコードの上位5ビット(ビット12～8)を下位5ビットとして0b000を上位に拡張したバイトデータを含みます。
データROM用の4ビットデータは、".csa"または"c.hex"ファイルに出力されます。
モトローラSファイルの他に、-iオプションの指定によりインテルHEX形式のHEXファイルを作成することができます。ただし、デバッガにロードするHEXファイルやマスクデータの作成に使用するHEXファイルは、モトローラS形式で作成してください。デバッガとマスクデータチェッカはインテルHEXファイルには対応していません。

6.5.2 モトローラSファイル

HEXコンバータはデフォルトで、IEEE-695形式のアブソリュートオブジェクトファイルをモトローラS2ファイルに変換します。
上位プログラムファイル用は".hsa"、下位プログラムファイル用は".lsa"、データROMファイル用は".csa"の名称で作成されます。
モトローラS2形式のデータ例を以下に示します。




```
S224000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFB
S224000020FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFDB
                                        :
S22400010008E000F04200420606FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF89
                                        :
S804000000FB
```

S2(2バイト): その行がデータレコードであることを示します。
S8(2バイト): その行がエンドレコード(データの終端)であることを示します。
length(1バイト): address + data + sumのレコード長を示します。最大長は0x24で、エンドレコードは0x04に固定です。
address(3バイト): 各レコードの先頭データが置かれるアドレスを示します。
data(max. 32バイト): オブジェクトコードのデータです。エンドレコードには含まれません。
sum(1バイト): lengthからレコード最後のデータまでのチェックサム(1の補数)です。

エンドレコードは常に"S804000000FB"となります。

6.5.3 インテルHEXファイル

HEXコンバータはiオプションが指定されると、IEEE-695形式のアブソリュートオブジェクトファイルをインテルHEXファイルに変換します。
上位プログラムファイル用は"<ファイル名>h.hex"、下位プログラムファイル用は"<ファイル名>l.hex"、データROMファイル用は"<ファイル名>c.hex"の名称で作成されます。
インテルHEX形式のデータ例を以下に示します。

data volume(1バイト): 各レコード(行)のデータ長を示します。データレコードの最大幅は0x10で、エンドレコードは0x00に固定されます。
address(2バイト): 各レコードの先頭データが置かれるアドレスを示します。
type(1バイト): HEX形式を示します。現在は"00"に固定されています。
data(max. 16バイト): オブジェクトコードのデータです。エンドレコードには含まれません。
sum(1バイト): data volumeからレコード最後のデータまでのチェックサム(2の補数)です。

エンドレコードは常に"00000001FF"となります。

注: HEXファイルをマスクデータの作成に使用する場合、マスクデータチェッカが対応していませんので、インテルHEX形式は指定しないでください。

6.5.4 変換範囲

HEXコンバータのデフォルト設定では、各機種で使用可能なROM領域全体のデータを含むHEXファイルが生成されます。未使用アドレスには0xffが出力されます。たとえば、2KBのコードROMを内蔵する機種でプログラムがアドレス0x0から0x6ffまでを使用している場合、0x700～0x7ffの領域には0xffが埋め込まれます。0x0から0x6ffの領域内の未使用のアドレスにも0xffが埋め込まれます。
マスクデータチェッカでマスクデータを作成する場合は、この形式でHEXファイルを作成してください。

-bオプションが指定されると、HEXコンバータはアブソリュートオブジェクトの未使用アドレスについてはデータを出力しません。HEXファイルのサイズ削減に有効ですが、この形式で作成したHEXファイルをマスクデータの作成には使用できません。

6.6 エラー/ワーニングメッセージ

6.6.1 エラー

エラーが発生した場合、HEXコンバータはエラーメッセージを表示して、直ちに処理を終了します。HEXファイルは出力されません。
HEXコンバータエラーメッセージの一覧を以下に示します。

エラーメッセージ	内　容
Cannot create <file kind> file <FILE NAME>	ファイルが作成できません。
Cannot open <file kind> file <FILE NAME>	ファイルが開けません。
Cannot read <file kind> file <FILE NAME>	ファイルが読み込めません。
Cannot write <file kind> file <FILE NAME>	ファイルへ書き込めません。
Illegal file name <FILE NAME> specified with option <option>	指定のHEXファイル名が不正です。
Illegal ICE parameter at line <line number> of <FILE NAME>	ICEパラメータファイルに不正なパラメータが含まれています。
Illegal file name <FILE NAME>	入力ファイル名が不正です。
Illegal option <option>	不正なオプションが指定されました。
Illegal absolute object format	入力ファイルがIEEE-695形式のオブジェクトではありません。
No ICE parameter file specified	ICEパラメータファイルが指定されていません。
Out of memory	メモリが確保できません。

6.6.2 ワーニング

ワーニングの場合、HEXコンバータは処理を継続します。他のエラーが発生しなければ、ワーニングメッセージを表示して処理を終了します。
ワーニングメッセージおよびその内容を次に示します。

ワーニングメッセージ	内　容
Input file name extension .XXX conflict	同じ拡張子で2つ以上のファイル名が指定されました。 最後に指定されたファイルを使用します。

6.7 注意事項

(1) マスクデータチェッカでマスクデータを作成するために使用するHEXファイルは、モトローラS形式で、有効メモリ領域全体を変換して作成してください(-bと-iオプションは指定しないでください)。これ以外のHEXファイルを作成した場合、マスクデータチェッカでエラーとなります。

(2) インテルHEX形式の変換で、8文字の出力ファイル名(DOSファイル名)を拡張子なしで指定した場合、出力HEXファイル名は"h.hex"、"l.hex"、"c.hex"が付加されたWindowsの長いファイル名となります。

最新バージョンの制限事項やサポート機能、バグ情報についてはS5U1C63000Aのrel_asm_J.txtを参照してください。

7 ディスアセンブラ

この章ではディスアセンブラds63の機能を説明します。

7.1 機能

ディスアセンブラds63は、IEEE-695形式またはモトローラS形式のオブジェクトファイルを読み込み、コードをニーモニックに逆アセンブルします。結果はソースファイルとして出力されます。復元したソースファイルは、アセンブラ、リンカ、HEXコンバータでオリジナルのソースと同様に処理可能で、復元前と同じアブソリュートオブジェクトまたはHEXファイルを生成できます。

7.2 入出力ファイル

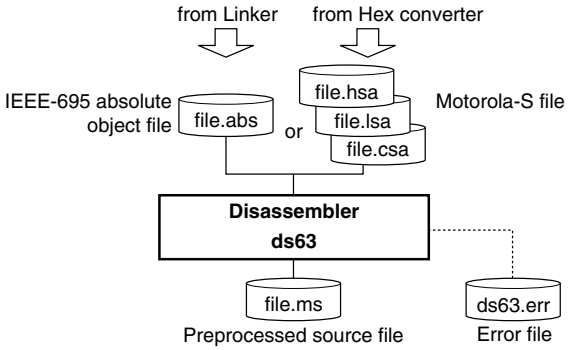


図7.2.1 フローチャート

7.2.1 入力ファイル

●アブソリュートオブジェクトファイル

ファイル形式: IEEE-695形式のバイナリファイル
ファイル名: <ファイル名>.abs （パス指定も可能）
内容: リンカで生成したアブソリュートオブジェクトファイルです。

●HEXファイル

ファイル形式: モトローラS形式のテキストファイル
ファイル名: <ファイル名>.hsa、<ファイル名>.lsa および <ファイル名>c.hex
内容: HEXコンバータで生成したHEXファイルです。オブジェクトコードの上位5ビットを8ビットに0拡張したデータを含むファイル(".hsa")、コードの下位8ビットを含むファイル(".lsa")、データROM用の4ビットデータを含むファイル(".csa")の3つが必要です。データROMがない場合、".csa"ファイルは必要ありません。

7.2.2 出力ファイル

●ソースファイル

ファイル形式: テキストファイル
ファイル名: <ファイル名>.ms
出力先: カレントディレクトリ
内容: 入力ファイルを逆アセンブルした内容が出力されます。

●エラーファイル

ファイル形式: テキストファイル
ファイル名: ds63.err
出力先: カレントディレクトリ
内容: 起動時オプション(-e)が指定された場合に出力されるファイルで、エラーメッセージなど、ディスアセンブラが標準出力(stdout)に対して出力する内容を記録します。ファイル名はデフォルトで"ds63.err"ですが、-oオプションにより指定の名称に変わります。

7.3 起動方法

●コマンドラインの一般形

ds63 ∧ [オプション] ∧ <入力ファイル名>

∧はスペースを示します。
[ ]は省略可能なことを示します。

●入力ファイル名

アブソリュートオブジェクトファイル: <File name>.abs
モトローラSファイル: <File name>.hsa, <File name>.lsa, <File name>.csa

入力ファイルは拡張子を含めて指定してください。モトローラSファイルは".hsa"、".lsa"または".csa"のいずれか1つを指定します。指定した以外の拡張子のファイルは自動的にロードされます。

Windowsの長いファイル名およびパスも指定可能です。ファイル名にスペースを挿入する場合は、ファイル名をダブルクォーテーションマーク(")で囲んで指定してください。

●オプション

ディスアセンブラには4つの起動時オプションが用意されています。

-cl
機能: 小文字の使用
説明: 命令とラベルのすべてを小文字で出力します。
デフォルト: 本オプションおよび-cuオプションを省略した場合、ソースはラベルに大文字を、命令に小文字を使用して作成されます。

-cu
機能: 大文字の使用
説明: 命令とラベルのすべてを大文字で出力します。
デフォルト: 本オプションおよび-clオプションを省略した場合、ソースはラベルに大文字を、命令に小文字を使用して作成されます。

-e
機能: エラーファイルの出力
説明: ディスアセンブラが標準出力デバイス(stdout)に出力するエラーメッセージなどの情報をエラーファイル(.err)に保存します。
デフォルト: 本オプションを省略した場合、エラーファイルは作成されません。

-o [出力ファイル名]
機能: ファイル名の指定
説明: ファイル名を拡張子なしで、あるいは拡張子".ms"を付けて指定します。
拡張子の指定を省略した場合は、ファイル名に".ms"が付加されます。
デフォルト: 本オプションを省略すると、入力ファイル名を出力ファイルに使用します。

コマンドラインにオプションを入力する場合、オプションの前後には1個以上のスペースが必要です。
例: `c:¥epson¥s1c63¥bin¥ds63 -e -o output.ms c:¥input.abs`

7.4 メッセージ

ディスアセンブラはメッセージをすべて標準出力(stdout)に対して出力します。

●起動メッセージ

起動時は次のメッセージを出力します。

```
Disassembler 63 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 1998-2001
```

●終了メッセージ

正常に終了した場合は、生成したファイル名を出力します。

```
Created preprocessed source file <FILE NAME>.MS
Created error log file DS63.ERR

Disassembly 0 error(s) 0 warning(s)
```

●Usage出力

ファイル名が指定されなかったり、オプション指定が正しくない場合、次の使用方法のメッセージを出力して終了します。

```
Usage: ds63 [options] <file name>
Options: -cl             Use lower case characters
         -cu             Use upper case characters
         -e              Output error log file (DS63.ERR)
         -o <file name>  Specify output file name
File names: Absolute object file (.ABS or .CSA/.LSA/.HSA)
```

●エラー・ワーニング発生時

エラーが発生した場合は、終了メッセージの前にエラーメッセージが表示されます。

例:
```
Error: Cannot open file TEST.ABS
  Disassembly 1 error(s) 0 warning(s)
```

エラーの場合、ディスアセンブラは出力ファイルを作成せずに終了します。

ワーニングが発生した場合は、終了メッセージの前にワーニングメッセージが表示されます。

例:
```
Warning: Input file name extension .HSA conflict
  Disassembly 0 error(s) 1 warning(s)
```

ワーニングの場合、ディスアセンブラは出力ファイルを作成して終了します。

エラーとワーニングの詳細については"7.6 エラー/ワーニングメッセージ"を参照してください。

7.5 逆アセンブル出力

オブジェクトコードが逆アセンブルによってニーモニックに変換されます。分岐命令については、オペランドと分岐先に"CODEx:"の形式でラベルが付けられます。"x"は16進数のラベル番号です。その他の参照シンボルも"LABELx"、"IOx"、"RAMx"として付加されます。各コードのブロックの先頭には、.org擬似命令を挿入します。
以下に逆アセンブル結果の例を示します。

●出力サンプル

アブソリュートリストファイル"test.als"

```
Linker 63 ver x.xx Absolute list file "TEST.ALS" Mon Jan 15 12:40:41 2001

          1:                    ; sub.s
          2:                    ; AS63 test program (subroutine)
          3:
          4:                            .global RAM_BLK1
          5:
          6:                    ;***** RAM block 1 initialize *****
          7:
          8:                            .global INIT_RAM_BLK1
          9:                    INIT_RAM_BLK1:
         10:   0000   0800              ldb    %ext,RAM_BLK1@h
         11:   0001   0a04              ldb    %xl,RAM_BLK1@l          ;set RAM_BLK1 address to x
         12:   0002   1e90              ld     [%x]+,0x0
         13:   0003   1e90              ld     [%x]+,0x0
         14:   0004   1e90              ld     [%x]+,0x0
         15:   0005   1e80              ld     [%x],0x0                ;set 0x0000 to RAM_BLK1
         16:   0006   1ff8              ret
         17:
         18:                    ;***** RAM block 1 increment *****
         19:
         20:                            .global INC_RAM_BLK1
         21:                    INC_RAM_BLK1:
         22:   0007   0800              ldb     %ext,RAM_BLK1@h
         23:   0008   0a04              ldb     %xl,RAM_BLK1@l          ;set RAM_BLK1 address to x
         24:   0009   1911              add     [%x]+,1
         25:   000a   1990              adc     [%x]+,0
         26:   000b   1990              adc     [%x]+,0
         27:   000c   1980              adc     [%x],0                  ; increment 16bit value
         28:   000d   1ff8              ret
         29:                    ; main.s
         30:                    ; AS63 test program (main routine)
         31:                    ;
         32:
         33:                    ;***** INITIAL SP1 & SP2 ADDRESS DEFINITION *****
         34:
         35:                    #ifdef SMALL_RAM
         36:                            .set SP1_INIT_ADDR 0xb          ;SP1 init addr = 0x2c
         37:                    #else
         38:                            .set SP1_INIT_ADDR 0x4b         ;SP1 init addr = 0x12c
         39:                    #endif
         40:
         41:                            .set SP2_INIT_ADDR 0x1f         ;SP2 init addr = 0x1f
         42:
         43:
         44:                    ;***** NMI & BOOT, LOOP *****
         45:
         46:                            .global INIT_RAM_BLK1           ; subroutine in sub.s
         47:                            .global INC_RAM_BLK1            ; subroutine in sub.s
         48:
         49:                            .org    0x100
         50:                    NMI:
         51:   0100   08fe   (+)         ldb  ext,fe
         52:   0101   02fe              calr    INIT_RAM_BLK1           ; initialize RAM block 1
         53:   0102   1ff9              reti                            ; in NMI(watchdog timer)
         54:
         55:                            .org    0x110
         56:                    BOOT:
```

```
        57:   0110   094b              ldb     %ba,SP1_INIT_ADDR
        58:   0111   1fc4              ldb     %sp1,%ba                ; set SP1
        59:   0112   091f              ldb     %ba,SP2_INIT_ADDR
        60:   0113   1fc6              ldb     %sp2,%ba                ; set SP2
        61:   0114   08fe   (+)        ldb  ext,fe
        62:   0115   02ea              calr    INIT_RAM_BLK1           ; initialize RAM block 1
        63:                      LOOP:
        64:   0116   08fe   (+)        ldb  ext,fe
        65:   0117   02ef              calr    INC_RAM_BLK1            ; increment RAM block 1
        66:   0118   00fd              jr      LOOP                    ; infinity loop
```

出力ソースファイル"test.ms"（デフォルト）

```
;Disassembler 63 Ver x.xx Assembly source file TEST.MS Mon Jan 15 12:40:41 2001

        .set LABEL1 0x4
        .set LABEL2 0x4
        .set LABEL3 0x4b
        .set LABEL4 0x1f
        .code
        .org     0x0
CODE1:
        ldb %ext,LABEL1@h
        ldb %xl,LABEL1@l
        ld [%x]+,0x0
        ld [%x]+,0x0
        ld [%x]+,0x0
        ld [%x],0x0
        ret
CODE2:
        ldb %ext,LABEL2@h
        ldb %xl,LABEL2@l
        add [%x]+,0x1
        adc [%x]+,0x0
        adc [%x]+,0x0
        adc [%x],0x0
        ret
        .code
        .org     0x100
        ldb %ext,CODE1@rh
        calr CODE1@rl
        reti
        .code
        .org     0x110
        ldb %ba,LABEL3@l
        ldb %sp1,%ba
        ldb %ba,LABEL4@l
        ldb %sp2,%ba
        ldb %ext,CODE1@rh
        calr CODE1@rl
CODE3:
        ldb %ext,CODE2@rh
        calr CODE2@rl
        jr CODE3@rl
```

出力ソースファイル"test.ms"（-clオプション選択時）

```
;Disassembler 63 Ver x.xx Assembly source file TEST.MS Mon Jan 15 12:40:41 2001

        .set label1 0x4
        .set label2 0x4
        .set label3 0x4b
        .set label4 0x1f
        .code
        .org     0x0
code1:
        ldb %ext,label1@h
        ldb %xl,label1@l
        ld [%x]+,0x0
        ld [%x]+,0x0
        ld [%x]+,0x0
```

```
        ld [%x],0x0
        ret
code2:
        ldb %ext,label2@h
        ldb %xl,label2@l
        add [%x]+,0x1
        adc [%x]+,0x0
        adc [%x]+,0x0
        adc [%x],0x0
        ret
        .code
        .org     0x100
        ldb %ext,code1@rh
        calr code1@rl
        reti
        .code
        .org     0x110
        ldb %ba,label3@l
        ldb %sp1,%ba
        ldb %ba,label4@l
        ldb %sp2,%ba
        ldb %ext,code1@rh
        calr code1@rl
code3:
        ldb %ext,code2@rh
        calr code2@rl
        jr code3@rl
```

出力ソースファイル"test.ms"（-cuオプション選択時）

```
;Disassembler 63 Ver x.xx Assembly source file TEST.MS Mon Jan 15 12:40:41 2001

        .SET LABEL1 0X4
        .SET LABEL2 0X4
        .SET LABEL3 0X4B
        .SET LABEL4 0X1F
        .CODE
        .ORG     0X0
CODE1:
        LDB %EXT,LABEL1@H
        LDB %XL,LABEL1@L
        LD [%X]+,0X0
        LD [%X]+,0X0
        LD [%X]+,0X0
        LD [%X],0X0
        RET
CODE2:
        LDB %EXT,LABEL2@H
        LDB %XL,LABEL2@L
        ADD [%X]+,0X1
        ADC [%X]+,0X0
        ADC [%X]+,0X0
        ADC [%X],0X0
        RET
        .CODE
        .ORG     0X100
        LDB %EXT,CODE1@RH
        CALR CODE1@RL
        RETI
        .CODE
        .ORG     0X110
        LDB %BA,LABEL3@L
        LDB %SP1,%BA
        LDB %BA,LABEL4@L
        LDB %SP2,%BA
        LDB %EXT,CODE1@RH
        CALR CODE1@RL
CODE3:
        LDB %EXT,CODE2@RH
        CALR CODE2@RL
        JR CODE3@RL
```

7.6 エラー/ワーニングメッセージ

7.6.1 エラー

エラーが発生した場合、ディスアセンブラはエラーメッセージを表示して、直ちに処理を終了します。ソースファイルは出力されません。
ディスアセンブラエラーメッセージの一覧を以下に示します。

エラーメッセージ	内　容
Cannot create <file kind> file <FILE NAME>	ファイルが作成できません。
Cannot open <file kind> file <FILE NAME>	ファイルが開けません。
Cannot read <file kind> file <FILE NAME>	ファイルが読み込めません。
Cannot write <file kind> file <FILE NAME>	ファイルへ書き込めません。
Illegal file name <FILE NAME> specified with option <option>	指定の出力ソースファイル名が不正です。
Illegal file name <FILE NAME>	入力ファイル名が不正です。
Illegal HEX data format	入力ファイルがモトローラSファイルではありません。
Illegal option <option>	無効なオプションが指定されました。
Out of memory	メモリが確保できません。

7.6.2 ワーニング

ワーニングの場合、ディスアセンブラは処理を継続します。他のエラーが発生しなければ、ワーニングメッセージを表示して処理を終了します。
ワーニングメッセージおよびその内容を次に示します。

ワーニングメッセージ	内　容
Input file name extension .XXX conflict	同じ拡張子で2つ以上のファイル名が指定されました。 最後に指定されたファイルを使用します。
Cannot open Hex file xxx.csa	現在の機種はデータROMを搭載していないため、".csa"ファイルを開くことはできません。

7.7 注意事項

最新バージョンの制限事項やサポート機能、バグ情報についてはS5U1C63000Aのrel_asm_J.txtを参照してください。

8 デバッガ

この章ではデバッガdb63の操作方法を説明します。

8.1 特長

デバッガdb63はリンカによって生成したIEEE-695形式のオブジェクトファイルを読み込み、プログラムのデバッグを行うためのソフトウェアです。
以下の特長と機能を持ちます。

- マルチウィンドウにより各種のデータを一度に参照可能
- 使用頻度の高いコマンドはツールバーおよびメニューからマウス操作で実行可能
- アセンブリソースコードに対応したソース表示およびシンボリックデバッグ機能
- プログラムの連続実行と2種類のステップ実行が可能
- 5種類のブレーク機能
- レジスタやメモリのオンザフライによるリアルタイム表示機能
- 時間とステップによる実行時間の表示機能
- 高度なトレース機能
- コマンドファイルによるコマンドの自動実行機能

8.2 入出力ファイル

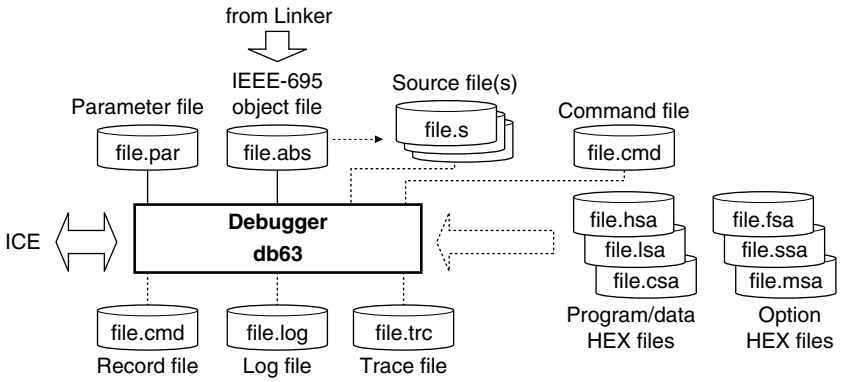


図8.2.1 フローチャート

8.2.1 入力ファイル

●パラメータファイル

ファイル形式: バイナリファイル
ファイル名: <ファイル名>.par
内容: 各機種のメモリ情報などが記録された、デバッガを起動するために必須のファイルです。各機種ごとに用意されています。

以下のファイルはすべてコマンドの指定によってデバッガが読み込みます。

●オブジェクトファイル

ファイル形式: IEEE-695形式のバイナリファイル
ファイル名: <ファイル名>.abs （".abs"以外の拡張子も使用可能）
内容: リンカで生成したオブジェクトファイルです。lfコマンドにより読み込みます。デバッグ情報を含んだIEEE-695形式のファイルを読み込むことで、ソース表示およびシンボリックデバッグが行えます。

●ソースファイル

ファイル形式: テキストファイル
ファイル名: <ファイル名>.s
内容: 上記オブジェクトファイルのソースファイルで、ソース表示を行うときに読み込まれます。

●プログラムファイル

ファイル形式: モトローラS形式のHEXファイル
ファイル名: <ファイル名>.hsa、<ファイル名>.lsa
内容: プログラムROMのロードイメージファイルで、loコマンドにより読み込みます。".hsa"はプログラムコードの上位5ビット、".lsa"は下位8ビットに対応しています。これらのファイルは、IEEE-695形式のオブジェクトファイルからマスクデータ作成用にHEXコンバータで生成したものです。IEEE-695形式のファイルのようにソース表示およびシンボリックデバッグは行えませんが、最終プログラムデータの動作確認が行えます。

●データROM用データファイル

ファイル形式: モトローラS形式のHEXファイル
ファイル名: <ファイル名>.csa
内容: データROMのロードイメージファイルで、loコマンドにより読み込みます。このファイルは、IEEE-695形式のオブジェクトファイルからマスクデータ作成用にHEXコンバータで生成したものです。IEEE-695形式のアブソリュートオブジェクトファイルをロードした場合、このファイルをロードする必要はありません。

●オプションデータファイル

ファイル形式: モトローラS形式のHEXファイル
ファイル名: <ファイル名>.fsa、<ファイル名>.ssa、<ファイル名>.msa(機種により異なる)
内容: 各機種のハードウェアオプション設定用のデータファイルで、loコマンドにより読み込みます。このファイルは、機種別のデベロップメントツールで生成したものです。

●コマンドファイル

ファイル形式: テキストファイル
ファイル名: <ファイル名>.cmd (".cmd"以外の拡張子も使用可能)
内容: 連続して実行させるデバッグコマンドを記述したファイルです。頻繁に使用する一連のコマンドを書き込んでおくことで、キーボードからのコマンド入力の手間を省くことができます。このファイルはcom、cmwコマンドにより読み込み、実行させます。

8.2.2 出力ファイル

●ログファイル

ファイル形式: テキストファイル
ファイル名: <ファイル名>.log (".log"以外の拡張子も使用可能)
内容: 実行したコマンドと実行結果が出力されます。このファイル出力はlogコマンドによって制御できます。

●レコードファイル

ファイル形式: テキストファイル
ファイル名: <ファイル名>.rec (".rec"以外の拡張子も使用可能)
内容: 実行したコマンドが出力されます。このファイル出力はrecコマンドによって制御できます。

●トレースファイル

ファイル形式: テキストファイル
ファイル名: <ファイル名>.trc (".trc"以外の拡張子も使用可能)
内容: トレースデータの指定範囲が出力されます。このファイル出力はtfコマンドによって制御できます。

8.3 起動方法

8.3.1 起動フォーマット

●コマンドラインの一般形

db63 ∧ <パラメータファイル名> ∧ [起動時オプション]

∧はスペースを示します。
[ ]は省略可能なことを示します。

注: パラメータファイルは拡張子".par"によって認識されます。したがって、コマンドラインのパラメータファイル名は".par"を含めて指定してください。

8.3.2 起動時オプション

デバッガには4種類の起動時オプションが用意されています。

<コマンドファイル名>
機能: コマンドファイルの指定
説明: デバッガ起動直後に一連のコマンドを実行させたい場合に、それらを書き込んだコマンドファイルを指定します。

次の2つのオプションはRS-232Cインタフェース版ICE使用時にのみ有効です。

-comX
機能: 通信ポートの指定(RS-232Cインタフェース用)
説明: ICEと通信を行うパソコンの通信ポートを指定します。Xの部分にはポート番号を指定してください。使用可能なポートはコンピュータによって異なります。本オプションが指定されない場合、COM1ポートを使用してICEと通信します。

-b <ボーレート>
機能: 通信速度の指定(RS-232Cインタフェース用)
説明: パソコン側の通信速度を指定します。<ボーレート>には2400、4800、9600、19200、38400のいずれか1つを指定してください。本オプションが指定されない場合、ボーレートは9600bpsに設定されます。この値はICEの初期設定と同じです。ICE側の設定はICE上のDIPスイッチで切り換えます。

次のオプションはUSBインタフェース版ICE使用時にのみ有効です。

-usb
機能: USB接続の指定(USBインタフェース用)
説明: USBインタフェース版ICEを接続する場合に指定します。

コマンドラインにオプションを入力する場合、オプションの前後には1個以上のスペースが必要です。
例: `c:¥epson¥s1c63¥bin¥db63 par63xxx.par startup.cmd -com2 -b 19200`

起動時オプションを指定しない場合のデフォルト設定は、-com1 および -b 9600です。
パラメータファイルが指定されない場合やオプションの指定が不正な場合、デバッガは次のメッセージを表示して終了します。

```
- Usage -
     db63.exe <parameter_file> <startup options>
Options:
     command_file:  ... specifies a command file
     -comX(X:1-4)   ... com port, default com1
     -b             ... baud rate, 2400,9600(default),19200,38400
     -usb           ... specifies usb communication
     -sim           ... specifies simulator mode
```

8.3.3 起動メッセージ

デバッガは起動時に次のメッセージを[Command]ウィンドウ(ウィンドウについては次項を参照)に出力します。

```
Debugger63 Ver x.xx Copyright SEIKO EPSON CORP. 1998-2007

Connecting COMx with xxxxx baud rate ... done  (*1)
Parameter file name      : xxxxxxxx.par
               Version   : xx
               Chip name : 63xxx
CPU version              : x.x
PRC board version        : x.x
LCD board version        : x.x
EXT board version        : x.x
ICE hardware version     : x.x
ICE software version     : x.x
DIAG test                : omitted
Map.............................. done
Initialize....................... done
>
```

*1 RS-232Cインタフェース版ICEを使用した場合の表示です。USBインタフェース版ICE使用時は、次のように表示されます。

```
Connecting USB ... done
```

以下の説明内のメッセージにも上記のRS-232Cインタフェース版を記載しています。USBインタフェース版使用時は、"Connecting USB"が表示されます。

8.3.4 起動時のハードウェアチェック

デバッガを起動すると、デバッガは以下のテストおよび初期化を行います。

(1)ICEの接続テスト

デバッガdb63は、はじめにICEが接続され、パソコンとの通信が問題なく行えることをチェックします。[Command]ウィンドウには次のメッセージが表示されます。

テスト中

```
Connecting COMx with xxxxx baud rate ...
```

正常終了時

```
Connecting COMx with xxxxx baud rate ... done
```

エラー発生時

```
Connecting COMx with xxxxx baud rate ... failure
<error message>
```

エラーメッセージ発生時はパソコンとICE間の通信が正常に行われていません。以下の点について確認してください。

- 正規のインタフェースケーブルを使用しているか
- COMポートは正しいか
- ボーレートは合っているか
- PRCボードが正しく装着されているか
- ICEの電源はONになっているか
- リセット状態になったままではないか

エラーの原因については8.10項「ステータス/エラー/ワーニングメッセージ」を参照してください。

このテストの際にICEが準備状態ではなかった場合、次のような処理を行います。

ICEがターゲットプログラムを実行中
この場合、デバッガdb63はICEに強制ブレークコマンドを送信し、数秒後に再度接続テストを行います。

ICEがBUSY状態
この場合、デバッガdb63は数秒後に再度接続テストを行います。

ICEがフリーラン状態
この場合、次のメッセージを表示します。

```
Connecting COMx with xxxxx baud rate ... failure
Error : ICE is free run mode
```

一度デバッガを終了し、ICEをICEモード(ICE/RUNスイッチをICE側)に設定してから再度デバッガを起動してください。

ICEが自己診断実施中
この場合、デバッガdb63はICEの自己診断が終了するのを待ってから接続テストを行います。なお、ICEの自己診断は、ICE上のDIP SW8が上にセットされていると電源投入時に実行されます。SW8が下にセットされている場合は実行されません。自己診断の開始から終了まで約5分かかります。終了するまで、お待ちください。
このときのメッセージは次のとおりです。

```
Connecting COMx with xxxxx baud rate ...
DIAG test, please wait 5 min.  .. done
```

自己診断でエラーが発見された場合は、"done"の代わりにエラーメッセージが表示されます。

(2)バージョンチェック

接続テストが正常に終了すると、次にパラメータファイルの内容と、ICEおよびICEに挿入されているペリフェラルボードのバージョンをチェックします。

```
Parameter file name      : xxxxxxxx.par
               Version   : xx
               Chip name : 63xxx
CPU version              : x.x
PRC board version        : x.x
LCD board version        : x.x
EXT board version        : x.x
ICE hardware version     : x.x
ICE software version     : x.x
DIAG test                : omitted
```

ここで、パラメータファイルの設定内容とICEのシステム構成(PRC(ペリフェラル)ボードやLCDボードなどの拡張ボードを含む)およびそのバージョンが合っているかチェックします。
必要なボードが装着されていない場合、あるいは不要なボード、バージョンの異なるボードが装着されている場合、ワーニングメッセージを表示します。

(3)ICEの初期化

最後に次のとおりICEの初期化を行います。
- マッピング(パラメータファイルに従ったメモリ構成の設定)
- マップされたメモリの初期化(RAM: 0xa、コードROM: 0x1fff、データROM: 0xf)
- オプションデータの初期化(すべて0クリア)
- ブレーク条件の初期化(すべてのブレーク条件のクリア)
- トレース条件の初期化(ノーマルトレースを設定、トレーストリガポイントを0に設定)
- 実行サイクル数カウンタを0に設定
- 監視データアドレスの初期設定(0、4、8、C番地)
- CPUレジスタの初期化

正常終了時

```
Map.............................. done
Initialize....................... done
>
```

エラー発生時

```
Map.............................. done
Initialize....................... done
    please quit db63 and restart!
>
```

上記処理においてエラーが発生した場合は、一度デバッガを終了後、原因の確認・修復を行い、再度起動してください。

初期化終了後、デバッガウィンドウは前回の終了時と同じウィンドウ構成(位置とサイズを含む)で開きます。このときの各ウィンドウの表示内容は以下のとおりです。

ウィンドウ	表示内容
[Command]ウィンドウ	初期化情報(コマンド入力待ち状態)
[Data]ウィンドウ	アドレス0からのデータメモリの内容
[Register]ウィンドウ	現在のレジスタ値
[Source]ウィンドウ	プログラムメモリアドレス0x0100からのプログラムコード 前回設定されていた表示モード(逆アセンブル、ソースまたはミックス)に設定
[Trace]ウィンドウ	空白

8.3.5 終了方法

デバッガを終了するには[File]メニューから[Exit]を選択してください。

[Command]ウィンドウ上でqコマンドを入力することによっても終了します。

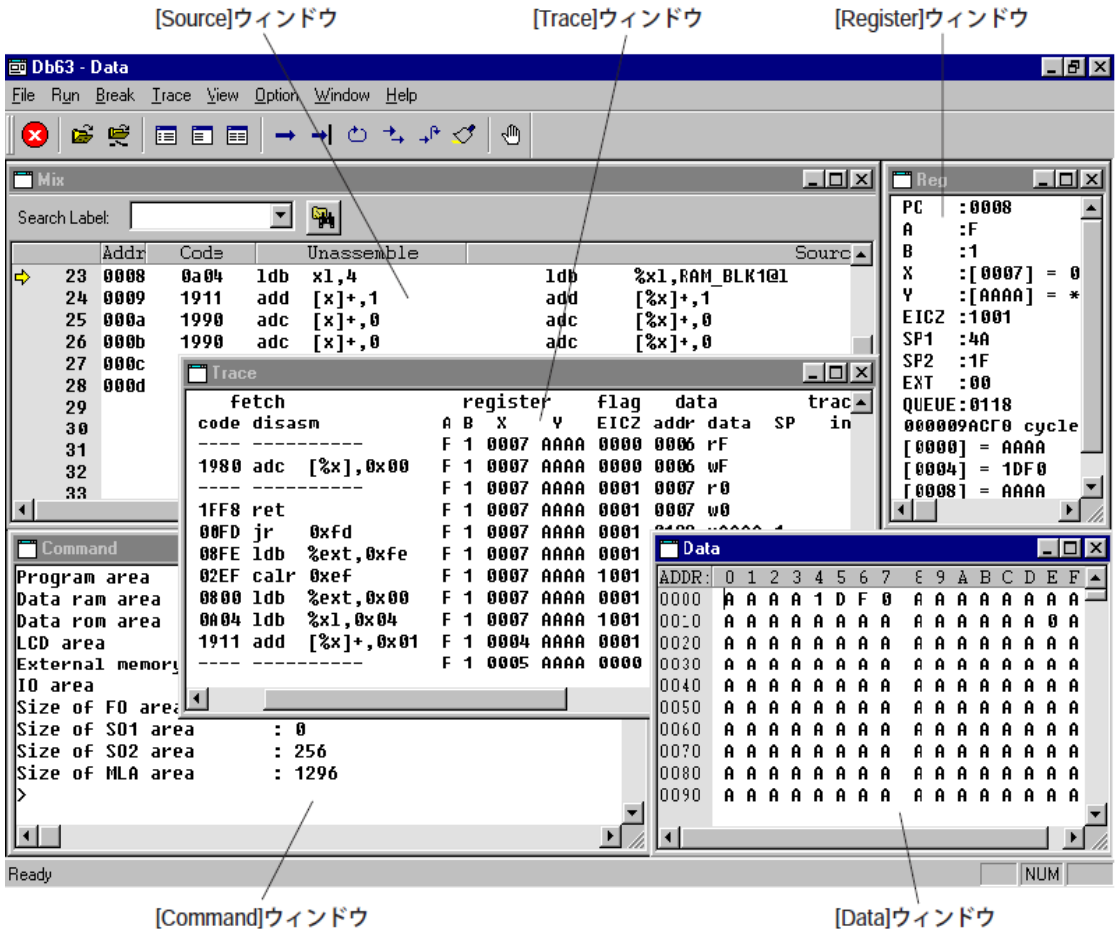
```
>q
```

8.4 ウィンドウ

ここでは、デバッガで使用するウィンドウの種類を説明します。

8.4.1 ウィンドウの基本構成

デバッガのウィンドウ構成は次のとおりです。



画面の解像度、システムフォントのドット数等によって上記とは表示は変わります。各ウィンドウのサイズは使いやすいように調整してください。

●全ウィンドウの共通な操作

(1) ウィンドウのオープン/クローズとアクティブ化

[Command]以外のウィンドウはすべて閉じることと開くことができます。
ウィンドウを開くには、[View]メニューからそのウィンドウ名を選択してください。
ウィンドウを閉じるには、ウィンドウの[閉じる]ボタンをクリックしてください。
デバッガの初期化終了後、デバッガウィンドウは前回の終了時と同じウィンドウ構成(位置とサイズを含む)で開きます。
開いているウィンドウは[Window]メニューにリストされます。そこからウィンドウ名を選択することで、そのウィンドウがアクティブになります。ウィンドウ上をクリックすることでも同様です。また、[Ctrl]+[Tab]のキー操作によってもアクティブウィンドウの切り替えが行えます。

(2) サイズの変更と移動

それぞれのウィンドウサイズは、ウィンドウの境界をドラッグすることによって任意の大きさに変更できます。 [最小化]ボタン、[最大化]ボタン等も一般のWindowsアプリケーションと同様です。各ウィンドウはタイトルバーをドラッグすることによって、表示位置を変更できます。ただし、サイズ変更、移動ともに、アプリケーションウィンドウの範囲内に限られます。

(3) スクロール

各ウィンドウはすべてスクロールできるようになっています。([Register]ウィンドウはサイズを小さくした場合のみ)スクロールは次の3つの方法のいずれかによって行います。

1. 矢印ボタンのクリック、または矢印キーの入力(カーソルの移動)による1行単位のスクロール
2. スクロールバーのクリックによるページ(現在のウィンドウサイズ)単位のスクロール
3. スクロールバーハンドルのドラッグによる任意のエリアへの移動

(4) その他

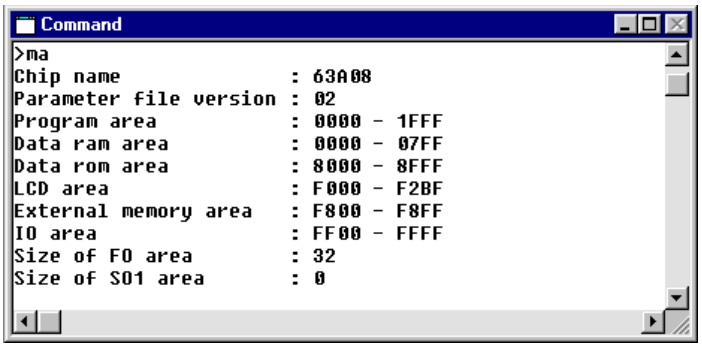
開いているウィンドウは、[Window]メニューの[Cascade]または[Tile]を選択することで整列させることができます。

●表示に関する注意事項

ビデオカードやビデオドライバとOSの互換性等の問題で、ウィンドウに不正な内容が表示されることがあります。問題が発生する場合は、以下の解決方法を試してください。

- 最新版のドライバに更新してください。(古いバージョンがインストールされている場合)
  ドライバのバージョンについては、ビデオカードのメーカーにお問い合わせください。
- コントロールパネルでドライバのアクセラレート機能が選択可能な場合は、OFFにしてください。
- 上記で解決されない場合は、Windowsに標準添付のドライバを使用してください。

8.4.2 [Command]ウィンドウ



[Command]ウィンドウは以下の目的に使用します。

(1) デバッグコマンドの入力

[Command]ウィンドウにプロンプト">"が表示され、キーボードからのコマンド入力を受け付けます。他のウィンドウが選択されているときは、[Command]ウィンドウをクリックすることによりプロンプトの後ろにカーソルが点滅し、コマンドが入力可能な状態となります。(8.7.1項「コマンドのキーボード入力」参照)

(2) メニュー/ツールバーから選択したコマンドの表示

メニューやツールバーからデバッグコマンドを選択して実行させた場合は、そのコマンドラインが[Command]ウィンドウに表示されます。

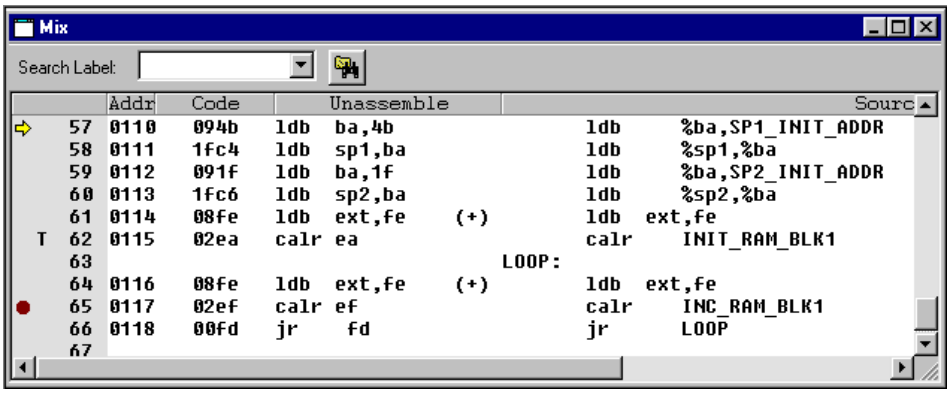
(3) コマンド実行結果の表示

コマンドの実行結果を表示します。ただし、コマンド実行結果の中には、[Source]ウィンドウ、[Data]ウィンドウ、[Register]ウィンドウ、[Trace]ウィンドウに表示される内容もあります。これらの内容は対応するウィンドウが開いていれば、その中に表示されます。ウィンドウが閉じている場合は、[Command]ウィンドウに表示されます。

ログファイルへの書き込みを設定中は、その書き込み内容が表示されます。(logコマンド参照)

注: [Command]ウィンドウを閉じることはできません。

8.4.3 [Source]ウィンドウ



[Source]ウィンドウは、次の(1)～(4)の内容を表示します。また、ブレークポイントの設定や、文字列/ラベルの検索もウィンドウ上で行えます。

(1) プログラムの逆アセンブルおよびソースコード

3種類の表示形式が選択できます。


[Mix]ボタン

1. ミックス表示モード([Mix]ボタンまたはmコマンド入力)
   アドレス、コード、逆アセンブル内容、対応するソース行番号とソースを表示します。(上図)


[Source]ボタン

2. ソース表示モード([Source]ボタンまたはscコマンド入力)
   対応するソース行番号とソースを表示します。


[Unassemble]ボタン

3. 逆アセンブル表示モード([Unassemble]ボタンまたはuコマンド入力)
   デバッガ起動時の表示形式で、アドレス、コード、逆アセンブル内容を表示します。

注: [Source]ウィンドウが開いている場合、m、sc、uコマンドは表示内容を更新します。[Source]ウィンドウが閉じていた場合。プログラムコードは[Command]ウィンドウに表示されます。

64Kのアドレス空間すべてのプログラムコードをスクロールさせて参照できます。ブレーク時は次に実行されるアドレスの行が表示されるように表示内容が更新され、先頭に"矢印"を表示します。
スクロールはスクロールバーまたは矢印キーによって行います。コマンドによって指定位置から表示させることもできます。

※ソース行番号とソースの表示

ソース行番号とソースは、ソース表示用のデバッグ情報を含むIEEE-695形式のアブソリュートオブジェクトファイルを読み込んだ場合に表示可能です。その中でも、アセンブラで-gオプションを指定したソースのみが表示されます。

※表示の更新

プログラムをロードして実行(g、gr、s、n、rstコマンド)するか、プログラムメモリの内容を変更(a(as)、pe、pf、pmコマンド)すると表示内容が更新されます。この場合、現在のPCが示すアドレスが[Source]ウィンドウ内に表示されるように表示が更新されます。また、表示形式を変更しても更新されます。

(2) カレントPC
現在のPC(プログラムカウンタ)が示すアドレスの行は先頭に"矢印"を表示します。(図のアドレス0x0110)

(3) PCブレークポイント
ブレークポイントに設定されたアドレスの行は、先頭に赤の"●"マークを表示します。(図のアドレス0x0117)

(4) トレーストリガポイント
トレーストリガポイントに設定されたアドレスの行は、先頭に"T"を表示します。(図のアドレス0x0115)

(5) カーソル位置でブレーク設定
ブレークポイントを設定したいアドレスの行(ソースのみの行は不可)にカーソルを置きます。そこで、

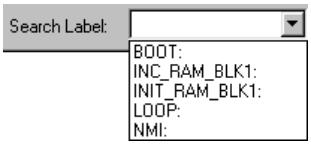

[Break]ボタン

[Break]ボタンをクリックすると、そのアドレスがPCブレークポイントに設定されます。PCブレークポイントに設定されたアドレスの行で同じ操作をすると、そのブレークポイントは解除されます。


[Go to Cursor]ボタン

[Go to Cursor]ボタンをクリックすると、プログラムが現在のPCから実行を開始し、カーソルのある行の実行後にブレークします。

(6) ラベルと文字列の検索

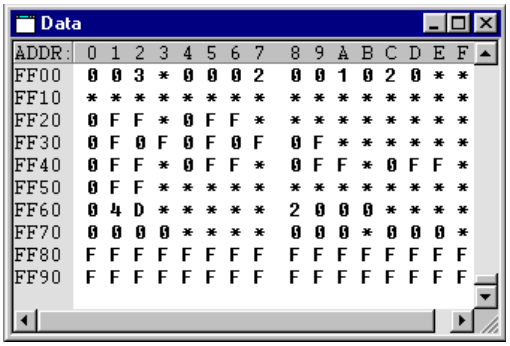


[Search Label]プルダウンボックス

[Source]ウィンドウ上の[Search Label]プルダウンボックスで、ソース内に定義されているラベル位置に表示を移動することができます。
また、[Find]ボタンで任意の文字列をソース内で検索することができます。


[Find]ボタン

8.4.4 [Data]ウィンドウ



(1) メモリ内容の表示

データメモリのダンプ結果を16進数で表示します。表示領域は64Kワードのデータメモリ全体(RAM、データROM、I/O)で、0x0000から0xffffまですべてのアドレスの内容をスクロールにより表示可能です。各機種のマップされないアドレスの内容は"*"となります。

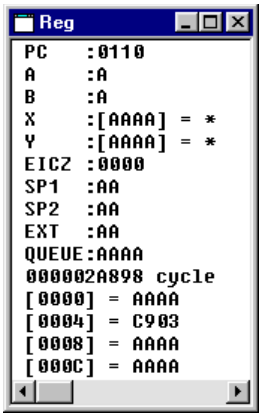
※表示の更新

メモリ内容をコマンド(de、df、dmコマンド)で変更すると、[Data]ウィンドウの表示内容が更新されます。また、プログラムを実行(g、gr、s、n、rstコマンド)した場合も更新されます。これ以外で、最新の内容を表示させるには、ddコマンドを実行するか、垂直スクロールバーをクリックしてください。
プログラムの実行を中断すると、実行前から変更された値は赤で表示されます。

(2) データメモリ内容の直接変更

[Data]ウィンドウ上で、データメモリを直接変更することができます。変更するデータの直前にカーソルを置くか、データをダブルクリック後、16進の数値(0～9、a～f)を入力してください。そのアドレスのデータが変更されます。カーソルは次のアドレスのデータに移動し、連続的なデータ変更を可能にしています。

8.4.5 [Register]ウィンドウ



(1) レジスタ内容

PC、Aレジスタ、Bレジスタ、Xレジスタとそれが示すメモリ、Yレジスタとそれが示すメモリ、フラグ(E、I、C、Z)、スタックポインタ(SP1、SP2)、EXTレジスタ、QUEUEレジスタの内容を表示します。

(2) 実行サイクルカウンタ

CPUがリセットされてからの実行サイクル数または実行時間を積算し、表示します。

(3) 監視データ

本デバッガでは、RAM上の4ヵ所のアドレスを指定して、その内容を監視することができます。
この4つの監視データアドレス(指定アドレスから4ワードずつ)の内容を表示します。起動時は0、4、8、C番地が監視データアドレスとして初期設定されます。左からアドレス順に並んでいます。

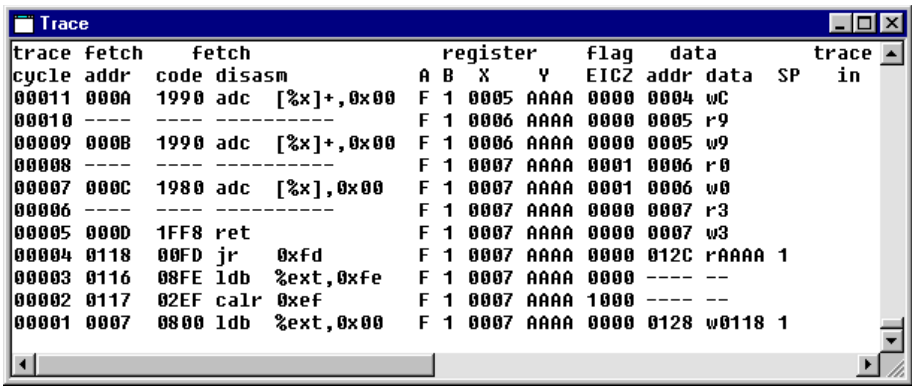
※表示の更新

レジスタダンプ時(rdコマンド)、監視データアドレス設定時(dwコマンド)、レジスタデータ変更時(rsコマンド)、CPUリセット時(rstコマンド)、プログラムの実行(g、gr、s、nコマンド)終了後に更新されます。
オンザフライ機能を有効に設定してプログラムを実行中は、0.5秒間隔でPC、フラグ、監視データの表示内容がリアルタイムに更新されます。他の内容はブレークするまで空白となります。
プログラムの実行を中断すると、実行前から変更された値は赤で表示されます。

(4) レジスタ内容の直接変更

[Register]ウィンドウ上で、レジスタの内容を直接変更することができます。変更するデータを選択(ハイライト)後、16進の数値(0～9、a～f)を入力し[Enter]キーを押してください。そのレジスタの内容が変更されます。

8.4.6 [Trace]ウィンドウ



[Trace]ウィンドウは、ICEのトレースメモリから最大8192サイクルのトレース結果を読み出して表示します。表示されるトレース内容は、次のとおりです。

- トレースサイクル数
- フェッチアドレス
- フェッチコードと逆アセンブル内容
- レジスタの内容(A、B、X、Yおよびフラグ)
- メモリのアクセス内容(アドレス、R/W、データおよびSP1/SP2)
- TRCIN端子の入力状態

[Trace]ウィンドウは、tsコマンドによるトレースデータの検索結果の表示にも使用します。

※表示の更新

ターゲットプログラムの実行により、[Trace]ウィンドウの内容はクリアされます。プログラム実行中は、[Trace]ウィンドウのスクロールやサイズ変更操作は受け付けられません。

プログラムの実行を中断すると、[Trace]ウィンドウは実行中にトレースした最新のデータを表示します。指定サイクルから表示させるには、tdコマンドを使用します。

8.5 ツールバー

ここでは、ツールバーの概要を説明します。

8.5.1 ツールバーの構成

デバッガのツールバーには14個のボタンがあり、頻繁に使用するコマンドが設定されています。

クリックすることにより、指定の機能を実行します。

8.5.2 [Key Break]ボタン

ターゲットプログラムの実行を強制的にブレークします。この機能はプログラムが永久ループ状態になった場合などにブレークさせることができます。

8.5.3 [Load File]、[Load Option]ボタン

[Load File]ボタン

IEEE-695形式のオブジェクトファイルを読み込みます。この選択はlfコマンドを実行するのと同じ働きがあります。

[Load Option]ボタン

モトローラS形式のプログラムファイル、データROM用のデータファイル、オプションHEXファイルを読み込みます。この選択はloコマンドを実行するのと同じ働きがあります。

8.5.4 [Source]、[Mix]、[Unassemble]ボタン

これらのボタンは、[Source]ウィンドウを開き表示モードを切り換えます。

[Source]ボタン

[Source]ウィンドウをソース表示モードに切り換えます。[Source]ウィンドウが閉じている場合は開きます。scコマンドを実行したのと同じ機能があります。

[Unassemble]ボタン

[Source]ウィンドウを逆アセンブル表示モードに切り換えます。[Source]ウィンドウが閉じている場合は開きます。uコマンドを実行したのと同じ機能があります。

[Mix]ボタン

[Source]ウィンドウの表示をミックス(逆アセンブル&ソース)表示モードに切り換えます。[Source]ウィンドウが閉じている場合は開きます。mコマンドを実行したのと同じ機能があります。

8.5.5 [Go]、[Go to Cursor]、[Go from Reset]、[Step]、[Next]、[Reset]ボタン

[Go]ボタン

現在のPC(プログラムカウンタ)からターゲットプログラムを実行します。この選択はgコマンドを実行するのと同じ働きがあります。

[Go to Cursor]ボタン

現在のPCから、[Source]ウィンドウのカーソル位置(その行のアドレス)までターゲットプログラムを実行します。この選択はg <address>コマンドを実行するのと同じ働きがあります。
このボタンを選択するには、[Source]ウィンドウを開き、ブレークするアドレスの行をクリックしておく必要があります。なお、クリックによるブレークアドレスの選択は、実コードのある行のみ有効で、ソースのみの行に対しては無効です。

[Go from Reset]ボタン
CPUをリセット後、プログラム開始アドレス(0x110)からターゲットプログラムを実行します。この選択はgrコマンドを実行するのと同じ働きがあります。

[Step]ボタン
現在のPCからターゲットプログラムを1ステップ実行します。この選択はsコマンドを実行するのと同じ働きがあります。

[Next]ボタン
現在のPCからターゲットプログラムを1ステップ実行します。実行命令がcalr、calz、int命令の場合は、次のアドレスにリターンするまでを1ステップとみなし、それらのサブルーチンのステップをすべて実行します。この選択はnコマンドを実行するのと同じ働きがあります。

[Reset]ボタン
CPUをリセットします。この選択はrstコマンドを実行するのと同じ働きがあります。

8.5.6 [Break]ボタン

[Source]ウィンドウ上のカーソルが置かれたアドレスに対し、ブレークポイントの設定と解除を行うのに使用します。[Source]ウィンドウが開いているときのみ有効です。なお、クリックによるブレークアドレスの選択は、実コードのある行のみ有効で、ソースのみの行に対しては無効です。

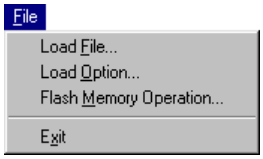
8.6 メニュー

ここでは、メニューバーの概要を説明します。

8.6.1 メニューの構成

デバッガのメニューバーには8つのメニュー項目があり、頻繁に使用するコマンドが設定されています。

8.6.2 [File]メニュー



[Load File...]

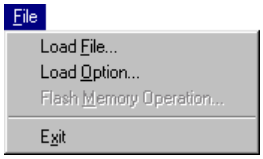
IEEE-695形式のアブソリュートオブジェクトファイルを読み込みます。この選択はlfコマンドを実行するのと同じ働きがあります。

[Load Option...]

モトローラS形式のプログラムファイル、データROM用のデータファイル、オプションHEXファイルを読み込みます。この選択はloコマンドを実行するのと同じ働きがあります。

[Flash Memory Option...]

フラッシュメモリの読み出し、書き込み、消去を行います。この選択はlfl、sfl、eflコマンドを実行するのと同じ働きがあります。

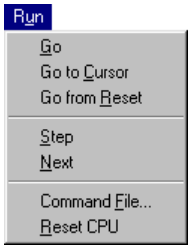


注: USBインタフェース版ICEを使用する場合、フラッシュメモリ操作コマンドは使用できません。左の図のように選択できなくなります。

[Exit]

デバッガを終了します。この選択はqコマンドを実行するのと同じ働きがあります。

8.6.3 [Run]メニュー



[Go]

現在のPC(プログラムカウンタ)からターゲットプログラムを実行します。この選択はgコマンドを実行するのと同じ働きがあります。

[Go to Cursor]

現在のPCから、[Source]ウィンドウのカーソル位置(その行のアドレス)までターゲットプログラムを実行します。この選択はg <address>コマンドを実行するのと同じ働きがあります。このメニュー項目を選択するには、[Source]ウィンドウを開き、ブレークするアドレスの行をクリックしておく必要があります。なお、クリックによるブレークアドレスの選択は、実コードのある行のみ有効で、ソースのみの行に対しては無効です。

[Go from Reset]

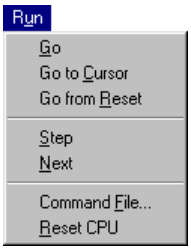
CPUをリセット後、プログラム開始アドレス(0x110)からターゲットプログラムを実行します。この選択はgrコマンドを実行するのと同じ働きがあります。

[Step]

現在のPCからターゲットプログラムを1ステップ実行します。この選択はsコマンドを実行するのと同じ働きがあります。

[Next]

現在のPCからターゲットプログラムを1ステップ実行します。実行命令がcalr、calz、int命令の場合は、次のアドレスにリターンするまでを1ステップとみなし、それらのサブルーチンのステップをすべて実行します。この選択はnコマンドを実行するのと同じ働きがあります。



[Command File...]
コマンドファイルを読み込み、記述されているコマンドを実行します。この選択はcom、cowコマンドを実行するのと同じ働きがあります。

[Reset CPU]
CPUをリセットします。この選択はrstコマンドを実行するのと同じ働きがあります。

8.6.4 [Break]メニュー

[Breakpoint Set...]
PCブレークポイントをダイアログボックスを使用して設定/解除します。この選択はbpコマンドを実行するのと同じ働きがあります。

[Data Break...]
データブレーク条件をダイアログボックスを使用して設定/解除します。この選択はbdコマンドを実行するのと同じ働きがあります。

[Register Break...]
レジスタブレーク条件をダイアログボックスを使用して設定/解除します。この選択はbrコマンドを実行するのと同じ働きがあります。

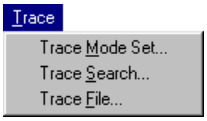
[Sequential Break...]
シーケンシャルブレーク条件をダイアログボックスを使用して設定/解除します。この選択はbsコマンドを実行するのと同じ働きがあります。

[Stack Break...]
スタックブレーク条件をダイアログボックスを使用して設定します。この選択はbspコマンドを実行するのと同じ働きがあります。

[Break List]
設定されているすべてのブレーク条件を表示します。この選択はblコマンドを実行するのと同じ働きがあります。

[Break All Clear]
すべてのブレーク条件を解除します。この選択はbacコマンドを実行するのと同じ働きがあります。

8.6.5 [Trace]メニュー

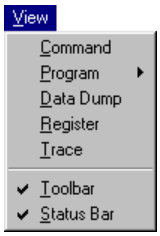


[Trace Mode Set...]
トレースモードおよびトレース条件をダイアログボックスを使用して設定します。この選択はtmコマンドを実行するのと同じ働きがあります。

[Trace Search...]
トレースメモリ内のトレース情報を検索します。検索条件はダイアログボックスで指定します。この選択はtsコマンドを実行するのと同じ働きがあります。

[Trace File...]
[Trace]ウィンドウに表示したトレース情報の指定範囲をファイルに保存します。この選択はtfコマンドを実行するのと同じ働きがあります。

8.6.6 [View]メニュー



[Command]

[Command]ウィンドウをアクティブにします。

[Program]

[Source]ウィンドウを開いてアクティブにします。[Source]ウィンドウはプログラムを現在のPCアドレスから、サブメニューで選択した表示モードで表示します。この選択はu、sc、mコマンドを実行するのと同じ働きがあります。

[Data Dump]

[Data]ウィンドウを開いてアクティブにします。[Data]ウィンドウはデータメモリの内容をメモリの先頭から表示します。

[Register]

[Register]ウィンドウを開いてアクティブにします。[Register]ウィンドウは各レジスタの内容を表示します。

[Trace]

[Trace]ウィンドウを開いてアクティブにします。[Trace]ウィンドウはICEトレースメモリの内容を表示します。

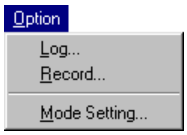
[Toolbar]

ツールバーの表示/非表示を切り換えます。

[Status Bar]

ステータスバーの表示/非表示を切り換えます。

8.6.7 [Option]メニュー



[Log...]

ログ出力のON/OFFを切り換えます。この選択はlogコマンドを実行するのと同じ働きがあります。

[Record...]

実行コマンドのファイルへの記録を制御します。この選択はrecコマンドを実行するのと同じ働きがあります。

[Mode Setting...]

オンザフライ、ブレークモード、実行カウンタのモードを設定します。この選択はmdコマンドを実行するのと同じ働きがあります。

8.6.8 [Window]メニュー

[Cascade]

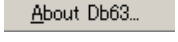
開いているウィンドウを斜めに整列させます。

[Tile]

開いているウィンドウを縦横に整列させます。

このメニューには、現在開いているウィンドウ名が表示されます。いずれかを選択すると、そのウィンドウがアクティブになります。

8.6.9 [Help]メニュー



[About Db63...]

デバッガのアバウトダイアログボックスを表示します。

8.7 コマンド実行方法

デバッグ機能はすべてデバッグコマンドによって実行できます。ここでは、コマンドを実行させる方法を説明します。コマンドのパラメータなど、詳細については各コマンド説明を参照してください。

デバッグコマンドを実行させるには、[Command]ウィンドウをアクティブにし、キーボードからコマンドを入力して実行させます。頻繁に使用するコマンドはメニュー、およびツールバーから実行させることもできます。

8.7.1 コマンドのキーボード入力

[Command]ウィンドウを選択してください([Command]ウィンドウ上をクリック)。その中の最終行にプロンプト">"が表示され、その後にカーソルが点滅していればコマンドが入力できる状態にあります。
そこに、デバッグコマンドを入力してください。コマンドは大文字でも、小文字でも受け付けます。

●コマンド入力の一般形

>コマンド [パラメータ [　パラメータ...　パラメータ]] ↲

・コマンドとパラメータ間にはスペースが必要です。
・パラメータ間にはスペースが必要です。

入力ミスの修正には矢印キー、[Back Space]キー、[Delete]キーが使用できます。
最後に[Enter]キーを入力すると、そのコマンドを実行します(ガイダンス付きのコマンドは、表示に従って必要なデータを入力した時点で実行されます)。

入力例:

```
>g↲                    (コマンドのみの入力)
>com test.cmd↲         (コマンドとパラメータの入力)
```

●ガイダンス付きのコマンド入力

パラメータが指定されないと実行できないコマンドや、既存のデータを変更するコマンドはコマンドのみの入力でガイダンスモードとなります。ガイダンスが表示されますので、そこにパラメータを入力してください。

入力例:

```
>lf↲
File name    ? test.abs↲        ...ガイダンスに従ってデータ(アンダーライン部)を入力
>
```

- パラメータ入力が必須のコマンド
  上記例のlfコマンドはアブソリュートオブジェクトファイルを読み込むコマンドです。このような、パラメータを持つコマンドでは、次のガイダンスが表示されますので、最後のパラメータまで順次入力してください。いずれかのガイダンスで[Enter]キーのみを入力すると、そのコマンドはキャンセルされ実行されません。

• 既存のデータを確認して置き換えるコマンド
メモリやレジスタを1つずつ書き換えるコマンドではガイダンスをスキップ(その内容を変更しない)、1つ前のガイダンスに戻す、および途中で入力を終了することができます。

[Enter]キー ... 入力をスキップ
[^]キー 1つ前のガイダンスに戻す
[q]キー ..... 入力を終了する

入力例:

```
>de↵                              ...データメモリを変更するコマンド
Data enter address ? :0↵          ...開始アドレスを入力
0000   A:1↵                       ...0x0000番地を1に変更
0001   A:^↵                       ...1つ前のアドレスに戻す
0000   1:0↵                       ...0x0000番地の入力をし直す
0001   A:↵                        ...[Enter]キーのみの入力で0x0001番地をスキップ
0002   A:↵
0001   A:q↵                       ...入力を終了
>
```

●パラメータの数値データ形式

パラメータとして入力する数値は、ほとんどのコマンドが16進数のみを受け付けます。
ただし、一部のコマンドのパラメータは、10進数または2進数で指定します。

数値として有効な文字は次のとおりです。
16進数: 0〜9、a〜f、A〜F、*
10進数: 0〜9
2進数: 0、1、*
(*はデータパターン指定でマスクするビットに使用します。)

●シンボルによる指定

アドレスを指定する場合、ソース内で定義されているシンボル(ラベル)を使用することができます。その場合、ロードしたアブソリュートオブジェクトファイルがデバッグ情報を含んでいる必要があります。シンボルは、次のように使用します。

グローバルシンボル: @<シンボル名> 例: `@RAM_BLK1`
ローカルシンボル: @<シンボル名>@<ソースファイル名> 例: `@LOOP@main.s`

パラメータの入力例については、各コマンドの説明を参照してください。

●[Enter]キーによる連続実行

以下に示すコマンドは、一度実行した後に[Enter]キーのみを入力することにより、前回と同じ実行を繰り返す、あるいは前回の表示内容に引き続いて表示を行うことができます。

実行系: g (go)、s (step)、n (next)、com (execute command file)
表示系: sc (source)、m (mix)、u (unassemble)、dd (data memory dump)、od (option data dump)、td (trace data display)、cv (coverage)、sy (symbol list)、ma (map information)

他のコマンドを実行すると、連続実行機能は終了します。

8.7.2 メニュー、ツールバーからの実行

8.5項、8.6項に示したように、メニューとツールバーには頻繁に使用するコマンドが登録されています。メニューコマンドを選択するか、ツールバーボタンをクリックするだけで、指定のコマンドを実行できます。表8.7.2.1に登録されているコマンドの一覧を示します。

表8.7.2.1 メニュー、ツールバーで指定可能なコマンド

コマンド	機能	メニュー	ボタン
lf	IEEE-695オブジェクトファイルのロード	[File ¦ Load File...]	
lo	モトローラSファイルのロード	[File ¦ Load Option...]	
g	プログラムの連続実行	[Run ¦ Go]	
g <address>	プログラムを<address>まで連続実行	[Run ¦ Go to Cursor]	
gr	CPUリセット後、プログラムの連続実行	[Run ¦ Go from Reset]	
s	ステップ実行	[Run ¦ Step]	
n	ステップ&スキップ	[Run ¦ Next]	
com, cmw	コマンドファイルのロード、実行	[Run ¦ Command File...]	–
rst	CPUリセット	[Run ¦ Reset CPU]	
bp, bc (bpc)	PCブレークポイント設定/解除	[Break ¦ Breakpoint Set...]	
bd, bdc	データブレーク条件の設定/解除	[Break ¦ Data Break...]	–
br, brc	レジスタブレーク条件の設定/解除	[Break ¦ Register Break...]	–
bs, bsc	シーケンシャルブレーク条件の設定/解除	[Break ¦ Sequential Break...]	–
bsp	スタック領域指定	[Break ¦ Stack Break...]	–
bl	全ブレーク条件の表示	[Break ¦ Break List...]	–
bac	全ブレーク条件の解除	[Break ¦ Break All Clear]	–
tm	トレースモードの設定	[Trace ¦ Trace Mode Set...]	–
ts	トレース情報の検索	[Trace ¦ Trace Search...]	–
tf	トレース情報のファイルへの保存	[Trace ¦ Trace File...]	–
u	逆アセンブル表示	[View ¦ Program ¦ Unassemble]	
sc	ソース表示	[View ¦ Program ¦ Source Display]	
m	ミックス表示	[View ¦ Program ¦ Mix Mode]	
lfl *	フラッシュメモリの読み出し	[Flash ¦ Flash Memory Operation...]	–
sfl *	フラッシュメモリへの書き込み	[Flash ¦ Flash Memory Operation...]	–
efl *	フラッシュメモリ消去	[Flash ¦ Flash Memory Operation...]	–
dd	データメモリダンプ	[View ¦ Data Dump]	–
rd	レジスタ値の表示	[View ¦ Register]	–
td	トレース情報の表示	[View ¦ Trace]	–
log	ログ出力ON/OFF	[Option ¦ Log...]	–
rec	実行コマンドの記録	[Option ¦ Record...]	–
md	モード設定	[Option ¦ Mode Setting...]	–

* USBインタフェース版ICEを使用する場合、フラッシュメモリ操作コマンド(lfl、sfl、efl)は使用できません。

8.7.3 コマンドファイルによる実行

一連のデバッグコマンドを記述したコマンドファイルを読み込んで、それらのコマンドを実行させることができます。

●コマンドファイルの作成

コマンドファイルはエディタでテキストファイルとして作成してください。
ファイル名の拡張子は特に制限していませんが、".cmd"を推奨します。

コマンドファイルは、recコマンドによっても作成できます。recコマンドを使用すると、デバッガで実際に実行したコマンドをコマンドファイルに記録することができます。

●コマンドファイル例

次の例は、オブジェクトファイルとオプションファイルを読み込むためのコマンド群です。

例: ファイル名=startup.cmd

```
lf test.abs
lo testf.fsa
lo tests.ssa
```

コマンドファイルにはガイダンスモードに対応した記述も可能です。その場合は、ガイダンス入力の各項目ごとに改行して記述してください。

●コマンドファイルの読み込み/実行

コマンドファイルを読み込み、実行させる方法は次の2通りです。

(1) 起動時オプションによる実行

デバッガの起動コマンドでコマンドファイルを指定すると、そのコマンドファイル(1個のみ)をデバッガの起動時に実行させることができます。
たとえば上記例のコマンドファイルを指定すると、デバッガ起動直後に必要なファイルが読み込まれ、すぐにデバッグが行える状態になります。

例: デバッガの起動コマンド

```
db63 startup.cmd par63xxx.par
```

(2) コマンドによる実行

コマンドファイルの実行用に、comコマンドとcmwコマンドが用意されています。
comコマンドは指定されたファイルを読み込み、その中のコマンドを記述順に実行します。
cmwコマンドも同様ですが、個々のコマンドはmdコマンドで指定された間隔(1～256秒)で実行されます。

例: `com startup.cmd`

```
cmw test.cmd
```

コマンドファイルに記述されたコマンドは[Command]ウィンドウに表示されます。

●制限事項

コマンドファイル内から別のコマンドファイルを読み込むことも可能です。ただし、最大5階層までに制限されます。6階層目のcomまたはcmwコマンドが現れるとエラーとなり、それ以後の実行を中止します。

8.7.4 ログファイル

実行したコマンドと実行結果を、テキスト形式のログファイルとして保存することができます。これによって、後からデバッグの手順と内容を確認することができます。
保存の対象となるのは[Command]ウィンドウに表示された内容です。

●コマンド例

```
>log tst.log
```

logコマンドでログモードに設定後(出力開始後)は、logコマンドがトグル動作(ログモード/出力ON↔通常モード/出力OFF)となりますので、必要な部分のみの出力が簡単に行えます。

●ログモードでの[Command]ウィンドウの表示

ログモードでは、[Command]ウィンドウに表示される内容が通常の場合と異なります。

(1)各ウィンドウが開いている場合のコマンド実行時

(ウィンドウに結果が表示されるコマンドをそのウィンドウが開いている状態で実行した場合)

通常モード: 表示先のウィンドウの内容が更新されます。[Command]ウィンドウに実行結果は表示されません。

ログモード: ウィンドウに表示される情報と同等の内容が[Command]ウィンドウにも表示されます。ただし、ウィンドウのスクロール操作、ウィンドウを開いたことによる表示内容は、[Command]ウィンドウには表示されません。

(2)各ウィンドウが閉じている場合のコマンド実行時

表示先のウィンドウが閉じている場合、ログモード/通常モードにかかわらず実行結果が[Command]ウィンドウに表示されます。

[Command]ウィンドウの表示形式については、各コマンドの説明を参照してください。

8.8 デバッグ機能

ここでは、デバッグ機能の概要を機能別に説明します。
各デバッグコマンドの詳細については"8.9 コマンドリファレンス"を参照してください。

8.8.1 プログラムとオプションデータの読み込み

●ファイルの種類

デバッガはIEEE-695形式、またはモトローラS形式のファイルを読み込んでデバッグすることができます。表8.8.1.1に読み込み可能なファイルの一覧と読み込みコマンドを示します。

表8.8.1.1 ファイルと読み込みコマンド一覧

ファイル形式	データタイプ	拡張子	生成ツール	コマンド	メニュー	ボタン
IEEE-695	プログラム/データ	.abs	リンカ	**lf**	[File ¦ Load File...]	
モトローラS	プログラム(上位5bit)	.hsa	HEXコンバータ	**lo**	[File ¦ Load Option...]	
	プログラム(下位8bit)	.lsa	HEXコンバータ			
	ファンクションオプション	.fsa	ファンクションオプション ジェネレータ			
	セグメントオプション	.ssa	セグメントオプション ジェネレータ			
	メロディーデータ	.msa	メロディーアセンブラ			

●ソースレベルデバッグを行うためには

ソース表示とシンボルを使用してデバッグを行うためには、IEEE-695形式のオブジェクトファイルを読み込む必要があります。他のプログラムファイルを読み込んだ場合は、逆アセンブル表示のみとなります。

8.8.2 ソース表示およびシンボリックデバッグ機能

デバッガでは、アセンブリソースを表示させてデバッグを行うことができます。また、シンボル名も使用したアドレスの指定も可能です。

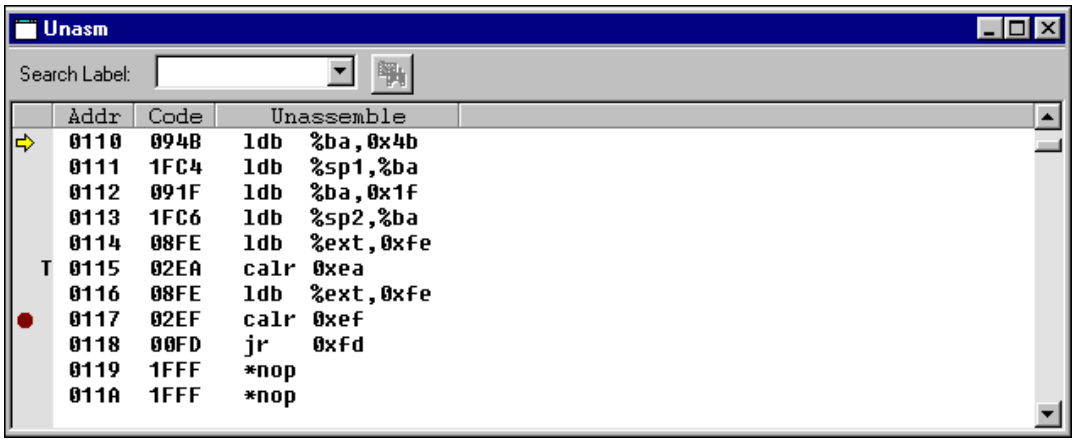
●プログラムコードの表示

[Source]ウィンドウは指定された表示モードでプログラムを表示します。表示モードは、逆アセンブル表示モード、ソース表示モード、ミックス表示モードの3種類から選択できます。

表8.8.2.1 表示モード切り換えコマンド

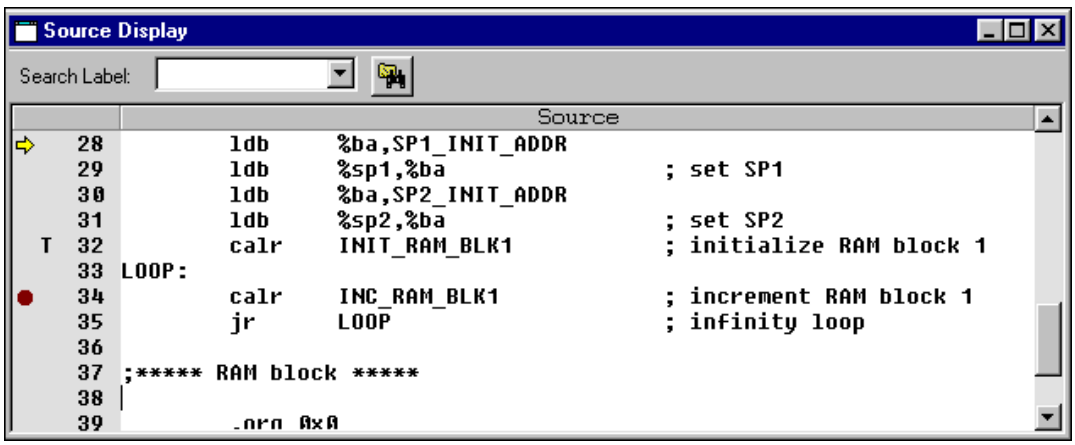
表示モード	コマンド	メニュー	ボタン
逆アセンブル表示モード	**u**	[View ¦ Program ¦ Unassemble]	
ソース表示モード	**sc**	[View ¦ Program ¦ Source Display]	
ミックス表示モード	**m**	[View ¦ Program ¦ Mix Mode]	

(1) 逆アセンブル表示モード



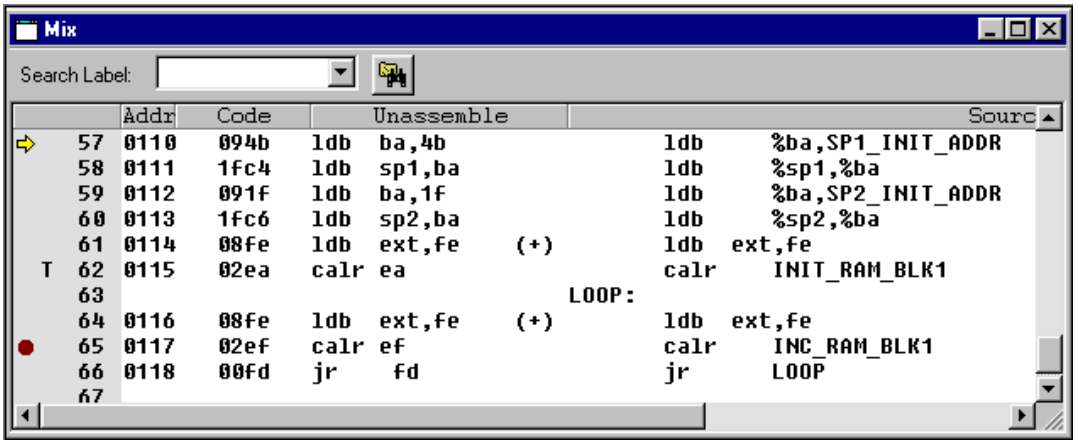
このモードでは、プログラムコードをニーモニックに逆アセンブルして表示します。

(2) ソース表示モード



このモードでは、現在のPCアドレス上のコードを含むソースがエディタと同様の形式で表示されます。このモードは、ソースデバッグ情報を含むアブソリュートオブジェクトファイルをロードしている場合にのみ指定可能です。

(3) ミックス表示モード



このモードでは、プログラムの逆アセンブル内容とソースの両方がアブソリュートリストファイルと同様の形式で表示されます。このモードは、ソースデバッグ情報を含むアブソリュートオブジェクトファイルをロードしている場合にのみ指定可能です。

表示内容については、"8.4.3 [Source]ウィンドウ"を参照してください。

●シンボル参照

IEEE-695形式のオブジェクトファイルを読み込んでデバッグを行う場合、ソースファイルで定義されたシンボルを使用してアドレスを指定することができます。パラメータに<address>を持つコマンドを[Command]ウィンドウ上で入力する際、あるいはアドレスをダイアログで指定する際に使用することができます。

(1) グローバルシンボルの参照

.global擬似命令および.comm擬似命令でグローバル宣言されたシンボル(ラベル)は、次のように指定します。

@<シンボル>

指定例:

```
>m @BOOT
>de @RAM_BLK1
```

(2) ローカルシンボルの参照

定義されたソースファイル内でのみ使用しているローカルシンボル(ラベル)は、次のように指定します。

@<シンボル>@<ファイル名>

ファイル名は、シンボルが定義されているソースファイル名(.s)です。

指定例:

```
>bp @SUB1@test.s
```

(3) シンボルリストの表示

デバッグ中のプログラムで使用しているすべてのシンボルと定義されたアドレスを、[Command]ウィンドウに表示させることができます。

表8.8.2.2 シンボルリスト表示コマンド

機 能	コマンド
シンボルリストの表示	**sy**

8.8.3 プログラム、データ、レジスタの表示と変更

デバッガはオプションデータと同様にプログラムメモリ、データメモリ、レジスタに対する操作機能を持っています。各メモリ領域はパラメータファイルで与えられるマップ情報に従ってデバッガに設定されます。

●プログラムメモリ領域の操作

プログラムメモリ領域に対しては以下の操作が行えます。

表8.8.3.1 プログラムメモリ操作コマンド

機　能	コマンド
コードの入力/変更	**pe**
インラインアセンブル	**a (as)**
指定領域の書き換え	**pf**
指定領域のコピー	**pm**

(1) コードの入力/変更

16進データを入力して、指定アドレスのプログラムコードを書き換えます。

(2) インラインアセンブル

ニーモニックを入力して、指定アドレスのプログラムコードを書き換えます。

(3) 指定領域の書き換え

指定した領域を指定したコードですべて書き換えます。

(4) 指定領域のコピー

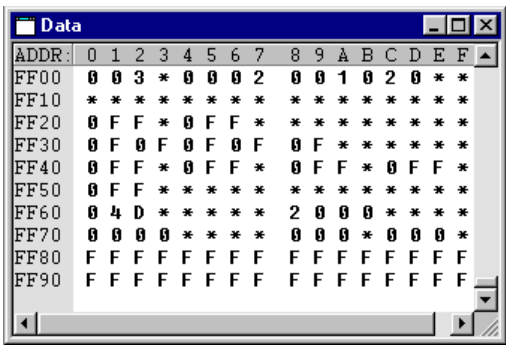
指定した領域の内容を、別の領域にコピーします。

●データメモリ領域の操作

データメモリ領域(RAM、データROM、表示メモリ、I/Oメモリ)に対しては以下の操作が行えます。

表8.8.3.2 データメモリ領域操作コマンド

機　能	コマンド	メニュー
データメモリダンプ	**dd**	[View ¦ Data Dump]
データの入力/変更	**de**	–
指定領域の書き換え	**df**	–
指定領域のコピー	**dm**	–



(1) データメモリダンプ

データメモリの内容を16進ダンプ形式で表示します。[Data]ウィンドウが開いていれば[Data]ウィンドウの内容を更新し、開いていなければ[Command]ウィンドウに表示します。

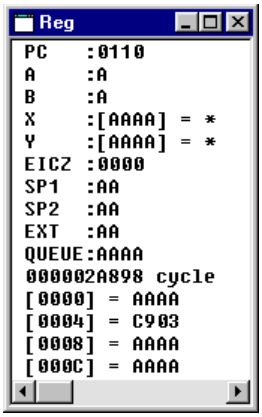
(2) データの入力/変更

16進データを入力して、指定アドレスのデータを書き換えます。[Data]ウィンドウ上で直接変更することもできます。

(3) 指定領域の書き換え
　指定した領域を指定したデータですべて書き換えます。

(4) 指定領域のコピー
　指定した領域の内容を、別の領域にコピーします。

(5) メモリの監視
　連続した4ワード分のメモリ領域を4ヵ所、監視データアドレスとして登録することができます。登録した監視データは[Register]ウィンドウ上で確認できます。オンザフライ機能により0.5秒間隔でその内容がリアルタイムに更新されます。初期設定では、0番地、4番地、8番地、C番地が監視データアドレスとなっています。



表示されているメモリの内容は左端が指定アドレスのデータ、右端が4ワード分上位のアドレスのデータです。

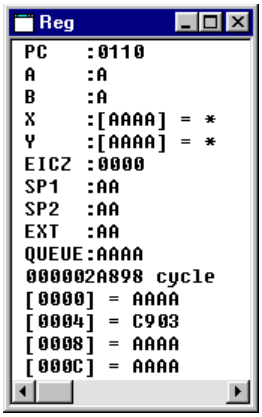
監視データ

●レジスタの操作

　レジスタに対しては以下の操作が行えます。

表8.8.3.3　レジスタ操作コマンド

機　能	コマンド	メニュー
レジスタの表示	**rd**	[View ¦ Register]
レジスタ値の変更	**rs**	–



(1) レジスタの表示
　レジスタの内容は[Register]ウィンドウまたは[Command]ウィンドウに表示させることができます。
　レジスタ: PC、A、B、Xおよび[X]、Yおよび[Y]、F、SP1、SP2、EXT、QUEUE

　プログラム実行中は、オンザフライ機能によってPCアドレスとFレジスタの内容が0.5秒ごとにリアルタイムに更新されます。

(2) レジスタ値の変更
　上記レジスタの内容を任意の値に設定できます。
　レジスタ値は[Register]ウィンドウ上で直接変更することもできます。

●オプションデータの表示

　ICEのオプション領域(ファンクションオプションデータ、セグメントオプションデータ、メロディーデータ)のデータを表示させることができます。データは[Command]ウィンドウに16進ダンプ形式で表示されます。

表8.8.3.4　オプションデータ表示コマンド

機　能	コマンド
オプションデータの表示	**od**

8.8.4 プログラムの実行

デバッガにはターゲットプログラムを連続実行およびシングルステップ実行させる機能があります。

●連続実行

(1) 連続実行の種類

2種類の連続実行機能があります。

- 現在のPCアドレスから連続実行
- CPUをリセット後にプログラム開始アドレス(0x0110)から連続実行

表8.8.4.1 連続実行コマンド

機　能	コマンド	メニュー	ボタン
現在のPCアドレスから連続実行	**g**	[Run ¦ Go]	
		[Run ¦ Go to Cursor]	
CPUをリセット後に連続実行	**gr**	[Run ¦ Go from Reset]	

(2) 連続実行の停止

連続実行コマンド(g <アドレス>)によって、その実行中にのみ有効なテンポラリブレークアドレスを2ヵ所まで指定することができます。

テンポラリブレークアドレスは[Source]ウィンドウで指定することもできます(1ヵ所のみ)。[Source]ウィンドウ上のブレークさせるアドレスの行にカーソルを置いて[Go to Cursor]ボタンをクリックすると、プログラムは現在のPCから実行を開始し、カーソル位置の命令を実行前にブレークします。

このテンポラリブレーク以外では、実行中のプログラムは次のいずれかの要因によってブレークするまで停止しません。

- ブレーク設定コマンドで設定したブレーク条件が成立
- [Key Break]ボタンのクリックまたは[Esc]キーの入力
- マップブレーク等の発生

 [Key Break]ボタン　※プログラムが停止しない場合は、このボタンで強制ブレークさせることができます。

(3) オンザフライ機能

ICEおよびデバッガには、連続実行中にPCアドレス、Fレジスタ、監視データの値を0.5秒(デフォルト)ごとに表示するオンザフライ機能があります。これらの内容は[Register]ウィンドウの該当個所に、[Register]ウィンドウが閉じている場合は[Command]ウィンドウに表示されます。デバッガの初期設定でオンザフライ機能が1秒間に2回となりますが、mdコマンドによってOFF～1秒間に5回まで設定することができます。オンザフライはICEハードウェアにより実現される、完全なリアルタイム表示です。

●シングルステップ実行

(1) シングルステップ実行の種類

2種類のシングルステップ実行機能があります。

・ 全命令をシングルステップ実行(STEP)
命令の種類によらずPCに従ってシングルステップで実行します。

・サブルーチン以外をシングルステップ実行(NEXT)
calr、calz、int命令を、リターン命令で次のステップに戻るまでを1ステップとして実行します。他の命令は、通常のシングルステップ実行と同様です。

どちらの場合も現在のPCから実行します。

表8.8.4.2　シングルステップコマンド

機　能	コマンド	メニュー	ボタン
全命令をシングルステップ実行	**s**	[Run ¦ Step]	
サブルーチン以外をシングルステップ実行	**n**	[Run ¦ Next]	

コマンド入力による実行では、実行するステップ数を最大65,535ステップまで指定することができます。メニューコマンド、ツールバーでは1ステップずつ実行します。
以下の場合には、シングルステップ実行が指定のステップ数の実行前に終了します。
• [Key Break]ボタンのクリックまたは[Esc]キーの入力
• マップブレーク等の発生

PCブレーク、データブレーク等のユーザ設定ブレークでは停止しません。

 [Key Break]ボタン　※プログラムが停止しない場合は、このボタンで強制ブレークさせることができます。

(2) ステップ動作中の表示

デバッガの初期設定では、次のように表示を更新します。
[Register]ウィンドウの表示内容は各ステップごとに更新されます。[Register]ウィンドウが閉じている場合は、その内容が[Command]ウィンドウに表示されます。この表示モードを、指定ステップ数の最終ステップでのみ更新するように、mdコマンドで切り換えることもできます。
[Source]ウィンドウおよび[Data]ウィンドウの表示は指定ステップ数の実行終了時に更新されます。

(3) HALT、SLEEP状態と割り込み

halt命令またはslp命令を実行するとCPUはスタンバイモードとなり、その解除には割り込みが必要です。
デバッガでは、シングルステップ動作用に外部割り込みの許可/禁止モードが設定されています。

表8.8.4.3　外部割り込みモード

	許可モード	禁止モード
外部割り込み	割り込みを処理する	割り込みを処理しない
halt、slp命令	halt命令として実行 外部割り込み、または[Key Break]ボタンで処理を継続	halt、slp命令をnop命令に置き換えて実行

デバッガの初期設定で、割り込み禁止モードに設定されます。mdコマンドによって割り込み許可モードに設定することもできます。

●実行サイクル/実行時間の測定

(1) 実行サイクルカウンタと測定モード

ICEは31ビットの実行サイクルカウンタを内蔵しており、プログラムの実行した時間またはバスサイクル数を測定することができます。mdコマンドによって測定モード(時間、バスサイクル)の選択が可能です。デバッガの初期設定は、バスサイクルモードになります。

実行サイクルカウンタで測定可能な最大値は次のとおりです。

実行時間モード:	2147483647μsec = 約36分	(エラー = ±1μsec)
バスサイクルモード:	2147483647cycle	(エラー = ±0)

(2) 測定結果の表示

測定結果は[Register]ウィンドウに表示されます。プログラム実行中はクリアされ、実行終了後に更新されます。

[Register]ウィンドウが閉じている場合は、rdコマンドによって[Command]ウィンドウに表示することができます。シングルステップの実行結果としても表示されます。

カウンタの最大値を越えた場合は"over flow"を表示します。

(3) ホールドモードとリセットモード

デバッガの初期設定では、実行サイクルカウンタはホールドモードに設定されます。このモードでは、カウンタがリセットされるまで測定値を積算します。

リセットモードはmdコマンドで設定することができ、プログラムの実行ごとにカウンタがリセットされます。

連続実行ではgコマンドの入力によるプログラムの実行開始時にリセットされ、その実行終了(ブレーク)までを測定します。(grコマンド実行時も同様ですが、CPUのリセットによりカウンタもリセットされるため、ホールドモードでもリセットモードと同じ結果となります。)

シングルステップ実行では、 s またはnコマンドの入力による実行開始時にリセットされ、指定ステップ数の実行終了までを測定します。1ステップのみの指定またはツールバーボタン/メニューコマンドによる実行では、1ステップごとにリセットされます。

(4) 実行サイクルカウンタのリセット

実行サイクルカウンタは次の場合にリセットされます。

- rstコマンド、[Run]メニュー/[Reset]、[Reset]ボタンによってCPUをリセットした場合
- grコマンド、[Run]メニュー/[Go after Reset]を実行した場合
- 実行サイクルカウンタのモードをmdコマンドにより切り換えた場合
  (実行時間モード↔バスサイクルモード、ホールドモード↔リセットモード)
- リセットモードでプログラムの実行を開始した場合

●CPUのリセット

CPUはgrコマンドの実行時、およびrstコマンドの実行によりリセットされます。

CPUのリセットによる初期設定内容は以下のとおりです。

(1) CPUの内部レジスタ

PC	...0x0110
A、B	...0xa
X、Y、QUEUE	...0xaaaa
F	...0b0000
SP1、SP2、EXT	...0xaa

(2) 実行サイクルカウンタを0に設定

(3) [Source]ウィンドウ、[Register]ウィンドウを再表示

PCが0x0110に設定されるため、そのアドレスから再表示します。

[Register]ウィンドウを上記の設定で再表示します。

データメモリの内容は変更されません。

8.8.5 ブレーク機能

ターゲットプログラムは次の要因により実行を中断します。
- ブレークコマンドの条件成立
- [Key Break]ボタン
- ICEのBRKIN端子へのLowレベル入力
- マップブレーク等の発生

●コマンドによるブレーク機能

デバッガはコマンドによってブレーク条件が設定できる5種類のブレーク機能を持っています。それぞれ、条件が成立すると実行中のプログラムはブレークします。

(1) PCによるブレーク機能

PCが設定したアドレスに一致するとブレークする機能です。プログラムは、そのアドレスの命令を実行前にブレークします。PCブレークポイントは複数のアドレスに設定できます。

表8.8.5.1 ブレークポイント設定コマンド

機　能	コマンド	メニュー	ボタン
ブレークポイント設定	**bp**	[Break ¦ Breakpoint Set...]	
ブレークポイントの解除	**bc (bpc)**	[Break ¦ Breakpoint Set...]	

PCブレークポイントに設定されたアドレスは、[Source]ウィンドウ上の行の先頭に●が表示されます。[Break]ボタンを使用して簡単にブレークポイントの設定と解除を行うこともできます。
[Source]ウィンドウ上で、ブレークさせるアドレスの行をクリック(カーソルを移動)し、[Break]ボタンをクリックします。その行の先頭に●マークが表示され、ブレークポイントに設定されます。そのアドレスはブレークポイントリストに登録されます。●で始まる行をクリックして[Break]ボタンをクリックすると、ブレーク設定が解除され、アドレスがブレークポイントリストから削除されます。

※ 連続実行コマンド(g)で指定可能なテンポラリブレークアドレスは、ブレークポイントリストに設定されたアドレスには影響を与えません。

(2) データブレーク機能

データブレークは、指定したデータメモリ領域内をアクセスした場合にブレークを発生させる機能です。アクセスを監視するメモリ領域のほかに、読み出し、書き込みのどちらが行われた場合にブレークさせるか、またそのデータの内容まで指定することができます。読み出し/書き込み条件はマスク可能で、どちらが行われた場合でもブレークさせることができます。同様にデータ条件もビット単位のマスクが可能です。
ブレークは上記の条件を満たす動作が行われたサイクルの終了時に発生します。

表8.8.5.2 データブレーク設定コマンド

機　能	コマンド	メニュー
データブレーク条件の設定	**bd**	[Break ¦ Data Break...]
データブレーク条件の解除	**bdc**	[Break ¦ Data Break...]

たとえば、アドレスを0x10、データパターンを*(マスク)、読み出し/書き込み条件を書き込みに設定してプログラムを実行すると、プログラムがデータメモリのアドレス0x10にデータの書き込みを行った場合にブレークが発生します。

(3) レジスタブレーク

レジスタブレークは、A、B、F、XおよびYレジスタがそれぞれ指定した値になった場合にブレークを発生させる機能です。各レジスタはマスク(ブレーク条件に含めない設定)が可能です。Fレジスタはビットごとにマスク可能です。ブレークは上記のレジスタがすべて設定条件を満たすように変更された時点で発生します。

表8.8.5.3 レジスタブレーク設定コマンド

機 能	コマンド	メニュー
レジスタブレーク条件の設定	**br**	[Break ¦ Register Break...]
レジスタブレーク条件の解除	**brc**	[Break ¦ Register Break...]

たとえば、Aレジスタのデータを0、Fレジスタ(Cフラグ = 1)のデータを"**1*"、他をすべてマスクに設定してプログラムを実行させると、Aレジスタが0になり、かつCフラグが1になった時点でブレークします。

(4) シーケンシャルブレーク機能

シーケンシャルブレークでは、3ヵ所までのブレークアドレスと、その中の最後のアドレスの実行回数を設定できます。プログラムは、設定した順序ですべてのアドレスを通過するとともに、最後に指定したアドレスを指定回数実行し、その後もう一度そのアドレスの命令をフェッチするとブレークします。

表8.8.5.4 シーケンシャルブレーク設定コマンド

機 能	コマンド	メニュー
シーケンシャルブレーク条件の設定	**bs**	[Break ¦ Sequential Break...]
シーケンシャルブレーク条件の解除	**bsc**	[Break ¦ Sequential Break...]

たとえば、bsコマンドでアドレスを0x1000番地と0x2000番地の2ヵ所に、実行回数を3回に設定してプログラムを実行させた場合、0x1000番地を1回以上実行した後で0x2000番地を3回実行し、さらにPCが0x2000になると4回目の実行前にブレークします。
実行回数は4095回まで指定できます。

(5) スタック領域外へのアクセス

スタックポインタSP1、SP2によってスタック領域外へのアクセスが行われた場合にブレークします。この機能を使用するためには、bspコマンドでSP1、SP2領域をあらかじめ設定しておく必要があります。初期値はSP1が0x0～0x3ff、SP2が0x0～0xffとなっています。SP1のアドレスは4ワード単位で指定する必要があります。

表8.8.5.5 スタックブレーク設定コマンド

機 能	コマンド	メニュー
スタックブレーク条件の設定	**bsp**	[Break ¦ Stack Break...]

●[Key Break]ボタンまたは[Esc]キーによる強制ブレーク

[Key Break]ボタンおよび[Esc]キーは、プログラムが永久ループまたはスタンバイ(HALT、SLEEP)状態から抜け出せない場合など、実行中のプログラムを強制的に終了させます。

 [Key Break]ボタン

●ICEのBRKIN端子へのLowレベル入力

ICEのBRKIN端子にLowレベルのパルス(20nsec以上)が入力されるとプログラムはブレークします。

●マップブレーク、不当命令ブレーク

プログラム実行中に以下のエラーが発生した場合もブレークします。

(1)未定義プログラム領域へのアクセス

プログラムメモリマップの未定義領域にアクセスした場合にブレークします。

(2)未定義データ領域へのアクセス

データメモリマップの未定義領域にアクセスした場合にブレークします。

(3)データROM領域への書き込み

データROM領域に書き込みが行われた場合にブレークします。

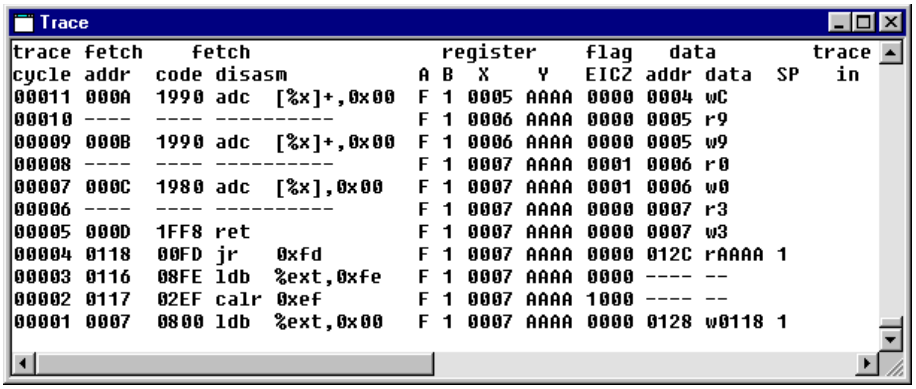
注:
- retやreti命令によるリターンアドレスのポップが16ビットアクセスの許されていない領域に対して行われると、メモリの接続されていない16ビットデータバスより不当なデータが読み出されます。ICE上ではバスがプルアップされるため0xffffが読み出され、そのアドレスへのリターンによりマップブレークが発生することがあります。
- 未定義プログラム領域へのアクセスによるブレークは実行前に発生しますが、未定義データ領域へのアクセスやデータROM領域への書き込みによるマップブレークは実行の1～2命令後に発生します。
- コマンド設定によるユーザブレークにおいても、PCブレーク、シーケンシャルブレークは実行前に発生しますが、それ以外のデータブレーク、レジスタブレーク、スタックブレークは実行の1～2命令後に発生します。

8.8.6 トレース機能

デバッガには、プログラムの実行をトレースする機能があります。

●トレースメモリとトレース情報

ICEはトレースメモリを内蔵しています。プログラムがトレースモードに従ったトレース範囲を実行すると、各サイクルごとのトレース情報がこのメモリに取り込まれます。トレースメモリは8192サイクル分の容量があり、最大4096命令(2サイクル命令のみの場合)のトレースが可能です。トレース情報がこの容量を越えると、シングルディレイトリガモード以外では古いデータから上書きしていきます。したがって、トレースメモリには常に8192サイクル以内のトレース情報が記録されています。プログラムの実行によりトレースメモリはクリアされ、新しい実行データをトレースします。



各サイクルでトレースメモリに取り込まれるトレース情報は次のとおりです。[Trace]ウィンドウの表示に対応させて示します。

trace cycle:	トレースサイクル(10進数)トレースメモリ内に最後に取り込まれた情報が00001となります。
fetch addr:	フェッチアドレス(16進数)
fetch code disasm:	フェッチコード(16進表示)と逆アセンブル内容
register:	サイクル実行後のA、B、X、Yレジスタ値(16進数)
flag:	サイクル実行後のE、I、C、Zフラグの状態(2進数)
data:	アクセスしたデータメモリアドレス(16進数)、リード/ライト(データの先頭のrまたはw)、データ(4ビットアクセスの場合1桁の16進数、16ビットアクセスの場合4桁の16進数)
SP:	スタックアクセス(SP1アクセス時が1、SP2アクセス時が2)
trace in:	TRCIN端子の入力(Lowの信号入力時にLを表示)

注: S1C63000 CPUはフェッチと実行の2段のパイプライン処理を行っています。そのため、以下の2点に注意してください。

- CPUは命令の最終実行サイクルに次の命令をフェッチします。命令の実行はフェッチの次のサイクルから行われますので、レジスタ等の表示結果は、同じ行に表示されるフェッチ命令の実行結果ではありません。
- ICEの動作タイミングの関係上、トレース開始時のフェッチサイクル、トレース終了時の実行サイクル等、境界のトレースデータが記録されない場合があります。

●トレースモード

トレース情報の採取方法の違いにより、3種類のトレースモードが設定されています。

表8.8.6.1 トレースモード設定コマンド

機 能	コマンド	メニュー
トレースモード設定	**tm**	[Trace ¦ Trace Mode Set...]

(1) ノーマルトレースモード

プログラム実行中は、すべてのバスサイクルのトレース情報をトレースメモリに取り込みます。したがって、トレースメモリはブレークが起きるまで、直前に実行されたバスサイクルまでの最新情報を常に保持しています。

(2) シングルディレイトリガトレースモード

シングルディレイトリガトレースモードの場合もプログラムの実行開始によってトレースを開始します。その後、tmコマンドで指定したアドレス(トレーストリガポイント)を実行すると、そこを基準に、やはりコマンドによる次の設定に従ってトレースを停止します。

- トレーストリガポイントを"start"に設定した場合

  トレーストリガポイントから8192サイクル分のトレース情報を採取後にトレースを停止します。この場合、トレーストリガポイントのトレース情報が、トレースメモリ内の最も古い情報となります。
  8192サイクル分のトレースの前にプログラムが停止した場合は、トレースメモリ容量内で、トレーストリガポイント以前のトレース情報も残ります。

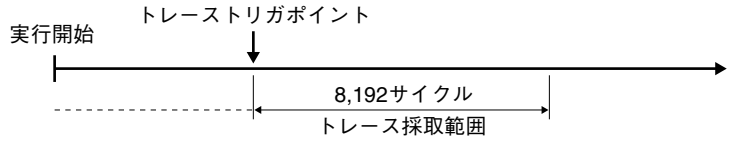


図8.8.6.1 "start"選択時のトレース範囲

- トレーストリガポイントを"middle"に設定した場合

  トレーストリガポイントから4096サイクル分のトレース情報を採取後にトレースを停止します。この場合、トレーストリガポイントの前後4096サイクルずつのトレース情報がトレースメモリに採取されます。
  4096サイクル分のトレースの前にプログラムが停止した場合は、トレースメモリ容量内で、トレーストリガポイントから4096サイクル以上前のトレース情報も残ります。

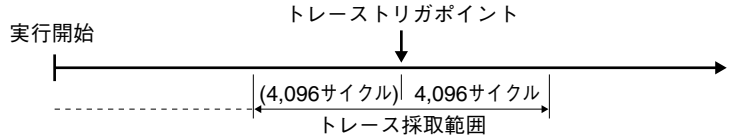


図8.8.6.2 "middle"選択時のトレース範囲

- トレーストリガポイントを"end"に設定した場合

  トレーストリガポイントのトレース情報を採取後にトレースを停止します。この場合、トレーストリガポイントのトレース情報が、トレースメモリ内の最も新しい情報となります。
  トレーストリガポイントのトレースの前にプログラムが停止した場合は、ノーマルモードと同様です。

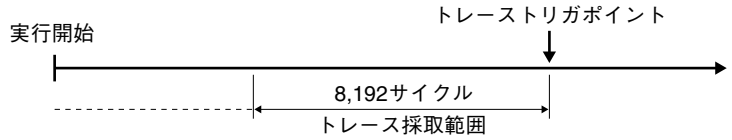


図8.8.6.3 "end"選択時のトレース範囲

シングルディレイトリガトレースの途中でプログラムを停止させた場合でも、次回の実行は新規のトレースを行います。

(3)PC範囲指定トレース

プログラムが指定したアドレスの範囲内(または範囲外)を実行したときにのみトレース情報をトレースメモリに取り込みます。最大4ヵ所のアドレス範囲と、トレースをその範囲内で行うか、または範囲外で行うかtmコマンドで指定できます。

※トレーストリガアドレス

tmコマンドではトレースモードの種類にかかわらずトレーストリガアドレスを設定します。[Source]ウィンドウを開いている場合は、設定したアドレスの先頭に"T"が表示されます。プログラムがそのアドレスを実行すると、ICEのTRGOUT端子からLowレベルのパルスが出力されます。

●トレース情報の表示と検索

採取したトレース情報はコマンドによって[Trace]ウィンドウに表示させることができます。[Trace]ウィンドウが閉じている場合は[Command]ウィンドウに表示します。[Trace]ウィンドウではスクロールによってトレースメモリの全データを見ることができます。指定したサイクルから表示させることもできます。表示内容は前記のとおりです。

表8.8.6.2 トレース情報表示コマンド

機　能	コマンド	メニュー
トレース情報表示	**td**	[View ¦ Trace]

検索条件を指定し、条件に合ったトレース情報のみを表示させることができます。
検索条件は次の3種類から選択できます。

1. プログラムの実行アドレス
2. データを読み出したアドレス
3. データを書き込んだアドレス

上記条件とアドレスを1ヵ所指定して検索を行います。条件に合ったトレース情報が見つかると、件数を[Command]ウィンドウに表示します。検索データは[Trace]ウィンドウ(閉じている場合は[Command]ウィンドウ)に表示します。

表8.8.6.3 トレース情報検索コマンド

機　能	コマンド	メニュー
トレース情報検索	**ts**	[Trace ¦ Trace Search...]

●トレース情報の保存

トレース情報を[Trace]ウィンドウに表示後(tdコマンド、もしくはtsコマンド実行後)、その中の指定サイクル範囲をファイルに保存することができます。

表8.8.6.4 トレース情報保存コマンド

機　能	コマンド	メニュー
トレース情報保存	**tf**	[Trace ¦ Trace File...]

8.8.7 フラッシュメモリの操作

インサーキットエミュレータICEはフラッシュメモリを内蔵しています。このメモリはコマンドによってICEのエミュレーションメモリおよびターゲットメモリとデータのやり取りが行えるようになっています。
フラッシュメモリはICEの電源をOFFにしてもデータが消えません。電源OFFの前にデバッグ中のプログラム/データをフラッシュメモリに書き込んでおくことにより、次回はそれを読み出してデバッグを続けることができます。また、ICEをフリーランモード(ICE単体でプログラムを実行するモード)で動作させる場合も、そのプログラムをフラッシュメモリに書き込んでおく必要があります。

注: USBインタフェース版ICEを使用する場合、フラッシュメモリ操作コマンドは使用できません。

フラッシュメモリに対しては以下の操作が行えます。

(1) フラッシュメモリの読み出し
フラッシュメモリのデータをエミュレーション/ターゲットメモリにロードします。

(2) フラッシュメモリへの書き込み
エミュレーション/ターゲットメモリのデータをフラッシュメモリにセーブします。また、パラメータファイルの内容も必要に応じて書き込みます。この書き込み以降、フラッシュメモリを読み出し/書き込み禁止に設定することもできます。

(3) フラッシュメモリの消去
フラッシュメモリの内容をすべて消去します。

表8.8.7.1 フラッシュメモリ操作コマンド

機 能	コマンド	メニュー
フラッシュメモリの読み出し	**lfl**	[File ¦ Flash Memory Operation...]
フラッシュメモリへの書き込み	**sfl**	[File ¦ Flash Memory Operation...]
フラッシュメモリの消去	**efl**	[File ¦ Flash Memory Operation...]

注: デバッガdb63起動時に指定したパラメータファイルの内容とフラッシュメモリ内のパラメータの内容が一致していないと、フラッシュメモリの書き込み(sfl)、読み出し(lfl)を行うことができません。ICEの工場出荷後やフラッシュメモリの消去後、また、異なるパラメータファイル(S1C63 Familyの他機種用)を使用する場合は、sflコマンドによってパラメータファイルの内容も他のデータと同時にフラッシュメモリに書き込んでください。

※ICEのフリーラン
ICEのフリーラン(ICE単独でプログラム実行)には、フラッシュメモリに書き込まれたデータが使用されます。フリーランを行う場合は、プログラム、データ、オプションデータをすべてフラッシュメモリに書き込んでください。
ICEはICE/RUNスイッチをRUN側にセットして電源をONするとフリーランを開始します。フリーラン中は、パラメータファイルで設定されているプログラム領域、データ領域に従ったマップブレークが有効です。マップブレーク時にはICE上のPC LEDが止まり、EMU LEDが消灯します。その他のブレーク設定はフラッシュメモリには書き込まれませんので無効です。

8.8.8 カバレッジ

ICEはカバレッジ情報(プログラムを実行したアドレスの情報)を保持しており、それを読み出して[Command]ウィンドウに表示することができます。
次のように実行したアドレス範囲を表示しますので、実行していない領域が分かります。

```
Coverage Information:
    0: 0110..0108
    1: 0200..020f
```

表8.8.8.1　カバレッジコマンド

機　能	コマンド
カバレッジ情報の表示	**cv**
カバレッジ情報のクリア	**cvc**

8.8.9 標準ペリフェラルボードのFPGAデータ書き込み

標準ペリフェラルボードは、ボード上のFPGAにデータを書き込むことによってサポートしている各機種の周辺機能が設定されます。標準ペリフェラルボードを最初に使用するとき、あるいは新機種の開発を始める前にはこのデータ書き込みが必要です。
db63は、ICEに装着した標準ペリフェラルボードのFPGAを消去したり、データを書き込む機能を持っています。
FPGAに対しては以下の操作が行えます。

(1) FPGAの消去
FPGAの内容をすべて消去します。

(2) FPGAへのデータ書き込み
指定したファイルのデータをFPGAに書き込みます。書き込みコマンドでFPGAの消去も行えます。
サポート機種のデータは"¥epson¥s1c63¥ice¥fpga"ディレクトリ(デフォルト)に"c63xxx.mot"ファイルとして用意されています。

(3) FPGAデータのコンペア
FPGAのデータと指定ファイルの内容を比較します。

(4) FPGAデータダンプ
FPGAのデータを16進ダンプ形式で表示します。

表8.8.9.1　FPGAコマンド

機　能	コマンド
FPGAの消去	**xfer/xfers**
FPGAデータ書き込み	**xfwr/xfwrs**
FPGAデータコンペア	**xfcp/xfcps**
FPGAデータダンプ	**xdp/xdps**

注:　標準ペリフェラルボードにはメインとサブ、2個のFPGAが搭載されており、FPGAコマンドはそれぞれに用意されています("s"付きがサブFPGA用コマンド)。
ただし、通常はサブFPGAにLCDのDC出力機能が登録されているため、データ書き込みの必要はありません。

8.9 コマンドリファレンス

8.9.1 コマンド一覧

表8.9.1.1にデバッグコマンドの一覧を示します。

表8.9.1.1 コマンド一覧表

分 類	コマンド		機 能	P
プログラムメモリ操作	a / as	(assemble)	インラインアセンブル	159
	pe	(program memory enter)	プログラムコード入力	161
	pf	(program memory fill)	プログラム領域のフィル	162
	pm	(program memory move)	プログラム領域のコピー	163
データメモリ操作	dd	(data memory dump)	データメモリダンプ	164
	de	(data memory enter)	データ入力	166
	df	(data memory fill)	データ領域のフィル	168
	dm	(data memory watch)	データ領域のコピー	169
	dw	(data memory fill)	監視データアドレス設定	170
オプション情報	od	(option data dump)	オプションデータダンプ	172
レジスタ操作	rd	(register display)	レジスタ表示	174
	rs	(register set)	レジスタ変更	175
プログラム実行	g	(go)	連続実行	177
	gr	(go after reset CPU)	CPUリセット&連続実行	179
	s	(step)	ステップ実行	180
	n	(next)	スキップ付きステップ実行	182
CPUリセット	rst	(reset CPU)	CPUリセット	183
ブレーク	bp	(breakpoint set)	ブレークポイントの設定	184
	bc / bpc	(breakpoint clear)	ブレークポイントの解除	186
	bd	(data break)	データブレークの設定	187
	bdc	(data break clear)	データブレークの解除	189
	br	(register break)	レジスタブレークの設定	190
	brc	(register break clear)	レジスタブレークの解除	192
	bs	(sequential break)	シーケンシャルブレークの設定	193
	bsc	(sequential break clear)	シーケンシャルブレークの解除	195
	bsp	(break stack pointer)	スタック領域指定(スタック不当アクセス検出用)	196
	bl	(breakpoint list)	全ブレーク条件の表示	198
	bac	(break all clear)	全ブレーク条件の解除	199
プログラム表示	u	(unassemble)	逆アセンブル表示	200
	sc	(source code)	ソース表示	202
	m	(mix)	ミックス表示	204
シンボル情報	sy	(symbol list)	シンボル一覧	206
ファイルロード	lf	(load file)	IEEE-695形式アブソリュートオブジェクトファイル読み込み	207
	lo	(load option)	モトローラS形式ファイル読み込み	208
フラッシュメモリ操作 *	lfl	(load from flash memory)	フラッシュメモリの読み出し	209
	sfl	(save to flash memory)	フラッシュメモリへの書き込み	211
	efl	(erase flash memory)	フラッシュメモリ消去	213
トレース	tm	(trace mode)	トレースモードの設定	214
	td	(trace data display)	トレース情報の表示	216
	ts	(trace search)	トレース情報の検索	219
	tf	(trace file)	トレース情報の保存	221
カバレッジ	cv	(coverage)	カバレッジ情報の表示	222
	cvc	(coverage clear)	カバレッジ情報のクリア	223
コマンドファイル	com	(execute command file)	コマンドファイル読み込み/実行	224
	cmw	(execute command file with wait)	実行間隔指定付きコマンドファイル読み込み/実行	225
	rec	(record commands to file)	実行コマンドの記録	226
ログ	log	(log)	ログ出力ON/OFF	227
マップ情報	ma	(map information)	マップ情報の表示	228
モード設定	md	(mode)	モード設定	229
FPGA操作	xfer/xfers	(xilinx fpga data erase)	FPGAの消去	232
	xfwr/xfwrs	(xilinx fpga data write)	FPGAデータ書き込み	233
	xfcp/xfcps	(xilinx fpga data compare)	FPGAデータコンペア	234
	xdp/xdps	(xilinx fpga data dump)	FPGAデータダンプ	235
終了	q	(quit)	デバッガ終了	236
ヘルプ	?	(help)	コマンドusageの表示	237

* USBインタフェース版ICEを使用する場合、フラッシュメモリ操作コマンド(lfl、sfl、efl)は使用できません。

8.9.2 各コマンド説明の見方

次項よりすべてのデバッグコマンドを機能別に説明します。
各コマンドの説明項目を以下に示します。

機　能

コマンドの機能が記述されています。

書　式

キーボードからのコマンド入力形式、およびパラメータの内容が記述されています。

例

コマンドの実行例が記述されています。

注

使用上の注意事項や補足説明が記述されています。

ＧＵＩ

コマンドを実行可能なメニューやボタン、パラメータを指定するダイアログボックスを示します。

注:
- 書式の記述で、<>で囲まれたパラメータはユーザによって指定される内容です。[]で囲まれたパラメータは省略可能であることを示します。
- ユーザ定義シンボルを除き、コマンドは大文字と小文字が区別されません。大文字、小文字のどちらでも、また混在した形でも入力できます。
- 直接入力モードでコマンドを実行する場合、必要なパラメータがすべて入力されないとエラーになります。

```
Error : Incorrect number of parameters
```

8.9.3 プログラムメモリ操作コマンド

a / as (assemble mnemonic)

機 能

ニーモニックを入力してプログラムメモリの内容を書き換えます。
入力したニーモニックは機械語コードに変換され、指定のアドレスに書き込まれます。

書 式

(1)>a <address> <mnemonic> [<file name>]↲　　(直接入力モード)

(2)>a [<address>]↲　　(ガイダンスモード)
Start address ? : <address>↲　　...<address>を省略した場合にのみ表示
アドレス　現在のコード　現在のニーモニック： <mnemonic>↲
.....
>

<address>: 書き込み開始アドレス　16進数、またはシンボル(IEEE-695形式のみ)
<mnemonic>: ニーモニックコード　S1C63000命令(式とシンボルも使用可能)
<file name>: オペランドに使用したシンボルが定義されているソースファイル
条件: 0≦address≦プログラムメモリ最終アドレス

例

書式1) `>a 200 "ld %a,f"↲`　　..."LD %A,0xF"をアセンブルし、0x200番地に書き込みます。

書式2) `>a↲`

```
Start address ? 200↲                          ...開始アドレスを入力
0200  1ff6  ld   %a,%f  : add %a,%b↲          ...ニーモニックを入力
Source file name (enter to ignore)   ?↲       ...無視*
0201  1fff  *nop  : ^↲                        ...前のアドレスに戻す
0200  1972  add  %a,%b  : ↲                   ...入力をスキップ
0201  1fff  *nop  : q↲                        ...コマンドを終了
>
```

* ソースファイル名は、オペランドにシンボル(ラベル)を使用した場合に入力してください。シンボルが定義されているソースファイルを指定します。

```
0200  1972  add  %a,%b  : jr LOOP↲              ...シンボルを使用
Source file name (enter to ignore)   ? main.s↲  ...ソースファイル名を入力
```

注

- aコマンドとasコマンドは同じ機能を持っています。
- 開始アドレスは、各機種のプログラムメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

```
Error : invalid value      (no such symbol / symbol type error)
```

  メモリの有効範囲を越えた場合、エラーとなります。

```
Error : Address out of range, use 0-0xXXXX
```

- S1C63000に無効なニーモニックが入力された場合はエラーとなります。

```
Error : illegal mnemonic
```

- ガイダンスモードでは、以下のキー操作も有効です。
  "q↲" ...コマンドを終了(入力を終了し、コマンド機能を実行)
  "^↲" ...直前のアドレスに戻る
  "↲" ...入力をスキップ(現在の内容を保持)

  プログラムメモリの最終アドレスに達した場合、"^↲"以外の有効な入力を行うと、コマンドは終了します。
- a(as)コマンドでプログラムメモリの内容を変更すると、[Source]ウィンドウの逆アセンブル内容は即時更新されます。
- プログラムメモリの書き換えにより、逆アセンブル表示内容は変更されますが、ソース表示内容は変更されません。

GUI

なし

pe (program memory enter)

機 能

16進データを入力してプログラムメモリの指定アドレスの内容を書き換えます。

書 式

(1)>pe <address> <code1> [<code2> [...<code8>]]↲ (直接入力モード)

(2)>pe↲ (ガイダンスモード)
Program enter address ? <address>↲ ...<address>を省略した場合にのみ表示
アドレス 現在のコード : <code>↲
.....
>

<address>: 書き込み開始アドレス 16進数、またはシンボル(IEEE-695形式のみ)
<code(1–8)>: 書き込みコード 16進数(S1C63000に有効な命令コード)
条件: 0≦address≦プログラムメモリ最終アドレス、0≦code≦0x1fff

例

書式1) >pe 200 1972↲ ...アドレス0x200のコードを0x1972で書き換え

書式2) >pe↲

```
Program enter address ? 200↲     ...開始アドレスを入力
0200 1fff   : 1972↲              ...コードを入力
0201 1fff   : ↲                  ...アドレス0x201をスキップ
0202 1fff   : q↲                 ...コマンドを終了
>
```

注

- 開始アドレスは、各機種のプログラムメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

  ```
  Error : invalid value      (no such symbol / symbol type error)
  ```

  メモリの有効範囲を越えた場合、エラーとなります。

  ```
  Error : Address out of range, use 0-0xXXXX
  ```

- コードは13ビット(0～0x1fff)の範囲内の16進数を入力してください。
  コードが16進数以外の場合、エラーとなります。

  ```
  Error : invalid value
  ```

  コードの有効範囲を越えた場合、あるいは機種が未対応のコード(パラメータファイルで指定されます)を入力するとエラーとなります。

  ```
  Error : illegal code
  ```

- ガイダンスモードでは、以下のキー操作も有効です。
  "q↲" ...コマンドを終了(入力を終了し、コマンド機能を実行)
  "^↲" ...直前のアドレスに戻る
  "↲" ...入力をスキップ(現在の内容を保持)

  プログラムメモリの最終アドレスに達した場合、"^↲"以外の有効な入力を行うと、コマンドは終了します。

- peコマンドでプログラムメモリの内容を変更すると、[Source]ウィンドウの逆アセンブル内容は即時更新されます。

- コードの書き換えにより、逆アセンブル表示内容は変更されますが、ソース表示内容は変更されません。

GUI

なし

pf (program memory fill)

機 能

指定のプログラムメモリ領域の内容すべてを指定のコードに書き換えます。

書 式

(1)>pf <address1> <address2> <code>↵ (直接入力モード)

(2)>pf↵ (ガイダンスモード)

```
Start address ? <address1>↵
End address ? <address2>↵
Fill code ? <code>↵
>
```

<address1>: 指定範囲先頭アドレス 16進数、またはシンボル(IEEE-695形式のみ)
<address2>: 指定範囲終了アドレス 16進数、またはシンボル(IEEE-695形式のみ)
<code>: 書き込みコード 16進数(S1C63000に有効な命令コード)
条件: 0≦address1≦address2≦プログラムメモリ最終アドレス、0≦code≦0x1fff

例

書式1) >pf 200 20f 1ffe↵ ...アドレス0x200～0x20fの範囲を0x1ffeのコードで書き換え

書式2) >pf↵

```
Start address ? 200↵      ...先頭アドレスを入力
End address ? 20f↵        ...終了アドレスを入力
Fill code ? 1fff↵         ...コードを入力
>
```

※[Enter]キーのみの入力でコマンドの実行を中止できます。

注

- アドレスは、各機種のプログラムメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

```
Error : invalid value      (no such symbol / symbol type error)
```

  メモリの有効範囲を越えた場合、エラーとなります。

```
Error : Address out of range, use 0-0xXXXX
```

- 先頭アドレスが終了アドレスより大きい場合、エラーとなります。

```
Error : end address < start address
```

- pfコマンドでプログラムメモリの内容を変更すると、[Source]ウィンドウの逆アセンブル内容も更新されます。
- コードの書き換えにより、逆アセンブル表示内容は変更されますが、ソース表示内容は変更されません。

GUI

なし

pm (program memory move)

機 能

指定のプログラムメモリ領域の内容を別の領域にコピーします。

書 式

(1)>pm <address1> <address2> <address3>↵　　(直接入力モード)

(2)>pm↵　　(ガイダンスモード)

```
Start address ? <address1>↵
End  address ? <address2>↵
Destination address ? <address3>↵
>
```

<address1>: コピー元の先頭アドレス　16進数、またはシンボル(IEEE-695形式のみ)
<address2>: コピー元の終了アドレス　16進数、またはシンボル(IEEE-695形式のみ)
<address3>: コピー先のアドレス　16進数、またはシンボル(IEEE-695形式のみ)
条件: 0≦address1≦address2≦プログラムメモリ最終アドレス
0≦address3≦プログラムメモリ最終アドレス

例

書式1) >pm 200 2ff 280↵ ...アドレス0x200～0x2ffの範囲をアドレス0x280から始まる領域にコピー

書式2) >pm↵

```
Start address ? 200↵          ...コピー元の先頭アドレスを入力
End address ? 2ff↵            ...コピー元の終了アドレスを入力
Destination address ? 280↵    ...コピー先のアドレスを入力
>
```

※[Enter]キーのみの入力でコマンドの実行を中止できます。

注

- アドレスは、各機種のプログラムメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。
  `Error : invalid value　　(no such symbol / symbol type error)`
  メモリの有効範囲を越えた場合、エラーとなります。
  `Error : Address out of range, use 0-0xXXXX`
- 先頭アドレスが終了アドレスより大きい場合、エラーとなります。
  `Error : end address < start address`
- pmコマンドでプログラムメモリの内容を変更すると、[Source]ウィンドウの逆アセンブル内容も更新されます。
- コードの書き換えにより、逆アセンブル表示内容は変更されますが、ソース表示内容は変更されません。

GUI

なし

8.9.4 データメモリ操作コマンド

dd (data memory dump)

機 能

データメモリの内容を16ワード/行の16進ダンプ形式で表示します。

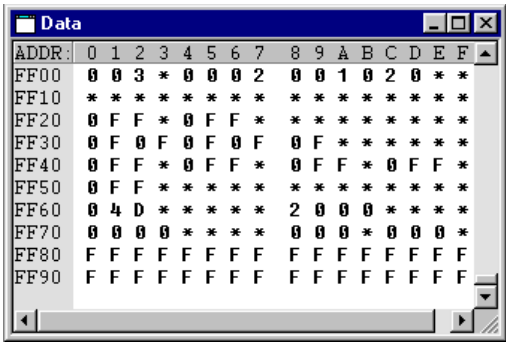
書 式

>dd [<address1> [<address2>]]↲　　（直接入力モード）

<address1>: 表示開始アドレス　16進数、またはシンボル（IEEE-695形式のみ）
<address2>: 表示終了アドレス　16進数、またはシンボル（IEEE-695形式のみ）
条件:　　0≦address1≦address2≦0xffff

表 示

(1) [Data]ウィンドウを開いている場合



<address1>と<address2>を省略した場合、[Data]ウィンドウはアドレス0x0000からデータを表示します。<address1>を指定すると、<address1>が最上部の行に表示されるように[Data]ウィンドウを再表示します。<address1>が16アドレス/行の途中を指定していても、データはその行の先頭から表示されます。たとえば、<address1>に0xff08を指定した場合でもデータはアドレス0xff00から表示されます。ただし、<address1>に0xffc0などのメモリ最上部（最終アドレスが0xffffの場合）付近のアドレスを指定した場合は、ウィンドウの最終行が0xfff0となり、指定のアドレスがウィンドウの先頭部とはなりません。

[Data]ウィンドウは、スクロールによってデータメモリ全体を表示可能なため、<address2>の指定は特に意味を持ちません。<address1>のみ指定した場合も、<address1>と<address2>の両方を指定した場合もまったく同じ表示となります。

(2) [Data]ウィンドウを閉じている場合

<address1>と<address2>を省略した場合、アドレス0x000から256ワードのデータを[Command]ウィンドウに表示します。

```
>dd↲
      0 1 2 3 4 5 6 7  8 9 A B C D E F
0000: A A A A D C 0 3  A A A A A A A A
0010: A A A A A A A A  A A A A A A A A
  :          :              :
00E0: A A A A A A A A  A A A A A A A A
00F0: A A A A A A A A  A A A A A A A A
>
```

<address1>のみを指定した場合、<address1>から256ワードのデータを表示します。

```
>dd ff00↲
FF00: 0 0 3 * 0 0 0 2  0 0 1 0 2 0 * *
FF10: * * * * * * * *  * * * * * * * *
 :          :              :
FFE0: 0 0 0 0 0 0 0 0  * * * * * * * *
FFF0: 0 0 0 0 0 0 0 0  * * * * * * * *
>
```

"*"は未使用アドレスを示します。

<address1>と<address2>の両方を指定すると、<address1>から<address2>までのデータを表示します。

```
>dd 008 017↵
      0 1 2 3 4 5 6 7  8 9 A B C D E F
0000:                  0 0 0 0 0 0 0 0
0010: 0 0 0 0 0 0 0 0
>
```

(3) ログ出力中

logコマンドの指定によってコマンド実行結果をログファイルに出力している場合は、[Data]ウィンドウが開いている場合でも [Command]ウィンドウにデータを表示し、その内容をログファイルにも出力します。

[Data]ウィンドウが閉じている場合、[Command]ウィンドウへの表示は上記(2)と同様です。

[Data]ウィンドウが開いていれば、上記(1)と同様にその再表示も行います。この場合、[Command]ウィンドウに表示される行数は[Data]ウィンドウの表示行数と同じになります。

(4) 連続表示機能

ddコマンドを一度実行すると、他のコマンドを実行するまでは[Enter]キーの入力のみでデータを連続して表示することができます。

[Enter]キーを入力すると、[Data]ウィンドウは1画面分スクロールします。

[Command]ウィンドウにデータを表示している場合は、前回表示したアドレスに続く16行分(ログ出力中は[Data]ウィンドウと同じ行数)を表示します。

```
>dd↵
      0 1 2 3 4 5 6 7  8 9 A B C D E F
0000: A A A A A A A A  A A A A A A A A
0010: A A A A A A A A  A A A A A A A A
  :          :              :
00F0: A A A A A A A A  A A A A A A A A
>↵
      0 1 2 3 4 5 6 7  8 9 A B C D E F
0100: A A A A A A A A  A A A A A A A A
0110: A A A A A A A A  A A A A A A A A
  :          :              :
01F0: A A A A A A A A  A A A A A A A A
>
```

アドレス0xfff0の行が表示されるとその時点でコマンド入力待ちとなり、そこでの[Enter]キー入力はアドレス0x0000からの表示を行います。

注

- アドレスは、各機種のデータメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

  ```
  Error : invalid value      (no such symbol / symbol type error)
  ```

  メモリの有効範囲を越えた場合、エラーとなります。

  ```
  Error : Address out of range, use 0-0xFFFF
  ```

- 先頭アドレスが終了アドレスより大きい場合、エラーとなります。

  ```
  Error : end address < start address
  ```

GUI

[View ¦ Data Dump]メニューコマンド

このメニューコマンドを選択することによって、[Data]ウィンドウがアクティブになり、現在のデータメモリの内容を表示します。

de (data memory enter)

機 能

16進データを入力してデータメモリの内容を書き換えます。指定したアドレスから連続してデータの書き込みが行えます。

書 式

(1)>de <address> <data1> [<data2> [...<data16>]]↲　　(直接入力モード)

(2)>de↲　　(ガイダンスモード)

Data enter address ? <address>↲
アドレス　現在のデータ : <data>↲
.....
>

<address>: 書き込み開始アドレス　16進数、またはシンボル(IEEE-695形式のみ)
<data(1–16)>: 書き込みデータ　16進数
条件: 0≦address≦0xffff、0≦data≦0xf

例

書式1) >de 100 0↲　　...アドレス0x100にデータ0を書き込み

書式2) >de↲
```
Data enter address ? :100↲      ...開始アドレスを入力
100   0 : a↲                    ...データを入力
101   0 : ↲                     ...入力をスキップ
102   0 : q↲                    ...コマンドを終了
>
```

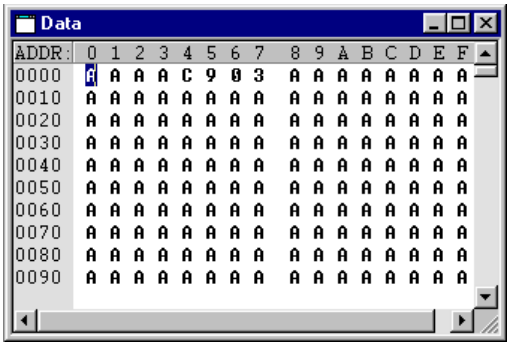
注

- 開始アドレスは、各機種のデータメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。
  ```
  Error : invalid value      (no such symbol / symbol type error)
  ```
  メモリの有効範囲を越えた場合、エラーとなります。
  ```
  Error : Address out of range, use 0-0xFFFF
  ```
- 未使用アドレスは、"*"を表示します。"*"が表示されたアドレスは、[Enter]キーでそのアドレスをスキップするか、コマンドを終了させてください。
- データは4ビット(0～0xf)の範囲内の16進数で入力してください。これを越えるとエラーとなります。
  ```
  Error : Data out of range, use 0-0xF
  ```
- deコマンドでデータメモリの内容を変更すると、[Data]ウィンドウの表示内容は自動的に更新されます。
- ガイダンスモードでは、以下のキー操作も有効です。
  "q↲"　...コマンドを終了(入力を終了し、コマンド機能を実行)
  "^↲"　...直前のアドレスに戻る
  "↲"　...入力をスキップ(現在の内容を保持)

  データメモリの最終アドレスに達した場合、"^↲"以外の有効な入力を行うと、コマンドは終了します。

GUI

[Data]ウィンドウ



[Data]ウィンドウ上で直接データを変更可能です。[Data]ウィンドウ内の変更するデータをマウスで選択し、16進数を入力してください。

df (data memory fill)

機 能

指定のデータメモリ領域の内容すべてを指定のデータに書き換えます。

書 式

(1)>df <address1> <address2> <data>↲　　(直接入力モード)

(2)>df↲　　(ガイダンスモード)

Start address ? <address1>↲
End address ? <address2>↲
Data pattern ? <data>↲
>

<address1>: 指定範囲先頭アドレス　16進数、またはシンボル(IEEE-695形式のみ)
<address2>: 指定範囲終了アドレス　16進数、またはシンボル(IEEE-695形式のみ)
<data>: 書き込みデータ　16進数
条件: 0≦address1≦address2≦0xffff、0≦data≦0xf

例

書式1) >df 200 2ff 0↲　　...アドレス0x200～0x2ffの範囲を0x0で書き換え

書式2) >df↲
```
Start address ? 200↲
End address ? 2ff↲
Data pattern ? 0↲
>
```
...先頭アドレスを入力
...終了アドレスを入力
...書き込みデータを入力

※[Enter]キーのみの入力でコマンドの実行を中止できます。

注

- アドレスは、各機種のデータメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。
  ```
  Error : invalid value      (no such symbol / symbol type error)
  ```
  メモリの有効範囲を越えた場合、エラーとなります。
  ```
  Error : Address out of range, use 0-0xFFFF
  ```
- 先頭アドレスが終了アドレスより大きい場合、エラーとなります。
  ```
  Error : end address < start address
  ```
- データは4ビット(0～0xf)の範囲内の16進数で入力してください。これを越えるとエラーとなります。
  ```
  Error : Data out of range, use 0-0xF
  ```
- I/O領域の読み出し専用アドレスに対して書き込みは行われません。
- 指定アドレス範囲に未使用領域が含まれている場合、未使用領域以外への書き込みは行われ、エラーとはなりません。
- dfコマンドでデータメモリの内容を変更すると、[Data]ウィンドウの表示内容は自動的に更新されます。

GUI

なし

dm (data memory move)

機 能

指定のデータメモリ領域の内容を別の領域にコピーします。

書 式

(1)>dm <address1> <address2> <address3>↵　　(直接入力モード)

(2)>dm↵　　(ガイダンスモード)

```
Start address ? <address1>↵
End  address ? <address2>↵
Destination address ? <address3>↵
>
```

<address1>: コピー元の先頭アドレス　16進数、またはシンボル(IEEE-695形式のみ)
<address2>: コピー元の終了アドレス　16進数、またはシンボル(IEEE-695形式のみ)
<address3>: コピー先のアドレス　16進数、またはシンボル(IEEE-695形式のみ)
条件: 0≦address1≦address2≦0xffff
0≦address3≦0xffff

例

書式1) >dm 200 2ff 280↵ ...アドレス0x200～0x2ffの範囲をアドレス0x280から始まる領域にコピー

書式2) >dm↵

```
Start address ? 200↵        ...コピー元の先頭アドレスを入力
End address ? 2ff↵           ...コピー元の終了アドレスを入力
Destination address ? 280↵ ...コピー先のアドレスを入力
>
```

※[Enter]キーのみの入力でコマンドの実行を中止できます。

注

- アドレスは、各機種のデータメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

```
Error : invalid value      (no such symbol / symbol type error)
```

  メモリの有効範囲を越えた場合、エラーとなります。

```
Error : Address out of range, use 0-0xFFFF
```

- I/O領域の読み出し専用アドレスに対して書き込みは行われません。
- 書き込み専用アドレスのデータは読み出せません。コピー元の領域が書き込み専用アドレスを含んでいる場合、対応するコピー先アドレスには0が書き込まれます。コピー先の領域が読み出し専用アドレスを含んでいる場合、そのアドレスに対して書き込みは行われません。コピー元またはコピー先に書き込み専用ビットまたは読み出し専用ビットを含むアドレスがある場合、それらのビットに対応した書き込み動作を行います。
- dmコマンドでデータメモリの内容を変更すると、[Data]ウィンドウの表示内容は自動的に更新されます。

GUI

なし

dw (data memory watch)

機 能

データメモリ領域の4ヵ所を監視データアドレスとして登録します。それぞれのアドレスから4ワード分の内容が[Register]ウィンドウに表示されます。

書 式

(1)>dw <address1> [... <address4>]↲　　(直接入力モード)

(2)>dw↲　　(ガイダンスモード)

Address 1 = *現在の値* :<address1>↲
Address 2 = *現在の値* :<address2>↲
Address 3 = *現在の値* :<address3>↲
Address 4 = *現在の値* :<address4>↲
>

<address1–4>: 監視アドレス　16進数、またはシンボル(IEEE-695形式のみ)
条件: 0≦address1≦address2≦0xffff

例

書式1) >dw 10 14 18 1C↲　　...監視アドレスを0x10、0x14、0x18、0x1cに設定

書式2) >dw↲

```
Address1 = 0010 :0↲
Address2 = 0014 :4↲
Address3 = 0018 :8↲
Address4 = 001c :c↲
>
```

注

- デバッガ起動時は0、4、8、0xc番地の4ヵ所が初期設定されます。
- アドレスは、各機種のデータメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

```
Error : invalid value      (no such symbol / symbol type error)
```

  メモリの有効範囲を越えた場合、エラーとなります。

```
Error : Address out of range, use 0-0xFFFF
```

- 監視データアドレスは4ワード単位で設定されます。このワード境界以外のアドレスを指定するとワーニングとなり、4の倍数に切り捨てられます。

例: >dw↲

```
Address1 = 0000 :0↲
Address2 = 0004 :10↲
Address3 = 0008 :15↲            ...不正アドレス
Address4 = 000C :19↲            ...不正アドレス
Warning : round down to multiple of 4
Address1 =    0
Address2 =   10
Address3 =   14
>
```

- ddコマンドでは"*"で表示される部分も値が表示されますので注意してください。その内容は不定です。
- 値の表示は、左端が開始アドレスの内容で、右端が開始アドレス+3のアドレスの内容です。

GUI

なし

8.9.5 オプション情報表示コマンド

od (option data dump)

機 能

ICEからオプションデータを読み出して、[Command]ウィンドウに16進ダンプ形式で表示します。

オプションデータ	ターゲットメモリアドレス範囲
ファンクションオプションデータ(fog)	0～0xef
セグメントオプションデータ(sog)	0～0x1fff
メロディーデータ(mla)	0～0xfff

書 式

(1)>od <type> [<address1> [<address2>]]↲ (直接入力モード)

(2)>od↲ (ガイダンスモード)

1. fog 2. sog 3. mla . . . ? <type>↲
Start address ? <address1>↲
End address ? <address2>↲
オプションデータ表示 . . . .
>

<type>: オプション形式 fog、sog、mla
<address1>: 指定範囲開始アドレス 16進数
<address2>: 指定範囲終了アドレス 16進数
条件: 0≦address1≦address2≦0xef(fog)、0x1fff(sog)、0xfff(mla)

例

書式1) >od fog 0 f↲ ...0～0xfの範囲にファンクションオプションデータを表示

```
      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
0000: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

書式2) >od↲

```
1.fog 2.sog 3.mla ...? 1↲  ...ファンクションオプションを選択
Start address ? 10↲        ...開始アドレスを入力
End address ? 1f↲          ...終了アドレスを入力
      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
0010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
>
```

注

- 開始アドレス、終了アドレスは[Enter]キーの入力のみで省略可能です。
  開始アドレスを省略すると、アドレス0から表示されます。
  終了アドレスを省略すると、オプション領域の範囲内で最大16行のデータを表示します。
- 未使用領域のデータは"*"として表示されます。
- 一度に表示可能な行数は16行(fogデータは15行で終了)です。終了アドレスを16行以上表示するように指定しても16行分の表示後、コマンド入力待ちとなります。なお、ddコマンド等と同様に[Enter]キーの入力で続くアドレスのデータ(最大16行)が表示されます。

- 開始、終了アドレスともに、各オプションの設定範囲内で指定してください。これを越えた場合、エラーとなります。

```
Error : FO address out of range, use 0-0xEF
                                    ...ファンクションオプションの指定アドレスが範囲外
Error : SO address out of range, use 0-0x1FFF
                                    ...セグメントオプションの指定アドレスが範囲外
Error : MLA address out of range, use 0-0xFFF
                                    ...メロディーデータの指定アドレスが範囲外
```

- 開始アドレスが終了アドレスより大きい場合、エラーとなります。

```
Error : end address < start address
```

- オプションデータのデフォルト値は0です。

GUI

なし

8.9.6 レジスタ操作コマンド

rd (register display)

機 能

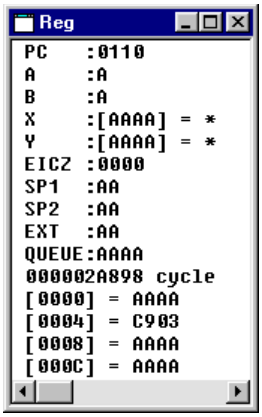
レジスタ、実行サイクルカウンタ、監視データの内容を表示します。

書 式

>rd↵ （直接入力モード）

表 示

(1) 表示内容



表示する内容は以下のとおりです。

PC:	プログラムカウンタ
A:	Aレジスタ
B:	Bレジスタ
X:	Xレジスタおよび間接アドレッシングされるデータメモリの内容
Y:	Yレジスタおよび間接アドレッシングされるデータメモリの内容
EICZ:	フラグ
SP1:	スタックポインタSP1
SP2:	スタックポインタSP2
EXT:	EXTレジスタ
QUEUE:	QUEUEレジスタ
bus cycle:	実行サイクルカウンタ
[xxxx]:	4ヵ所の監視データ

※ X、Yレジスタが示すメモリが未使用領域の場合、そのデータは"*"で表示されます。
監視データについては、未使用領域(不定)であっても値が表示されますので注意してください。

(2) [Register]ウィンドウを開いている場合

[Register]ウィンドウを開いている場合は、プログラムの実行終了後に上記の内容がすべて[Register]ウィンドウに表示されます。rdコマンドは[Register]ウィンドウの表示を更新します。

(3) [Register]ウィンドウが閉じている場合

[Command]ウィンドウに次のようにデータを表示します。

```
>rd↵
PC:0110 A:A B:A X:[AAAA] = * Y:[AAAA] = * EICZ:0000 SP1:AA SP2:AA EXT:AA
QUEUE:AAAA  bus cycle:000002AB3D cycle
[0000] = 0000 [0010] = AAAA [0014] = AAAA [0018] = AAAA
>
```

(4) ログ出力中

logコマンドの指定によってコマンド実行結果をログファイルに出力している場合は、[Register]ウィンドウが開いている場合でも [Command]ウィンドウにデータを表示し、その内容をログファイルにも出力します。

GUI

[View ¦ Register]メニューコマンド

このメニューコマンドを選択することによって、[Register]ウィンドウがアクティブになり、現在の各レジスタの内容を表示します。

rs (register set)

機 能

レジスタの値を変更します。

書 式

(1)>rs <register> <value> [<register> <value> [...<register> <value>]]↲　　(直接入力モード)

(2)>rs↲　　(ガイダンスモード)

```
    PC = 現在の値 : <value>↲
     A = 現在の値 : <value>↲
     B = 現在の値 : <value>↲
     X = 現在の値 : <value>↲
     Y = 現在の値 : <value>↲
    FE = 現在の値 : <value>↲
    FI = 現在の値 : <value>↲
    FC = 現在の値 : <value>↲
    FZ = 現在の値 : <value>↲
   SP1 = 現在の値 : <value>↲
   SP2 = 現在の値 : <value>↲
   EXT = 現在の値 : <value>↲
     Q = 現在の値 : <value>↲
  >
```

<register>: レジスタ名(PC、A、B、X、Y、F、SP1、SP2、EXT、Q)
<value>: レジスタの設定値　16進数

例

書式1) >rs PC 110 F 0000↲　　...PCを0x0110に設定し、全フラグをクリア

書式2) >rs↲

```
    PC= 116: 110↲
     A=   0: f↲
     B=   0: ↲
     X=   0: 100↲
     Y=   0: 100↲
    FE=   0: ↲
    FI=   0: ↲
    FC=   1: 0↲
    FZ=   1: 0↲
   SP1=  aa: ff↲
   SP2=  aa: ff↲
   EXT=   0: ↲
     Q=   0: ↲
```

レジスタを変更する場合、[Register]ウィンドウは入力した内容に更新されます。
"q↲"によって途中で入力を中止した場合、それまでに入力した内容は変更されます。

注

- レジスタのビット数を越える値を入力するとエラーとなります。
  ```
  Error : invalid value
  ```
- 直接入力モードで、PC、A、B、X、Y、F、SP1、SP2、EXT、Q以外のレジスタ名を入力するとエラーとなります。
  ```
  Error : Incorrect register name, use PC/A/B/X/Y/F/SP1/SP2/EXT/Q
  ```
- ガイダンスモードでは、以下のキー操作も有効です。
  "q↲" ...コマンドを終了(入力を終了し、コマンド機能を実行)
  "^↲" ...直前のアドレスに戻る
  "↲" ...入力をスキップ(現在の内容を保持)

GUI

[Register]ウィンドウ

[Register]ウィンドウ上で直接データを変更可能です。[Register]ウィンドウ内の変更するデータをダブルクリックして選択し、設定値を入力後に[Enter]キーを押してください。

8.9.7 プログラム実行コマンド

g (go)

機 能

現在のPCからターゲットプログラムを実行します。

書 式

>g [<address1> [<address2>]]↲ （直接入力モード）

<address1–2>: テンポラリブレークアドレス 16進数、またはシンボル（IEEE-695形式のみ）

条件: 0≦address1（2）≦プログラムメモリ最終アドレス

動 作

（1）プログラムの実行

PCが示すアドレスからターゲットプログラムを実行します。プログラムの実行は次のいずれかの要因によってブレークするまで継続します。

- すでに設定してあるブレーク条件が成立
- [Key Break]ボタンのクリックまたは[Esc]キーの入力
- マップブレーク等の発生

テンポラリブレークを指定すると、プログラムは指定アドレスの命令実行直前にブレークします。テンポラリブレークアドレスは、2つまで指定することができます。

```
>g 1a0↲     ...プログラムは現在のPCアドレスから実行を開始し、アドレス0x1a0の命令を実行後に停止します。
```

プログラムの実行がブレークすると、ブレークステータスメッセージを表示後コマンド入力待ちとなります。ここで[Enter]キーを入力すると、ブレーク後のPCアドレスからプログラムの実行を再開します。テンポラリブレークアドレスの設定も有効です。

（2）プログラム実行によるウィンドウの表示

デバッガの初期設定で、オンザフライ機能が働くようになっています。

プログラム実行中はオンザフライ機能により、[Register]ウィンドウ内のPC、フラグ、監視データの内容が0.5秒（デフォルト）ごとにリアルタイムで更新されます。[Register]ウィンドウを閉じている場合は、上記の内容が[Command]ウィンドウに表示されます。オンザフライ機能はotfコマンドによって無効にすることもできます。その場合、[Register]ウィンドウはプログラムのブレーク後に更新されます。

[Source]ウィンドウはブレーク後に、ブレークしたアドレスがウィンドウ内に表示されるように更新されます。

[Trace]ウィンドウが開いている場合は、プログラムの実行により表示内容がクリアされます。ブレーク後に、新しいトレース情報が表示されます。

[Data]ウィンドウが開いている場合、表示内容はブレーク後に更新されます。

（3）ログモード時の表示

ログモードをONにしてプログラムを実行した場合、[Command]ウィンドウにもオンザフライ表示が行われます。

例: >g

```
PC:0007  EICZ:0001 [0000] = AAAA [0004] = 3D30 [0008] = AAAA [000C] = AAAA
PC:000C  EICZ:0000 [0000] = AAAA [0004] = 5250 [0008] = AAAA [000C] = AAAA
PC:0117  EICZ:1001 [0000] = AAAA [0004] = 6760 [0008] = AAAA [000C] = AAAA
PC:000B  EICZ:0000 [0000] = AAAA [0004] = 8C70 [0008] = AAAA [000C] = AAAA
Key Break
PC:0008 A:F B:1 X:[0007] = 0 Y:[AAAA] = * EICZ:1001 SP1:4A(128) SP2:1F EXT:00
QUEUE:0118  bus cycle:0000029332 cycle [0000] = AAAA [0004] = E280 [0008] = AAAA
[000C] = AAAA
>
```

ブレーク時はrdコマンドによる場合と同じ表示を行います。

(4) 実行サイクルカウンタ

[Register]ウィンドウに表示される実行サイクルカウンタは、ターゲットプログラムの実行サイクル数または実行時間を表示します。(詳細は8.8.4項参照)
デバッガの初期設定では、実行サイクルカウンタはCPUがリセットされるまで積算されるようになっています。
mdコマンドでこれをリセットモードにすると、gコマンドを実行するたびに実行サイクルカウンタが0にリセットされるようになります。[Enter]キーによって実行を再開させた場合も、その時点でリセットされます。

注

- ブレーク条件が成立すると、プログラムは実行を中断します。このときのPCアドレスはブレークポイントの次のアドレスとなります。
- アドレスは、各機種のプログラムメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

```
Error : invalid value      (no such symbol / symbol type error)
```

  メモリの有効範囲を越えた場合、エラーとなります。

```
Error : Address out of range, use 0-0xXXXX
```

GUI

[Run ¦ Go]メニューコマンド、[Go]ボタン
このメニューコマンドまたはボタンを選択することによって、テンポラリブレーク指定のないgコマンドが実行されます。

 [Go]ボタン

[Run ¦ Go to Cursor]メニューコマンド、[Go to Cursor]ボタン
[Source]ウィンドウ内のブレークさせるアドレスの行にカーソルを置いてこのメニューコマンドまたはボタンを選択することによって、テンポラリブレーク指定付きのgコマンドが実行されます。プログラムは、カーソル位置の命令を実行後に中断します。

 [Go to Cursor]ボタン

gr (go after reset CPU)

機 能

CPUをリセットし、ブートアドレスからターゲットプログラムを実行します。

書 式

>gr [<address1> [<address2>]]↵ （直接入力モード）

<address1–2>: テンポラリブレークアドレス 16進数、またはシンボル（IEEE-695形式のみ）

条件: 0≦address1（2）≦プログラムメモリ最終アドレス

動 作

プログラムを実行する前にCPUをリセットします。これによりPCが0x100番地に設定され、そこからプログラムの実行を開始します。

プログラム実行開始後の動作はgコマンドと同様です。ただし、[Enter]キーの入力による実行の再開は行えません。詳細については、gコマンドの説明を参照してください。

注

- ブレーク条件が成立すると、プログラムは実行を中断します。このときのPCアドレスはブレークポイントの次のアドレスとなります。
- テンポラリブレークアドレスは、各機種のプログラムメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

```
Error : invalid value      (no such symbol / symbol type error)
```

  メモリの有効範囲を越えた場合、エラーとなります。

```
Error : Address out of range, use 0-0xXXXX
```

GUI

[Run ¦ Go from Reset]メニューコマンド、[Go from Reset]ボタン

このメニューコマンドまたはボタンを選択することによって、grコマンドが実行されます。

 [Go from Reset]ボタン

s (step)

機 能

現在のPCからターゲットプログラムをステップ実行します。

書 式

>s [<step>]↲ （直接入力モード）

<step>: 実行ステップ数 10進数(デフォルト=1)

条件: 0≦step≦65,535

動 作

(1) ステップ実行

<step>の指定を省略すると、PCが示すアドレスのプログラムを1ステップ実行します。<step>を指定した場合は、PCが示すアドレスから指定したステップ分のプログラムをステップ実行します。

```
>s↲          ...PCアドレスの1ステップを実行
>s 20↲       ...PCアドレスから20ステップを実行
```

プログラムの実行は、次の要因によって指定ステップ数の実行途中でも終了します。

- [Key Break]ボタンのクリックまたは[Esc]キーの入力
- マップブレーク等の発生

プログラム実行中は各ステップごとに[Register]ウィンドウ内の表示が更新されます。[Register]ウィンドウが閉じている場合は、各ステップごとにrdコマンドの実行時と同じ内容を[Command]ウィンドウに表示します。

プログラムのステップ実行が終了すると、コマンド入力待ちとなります。

ここで[Enter]キーを入力すると、再度同じステップ実行を行います。

(2) HALT、SLEEP状態と割り込み

halt命令またはslp命令を実行するとCPUはスタンバイモードとなり、その解除には割り込みが必要です。

デバッガでは、シングルステップ動作用に外部割り込みの許可/禁止モードが設定されています。

	許可モード	禁止モード
外部割り込み	割り込みを処理する	割り込みを処理しない
halt、slp命令	halt命令として実行 外部割り込み、または [Key Break]ボタンで処理を継続	halt、slp命令をnop命令に置き換えて実行

デバッガの初期設定で、割り込み禁止モードに設定されます。

mdコマンドによって割り込み許可モードに設定することもできます。

(3) 実行サイクルカウンタ

[Register]ウィンドウに表示される実行サイクルカウンタは、ターゲットプログラムの実行サイクル数または実行時間を表示します。

デバッガの初期設定では、実行サイクルカウンタはCPUがリセットされるまで積算されるようになっています。

mdコマンドでこれをリセットモードにすると、sコマンドを実行するたびに実行サイクルカウンタが0にリセットされるようになります。[Enter]キーによって再実行させた場合も、その時点でリセットされます。

(4) ログモード時

ログモードをONにしてステップ実行した場合、各ステップ終了時にrdコマンドの実行時と同じ内容を[Command]ウィンドウに表示します。

注

- ステップ数は、0～65,535の範囲内で指定してください。これを越えた場合、エラーとなります。

```
Error : Number of steps out of range, use 0-65535
```

- [Data]ウィンドウが開いている場合、その表示内容はステップ実行終了後に更新されます。
- シングルステップ動作中は、コマンドで設定したブレーク条件が成立してもブレークしません。
- 連続実行(g、grコマンド)とは異なり、[Register]ウィンドウはステップごとに更新されます。

GUI

[Run ¦ Step]メニューコマンド、[Step]ボタン
このメニューコマンドまたはボタンを選択することによって、<step>指定なしのsコマンドが実行されます。

 [Step]ボタン

n (next)

機 能

現在のPCからターゲットプログラムをステップ実行します。

書 式

>n [<step>]↲ （直接入力モード）

<step>: 実行ステップ数 10進数（デフォルト=1）

条件: 0≦step≦65,535

動 作

基本的な動作はsコマンドと同様です。

ただし、calr、calz、int命令は、次のアドレスに戻るまでのサブルーチンをすべて含めて1ステップとして実行します。

注

- ステップ数は、0～65,535の範囲内で指定してください。これを越えた場合、エラーとなります。

```
Error : Number of steps out of range, use 0-65535
```

- [Data]ウィンドウが開いている場合、その表示内容はステップ実行終了後に更新されます。
- シングルステップ動作中は、コマンドで設定したブレーク条件が成立してもブレークしません。
- 連続実行（g、grコマンド）とは異なり、[Register]ウィンドウはステップごとに更新されます。

GUI

[Run ¦ Next]メニューコマンド、[Next]ボタン

このメニューコマンドまたはボタンを選択することによって、<step>指定なしのnコマンドが実行されます。

 [Next]ボタン

8.9.8 CPUリセットコマンド

rst (reset CPU)

機 能

CPUをリセットします。

書 式

>rst↲ （直接入力モード）

注

- 各レジスタとフラグは以下のように設定されます。

PC:	0110
A:	A
B:	A
X:	AAAA
Y:	AAAA
EICZ:	0000
SP1:	AA
SP2:	AA
EXT:	AA
QUEUE:	AAAA

- 実行サイクルカウンタは0になります。
- [Source]ウィンドウが開いている場合は0x0110番地から再表示されます。[Register]ウィンドウが開いている場合は上記の内容で再表示されます。
- メモリ内容、ブレークやトレースなどのデバッグステータスはリセットされません。

GUI

[Run ¦ Reset CPU]メニューコマンド、[Reset]ボタン
このメニューコマンドまたはボタンを選択することによって、rstコマンドが実行されます。

 [Reset]ボタン

8.9.9 ブレーク設定コマンド

bp (break point set)

機 能

プログラムの実行アドレスによるブレークの設定と解除、表示を行います。

書 式

(1)>bp <break1> [<break2> [... <break16>]]↲　　(直接入力モード)

(2)>bp↲　　(ガイダンスモード)

現在のPCブレーク設定状態
1. set　2. clear　3. clear all　... ? <1 | 2 | 3>↲
..... (上記の選択に従ってガイダンスを表示)
>

<break1–16>: ブレークアドレス　16進数、またはシンボル(IEEE-695形式のみ)
条件:　　0≦address≦プログラムメモリ最終アドレス

例

書式1) >bp 116 200↲　　...ブレークポイントをアドレス0x0116と0x0200に設定
＊直接入力モードではブレークポイントの解除は行えません。

書式2) >bp↲ (Set)

```
No PC break is set.
1. set   2. clear   3. clear all ...? 1↲    ..."1. set"を選択
Set break address ? : 116↲                  ...アドレス0x0116をブレークポイントに設定
Set break address ? : 200↲                  ...アドレス0x0200をブレークポイントに設定
Set break address ? : ↲                     ...[Enter]で終了
>bp↲ (Clear)
    1: 0116
    2: 0200
1. set   2. clear   3. clear all ...? 2↲    ..."2. clear"を選択
Clear break address ? : 200↲                ...0x0200のブレークアドレスを解除
Clear break address ? : ↲                   ...[Enter]で終了
>bp↲ (Clear all)
    1: 0116
1. set   2. clear   3. clear all ...? 3↲    ..."3. clear all"を選択
>bp↲
No PC break is set.
1. set   2. clear   3. clear all ...? ↲     ...[Enter]で終了
>
```

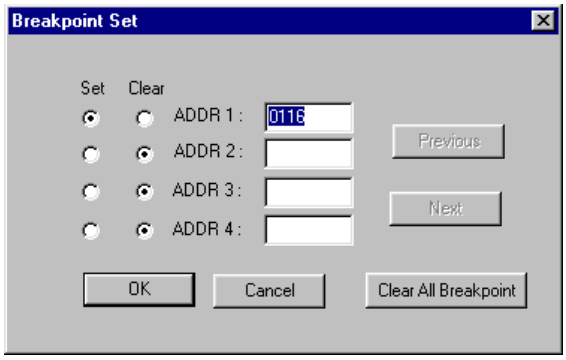
注

- アドレスは、各機種のプログラムメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。
  `Error : invalid value　　(no such symbol / symbol type error)`
  メモリの有効範囲を越えた場合、エラーとなります。
  `Error : Address out of range, use 0-0xXXXX`
- 設定されていないアドレスを解除しようとするとエラーになります。
  `Error : Input address does not exist`
- 直接入力モードでは一度に16ヵ所までブレークアドレスが指定可能で、それを越えた場合はエラーとなります。ガイダンスモードでは、この制限はありません。[Enter]キーでコマンドを終了するまで、17ヵ所以上のブレークアドレスが指定できます。
- 本コマンドを複数回実行して、新たなブレークポイントを追加することができます。

GUI

[Break ¦ Breakpoint Set...]メニューコマンド

このメニューコマンドを選択すると、ブレークポイントを設定/解除するためのダイアログボックスが表示されます。



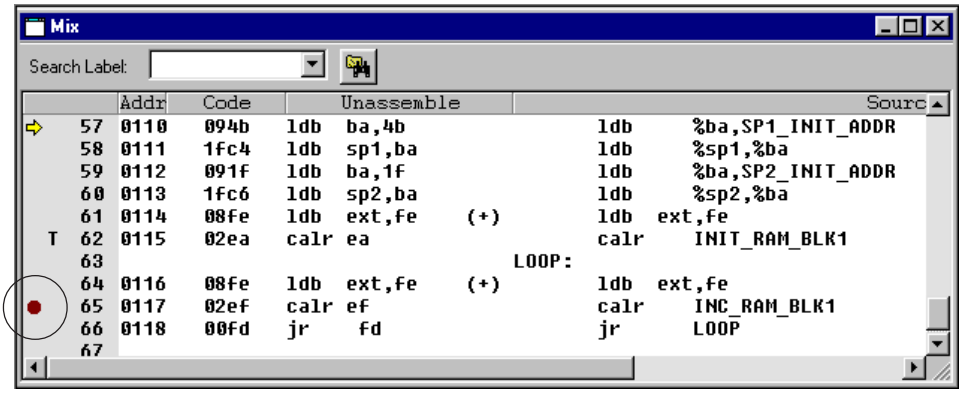
ブレークポイントを設定するには、[Set]ボタンを選択し、対応するテキストボックスにアドレスを入力します。
4ヵ所まで設定し、さらにそれ以上のブレークポイントを設定するには、[Next]ボタンをクリックして設定を続けます。
[Previous]および[Next]ボタンは、現在表示されている設定の前後の4ヵ所の設定を表示させるのに使用します。
ブレークポイントを解除するには、解除するアドレスの[Clear]ボタンを選択します。
[Clear All Breakpoint]ボタンは、設定されているすべてのブレークポイントを解除します。

[Break]ボタン

[Source]ウィンドウ内で、ブレークポイントを設定するアドレスの行をクリック(カーソルを移動)して[Break]ボタンをクリックしてください。その行がブレークポイントに設定されます。逆に、ブレークポイントに設定されている行をクリックして[Break]ボタンをクリックすると、そのブレークアドレスが解除されます。

 [Break]ボタン

[Source]ウィンドウは、ブレークポイントに設定されたアドレスを行の先頭の●マークで示します。



bc / bpc (break point clear)

機 能

設定されているブレークポイントを解除します。

書 式

>bc <break1> [... <break16>]]↲　　（直接入力モード）

<break1–16>: ブレークアドレス　16進数、またはシンボル（IEEE-695形式のみ）

例

```
>bp↲
    1: 0116                                  ...設定されているブレークポイント
    2: 0200
    3: 0260
1. set   2. clear   3. clear all ...? ↲
>bc 200↲                                     ...アドレス0x0200のブレークポイントを解除
>bp↲
    1: 0116
    2: 0260
1. set   2. clear   3. clear all ...? ↲
>bc↲                                         ...すべてのブレークポイントを解除
>bp↲
No PC break is set.
1. set   2. clear   3. clear all ...? ↲
>
```

注

- bcコマンドとbpcコマンドは同じ機能を持っています。
- アドレスの指定を省略すると、設定されているすべてのブレークポイントを解除します。
- パラメータの指定方法はbpコマンドと同じです。bpコマンドのガイダンスモードでもPCブレークポイントの解除が行えます。
- 本コマンドを複数回実行して、ブレークポイントを解除することもできます。
- ブレークポイントに設定されていないアドレスを指定するとエラーになります。

```
Error : Input address does not exist
```

GUI

[Break ¦ Breakpoint Set...]メニューコマンド

このメニューコマンドを選択すると、ブレークポイントを設定/解除するためのダイアログボックスが表示されます。（bpコマンドの説明を参照してください。）

[Break]ボタン

[Source]ウィンドウ内で、ブレークポイントに設定されているアドレスの行をクリック（カーソルを移動）して[Break]ボタンをクリックすると、そのブレークアドレスが解除されます。逆に、ブレークポイント以外のアドレスの行をクリックして[Break]ボタンをクリックすると、その行がブレークポイントに設定されます。

 [Break]ボタン

bd (data break)

機 能

データブレークの設定と解除を行います。本コマンドでは次のブレーク条件が指定できます。
1. リード/ライトするメモリアドレス範囲(1領域)
2. リード/ライトするデータパターン(ビットのマスクが可能)
3. メモリのリード/ライト(リード、ライト、リードまたはライトの3条件)

プログラムは、上記条件を満たすメモリアクセスの終了後にブレークします。

書 式

(1)>bd <data> <option> <address1> <address2>↲　　(直接入力モード)

(2)>bd↲　　(ガイダンスモード)

現在のデータブレーク設定状態
1. set　2. clear　　…? <1 | 2>↲　　("2"を選択すると以下のガイダンスなしで実行)
data　　*現在の設定データ*　　: <data>↲
R/W (R, W, ∗)　*現在の設定オプション*: <option>↲
Start address　*現在の設定アドレス*　: <address1>↲
End address　*現在の設定アドレス*　: <address2>↲
>

<data>:　　データパターン　2進数("∗"によりビットマスクが可能)
<option>:　　リード/ライトオプション　r、w、または∗
<address1–2>:　アドレス指定　16進数、またはシンボル(IEEE-695形式のみ)
条件:　　0≦address1≦address2≦0xffff、0≦data≦0b1111

例

書式1) >bd 1000 w 0 f↲　　...アドレス範囲0x0～0xfに0x8を書き込んだ場合にブレークするようにデータブレーク条件を設定
∗直接入力モードでは、条件を解除することはできません。

書式2) >bd↲

```
data: -    R/W: - area:       -
1. set   2. clear ...? 1↲                  ..."1. set"を選択
data          ---- : 1***↲                 ...データパターンを0b1***に設定
R/W (R,W,*)   -    : W↲                    ...R/W条件をライトに設定
Start address ---- : 0↲                    ...ブレークアドレスを0x0～0xfに設定
End   address ---- : f
>bd↲
data: 1***  R/W: W  area: 0000 - 000F       ...現在設定されている条件
1. set   2. clear ...? 2↲                  ..."2. clear"を選択
>bd↲
data: -    R/W: - area:       -
1. set   2. clear ...? ↲                   ...[Enter]で終了
>
```

2進データパターン内の"∗"は、そのビットを実際にアクセスしたデータと比較しないことを指定します。

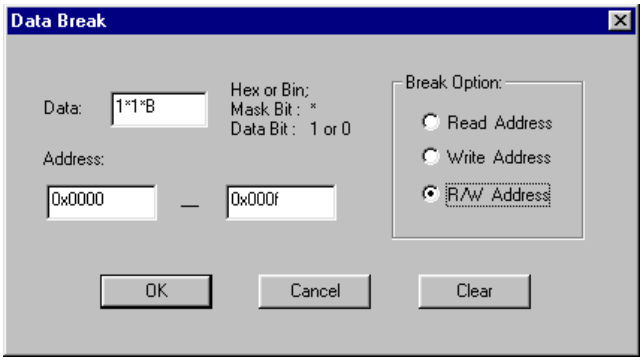
注

- デバッガを起動後はデータブレーク条件がすべて解除された状態になりますので、最初にbdコマンドを実行する際にはすべてのパラメータを指定する必要があります。
- ガイダンスモードでは、以下のキー操作も有効です。
  "q↵" ...コマンドを終了(入力を終了し、コマンド機能を実行)
  "^↵" ...直前のアドレスに戻る
  "↵" ...入力をスキップ(現在の内容を保持)

  ガイダンスの途中で"q↵"によりコマンドを終了させた場合、そこまでに指定した条件は変更されます。ただし、解除されている条件が残っている場合は変更されません。
- アドレスは、各機種のデータメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。
  ```
  Error : invalid value      (no such symbol / symbol type error)
  ```
  メモリの有効範囲を越えた場合、エラーとなります。
  ```
  Error : Address out of range, use 0-0xFFFF
  ```
- アドレス範囲内で、先頭アドレスが終了アドレスより大きい場合、エラーとなります。
  ```
  Error : end address < start address
  ```
- 16ビットアクセス(SP1スタックに対するpush/pop)に対してもアドレス、R/Wの指定は有効です。ただし、データは4ビットバスと比較しているため、意味を持ちません。この場合は、データを"****"で指定してください。また、4ビットアクセスに対するブレークを設定する場合、誤動作のおそれがありますので、16ビットアクセス領域にかかるようなアドレス指定はしないでください。
- データパターンは、4ビットの有効範囲(0～0xf)内で、マスクなし/マスク付きの2進数、あるいは16進数で指定します。この制限を越えた場合はエラーとなります。
  ```
  Error : invalid data pattern
  ```
- リード/ライトオプションに"r"、"w"、または"*"以外を指定するとエラーとなります。
  ```
  Error : Incorrect r/w option, use r/w/*
  ```
- プログラムはブレーク成立から1～2命令後に停止します。

GUI

[Break ¦ Data Break...]メニューコマンド
このメニューコマンドを選択すると、データブレーク条件を設定/解除するためのダイアログボックスが表示されます。



データブレーク条件を設定するには、アドレス、データパターンをテキストボックスに入力し、リード/ライト条件をラジオボタンで選択します。[OK]をクリックすると設定されます。
設定されているデータブレーク条件を解除するには、[Clear]をクリックしてください。

bdc (data break clear)

機 能

設定されているデータブレーク条件を解除します。

書 式

>bdc↵　　（直接入力モード）

GUI

[Break ¦ Data Break...]メニューコマンド
このメニューコマンドを選択すると、データブレーク条件を設定/解除するためのダイアログボックスが表示されます。（bdコマンドの説明を参照してください。）

br (register break)

機 能

レジスタブレークの設定と解除を行います。
本コマンドでは、A、B、F、X、Yの各レジスタ個々にブレーク条件となるデータまたはマスクを指定できます。プログラムは、設定条件がすべて満たされるとブレークします。

書 式

(1)>br <register> <value> [<register> <value> [...<register> <value>]]↲　　(直接入力モード)

(2)>br↲　　(ガイダンスモード)

現在のレジスタブレーク設定状態
1. set　2. clear　　...? <1 | 2>↲　　("2"を選択すると以下のガイダンスなしで実行)
A *現在の設定値*: <value>↲
B *現在の設定値*: <value>↲
FE *現在の設定値*: <value>↲
FI *現在の設定値*: <value>↲
FC *現在の設定値*: <value>↲
FZ *現在の設定値*: <value>↲
X *現在の設定値*: <value>↲
Y *現在の設定値*: <value>↲
>

<register>: レジスタ名　A、B、F、X、またはY
<value>:　データパターン　16進数、または2進数(Fレジスタ)("*"によりビットマスクが可能)

例

書式1) >br f **1*↲　　...Cフラグのセットによりブレークが発生するようにレジスタブレーク条件を設定

書式2) >br↲

```
A: -  B: -  X: -     Y: -     EICZ: -
1. set   2. clear ...? 1↲                  ..."1. set"を選択
A          - : a↲                          ...Aレジスタ条件に0xaを設定
B          - : *↲                          ..."*"でレジスタ条件をマスク
FE         - : *↲
FI         - : *↲
FC         - : 1↲
FZ         - : *↲
X          - : 20↲
Y          - : ^↲                          ..."^↲"で直前のガイダンスにリターン
X        20 : 60↲
Y          - : *↲
>br↲
A:a  B:*  X:06  Y:*  EICZ:**1*
1. set   2. clear ...? 2↲                  ..."2. clear"を選択
>br↲
A: -  B: -  X: -     Y: -     EICZ: -
1. set   2. clear ...? ↲                   ...[Enter]で終了
>
```

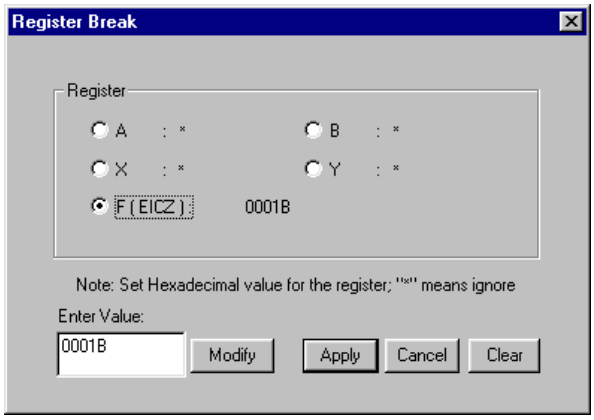
注

- デバッガを起動後はレジスタブレーク条件がすべて解除された状態になりますので、最初にbrコマンドを実行する際にはすべてのパラメータを指定する必要があります。
- ガイダンスモードでは、以下のキー操作も有効です。
  "q↵" ...コマンドを終了(入力を終了し、コマンド機能を実行)
  "^↵" ...直前のアドレスに戻る
  "↵" ...入力をスキップ(現在の内容を保持)

  ガイダンスの途中で"q↵"によりコマンドを終了させた場合、そこまでに指定した条件は変更されます。ただし、解除されている条件が残っている場合は変更されません。
- 直接入力モード時、レジスタ名にA、B、X、Y、またはF以外を指定するとエラーとなります。
  ```
  Error : Incorrect register name, use A/B/X/Y/F
  ```
- 直接入力モードでレジスタブレーク条件すべてを一度に、または1つあるいは複数のレジスタ条件を設定することができます。
- レジスタ値は、各レジスタのビット幅以内で、マスクなし/マスク付きの2進数、あるいは16進数で指定します。この制限を越えた場合はエラーとなります。
  ```
  Error : invalid data pattern
  ```
- プログラムはブレーク成立から1～2命令後に停止します。

GUI

[Break ¦ Register Break...]メニューコマンド
このメニューコマンドを選択すると、レジスタブレーク条件を設定/解除するためのダイアログボックスが表示されます。



レジスタブレーク条件を設定するには、ラジオボタンでレジスタを選択して[Enter Value:]ボックスに値を入力し、[Modify]をクリックしてください。
すべてのレジスタ条件を設定しておく必要があります。ブレーク条件に含めないレジスタは"*"を入力してください。
[Apply]ボタンをクリックすると、ダイアログボックスが閉じ、指定したブレーク条件が設定されます。ただし、未設定のレジスタ("---"で示されます)があると、ブレーク条件は設定されません。
レジスタブレーク条件を解除するには、[Clear]をクリックします。

brc (register break clear)

機 能

設定されているレジスタブレーク条件を解除します。

書 式

>brc↵　　（直接入力モード）

GUI

[Break ¦ Register Break...]メニューコマンド
このメニューコマンドを選択すると、レジスタブレーク条件を設定/解除するためのダイアログボックスが表示されます。（brコマンドの説明を参照してください。）

bs (sequential break)

機 能

シーケンシャルブレークの設定と解除、表示を行います。
3ヵ所までのブレークアドレスと、その中の最後のアドレスの実行回数を指定できます。プログラムは、設定した順序ですべてのアドレスを通過するとともに、最後に指定したアドレスを指定回数実行し、その後もう一度そのアドレスの命令をフェッチするとブレークします。

書 式

(1)>bs <pass> <address1> [<address2> [<address3>]]↵　　(直接入力モード)

(2)>bs↵　　(ガイダンスモード)

現在のシーケンシャルブレーク設定状態
1. set　2. clear　…? <1 | 2>↵　　("2"を選択すると以下のガイダンスなしで実行)
Number of sequential address (1–3) ? : <1 | 2 | 3>↵
Set address ?　*現在の設定値*: <address1>↵
Set address ?　*現在の設定値*: <address2>↵
Set address ?　*現在の設定値*: <address3>↵
Pass count ?　*現在の設定値*: <pass>↵
>

<pass>:　実行回数　10進数
<address1–3>:　プログラム実行アドレス　16進数、またはシンボル(IEEE-695形式のみ)
条件:　0≦address1–3≦プログラムメモリ最終アドレス、0≦pass≦4095

例

書式1) >bs 3 116 120↵　...2つのシーケンシャルアドレスと実行回数を指定します。この例では、CPUがアドレス0x0116の命令を実行後にアドレス0x0120の命令を3回実行し、さらに0x0120の命令をフェッチするとブレークが発生します。

書式2) >bs↵

```
1: -     2: -     3: -     pass: -
1. set   2. clear ...? 1↵                          ..."1. set"を選択
Number of sequential address (1-3) ? : 2↵          ...アドレス数を入力
Set address ? : 116↵                               ...アドレス1を入力
Set address ? : 120↵                               ...アドレス2を入力
Pass count  ? : 3↵                                 ...実行回数を入力
>bs↵
1: 0116  2: 0120  3: -     pass:3
1. set   2. clear ...? 2↵                          ..."2. clear"を選択
>bs↵
1: -     2: -     3: -     pass: -
1. set   2. clear ...? ↵                           ...[Enter]で終了
>
```

ガイダンス途中で[Enter]キーを入力した場合はコマンドがキャンセルされます。

注

- 指定可能な実行回数は最大4095回です。これを越えた指定はエラーとなります。

```
Error : Number of passes out of range, use 0-4095
```

- アドレスは、各機種のプログラムメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

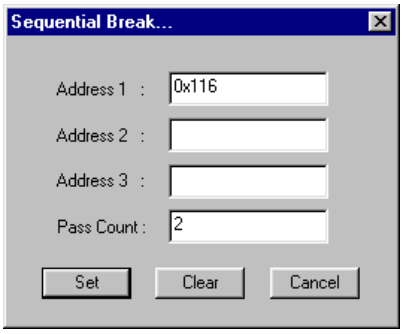
```
Error : invalid value      (no such symbol / symbol type error)
```

  メモリの有効範囲を越えた場合、エラーとなります。

```
Error : Address out of range, use 0-0xXXXX
```

GUI

[Break ¦ Sequential Break...]メニューコマンド
このメニューコマンドを選択すると、シーケンシャルブレーク条件を設定/解除するためのダイアログボックスが表示されます。



シーケンシャルブレーク条件を設定するには、テキストボックスにシーケンシャルアドレスと実行回数を入力後、[OK]ボタンをクリックします。少なくとも1つのアドレス(Address1)と実行回数を設定する必要があります。
シーケンシャルブレーク条件を解除するには、[Clear]をクリックします。

bsc (sequential break clear)

機 能

設定されているシーケンシャルブレーク条件を解除します。

書 式

>bsc↵ （直接入力モード）

GUI

[Break ¦ Sequential Break...]メニューコマンド
このメニューコマンドを選択すると、シーケンシャルブレーク条件を設定/解除するためのダイアログボックスが表示されます。(bsコマンドの説明を参照してください。)

bsp (break stack pointer)

機 能

スタック不当アクセスブレークを発生させるため、スタック領域を指定します。
本コマンドで指定した領域外に対してスタック操作が行われるとブレークします。

書 式

(1)>bsp <address1> <address2> <address3> <address4>↲ (直接入力モード)

(2)>bsp↲ (ガイダンスモード)

現在のスタック領域設定状態
SP1 start address ? : <address1>↲
SP1 end address ? : <address2>↲
SP2 start address ? : <address3>↲
SP2 end address ? : <address4>↲
>

<address1>: SP1先頭アドレス 16進数、またはシンボル(IEEE-695形式のみ)能)
<address2>: SP1終了アドレス 16進数、またはシンボル(IEEE-695形式のみ)能)
<address3>: SP2先頭アドレス 16進数、またはシンボル(IEEE-695形式のみ)能)
<address4>: SP2終了アドレス 16進数、またはシンボル(IEEE-695形式のみ)能)
条件: 0≦address1(2)≦0x03ff、0≦address3(4)≦0x00ff

例

書式1) >bsp 0 3ff 0 ff↲ ...SP1領域を0x0～0x3FFに、SP2領域を0x0～0xFFに設定

書式2) >bsp↲

```
SP1 : 0000 - 03FF    SP2 : 0000 - 00FF
SP1 start address ? : 0↲                    ...アドレスを入力
SP1 end   address ? : 1ff↲
SP2 start address ? : 0↲
SP2 end   address ? : ff↲
>bsp↲
SP1 : 0000 - 01FF    SP2 : 0000 - 00FF
SP1 start address ? : ↲                     ...[Enter]キーで終了
>
```

ガイダンス途中で[Enter]キーを入力した場合はコマンドがキャンセルされます。

注

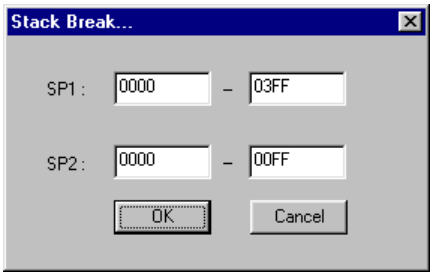
- 設定したスタック領域はプログラム上のスタック操作には影響を与えません。
- SP1アドレスは0～0x3ffの範囲で、SP2アドレスは0～0xffの範囲で指定してください。この範囲を越えるアドレスを入力するとエラーになります。

```
Error : SP1 address out of range, use 0-0x3FF
Error : SP2 address out of range, use 0-0xFF
```

- SP1アドレスは4ワード単位(先頭アドレス=4の倍数、終了アドレス=4の倍数+3)で指定してください。
- SP1はS1C63000 CPUのプリフェッチ機能により、実際に使用するスタック上端+4ワードまでをアクセスする可能性があります。場合により、終了アドレスを+4して設定してください。
- プログラムはブレーク成立から1～2命令後に停止します。

GUI

[Break ¦ Stack Break...]メニューコマンド
このメニューコマンドを選択すると、スタック領域を設定するためのダイアログボックスが表示されます。



スタック領域を設定するには、テキストボックスに先頭および終了アドレスを入力後、[OK]ボタンをクリックします。

bl (break point list)

機 能

すべてのブレーク条件を表示します。

書 式

>bl↵　　（直接入力モード）

例

```
>bl↵
PC break:
    1: 0116
    2: 0200
Sequential break:
1: 0116  2: 0120  3: -     pass:3
Data break:
data: 1***  R/W: W  area: 0000 - 000F
Register break:
A:*  B:*  X:*  Y:*  EICZ:**1*
Stack break:
SP1 : 0000 - 03FF    SP2 : 0000 - 00FF
>
```

GUI

[Break ¦ Break List]メニューコマンド
このメニューコマンドを選択すると、blコマンドが実行されます。

bac (break all clear)

機 能

bp、bd、br、bsコマンドで設定したブレーク条件をすべて解除します。

書 式

>bac↵　　（直接入力モード）

GUI

[Break ¦ Break All Clear]メニューコマンド
このメニューコマンドを選択すると、bacコマンドが実行されます。

8.9.10 プログラム表示コマンド

u (unassemble)

機 能

プログラムを逆アセンブルして[Source]ウィンドウに表示します。表示内容は次のとおりです。

- プログラムメモリアドレス
- オブジェクトコード
- プログラムの逆アセンブル内容

書 式

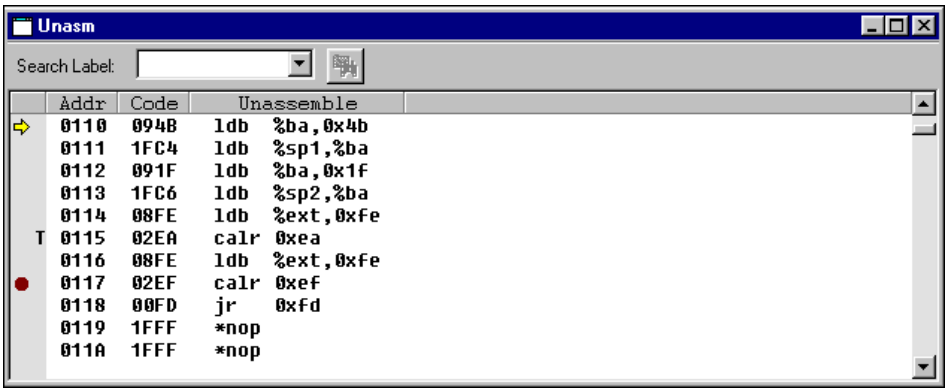
>u [<address>]↲ （直接入力モード）

<address>: 表示開始アドレス 16進数、またはシンボル（IEEE-695形式のみ）

条件: 0≦address≦プログラムメモリ最終アドレス

表 示

(1) [Source]ウィンドウを開いている場合



<address>を省略した場合、[Source]ウィンドウの表示を逆アセンブル形式に変更します。<address>を指定すると、[Source]ウィンドウの表示を逆アセンブル形式に変更するとともに、<address>からコードの表示を行います。

(2) [Source]ウィンドウを閉じている場合

[Command]ウィンドウに16行分の逆アセンブル結果を表示して、コマンド入力待ちとなります。<address>を省略した場合は現在のPC（[Register]ウィンドウに表示）から、<address>を指定すると<address>から表示します。

```
>u↲
 ADDR CODE  UNASSEMBLE
 0110 094B  ldb  %ba,0x4b
 0111 1FC4  ldb  %sp1,%ba
 0112 091F  ldb  %ba,0x1f
 0113 1FC6  ldb  %sp2,%ba
  :    :     :      :
 011E 1FFF  *nop
 011F 1FFF  *nop
>
```

(3) ログ出力中

logコマンドの指定によってコマンド実行結果をログファイルに出力している場合は、[Command]ウィンドウにコードを表示し、その内容をログファイルにも出力します。
[Source]ウィンドウが閉じている場合の表示は上記(2)と同様です。
[Source]ウィンドウが開いていれば、その再表示も行います。この場合、[Command]ウィンドウに表示される行数は[Source]ウィンドウの表示行数と同じになります。

(4) 連続表示機能

uコマンドをキーボードより入力して実行すると、他のコマンドを実行するまでは[Enter]キーの入力のみでコードを連続して表示することができます。
[Enter]キーを入力すると、[Source]ウィンドウは1画面分スクロールします。
[Command]ウィンドウにコードを表示している場合は、前回表示したアドレスに続く16行分(ログ出力中は[Source]ウィンドウと同じ行数)を表示します。

注

表示開始アドレスは、各機種のプログラムメモリ領域の範囲内で指定してください。
アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

```
Error : invalid value      (no such symbol / symbol type error)
```

メモリの有効範囲を越えた場合、エラーとなります。

```
Error : Address out of range, use 0-0xXXXX
```

GUI

[View ¦ Program ¦ Unassemble]メニューコマンド、[Unassemble]ボタン
このメニューコマンドまたはボタンを選択すると、[Source]ウィンドウがアクティブとなり、プログラムを現在のPCアドレスから表示します。

 [Unassemble]ボタン

sc (source code)

機 能

プログラムのソースファイルの内容を[Source]ウィンドウに表示します。表示内容は次のとおりです。

- ソース行番号
- ソースコード

書 式

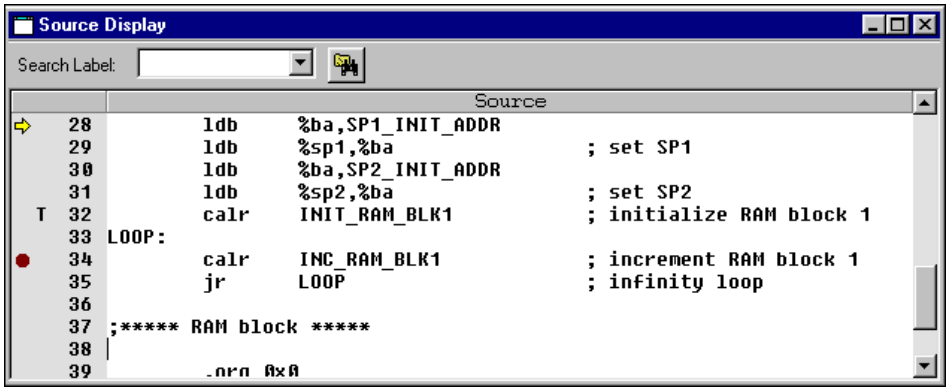
>sc [<address>]↵ （直接入力モード）

<address>: 表示開始アドレス 16進数、またはシンボル（IEEE-695形式のみ）

条件: 0≦address≦プログラムメモリ最終アドレス

表 示

(1) [Source]ウィンドウを開いている場合



<address>を省略した場合、[Source]ウィンドウの表示をソース形式に変更します。<address>を指定すると、[Source]ウィンドウの表示をソース形式に変更するとともに、<address>からコードの表示を行います。

(2) [Source]ウィンドウを閉じている場合

[Command]ウィンドウに16行分のソースを表示して、コマンド入力待ちとなります。

<address>を省略した場合は現在のPC（[Register]ウィンドウに表示）から、<address>を指定すると<address>から表示します。

```
>sc↵
        ldb   %ba,SP1_INIT_ADDR
        ldb   %sp1,%ba              ; set SP1
        ldb   %ba,SP2_INIT_ADDR
        ldb   %sp2,%ba              ; set SP2
        calr  INIT_RAM_BLK1         ; initialize RAM block 1
LOOP:
        ldb   %ext,INC_RAM_BLK1@rh
        calr  INC_RAM_BLK1@rl       ; increment RAM block 1
        ldb   %ext,LOOP@rh
        jr    LOOP@rl               ; infinity loop
>
```

(3) ログ出力中

logコマンドの指定によってコマンド実行結果をログファイルに出力している場合は、[Command]ウィンドウにコードを表示し、その内容をログファイルにも出力します。
[Source]ウィンドウが閉じている場合の表示は上記(2)と同様です。
[Source]ウィンドウが開いていれば、その再表示も行います。この場合、[Command]ウィンドウに表示される行数は[Source]ウィンドウの表示行数と同じになります。

(4) 連続表示機能

scコマンドをキーボードより入力して実行すると、他のコマンドを実行するまでは[Enter]キーの入力のみでコードを連続して表示することができます。
[Enter]キーを入力すると、[Source]ウィンドウは1画面分スクロールします。
[Command]ウィンドウにコードを表示している場合は、前回表示したアドレスに続く16行分(ログ出力中は[Source]ウィンドウと同じ行数)を表示します。

注

- ソースは、ソースデバッグ情報を含むアブソリュートオブジェクトファイルをロードした場合にのみ表示可能です。
- 表示開始アドレスは、各機種のプログラムメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

```
Error : invalid value      (no such symbol / symbol type error)
```

  メモリの有効範囲を越えた場合、エラーとなります。

```
Error : Address out of range, use 0-0xXXXX
```

GUI

[View ¦ Program ¦ Source Display]メニューコマンド、[Source]ボタン
このメニューコマンドまたはボタンを選択すると、[Source]ウィンドウがアクティブとなり、プログラムを現在のPCアドレスから表示します。

 [Source]ボタン

m (mix)

機 能

プログラムの逆アセンブル結果とソースファイルの内容を[Source]ウィンドウに表示します。表示内容は次のとおりです。

- ソース行番号
- プログラムメモリアドレス
- オブジェクトコード
- プログラムの逆アセンブル内容
- ソースコード

書 式

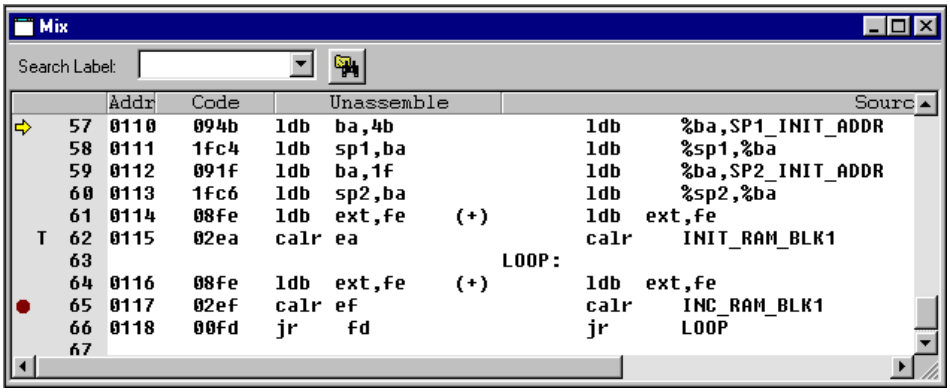
>m [<address>]↲ （直接入力モード）

<address>: 表示開始アドレス 16進数、またはシンボル(IEEE-695形式のみ)

条件: 0≦address≦プログラムメモリ最終アドレス

表 示

(1)[Source]ウィンドウを開いている場合



<address>を省略した場合、[Source]ウィンドウの表示をミックス形式(逆アセンブル&ソース)に変更します。<address>を指定すると、[Source]ウィンドウの表示をミックス形式(逆アセンブル&ソース)に変更するとともに、<address>からコードの表示を行います。

(2)[Source]ウィンドウを閉じている場合

[Command]ウィンドウに16行分のミックス表示を行い、コマンド入力待ちとなります。

<address>を省略した場合は現在のPC([Register]ウィンドウに表示)から、<address>を指定すると<address>から表示します。

```
>m↲
0110   094b   ldb  ba,4b                  ldb     %ba,SP1_INIT_ADDR
0111   1fc4   ldb  sp1,ba                 ldb     %sp1,%ba
0112   091f   ldb  ba,1f                  ldb     %ba,SP2_INIT_ADDR
0113   1fc6   ldb  sp2,ba                 ldb     %sp2,%ba
0114   08fe   ldb  ext,fe    (+)          ldb  ext,fe
0115   02ea   calr ea                     calr    INIT_RAM_BLK1
                                LOOP:
0116   08fe   ldb  ext,fe    (*)          ldb     %ext,INC_RAM_BLK1@rh
0117   02ef   calr ef                     calr    INC_RAM_BLK1@rl
                              (-)          ldb     %ext,LOOP@rh
 :      :      :     :              :       :          :
```

(3) ログ出力中

logコマンドの指定によってコマンド実行結果をログファイルに出力している場合は、[Command]ウィンドウにコードを表示し、その内容をログファイルにも出力します。
[Source]ウィンドウが閉じている場合の表示は上記(2)と同様です。
[Source]ウィンドウが開いていれば、その再表示も行います。この場合、[Command]ウィンドウに表示される行数は[Source]ウィンドウの表示行数と同じになります。

(4) 連続表示機能

mコマンドをキーボードより入力して実行すると、他のコマンドを実行するまでは[Enter]キーの入力のみでコードを連続して表示することができます。
[Enter]キーを入力すると、[Source]ウィンドウは1画面分スクロールします。
[Command]ウィンドウにコードを表示している場合は、前回表示したアドレスに続く16行分(ログ出力中は[Source]ウィンドウと同じ行数)を表示します。

注

- ソースは、ソースデバッグ情報を含むアブソリュートオブジェクトファイルをロードした場合にのみ表示可能です。
- 表示開始アドレスは、各機種のプログラムメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

```
Error : invalid value      (no such symbol / symbol type error)
```

  メモリの有効範囲を越えた場合、エラーとなります。

```
Error : Address out of range, use 0-0xXXXX
```

GUI

[View ¦ Program ¦ Mix Mode]メニューコマンド、[Mix]ボタン
このメニューコマンドまたはボタンを選択すると、[Source]ウィンドウがアクティブとなり、プログラムを現在のPCアドレスから表示します。

 [Mix]ボタン

8.9.11 シンボル情報表示コマンド

sy (symbol list)

機 能

定義されているシンボルのリストを[Command]ウィンドウに表示します。

書 式

(1) >sy [/a]↵ （直接入力モード）
(2) >sy $<keyword> [/a]↵ （直接入力モード）
(3) >sy #<keyword> [/a]↵ （直接入力モード）

<keyword>: 検索文字列 ASCIIキャラクタ
条件: 1≦keywordの長さ≦32

例

書式1) >sy↵

```
INC_RAM_BLK1                                0007
INIT_RAM_BLK1                               0000
RAM_BLK0                                    0000
RAM_BLK1                                    0004
BOOT@C:¥E0C63¥TEST¥MAIN.S                   0110
LOOP@C:¥E0C63¥TEST¥MAIN.S                   0116
NMI@C:¥E0C63¥TEST¥MAIN.S                    0100
>
```

書式1では定義されているシンボルをアルファベット順に表示します。グローバルシンボルから始め、それが終了するとローカルシンボルを表示します。シンボルの右には、そのシンボルに定義されたアドレスが表示されます。

書式2) >sy $R↵

```
INC_RAM_BLK1                                0007
INIT_RAM_BLK1                               0000
RAM_BLK0                                    0000
RAM_BLK1                                    0004
>
```

書式2では、<keyword>で指定した文字列を含むグローバルシンボルを表示します。

書式3) >sy #B↵

```
BOOT@C:¥E0C63¥TEST¥MAIN.S                   0110
>
```

書式3では、<keyword>で指定した文字列を含むローカルシンボルを表示します。

ローカルシンボルには、@とシンボルを定義したソースファイル名が付加されます。

注

- オプションを指定しない場合、シンボルはアルファベット順に表示されます。/aを指定すると、アドレス順に表示されます。
- シンボルリストの表示は、IEEE-695形式のオブジェクトファイルを読み込んでいる場合にのみ可能です。
- <keyword>の指定はアセンブラのシンボル表記の規定に従います。

GUI

なし

8.9.12 ファイル読み込みコマンド

lf (load file)

機 能

IEEE-695形式のオブジェクトファイルを読み込みます。

書 式

(1) >lf <file name>↲ （直接入力モード）

(2) >lf↲ （ガイダンスモード）

File Name ? <file name>↲

>

<file name>: 読み込みファイル名(パスも指定可能)

例

書式1) `>lf test.abs↲`

```
Loading file ... OK!
>
```

書式2) `>lf↲`

```
File name   ? test.abs
Loading file ... OK!
>
```

注

- デバッガが現在使用しているICEパラメータファイル以外を使用してリンクしたオブジェクトファイルを指定した場合、エラーとなります。

  `Error : Different chip type, cannot load this file`

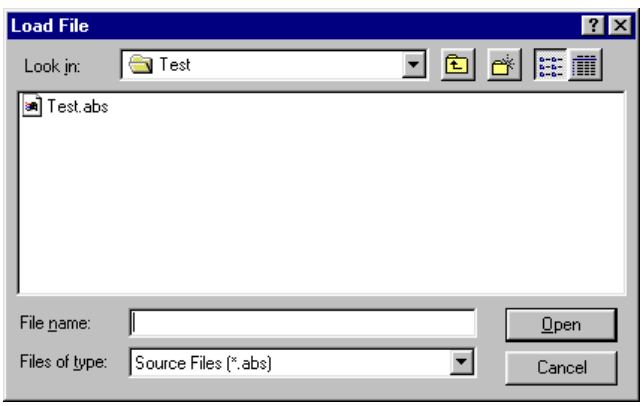
- lfコマンドではIEEE-695形式のオブジェクトファイル(リンカにより生成)以外は読み込めません。
- デバッグにソースの表示やシンボルを使用する場合、デバッグ情報を含んだIEEE-695形式のオブジェクトファイルを読み込む必要があります。
- ファイルの読み込み時に[Source]ウィンドウが開いていれば、その内容を更新します。プログラムの表示は現在のPCアドレスから行われます。
- ファイル読み込み中にエラーが発生した場合、すでに読み込まれた部分はエミュレーションメモリに残ります。

GUI

[File ¦ Load File...]メニューコマンド、[Load File]ボタン

このメニューコマンドまたはボタンを選択すると、読み込むオブジェクトファイルを指定するダイアログボックスが表示されます。


[Load File]ボタン



lo (load option)

機 能

モトローラS形式のプログラムまたはオプションファイルを読み込みます。

ファイル	ファイル名
プログラムファイル	~.hsa(上位5ビット)、~.lsa(下位8ビット)
データファイル*	~.csa
ファンクションオプションデータファイル	~.fsa
セグメントオプションデータファイル*	~.ssa
メロディーデータファイル*	~.msa

* 機種によっては使用しません。

書 式

(1) >lo <file name>↵ (直接入力モード)

(2) >lo↵ (ガイダンスモード)

File Name ...? <file name>↵

>

<file name>: 読み込みファイル名(パスも指定可能)

例

書式1) >lo test.lsa↵ ...プログラムファイルtest.lsaおよびtest.hsaをロード

```
Loading file ... OK!
>
```

書式2) >lo↵

```
File name   ? test.fsa↵   ...ファンクションオプションファイルをロード
Loading file ... OK!
>
```

注

- デバッガは指定されたファイル名でファイルの種類を判断します。このため、上記のファイル名以外のファイルは読み込めません。指定した場合はエラーとなります。

```
Error : invalid file name
```

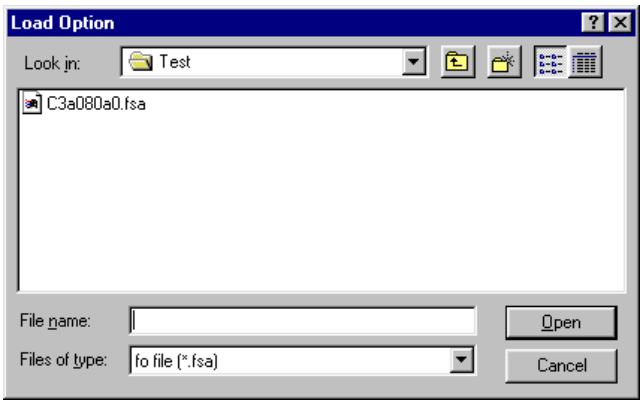
- ファイル読み込み中にエラーが発生した場合、すでに読み込まれた部分はそのまま残ります。

GUI

[File ¦ Load Option...]メニューコマンド、[Load Option]ボタン

このメニューコマンドまたはボタンを選択すると、読み込むHEXファイルを指定するダイアログボックスが表示されます。


[Load Option]ボタン



8.9.13 フラッシュメモリ操作コマンド

注: USBインタフェース版ICEを使用する場合、フラッシュメモリ操作コマンドは使用できません。

lfl (load from flash memory)

機 能

ICE上のフラッシュメモリの内容をターゲットメモリに読み込みます。
前回フラッシュメモリに保存した内容に引き続いてデバッグを行うことができます。

書 式

(1)>lfl <content> [... <content>]↲ (直接入力モード)

(2)>lfl↲ (ガイダンスモード)

```
Read program   1. yes  2. no ...? <1 | 2>↲
     data      1. yes  2. no ...? <1 | 2>↲
     fog       1. yes  2. no ...? <1 | 2>↲
     sog       1. yes  2. no ...? <1 | 2>↲
     mla       1. yes  2. no ...? <1 | 2>↲
読み込み...
>
```

<content>: データ形式 p (program) / d (data) / f (fog) / s (sog) / m (mla)

入力例

書式1) >lfl p↲

```
Loading from flash memory ... done!
```

...プログラムデータの読み込み

書式2) >lfl↲

```
Read program  1.yes  2.no ...? 1↲
     data     1.yes  2.no ...? 1↲
     fog      1.yes  2.no ...? 1↲
     sog      1.yes  2.no ...? 1↲
     mla      1.yes  2.no ...? 1↲
Loading from flash memory ... done!
>
```

...読み込む内容の選択

注

- USBインタフェース版ICEを使用する場合、本コマンドは使用できません。
- フラッシュメモリにプロテクトがかかっている場合、エラーとなり読み込みは行えません。

```
Error : flash ROM is protected
```

- フラッシュメモリが消去されている場合、エラーとなり読み込みは行えません。

```
Error : format error
```

- フラッシュメモリとターゲットメモリのマップ状態が異なる場合(フラッシュメモリに保存したときと現在のデバッグで異なるパラメータファイルを使用している場合)、エラーとなり読み込みは行えません。

```
Error : Map information is not same
```

この場合は、さらに両メモリのマップ情報を表示します。

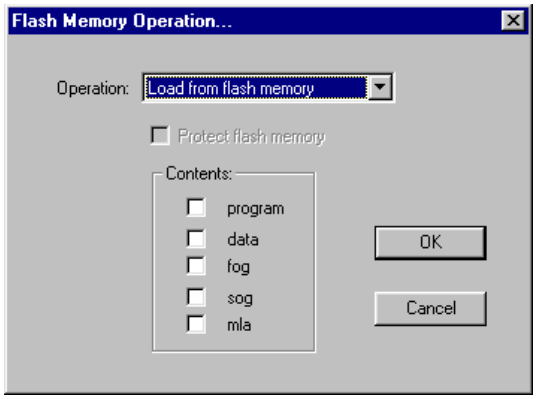
```
                          ICE  flash
---------------------------------
Chip name               63A08
Parameter version         02    00
Size of program         2000     0
        data RAM         800  8000
        data ROM        1000  7000
        ext. memory      100   700
        LCD              2C0   800
        IO                20    20
        FO                20    F0
        SO1                0  1000
        SO2              100  1000
        MLA              510  1000
```

eflやsflコマンドより、正しいパラメータファイルでやり直してください。

- データ読み込み中にエラーが発生した場合、すでに読み込まれた部分はそのまま残ります。

GUI

[File ¦ Flash Memory Operation...]メニューコマンド
このメニューコマンドを選択すると、フラッシュメモリの操作を選択するダイアログボックスが表示されます。



lflコマンドを実行するには、[Operation]リストボックスから"Load from flash memory"を選択し、ロードする内容をチェックして[OK]ボタンをクリックしてください。

sfl (save to flash memory)

機 能

ICE上のターゲットメモリの内容をフラッシュメモリに書き込みます。
フラッシュメモリへの書き込みにより、ICEのフリーランが行えます。また、次回のデバッグを現在の内容から引き続いて行うことも可能となります。

書 式

(1)>sfl <content> [... <content>] [-p]↲　　(直接入力モード)

(2)>sfl↲　　(ガイダンスモード)

```
Protect flash memory    1. yes  2. no ...? <1 | 2>↲
        Write program   1. yes  2. no ...? <1 | 2>↲
              fog       1. yes  2. no ...? <1 | 2>↲
              sog       1. yes  2. no ...? <1 | 2>↲
              mla       1. yes  2. no ...? <1 | 2>↲
読み込み...
>
```

<content>: データ形式　p (program) / d (data) / f (fog) / s (sog) / m (mla)
P:　　プロテクトオプション

入力例

書式1) >sfl p d f s m -p↲　　...すべての内容の保存とプロテクトの設定

```
Please wait few minutes
Save to flash memory ... done!
>
```

書式2) >sfl↲

```
Protect flash memory    1.yes   2.no ...? 1↲     ...プロテクトを設定
        Write program   1.yes   2.no ...? 1↲     ...書き込み内容の選択
              data      1.yes   2.no ...? 1↲
              fog       1.yes   2.no ...? 1↲
              sog       1.yes   2.no ...? 1↲
              mla       1.yes   2.no ...? 1↲
Please wait few minutes
Save to flash memory ... done!
>
```

※ ガイダンスの途中で[Enter]キーのみを入力した場合はガイダンスを終了し、それまでに選択した領域のみをフラッシュメモリに書き込みます。

注

- USBインタフェース版ICEを使用する場合、本コマンドは使用できません。
- フラッシュメモリにプロテクトがかかっている場合、エラーとなり書き込みは行えません。

```
Error : flash ROM is protected
```

プロテクトはeflコマンドでフラッシュメモリを消去することによって解除されます。

- フラッシュメモリが消去されている場合、エラーとなります。この場合は処理の続行/中止を選択することができます。

```
Error : format error
Save with the map information,
 or quit the command ?  1.save  2.quit ...? 1
Protect flash memory    1.yes   2.no ...?
```

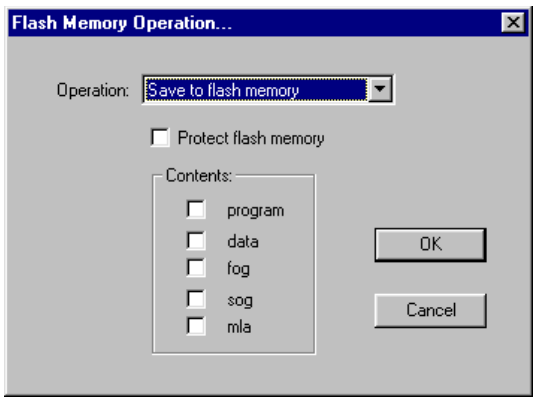
- フラッシュメモリとターゲットメモリのマップ状態が異なる場合、エラーとなります。両メモリのマップ情報を表示し、処理の続行/中止を選択するメッセージを表示します。

```
Error : Map information is not the same
                      ICE  flash
--------------------------------
Chip name            63A08
Parameter version      02    00
Size of program      2000     0
        data RAM      800  8000
        data ROM     1000  7000
        ext. memory   100   700
        LCD           2C0   800
        IO             20    20
        FO             20    F0
        SO1             0  1000
        SO2           100  1000
        MLA           510  1000
Save with the map information,
 or quit the command ?  1.save  2.quit ...? 1
Protect flash memory    1.yes   2.no ...?
```

- 工場出荷時やeflコマンドによる消去後、フラッシュメモリ内の全データは0xffになっています。その後、slfコマンドでプログラム等一部のデータのみを書き込んでも、その他のデータが0xffのままとなります。その状態ではICEのフリーランが行えません。フリーランを行う場合はフラッシュメモリの消去後、必ず全データの書き込みを行ってください。
  またICEではオプションデータのデフォルト値がすべて0x00となっています。オプションデータをロード(loコマンド)する前にフラッシュメモリに書き込みを行うと、0x00が書き込まれてしまいますので注意してください。

GUI

[File ¦ Flash Memory Operation...]メニューコマンド
このメニューコマンドを選択すると、フラッシュメモリの操作を選択するダイアログボックスが表示されます。



sflコマンドを実行するには、[Operation]リストボックスから"Save to flash memory"を選択し、ロードする内容をチェックして[OK]ボタンをクリックしてください。
[Protect flash memory]チェックボックスで-pオプションが指定できます。

efl (erase flash memory)

機 能

ICE上のフラッシュメモリの内容をマップ情報も含めて消去し、プロテクトを解除します。

書 式

efl↵ （直接入力モード）

入力例

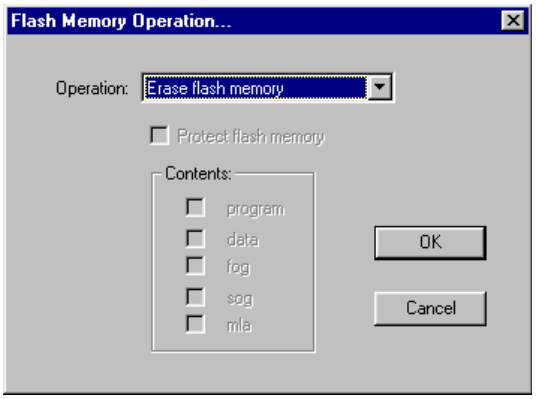
```
>efl↵
Clear flash memory ... done!
>
```

注

- USBインタフェース版ICEを使用する場合、本コマンドは使用できません。
- eflコマンドによる消去後、フラッシュメモリ内の全データは0xffになります。その後、slfコマンドでプログラム等一部のデータのみを書き込んでも、その他のデータが0xffのままとなります。その状態ではICEのフリーランが行えません。フリーランを行う場合はフラッシュメモリの消去後、必ず全データの書き込みを行ってください。
  またICEではオプションデータのデフォルト値がすべて0x00となっています。オプションデータをロード(loコマンド)する前にフラッシュメモリに書き込みを行うと、0x00が書き込まれてしまいますので注意してください。

GUI

[File ¦ Flash Memory Operation...]メニューコマンド
このメニューコマンドを選択すると、フラッシュメモリの操作を選択するダイアログボックスが表示されます。



eflコマンドを実行するには、[Operation]リストボックスから"Erase flash memory"を選択し、[OK]ボタンをクリックします。

8.9.14 トレースコマンド

tm (trace mode display/set)

機 能

トレースモードの設定、表示を行います。
次の3種類のトレースモードとトレーストリガポイント(指定のアドレスを実行するとTRGOUT端子からパルスが出力されます)が設定可能です。

1. ノーマルトレースモード
   常に最新のトレース情報をトレースメモリに書き込みます。
2. シングルディレイトリガトレースモード
   トレーストリガポイントを基準に次の3種類のトレース採取領域を指定できます。
   ・Start: トレーストリガポイントからトレース情報を採取
   ・Middle: トレーストリガポイントの前後のトレース情報を採取
   ・End: トレーストリガポイントまでトレース情報を採取
3. PC範囲指定トレースモード
   指定アドレスの範囲内または範囲外の実行によってトレースを行います。その範囲を4ヵ所まで指定可能です。

書 式

(1)>tm <mode> <trigger> [<option>] [<addr1> <addr2> [... <addr7> <addr8>]↲ (直接入力モード)

(2)>tm↲ (ガイダンスモード)

現在設定
1. normal 2. single delay 3. address area ... ? <1 | 2 | 3>↲
Trigger address ? : <trigger>↲
..... (上記の選択に従ってガイダンスを表示)
>

<mode>: トレースモード -n(ノーマル)、-s(シングルディレイ)または-a(PC範囲指定)
<trigger>: トレーストリガアドレス 16進数、またはシンボル(IEEE-695形式のみ)
<option>: シングルディレイトレースモード s(Start) / m(Middle) / e(End)
PC範囲指定トレースモード i(範囲内) / o(範囲外)
<addr1–8>: アドレス範囲 16進数、またはシンボル(IEEE-695形式のみ)
条件: 0≦trigger、addr1–8≦プログラムメモリ最終アドレス

例

書式1) >tm -n 116↲ ...ノーマルトレースモードに設定し、トリガポイントをアドレス0x0116に設定

書式2) >tm↲

```
Normal mode
Trigger Address : 0
1.normal  2.single delay  3.address area ...? 1↲      ..."1. normal"を選択
Trigger address ? :116↲                               ...トリガアドレスを入力
>tm↲
Normal mode
Trigger Address : 0116
1.normal  2.single delay  3.address area ...? 2↲      ..."2. single delay"を選択
Trigger address ? :116↲                               ...トリガアドレスを入力
1.start  2.middle  3.end ...? 2↲                      ...トレース採取領域を選択
>
```

```
>tm↲
Single delay mode
Trigger Address : 0116
Position: Middle
1.normal  2.single delay  3.address area ...? 3↲   ..."3. address area"を選択
Trigger address ? :116↲                             ...トリガアドレスを入力
1.in area  2.out area ...? 1↲                       ... PC範囲内/範囲外を選択
Start address ? 110↲                                ...最大4ヶ所までのPC範囲を入力
End address ? 200↲
Start address ? ↲                                   ...[Enter]で終了
>
```

[Enter]キーのみを入力した場合はコマンドがキャンセルされます。
ただし、PC範囲指定トレースモードの選択において、1組以上のPC範囲が指定されている場合(上記例では1組を指定)はその範囲をトレース領域として設定します。

注

- [Source]ウィンドウ上には、ここで設定したトリガアドレスの行の先頭に"T"が表示されます。
- アドレスは、各機種のプログラムメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

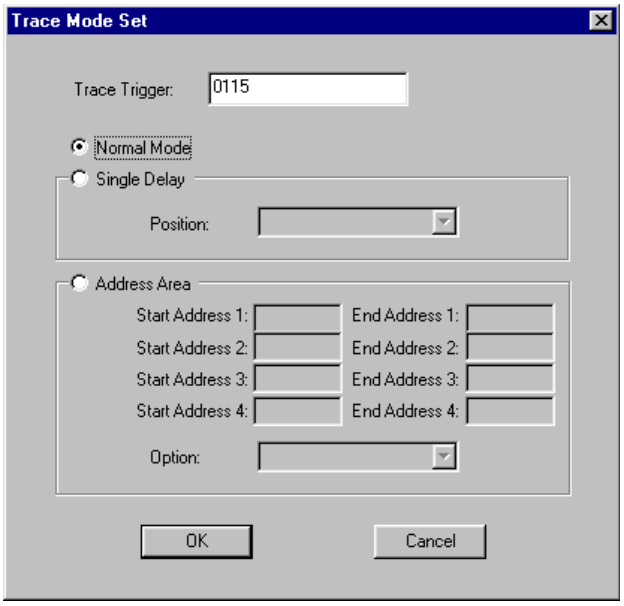
  ```
  Error : invalid value      (no such symbol / symbol type error)
  ```

  メモリの有効範囲を越えた場合、エラーとなります。

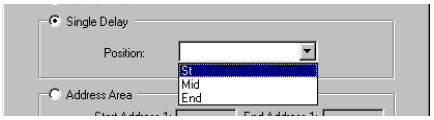
  ```
  Error : Address out of range, use 0-0xXXXX
  ```

GUI

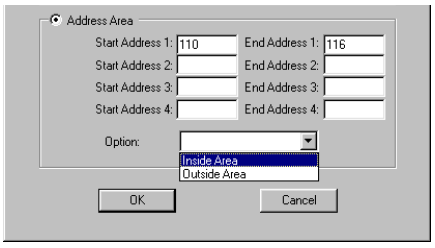
[Trace ¦ Trece Mode Set...]メニューコマンド
このメニューコマンドを選択すると、トレースモードを選択するダイアログボックスが表示されます。ラジオボタンでトレースモードを選択してください。アドレスを入力して、[OK]ボタンをクリックします。



ノーマルトレースモード



シングルディレイトレースモード



PCアドレス範囲トレースモード

td (trace data display)

機 能

ICEのトレースメモリに採取したトレース情報を表示します。

書 式

(1)>td [cycle]↲ (直接入力モード)

(2)>td↲ (ガイダンスモード)

Start point ?: (ENTER from the latest) <num>↲

(トレース情報が表示されます)

>

<cycle>: トレースサイクル番号 10進数(0～8,191)

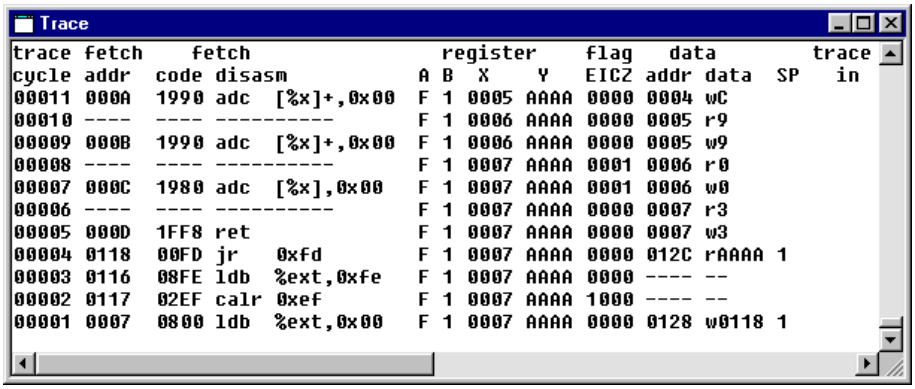
表 示

トレース情報の表示内容は次のとおりです。

trace cycle:	トレースサイクル(10進数)トレースメモリ内に最後に取り込まれた情報が00001となります。
fetch addr:	フェッチアドレス(16進数)
fetch code disasm:	フェッチコード(16進表示)と逆アセンブル内容
register:	サイクル実行後のA、B、X、Yレジスタ値(16進数)
flag:	サイクル実行後のE、I、C、Zフラグの状態(2進数)
data:	アクセスしたデータメモリアドレス(16進数)、リード/ライト(データの先頭のrまたはw)、データ(4ビットアクセスの場合1桁の16進数、16ビットアクセスの場合4桁の16進数)
SP:	スタックアクセス(SP1アクセス時が1、SP2アクセス時が2)
trace in:	TRCIN端子の入力(Lowの信号入力時にLを表示)

(1)[Trace]ウィンドウが開いている場合

<cycle>を省略してtdコマンドを実行すると、[Trace]ウィンドウは最新のデータを再表示します。
<cycle>を指定してtdコマンドを実行すると、指定したサイクルから表示されます。
[Trace]ウィンドウの内容は、ターゲットプログラムの実行後に更新されます。
スクロールによってすべてのトレースデータを表示させることができます。



(2) [Trace]ウィンドウが閉じている場合

<cycle>を省略してtdコマンドを実行すると、11行分の最新データを[Command]ウィンドウに表示します。
<cycle>を指定してtdコマンドを実行すると、指定したトレース番号から11行分のデータを[Command]ウィンドウに表示します。

```
>td↵
Start point ?:(ENTER from the latest)↵
trace fetch    fetch                 register    flag   data        trace
cycle addr  code disasm           A B  X    Y   EICZ addr data  SP   in
00011 0118  00FD jr   0xfd        F 1 0007 AAAA 0000 012C rAAAA 1
00010 0116  08FE ldb  %ext,0xfe   F 1 0007 AAAA 0000 ---- --
00009 0117  02EF calr 0xef        F 1 0007 AAAA 1000 ---- --
00008 0007  0800 ldb  %ext,0x00   F 1 0007 AAAA 0000 0128 w0118 1
00007 0008  0A04 ldb  %xl,0x04    F 1 0007 AAAA 1000 ---- --
00006 0009  1911 add  [%x]+,0x01  F 1 0004 AAAA 0000 ---- --
00005 ----  ---- ----------       F 1 0005 AAAA 0000 0004 rD
00004 000A  1990 adc  [%x]+,0x00  F 1 0005 AAAA 0000 0004 wE
00003 ----  ---- ----------       F 1 0006 AAAA 0000 0005 r5
00002 000B  1990 adc  [%x]+,0x00  F 1 0006 AAAA 0000 0005 w5
00001 ----  ---- ----------       F 1 0007 AAAA 0000 0006 rE
>td 10↵
trace fetch    fetch                 register    flag   data        trace
cycle addr  code disasm           A B  X    Y   EICZ addr data  SP   in
00020 0009  1911 add  [%x]+,0x01  F 1 0004 AAAA 0000 ---- --
00019 ----  ---- ----------       F 1 0005 AAAA 0000 0004 rC
00018 000A  1990 adc  [%x]+,0x00  F 1 0005 AAAA 0000 0004 wD
00017 ----  ---- ----------       F 1 0006 AAAA 0000 0005 r5
00016 000B  1990 adc  [%x]+,0x00  F 1 0006 AAAA 0000 0005 w5
00015 ----  ---- ----------       F 1 0007 AAAA 0000 0006 rE
00014 000C  1980 adc  [%x],0x00   F 1 0007 AAAA 0000 0006 wE
00013 ----  ---- ----------       F 1 0007 AAAA 0000 0007 r4
00012 000D  1FF8 ret              F 1 0007 AAAA 0000 0007 w4
00011 0118  00FD jr   0xfd        F 1 0007 AAAA 0000 012C rAAAA 1
00010 0116  08FE ldb  %ext,0xfe   F 1 0007 AAAA 0000 ---- --
>
```

(3) ログ出力中

logコマンドの指定によってコマンド実行結果をログファイルに出力している場合は、 [Command]ウィンドウにトレースデータを表示し、その内容をログファイルにも出力します。
[Trace]ウィンドウが閉じている場合の表示は上記(2)と同様です。
[Trace]ウィンドウが開いていれば、その再表示も行います。この場合、[Command]ウィンドウに表示される行数は[Trace]ウィンドウの表示行数と同じになります。

(4) 連続表示機能

tdコマンドを実行すると、他のコマンドを実行するまでは[Enter]キーの入力のみでトレースデータを連続して表示することができます。
[Enter]キーを入力すると、[Trace]ウィンドウは1画面分前にスクロールします。
[Command]ウィンドウに表示している場合は、前回表示したサイクルの前の11行分(ログ出力中は[Trace]ウィンドウと同じ行数)を表示します。

表示方向は、[Enter]キーの入力ごとに古い実行サイクル(FORWARD)に向かいますが、[B]キーによってこの方向を逆(BACKWORD)にすることができます。表示方向を戻すには[F]キーを入力します。[Trace]ウィンドウが開いている場合は、そのスクロール方向も変更されます。

```
>td 100↲                                  ...FORWORDで表示開始
(サイクルNo.110～100のデータを表示)
>b↲                                       ...BACKWORDに変更
(サイクルNo. 99～89のデータを表示)
>↲                                        ...BACKWORDで表示を継続
(サイクルNo.88～78のデータを表示)
>f↲                                       ...FORWORDに変更
(サイクルNo. 99～89のデータを表示)
>
```

注

- トレースサイクルNo.は0～8,191の領域内で指定してください。これを越えた場合、エラーとなります。
  `Error : Address out of range, use 0-8191`
- トレースメモリはプログラムの実行がブレークするまで新しいデータを取り込みます。トレース情報がトレースメモリの容量を越えた場合は、古いデータから上書きされます。
- ICEの動作タイミングの関係上、トレース開始時のフェッチサイクル、トレース終了時の実行サイクル等、境界上のサイクルが記録されない場合があります。

GUI

[View ¦ Trace]メニューコマンド
このメニューコマンドを選択すると、[Trace]ウィンドウが開き、最新のトレース情報を表示します。

ts (trace search)

機 能

トレースメモリの中から指定した条件でトレース情報を検索します。
検索条件を次の3種類から選択することができます。

1. 実行したアドレスによる検索
   プログラムメモリアドレスを指定して、そのアドレスを実行したサイクルを検索します。
2. 指定メモリの読み出しサイクルを検索
   データメモリアドレスを指定して、そのアドレスから読み出しを行ったサイクルを検索します。
3. 指定メモリへの書き込みサイクルを検索
   データメモリアドレスを指定して、そのアドレスへの書き込みを行ったサイクルを検索します。

書 式

(1)>ts <option> <address>↲　　(直接入力モード)

(2)>ts↲　　(ガイダンスモード)

1. pc address 2. data read address 3. data write address ...? <1 | 2 | 3>↲
Search address ?: <address>↲
(検索結果を表示)
>

<option>: 検索条件 pc(=実行アドレス)、dr(=データ読み出し)、dw(=データ書き込み)
<address>: 検索アドレス 16進数、またはシンボル(IEEE-695形式のみ)

例

検索結果は、[Trace]ウィンドウが開いていれば[Trace]ウィンドウに表示されます。[Trace]ウィンドウが閉じている場合は、tdコマンドと同様に[Command]ウィンドウに表示されます。

書式1) >ts pc 116↲

```
Trace searching ... Done!
trace fetch    fetch                register    flag   data        trace
cycle addr  code disasm           A B  X    Y   EICZ addr data  SP   in
00010 0116  08FE ldb  %ext,0xfe   F 1 0007 AAAA 0000 ---- --
>
```

書式2) >ts↲

```
1.pc address  2.data read address  3.data write address ...? 1↲
Search address ?:116↲
Trace searching ... Done!
trace fetch    fetch                register    flag   data        trace
cycle addr  code disasm           A B  X    Y   EICZ addr data  SP   in
00010 0116  08FE ldb  %ext,0xfe   F 1 0007 AAAA 0000 ---- --
>
```

logコマンドの指定によってコマンド実行結果をログファイルに出力しているときには、[Trace]ウィンドウが開いている場合でも、検索結果が[Command]ウィンドウに表示され、ログファイルに出力されます。

注

アドレスは、各機種のプログラム/データメモリ領域の範囲内で指定してください。
アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

```
Error : invalid value      (no such symbol / symbol type error)
```

プログラムメモリの有効範囲を越えた場合、エラーとなります。

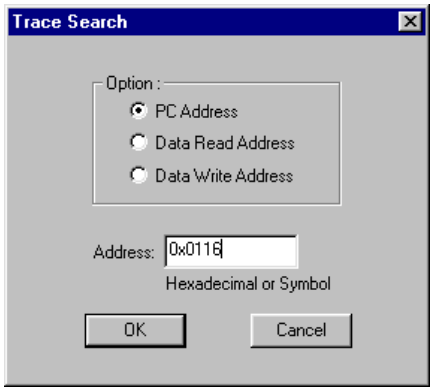
```
Error : Address out of range, use 0-0xXXXX
```

データメモリの有効範囲を越えた場合、エラーとなります。

```
Error : Address out of range, use 0-0xFFFF
```

GUI

[Trace ¦ Trace Search...]メニューコマンド
このメニューコマンドを選択すると、検索条件を設定するダイアログボックスが表示されます。



ラジオボタンでオプションを選択して、アドレスをテキストボックスに入力し、[OK]ボタンをクリックします。

tf (trace file)

機 能

td、tsコマンドの実行によって現在表示されているトレースデータの中から、指定範囲のデータをファイルに保存します。

書 式

(1)>tf [<cycle1> [<cycle2>]] <file name>↲ （直接入力モード）

(2)>tf↲ （ガイダンスモード）

```
Start cycle number (max 8191) ? : <cycle1>↲
End cycle number  (min       0) ? : <cycle2>↲
File Name     ? : <file name>↲
>
```

<cycle1>: 開始サイクル数 10進数(max 8,191)
<cycle2>: 終了サイクル数 10進数(min 0)
<file name>: 出力ファイル名(パスも指定可能)

例

書式1) >tf trace.trc↲ ...tdコマンドで表示させたすべてのトレース情報を保存

```
8191-8000
8000-7000
    :
1000-   1
OK!
>
```

書式2) >tf↲

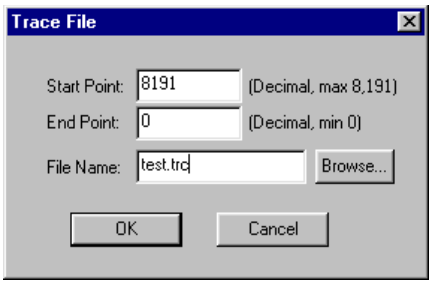
```
Start cycle number (max 8191) ? :1000↲
End cycle number   (min    0) ? :1↲
File name   ? :test.trc↲
1000-   1
OK!
>
```

注

- 既存のファイルを指定すると、データを上書きします。
- <cycle1>のデフォルト値は最終データの位置、<cycle2>のデフォルト値は"1"です。

GUI

[Trace ¦ Trace File...]メニューコマンド
このメニューコマンドを選択すると、パラメータを設定するダイアログボックスが表示されます。



開始サイクル数、終了サイクル数、ファイル名を入力し、[OK]ボタンをクリックします。
すべてのトレース情報を保存するには、[Start Point]と[End Point]は空白のままにしてください。
ファイル名は、[Browse...]ボタンで表示される標準ファイル選択ダイアログボックスによっても選択可能です。

8.9.15 カバレッジコマンド

cv (coverage)

機 能

カバレッジ情報(プログラムが実行されたアドレス)を[Command]ウィンドウに表示します。

書 式

>cv [<address1> [<address2>]]↲ (直接入力モード)

<address1>: 開始アドレス 16進数、またはシンボル(IEEE-695形式のみ)
<address2>: 終了アドレス 16進数、またはシンボル(IEEE-695形式のみ)
条件: 0≦address1≦address2≦プログラムメモリ最終アドレス

例

```
>cv 100 1ff↲                ...0x100～0x1ffの範囲で実行したアドレスを表示
Coverage Information:
    1: 0100..0102
    2: 0110..0118
>cv↲                        ...実行したすべてのアドレスを表示
Coverage Information:
    1: 0000..000d
    2: 0100..0102
    3: 0110..0118
>
```

注

- <address1>と<address2>を省略すると、全プログラムメモリアドレスを対象としてカバレッジ情報を表示します。<address1>と<address2>を指定すると、指定したアドレス範囲内のカバレッジ情報を表示します。<address1>のみを指定した場合、指定したアドレスからプログラムメモリ最終アドレスまでの範囲内のカバレッジ情報を表示します。
- アドレスは、各機種のプログラムメモリ領域の範囲内で指定してください。
  アドレスが16進数、または有効なシンボル以外の場合、エラーとなります。

  ```
  Error : invalid value      (no such symbol / symbol type error)
  ```

  メモリの有効範囲を越えた場合、エラーとなります。

  ```
  Error : Address out of range, use 0-0xXXXX
  ```

- 先頭アドレスが終了アドレスより大きい場合、エラーとなります。

  ```
  Error : end address < start address
  ```

GUI

なし

cvc (coverage clear)

機 能

カバレッジ情報をクリアします。

書 式

>cvc↵」　　（直接入力モード）

GUI

なし

8.9.16 コマンドファイル実行コマンド

com (execute command file)

機 能

コマンドファイルを読み込み、その中に記述されたデバッグコマンドを連続実行します。各コマンドの実行間隔を0～30秒の範囲で1秒間隔に指定可能です。

書 式

(1)>com <file name> [<interval>]↲ （直接入力モード）

(2)>com↲ （ガイダンスモード）

File name ? <file name>↲

Execute commands 1. successively 2. with wait ...? <1 | 2>↲

Interval (0 - 256 seconds) : <interval>↲ （"2. with wait"選択時にのみ表示）

>*(実行コマンドを表示)*

<file name>: コマンドファイル名(パスも指定可能)

<interval>: コマンドの実行間隔(ウェイト時間) 10進数(0～256)

例

書式1) >com batch1.cmd↲

>..... ..."batch1.com"に記述されたコマンドを連続実行

書式2) >com↲

```
File name   ? test.cmd↲
Execute commands   1. successively   2. with wait   ...? 2↲
Wait time (0 - 256 seconds) : 2↲
>.....                          ...各コマンド実行後に2秒の待ち時間を挿入
```

注

- コマンドファイルにコマンド以外の内容は記述しないでください。
- 指定のファイルが見つからない場合はエラーとなります。

```
Error : Cannot open file
```

- コマンドファイル内から別のコマンドファイルを読み込むことも可能です。ただし、最大5階層までに制限されます。6階層目のcom(cmw)コマンドが現れるとエラーとなり、そのcom(cmw)コマンドで指定されるコマンドファイルは実行されません。そのcom(cmw)コマンドをスキップし、それ以後の実行を継続します。

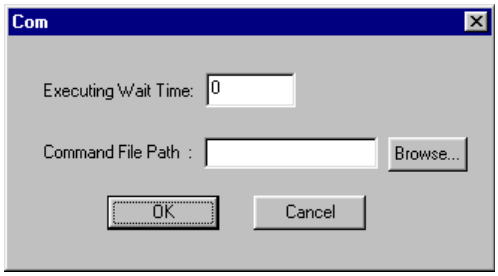
```
Error : Maximum nesting level(5) is exceeded, cannot open file
```

- 実行間隔に256以上の数値を指定した場合、256秒が指定されたものとして処理されます。
- [Ctrl]+[Q]キーの入力により、コマンドファイルの実行を停止することができます。

GUI

[Run ¦ Command File...]メニューコマンド

このメニューコマンドを選択すると、コマンドファイルを選択するダイアログボックスが表示されます。



実行間隔とファイル名を入力し、[OK]ボタンをクリックします。

ファイル名は、[Browse...]ボタンで表示される標準ファイル選択ダイアログボックスによっても選択可能です。

cmw (execute command file with wait)

機 能

コマンドファイルを読み込み、その中に記述されたデバッグコマンドを一定時間ごとに実行します。個々のコマンドの実行間隔は、mdコマンドにより1～256秒の範囲(1秒単位)で設定できます。デバッガの初期設定は1秒です。

書 式

(1)>cmw <file name>↵ (直接入力モード)

(2)>cmw↵ (ガイダンスモード)

File name ? <file name>↵

>(*実行コマンドを表示*)

<file name>: コマンドファイル名(パスも指定可能)

例

書式1) >cmw batch1.cmd↵

```
>.....
```

書式2) >cmw↵

```
File name   ? test.cmd↵
>.....
```

注

- コマンドファイルにコマンド以外の内容は記述しないでください。
- 指定のファイルが見つからない場合はエラーとなります。

```
Error : Cannot open file
```

- コマンドファイル内から別のコマンドファイルを読み込むことも可能です。ただし、最大5階層までに制限されます。6階層目のcmw(com)コマンドが現れるとエラーとなり、そのcmw(com)コマンドで指定されるコマンドファイルは実行されません。そのcmw(com)コマンドをスキップし、それ以後の実行を継続します。

```
Error : Maximum nesting level(5) is exceeded, cannot open file
```

- comコマンドにより読み込むコマンドファイル内にcmwコマンドを記述すると、それ以降のファイル内のコマンドはすべて(comコマンドがあった場合でも)指定の時間ごとに実行されます。
- [Ctrl]+[Q]キーの入力により、コマンドファイルの実行を停止することができます。

GUI

なし

ただし、[Run]メニューの[Command File...]で同様の機能が実行可能です(comコマンド参照)。

rec (record commands file to file)

機 能

実行したデバッグコマンドを指定のコマンドファイルに保存します。

書 式

(1) >rec <file name>↲ （直接入力モード）

(2) >rec↲ （ガイダンスモード） ...ガイダンスについては例を参照

<file name>: コマンドファイル名(パスも指定可能)

例

(1)デバッガ起動後、最初のrecコマンド実行時

```
>rec↲
File name   ? sample.cmd↲
1. append   2. clear and open   ...? 2↲      ...既存のファイルを指定した場合に表示
>
```

(2)2回目以降のrecコマンド実行時

```
>rec↲
Set to record off mode.            ...recコマンド実行ごとにレコードON/OFFを切り換え
.....
>rec↲
Set to record on mode.
```

注

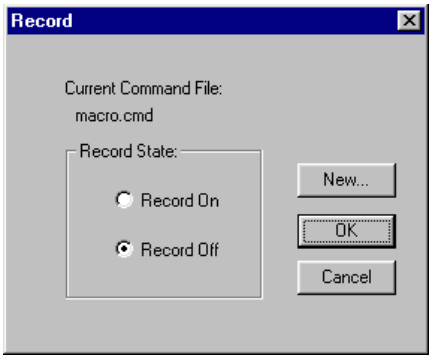
- レコード機能がONの場合、[Command]ウィンドウに直接入力したコマンド以外にも、メニューやツールバーボタンで選択したコマンド([Help]メニューコマンド/ボタンを除く)が[Command]ウィンドウに表示され、指定のファイルに出力されます。
  [Register]ウィンドウ、[Data]ウィンドウで直接レジスタ値やデータメモリの内容を修正した場合、あるいは[Source]ウィンドウ内でのダブルクリックによってブレークポイントを設定した場合も、対応するコマンドが[Command]ウィンドウに表示され、ファイルに出力されます。
- 最初にrecコマンドを実行する場合は、それに続く実行コマンドを記録するためのファイル名を指定する必要があります。
- 一度コマンドファイルが開かれると、それ以降はrecコマンドの実行ごとに記録を中断、再開(トグル)します。この切り換えは、デバッガを終了するまで有効です。コマンドの記録を別のファイルに変更するには、書式1を使用してファイルを指定し直してください。その時点でそれまでのコマンドファイルは閉じられ、以降の記録は新しく指定したファイルに対して行われます。
- 頻繁に使用する一連のコマンドをrecコマンドでコマンドファイルに記録することにより、2回目の実行からはcom、cmwコマンドが使用できます。

GUI

[Option ¦ Record...]メニューコマンド

このメニューコマンドを選択すると、コマンドファイルを指定する標準のファイル選択ダイアログボックスが表示されます。

すでにコマンドの記録を開始している場合は、レコードON/OFFを切り換えるダイアログボックスが表示されます。ここで、新しいコマンドファイルを指定するには、[New...]ボタンをクリックしてください。



8.9.17 ログコマンド

log (log)

機 能

入力したコマンドと実行結果をファイルに保存します。

書 式

(1) >log <file name>↲ (直接入力モード)

(2) >log↲ (ガイダンスモード) ...ガイダンスについては例を参照

<file name>: ログファイル名(パスも指定可能)

例

(1)デバッガ起動後、最初のlogコマンド実行時

```
>log↲
File name   ? debug1.log↲
1. append   2. clear and open   ...? 2↲      ...既存のファイルを指定した場合に表示
>
```

(2)2回目以降のlogコマンド実行時

```
>log↲
Set to log off mode.                ...logコマンド実行ごとにログ出力ON/OFFを切り換え
.....
>log↲
Set to log on mode.
```

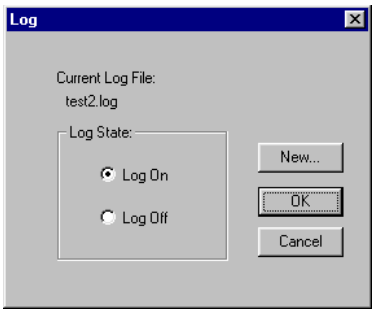
注

- ログファイルには[Command]ウィンドウに表示された内容がそのままファイルに書き込まれます。
  ログ出力中にはメニューやツールバーボタンで選択したコマンド([Help]メニューコマンド/ボタンを除く)も[Command]ウィンドウに表示され、指定のファイルに出力されます。
  [Register]ウィンドウ、[Data]ウィンドウで直接レジスタ値やデータメモリの内容を修正した場合、あるいは[Source]ウィンドウ内でのダブルクリックによってブレークポイントを設定した場合も、対応するコマンドが[Command]ウィンドウに表示され、ファイルに出力されます。

  コマンドの実行による[Source]、[Data]、[Trace]、[Register]ウィンドウへの表示内容が、[Command]ウィンドウにも表示されます。オンザフライ情報も表示されます。
  ただし、ウィンドウ操作コマンドによる表示の更新や、各ウィンドウのスクロールバーまたは矢印キーによるスクロール結果は表示されません。
- 最初にlogコマンドを実行する場合は、それに続く実行コマンドと実行結果を記録するためのファイル名を指定する必要があります。
- 一度ログファイルが開かれると、それ以降はlogコマンドの実行ごとに記録を中断、再開(トグル)します。この切り換えは、デバッガを終了するまで有効です。ログ出力を別のファイルに変更するには、書式1を使用してファイルを指定し直してください。その時点でそれまでのログファイルは閉じられ、以降の記録は新しく指定したファイルに対して行われます。

GUI

[Option ¦ Log...]メニューコマンド
このメニューコマンドを選択すると、ログファイルを指定する標準のファイル選択ダイアログボックスが表示されます。
すでにログ出力を開始している場合は、出力ON/OFFを切り換えるダイアログボックスが表示されます。ここで、新しいログファイルを指定するには、[New...]ボタンをクリックしてください。



8.9.18 マップ情報表示コマンド

ma (map information)

機 能

パラメータファイルによって設定されたマップ情報を表示します。

書 式

>ma↵　　（直接入力モード）

例

コマンド入力後、チップ名、パラメータファイルのバージョン、各領域のマップ情報を表示します。引き続き[Enter]キーを入力することによりI/O領域、LCD領域のマップ情報を表示します。

```
>ma↵
Chip name               : 63A08
Parameter file version  : 02
Program area            : 0000 - 1FFF
Data ram area           : 0000 - 07FF
Data rom area           : 8000 - 8FFF
LCD area                : F000 - F2BF
External memory area    : F800 - F8FF
IO area                 : FF00 - FFFF
Size of FO area         : 32
Size of SO1 area        : 0
Size of SO2 area        : 256
Size of MLA area        : 1296
>↵
IO Area
     01234567 89ABCDEF 01234567 89ABCDEF 01234567 89ABCDEF 01234567 89ABCDEF
FF00 mmmm-mmm --mmmmmm -------- -------- -mmm-mmm -------- mmmmmmmm ------mm
FF40 -mmm-mmm -mmm-mmm -----mmm -------- -----mmm ----mmmm ----mmmm -mmm-mmm
FF80 mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm
FFC0 mmmmmmmm ----mmmm -------- -------- mmmmmmmm -------- mmmmmmmm --------
>↵
LCD Area
     01234567 89ABCDEF 01234567 89ABCDEF 01234567 89ABCDEF 01234567 89ABCDEF
F000 mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm
F040 mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm
F080 mmmmmmmm mmmmmmmm -------- -------- -------- -------- -------- --------
F0C0 -------- -------- -------- -------- -------- -------- -------- --------
F100 mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm
F140 mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm mmmmmmmm
F180 mmmmmmmm mmmmmmmm -------- -------- -------- -------- -------- --------
F1C0 -------- -------- -------- -------- -------- -------- -------- --------
F200 -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m
F240 -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m -m-m-m-m
F280 -m-m-m-m -m-m-m-m -------- -------- -------- -------- -------- --------
>
```

※ I/O領域、LCD領域の表示は、マップされているアドレスを"m"によって示します。

GUI

なし

8.9.19 モード設定コマンド

md (mode)

機 能

以下のモード設定を行います。

1. オンザフライ情報の表示
   オンザフライ情報の表示間隔を0～5回/秒に設定できます。
   0回を指定すると、表示を行いません。

2. 実行サイクルカウンタの測定モード
   実時間測定モード(usec表示)またはバスサイクルモード(cycle表示)が選択できます。

3. 割り込みモード(ステップ実行)
   シングルステップ実行時に割り込みを許可するかしないか選択できます。

4. シングルステップ表示モード
   シングルステップ動作時に、各ステップの実行結果を表示するか、最終ステップのみを表示するか選択できます。
   レジスタ値は、[Register]ウィンドウが開いているときにはその内容を更新し、閉じているときには[Command]ウィンドウに表示します。
   [Source]ウィンドウが開いている場合は、このモード設定に従って実行する行を矢印で表示します。

5. 実行サイクルカウンタのモード
   ホールドモードまたはリセットモードが選択できます。
   リセットモード時は、プログラム実行コマンドを入力するごとに([Enter]キーによる実行も含む)、カウンタの値がリセットされます。
   実行サイクルカウンタの値はgrコマンドの実行、本モードの変更、カウンタ測定モードの変更、rstコマンドの実行によってもリセットされます。

6. 無効命令チェックモード
   lfコマンドまたはloコマンドでプログラムファイルを読み込む際に、無効命令をチェックするかしないか選択できます。
   プログラムメモリをpeコマンド、pfコマンドで書き換えるときには、このチェックを行いません。

7. cmwコマンドウェイト時間
   cmwコマンドのウェイト時間を1～256秒の範囲内(1秒単位)で設定します。

モードのデフォルト値

モード	デフォルト設定
オンザフライ機能	2回/秒
カウンタ測定モード	バスサイクル
ステップ割り込み	許可しない
ステップ表示	各ステップ
実行サイクルカウンタリセット	ホールド
無効命令チェック	チェックする
cmwウェイト時間	1秒

書 式

(1) >md <option> <num> [... <option> <num>]↲　　(直接入力モード)

(2) >md↲　　(ガイダンスモード)

現在の設定

On the fly interval	0 - 5 times/sec		...? *現在の設定* : <0 ... 5>↲
Counter unit	1. time	2. cycle	...? *現在の設定* : <1 | 2>↲
Interrupt at step	1. allowed	2. not allowed	...? *現在の設定* : <1 | 2>↲
Step display mode	1. each	2. last	...? *現在の設定* : <1 | 2>↲
Counter mode	1. reset	2. hold	...? *現在の設定* : <1 | 2>↲
Illegal instruction	1. check	2. no check	...? *現在の設定* : <1 | 2>↲
Cmw wait time	1 - 256 s		...? *現在の設定* : <1 ... 256>↲

>

<option>:	<num>:	
-f (on the fly interval)	0–5 times/sec	
-u (counter unit)	1. Time	2. Cycle
-i (interrupt at step)	1. Allowed	2. Not allowed
-s (step display mode)	1. Each	2. Last
-c (counter mode)	1. Reset	2. Hold
-il (illegal instruction)	1. Check	2. No check
-cm (cmw wait time)	1–256 sec	

入力例

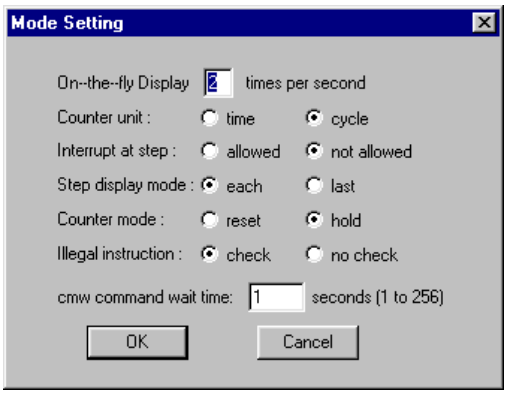
```
>md↲
>md -u 1↲           ...実行サイクルカウンタをタイマ測定モードに設定
>md↲
On the fly interval : 2 times/sec
Counter unit        : time
Interrupt at step   : not allowed
Step display mode   : each
Counter mode        : hold
Illegal instruction : check
Cmw wait time       : 1 s

On the fly interval 0 - 5 times/sec          ...? 2 times/sec : 5↲
Counter unit        1.time     2.cycle       ...? time        : 2↲
Interrupt at step   1.allowed  2.not allowed ...? not allowed : 1↲
Step display mode   1.each     2.last        ...? each        : 2↲
Counter mode        1.reset    2.hold        ...? hold        : ↲
Illegal instruction 1.check    2.no check    ...? check       : ↲
Cmw wait time       1 - 256 s                ...?   1       s : 3↲
>
```

注

- オンザフライの実際の表示間隔は次のようになります。
  (1[sec]/設定回数) + (PC, RS-232C, ICEのオーバーヘッド[sec]) = 表示間隔[sec]
  オーバーヘッドは、PCの性能、RS-232Cのボーレートによりますが、0.05～0.1秒ほどありますので注意してください。
- ガイダンスモードでは、以下のキー操作も有効です。
  "q↲"　...コマンドを終了(入力を終了し、コマンド機能を実行)
  "^↲"　...直前のアドレスに戻る
  "↲"　...入力をスキップ(現在の内容を保持)

GUI

[Option ¦ Mode Setting...]メニューコマンド
このメニューコマンドを選択すると、モードを選択するダイアログボックスが表示されます。



チェックボックスでモードを選択、あるいは間隔設定値を入力して[OK]ボタンをクリックしてください。

8.9.20 FPGA操作コマンド

xfer/xfers (xilinx fpga data erase)

機 能

ICEに挿入されている標準ペリフェラルボード上のFPGAの内容を消去します。

書 式

>xfer↲　メインFPGA用　（直接入力モード）
>xfers↲　サブFPGA用　（直接入力モード）

例

```
>xfer↲
>
```

コマンド入力後、消去実行前に確認のダイアログボックスが表示され、実行と中止が選択できます。

注

- 消去中はプログレスバーにより進捗状況がわかります。そのダイアログボックス上の[Cancel]ボタンか[ESC]キーで消去を中断できます。中断した場合は次のワーニングメッセージが表示されます。

  ```
  Warning : User cancel
  ```

  この場合、再度消去およびデータ書き込みを行うまで標準ペリフェラルボードは使用できません。
- ICEとの通信速度が38400bpsの場合、消去には2分40秒程度（最大）の時間がかかります。
- 通常はサブFPGAにLCD機能の一部が登録されているため、特に指定のない場合は消去しないでください。

GUI

なし

xfwr/xfwrs (xilinx fpga data write)

機 能

ICEに挿入されている標準ペリフェラルボード上のFPGAに周辺回路データを書き込みます。

書 式

>xfwr <file name> ;{H | S} [;N]↲　　メインFPGA用　　（直接入力モード）
>xfwrs <file name> ;{H | S} [;N]↲　　サブFPGA用　　（直接入力モード）

<file name>: FPGAデータファイル（.mot: モトローラS、.mcs: インテルHEX）
H:　　インテルHEXファイルを指定
S:　　モトローラSファイルを指定
N:　　書き込み前の消去を省略

例

```
>xfwr ..¥ice¥fpga¥c63xxx.mot ;S↲
>
```

この例では、FPGAを消去後、c63xxx.mot（モトローラSファイル）のデータをメインFPGAに書き込みます。

```
>xfwr ..¥ice¥fpga¥c63xxx.mot ;S ;N↲
>
```

この例では、データ書き込み前の消去を省略します。ただし、FPGAは事前に消去しておく必要があります。

注

- 書き込むデータファイルはセイコーエプソンが用意したものをそのまま使用してください。ファイル名の拡張子も.mot（モトローラS）、.mcs（インテルHEX）に固定で変更することはできません。不正なファイルを指定すると、エラーとなり書き込みは行えません。

  ```
  Error : Cannot open file
  ```

- Nオプションは、事前にxfer/xfersコマンドでFPGAを完全に消去している場合にのみ指定できます。消去していないFPGAにデータを書き込む場合、Nオプションを指定しないでください。
- 実行中はプログレスバーにより進捗状況がわかります。そのダイアログボックス上の[Cancel]ボタンか[ESC]キーで実行を中断できます。中断した場合は次のワーニングメッセージが表示されます。

  ```
  Warning : User cancel
  ```

  この場合、再度消去およびデータ書き込みを行うまで標準ペリフェラルボードは使用できません。
- ICEとの通信速度が38400bpsの場合、データ書き込みには消去も含め8分程度（最大）の時間がかかります。
- 通常はサブFPGAにLCD機能の一部が登録されているため、特に指定のない場合はデータを書き込まないでください。

GUI

なし

xfcp/xfcps (xilinx fpga data compare)

機 能

標準ペリフェラルボード上のFPGAの内容と指定ファイルの内容を比較します。

書 式

>xfcp <file name> ;{H | S}↲　メインFPGA用　（直接入力モード）
>xfcps <file name> ;{H | S}↲　サブFPGA用　（直接入力モード）

<file name>: FPGAデータファイル（.mot: モトローラS、.mcs: インテルHEX）
H:　インテルHEXファイルを指定
S:　モトローラSファイルを指定

例

```
>xfcp ..¥ice¥fpga¥c63xxx.mot ;S↲
>                                   ...エラーがなかった場合

>xfcp ..¥ice¥fpga¥c63yyy.mot ;S↲
Warning : Verify error              ...データの不一致が見つかった場合
0X00000  0XFF                       ...FPGA内のエラーアドレスとデータを表示
0X00001  0X84
0X00002  0XAB
   :      :
>
```

注

- 実際に比較されるのは、指定ファイル内でデータが存在するアドレス範囲のみです。その範囲外にあるFPGA内のデータは比較されません。
- 比較するデータファイルはセイコーエプソンが用意したものをそのまま使用してください。ファイル名の拡張子も.mot（モトローラS）、.mcs（インテルHEX）に固定で変更することはできません。不正なファイルを指定すると、エラーとなります。

```
Error : Cannot open file
```

- 実行中はプログレスバーにより進捗状況がわかります。そのダイアログボックス上の[Cancel]ボタンか[ESC]キーで実行を中断できます。中断した場合は次のワーニングメッセージが表示されます。

```
Warning : User cancel
```

GUI

なし

xdp/xdps (xilinx fpga data dump)

機 能

標準ペリフェラルボード上のFPGAの内容を、16バイト/行の16進ダンプ形式で[Command]ウィンドウに表示します。

書 式

>xdp <address1> [<address2>]↵ メインFPGA用 （直接入力モード）
>xdps <address1> [<address2>]↵ サブFPGA用 （直接入力モード）

<address1>: 表示開始アドレス 16進数
<address2>: 表示終了アドレス 16進数
条件: 0≦address1≦address2≦FPGAエンドアドレス

例

<address1>のみを指定した場合、<address1>から256バイトのデータを表示します。

```
>xdp 0↵
Addr   +0 +1 +2 +3 +4 +5 +6 +7  +8 +9 +A +B +C +D +E +F
00000: FF 84 AB EF F9 D8 FF BB  FB BB BF FB BF BF FB BF
00010: BB FB BB BF BB BF FB BB  BF BF FB BB FF EE FF EE
00020: EF FE D7 FB FE EE EF EF  EE EE FE EE FB FE EF EF
  :               :                           :
000E0: FF FF FF FF FB FF FF FF  BD DF FB FD DF FF FF FF
000F0: FF FF BF FF FF FF FF F9  FF FF FF FF FF FF FF FF
>
```

<address1>と<address2>を指定した場合、その範囲のデータを表示します。

```
>xdp 100 100↵
Addr   +0 +1 +2 +3 +4 +5 +6 +7  +8 +9 +A +B +C +D +E +F
00100: FF
>
```

注

- アドレスが16進数以外の場合、エラーとなります。

```
Error : invalid value
```

- 先頭アドレスが終了アドレスより大きい場合、エラーとなります。

```
Error : end address < start address
```

GUI

なし

8.9.21 終了コマンド

q (quit)

機 能

デバッガを終了します

書 式

>q↲ （直接入力モード）

GUI

[File ¦ Exit]メニューコマンド
このメニューコマンドの選択によっても、デバッガを終了できます。

8.9.22 ヘルプコマンド

? (help)

機 能

各コマンドの入力書式を表示します。

書 式

(1) ? （直接入力モード）
(2) ? <n> （直接入力モード）
(3) ? <command> （直接入力モード）

<n>: コマンドグループ番号 10進数
<command>: コマンド名
条件: 1≦n≦6

例

書式1、書式2のコマンド入力によって機能別のコマンド一覧を表示します。
個別のコマンドの入力書式は書式3のコマンド入力によって表示させます。

```
>?↲
group 1: program, data & register . pe,pf,pm,a(as)/dd,de,df,dm,dw/od/rd,rs
group 2: execution & break ........ g,gr,s,n,rst/bp,bc(bpc),bd,bdc,br,brc,bs,bsc,bsp,bl,bac
group 3: source & symbol .......... u,sc,m/sy
group 4: file & flash rom ......... lf,lo/lfl,sfl,efl/xfer,xfers,xfwr,xfwrs,xfcp,xfcps,xdp,xdps
group 5: trace & coverage ......... tm,td,ts,tf/cv,cvc
group 6: others ................... com,cmw,rec,log/ma,md,q,?
 Type "? <group #>" to show group or type "? <command>" to get usage of the command
>? 1↲
group 1: program, data & register
pe (program enter), pf (program fill), pm (program move), a/as (inline assemble),
dd (data dump), de (data enter), df (data fill), dm (data move), dw (data watch),
od (option dump),
rd (register display), rs (register set)
 Type "? <command>" to get usage of the command
>? pe↲
pe (program enter): change program memory
usage: pe [address]                    ... change program with guidance
       pe address code1 [... code8]    ... change program with specified code
>
```

GUI

[Help ¦ Contents...]メニューコマンド、[Help]ボタン
このコマンドおよびボタンを選択すると、ヘルプトピックを表示する[Help]ウィンドウが開きます。

 [Help]ボタン

8.10 ステータス/エラー/ワーニングメッセージ

1. ICEエラー

エラーメッセージ	メッセージ内容
Break by PC break	PCブレークポイントによりブレーク
Break by data break	データブレーク条件によりブレーク
Break by register break	レジスタブレーク条件によりブレーク
Break by sequential break	シーケンシャルブレーク条件によりブレーク
Key Break	[Key Break]ボタンによりブレーク
Break by accessing no map program area	未定義プログラムメモリ領域のアクセスによりブレーク
Break by accessing no map data area	未定義データメモリ領域のアクセスによりブレーク
Break by accessing ROM area	データROM領域への書き込みによりブレーク
Out of SP1 area	SP1スタック領域外に対するスタック操作によりブレーク
Out of SP2 area	SP2スタック領域外に対するスタック操作によりブレーク
Break by external break	ICEのBRKIN端子への信号入力によりブレーク

2. ICEステータス

ステータスメッセージ	メッセージ内容
communication error	タイムアウト以外の通信エラー （オーバーラン、フレーミング、BCCエラー）
CPU is running	CPUが実行中
ICE is busy	ICEが処理中
ICE is free run mode	ICEがフリーランモード
ICE is maintenance mode	ICEが保守モード
no map area, XXXX	マップ領域外をアクセス
not defined ID, XXXX	ICEの応答IDが無効
on tracing	実行データのトレース中
reset time out	CPUがリセットできない（1秒以上）
target down	ペリフェラルボードが正常に動作していない、もしくはリセットが解除されない
Time Out!	通信タイムアウト

3. フラッシュメモリエラー

エラーメッセージ	メッセージ内容
flash memory error, XXXX	フラッシュメモリの書き込みまたは消去エラー
flash ROM is protected	フラッシュメモリにプロテクトがかかっている
format error	フラッシュメモリがマップされていない
Map information is not the same	パラメータファイルからのマップ情報とフラッシュメモリのマップ情報に違いがある
verify error, XXXX	フラッシュメモリ書き込み時のベリファイエラー

4. コマンドエラー/ワーニング

エラーメッセージ	メッセージ内容（エラーが発生するコマンド）
Address out of range, use 0–0xXXXX	指定アドレスがプログラムメモリの有効範囲外 （a/as, pe, pf, pm, sc, m, u, g, gr, bp, bc, bs, tm, ts, cv）
Address out of range, use 0–0xFFFF	指定アドレスがデータメモリの有効範囲外 （dd, de, df, dm, dw, bd, ts）
Cannot load program/ROM data, check ABS file	プログラム/ROMデータのロードに失敗 IEEE-695実行形式以外のファイルが指定された（lf）
Cannot open file	ファイルがオープンできない （lf, lo, com, cmw, log, rec, xfwr/xfwrs, xfcp/xfcps）
Data out of range, use 0–0xF	指定の数値はデータの有効範囲外（de, df）
Different chip type, cannot load this file	指定ファイルは異なるICEパラメータで作成されている（lf）
end address < start address	開始アドレスより小さい終了アドレスが指定された （pf, pm, df, dm, bd, cv, xdp/xdps）
error file type (extension should be CMD)	".cmd"以外の拡張子を持つコマンドファイルが指定された（com, cmw）
FO address out of range, use 0–0xEF	FOアドレスが有効範囲外（od）
illegal code	入力コードが不正（pe, pf）
illegal mnemonic	S1C63000用の入力ニーモニックが不正（a/as）
Incorrect number of parameters	パラメータ数が不正（全コマンド）
Incorrect option, use -f/-u/-i/-s/-c/-il/-cm	無効なモード設定オプションが指定された（md）
Incorrect r/w option, use r/w/*	不正なR/Wオプションが指定された（bd）
Incorrect register name, use A/B/X/Y/F	無効なレジスタ名が指定された（br）

エラーメッセージ	メッセージ内容(エラーが発生するコマンド)
Incorrect register name, use PC/A/B/X/Y/F/SP1/SP2/EXT/Q	指定されたレジスタ名が無効(rs)
Input address does not exist	未設定ブレークアドレスを解除しようとした(bp)
invalid command	無効なコマンドが入力された(全コマンド)
invalid data pattern	入力データパターンが不正(bd, br)
invalid file name	オプションファイルの拡張子が無効(lo)
invalid value	入力データ、アドレス、シンボルが不正(全コマンド)
Maximum nesting level(5) is exceeded, cannot open file	com/cmwコマンドのネストレベルが制限を越えた(com, cmw)
MLA address out of range, use 0–0xFFF	MLAアドレスが有効範囲外(od)
no such symbol	該当するシンボルがない(シンボル対応の全コマンド)
no symbol information	".ABS"ファイルがロードされていないため、シンボルは使用不可(sy)
Number of passes out of range, use 0–4095	シーケンシャルブレークの実行回数指定が有効範囲外(bs)
Number of steps out of range, use 0–65535	指定ステップ数が有効範囲外(s, n)
SO address out of range, use 0–0x1FFF	SOアドレスが有効範囲外(od)
SP1 address out of range, use 0–0x3FF	指定SP1アドレスが有効範囲外(bsp)
SP2 address out of range, use 0–0xFF	指定SP2アドレスが有効範囲外(bsp)
symbol type error	指定シンボルタイプ(プログラム/データ)が不正 (シンボル対応の全コマンド)

ワーニングメッセージ	メッセージ内容(エラーが発生するコマンド)
Break address already exists	設定済みのブレークアドレスを再設定(bp)
Identical break address input	ブレークアドレスの二重設定
round down to multiple of 4	監視データアドレスが不正(dw)
User cancel	コマンドをユーザが中断(xfer/xfers, xfwr/xfwrs, xfcp/xfcps)
Verify error	FPGAのコンペアエラー(xfcp/xfcps)

8.11 注意事項

最新バージョンの制限事項やサポート機能、バグ情報についてはS5U1C63000Aのrel_debug_J.txtを参照してください。
最新バージョンのサポート機種はS5U1C63000AのReadme_J.txtを参照してください。

9 ファンクションオプションジェネレータ

9.1 ファンクションオプションジェネレータwinfogの概要

S1C63チップは、I/Oポート機能など、いくつかのハードウェア仕様をマスクオプションとして選択できるようになっています。これにより、開発する製品の仕様に合わせてS1C63チップのマスクパターンを変更し、ハードウェアを構成することができます。
ファンクションオプションジェネレータwinfogは、マスクパターン生成のためのファイルを作成するソフトウェアツールで、マスクオプションがGUIにより容易に選択できます。このwinfogで作成されたファイルから、セイコーエプソンはS1C63チップのマスクパターンを生成します。
また、ICEを用いてデバッグを行う際に必要なマスクオプション設定用ファイル(モトローラS2フォーマットデータ)も同時に作成できます。ICEでデバッグする場合、このファイルをホストマシンからダウンロードすることによって、実ICと同等のオプション機能がICE上で実現できます。

9.2 入出力ファイル

図9.2.1にwinfogの入出力ファイルを示します。

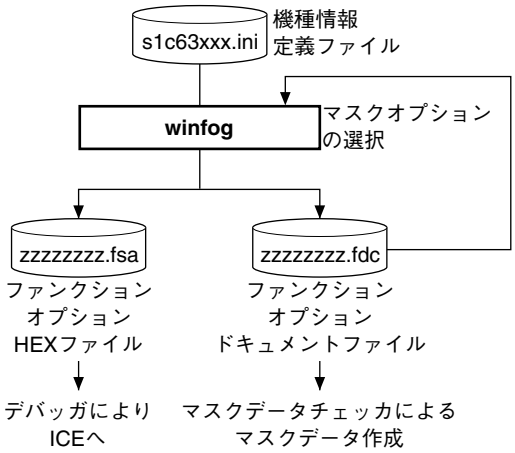


図9.2.1 winfogの入出力ファイル

●機種情報定義ファイル(s1c63xxx.ini)
各機種のオプションリストやその他の情報が記録されています。必ずセイコーエプソンが提供するファイルを使用してください。このファイルはファイル名で示される機種にのみ有効です。ファイルの内容を修正したり、他の機種で使用しないでください。

●ファンクションオプションドキュメントファイル(zzzzzzzz.fdc)
マスクオプションの選択内容が記録されるテキスト形式のファイルです。このファイルをwinfogに読み込ませて選択済みのオプション設定を修正することもできます。このファイルは完成した他のプログラム/データファイルと共に、マスクデータチェッカwinmdcによって1つのファイルにパックし、マスクデータファイルとしてセイコーエプソンに提出していただきます。セイコーエプソンは、そのマスクデータファイルからICのマスクパターンを作成します。

●ファンクションオプションHEXファイル(zzzzzzzz.fsa)
ICEに選択したマスクオプションを設定するための、モトローラS2フォーマットのファイルです。ICEでデバッグを行う場合、デバッガdb63のコマンドによって、このファイルをICE上にダウンロードします。

*1 ファイル名の"xxx"は機種名です。"zzzzzzzz"の部分には任意の名前を付けてください。

*2 ICEへのマスクオプションのダウンロード方法については、"8 デバッガ"を参照してください。

9.3　操作方法

9.3.1 起動方法

エクスプローラからの起動

winfog.exeアイコンをダブルクリックするか、スタートメニューからwinfogを選択してください。
前回の実行時に機種情報定義ファイル(s1c63xxx.ini)を読み込んでいる場合は、winfog起動時に同じファイルを自動的に読み込みます。
また、機種情報定義ファイルのアイコンをwinfog.exeアイコンにドラッグすることによってもwinfogが起動し、その機種情報定義ファイルを読み込みます。

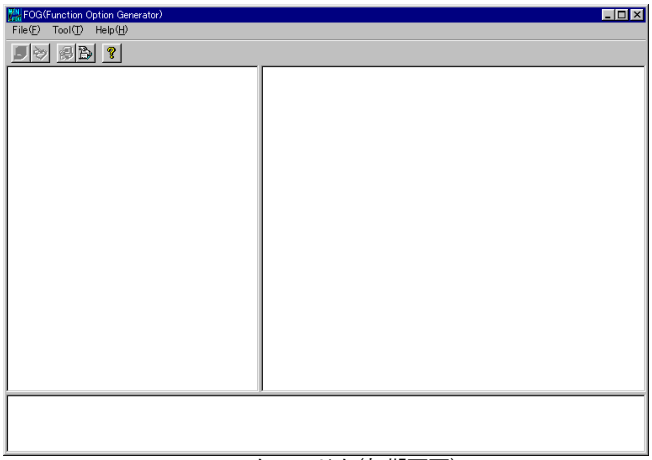
コマンド入力による起動

winfogはコマンドプロンプトからも次のコマンドで起動可能です。

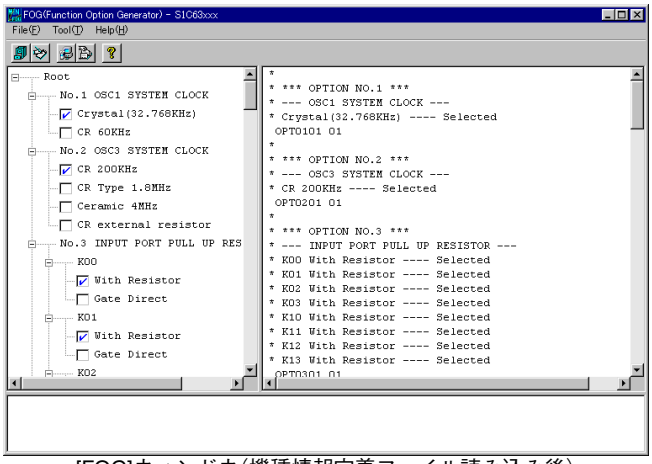
>winfog [s1c63xxx.ini]↵

↵はリターンキーの入力を表します。
コマンドオプションとして機種情報定義ファイル(s1c63xxx.ini)が指定できます(パスも指定可能)。ここで指定すると、winfog起動時に機種情報定義ファイルが読み込まれます。この指定は省略可能です。

起動すると[FOG]ウィンドウを表示します。以下に機種情報定義ファイルを読み込まなかった場合と読み込んだ場合のウィンドウの表示例を示します。

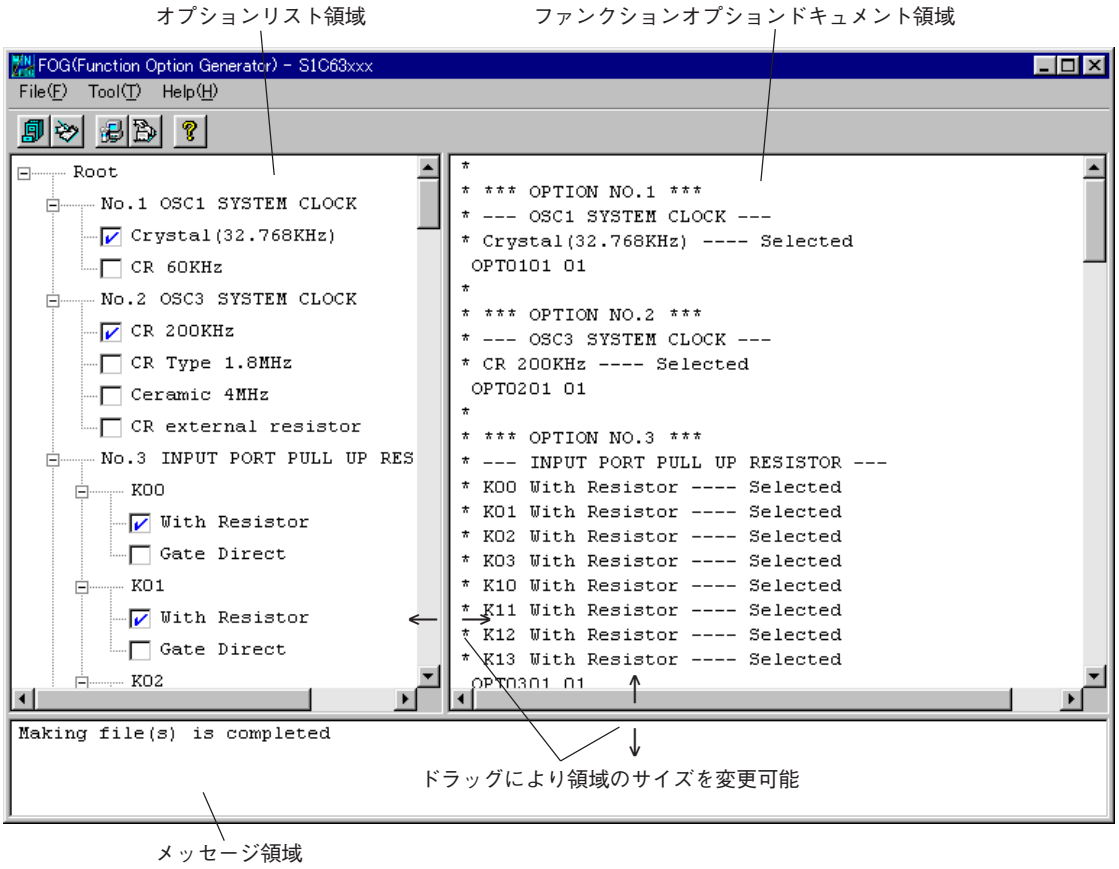


[FOG]ウィンドウ(初期画面)



[FOG]ウィンドウ(機種情報定義ファイル読み込み後)

9.3.2 ウィンドウ



* タイトルバーの機種名は、読み込んだ機種情報定義ファイルのファイル名(パスと拡張子を除く)です。
* オプションリストとファンクションオプションドキュメントの内容は機種により異なります。

図9.3.2.1　ウィンドウの構成

[FOG]ウィンドウは図に示すとおり、3つの領域に分割されています。

オプションリスト領域

機種情報定義ファイル(s1c63xxx.ini)で設定される、マスクオプションの一覧です。チェックボックスを使用して、各オプションを選択します。チェックマーク(✓)は現在選択されているオプションを示します。

ファンクションオプションドキュメント領域

オプションの選択内容がファンクションオプションドキュメントの形式で表示されます。ファンクションオプションドキュメントファイルには、この領域の表示内容が出力されます。オプションリスト領域で選択項目を変更すると、表示が即時更新されます。

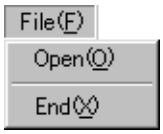
メッセージ領域

[Tool]メニューから[Generate]を選択、あるいは[Generate]ボタンをクリックしてファイルを作成した際に、その結果を示すメッセージを表示します。

9.3.3 メニューとツールバーボタン

以下、各メニュー項目と、ツールバーボタンについて説明します。

[File]メニュー

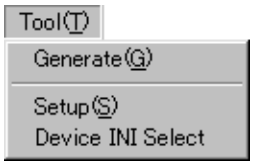


Open
ファンクションオプションドキュメントファイルを開きます。既存のファイルを修正する場合などに使用します。[Open]ボタンも同機能です。


[Open]ボタン

End
winfogを終了します。

[Tool]メニュー



Generate
オプションリストの選択内容でファイルを作成します。[Generate]ボタンも同機能です。


[Generate]ボタン

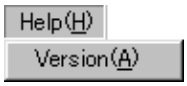
Setup
作成日や出力ファイル名、ファンクションオプションドキュメントファイルに含めるコメントなどを設定します。[Setup]ボタンも同機能です。


[Setup]ボタン

Device INI Select
機種情報定義ファイル(s1c63xxx.ini)をロードします。[Device INI Select]ボタンも同機能です。このファイルのロードは最初に行っておく必要があります。


[Device INI Select]ボタン

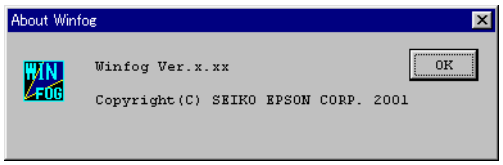
[Help]メニュー



Version
winfogのバージョンを表示します。[Help]ボタンも同機能です。


[Help]ボタン

次のダイアログボックスが表示されます。閉じるには[OK]をクリックしてください。



9.3.4 操作手順

基本的な操作手順を以下に示します。

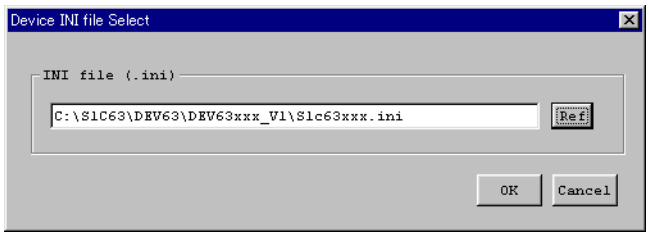
(1) 機種情報定義ファイルのロード

最初に機種情報定義ファイル(s1c63xxx.ini)を選択してロードします。

[Tool]メニューから[Device INI Select]を選択するか、[Device INI Select]ボタンをクリックします。


[Device INI Select]ボタン

次のダイアログが表示されますので、テキストボックスにパスを含むファイル名を入力するか、[Ref]ボタンをクリックしてファイルの選択を行ってください。



[OK]をクリックすると、ファイルをロードします。指定したファイルが存在し、内容に問題がなければ、デフォルト設定のオプションリストとファンクションオプションドキュメントがそれぞれの領域に表示されます。
ファイルのロードを中止するには[Cancel]をクリックします。

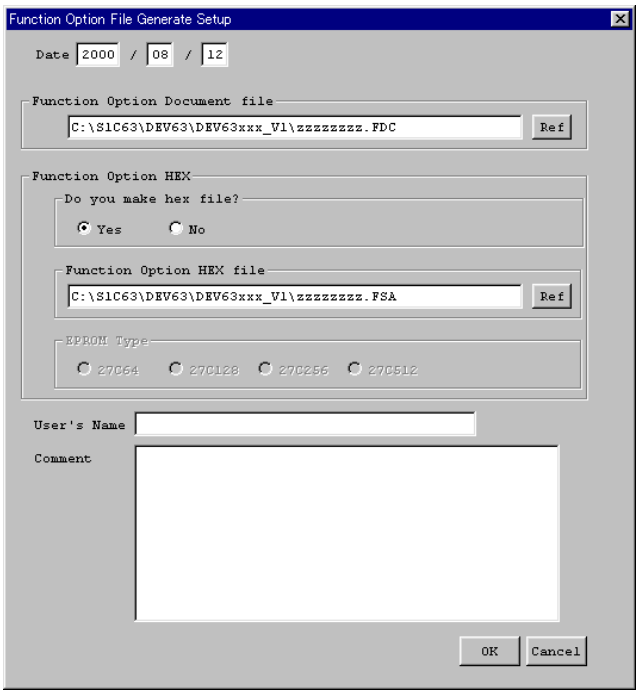
一度、機種情報定義ファイルを選択すると、次回の起動時は同じファイルが自動的にロードされます。

注: オプションをすでに設定している状態で機種情報定義ファイルをロードすると、設定がすべてデフォルトの状態に戻ります。

(2) セットアップ

[Tool]メニューから[Setup]を選択するか、[Setup]ボタンをクリックして[Setup]ダイアログボックスを表示させ、必要な選択と入力を行います。


[Setup]ボタン



Date
現在の日付が表示されます。必要に応じて変更してください。

Function Option Document file
作成するファンクションオプションドキュメントファイル名を、ここで指定します。デフォルトで表示される名前を修正して使用してください。[Ref]ボタンで他のフォルダも参照できます。

Function Option HEX
Do you make hex file?
ファンクションオプションHEXファイルを作成するか選択します。
ICEを使用したデバッグを行う場合は作成してください。

Function Option HEX file
ファンクションオプションHEXファイルを作成する場合に、そのファイル名をここで指定します。デフォルトで表示される名前を修正して使用してください。[Ref]ボタンで他のフォルダも参照できます。

EPROM Type
これはS1C63 Familyでは選択できません。

User's Name
お客様の会社名を入力します。最大40文字まで入力することができます。英文字、数字、記号およびスペースが入力可能です。
ここに入力した内容は、ファンクションオプションドキュメントファイルのUSER'S NAMEフィールドに記録されます。

Comment
コメントを入力します。1行に入力可能な文字数は50文字まで、最大10行まで入力することができます。英文字、数字、記号およびスペースが入力可能です。また、[Enter]キーで改行できます。
なお、コメントには、次のような内容を含めるようにお願いします。

・事業所、所属
・所在地、電話番号、FAX番号
・その他、技術情報など

ここに入力した内容は、ファンクションオプションドキュメントファイルのCOMMENTフィールドに記録されます。

上記の必要な項目を入力後、[OK]をクリックすると設定内容が保存され、ダイアログボックスが閉じます。設定内容は即時有効となります。
[Cancel]をクリックした場合、現在の設定は変更されずにダイアログボックスが閉じます。

注:• ファイル名の指定には以下の制限があります。
1. パスを含めたファイル名指定の文字数は最大2048文字です。
2. ファイル名(拡張子を除く)は最大15文字、拡張子は最大3文字です。
3. ファイル名の先頭にハイフン(-)は使用できません。また、ディレクトリ名(フォルダ名)、ファイル名、拡張子に、以下の記号の使用を禁止します。
/ : , ; * ? " < > ¦

• User's NameとCommentに以下の記号は使用できません。
$ ¥ ¦ `

(3) オプションの選択

オプションリストのチェックボックスをクリックして必要なオプションを選択します。オプションリスト領域で選択項目を変更すると、ファンクションオプションドキュメント領域の表示が更新されます。
なお、オプションリストは、機種情報定義ファイルをロードした時点でデフォルトの選択状態になります。
オプション仕様については、各機種のテクニカルマニュアルを参照してください。

(4) ファイルの作成

オプションの選択が終了後、[Tool]メニューから[Generate]を選択するか、[Generate]ボタンをクリックしてファイルを作成します。


[Generate]ボタン

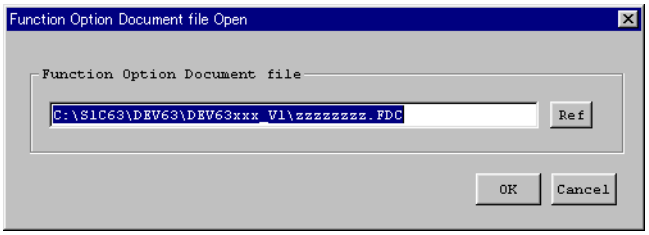
[Setup]ダイアログボックスで指定したファンクションオプションドキュメントファイルとファンクションオプションHEXファイル(指定時のみ)が作成されます。
ファイル作成が正常に終了した場合は、"Making file(s) is completed"がメッセージ領域に表示されます。
エラーが発生した場合は、エラーメッセージが表示されます。

(5)既存ドキュメントファイルの修正

既存のファンクションオプションドキュメントファイルを読み込んで、必要個所を修正することもできます。
ファイルを読み込むには、[File]メニューから[Open]を選択するか、[Open]ボタンをクリックします。

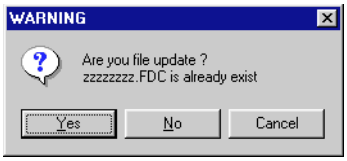

[Open]ボタン

次のダイアログが表示されますので、テキストボックスにパスを含むファイル名を入力するか、[Ref]ボタンをクリックしてファイルの選択を行ってください。



[OK]をクリックすると、ファイルをロードします。指定したファイルが存在し、内容に問題がなければ、オプションリストとファンクションオプションドキュメント領域がファイルの内容に更新されます。
ファイルのロードを中止するには[Cancel]をクリックします。

(2)～(4)の作業を行い、ファイルを更新してください。
ファイル名を変更せずに[Generate]を選択すると、上書きを確認する次のメッセージが表示され、[Yes]をクリックして書き込み、[No]または[Cancel]をクリックして中止できます。ファイル名の変更は[Setup]ダイアログボックスで行ってください。



注: ファンクションオプションドキュメントファイルの読み込みは、機種情報定義ファイルがロードされている場合にのみ行えます。

(6)終了

winfogを終了するには、[File]メニューから[End]を選択してください。

9.4　エラーメッセージ

winfogのエラーメッセージの一覧を示します。表示の"Dialog"はダイアログボックスに表示されるメッセージを、"Message"は[FOG]ウィンドウのメッセージ領域に表示されるメッセージを示します。

表9.4.1　winfogエラーメッセージ一覧

メッセージ	説　　明	表示
File name error	ファイル名または拡張子名の文字数が使用可能範囲を超えている。	Dialog
Illegal character	入力禁止文字が入力された。	Dialog
Please input file name	ファイル名が未入力。	Dialog
Can't open File : xxxx	ファイル(xxxx)がオープンできない。	Dialog
INI file is not found	指定した機種情報定義ファイル(.ini)が存在しない。	Dialog
INI file does not include FOG information	指定した機種情報定義ファイル(.ini)にファンクションオプション情報が含まれていない。	Dialog
Function Option document file is not found	指定したファンクションオプションドキュメントファイルが存在しない。	Dialog
Function Option document file does not match INI file	指定したファンクションオプションドキュメントファイルの内容が機種情報定義ファイル(.ini)と異なる。	Dialog
A lot of parameter	コマンドラインの引数が多すぎる。	Dialog
Making file(s) is completed [xxxx is no data exist]	ファイル作成完了。ただし、作成したファイル(xxxx)にはデータが含まれていない。	Message
Can't open File: xxxx Making file(s) is not completed	Generate実行時、ファイル(xxxx)がオープンできない。	Message
Can't write File: xxxx Making file(s) is not completed	Generate実行時、ファイル(xxxx)に書き込みができない。	Message

表9.4.2　winfogワーニングメッセージ

メッセージ	説　　明	表示
Are you file update? xxxx is already exist	上書き確認メッセージ (指定したファイルは既に存在する。)	Dialog

9.5　注意事項

(1) winfogは日本語(2バイト文字)に対応していませんので、パスやファイル名に使用しないでください。

最新バージョンの制限事項やサポート機能、バグ情報についてはS5U1C63000Aのrel_windev_J.txtを参照してください。
最新バージョンのサポート機種はS5U1C63000AのReadme_J.txtを参照してください。

9.6 出力ファイル例

注: オプションの構成等は、機種により異なります。

●ファンクションオプションドキュメントファイル例

```
* S1C63xxx FUNCTION OPTION DOCUMENT Vx.xx     ←バージョン
*
* FILE NAME    zzzzzzzz.FDC                  ←ファイル名([Setup]で指定)
* USER'S NAME  SEIKO EPSON CORPORATION       ←ユーザ名([Setup]で指定)
* INPUT DATE   yyyy/mm/dd                    ←作成年月日([Setup]で指定)
* COMMENT      SAMPLE DATA                   ←コメント([Setup]で指定)
*
* *** OPTION NO.1 ***                        ←オプション番号
* --- OSC1 SYSTEM CLOCK ---                  ←オプション名
* Crystal(32.768KHz) ---- Selected           ←選択した仕様
 OPT0101 01                                  ←マスクデータ
*
* *** OPTION NO.2 ***
* --- OSC3 SYSTEM CLOCK ---
* CR 200KHz ---- Selected
 OPT0201 01
*
* *** OPTION NO.3 ***
* --- INPUT PORT PULL UP RESISTOR ---
* K00 With Resistor ---- Selected
* K01 With Resistor ---- Selected
* K02 With Resistor ---- Selected
* K03 With Resistor ---- Selected
* K10 With Resistor ---- Selected
* K11 With Resistor ---- Selected
* K12 With Resistor ---- Selected
* K13 With Resistor ---- Selected
 OPT0301 01
 OPT0302 01
 OPT0303 01
 OPT0304 01
 OPT0305 01
 OPT0306 01
 OPT0307 01
 OPT0308 01
*
* *** OPTION NO.4 ***
* --- OUTPUT PORT OUTPUT SPECIFICATION ---
* R00 Complementary ---- Selected
* R01 Complementary ---- Selected
* R02 Complementary ---- Selected
* R03 Complementary ---- Selected
 OPT0401 01
 OPT0402 01
 OPT0403 01
 OPT0404 01
*
               :
*
* *** OPTION NO.8 ***
* --- SOUND GENERATOR POLARITY ---
* NEGATIVE ---- Selected
 OPT0801 01
*EOF                                         ←エンドマーク
```

●ファンクションオプションHEXファイル例(モトローラS2フォーマット)

```
S22400000022FF0200FFFFFFFFFFFFFFFFFFFFFFFF00000000000000FFFFFFFFFFFFFFFFFFCD
S804000000FB
```

モトローラS2フォーマットについては"6.5.2 モトローラSファイル"を参照してください。

10 セグメントオプションジェネレータ

10.1 セグメントオプションジェネレータwinsogの概要

S1C63 Familyの一部の機種はLCD出力端子の出力仕様、表示メモリとLCD出力端子の割り付けをハードウェアオプションで設定できるようになっており、オプション設定に従ってICのマスクパターンが作成されます。セグメントオプションジェネレータwinsogは、マスクパターン生成のためのファイルを作成するソフトウェアツールで、セグメントオプションがGUIにより容易に設定できます。
また、ICEを用いてデバッグを行う際に必要なマスクオプション設定用ファイル（モトローラS2フォーマットデータ）も同時に作成できます。ICEでデバッグする場合、このファイルをホストマシンからダウンロードすることによって、実ICと同等のオプション機能がICE上で実現できます。

注: セグメントオプションジェネレータwinsogは、セグメントオプションが設定されている機種の開発にのみ使用します。

10.2 入出力ファイル

図10.2.1にwinsogの入出力ファイルを示します。

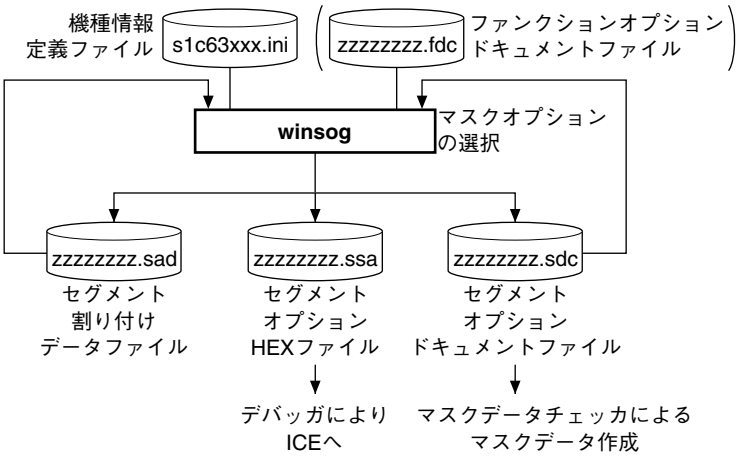


図10.2.1　winsogの入出力ファイル

●機種情報定義ファイル（s1c63xxx.ini）
各機種のオプションリストやその他の情報が記録されています。必ずセイコーエプソンが提供するファイルを使用してください。このファイルはファイル名で示される機種にのみ有効です。ファイルの内容を修正したり、他の機種で使用しないでください。

●ファンクションオプションドキュメントファイル（zzzzzzzz.fdc）
ファンクションオプションジェネレータwinfogで作成した、マスクオプションの選択内容が記録されているテキスト形式のファイルです。このファイルは、セグメントオプションの設定条件がwinfogのマスクオプションの選択によって決定する機種にのみ必要となります。

●セグメントオプションドキュメントファイル（zzzzzzzz.sdc）
セグメントオプションの設定内容が記録されるテキスト形式のファイルです。このファイルをwinsogに読み込ませてオプション設定を修正することもできます。このファイルは完成した他のプログラム/データファイルと共に、マスクデータチェッカwinmdcによって1つのファイルにパックし、マスクデータファイルとしてセイコーエプソンに提出していただきます。セイコーエプソンは、そのマスクデータファイルからICのマスクパターンを作成します。

●セグメントオプションHEXファイル（zzzzzzzz.ssa）
ICEに選択したセグメントオプションを設定するための、モトローラS2フォーマットのファイルです。ICEでデバッグを行う場合、デバッガdb63のコマンドによって、このファイルをICE上にダウンロードします。

●セグメント割り付けデータファイル（zzzzzzzz.sad）
割り付け途中のセグメントオプションを記録しておくためのテキスト形式のファイルです。作業途中でwinsogを終了する場合などにこのファイルを作成しておきます。次回はこのファイルをwinsogに読み込ませることで続きのオプション設定が行えます。

*1 ファイル名の"xxx"は機種名です。"zzzzzzzz"の部分には任意の名前を付けてください。

*2 ICEへのマスクオプションのダウンロード方法については、"8 デバッガ"を参照してください。

10.3 操作方法

10.3.1 起動方法

エクスプローラからの起動

winsog.exe

winsog.exeアイコンをダブルクリックするか、スタートメニューからwinsogを選択してください。
前回の実行時に機種情報定義ファイル（s1c63xxx.ini）を読み込んでいる場合は、winsog起動時に同じファイルを自動的に読み込みます。
また、機種情報定義ファイルのアイコンをwinsog.exeアイコンにドラッグすることによってもwinsogが起動し、その機種情報定義ファイルを読み込みます。
ファンクションオプションドキュメントファイルが必要な機種では、そのファイル名を入力するダイアログボックスが表示されますので、テキストボックスにパスも含め入力してください。あるいは、[Ref]ボタンをクリックしてファイルを選択してください。

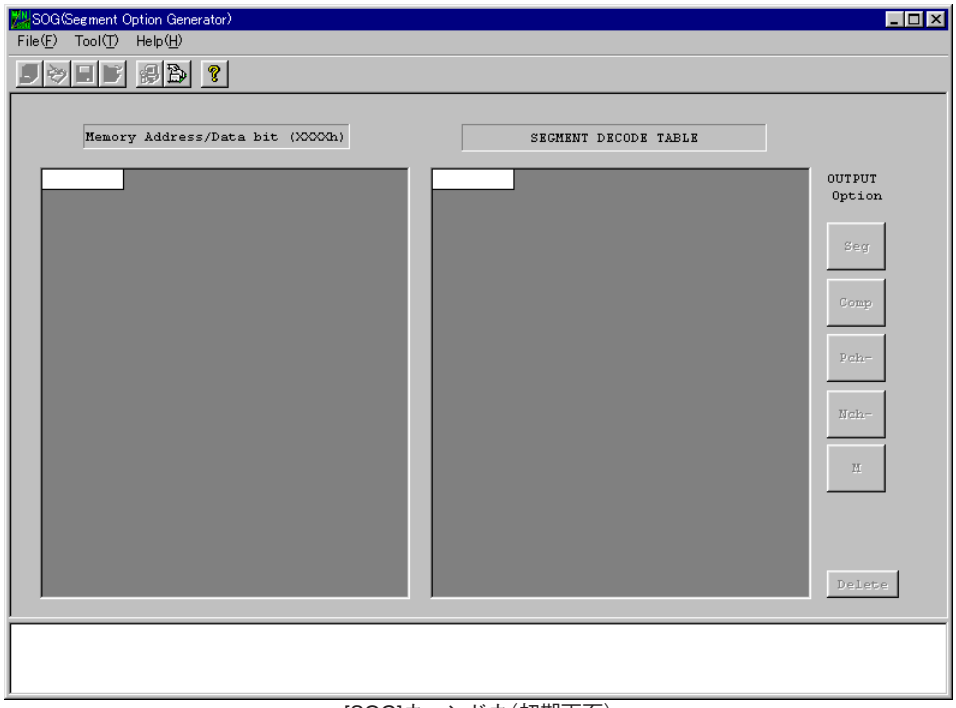
コマンド入力による起動

winsogはコマンドプロンプトからも次のコマンドで起動可能です。

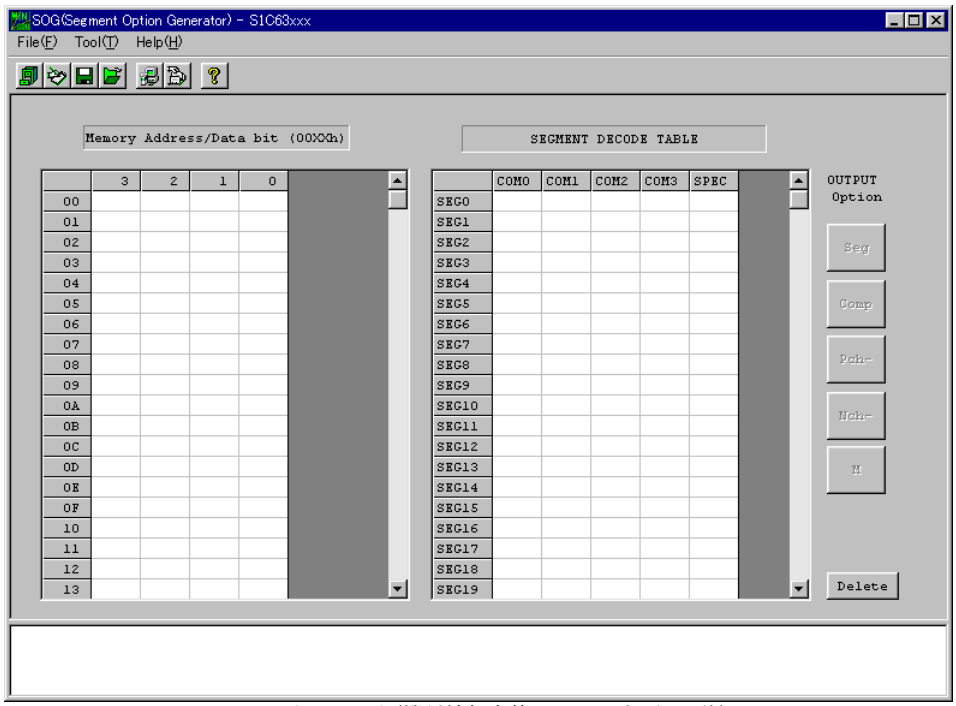
>winsog [s1c63xxx.ini]↵

↵はリターンキーの入力を表します。
コマンドオプションとして機種情報定義ファイル（s1c63xxx.ini）が指定できます（パスも指定可能）。ここで指定すると、winsog起動時に機種情報定義ファイルが読み込まれます。ファンクションオプションドキュメントファイルが必要な機種の場合は、そのファイルがs1c63xxx.iniおよびwinsog.exeと同じフォルダに用意されている状態でコマンドを入力してください。コマンド実行後、ファンクションオプションドキュメントファイル名を入力するダイアログボックスが表示されますので、テキストボックスにパスも含め入力するか、[Ref]ボタンをクリックしてファイルを選択してください。
機種情報定義ファイルの指定は省略可能です。

起動すると[SOG]ウィンドウを表示します。以下に機種情報定義ファイルを読み込まなかった場合と読み込んだ場合のウィンドウの表示例を示します。

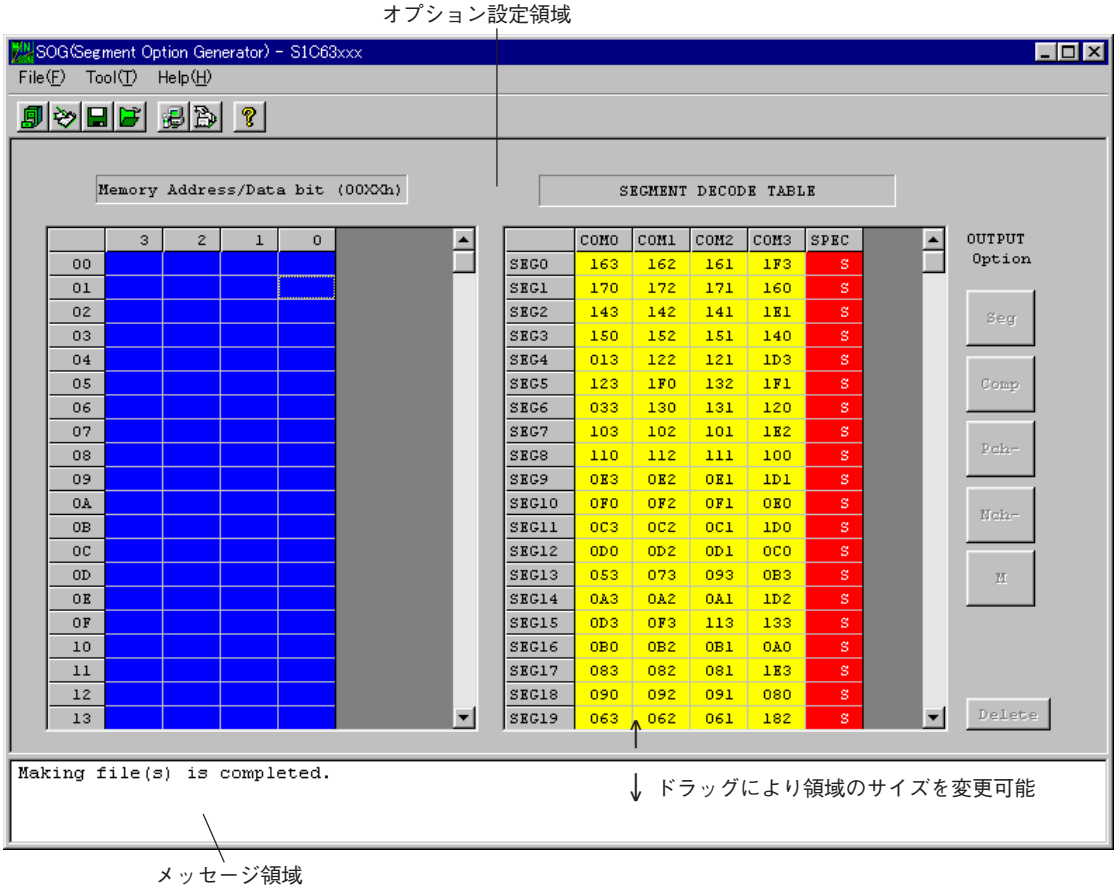


[SOG]ウィンドウ(初期画面)



[SOG]ウィンドウ(機種情報定義ファイル読み込み後)

10.3.2 ウィンドウ



* タイトルバーの機種名は、読み込んだ機種情報定義ファイルのファイル名(パスと拡張子を除く)です。
* 表示メモリアドレスとセグメントの構成は機種により異なります。

図10.3.2.1　ウィンドウの構成

[SOG]ウィンドウは図に示すとおり、2つの領域に分割されています。

オプション設定領域

表示メモリマップとセグメントデコードテーブル、端子の仕様を選択するボタンで構成されています。表示メモリマップとセグメントデコードテーブルのセルをクリックすることで、表示メモリアドレス/ビットの割り付けが行えます。

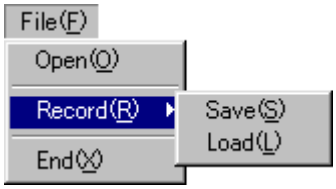
メッセージ領域

[Tool]メニューから[Generate]を選択、あるいは[Generate]ボタンをクリックしてファイルを作成した際に、その結果を示すメッセージを表示します。

10.3.3 メニューとツールバーボタン

以下、各メニュー項目と、ツールバーボタンについて説明します。

[File]メニュー



Open
セグメントオプションドキュメントファイルを開きます。既存のファイルを修正する場合などに使用します。[Open]ボタンも同機能です。


[Open]ボタン

Record - Save
現在のオプション設定内容をファイル(セグメント割り付けデータファイル)に保存します。[Save]ボタンも同機能です。

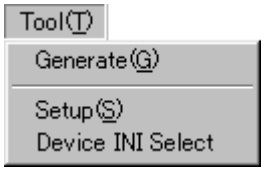

[Save]ボタン

Record - Load
セグメント割り付けデータファイルを読み込みます。[Load]ボタンも同機能です。


[Load]ボタン

End
winsogを終了します。

[Tool]メニュー



Generate
設定したセグメントオプションの内容でファイルを作成します。[Generate]ボタンも同機能です。


[Generate]ボタン

Setup
作成日や出力ファイル名、セグメントオプションドキュメントファイルに含めるコメントなどを設定します。[Setup]ボタンも同機能です。


[Setup]ボタン

Device INI Select
機種情報定義ファイル(s1c63xxx.ini)をロードします。[Device INI Select]ボタンも同機能です。このファイルのロードは最初に行っておく必要があります。

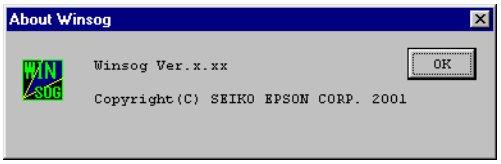

[Device INI Select]ボタン

[Help]メニュー

Version
winsogのバージョンを表示します。[Help]ボタンも同機能です。


[Help]ボタン

次のダイアログボックスが表示されます。閉じるには[OK]をクリックしてください。



10.3.4 オプション選択用ボタン

オプション設定領域には以下のボタンが用意されています。

OUTPUT Optionボタン

SEG端子の出力仕様を選択するボタンです。[SEGMENT DECODE TABLE]のSPECのセルをクリックして選択した場合に有効となります。

Seg	LCDセグメント出力を選択します。
Comp	DC-コンプリメンタリ出力を選択します。
Pch-	DC-Pchオープンドレイン出力を選択します。
Nch-	DC-Nchオープンドレイン出力を選択します。
M	セグメント/コモン共有出力を選択します。

[Delete]ボタン

Delete　選択したセグメント割り付けをクリアします。[Delete]キーも同機能です。

10.3.5 操作手順

基本的な操作手順を以下に示します。

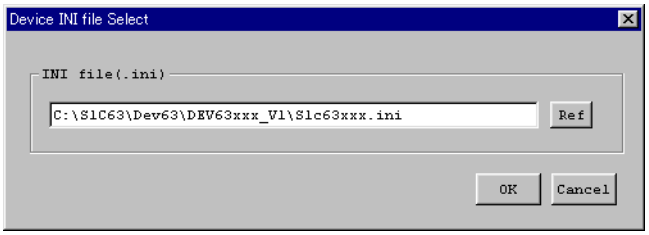
(1)機種情報定義ファイルのロード

最初に機種情報定義ファイル(s1c63xxx.ini)を選択してロードします。

[Tool]メニューから[Device INI Select]を選択するか、[Device INI Select]ボタンをクリックします。

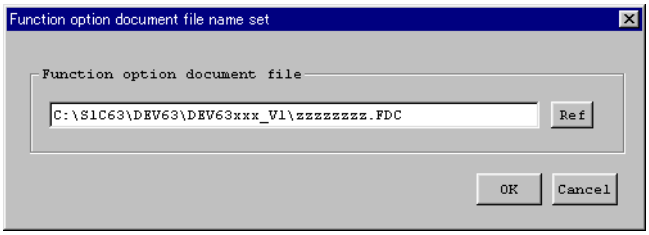

[Device INI Select]ボタン

次のダイアログが表示されますので、テキストボックスにパスを含むファイル名を入力するか、[Ref]ボタンをクリックしてファイルの選択を行ってください。



[OK]をクリックすると、ファイルをロードします。指定したファイルが存在し、内容に問題がなければ、読み込まれた機種情報によりwinsog内の各種設定が初期化されます。
ファイルのロードを中止するには[Cancel]をクリックします。

一度、機種情報定義ファイルを選択すると、次回の起動時は同じファイルが自動的にロードされます。機種情報定義ファイルをロード後、ファンクションオプションドキュメントファイルが必要な機種ではそのファイル名を入力するダイアログボックスが表示されますので、テキストボックスにパスも含め入力してください。あるいは、[Ref]ボタンをクリックしてファイルを選択してください。

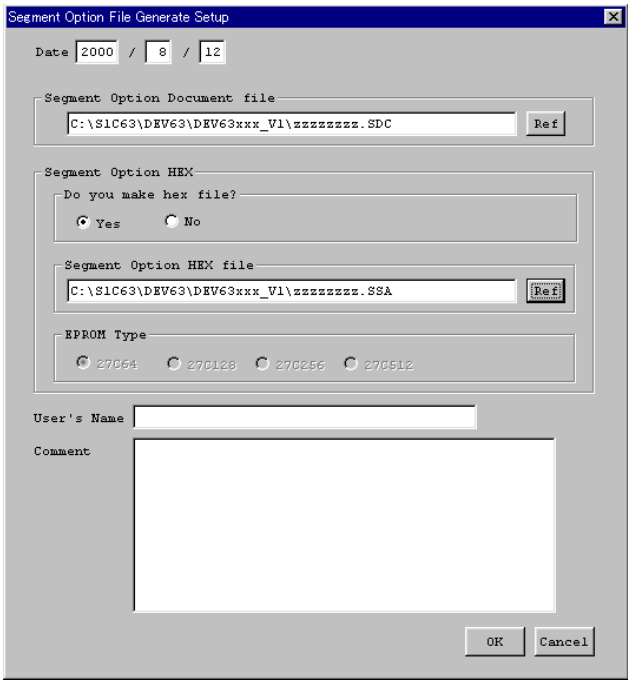


注: オプションをすでに設定している状態で機種情報定義ファイルをロードすると、設定がすべてクリアされます。

(2) セットアップ

[Tool]メニューから[Setup]を選択するか、[Setup]ボタンをクリックして[Setup]ダイアログボックスを表示させ、必要な選択と入力を行います。


[Setup]ボタン



Date
現在の日付が表示されます。必要に応じて変更してください。

Segment Option Document file
作成するセグメントオプションドキュメントファイル名を、ここで指定します。デフォルトで表示される名前を修正して使用してください。[Ref]ボタンで他のフォルダも参照できます。

Segment Option HEX
Do you make hex file?
セグメントオプションHEXファイルを作成するか選択します。
ICEを使用したデバッグを行う場合は作成してください。

Segment Option HEX file
セグメントオプションHEXファイルを作成する場合に、そのファイル名をここで指定します。デフォルトで表示される名前を修正して使用してください。[Ref]ボタンで他のフォルダも参照できます。

EPROM Type
これはS1C63 Familyでは選択できません。

User's Name
お客様の会社名を入力します。最大40文字まで入力することができます。英文字、数字、記号およびスペースが入力可能です。
ここに入力した内容は、セグメントオプションドキュメントファイルのUSER'S NAMEフィールドに記録されます。

Comment
コメントを入力します。1行に入力可能な文字数は50文字まで、最大10行まで入力することができます。英文字、数字、記号およびスペースが入力可能です。また、[Enter]キーで改行できます。
なお、コメントには、次のような内容を含めるようにお願いします。

・事業所、所属
・所在地、電話番号、FAX番号
・その他、技術情報など

ここに入力した内容は、セグメントオプションドキュメントファイルのCOMMENTフィールドに記録されます。

上記の必要な項目を入力後、[OK]をクリックすると設定内容が保存され、ダイアログボックスが閉じます。設定内容は即時有効となります。
[Cancel]をクリックした場合、現在の設定は変更されずにダイアログボックスが閉じます。

注: • ファイル名の指定には以下の制限があります。
1. パスを含めたファイル名指定の文字数は最大2048文字です。
2. ファイル名(拡張子を除く)は最大15文字、拡張子は最大3文字です。
3. ファイル名の先頭にハイフン(-)は使用できません。また、ディレクトリ名(フォルダ名)、ファイル名、拡張子に、以下の記号の使用を禁止します。
/ : , ; * ? " < > ¦

• User's NameとCommentに以下の記号は使用できません。
$ ¥ ¦ `

(3) セグメント出力の設定

S1C63 Familyでセグメントオプションが設定されているLCD駆動回路は、通常、2端子ごと(機種によっては端子個々)にセグメント出力とDC出力の選択が可能です。LCDパネルの駆動に用いる場合はセグメント出力を選択します。
セグメント出力ポートはセグメントデコーダを内蔵しており、表示メモリ領域の任意のアドレス、データビットを任意のセグメントに割り付けることができます。このセグメントメモリのビットを1に設定すると割り付けられたセグメントが点灯し、0にすると消灯します。セグメントと表示メモリビットは1対1に対応しており、複数のセグメントに同一の表示メモリビットを重複して設定することはできません。したがって、セグメントをすべて異なるアドレス、データビットにする必要があります。
表示メモリマップおよびセグメント割り付けの詳細については、各機種のテクニカルマニュアルを参照してください。
以下の説明は、コモン端子がCOM0～COM3の4本あるものとして行います。

セグメントの割り付けは次のように行います。

1. [Memory Address/Data bit]のテーブルから、割り付けるメモリアドレス/データビットのセルをクリックして選択します。セルが青色に変わります。
間違ったセルを選択してしまった場合は、正しいセルを選択し直してください。
テーブルの横の行が表示メモリアドレスに対応します。Memory Address/Data bitのタイトルの横に表示されている16進数が表示メモリのベースアドレスで、テーブル内の各行にはアドレスの下位バイトのみが表示されます。テーブルの縦の列はデータビットに対応します。

2. [SEGMENT DECODE TABLE]から、1で選択したメモリアドレス/データビットを割り付けるSEG端子/COM端子のセルをクリックして選択します。セルにアドレス(上位2桁)とデータビット(下位1桁)を示す3桁の数値が表示され、セルは黄色に変わります。

   選択例:

   間違ったセルを選択してしまった場合は、[Delete]ボタンをクリックしてその割り付けをクリアし、再度1から指定し直してください。セルを範囲選択し、[Delete]ボタンでクリアすることも可能です。
   [SEGMENT DECODE TABLE]のセルを選択する前に、必ず[Memory Address/Data bit]のセルを選択してください。

3. 2で選択したセグメントのSPECのセルをクリックし、[Seg]ボタンをクリックします。
   セルはSを表示して赤色に変わります。これにより、そのセグメントがLCDセグメント出力端子に設定されます。
   セグメント出力とDC出力の選択が2端子ごとの場合は、ペアとなるもう一方の端子の仕様も同じに設定されます。

   選択例:

4. 1～2をLCD出力に使用するすべてのセグメントについて行います。なお、3の仕様の選択は、後から行ってもかまいません。
   1つのSEG端子の中で使用しないCOMのセルは空白のままにしておいてください。

   選択例:

(4)DC出力の設定

SEG端子を汎用DC出力として使用する場合も、"(3)セグメント出力の設定"のステップ1と2の手順でセグメント割り付けを行います。ただし、出力制御はCOM0に割り付けられた表示メモリが有効となり、COM1～COM3に割り付けられた表示メモリは無効となります。したがって、メモリアドレス/データビットはCOM0のセルにのみ設定し、COM1～COM3のセルは空白にしておきます。
DC出力の場合は、出力仕様としてコンプリメンタリ出力とNch(またはPch)オープンドレイン出力のどちらかを選択できます。
SPECのセルで、以下のボタンを使用して選択してください。
[Comp]ボタン:　コンプリメンタリ出力(C)
[Nch-]ボタン:　Nchオープンドレイン出力(N)
[Pch-]ボタン:　Pchオープンドレイン出力(P)
選択が2端子ごとの場合は、ペアとなるもう一方の端子の仕様も同じに設定されます。

選択例:

選択可能な出力仕様については、各機種のテクニカルマニュアルを参照してください。

(5)セグメント/コモン共有端子の出力設定

セグメント出力とコモン出力を共有する端子は、ファンクションオプションの選択によって出力内容が決まります。
SEG端子として使用する場合は前記のように割り付けを行い、使用しないCOMのセルは空白にしておきます。COM端子として使用する場合は割り付けを行わずに、出力仕様にセグメント/コモン共有出力([M]ボタン)を選択します。

注: この設定はセグメント/コモン共有端子がある機種にのみ必要です。

(6)未使用SEG端子の設定

LCD出力およびDC出力のどちらにも使用しないSEG端子は、[SEGMENT DECODE TABLE]のCOM0～COM3のセルを空白にしておきます。ただし、SPECのセルは空白が許されませんので、セグメント出力(S)を選択してください。

選択例:

(7)ファイルの作成

オプションの選択が終了後、[Tool]メニューから[Generate]を選択するか、[Generate]ボタンをクリックしてファイルを作成します。

 [Generate]ボタン

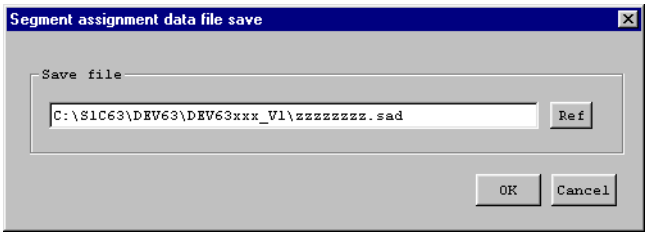
[Setup]ダイアログボックスで指定したセグメントオプションドキュメントファイルとセグメントオプションHEXファイル(指定時のみ)が作成されます。
ファイル作成が正常に終了した場合は、"Making file(s) is completed"がメッセージ領域に表示されます。
エラーが発生した場合は、エラーメッセージが表示されます。

(8)割り付け途中のセグメントオプションデータの保存

セグメントオプションの割り付け途中でデータを保存することができます。データを保存するには、[File]メニューから[Record - Save]を選択するか、[Save]ボタンをクリックします。

 [Save]ボタン

次のダイアログが表示されますので、テキストボックスにパスを含むファイル名を入力するか、[Ref]ボタンをクリックして保存先のフォルダを選択し、ファイルの選択またはファイル名の入力を行ってください。

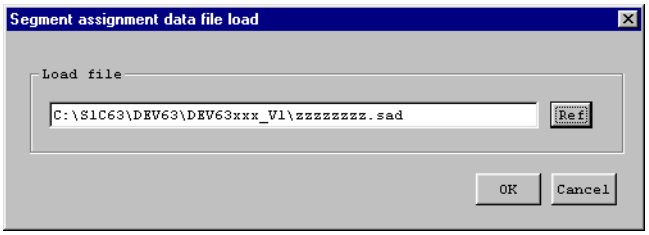


[OK]をクリックすると、割り付けデータを指定したファイルに保存します。
ファイルの保存を中止するには[Cancel]をクリックします。

保存したセグメント割り付けデータファイルをロードするには、[File]メニューから[Record - Load]を選択するか、[Load]ボタンをクリックします。

 [Load]ボタン

次のダイアログが表示されますので、テキストボックスにパスを含むファイル名を入力するか、[Ref]ボタンをクリックしてファイルの選択を行ってください。



[OK]をクリックすると、ファイルをロードします。指定したファイルが存在し、内容に問題がなければ、保存した割り付け内容が表示されますので、割り付けの続きを行うことができます。
ファイルのロードを中止するには[Cancel]をクリックします。

注: • セグメント割り付けデータファイルの読み込みは、機種情報定義ファイルがロードされている場合にのみ行えます。

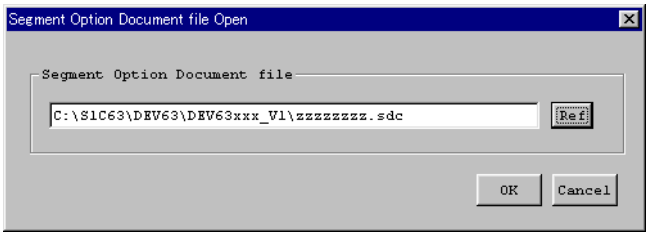
• ファンクションオプションドキュメントファイルが必要な機種では、起動時に読み込んだファンクションオプションドキュメントファイルにより設定条件が変わりますので、異なる内容があるセグメント割り付けデータファイルを読み込むことはできません。

(9)既存ドキュメントファイルの修正

既存のセグメントオプションドキュメントファイルを読み込んで、必要個所を修正することもできます。ファイルを読み込むには、[File]メニューから[Open]を選択するか、[Open]ボタンをクリックします。


[Open]ボタン

次のダイアログが表示されますので、テキストボックスにパスを含むファイル名を入力するか、[Ref]ボタンをクリックしてファイルの選択を行ってください。



[OK]をクリックすると、ファイルをロードします。指定したファイルが存在し、内容に問題がなければ、[Memory Address/Data bit]と[SEGMENT DECODE TABLE]がファイルの内容に更新されます。
ファイルのロードを中止するには[Cancel]をクリックします。

割り付けアドレスを変更する場合は、一度そのセルの割り付けを[Delete]ボタンでクリアしてから再度割り付けを行ってください。出力仕様を変更する場合も、一度SPECのセルを選択して[Delete]ボタンでクリアしてから再度選択してください。セルを範囲選択し、[Delete]ボタンでクリアすることも可能です。
ファイル名を変更せずに[Generate]を選択すると、上書きを確認するダイアログボックスが表示され、[Yes]をクリックして書き込み、[No]または[Cancel]をクリックして中止できます。ファイル名の変更は[Setup]ダイアログボックスで行ってください。

注: • セグメントオプションドキュメントファイルの読み込みは、機種情報定義ファイルがロードされている場合にのみ行えます。

• ファンクションオプションドキュメントファイルが必要な機種では、起動時に読み込んだファンクションオプションドキュメントファイルにより設定条件が変わりますので、異なる内容があるセグメントオプションドキュメントファイルを読み込むことはできません。

(10)終了

winsogを終了するには、[File]メニューから[End]を選択してください。

10.4 エラーメッセージ

winsogのエラーメッセージの一覧を示します。表示の"Dialog"はダイアログボックスに表示されるメッセージを、"Message"は[SOG]ウィンドウのメッセージ領域に表示されるメッセージを示します。

表10.4.1　winsogエラーメッセージ一覧

メッセージ	説　明	表示
File name error	ファイル名または拡張子名の文字数が使用可能範囲を超えている。	Dialog
Illegal character	入力禁止文字が入力された。	Dialog
Please input file name	ファイル名が未入力。	Dialog
Can't open File : xxxx	ファイル(xxxx)がオープンできない。	Dialog
INI file is not found	指定した機種情報定義ファイル(.ini)が存在しない。	Dialog
INI file does not include SOG information	指定した機種情報定義ファイル(.ini)にセグメントオプション情報が含まれていない。	Dialog
Function Option document file is not found	指定したファンクションオプションドキュメントファイルが存在しない。	Dialog
Function Option document file does not match INI file	指定したファンクションオプションドキュメントファイルの内容が機種情報定義ファイル(.ini)と異なる。	Dialog
Segment Option document file is not found	指定したセグメントオプションドキュメントファイルが存在しない。	Dialog
Segment Option document file does not match INI file	指定したセグメントオプションドキュメントファイルの内容が機種情報定義ファイル(.ini)と異なる。	Dialog
Segment assignment data file is not found	指定したセグメント割り付けデータファイルが存在しない。	Dialog
Segment assignment data file does not match INI file	指定したセグメント割り付けデータファイルの内容が機種情報定義ファイル(.ini)と異なる。	Dialog
Can't open File: xxxx Making file(s) is not completed	Generate実行時、ファイル(xxxx)がオープンできない。	Message
Can't write File: xxxx Making file(s) is not completed	Generate実行時、ファイル(xxxx)に書き込みができない。	Message
ERROR: SPEC is not set Making file(s) is not completed	空白のSPECセルがある状態でGenerateを実行した。	Message
ERROR: SEGMENT DECODE TABLE is not set. Making file(s) is not completed	選択したメモリアドレス/データビットセルがSEG/COM端子セルに割り付けられていない状態でGenerateを実行した。	Message

表10.4.2　winsogワーニングメッセージ

メッセージ	説　明	表示
Are you file update? xxxx is already exist	上書き確認メッセージ (指定したファイルは既に存在する。)	Dialog

10.5 注意事項

(1) winsogは日本語(2バイト文字)に対応していませんので、パスやファイル名に使用しないでください。

最新バージョンの制限事項やサポート機能、バグ情報についてはS5U1C63000Aのrel_windev_J.txtを参照してください。
最新バージョンのサポート機種はS5U1C63000AのReadme_J.txtを参照してください。

10.6 出力ファイル例

注: 表示メモリアドレス、SEG/COM端子数および出力仕様は機種により異なります。

●セグメントオプションドキュメントファイル例

```
* S1C63xxx SEGMENT OPTION DOCUMENT Vx.xx      ←バージョン
*
* FILE NAME    zzzzzzzz.SDC                  ←ファイル名(Setup]で指定)
* USER'S NAME  SEIKO EPSON CORPORATION       ←ユーザ名(Setup]で指定)
* INPUT DATE   yyyy/mm/dd                    ←作成年月日(Setup]で指定)
* COMMENT      SAMPLE DATA                   ←コメント(Setup]で指定)
*
*
* OPTION NO.xx                               ←オプション番号(機種により異なる)
*
* < LCD SEGMENT DECODE TABLE >
*
* SEG COM0 COM1 COM2 COM3 SPEC
*
   0  163  162  161  1F3  S                   ←セグメントデコードテーブル
   1  170  172  171  160  S
   2  143  142  141  1E1  S
   3  150  152  151  140  S
            :
  nn  3B0  3B1  3B2  3B3  S
*EOF                                         ←エンドマーク
```

●セグメント割り付けデータファイル例

```
* S1C63xxx SEGMENT OPTION DOCUMENT Vx.xx      ←バージョン
*
* FILE NAME    zzzzzzzz.SAD                  ←ファイル名
* USER'S NAME                                ←ユーザ名(Setup]で指定)
* INPUT DATE   yyyy/mm/dd                    ←作成年月日(Setup]で指定)
* COMMENT                                    ←コメント(Setup]で指定)
*
*
* OPTION NO.xx                               ←オプション番号(機種により異なる)
*
* < LCD SEGMENT DECODE TABLE >
*
* SEG COM0 COM1 COM2 COM3 SPEC
*
   0  163  162  161  1F3  S                   ←割り付け済みセグメントデータ
   1  170  172  171  160  S
   2  143  142  141  1E1  S
            :
  mm  FRE  FRE  FRE  FRE  X                   ←FRE: セグメントアドレス/データビット未割り付け
  nn  FRE  FRE  FRE  FRE  X                   ←X:   出力仕様未設定
  oo  FRE  FRE  FRE  FRE  X
*EOF                                         ←エンドマーク
```

●セグメントオプションHEXファイル例(モトローラS2フォーマット)

```
S2240000001603160216011F03FFFFFFFFFFFFFFFF1700170217011600FFFFFFFFFFFFFFFF23
S2240000201403140214011E01FFFFFFFFFFFFFFFF1500150215011400FFFFFFFFFFFFFFFF14
S2240000400103120212011D03FFFFFFFFFFFFFFFF12031F0013021F01FFFFFFFFFFFFFFFF7
S2240000600303130013011200FFFFFFFFFFFFFFFF1003100210011E02FFFFFFFFFFFFFFFF6
                                    :
S2240010E0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0B
S804000000FB
```

モトローラS2フォーマットについては"6.5.2 モトローラSファイル"を参照してください。

11 マスクデータチェッカ

11.1 マスクデータチェッカwinmdcの概要

マスクデータチェッカwinmdcは、HEXコンバータhx63によって生成されたコードROMとデータROMのHEXファイル、ファンクションオプションジェネレータwinfogによって生成されたファンクションオプションドキュメントファイル、セグメントオプションジェネレータwinsogによって生成されたセグメントオプションドキュメントファイルの各フォーマットをチェックし、マスクパターン生成のためのデータファイルを作成するソフトウェアツールです。
また、作成されたマスクデータファイルを元のファイル形式に復元する機能も持っています。

11.2 入出力ファイル

図11.2.1にwinmdcの入出力ファイルを示します。

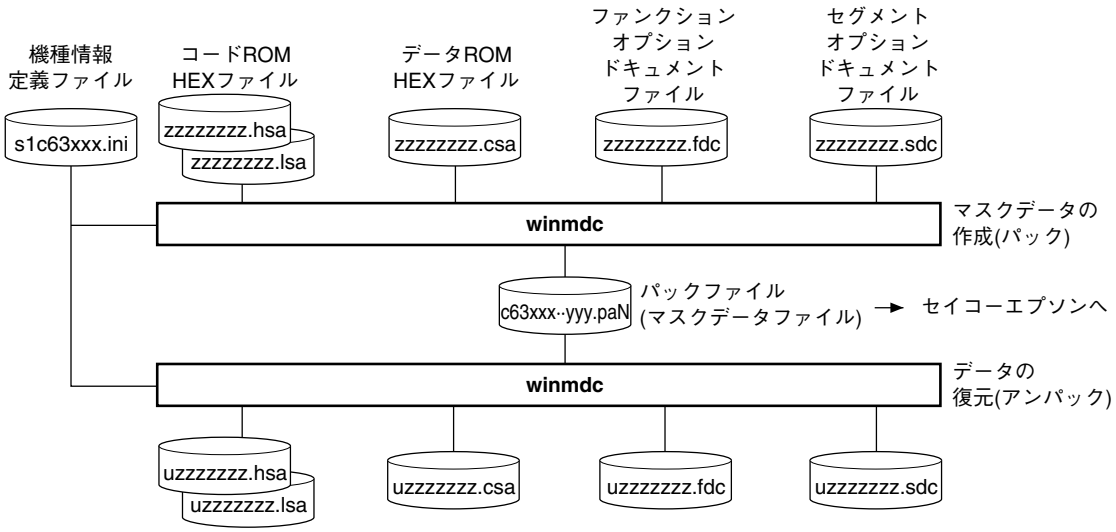


図11.2.1 winmdcの入出力ファイル

●機種情報定義ファイル(s1c63xxx.ini)

各機種のオプションリストやその他の情報が記録されています。必ずセイコーエプソンが提供するファイルを使用してください。このファイルはファイル名で示される機種にのみ有効です。ファイルの内容を修正したり、他の機種で使用しないでください。

●コードROM HEXファイル(zzzzzzzz.hsa, zzzzzzzz.lsa)

どちらもモトローラS2フォーマットのHEXファイルで、".hsa"にはオブジェクトコード(13ビット)の上位5ビットが、".lsa"にはオブジェクトコードの下位8ビットが記録されています。リンカlk63が出力したオブジェクトファイルをHEXコンバータhx63で変換して作成します。hx63については、"6 HEXコンバータ"を参照してください。

●データROM HEXファイル(zzzzzzzz.csa)

データROMに書き込むデータ(4ビット)が記録されたモトローラS2フォーマットのファイルです。コードROM HEXファイルと同時にhx63で作成されます。このファイルはデータROMを内蔵している機種にのみ存在します。

●ファンクションオプションドキュメントファイル(zzzzzzzz.fdc)

ファンクションオプションの選択内容が記録されたテキスト形式のファイルです。ファンクションオプションジェネレータwinfogで作成します。

●セグメントオプションドキュメントファイル（zzzzzzzz.sdc）
セグメントオプションの設定内容が記録されるテキスト形式のファイルです。セグメントオプションジェネレータwinsogで作成します。このファイルはセグメントオプションの設定されている機種にのみ存在します。

●パックファイル（c63xxx・・yyy.paN, N=0～）
上記のデータファイルを1つにまとめたテキスト形式のファイルです。これをマスクデータファイルとしてセイコーエプソンに提出していただきます。セイコーエプソンは、そのマスクデータファイルからICのマスクパターンを作成します。

＊ ファイル名の"xxx・・"は機種名です。ファイル名が"yyy"の部分は、お客様のカスタムコードが入りますので、セイコーエプソンより提示されるコードを入れてください。"zzzzzzzz"および"uzzzzzzz"の部分には任意の名前を付けてください。

11.3 操作方法

11.3.1 起動方法

エクスプローラからの起動

winmdc.exeアイコンをダブルクリックするか、スタートメニューからwinmdcを選択してください。
前回の実行時に機種情報定義ファイル（s1c63xxx.ini）を読み込んでいる場合は、winmdc起動時に同じファイルを自動的に読み込みます。
また、機種情報定義ファイルのアイコンをwinmdc.exeアイコンにドラッグすることによってもwinmdcが起動し、その機種情報定義ファイルを読み込みます。

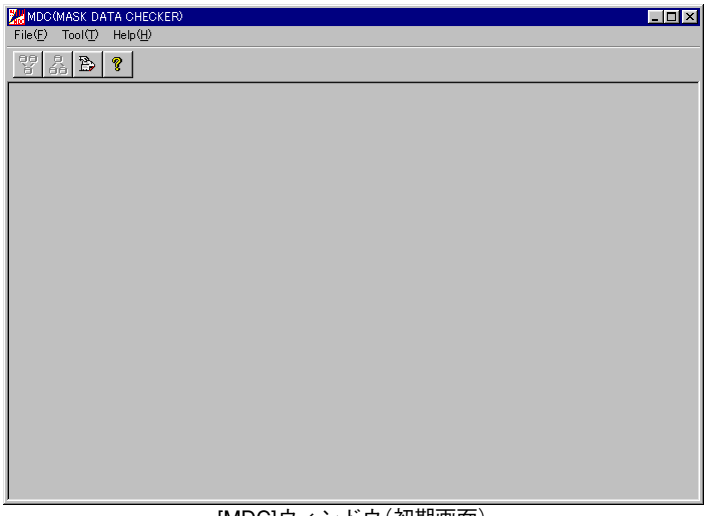
コマンド入力による起動

winmdcはコマンドプロンプトからも次のコマンドで起動可能です。

>winmdc [s1c63xxx.ini]↵

↵はリターンキーの入力を表します。
コマンドオプションとして機種情報定義ファイル（s1c63xxx.ini）が指定できます（パスも指定可能）。ここで指定すると、winmdc起動時に機種情報定義ファイルが読み込まれます。この指定は省略可能です。

起動すると[MDC]ウィンドウを表示します。



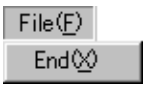
[MDC]ウィンドウ（初期画面）

＊ タイトルバーの機種名は、読み込んだ機種情報定義ファイルのファイル名（パスと拡張子を除く）です。
＊ ツールバーの[Pack]と[Unpack]ボタンは、機種情報定義ファイルが読み込まれると有効になります。

11.3.2 メニューとツールバーボタン

以下、各メニュー項目と、ツールバーボタンについて説明します。

[File]メニュー



End

winmdcを終了します。

[Tool]メニュー

Pack

ROMデータファイルとオプションドキュメントファイルをパックして、提出用のマスクデータファイルを作成します。[Pack]ボタンも同機能です。

 [Packボタン]

Unpack

パック後のファイルから元の形式のファイルを復元します。[Unpack]ボタンも同機能です。

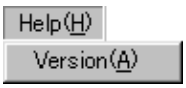
 [Unpack]ボタン

Device INI Select

機種情報定義ファイル(s1c63xxx.ini)をロードします。[Device INI Select]ボタンも同機能です。このファイルのロードは最初に行っておく必要があります。

 [Device INI Select]ボタン

[Help]メニュー

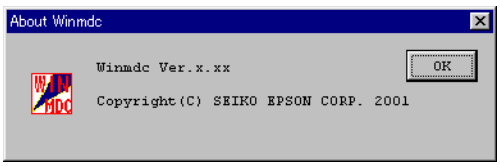


Version

winmdcのバージョンを表示します。[Help]ボタンも同機能です。

 [Help]ボタン

次のダイアログボックスが表示されます。閉じるには[OK]をクリックしてください。



11.3.3 操作手順

基本的な操作手順を以下に示します。

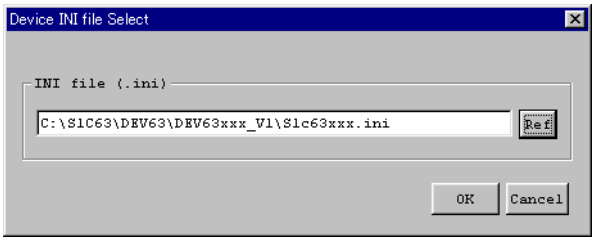
(1)機種情報定義ファイルのロード

最初に機種情報定義ファイル(s1c63xxx.ini)を選択してロードします。

[Tool]メニューから[Device INI Select]を選択するか、[Device INI Select]ボタンをクリックします。


[Device INI Select]ボタン

次のダイアログが表示されますので、テキストボックスにパスを含むファイル名を入力するか、[Ref]ボタンをクリックしてファイルの選択を行ってください。



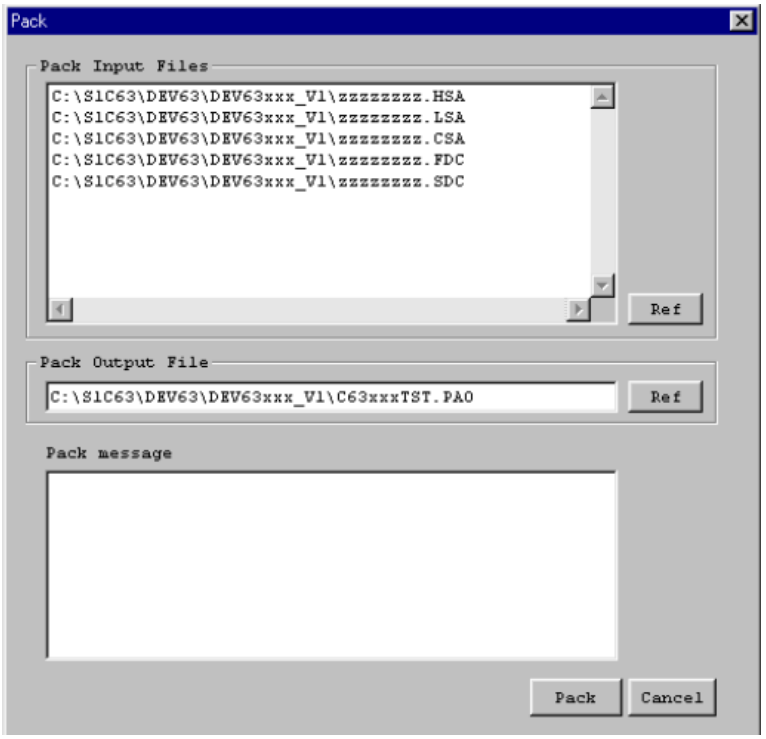
[OK]をクリックすると、ファイルをロードします。指定したファイルが存在し、内容に問題がなければ、読み込まれた機種情報によりwinmdc内の各種設定が初期化されます。
ファイルのロードを中止するには[Cancel]をクリックします。

一度、機種情報定義ファイルを選択すると、次回の起動時は同じファイルが自動的にロードされます。

(2)パック

1. [Tool]メニューから[Pack]を選択するか、ツールバーの[Pack]ボタンをクリックして[Pack]ダイアログボックスを表示させます。

   
   [Pack]ボタン



2. 入力するファイルを選択します。
   [Pack Input Files]には、機種情報定義ファイルで指定される種類のファイルがデフォルトのファイル名でリストされます。
   入力するデータファイルをリストと異なる名前で用意してある場合は、次の手順でファイル名を置き換えてください。
   a. リストボックス内の変更するファイル名をクリックして選択します。
   b. [Ref]ボタンをクリックし、入力するデータファイルを選択してください。
   これを、リストされているすべてのファイルについて行います。
   置き換える場合は、ファイルを間違えないように注意してください。入力ファイルが不正な場合、パック時にエラーとなります。

3. 出力ファイル名を設定します。
   [Pack Output File]テキストボックスで、出力するパックファイル名を指定します。デフォルトで表示される名前を修正して使用してください。[Ref]ボタンで他のフォルダも参照できます。
   出力ファイル名の拡張子は".pa0"としてください。一度セイコーエプソンにデータを提出された後、プログラム等の不具合により再提出される場合は、最後の数値を1つ増やして入力します。たとえば、二度目の提出ファイルは、"c63xxx‥yyy. pa1"とします。

注: ファイル名の指定には以下の制限があります。
1. パスを含めたファイル名指定の文字数は最大2048文字です。
2. ファイル名(拡張子を除く)は最大15文字、拡張子は最大3文字です。
3. ファイル名の先頭にハイフン(-)は使用できません。また、ディレクトリ名(フォルダ名)、ファイル名、拡張子に、以下の記号の使用を禁止します。
   / : , ; * ? " < > ¦

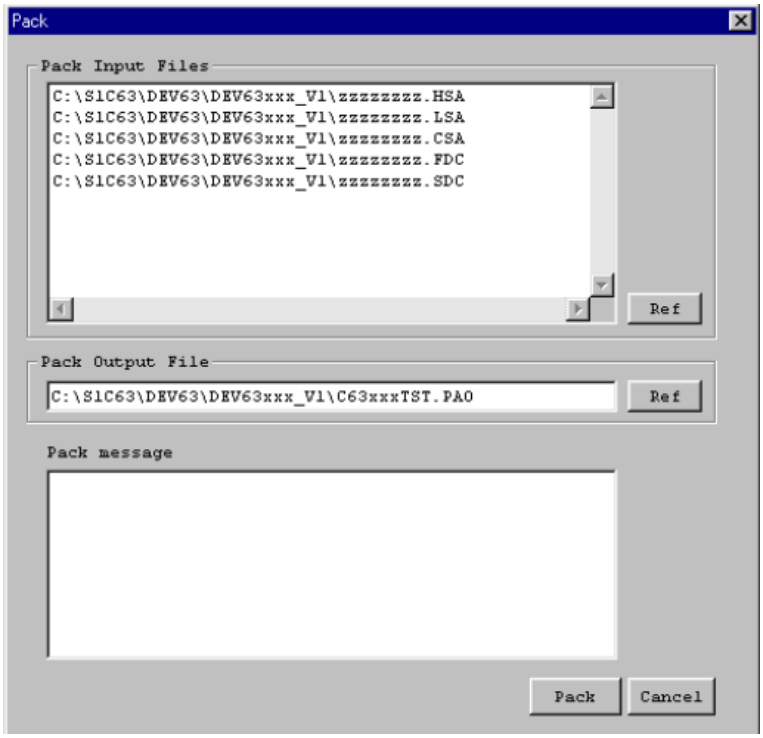
4. [Pack]ボタンをクリックしてパックを実行します。
   正常に終了した場合は、[Pack message]テキストボックスに"Pack completed !"が表示されます。
   エラーが発生した場合は、エラーメッセージが表示されます。

5. [Cancel]ボタンをクリックしてダイアログボックスを閉じます。
   パック実行前に[Cancel]ボタンをクリックして終了することもできます。

(3)アンパック

1. [Tool]メニューから[Unpack]を選択するか、ツールバーの[Unpack]ボタンをクリックして[Unpack]ダイアログボックスを表示させます。

   [Unpack]ボタン



2. アンパックするファイルを選択します。
   [Packed Input File]テキストボックスで、入力するパックファイル名を指定します。デフォルトで表示される名前を修正して使用してください。[Ref]ボタンでファイルを選択することもできます。

3. 出力ファイル名を設定します。
   [Unpack Output Files]には、機種情報定義ファイルで指定される種類のファイルがデフォルトのファイル名でリストされます。デフォルトで表示されるファイル名を次の手順で修正して使用してください。
   a. リストボックス内の変更するファイル名をクリックして選択します。
   b. [Ref]ボタンをクリックし、出力するフォルダを選択してファイル名の入力を行ってください。これを、リストされているすべてのファイルについて行います。拡張子は変更できません。

4. [Unpack]ボタンをクリックしてアンパックを実行します。
   正常に終了した場合は、[Unpack message]テキストボックスに"Unpack completed !"が表示されます。
   エラーが発生した場合は、エラーメッセージが表示されます。

5. [Cancel]ボタンをクリックしてダイアログボックスを閉じます。
   アンパック実行前に[Cancel]ボタンをクリックして終了することもできます。

(4)終了

winmdcを終了するには、[File]メニューから[End]を選択してください。

11.4 エラーメッセージ

winmdcのエラーメッセージの一覧を示します。表示の"Dialog"はダイアログボックスに表示されるメッセージを、"Message"は[Pack]または[Unpack]ダイアログボックスのメッセージ領域に表示されるメッセージを示します。

表11.4.1　I/Oエラーメッセージ一覧

メッセージ	説　明	表示
File name error	ファイル名または拡張子名の文字数が使用可能範囲を超えている。	Dialog
Illegal character	入力禁止文字が入力された。	Dialog
Please input file name	ファイル名が未入力。	Dialog
INI file is not found	指定した機種情報定義ファイル(.ini)が存在しない。	Dialog
INI file does not include MDC information	指定した機種情報定義ファイル(.ini)にMDC情報が含まれていない。	Dialog
Can't open file : xxxx	ファイル(xxxx)がオープンできない。	Dialog
Can't write file: xxxx	ファイル(xxxx)に書き込みができない。	Dialog

表11.4.2　ROMデータエラーメッセージ一覧

メッセージ	説　明	表示
Hex data error: Not S record.	データが"S"で始まっていない。	Message
Hex data error: Data is not sequential.	データが昇順に並んでいない。	Message
Hex data error: Illegal data.	不当なキャラクタがある。	Message
Hex data error: Too many data in one line.	1行中のデータ数が多すぎる。	Message
Hex data error: Check sum error.	チェックサムが合わない。	Message
Hex data error: ROM capacity over.	データ容量が大きい。(データサイズ>ROMサイズ)	Message
Hex data error: Not enough the ROM data.	データ容量が少ない。(データサイズ<ROMサイズ)	Message
Hex data error: Illegal start mark.	スタートマークが不当である。	Message
Hex data error: Illegal end mark.	エンドマークが不当である。	Message
Hex data error: Illegal comment.	データの最初の機種名表示が不当である。	Message

表11.4.3　ファンクションオプションデータエラーメッセージ一覧

メッセージ	説　明	表示
Option data error : Illegal model name.	機種名が不当である。	Message
Option data error : Illegal version.	バージョンが不当である。	Message
Option data error : Illegal option number.	オプションNo.が不当である。	Message
Option data error : Illegal select number.	選択肢No.が不当である。	Message
Option data error : Mask data is not enough.	マスクデータが充分でない。	Message
Option data error : Illegal start mark.	スタートマークが不当である。	Message
Option data error : Illegal end mark.	エンドマークが不当である。	Message

表11.4.4　セグメントオプションデータエラーメッセージ一覧

メッセージ	説　明	表示
LCD segment data error : Illegal model name.	機種名が不当である。	Message
LCD segment data error : Illegal version.	バージョンが不当である。	Message
LCD segment data error : Illegal segment No.	セグメントNo.が不当である。	Message
LCD segment data error : Illegal segment area.	表示メモリのアドレスが範囲外である。	Message
LCD segment data error : Illegal segment output specification.	出力仕様が不当である。	Message
LCD segment data error : Illegal data in this line.	16進数と出力仕様以外の記述がある。	Message
LCD segment data error : Data is not enough.	セグメントデータが充分でない。	Message
LCD segment data error : Illegal start mark.	スタートマークが不当である。	Message
LCD segment data error : Illegal end mark.	エンドマークが不当である。	Message

11.5 注意事項

(1) winmdcは日本語(2バイト文字)に対応していませんので、パスやファイル名に使用しないでください。

最新バージョンの制限事項やサポート機能、バグ情報についてはS5U1C63000Aのrel_windev_J.txtを参照してください。
最新バージョンのサポート機種はS5U1C63000AのReadme_J.txtを参照してください。

11.6 出力ファイル例

注: データの構成および内容は機種により異なります。

●パックファイル(マスクデータファイル)例

```
*
* S1C63xxx MASK DATA VER x.xx                    ←バージョン
*
¥PROM                                            ←コードROM HEXデータスタートマーク
S1C63xxxyyy PROGRAM ROM                          ←マスタスライス機種名
S224000000.............................┐
   :       :       :       :       :   │
S804000000FB                             │       "zzzzzzzz.hsa", "zzzzzzzz.lsa"
S224000000.............................┤
   :       :       :       :       :   │
S804000000FB                             ┘
¥END                                             ←コードROM HEXデータエンドマーク
¥CHROM                                           ←データROM HEXデータスタートマーク
S1C63xxxyyy CHARACTER ROM                        ←マスタスライス機種名
S224000000.............................┐       "zzzzzzzz.csa"
S804000000FB                             ┘
¥END                                             ←データROM HEXデータスタートマーク
¥FOPTION                                         ←ファンクションオプションスタートマーク
* S1C63xxx FUNCTION OPTION DOCUMENT Vx.xx ┐      ←機種名/バージョン
*                                        │
* FILE NAME    zzzzzzzz.FDC               │
* USER'S NAME                             │
* INPUT DATE   2000/06/27                 │
* COMMENT                                 │
*                                        │       "zzzzzzzz.fdc"
* *** OPTION NO.1 ***                     │
* --- OSC1 SYSTEM CLOCK ---               │
* CR 60KHz(Special Reset) ---- Selected   │
 OPT0101 03                              │
   :       :       :       :       :     │
 OPTii01 02                              │
*EOF                                     ┘
¥END                                             ←ファンクションオプションエンドマーク
¥SEGMENT                                         ←セグメントオプションスタートマーク
* S1C63xxx SEGMENT OPTION DOCUMENT Vx.xx ┐      ←機種名/バージョン
*                                        │
* FILE NAME    zzzzzzzz.SDC               │
* USER'S NAME                             │
* INPUT DATE   2000/9/1                   │
* COMMENT                                 │
*                                        │
*                                        │
* OPTION NO.mm                            │
*                                        │       "zzzzzzzz.sdc"
* < LCD SEGMENT DECODE TABLE >            │
*                                        │
* SEG COM0 COM1 COM2 COM3 SPEC            │
*                                        │
   0  000  001  002  003  S               │
   1  020  021  022  023  S               │
            :                            │
  nn  760  761  762  763  N               │
*EOF                                     ┘
¥END                                             ←セグメントオプションエンドマーク
```

S1C63 Family Assembler Package

Quick Reference

EPSON

CMOS 4-bit Single Chip Microcomputer

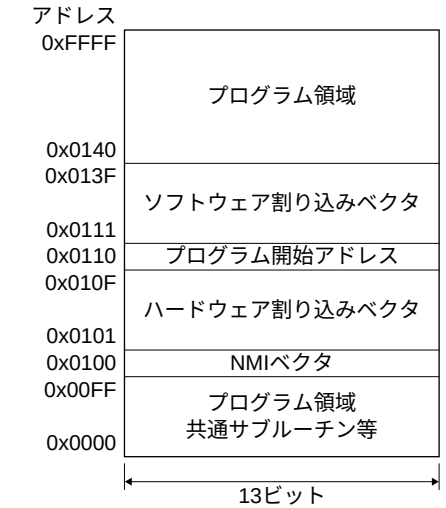
S1C63 Family Assembler Package

Quick Reference for Development

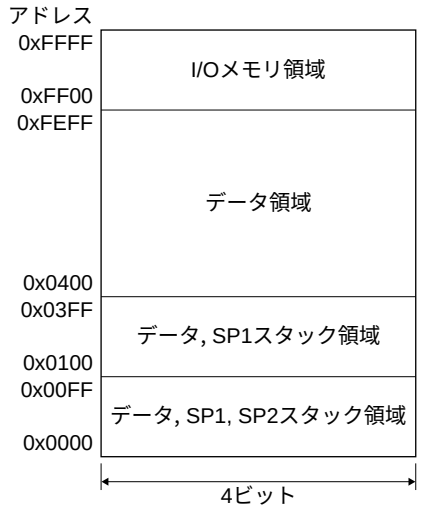
メモリマップ

S1C63000 Core CPU

プログラムメモリマップ



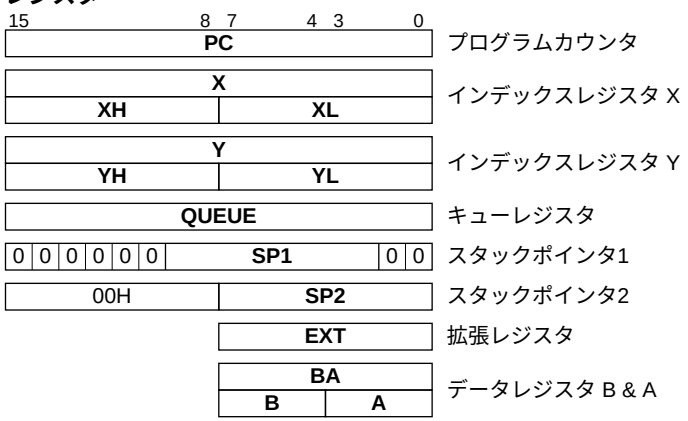
データメモリマップ



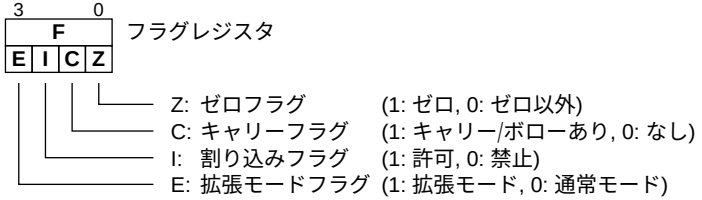
レジスタ

S1C63000 Core CPU

レジスタ



フラグ



命令一覧中のシンボル

レジスタ/レジスタデータ

%A, A:	データレジスタAまたはレジスタの内容 (4ビット)
%B, B:	データレジスタBまたはレジスタの内容 (4ビット)
%BA, BA:	データレジスタBAまたはレジスタの内容 (8ビット, Bレジスタが上位4ビット)
%X, X:	インデックスレジスタXまたはレジスタの内容 (16ビット)
%XH, XH:	インデックスレジスタXHまたはレジスタの内容 (Xレジスタの上位8ビット)
%XL, XL:	インデックスレジスタXLまたはレジスタの内容 (Xレジスタの下位8ビット)
%Y, Y:	インデックスレジスタYまたはレジスタの内容 (16ビット)
%YH, YH:	インデックスレジスタYHまたはレジスタの内容 (Yレジスタの上位8ビット)
%YL, YL:	インデックスレジスタYLまたはレジスタの内容 (Yレジスタの下位8ビット)
%F, F:	フラグレジスタFまたはレジスタの内容 (4ビット)
%EXT, EXT:	拡張レジスタEXTまたはレジスタの内容 (8ビット)
%SP1, SP1:	スタックポインタSP1またはスタックポインタの内容 (16ビット, 設定データ = SP1(9:2))
%SP2, SP2:	スタックポインタSP2またはスタックポインタの内容 (16ビット, 設定データ = SP2(7:0))
PC:	プログラムカウンタPCの内容 (16ビット)

メモリ/アドレス/メモリデータ

[%X], [X]:	Xによるレジスタ間接アドレッシングまたは指定メモリの内容
[%Y], [Y]:	Yによるレジスタ間接アドレッシングまたは指定メモリの内容
[00addr6]:	addr6による6ビット絶対アドレッシングまたは指定メモリの内容 (0x0000–0x003F)
[FFaddr6]:	addr6による6ビット絶対アドレッシングまたは指定メモリの内容 (0xFFC0–0xFFFF)
[00imm8]:	imm8による8ビット絶対アドレッシングまたは指定メモリの内容 (0x0000–0x00FF)
[FFimm8]:	imm8による8ビット絶対アドレッシングまたは指定メモリの内容 (0xFF00–0xFFFF)
[%SP1], [SP1]:	16ビットスタックアドレス指定または指定アドレスの内容
[%SP2], [SP2]:	4ビットスタックアドレス指定または指定アドレスの内容

即値

immN:	Nビット符号なし即値データ (N = 2, 4, 6または8)
i7–i0:	immNの構成ビット
n4:	4ビットn進指定データ
n3–n0:	n4の構成ビット
sign8:	符号付き8ビット即値データ
s7–s0:	sign8の構成ビット
addr6:	6ビット絶対アドレス
a5–a0:	addr6の構成ビット
00addr6:	addr6で指定されるアドレス (0x0000–0x003F)
FFaddr6:	addr6で指定されるアドレス (0xFFC0–0xFFFF)

機能

:	右側の内容が左側の項目にロードされることを示します。
:	左右のデータが交換されることを示します。
+:	加算
-:	減算
:	論理積
/ :	論理和
:	排他的論理和

フラグ

Z:	ゼロフラグ
C:	キャリーフラグ
I:	割り込みフラグ
E:	拡張モードフラグ
–:	変更なし
:	セット(1)、リセット(0)または変更なし
1:	セット(1)
0:	リセット(0)

Clk

実行サイクル数を示します。

シンボル

○は8ビットまたは6ビットの即値オペランドにシンボルが使用できることを示します。ただし、シンボルの値は即値で指定可能な範囲内であることが必要です。
シンボルマスクが記述されている場合は、その命令にシンボルおよび記載のシンボルマスクが使用可能です。
〈シンボルマスク〉

@l:	絶対アドレスの下位8ビットを取得
@h:	絶対アドレスの上位8ビットを取得
@rl:	相対アドレスの下位8ビットを取得
@rh:	相対アドレスの上位8ビットを取得
@xh:	絶対アドレスの上位8ビットをビット反転して取得

注意

"拡張機能"は、"LDB %EXT, imm8"命令の直後に実行した場合の動作を示します。

分　類	ニーモニック		基本機能	拡張機能 ("LDB %EXT, imm8"実行直後)	Clk	フラグ				シンボル
	オペコード	オペランド				E	I	C	Z	
4ビット データ転送	LD	%A,%A	A ‹ A	–	1	0	–	–	–	–
		%A,%B	A ‹ B	–	1	0	–	–	–	–
		%A,%F	A ‹ F	–	1	0	–	–	–	–
		%A,imm4	A ‹ imm4	–	1	0	–	–	–	–
		%A,[%X]	A ‹ [X]	A ‹ [00imm8]	1	0	–	–	–	–
		%A,[%X]+	A ‹ [X], X ‹ X+1	–	1	0	–	–	–	–
		%A,[%Y]	A ‹ [Y]	A ‹ [FFimm8]	1	0	–	–	–	–
		%A,[%Y]+	A ‹ [Y], Y ‹ Y+1	–	1	0	–	–	–	–
	LD	%B,%A	B ‹ A	–	1	0	–	–	–	–
		%B,%B	B ‹ B	–	1	0	–	–	–	–
		%B,imm4	B ‹ imm4	–	1	0	–	–	–	–
		%B,[%X]	B ‹ [X]	B ‹ [00imm8]	1	0	–	–	–	–
		%B,[%X]+	B ‹ [X], X ‹ X+1	–	1	0	–	–	–	–
		%B,[%Y]	B ‹ [Y]	B ‹ [FFimm8]	1	0	–	–	–	–
		%B,[%Y]+	B ‹ [Y], Y ‹ Y+1	–	1	0	–	–	–	–
	LD	%F,%A	F ‹ A	–	1	«	«	«	«	–
		%F,imm4	F ‹ imm4	–	1	«	«	«	«	–
	LD	[%X],%A	[X] ‹ A	[00imm8] ‹ A	1	0	–	–	–	–
		[%X],%B	[X] ‹ B	[00imm8] ‹ B	1	0	–	–	–	–
		[%X],imm4	[X] ‹ imm4	[00imm8] ‹ imm4	1	0	–	–	–	–
		[%X],[%Y]	[X] ‹ [Y]	–	2	0	–	–	–	–
		[%X],[%Y]+	[X] ‹ [Y], Y ‹ Y+1	–	2	0	–	–	–	–
		[%X]+,%A	[X] ‹ A, X ‹ X+1	–	1	0	–	–	–	–
		[%X]+,%B	[X] ‹ B, X ‹ X+1	–	1	0	–	–	–	–
		[%X]+,imm4	[X] ‹ imm4, X ‹ X+1	–	1	0	–	–	–	–
		[%X]+,[%Y]	[X] ‹ [Y], X ‹ X+1	–	2	0	–	–	–	–
		[%X]+,[%Y]+	[X] ‹ [Y], X ‹ X+1, Y ‹ Y+1	–	2	0	–	–	–	–
	LD	[%Y],%A	[Y] ‹ A	[FFimm8] ‹ A	1	0	–	–	–	–
		[%Y],%B	[Y] ‹ B	[FFimm8] ‹ B	1	0	–	–	–	–
		[%Y],imm4	[Y] ‹ imm4	[FFimm8] ‹ imm4	1	0	–	–	–	–
		[%Y],[%X]	[Y] ‹ [X]	–	2	0	–	–	–	–
		[%Y],[%X]+	[Y] ‹ [X], X ‹ X+1	–	2	0	–	–	–	–
		[%Y]+,%A	[Y] ‹ A, Y ‹ Y+1	–	1	0	–	–	–	–
		[%Y]+,%B	[Y] ‹ B, Y ‹ Y+1	–	1	0	–	–	–	–
		[%Y]+,imm4	[Y] ‹ imm4, Y ‹ Y+1	–	1	0	–	–	–	–
		[%Y]+,[%X]	[Y] ‹ [X], Y ‹ Y+1	–	2	0	–	–	–	–
		[%Y]+,[%X]+	[Y] ‹ [X], Y ‹ Y+1, X ‹ X+1	–	2	0	–	–	–	–

注意

分　類	ニーモニック		基本機能	拡張機能 ("LDB %EXT, imm8"実行直後)	Clk	フラグ				シンボル
	オペコード	オペランド				E	I	C	Z	
4ビット データ転送	EX	%A,%B	A « B	–	1	0	–	–	–	–
	EX	%A,[%X]	A « [X]	A « [00imm8]	2	0	–	–	–	–
		%A,[%X]+	A « [X], X ‹ X+1	–	2	0	–	–	–	–
		%A,[%Y]	A « [Y]	A « [FFimm8]	2	0	–	–	–	–
		%A,[%Y]+	A « [Y], Y ‹ Y+1	–	2	0	–	–	–	–
	EX	%B,[%X]	B « [X]	B « [00imm8]	2	0	–	–	–	–
		%B,[%X]+	B « [X], X ‹ X+1	–	2	0	–	–	–	–
		%B,[%Y]	B « [Y]	B « [FFimm8]	2	0	–	–	–	–
		%B,[%Y]+	B « [Y], Y ‹ Y+1	–	2	0	–	–	–	–
算術演算	ADD	%A,%A	A ‹ A+A	–	1	0	–	«	«	–
		%A,%B	A ‹ A+B	–	1	0	–	«	«	–
		%A,imm4	A ‹ A+imm4	–	1	0	–	«	«	–
		%A,[%X]	A ‹ A+[X]	A ‹ A+[00imm8]	1	0	–	«	«	–
		%A,[%X]+	A ‹ A+[X], X ‹ X+1	–	1	0	–	«	«	–
		%A,[%Y]	A ‹ A+[Y]	A ‹ A+[FFimm8]	1	0	–	«	«	–
		%A,[%Y]+	A ‹ A+[Y], Y ‹ Y+1	–	1	0	–	«	«	–
	ADD	%B,%A	B ‹ B+A	–	1	0	–	«	«	–
		%B,%B	B ‹ B+B	–	1	0	–	«	«	–
		%B,imm4	B ‹ B+imm4	–	1	0	–	«	«	–
		%B,[%X]	B ‹ B+[X]	B ‹ B+[00imm8]	1	0	–	«	«	–
		%B,[%X]+	B ‹ B+[X], X ‹ X+1	–	1	0	–	«	«	–
		%B,[%Y]	B ‹ B+[Y]	B ‹ B+[FFimm8]	1	0	–	«	«	–
		%B,[%Y]+	B ‹ B+[Y], Y ‹ Y+1	–	1	0	–	«	«	–
	ADD	[%X],%A	[X] ‹ [X]+A	[00imm8] ‹ [00imm8]+A	2	0	–	«	«	–
		[%X],%B	[X] ‹ [X]+B	[00imm8] ‹ [00imm8]+B	2	0	–	«	«	–
		[%X],imm4	[X] ‹ [X]+imm4	[00imm8] ‹ [00imm8]+imm4	2	0	–	«	«	–
		[%X]+,%A	[X] ‹ [X]+A, X ‹ X+1	–	2	0	–	«	«	–
		[%X]+,%B	[X] ‹ [X]+B, X ‹ X+1	–	2	0	–	«	«	–
		[%X]+,imm4	[X] ‹ [X]+imm4, X ‹ X+1	–	2	0	–	«	«	–
	ADD	[%Y],%A	[Y] ‹ [Y]+A	[FFimm8] ‹ [FFimm8]+A	2	0	–	«	«	–
		[%Y],%B	[Y] ‹ [Y]+B	[FFimm8] ‹ [FFimm8]+B	2	0	–	«	«	–
		[%Y],imm4	[Y] ‹ [Y]+imm4	[FFimm8] ‹ [FFimm8]+imm4	2	0	–	«	«	–
		[%Y]+,%A	[Y] ‹ [Y]+A, Y ‹ Y+1	–	2	0	–	«	«	–
		[%Y]+,%B	[Y] ‹ [Y]+B, Y ‹ Y+1	–	2	0	–	«	«	–
		[%Y]+,imm4	[Y] ‹ [Y]+imm4, Y ‹ Y+1	–	2	0	–	«	«	–

注意

分　類	ニーモニック		基本機能	拡張機能 ("LDB %EXT, imm8"実行直後)	Clk	フラグ				シンボル
	オペコード	オペランド				E	I	C	Z	
算術演算	ADC	%A,%A	A ‹ A+A+C	–	1	0	–	«	«	–
		%A,%B	A ‹ A+B+C	–	1	0	–	«	«	–
		%A,imm4	A ‹ A+imm4+C	–	1	0	–	«	«	–
		%A,[%X]	A ‹ A+[X]+C	A ‹ A+[00imm8]+C	1	0	–	«	«	–
		%A,[%X]+	A ‹ A+[X]+C, X ‹ X+1	–	1	0	–	«	«	–
		%A,[%Y]	A ‹ A+[Y]+C	A ‹ A+[FFimm8]+C	1	0	–	«	«	–
		%A,[%Y]+	A ‹ A+[Y]+C, Y ‹ Y+1	–	1	0	–	«	«	–
	ADC	%B,%A	B ‹ B+A+C	–	1	0	–	«	«	–
		%B,%B	B ‹ B+B+C	–	1	0	–	«	«	–
		%B,imm4	B ‹ B+imm4+C	–	1	0	–	«	«	–
		%B,[%X]	B ‹ B+[X]+C	B ‹ B+[00imm8]+C	1	0	–	«	«	–
		%B,[%X]+	B ‹ B+[X]+C, X ‹ X+1	–	1	0	–	«	«	–
		%B,[%Y]	B ‹ B+[Y]+C	B ‹ B+[FFimm8]+C	1	0	–	«	«	–
		%B,[%Y]+	B ‹ B+[Y]+C, Y ‹ Y+1	–	1	0	–	«	«	–
	ADC	[%X],%A	[X] ‹ [X]+A+C	[00imm8] ‹ [00imm8]+A+C	2	0	–	«	«	–
		[%X],%B	[X] ‹ [X]+B+C	[00imm8] ‹ [00imm8]+B+C	2	0	–	«	«	–
		[%X],imm4	[X] ‹ [X]+imm4+C	[00imm8] ‹ [00imm8]+imm4+C	2	0	–	«	«	–
		[%X]+,%A	[X] ‹ [X]+A+C, X ‹ X+1	–	2	0	–	«	«	–
		[%X]+,%B	[X] ‹ [X]+B+C, X ‹ X+1	–	2	0	–	«	«	–
		[%X]+,imm4	[X] ‹ [X]+imm4+C, X ‹ X+1	–	2	0	–	«	«	–
	ADC	[%Y],%A	[Y] ‹ [Y]+A+C	[FFimm8] ‹ [FFimm8]+A+C	2	0	–	«	«	–
		[%Y],%B	[Y] ‹ [Y]+B+C	[FFimm8] ‹ [FFimm8]+B+C	2	0	–	«	«	–
		[%Y],imm4	[Y] ‹ [Y]+imm4+C	[FFimm8] ‹ [FFimm8]+imm4+C	2	0	–	«	«	–
		[%Y]+,%A	[Y] ‹ [Y]+A+C, Y ‹ Y+1	–	2	0	–	«	«	–
		[%Y]+,%B	[Y] ‹ [Y]+B+C, Y ‹ Y+1	–	2	0	–	«	«	–
		[%Y]+,imm4	[Y] ‹ [Y]+imm4+C, Y ‹ Y+1	–	2	0	–	«	«	–
	SUB	%A,%A	A ‹ A-A	–	1	0	–	«	«	–
		%A,%B	A ‹ A-B	–	1	0	–	«	«	–
		%A,imm4	A ‹ A-imm4	–	1	0	–	«	«	–
		%A,[%X]	A ‹ A-[X]	A ‹ A-[00imm8]	1	0	–	«	«	–
		%A,[%X]+	A ‹ A-[X], X ‹ X+1	–	1	0	–	«	«	–
		%A,[%Y]	A ‹ A-[Y]	A ‹ A-[FFimm8]	1	0	–	«	«	–
		%A,[%Y]+	A ‹ A-[Y], Y ‹ Y+1	–	1	0	–	«	«	–
	SUB	%B,%A	B ‹ B-A	–	1	0	–	«	«	–
		%B,%B	B ‹ B-A	–	1	0	–	«	«	–
		%B,imm4	B ‹ B-imm4	–	1	0	–	«	«	–

注意

分　類	ニーモニック		基本機能	拡張機能 ("LDB %EXT, imm8"実行直後)	Clk	フラグ				シンボル
	オペコード	オペランド				E	I	C	Z	
算術演算	SUB	%B,[%X]	B ‹ B-[X]	B ‹ B-[00imm8]	1	0	–	«	«	–
		%B,[%X]+	B ‹ B-[X], X ‹ X+1	–	1	0	–	«	«	–
		%B,[%Y]	B ‹ B-[Y]	B ‹ B-[FFimm8]	1	0	–	«	«	–
		%B,[%Y]+	B ‹ B-[Y], Y ‹ Y+1	–	1	0	–	«	«	–
	SUB	[%X],%A	[X] ‹ [X]-A	[00imm8] ‹ [00imm8]-A	2	0	–	«	«	–
		[%X],%B	[X] ‹ [X]-B	[00imm8] ‹ [00imm8]-B	2	0	–	«	«	–
		[%X],imm4	[X] ‹ [X]-imm4	[00imm8] ‹ [00imm8]-imm4	2	0	–	«	«	–
		[%X]+,%A	[X] ‹ [X]-A, X ‹ X+1	–	2	0	–	«	«	–
		[%X]+,%B	[X] ‹ [X]-B, X ‹ X+1	–	2	0	–	«	«	–
		[%X]+,imm4	[X] ‹ [X]-imm4, X ‹ X+1	–	2	0	–	«	«	–
	SUB	[%Y],%A	[Y] ‹ [Y]-A	[FFimm8] ‹ [FFimm8]-A	2	0	–	«	«	–
		[%Y],%B	[Y] ‹ [Y]-B	[FFimm8] ‹ [FFimm8]-B	2	0	–	«	«	–
		[%Y],imm4	[Y] ‹ [Y]-imm4	[FFimm8] ‹ [FFimm8]-imm4	2	0	–	«	«	–
		[%Y]+,%A	[Y] ‹ [Y]-A, Y ‹ Y+1	–	2	0	–	«	«	–
		[%Y]+,%B	[Y] ‹ [Y]-B, Y ‹ Y+1	–	2	0	–	«	«	–
		[%Y]+,imm4	[Y] ‹ [Y]-imm4, Y ‹ Y+1	–	2	0	–	«	«	–
	SBC	%A,%A	A ‹ A-A-C	–	1	0	–	«	«	–
		%A,%B	A ‹ A-B-C	–	1	0	–	«	«	–
		%A,imm4	A ‹ A-imm4-C	–	1	0	–	«	«	–
		%A,[%X]	A ‹ A-[X]-C	A ‹ A-[00imm8]-C	1	0	–	«	«	–
		%A,[%X]+	A ‹ A-[X]-C, X ‹ X+1	–	1	0	–	«	«	–
		%A,[%Y]	A ‹ A-[Y]-C	A ‹ A-[FFimm8]-C	1	0	–	«	«	–
		%A,[%Y]+	A ‹ A-[Y]-C, Y ‹ Y+1	–	1	0	–	«	«	–
	SBC	%B,%A	B ‹ B-A-C	–	1	0	–	«	«	–
		%B,%B	B ‹ B-B-C	–	1	0	–	«	«	–
		%B,imm4	B ‹ B-imm4-C	–	1	0	–	«	«	–
		%B,[%X]	B ‹ B-[X]-C	B ‹ B-[00imm8]-C	1	0	–	«	«	–
		%B,[%X]+	B ‹ B-[X]-C, X ‹ X+1	–	1	0	–	«	«	–
		%B,[%Y]	B ‹ B-[Y]-C	B ‹ B-[FFimm8]-C	1	0	–	«	«	–
		%B,[%Y]+	B ‹ B-[Y]-C, Y ‹ Y+1	–	1	0	–	«	«	–
	SBC	[%X],%A	[X] ‹ [X]-A-C	[00imm8] ‹ [00imm8]-A-C	2	0	–	«	«	–
		[%X],%B	[X] ‹ [X]-B-C	[00imm8] ‹ [00imm8]-B-C	2	0	–	«	«	–
		[%X],imm4	[X] ‹ [X]-imm4-C	[00imm8] ‹ [00imm8]-imm4-C	2	0	–	«	«	–
		[%X]+,%A	[X] ‹ [X]-A-C, X ‹ X+1	–	2	0	–	«	«	–
		[%X]+,%B	[X] ‹ [X]-B-C, X ‹ X+1	–	2	0	–	«	«	–
		[%X]+,imm4	[X] ‹ [X]-imm4-C, X ‹ X+1	–	2	0	–	«	«	–

注意

分　類	ニーモニック		基本機能	拡張機能 ("LDB %EXT, imm8"実行直後)	Clk	フラグ				シンボル
	オペコード	オペランド				E	I	C	Z	
算術演算	SBC	[%Y],%A	[Y] ‹ [Y]-A-C	[FFimm8] ‹ [FFimm8]-A-C	2	0	–	«	«	–
		[%Y],%B	[Y] ‹ [Y]-B-C	[FFimm8] ‹ [FFimm8]-B-C	2	0	–	«	«	–
		[%Y],imm4	[Y] ‹ [Y]-imm4-C	[FFimm8] ‹ [FFimm8]-imm4-C	2	0	–	«	«	–
		[%Y]+,%A	[Y] ‹ [Y]-A-C, Y ‹ Y+1	–	2	0	–	«	«	–
		[%Y]+,%B	[Y] ‹ [Y]-B-C, Y ‹ Y+1	–	2	0	–	«	«	–
		[%Y]+,imm4	[Y] ‹ [Y]-imm4-C, Y ‹ Y+1	–	2	0	–	«	«	–
	CMP	%A,%A	A-A	–	1	0	–	«	«	–
		%A,%B	A-B	–	1	0	–	«	«	–
		%A,imm4	A-imm4	–	1	0	–	«	«	–
		%A,[%X]	A-[X]	A-[00imm8]	1	0	–	«	«	–
		%A,[%X]+	A-[X], X ‹ X+1	–	1	0	–	«	«	–
		%A,[%Y]	A-[Y]	A-[FFimm8]	1	0	–	«	«	–
		%A,[%Y]+	A-[Y], Y ‹ Y+1	–	1	0	–	«	«	–
	CMP	%B,%A	B-A	–	1	0	–	«	«	–
		%B,%B	B-B	–	1	0	–	«	«	–
		%B,imm4	B-imm4	–	1	0	–	«	«	–
		%B,[%X]	B-[X]	B-[00imm8]	1	0	–	«	«	–
		%B,[%X]+	B-[X], X ‹ X+1	–	1	0	–	«	«	–
		%B,[%Y]	B-[Y]	B-[FFimm8]	1	0	–	«	«	–
		%B,[%Y]+	B-[Y], Y ‹ Y+1	–	1	0	–	«	«	–
	CMP	[%X],%A	[X]-A	[00imm8]-A	2	0	–	«	«	–
		[%X],%B	[X]-B	[00imm8]-B	2	0	–	«	«	–
		[%X],imm4	[X]-imm4	[00imm8]-imm4	2	0	–	«	«	–
		[%X]+,%A	[X]-A, X ‹ X+1	–	2	0	–	«	«	–
		[%X]+,%B	[X]-B, X ‹ X+1	–	2	0	–	«	«	–
		[%X]+,imm4	[X]-imm4, X ‹ X+1	–	2	0	–	«	«	–
	CMP	[%Y],%A	[Y]-A	[FFimm8]-A	2	0	–	«	«	–
		[%Y],%B	[Y]-B	[FFimm8]-B	2	0	–	«	«	–
		[%Y],imm4	[Y]-imm4	[FFimm8]-imm4	2	0	–	«	«	–
		[%Y]+,%A	[Y]-A, Y ‹ Y+1	–	2	0	–	«	«	–
		[%Y]+,%B	[Y]-B, Y ‹ Y+1	–	2	0	–	«	«	–
		[%Y]+,imm4	[Y]-imm4, Y ‹ Y+1	–	2	0	–	«	«	–
	INC	[00addr6]	[00addr6] ‹ [00addr6]+1	–	2	0	–	«	«	○
	DEC	[00addr6]	[00addr6] ‹ [00addr6]-1	–	2	0	–	«	«	○
	ADC	%B,%A,n4	B ‹ N's adjust (B+A+C)	–	2	0	–	«	«	–
		%B,[%X],n4	B ‹ N's adjust (B+[X]+C)	B ‹ N's adjust (B+[00imm8]+C)	2	0	–	«	«	–

注意

分　類	ニーモニック		基本機能	拡張機能 ("LDB %EXT, imm8"実行直後)	Clk	フラグ				シンボル
	オペコード	オペランド				E	I	C	Z	
算術演算	ADC	%B,[%X]+,n4	B ‹ N's adjust (B+[X]+C), X ‹ X+1	–	2	0	–	«	«	–
		%B,[%Y],n4	B ‹ N's adjust (B+[Y]+C)	B ‹ N's adjust (B+[FFimm8]+C)	2	0	–	«	«	–
		%B,[%Y]+,n4	B ‹ N's adjust (B+[Y]+C), Y ‹ Y+1	–	2	0	–	«	«	–
	ADC	[%X],%B,n4	[X] ‹ N's adjust ([X]+B+C)	[00imm8] ‹ N's adjust ([00imm8]+B+C)	2	0	–	«	«	–
		[%X],0,n4	[X] ‹ N's adjust ([X]+0+C)	[00imm8] ‹ N's adjust ([00imm8]+0+C)	2	0	–	«	«	–
		[%X]+,%B,n4	[X] ‹ N's adjust ([X]+B+C), X ‹ X+1	–	2	0	–	«	«	–
		[%X]+,0,n4	[X] ‹ N's adjust ([X]+0+C), X ‹ X+1	–	2	0	–	«	«	–
	ADC	[%Y],%B,n4	[Y] ‹ N's adjust ([Y]+B+C)	[FFimm8] ‹ N's adjust ([FFimm8]+B+C)	2	0	–	«	«	–
		[%Y],0,n4	[Y] ‹ N's adjust ([Y]+0+C)	[FFimm8] ‹ N's adjust ([FFimm8]+0+C)	2	0	–	«	«	–
		[%Y]+,%B,n4	[Y] ‹ N's adjust ([Y]+B+C), Y ‹ Y+1	–	2	0	–	«	«	–
		[%Y]+,0,n4	[Y] ‹ N's adjust ([Y]+0+C), Y ‹ Y+1	–	2	0	–	«	«	–
	SBC	%B,%A,n4	B ‹ N's adjust (B-A-C)	–	2	0	–	«	«	–
		%B,[%X],n4	B ‹ N's adjust (B-[X]-C)	B ‹ N's adjust (B-[00imm8]-C)	2	0	–	«	«	–
		%B,[%X]+,n4	B ‹ N's adjust (B-[X]-C), X ‹ X+1	–	2	0	–	«	«	–
		%B,[%Y],n4	B ‹ N's adjust (B-[Y]-C)	B ‹ N's adjust (B-[FFimm8]-C)	2	0	–	«	«	–
		%B,[%Y]+,n4	B ‹ N's adjust (B-[Y]-C), Y ‹ Y+1	–	2	0	–	«	«	–
	SBC	[%X],%B,n4	[X] ‹ N's adjust ([X]-B-C)	[00imm8] ‹ N's adjust ([00imm8]-B-C)	2	0	–	«	«	–
		[%X],0,n4	[X] ‹ N's adjust ([X]-0-C)	[00imm8] ‹ N's adjust ([00imm8]-0-C)	2	0	–	«	«	–
		[%X]+,%B,n4	[X] ‹ N's adjust ([X]-B-C), X ‹ X+1	–	2	0	–	«	«	–
		[%X]+,0,n4	[X] ‹ N's adjust ([X]-0-C), X ‹ X+1	–	2	0	–	«	«	–
	SBC	[%Y],%B,n4	[Y] ‹ N's adjust ([Y]-B-C)	[FFimm8] ‹ N's adjust ([FFimm8]-B-C)	2	0	–	«	«	–
		[%Y],0,n4	[Y] ‹ N's adjust ([Y]-0-C)	[FFimm8] ‹ N's adjust ([FFimm8]-0-C)	2	0	–	«	«	–
		[%Y]+,%B,n4	[Y] ‹ N's adjust ([Y]-B-C), Y ‹ Y+1	–	2	0	–	«	«	–
		[%Y]+,0,n4	[Y] ‹ N's adjust ([Y]-0-C), Y ‹ Y+1	–	2	0	–	«	«	–
	INC	[%X],n4	[X] ‹ N's adjust ([X]+1)	[00imm8] ‹ N's adjust ([00imm8]+1)	2	0	–	«	«	–
		[%X]+,n4	[X] ‹ N's adjust ([X]+1), X ‹ X+1	–	2	0	–	«	«	–
	INC	[%Y],n4	[Y] ‹ N's adjust ([Y]+1)	[FFimm8] ‹ N's adjust ([FFimm8]+1)	2	0	–	«	«	–
		[%Y]+,n4	[Y] ‹ N's adjust ([Y]+1), Y ‹ Y+1	–	2	0	–	«	«	–
	DEC	[%X],n4	[X] ‹ N's adjust ([X]-1)	[00imm8] ‹ N's adjust ([00imm8]-1)	2	0	–	«	«	–
		[%X]+,n4	[X] ‹ N's adjust ([X]-1), X ‹ X+1	–	2	0	–	«	«	–
	DEC	[%Y],n4	[Y] ‹ N's adjust ([Y]-1)	[FFimm8] ‹ N's adjust ([FFimm8]-1)	2	0	–	«	«	–
		[%Y]+,n4	[Y] ‹ N's adjust ([Y]-1), Y ‹ Y+1	–	2	0	–	«	«	–

注意

分　類	ニーモニック		基本機能	拡張機能 ("LDB %EXT, imm8"実行直後)	Clk	フラグ				シンボル
	オペコード	オペランド				E	I	C	Z	
論理演算	AND	%A,%A	A ← A A	–	1	0	–	–	«	–
		%A,%B	A ← A B	–	1	0	–	–	«	–
		%A,imm4	A ← A imm4	–	1	0	–	–	«	–
		%A,[%X]	A ← A [X]	A ← A [00imm8]	1	0	–	–	«	–
		%A,[%X]+	A ← A [X], X ← X+1	–	1	0	–	–	«	–
		%A,[%Y]	A ← A [Y]	A ← A [FFimm8]	1	0	–	–	«	–
		%A,[%Y]+	A ← A [Y], Y ← Y+1	–	1	0	–	–	«	–
	AND	%B,%A	B ← B A	–	1	0	–	–	«	–
		%B,%B	B ← B B	–	1	0	–	–	«	–
		%B,imm4	B ← B imm4	–	1	0	–	–	«	–
		%B,[%X]	B ← B [X]	B ← B [00imm8]	1	0	–	–	«	–
		%B,[%X]+	B ← B [X], X ← X+1	–	1	0	–	–	«	–
		%B,[%Y]	B ← B [Y]	B ← B [FFimm8]	1	0	–	–	«	–
		%B,[%Y]+	B ← B [Y], Y ← Y+1	–	1	0	–	–	«	–
	AND	%F,imm4	F ← F imm4	–	1	0	0	0	0	–
	AND	[%X],%A	[X] ← [X] A	[00imm8] ← [00imm8] A	2	0	–	–	«	–
		[%X],%B	[X] ← [X] B	[00imm8] ← [00imm8] B	2	0	–	–	«	–
		[%X],imm4	[X] ← [X] imm4	[00imm8] ← [00imm8] imm4	2	0	–	–	«	–
		[%X]+,%A	[X] ← [X] A, X ← X+1	–	2	0	–	–	«	–
		[%X]+,%B	[X] ← [X] B, X ← X+1	–	2	0	–	–	«	–
		[%X]+,imm4	[X] ← [X] imm4, X ← X+1	–	2	0	–	–	«	–
	AND	[%Y],%A	[Y] ← [Y] A	[FFimm8] ← [FFimm8] A	2	0	–	–	«	–
		[%Y],%B	[Y] ← [Y] B	[FFimm8] ← [FFimm8] B	2	0	–	–	«	–
		[%Y],imm4	[Y] ← [Y] imm4	[FFimm8] ← [FFimm8] imm4	2	0	–	–	«	–
		[%Y]+,%A	[Y] ← [Y] A, Y ← Y+1	–	2	0	–	–	«	–
		[%Y]+,%B	[Y] ← [Y] B, Y ← Y+1	–	2	0	–	–	«	–
		[%Y]+,imm4	[Y] ← [Y] imm4, Y ← Y+1	–	2	0	–	–	«	–
	OR	%A,%A	A ← A A	–	1	0	–	–	«	–
		%A,%B	A ← A B	–	1	0	–	–	«	–
		%A,imm4	A ← A imm4	–	1	0	–	–	«	–
		%A,[%X]	A ← A [X]	A ← A [00imm8]	1	0	–	–	«	–
		%A,[%X]+	A ← A [X], X ← X+1	–	1	0	–	–	«	–
		%A,[%Y]	A ← A [Y]	A ← A [FFimm8]	1	0	–	–	«	–
		%A,[%Y]+	A ← A [Y], Y ← Y+1	–	1	0	–	–	«	–
	OR	%B,%A	B ← B A	–	1	0	–	–	«	–
		%B,%B	B ← B B	–	1	0	–	–	«	–

注意

分　類	ニーモニック		基本機能	拡張機能 ("LDB %EXT, imm8"実行直後)	Clk	フラグ				シンボル
	オペコード	オペランド				E	I	C	Z	
論理演算	OR	%B,imm4	B ‹ B imm4	–	1	0	–	–	«	–
		%B,[%X]	B ‹ B [X]	B ‹ B [00imm8]	1	0	–	–	«	–
		%B,[%X]+	B ‹ B [X], X ‹ X+1	–	1	0	–	–	«	–
		%B,[%Y]	B ‹ B [Y]	B ‹ B [FFimm8]	1	0	–	–	«	–
		%B,[%Y]+	B ‹ B [Y], Y ‹ Y+1	–	1	0	–	–	«	–
	OR	%F,imm4	F ‹ F imm4	–	1	1	1	1	1	–
	OR	[%X],%A	[X] ‹ [X] A	[00imm8] ‹ [00imm8] A	2	0	–	–	«	–
		[%X],%B	[X] ‹ [X] B	[00imm8] ‹ [00imm8] B	2	0	–	–	«	–
		[%X],imm4	[X] ‹ [X] imm4	[00imm8] ‹ [00imm8] imm4	2	0	–	–	«	–
		[%X]+,%A	[X] ‹ [X] A, X ‹ X+1	–	2	0	–	–	«	–
		[%X]+,%B	[X] ‹ [X] B, X ‹ X+1	–	2	0	–	–	«	–
		[%X]+,imm4	[X] ‹ [X] imm4, X ‹ X+1	–	2	0	–	–	«	–
	OR	[%Y],%A	[Y] ‹ [Y] A	[FFimm8] ‹ [FFimm8] A	2	0	–	–	«	–
		[%Y],%B	[Y] ‹ [Y] B	[FFimm8] ‹ [FFimm8] B	2	0	–	–	«	–
		[%Y],imm4	[Y] ‹ [Y] imm4	[FFimm8] ‹ [FFimm8] imm4	2	0	–	–	«	–
		[%Y]+,%A	[Y] ‹ [Y] A, Y ‹ Y+1	–	2	0	–	–	«	–
		[%Y]+,%B	[Y] ‹ [Y] B, Y ‹ Y+1	–	2	0	–	–	«	–
		[%Y]+,imm4	[Y] ‹ [Y] imm4, Y ‹ Y+1	–	2	0	–	–	«	–
	XOR	%A,%A	A ‹ A" A	–	1	0	–	–	«	–
		%A,%B	A ‹ A" B	–	1	0	–	–	«	–
		%A,imm4	A ‹ A" imm4	–	1	0	–	–	«	–
		%A,[%X]	A ‹ A" [X]	A ‹ A" [00imm8]	1	0	–	–	«	–
		%A,[%X]+	A ‹ A" [X], X ‹ X+1	–	1	0	–	–	«	–
		%A,[%Y]	A ‹ A" [Y]	A ‹ A" [FFimm8]	1	0	–	–	«	–
		%A,[%Y]+	A ‹ A" [Y], Y ‹ Y+1	–	1	0	–	–	«	–
	XOR	%B,%A	B ‹ B" A	–	1	0	–	–	«	–
		%B,%B	B ‹ B" B	–	1	0	–	–	«	–
		%B,imm4	B ‹ B" imm4	–	1	0	–	–	«	–
		%B,[%X]	B ‹ B" [X]	B ‹ B" [00imm8]	1	0	–	–	«	–
		%B,[%X]+	B ‹ B" [X], X ‹ X+1	–	1	0	–	–	«	–
		%B,[%Y]	B ‹ B" [Y]	B ‹ B" [FFimm8]	1	0	–	–	«	–
		%B,[%Y]+	B ‹ B" [Y], Y ‹ Y+1	–	1	0	–	–	«	–
	XOR	%F,imm4	F ‹ F" imm4	–	1	«	«	«	«	–
	XOR	[%X],%A	[X] ‹ [X]" A	[00imm8] ‹ [00imm8]" A	2	0	–	–	«	–
		[%X],%B	[X] ‹ [X]" B	[00imm8] ‹ [00imm8]" B	2	0	–	–	«	–
		[%X],imm4	[X] ‹ [X]" imm4	[00imm8] ‹ [00imm8]" imm4	2	0	–	–	«	–

注意

分　類	ニーモニック		基本機能	拡張機能 ("LDB %EXT, imm8"実行直後)	Clk	フラグ				シンボル
	オペコード	オペランド				E	I	C	Z	
論理演算	XOR	[%X]+,%A	[X] ‹ [X]" A, X ‹ X+1	–	2	0	–	–	«	–
		[%X]+,%B	[X] ‹ [X]" B, X ‹ X+1	–	2	0	–	–	«	–
		[%X]+,imm4	[X] ‹ [X]" imm4, X ‹ X+1	–	2	0	–	–	«	–
	XOR	[%Y],%A	[Y] ‹ [Y]" A	[FFimm8] ‹ [FFimm8]" A	2	0	–	–	«	–
		[%Y],%B	[Y] ‹ [Y]" B	[FFimm8] ‹ [FFimm8]" B	2	0	–	–	«	–
		[%Y],imm4	[Y] ‹ [Y]" imm4	[FFimm8] ‹ [FFimm8]" imm4	2	0	–	–	«	–
		[%Y]+,%A	[Y] ‹ [Y]" A, Y ‹ Y+1	–	2	0	–	–	«	–
		[%Y]+,%B	[Y] ‹ [Y]" B, Y ‹ Y+1	–	2	0	–	–	«	–
		[%Y]+,imm4	[Y] ‹ [Y]" imm4, Y ‹ Y+1	–	2	0	–	–	«	–
	BIT	%A,%A	A A	–	1	0	–	–	«	–
		%A,%B	A B	–	1	0	–	–	«	–
		%A,imm4	A imm4	–	1	0	–	–	«	–
		%A,[%X]	A [X]	A [00imm8]	1	0	–	–	«	–
		%A,[%X]+	A [X], X ‹ X+1	–	1	0	–	–	«	–
		%A,[%Y]	A [Y]	A [FFimm8]	1	0	–	–	«	–
		%A,[%Y]+	A [Y], Y ‹ Y+1	–	1	0	–	–	«	–
	BIT	%B,%A	B A	–	1	0	–	–	«	–
		%B,%B	B B	–	1	0	–	–	«	–
		%B,imm4	B imm4	–	1	0	–	–	«	–
		%B,[%X]	B [X]	B [00imm8]	1	0	–	–	«	–
		%B,[%X]+	B [X], X ‹ X+1	–	1	0	–	–	«	–
		%B,[%Y]	B [Y]	B [FFimm8]	1	0	–	–	«	–
		%B,[%Y]+	B [Y], Y ‹ Y+1	–	1	0	–	–	«	–
	BIT	[%X],%A	[X] A	[00imm8] A	1	0	–	–	«	–
		[%X],%B	[X] B	[00imm8] B	1	0	–	–	«	–
		[%X],imm4	[X] imm4	[00imm8] imm4	1	0	–	–	«	–
		[%X]+,%A	[X] A, X ‹ X+1	–	1	0	–	–	«	–
		[%X]+,%B	[X] B, X ‹ X+1	–	1	0	–	–	«	–
		[%X]+,imm4	[X] imm4, X ‹ X+1	–	1	0	–	–	«	–
	BIT	[%Y],%A	[Y] A	[FFimm8] A	1	0	–	–	«	–
		[%Y],%B	[Y] B	[FFimm8] B	1	0	–	–	«	–
		[%Y],imm4	[Y] imm4	[FFimm8] imm4	1	0	–	–	«	–
		[%Y]+,%A	[Y] A, Y ‹ Y+1	–	1	0	–	–	«	–
		[%Y]+,%B	[Y] B, Y ‹ Y+1	–	1	0	–	–	«	–
		[%Y]+,imm4	[Y] imm4, Y ‹ Y+1	–	1	0	–	–	«	–

注意

分　類	ニーモニック		基本機能	拡張機能 ("LDB %EXT, imm8"実行直後)	Clk	フラグ				シンボル
	オペコード	オペランド				E	I	C	Z	
論理演算	CLR	[00addr6],imm2	[00addr6] ‹ [00addr6] not (2^{imm2})	–	2	0	–	–	«	○
		[FFaddr6],imm2	[FFaddr6] ‹ [FFaddr6] not (2^{imm2})	–	2	0	–	–	«	○
	SET	[00addr6],imm2	[00addr6] ‹ [00addr6] (2^{imm2})	–	2	0	–	–	«	○
		[FFaddr6],imm2	[FFaddr6] ‹ [FFaddr6] (2^{imm2})	–	2	0	–	–	«	○
	TST	[00addr6],imm2	[00addr6] (2^{imm2})	–	1	0	–	–	«	○
		[FFaddr6],imm2	[FFaddr6] (2^{imm2})	–	1	0	–	–	«	○
シフト &ローテート	SLL	%A	A (C‹ D3‹ D2‹ D1‹ D0‹ 0)	–	1	0	–	«	«	–
		%B	B (C‹ D3‹ D2‹ D1‹ D0‹ 0)	–	1	0	–	«	«	–
		[%X]	[X] (C‹ D3‹ D2‹ D1‹ D0‹ 0)	[00imm8] (C‹ D3‹ D2‹ D1‹ D0‹ 0)	2	0	–	«	«	–
		[%X]+	[X] (C‹ D3‹ D2‹ D1‹ D0‹ 0), X ‹ X+1	–	2	0	–	«	«	–
		[%Y]	[Y] (C‹ D3‹ D2‹ D1‹ D0‹ 0)	[FFimm8] (C‹ D3‹ D2‹ D1‹ D0‹ 0)	2	0	–	«	«	–
		[%Y]+	[Y] (C‹ D3‹ D2‹ D1‹ D0‹ 0), Y ‹ Y+1	–	2	0	–	«	«	–
	SRL	%A	A (0fi D3fi D2fi D1fi D0fi C)	–	1	0	–	«	«	–
		%B	B (0fi D3fi D2fi D1fi D0fi C)	–	1	0	–	«	«	–
		[%X]	[X] (0fi D3fi D2fi D1fi D0fi C)	[00imm8] (0fi D3fi D2fi D1fi D0fi C)	2	0	–	«	«	–
		[%X]+	[X] (0fi D3fi D2fi D1fi D0fi C), X ‹ X+1	–	2	0	–	«	«	–
		[%Y]	[Y] (0fi D3fi D2fi D1fi D0fi C)	[FFimm8] (0fi D3fi D2fi D1fi D0fi C)	2	0	–	«	«	–
		[%Y]+	[Y] (0fi D3fi D2fi D1fi D0fi C), Y ‹ Y+1	–	2	0	–	«	«	–
	RL	%A	A (C‹ D3‹ D2‹ D1‹ D0‹ C)	–	1	0	–	«	«	–
		%B	B (C‹ D3‹ D2‹ D1‹ D0‹ C)	–	1	0	–	«	«	–
		[%X]	[X] (C‹ D3‹ D2‹ D1‹ D0‹ C)	[00imm8] (C‹ D3‹ D2‹ D1‹ D0‹ C)	2	0	–	«	«	–
		[%X]+	[X] (C‹ D3‹ D2‹ D1‹ D0‹ C), X ‹ X+1	–	2	0	–	«	«	–
		[%Y]	[Y] (C‹ D3‹ D2‹ D1‹ D0‹ C)	[FFimm8] (C‹ D3‹ D2‹ D1‹ D0‹ C)	2	0	–	«	«	–
		[%Y]+	[Y] (C‹ D3‹ D2‹ D1‹ D0‹ C), Y ‹ Y+1	–	2	0	–	«	«	–
	RR	%A	A (Cfi D3fi D2fi D1fi D0fi C)	–	1	0	–	«	«	–
		%B	B (Cfi D3fi D2fi D1fi D0fi C)	–	1	0	–	«	«	–
		[%X]	[X] (Cfi D3fi D2fi D1fi D0fi C)	[00imm8] (Cfi D3fi D2fi D1fi D0fi C)	2	0	–	«	«	–
		[%X]+	[X] (Cfi D3fi D2fi D1fi D0fi C), X ‹ X+1	–	2	0	–	«	«	–
		[%Y]	[Y] (Cfi D3fi D2fi D1fi D0fi C)	[FFimm8] (Cfi D3fi D2fi D1fi D0fi C)	2	0	–	«	«	–
		[%Y]+	[Y] (Cfi D3fi D2fi D1fi D0fi C), Y ‹ Y+1	–	2	0	–	«	«	–
8/16ビット 転送・演算	LDB	%BA,%XL	BA ‹ XL	–	1	0	–	–	–	–
		%BA,%XH	BA ‹ XH	–	1	0	–	–	–	–
		%BA,%YL	BA ‹ YL	–	1	0	–	–	–	–
		%BA,%YH	BA ‹ YH	–	1	0	–	–	–	–
		%BA,%EXT	BA ‹ EXT	–	1	0	–	–	–	–
		%BA,%SP1	BA ‹ SP1	–	1	0	–	–	–	–
		%BA,%SP2	BA ‹ SP2	–	1	0	–	–	–	–

注意

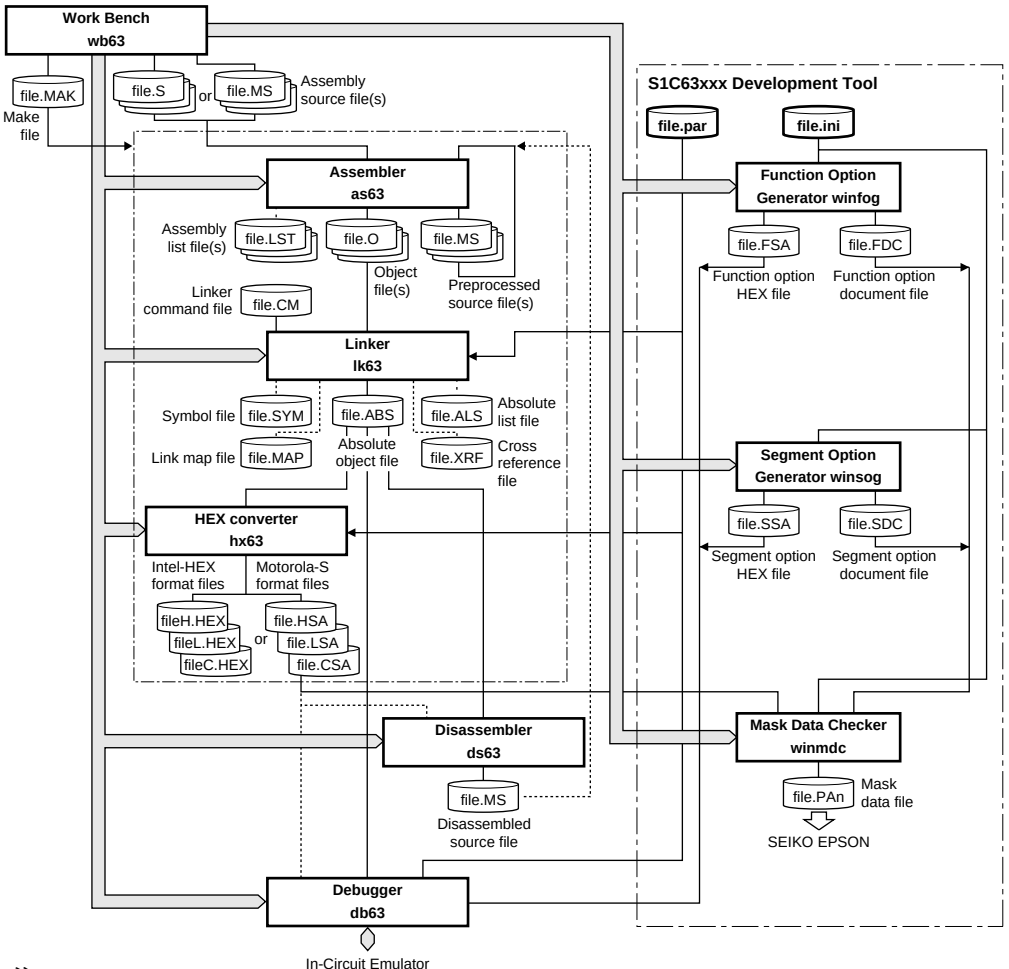
分　類	ニーモニック		基本機能	拡張機能 ("LDB %EXT, imm8"実行直後)	Clk	フラグ				シンボル
	オペコード	オペランド				E	I	C	Z	
8/16ビット転送・演算	LDB	%BA,imm8	BA ‹ imm8	–	1	0	–	–	–	○,@h
		%BA,[%X]+	A ‹ [X], B ‹ [X+1], X ‹ X+2	–	2	0	–	–	–	–
		%BA,[%Y]+	A ‹ [Y], B ‹ [Y+1], Y ‹ Y+2	–	2	0	–	–	–	–
	LDB	%XL,%BA	XL ‹ BA	–	1	0	–	–	–	–
		%XL,imm8	XL ‹ imm8	X ‹ imm16 (EXT内のimm8を上位8ビットとして使用)	1	0	–	–	–	○,@l
		%XH,%BA	XH ‹ BA	–	1	0	–	–	–	–
	LDB	%YL,%BA	YL ‹ BA	–	1	0	–	–	–	–
		%YL,imm8	YL ‹ imm8	Y ‹ imm16 (EXT内のimm8を上位8ビットとして使用)	1	0	–	–	–	○,@l
		%YH,%BA	YH ‹ BA	–	1	0	–	–	–	–
	LDB	%EXT,%BA	EXT ‹ BA	–	1	1	–	–	–	–
		%EXT,imm8	EXT ‹ imm8	–	1	1	–	–	–	○,@l,@h @rh,@xh
	LDB	%SP1,%BA	SP1 ‹ BA	–	1	0	–	–	–	–
		%SP2,%BA	SP2 ‹ BA	–	1	0	–	–	–	–
	LDB	[%X]+,%BA	[X] ‹ A, [X+1] ‹ B, X ‹ X+2	–	2	0	–	–	–	–
		[%X]+,imm8	[X] ‹ i3~0, [X+1] ‹ i7~4, X ‹ X+2	–	2	0	–	–	–	○,@l,@h
	LDB	[%Y]+,%BA	[Y] ‹ A, [Y+1] ‹ B, Y ‹ Y+2	–	2	0	–	–	–	–
	ADD	%X,%BA	X ‹ X+BA	–	1	0	–	–	«	–
		%X,sign8	X ‹ X+sign8 (sign8=-128~127)	X ‹ X+imm16 (EXT内のimm8を上位8ビットとして使用)	1	0	–	–	«	○,@l
		%Y,%BA	Y ‹ Y+BA	–	1	0	–	–	«	–
		%Y,sign8	Y ‹ Y+sign8 (sign8=-128~127)	Y ‹ Y+imm16 (EXT内のimm8を上位8ビットとして使用)	1	0	–	–	«	○,@l
	CMP	%X,imm8	X-imm8 (imm8=0~255)	X-imm16 (EXT内のimm8を上位8ビットとして使用)	1	0	–	«	«	○,@l
		%Y,imm8	Y-imm8 (imm8=0~255)	Y-imm16 (EXT内のimm8を上位8ビットとして使用)	1	0	–	«	«	○,@l
	INC	%SP1	SP1 ‹ SP1+1	–	1	0	–	–	«	–
		%SP2	SP2 ‹ SP2+1	–	1	0	–	–	«	–
	DEC	%SP1	SP1 ‹ SP1-1	–	1	0	–	–	«	–
		%SP2	SP2 ‹ SP2-1	–	1	0	–	–	–	–
スタック操作	PUSH	%A	[SP2-1] ‹ A, SP2 ‹ SP2-1	–	1	0	–	–	–	–
		%B	[SP2-1] ‹ B, SP2 ‹ SP2-1	–	1	0	–	–	–	–
		%F	[SP2-1] ‹ F, SP2 ‹ SP2-1	–	1	0	–	–	–	–
		%X	([(SP1-1)*4+3]~[(SP1-1)*4]) ‹ X, SP1 ‹ SP1-1	–	1	0	–	–	–	–
		%Y	([(SP1-1)*4+3]~[(SP1-1)*4]) ‹ Y, SP1 ‹ SP1-1	–	1	0	–	–	–	–
	POP	%A	A ‹ [SP2], SP2 ‹ SP2+1	–	1	0	–	–	–	–
		%B	B ‹ [SP2], SP2 ‹ SP2+1	–	1	0	–	–	–	–
		%F	F ‹ [SP2], SP2 ‹ SP2+1	–	1	«	«	«	«	–
		%X	X ‹ ([SP1*4+3]~[SP1*4]), SP1 ‹ SP1+1	–	1	0	–	–	–	–
		%Y	Y ‹ ([SP1*4+3]~[SP1*4]), SP1 ‹ SP1+1	–	1	0	–	–	–	–

注意

分　類	ニーモニック		基本機能	拡張機能 ("LDB %EXT, imm8"実行直後)	Clk	フラグ				シンボル
	オペコード	オペランド				E	I	C	Z	
分岐	JR	sign8	PC　PC+sign8+1 (sign8=-128~127)	PC　PC+sign16+1 (sign16=-32768~32767)*1	1	0	–	–	–	○,@rl
	JR	%A	PC　PC+A+1	–	1	0	–	–	–	–
		%BA	PC　PC+BA+1	–	1	0	–	–	–	–
	JR	[00addr6]	PC　PC+[00addr6]+1	–	2	0	–	–	–	○
	JRC	sign8	If C=1 then PC　PC+sign8+1 (sign8=-128~127)	If C=1 then PC　PC+sign16+1 (sign16=-32768~32767)*1	1	0	–	–	–	○,@rl
	JRNC	sign8	If C=0 then PC　PC+sign8+1 (sign8=-128~127)	If C=0 then PC　PC+sign16+1 (sign16=-32768~32767)*1	1	0	–	–	–	○,@rl
	JRZ	sign8	If Z=1 then PC　PC+sign8+1 (sign8=-128~127)	If Z=1 then PC　PC+sign16+1 (sign16=-32768~32767)*1	1	0	–	–	–	○,@rl
	JRNZ	sign8	If Z=0 then PC　PC+sign8+1 (sign8=-128~127)	If Z=0 then PC　PC+sign16+1 (sign16=-32768~32767)*1	1	0	–	–	–	○,@rl
	JP	%Y	PC　Y	–	1	0	–	–	–	–
	CALZ	imm8	([(SP1-1) 4+3]~[(SP1-1) 4])　PC+1, SP1　SP1-1, PC　imm8	–	1	0	–	–	–	○
	CALR	sign8	([(SP1-1) 4+3]~[(SP1-1) 4])　PC+1, SP1　SP1-1, PC　PC+sign8+1 (sign8=-128~127)	([(SP1-1) 4+3]~[(SP1-1) 4])　PC+1, SP1　SP1-1, PC　PC+sign16+1 (sign16=-32768~32767)*1	1	0	–	–	–	○,@rl
	CALR	[00addr6]	([(SP1-1) 4+3]~[(SP1-1) 4])　PC+1, SP1　SP1-1, PC　PC+[00addr6]+1	– –	2	0	–	–	–	○
	INT	imm6	[SP2-1]　F, SP2　SP2-1, ([(SP1-1) 4+3]~[(SP1-1) 4])　PC+1, SP1　SP1-1, PC　imm6 (imm6=0100H~013FH)	– – –	3	0	–	–	–	–
	RET		PC　([SP1 4+3]~[SP1 4]), SP1　SP1+1	–	1	0	–	–	–	–
	RETS		PC　([SP1 4+3]~[SP1 4]), SP1　SP1+1, PC　PC+1	–	2	0	–	–	–	–
	RETD	imm8	PC　([SP1 4+3]~[SP1 4]), SP1　SP1+1 [X]　i3~0, [X+1]　i7~4, X　X+2	– –	3	0	–	–	–	○,@h,@l
	RETI		PC　([SP1 4+3]~[SP1 4]), SP1　SP1+1 F　[SP2], SP2　SP2+1	– –	2					–
システム制御	HALT		Halt	–	2	0	–	–	–	–
	SLP		Sleep	–	2	0	–	–	–	–
	NOP		No operation (PC　PC+1)	–	1	0	–	–	–	–

注意

1: sign16(s15–s8) = imm8, sign16(s7–s0) = sign8



1. プログラミング
ワークベンチまたはエディタを使用してアセンブリソースファイルを作成します。

2. アセンブル、リンク
2-1) ワークベンチを起動します。
2-2) プロジェクトファイルを作成し、ソースファイルをプロジェクトに追加します。
2-3) ビルドを実行します。
ワークベンチは、アセンブラとリンカを順次実行し、実行形式のオブジェクトファイルを生成します。

3. オプションデータの作成 *
オプションデータ作成ツールによってオプションHEX/ドキュメントファイル(ファンクションオプション、セグメントオプション)を作成します。

4. デバッグ
4-1) ワークベンチからデバッガを起動します。
4-2) 実行形式のオブジェクトファイルおよびオプションHEXファイルをデバッガにロードし、デバッグコマンドを使用してデバッグを行います。

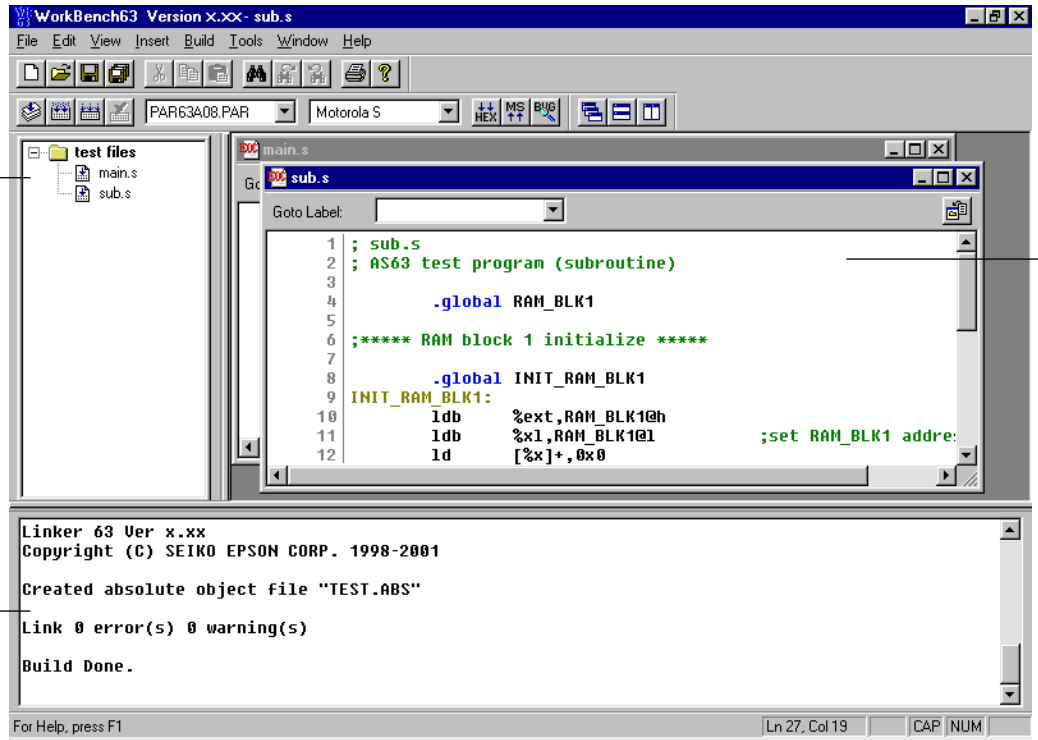
5. マスクデータの作成 *
プログラムの完成後、マスク化のためのマスクデータファイルを作成します。
5-1) HEXコンバータを使用して、プログラムHEXファイルを生成します。
5-2) マスクデータチェッカでプログラムHEXファイルおよびオプションドキュメントファイルをマスクデータファイルに変換します。
5-3) できあがったマスクデータファイルをセイコーエプソンに提出してください。

注:
図の中で"S1C63xxx Development Tool"として示された部分および で示した処理(3と5)は、S1C63アセンブラパッケージでは対応していない機種もあります。その場合は別途開発ツールが用意されています。
詳細については各機種のツールマニュアルを参照してください。

概要

Windows GUIベースのアプリケーションで、統合開発環境を提供します。ソースの作成や編集、ファイルの選択、各ツールの起動時オプションの選択と起動などが、Windowsの基本操作のみで可能となります。

ウィンドウ



[Project]ウィンドウ

現在開いているワークスペースフォルダとプロジェクト中のすべてのソースファイル名を、Windowsエクスプローラと同様に表示します。
リストされているソースファイルのアイコンをダブルクリックするとそのソースファイルが開き、内容が[Edit]ウィンドウに表示されます。

[Edit]ウィンドウ

ソースファイルの編集に使用します。ソース以外の標準テキストファイルも表示可能です。[Edit]ウィンドウ領域には複数の[Edit]ウィンドウを開くことができます。

[Output]ウィンドウ

ビルドやアセンブルの処理で実行されるツールが出力するメッセージを表示します。
このウィンドウに表示されるソース行番号を含む文法エラーなどのエラーメッセージをダブルクリックすると、対応するソースの[Edit]ウィンドウが開き、あるいはアクティブになり、エラーが発生したソース行を表示します。

ツールバー

標準ツールバー

[New]ボタン
ドキュメント(ソース、ヘッダ、プロジェクト)を新規作成します。

[Open]ボタン
ドキュメント(ソース、ヘッダ、プロジェクト)を開きます。

[Save]ボタン
アクティブな[Edit]ウィンドウ内のテキストをファイルにセーブ(上書き)します。

[Save All]ボタン
すべての[Edit]ウィンドウの内容とプロジェクト情報をそれぞれのファイルにセーブします。

[Cut]ボタン
[Edit]ウィンドウ内で選択されているテキストをクリップボードにカットします。

[Copy]ボタン
[Edit]ウィンドウ内で選択されているテキストをクリップボードにコピーします。

[Paste]ボタン
クリップボードのテキストを[Edit]ウィンドウにペーストします。

[Find]ボタン
アクティブな[Edit]ウィンドウ内で、指定の文字列を検索します。

[Find Next]ボタン
ファイルの終端方向に次の検索を行います。

[Find Previous]ボタン
ファイルの先頭方向に次の検索を行います。

[Print]ボタン
アクティブな[Edit]ウィンドウの内容を印刷します。

[Help]ボタン
ヘルプウィンドウを表示します。

ビルドツールバー

[Assemble]ボタン
アクティブな[Edit]ウィンドウのアセンブリソースをアセンブルします。

[Build]ボタン
現在開いているプロジェクトのmake処理を行い、実行形式オブジェクトをビルドします。

[Rebuild All]ボタン
現在開いているプロジェクトの再ビルドを行います。

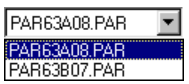
[Stop Build]ボタン
実行中のビルド処理を中止します。

ビルドツールバー

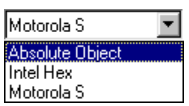
[HEX Convert]ボタン
HEXコンバータを起動します。

[Disassemble]ボタン
ディスアセンブラを起動します。

[Debug]ボタン
指定のICEパラメータファイルでデバッガを起動します。



[ICE Parameter]プルダウンリストボックス
開発機種のICEパラメータファイルを選択します。ここには、"Dev63"ディレクトリ内に存在するICEパラメータファイルがリストされます。



[Output Format]プルダウンリストボックス
ビルドで生成する実行形式オブジェクトのファイル形式を選択します。

ウィンドウツールバー

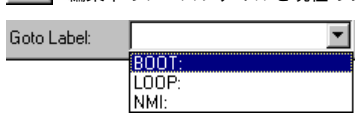
[Cascade]ボタン
開いている[Edit]ウィンドウを斜めに重ねて表示します。

[Tile Horizontally]ボタン
開いている[Edit]ウィンドウを縦に並べて表示します。

[Tile Vertically]ボタン
開いている[Edit]ウィンドウを横に並べて表示します。

[Edit]ウィンドウ上のコントロール

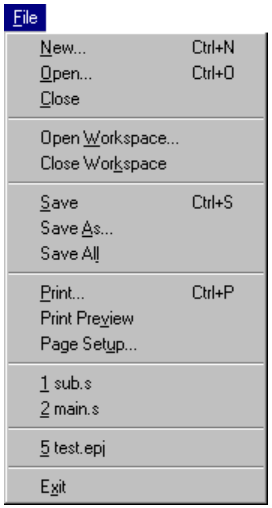
[Insert Into project]ボタン
編集中のソースファイルを現在のプロジェクトに追加します。



[Goto Label]プルダウンリストボックス
選択したラベル位置にジャンプします。

メニュー

[File]メニュー



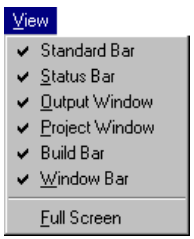
このメニューにリストされているファイル名は最近開いたソースとプロジェクトファイルです。ここからの選択でも、そのファイルを開くことができます。

New... ([Ctrl]+[N])
ドキュメント(ソース、ヘッダ、プロジェクト)を新規作成します。
Open... ([Ctrl]+[O])
ドキュメント(ソース、ヘッダ、プロジェクト)を開きます。
Close
アクティブな[Edit]ウィンドウを閉じます。
Open Workspace...
プロジェクトを開きます。
Close Workspace
現在のプロジェクトを閉じます。
Save ([Ctrl]+[S])
アクティブな[Edit]ウィンドウ内のテキストをセーブします。
Save As...
アクティブな[Edit]ウィンドウ内のテキストを他のファイル名でセーブします。
Save All
すべての[Edit]ウィンドウの内容とプロジェクト情報をそれぞれのファイルにセーブします。
Print... ([Ctrl]+[P])
アクティブな[Edit]ウィンドウの内容を印刷します。
Print Preview
アクティブな[Edit]ウィンドウの印刷イメージを表示します。
Page Setup...
用紙とプリンタを選択するダイアログボックスを表示します。
Exit
ワークベンチを終了します。

[Edit]メニュー

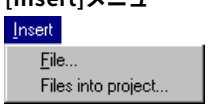
Undo ([Ctrl]+[Z])
[Edit]ウィンドウに対する直前の操作を取り消します。
Cut ([Ctrl]+[X])
[Edit]ウィンドウ内で選択されているテキストをカットします。
Copy ([Ctrl]+[C])
[Edit]ウィンドウ内で選択されているテキストをコピーします。
Paste ([Ctrl]+[V])
クリップボードのテキストを[Edit]ウィンドウにペーストします。
Select All ([Ctrl]+[A])
アクティブな[Edit]ウィンドウ内のテキストをすべて選択します。
Find... ([Ctrl]+[F])
アクティブな[Edit]ウィンドウ内で、指定の文字列を検索します。
Replace ([Ctrl]+[H])
アクティブな[Edit]ウィンドウ内の指定文字列を置き換えます。
Go To ([Ctrl]+[G])
指定の行番号またはラベル位置にジャンプします。

[View]メニュー



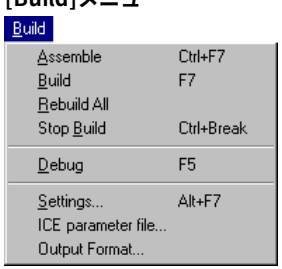
Standard Bar
標準ツールバーの表示/非表示を切り替えます。
Status Bar
ステータスバーの表示/非表示を切り替えます。
Output Window
[Output]ウィンドウ表示/非表示を切り替えます。
Project Window
[Project]ウィンドウの表示/非表示を切り替えます。
Build Bar
ビルドツールバーの表示/非表示を切り替えます。
Window Bar
ウィンドウツールバーの表示/非表示を切り替えます。
Full Screen
[Edit]ウィンドウ領域を全画面サイズに拡大します。

[Insert]メニュー



File...
指定のファイルを[Edit]ウィンドウに挿入します。
Files into project...
指定のソースファイルを現在のプロジェクトに追加します。

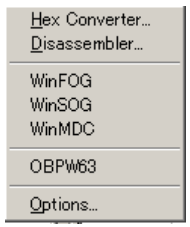
[Build]メニュー



Assemble ([Ctrl]+[F7])
アクティブな[Edit]ウィンドウのソースをアセンブルします。
Build ([F7])
現在のプロジェクトをmake処理によりビルドします。
Rebuild All
現在開いているプロジェクトの再ビルドを行います。
Stop Build ([Ctrl]+[Break])
実行中のビルド処理を中止します。
Debug ([F5])
指定のICEパラメータファイルでデバッガを起動します。
Settings... ([Alt]+[F7])
ツールオプションを設定するダイアログボックスを表示します。
ICE parameter file...
ICEパラメータファイル選択用ダイアログボックスを表示します。
Output Format...
出力形式を選択するダイアログボックスを表示します。

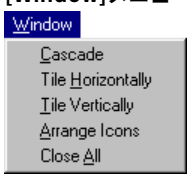
メニュー

[Tools]メニュー



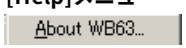
HEX Converter...
HEXコンバータを起動します。
Disassembler...
ディスアセンブラを起動します。
WinFOG
ファンクションオプションジェネレータを起動します。
WinSOG
セグメントオプションジェネレータを起動します。
WinMDC
マスクデータチェッカを起動します。
OBPW63
On Board Writerコントロールソフトウェアを起動します。
Options...
ワークベンチのオプションを設定します。

[Window]メニュー



このメニューは[Edit]ウィンドウを開くと表示されます。
Cascade
開いている[Edit]ウィンドウを斜めに重ねて表示します。
Tile Horizontally
開いている[Edit]ウィンドウを縦に並べて表示します。
Tile Vertically
開いている[Edit]ウィンドウを横に並べて表示します。
Arrange Icons
最小化された[Edit]ウィンドウアイコン整列させます。
Close All
開いている[Edit]ウィンドウをすべて閉じます。

[Help]メニュー



About WB63...
ワークベンチのバージョン情報を表示します。

ショートカットキー一覧

Ctrl + N	ドキュメントの新規作成
Ctrl + O	ドキュメントのオープン
Ctrl + F12	ドキュメントのオープン
Ctrl + S	ファイルへのセーブ
Ctrl + P	アクティブなドキュメントの印刷
Ctrl + Shift + F12	アクティブなドキュメントの印刷
Ctrl + Z	直前の操作の取り消し
Alt + BackSpace	直前の操作の取り消し
Ctrl + X	選択範囲のカット
Shift + Delete	選択範囲のカット
Ctrl + C	選択範囲のコピー
Ctrl + Insert	選択範囲のコピー
Ctrl + V	クリップボードからのペースト
Shift + Insert	クリップボードからのペースト
Ctrl + A	ドキュメント内のテキストをすべて選択
Ctrl + F	文字列の検索
F3	次を検索
Shift + F3	前を検索
Ctrl + H	文字列の置き換え
Ctrl + G	ソース行、ラベルへのジャンプ
Ctrl + F7	ソースのアセンブル
F7	プロジェクトのビルド
Ctrl + Break	ビルドの中止
F5	デバッガの起動
Alt + F7	ツールオプションの設定
Ctrl + Tab	アクティブウィンドウの切り替え
Short-cut-key	ポップアップメニューの表示
Shift + F10	ポップアップメニューの表示

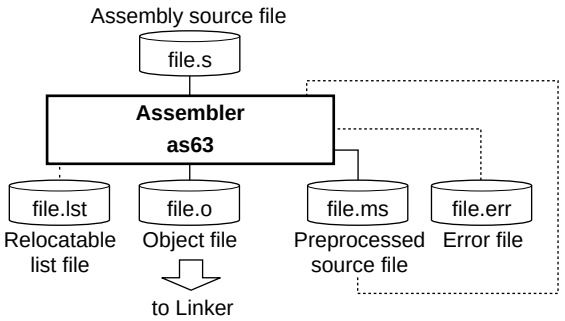
エラーメッセージ

<filename> is changed by another editor. Reopen this file ?	現在開いているファイルが他のエディタで変更されました。
Cannot create file : <filename>	ファイル(リンカコマンドファイル、デバッグコマンドファイル等)が作成できません。
Cannot find file : <filename>	ソースファイルが見つかりません。
Cannot find ICE parameter file	ICEパラメータファイルが見つかりません。
Cannot open file : <filename>	ソースファイルが開けません。
You cannot close workspace while a build is in progress. Select the Stop Build command before closing.	ビルド中にプロジェクトを閉じるまたはワークベンチを終了するコマンドが指定されました。
Would you like to build it ?	ビルド前にデバッガを起動しようとしました。

概要

アセンブラas63はアセンブリソースファイルをアセンブルし、ソースファイルのニーモニックをS1C63000のオブジェクト（機械語）コードに変換します。結果はリロケータブルオブジェクトファイルとして出力されます。また、マクロや、条件アセンブル、インクルードなど、アセンブルの前処理のための付加機能も含まれています。

フローチャート



起動コマンド

```
Usage: as63 [options] <file name>
Options: -d <symbol>    Add preprocess definition
         -e             Output error log file (.ERR)
         -g             Add source debug information in object
         -l             Output relocatable list file (.LST)
         -c             Ignore character case of symbols
         -o <file name> Specify output file name
File name: Source file name (.S or .MS)
```

擬似命令

#include	<file name>	ファイルの挿入
#define	<define name> [<string>]	指定の名称で文字列を定義
#defnum	<defnum name> <value>	指定の名称で数値を定義
#macro	<macro name> [par] [,par] ...	指定のマクロ名でステートメント列を定義
	<statements>	(par: ダミーパラメータ)
#endm		
#ifdef	<name>	条件アセンブル
	<statements 1>	<name>が定義済み: <statements 1>をアセンブル
[#else		<name>が未定義: <statements 2>をアセンブル
	<statements 2>]	
#endif		
#ifndef	<name>	条件アセンブル
	<statements 1>	<name>が未定義: <statements 1>をアセンブル
[#else		<name>が定義済み: <statements 2>をアセンブル
	<statements 2>]	
#endif		
.code		CODEセクションの開始を宣言
.data		DATAセクションの開始を宣言
.bss		BSSセクションの開始を宣言
.abs		アブソリュートアセンブルを指定
.org	<address>	絶対アドレスを設定
.align	<alignment number>	セクションのアライメントを設定
.comm	<global symbol> <size>	BSSセクション内にグローバルシンボルを定義し領域を確保
.lcomm	<local symbol> <size>	BSSセクション内にローカルシンボルを定義し領域を確保
.set	<symbol> <address>	シンボルに絶対アドレスを定義
.global	<symbol>	シンボルのグローバル宣言
.codeword	<data>[<data> ... <data>]	CODEセクション内にデータを定義
.word	<data>[<data> ... <data>]	DATAセクション内にデータを定義
.list		アセンブリリスト出力をON(.list)/OFF(.nolist)
.nolist		(-lオプション指定時のみ有効)
.stabs	"<file name>", FileName	ソースデバッグ情報の出力
.stabn	0, FileEnd	(-gオプション指定時のみ出力可)
.stabn	<line number>, LineInfo	

演算子

		優先順位
()	カッコ	1
+	プラス符号	2
-	マイナス符号	2
~	ビット反転	2
^H	上位8ビットを取得	3
^L	下位8ビットを取得	3
*	乗算	4
/	除算	4
% (%%)	剰余	4
+	加算	5
-	減算	5
<<	左シフト	6
>>	右シフト	6
==	等しい(関係演算子)	7
!=	等しくない(関係演算子)	7
<	小さい(関係演算子)	7
<=	小さいか等しい(関係演算子)	7
>	大きい(関係演算子)	7
>=	大きいか等しい(関係演算子)	7
&	ビットAND	8
^	ビットXOR	9
|	ビットOR	10
&&	AND (関係演算子)	11
||	OR (関係演算子)	12

式の項には数値とシンボルが使用可能です。
式は符号付き16ビットデータとして計算されます。
演算子と数値間にはスペースやTABを挿入しないでください。

エラーメッセージ

Address out of range	アドレスが指定可能範囲外です。
Cannot read <file kind> file <FILE NAME>	指定ファイルが読み込めません。
Cannot read <file kind> file <FILE NAME>	指定ファイルが読み込めません。
Cannot write <file kind> file <FILE NAME>	指定ファイルに書き込めません。
Directory path length limit <directory path length limit> exceeded	パスの長さが制限を越えています。
Division by zero	0で除算が行われました。
File name length limit <file name length limit> exceeded	ファイル名の長さが制限を越えています。
Illegal macro label <label>	マクロ定義内の内部分岐ラベルの記述が不正です。
Illegal macro parameter <parameter>	マクロパラメータの記述が不正です。
Illegal syntax	不正な文法で記述されています。
Line length limit <line length limit> exceeded	1行の文字数が制限を越えています。
Macro parameter range <macro parameter range> exceeded	マクロパラメータ数が制限を越えています。
Memory mapping conflict	アドレスが重複しています。
Multiple statements on the same line	1行に2つ以上のステートメントが記述されています。
Nesting level limit <nesting level limit> exceeded	#includeのネストが制限を越えました。
Number of macro labels limit <number of macro label limit> exceeded	マクロ内の内部分岐ラベル数が制限を越えました。
Out of memory	メモリを確保できません。
Second definition of label <label>	ラベルが二重定義されました。
Second definition of symbol <symbol>	シンボルが二重定義されました。
Symbol name length limit <symbol name length limit> exceeded	シンボル名の長さが制限を越えています。
Token length limit <token length limit> exceeded	トークンの長さが制限を越えています。
Unexpected character <name>	無効な文字を使用しています。
Unknown label <label>	未定義のラベルを参照しています。
Unknown mnemonic <name>	無効な命令が記述されています。
Unknown symbol <name>	未定義のシンボルを参照しています。
Unknown register <name>	無効なレジスタ名が記述されています。
Unknown symbol mask <name>	無効なシンボルマスクが記述されています。
Unsupported directive <directive>	無効な擬似命令が記述されています。

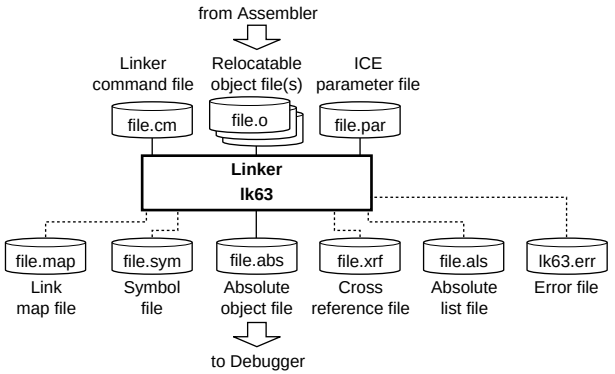
ワーニングメッセージ

Expression out of range	式の計算結果が有効範囲を越えました。
Invalid symbol mask	シンボルマスクが正しく定義されていません。
Second definition of define symbol <symbol>	#defineでシンボルが二重定義されました。
Section activation expected, use <.code/.bss>	セクション定義がありません。
Missing Delimiter	オペランド間のカンマが省略されています。

概要

アセンブラによって生成したリロケータブルオブジェクトのメモリロケーションを決定し、実行可能なアブソリュートオブジェクトコードを生成します。また、分岐命令の拡張コードを自動挿入、削除、修正する分岐命令最適化機能も提供します。

フローチャート



起動コマンド

```
  Options: -d                          Disable all branch optimizations
           -di                         Disable insertion of branch extension
           -dr                         Disable removal branch optimization
           -e                          Output error log file (.ERR)
           -g                          Add source debug information
           -l                          Output absolute list file (.ALS)
           -m                          Output map file (.MAP)
           -o <file name>              Specify output file name
           -s                          Output symbol file (.SYM)
           -x                          Output cross reference file (.XRF)
           -code <address>             Specify CODE start address
           -data <address>             Specify DATA start address
           -bss <address>              Specify BSS start address
           -rcode <file name>=<address> Specify CODE start address of the file
           -rdata <file name>=<address> Specify DATA start address of the file
           -rbss <file name>=<address>  Specify BSS  start address of the file
           -defsym <symbol>=<address>   Define symbol address
  File names: Relocatable object file (.O)
              Command parameter file (.CM)
              ICE parameter file (.PAR)
```

エラーメッセージ

Branch destination too far from <address>	分岐先が分岐可能範囲外です。
CALZ for non zero page at <address>	指定アドレスが0x0000~0x00ffの範囲外です。
Cannot create absolute object file <FILE NAME>	アブソリュートオブジェクトファイルが作成できません。
Cannot open <file kind> file <FILE NAME>	ファイルが開けません。
Cannot read <file kind> file <FILE NAME>	ファイルが読み込めません。
Cannot relocate <section kind> section of <FILE NAME>	リロケータブルセクションが配置できません。
Cannot write <file kind> file <FILE NAME>	ファイルへの書き込みができません。
Illegal address range <address> for a code at <address>	TST/SET/CLRで指定されたアドレスが0x0000~0x003fまたは0xffC0~0xffffの範囲外です。
Illegal file name <FILE NAME>	ファイル名が不正です。
Illegal file name <FILE NAME> specified with option <option>	オプションで指定されているファイル名が不正です。
Illegal ICE parameter at line <line number> of <FILE NAME>	ICEパラメータファイルに不正なパラメータが記述されています。
Illegal object <FILE NAME>	入力ファイルがIEEE-695形式のアブソリュートオブジェクトファイルではありません。
Illegal option <option>	不正なオプションが指定されました。
No address specified with option <option>	オプションにアドレスの指定がありません。
No code to locate	マップする有効なコードがありません。
No ICE parameter file specified	ICEパラメータファイルが指定されていません。
No name and address specified with option <option>	オプションに名前とアドレスの指定がありません。
No object file specified	リンクするオブジェクトファイルが指定されていません。
Out of memory	メモリが確保できません。
<section kind> section <address>-<address> overlaps with <section kind> section <address>-<address>	セクションのアドレス範囲が他のセクションのアドレス範囲に重なっています。
<section kind> section <address>-<address> overlaps with the unavailable memory	セクションのアドレス範囲がメモリの未使用領域にかかっています。
Unresolved external <label> in <FILE NAME>	未定義シンボルを参照しています。
Unusable instruction code <instruction code> in <FILE NAME>	オブジェクトファイルが開発機種が未対応の命令を含んでいます。

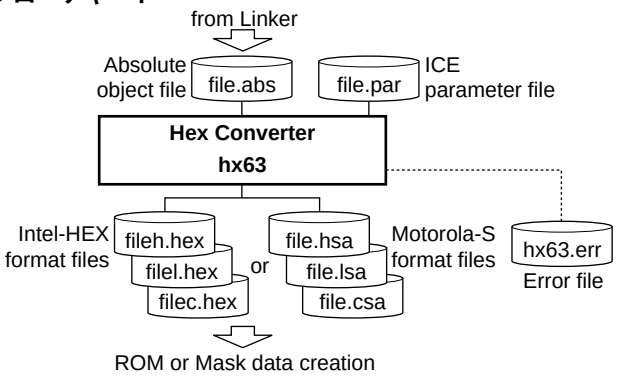
ワーニングメッセージ

Cannot create <file kind> file <FILE NAME>	ファイルが作成できません。
Cannot open <file kind> file <FILE NAME>	ファイルが開けません。
No debug information in <FILE NAME>	入力ファイルにデバッグ情報が含まれていません。
No symbols found	シンボルが見つかりません。
Second definition of label <label> in <FILE NAME>	指定のラベルは既に定義されています。
Second ICE parameter file <FILE NAME> ignored	2つ以上のICEパラメータファイルが指定されています。

概要

リンカが出力するIEEE-695形式のアブソリュートオブジェクトファイルをインテルHEX形式あるいはモトローラS形式のROMイメージデータに変換します。この変換は、デバッグ用のプログラムROMを作成する場合や、各機種の"Development Tool"でマスクデータを作成する場合に必要です。

フローチャート



起動コマンド

```
Usage: hx63 [options] <file names>
Options: -b             Do not fill unused memory with 0xff
         -e             Output error log file (HX63.ERR)
         -i             Use Intel Hex format
         -o <file name> Specify output file name
File names: Absolute object file (.ABS)
            ICE parameter file (.PAR)
```

エラーメッセージ

Cannot create <file kind> file <FILE NAME>	ファイルが作成できません。
Cannot open <file kind> file <FILE NAME>	ファイルが開けません。
Cannot read <file kind> file <FILE NAME>	ファイルが読み込めません。
Cannot write <file kind> file <FILE NAME>	ファイルへ書き込めません。
Illegal file name <FILE NAME> specified with option <option>	指定のHEXファイル名が不正です。
Illegal ICE parameter at line <line number> of <FILE NAME>	ICEパラメータファイルに不正なパラメータが含まれています。
Illegal file name <FILE NAME>	入力ファイル名が不正です。
Illegal option <option>	無効なオプションが指定されました。
Illegal absolute object format	入力ファイルがIEEE-695形式のオブジェクトではありません。
No ICE parameter file specified	ICEパラメータファイルが指定されていません。
Out of memory	メモリが確保できません。

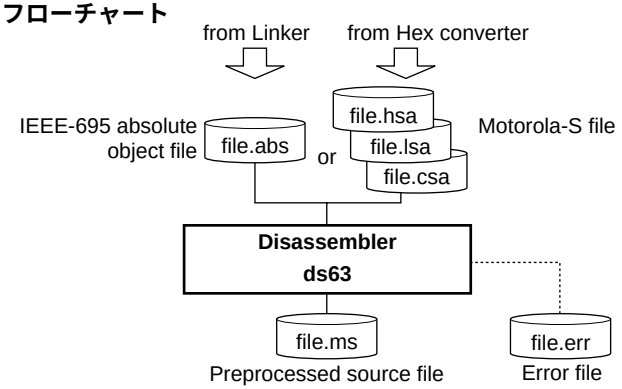
ワーニングメッセージ

Input file name extension .XXX conflict	同じ拡張子で2つ以上のファイル名が指定されました。 最後に指定されたファイルを使用します。

概要

IEEE-695形式またはモトローラS形式のアブソリュートオブジェクトを逆アセンブルし、ソースファイルに復元します。復元したソースファイルは、アセンブラ、リンカ、HEXコンバータでオリジナルのソースと同様に処理可能で、復元前と同じアブソリュートオブジェクトまたはHEXファイルが生成されます。

フローチャート



起動コマンド

```
Usage: ds63 [options] <file name>
Options: -cl            Use lower case characters
         -cu            Use upper case characters
         -e             Output error log file (DS63.ERR)
         -o <file name> Specify output file name
File names: Absolute object file (.ABS or .CSA/.LSA/.HSA)
```

エラーメッセージ

Cannot create <file kind> file <FILE NAME>	ファイルが作成できません。
Cannot open <file kind> file <FILE NAME>	ファイルが開けません。
Cannot read <file kind> file <FILE NAME>	ファイルが読み込めません。
Cannot write <file kind> file <FILE NAME>	ファイルへ書き込めません。
Illegal file name <FILE NAME> specified with option <option>	指定の出力ソースファイル名が不正です。
Illegal file name <FILE NAME>	入力ファイル名が不正です。
Illegal HEX data format	入力ファイルがモトローラS形式のファイルではありません。
Illegal option <option>	無効なオプションが指定されました。
Out of memory	メモリが確保できません。

ワーニングメッセージ

Input file name extension .XXX conflict	同じ拡張子で2つ以上のファイル名が指定されました。 最後に指定されたファイルを使用します。
Cannot open Hex file xxx.csa	現在の機種はデータROMを搭載していないため、".csa"ファイルを開くことはできません。

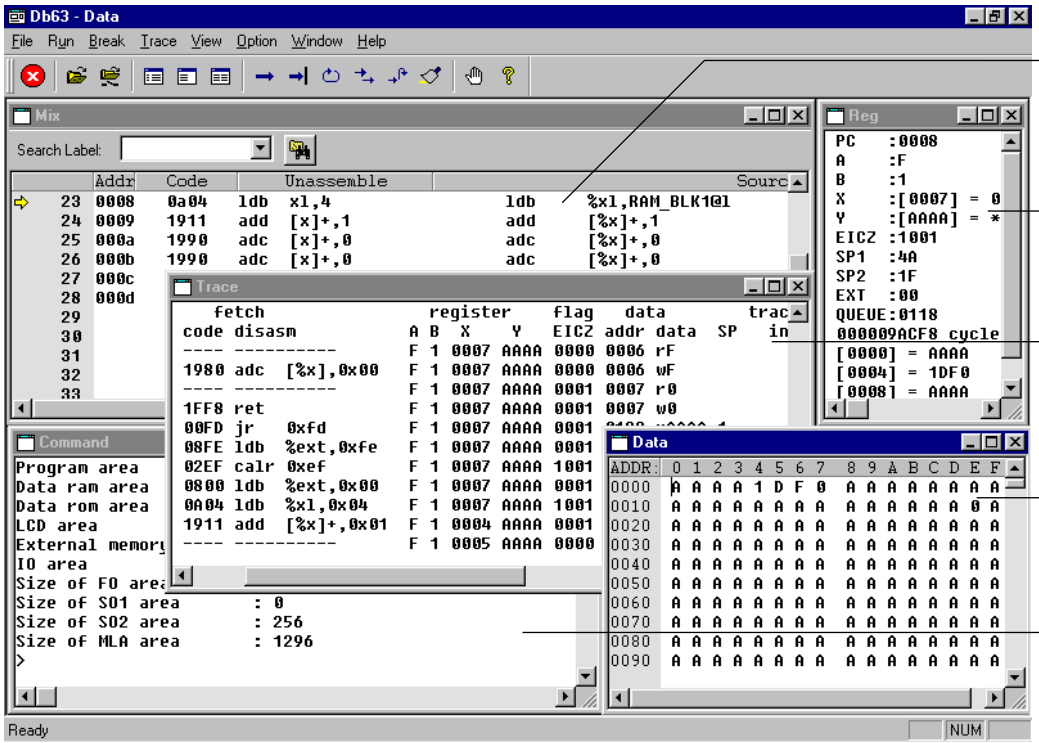
概要

ハードウェアツールとして用意されているICEを制御してデバッグを行うソフトウェアです。ブレークやステップ実行など、頻繁に使用するコマンドはツールバーに登録されており、キーボード操作の量を抑えています。また、ソースやレジスタ内容、コマンド実行結果がマルチウィンドウ上に表示できるため、デバッグ作業が効率良く行えます。

起動コマンド

```
 - Usage -
      db63.exe <parameter_file> <startup options>
 Options:
      command_file:  ... specifies a command file
      -comX(X:1-4)   ... com port, default com1
      -b             ... baud rate, 2400,9600(default),19200,38400
      -usb           ... specifies usb communication
      -sim           ... specifies simulator mode
```

ウィンドウ



[Source]ウィンドウ
プログラムを逆アセンブルコード、ソースコードまたは両方をミックスした形で表示します。

[Register]ウィンドウ
レジスタ値およびモニタデータを表示します。

[Trace]ウィンドウ
トレースデータを表示します。

[Data]ウィンドウ
データメモリの内容を表示します。

[Command]ウィンドウ
デバッグコマンドの入力、コマンド実行結果の表示に使用します。

ボタン

ツールバー

[Key Break]ボタン
ターゲットプログラムの実行を強制的にブレークします。

[Load File]ボタン
IEEE-695形式のオブジェクトファイルを読み込みます。

[Load Option]ボタン
モトローラS形式のプログラム、オプションHEXファイルを読み込みます。

[Source]ボタン
[Source]ウィンドウをソース表示モードに切り換えます。

[Unassemble]ボタン
[Source]ウィンドウを逆アセンブル表示モードに切り換えます。

[Mix]ボタン
[Source]ウィンドウの表示をミックス表示モードに切り換えます。

[Go]ボタン
現在のPCからターゲットプログラムを実行します。

[Go to Cursor]ボタン
現在のPCから、[Source]ウィンドウのカーソル位置(その行のアドレス)まで
ターゲットプログラムを実行します。

[Go from Reset]ボタン
CPUをリセット後、プログラム開始アドレス(0x110)からターゲットプログラムを実行します。

[Step]ボタン
現在のPCからターゲットプログラムを1ステップ実行します。

[Next]ボタン
現在のPCからターゲットプログラムを1ステップ実行します。実行命令がcalr、calz、
int命令の場合は、その命令とサブルーチンを1ステップとして実行します。

[Reset]ボタン
CPUをリセットします。

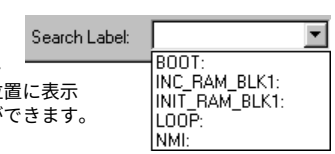
[Break]ボタン
[Source]ウィンドウ上のカーソル位置でブレークポイントの設定と解除を行います。

[Help]ボタン
ヘルプウィンドウが開き、ヘルプトピックを表示します。

[Source]ウィンドウ上のコントロール

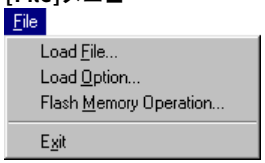
[Find]ボタン
任意の文字列をソース内で検索し、その位置にジャンプします。

[Search Label]
プルダウンリスト
選択したラベル位置に表示を移動することができます。



メニュー

[File]メニュー

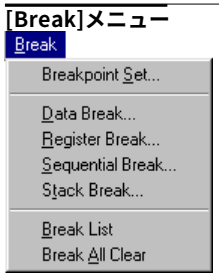


Load File...
IEEE-695形式のアブソリュートオブジェクトファイルを読み込みます。
Load Option...
モトローラS形式のプログラム、オプションHEXファイルを読み込みます。
Flash Memory Operation... (USB版ICE接続時は使用不可)
Flashメモリに対するデータのロード/セーブ、イレースを行います。
Exit
デバッガを終了します。

[Run]メニュー

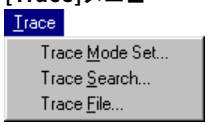
Go
現在のPCからターゲットプログラムを実行します。
Go to Cursor
現在のPCから、[Source]ウィンドウのカーソル位置(その行のアドレス)までターゲットプログラムを実行します。
Go from Reset
CPUをリセット後、プログラム開始アドレス(0x110)からターゲットプログラムを実行します。
Step
現在のPCからターゲットプログラムを1ステップ実行します。
Next
現在のPCからターゲットプログラムを1ステップ実行します。calr、calz、int命令は、その命令とサブルーチンを1ステップとして実行します。
Command File...
コマンドファイルを読み込み、記述されているコマンドを実行します。
Reset CPU
CPUをリセットします。

[Break]メニュー



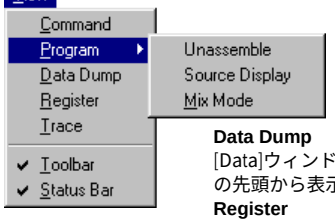
Breakpoint Set...
PCブレークポイントを表示、設定、解除します。
Data Break...
データブレーク条件を表示、設定、解除します。
Register Break...
レジスタブレーク条件を表示、設定、解除します。
Sequential Break...
シーケンシャルブレーク条件を表示、設定、解除します。
Stack Break...
スタックブレーク用にスタック領域を設定します。
Break List
設定されているすべてのブレーク条件を表示します。
Break All Clear
すべてのブレーク条件を解除します。

[Trace]メニュー



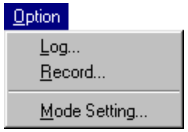
Trace Mode Set...
トレースモードとトレース条件を設定します。
Trace Search...
トレースメモリ内のトレース情報を検索します。
Trace File...
[Trace]ウィンドウに表示したトレース情報の指定範囲をファイルに保存します。

[View]メニュー



Command
[Command]ウィンドウをアクティブにします。
Program (Unassemble, Source Display, Mix Mode)
[Source]ウィンドウを開いてアクティブにし、プログラムを現在のPCアドレスから、サブメニューで選択した表示モードで表示します。
Data Dump
[Data]ウィンドウを開いてアクティブにし、データメモリの内容をメモリの先頭から表示します。
Register
[Register]ウィンドウを開いてアクティブにし、各レジスタの内容を表示します。
Trace
[Trace]ウィンドウを開いてアクティブにし、ICEトレースメモリの内容を表示します。
Toolbar
ツールバーの表示/非表示を切り換えます。
Status Bar
ステータスバーの表示/非表示を切り換えます。

[Option]メニュー



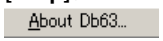
Log...
ログ出力のON/OFFを切り換えます。
Record...
実行コマンドのファイルへの記録を制御します。
Mode Setting...
デバッガのモードを設定します。

[Window]メニュー

Cascade
開いているウィンドウを斜めに整列させます。
Tile
開いているウィンドウを縦横に整列させます。

このメニューには、現在開いているウィンドウ名が表示されます。
いずれかを選択すると、そのウィンドウがアクティブになります。

[Help]メニュー



About Db63...
デバッガのアバウトダイアログボックスを表示します。

デバッグコマンド

プログラムメモリ操作

a (as) [<addr> <mnemonic> [<file name>]]	インラインアセンブル
pe [<addr> <code1> [..<code8>]]	プログラムコード入力
pf [<addr1> <addr2> <code>]	プログラム領域のフィル
pm [<addr1> <addr2> <addr3>]	プログラム領域のコピー

データメモリ操作

dd [<addr1> [<addr2>]]	データメモリダンプ
de [<addr> <data1> [..<data16>]]	データ領域のフィル
df [<addr1> <addr2> <data>]	データ領域のフィル
dm [<addr1> <addr2> <addr3>]	データ領域のコピー
dw [<addr1> [..<addr4>]]	監視データアドレス設定

レジスタ操作

od [{fog|sog|mla} [<addr1> [<addr2>]]]	オプションデータダンプ

プログラム実行

rd	レジスタ表示
rs [<reg> <value> [..<reg> <value>]]	レジスタ変更 reg={pc|a|b|x|y|f|sp1|sp2|ext|q}

Program execution

g [<addr1> [<addr2]]	連続実行
gr [<addr1> [<addr2]]	リセット&連続実行
s [<step>]	ステップ実行
n [<step>]	スキップ付きステップ実行

CPUリセット

rst	リセット

ブレーク

bp [<addr1> [..<addr16>]]	ブレークポイントの設定
bc (bpc) [<addr1> [..<addr16>]]	ブレークポイントの解除
bd [<data> {r|w| } <addr1> <addr2>]	データブレークの設定
bdc	データブレークの解除
br [<reg> <value> [..<reg> <value>]]	レジスタブレークの設定 reg={pc|a|b|x|y|f|sp1|sp2|ext|q}
brc	レジスタブレークの解除
bs [<pass> <addr1> [<addr2> [<addr3>]]]	シーケンシャルブレークの設定
bsc	シーケンシャルブレークの解除
bsp [<addr1> <addr2> <addr3> <addr4>]	スタック領域指定
bl	全ブレーク条件の表示
bac	全ブレーク条件の解除

プログラム表示

u [<addr>]	逆アセンブル表示
sc [<addr>]	ソース表示
m [<addr>]	ミックス表示

シンボル情報

sy [{$<keyword>|#<keyword>}] [/a]	シンボル一覧

ファイルロード

lf [<file name>]	IEEE-695形式オブジェクトファイル読み込み
lo [<file name>]	モトローラS形式ファイル読み込み

フラッシュメモリ/FPGA操作

lfl [{p|d|f|s|m} [..{p|d|f|s|m}]]	フラッシュメモリの読み出し(USB接続時は使用不可)
sfl [{p|d|f|s|m} [..{p|d|f|s|m}]] [-p]	フラッシュメモリへの書き込み(USB接続時は使用不可)
efl	フラッシュメモリ消去(USB接続時は使用不可)
xfer(s)	FPGA消去
xfwr(s) <file name> ;{H|S} [;N]	FPGAデータ書き込み
xfcp(s) <file name> ;{H|S}	FPGAデータコンペア
xdp(s) <addr1> [<addr2>]	FPGAデータダンプ

トレース

tm [{-n|-s|-a} <trigger> [{a|m|e}{i|o}] [<addr1> <addr2> [..<addr7> <addr8>]]	トレースモードの設定
td [<cycle>]	トレース情報の表示
ts [{pc|dr|dw} <addr>]	トレース情報の検索
tf [[<cycle1> [<cycle2>]] <file name>]	トレース情報の保存

その他

cv [<addr1> [<addr2>]]	カバレッジ情報の表示
cvc	カバレッジ情報のクリア
com [<file name> [<interval>]]	コマンドファイル読み込み/実行
cmw [<file name>]	実行間隔指定付きコマンドファイル読み込み/実行
rec [<file name>]	実行コマンドの記録
log [<file name>]	ログ出力ON/OFF
ma	マップ情報の表示
md [<option> <num> [..<option> <num>]]	モード設定 option={-f|-u|-i|-s|-c|-il|-cm}
q	デバッガ終了
?	コマンドusageの表示

アドレスの指定には、次のようにシンボルを使用可能です。
@<グローバルシンボル> または @<ローカルシンボル>@<ソースファイル名>

デバッガメッセージ

ICEステータス

Break by PC break	ブレークポイントによりブレーク
Break by data break	データブレーク条件によりブレーク
Break by register break	レジスタブレーク条件によりブレーク
Break by sequential break	シーケンシャルブレーク条件によりブレーク
Key Break	ボタンによりブレーク
Break by accessing no map program area	未定義プログラムメモリ領域のアクセスによりブレーク
Break by accessing no map data area	未定義データメモリ領域のアクセスによりブレーク
Break by accessing ROM area	データROM領域への書き込みによりブレーク
Out of SP1 area	スタック領域外に対するスタック操作によりブレーク
Out of SP2 area	スタック領域外に対するスタック操作によりブレーク
Break by external break	ICEのBRKIN端子への信号入力によりブレーク

ICEエラー

communication error	タイムアウト以外の通信エラー （オーバーラン、フレーミング、BCCエラー）
CPU is running	CPUが実行中
ICE is busy	ICEが処理中
ICE is free run mode	ICEがフリーランモード
ICE is maintenance mode	ICEが保守モード
no map area, XXXX	マップ領域外をアクセス
not defined ID, XXXX	ICEの応答IDが無効
on tracing	実行データのトレース中
reset time out	CPUがリセットできない（1秒以上）
target down	ペリフェラルボードが正常に動作していない、 もしくはリセットが解除されない
Time Out!	通信タイムアウト

フラッシュメモリエラー

flash memory error, XXXX	フラッシュメモリの書き込みまたは消去エラー
flash ROM is protected	フラッシュメモリにプロテクトがかかっている
format error	フラッシュメモリがマップされていない
Map information is not the same	パラメータファイルからのマップ情報とフラッシュメモリの マップ情報に違いがある
verify error, XXXX	フラッシュメモリ書き込み時のベリファイエラー

コマンドエラー

Address out of range, use 0–0xXXXX	指定アドレスがプログラムメモリの有効範囲外
Address out of range, use 0–0xFFFF	指定アドレスがデータメモリの有効範囲外
Cannot load program/ROM data, check ABS file	プログラム/ROMデータのロードに失敗 IEEE-695実行形式以外のファイルが指定された
Cannot open file	ファイルがオープンできない
Data out of range, use 0–0xF	指定の数値はデータの有効範囲外
Different chip type, cannot load this file	指定ファイルは異なるICEパラメータで作成されている
end address < start address	開始アドレスより小さい終了アドレスが指定された
error file type (extension should be CMD)	".cmd"以外の拡張子を持つコマンドファイルが指定された
FO address out of range, use 0–0xEF	アドレスが有効範囲外
illegal code	入力コードが不正
illegal mnemonic	S1C63000用の入力ニーモニックが不正
Incorrect number of parameters	パラメータ数が不正
Incorrect option, use -f/-u/-i/-s/-c/-il/-cm	無効なモード設定オプションが指定された
Incorrect r/w option, use r/w/*	不正なR/Wオプションが指定された
Incorrect register name, use A/B/X/Y/F	無効なレジスタ名が指定された
Incorrect register name, use PC/A/B/X/Y/F/SP1/SP2/EXT/Q	指定されたレジスタ名が無効
Input address does not exist	未設定ブレークアドレスを解除しようとした
invalid command	無効なコマンドが入力された
invalid data pattern	入力データパターンが不正
invalid file name	オプションファイルの拡張子が無効
invalid value	入力データ、アドレス、シンボルが不正
Maximum nesting level(5) is exceeded, cannot open file	コマンドのネストレベルが制限を越えた
MLA address out of range, use 0–0xFFF	MLAアドレスが有効範囲外
no such symbol	該当するシンボルがない
no symbol information	".ABS" ファイルがロードされていないためシンボルは使用不可
Number of passes out of range, use 0–4095	シーケンシャルブレークの実行回数指定が有効範囲外
Number of steps out of range, use 0–65535	指定ステップ数が有効範囲外
SO address out of range, use 0–0x1FFF	アドレスが有効範囲外
SP1 address out of range, use 0–0x3FF	指定SP1アドレスが有効範囲外
SP2 address out of range, use 0–0xFF	指定SP2アドレスが有効範囲外
symbol type error	指定シンボルタイプ（プログラム/データ）が不正

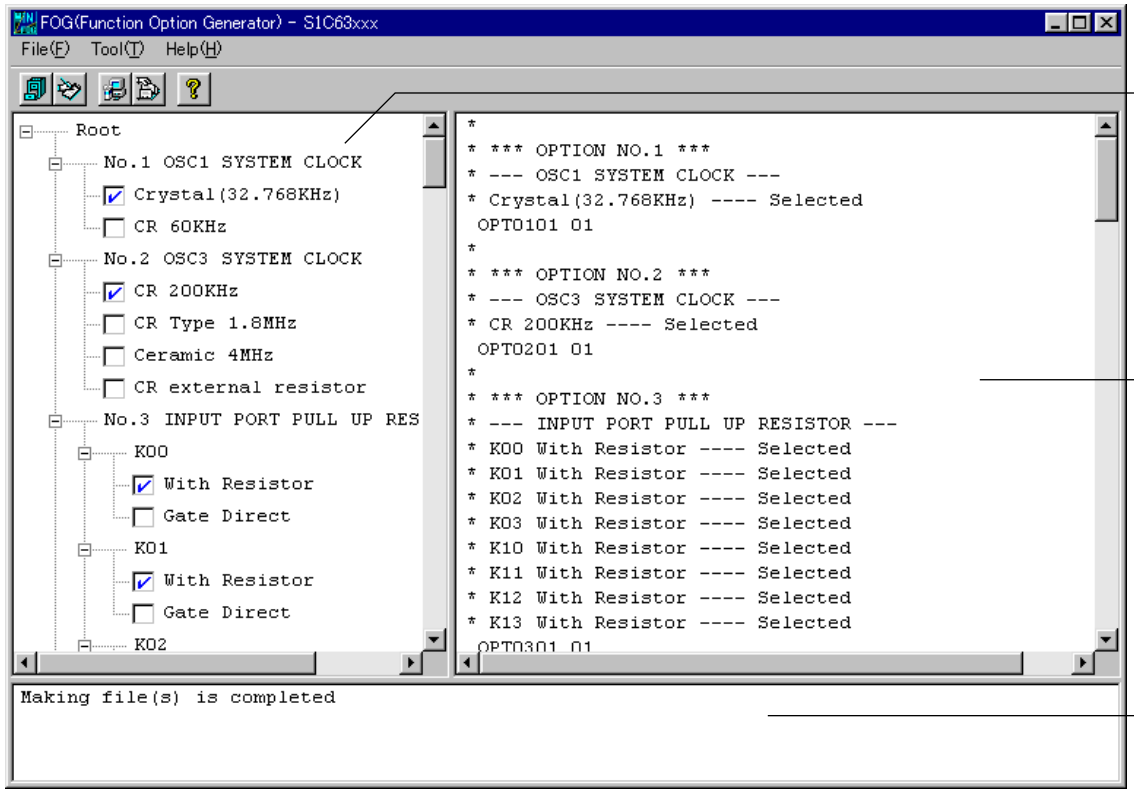
コマンドワーニング

Break address already exists	設定済みのブレークアドレスを再設定
Identical break address input	ブレークアドレスの二重設定
round down to multiple of 4	監視データアドレスが不正
User cancel	FPGAコマンドをユーザが中断
Verify error	FPGAのコンペアエラー

概要

ファンクションオプションジェネレータwinfogは、I/Oポート機能など、いくつかのハードウェア仕様のマスクパターン生成のためのファイルを作成するソフトウェアツールです。
また、ICEを用いてデバッグを行う際に必要なマスクオプション設定用ファイルも同時に作成できます。

ウィンドウ



オプションリスト領域

機種情報定義ファイル(s1c63xxx.ini)で設定される、マスクオプションの一覧です。チェックボックスを使用して、各オプションを選択します。
チェックマーク(✓)は現在選択されているオプションを示します。

ファンクションオプションドキュメント領域

オプションの選択内容がファンクションオプションドキュメントの形式で表示されます。ファンクションオプションドキュメントファイルには、この領域の表示内容が出力されます。オプションリスト領域で選択項目を変更すると、表示が即時更新されます。

メッセージ領域

[Tool]メニューから[Generate]を選択、あるいは[Generate]ボタンをクリックしてファイルを作成した際に、その結果を示すメッセージを表示します。

ボタン

ツールバー

[Open]ボタン
ファンクションオプションドキュメントファイルを開きます。

[Generate]ボタン
オプションリストの選択内容でファイルを作成します。

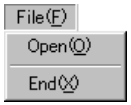
[Setup]ボタン
作成日や出力ファイル名、ファンクションオプションドキュメントファイルに含めるコメントなどを設定します。

[Device INI Select]ボタン
機種情報定義ファイル(s1c63xxx.ini)をロードします。

[Help]ボタン
winfogのバージョンを表示します。

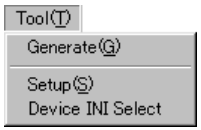
メニュー

[File]メニュー



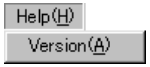
Open
ファンクションオプションドキュメントファイルを開きます。
End
winfogを終了します。

[Tool]メニュー



Generate
オプションリストの選択内容でファイルを作成します。
Setup
作成日や出力ファイル名、ファンクションオプションドキュメントファイルに含めるコメントなどを設定します。
Device INI Select
機種情報定義ファイル(s1c63xxx.ini)をロードします。

[Help]メニュー



Version
winfogのバージョンを表示します。

エラーメッセージ

File name error	ファイル名または拡張子名の文字数が使用可能範囲を超えている。
Illegal character	入力禁止文字が入力された。
Please input file name	ファイル名が未入力。
Can't open File : xxxx	ファイル(xxxx)がオープンできない。
INI file is not found	指定した機種情報定義ファイル(.ini)が存在しない。
INI file does not include FOG information	指定した機種情報定義ファイル(.ini)にファンクションオプション情報が含まれていない。
Function Option document file is not found	指定したファンクションオプションドキュメントファイルが存在しない。
Function Option document file does not match INI file	指定したファンクションオプションドキュメントファイルの内容が機種情報定義ファイル(.ini)と異なる。
A lot of parameter	コマンドラインの引数が多すぎる。
Making file(s) is completed [xxxx is no data exist]	ファイル作成完了。ただし、作成したファイル(xxxx)にはデータが含まれていない。
Can't open File: xxxx Making file(s) is not completed	Generate実行時、ファイル(xxxx)がオープンできない。
Can't write File: xxxx Making file(s) is not completed	Generate実行時、ファイル(xxxx)に書き込みができない。

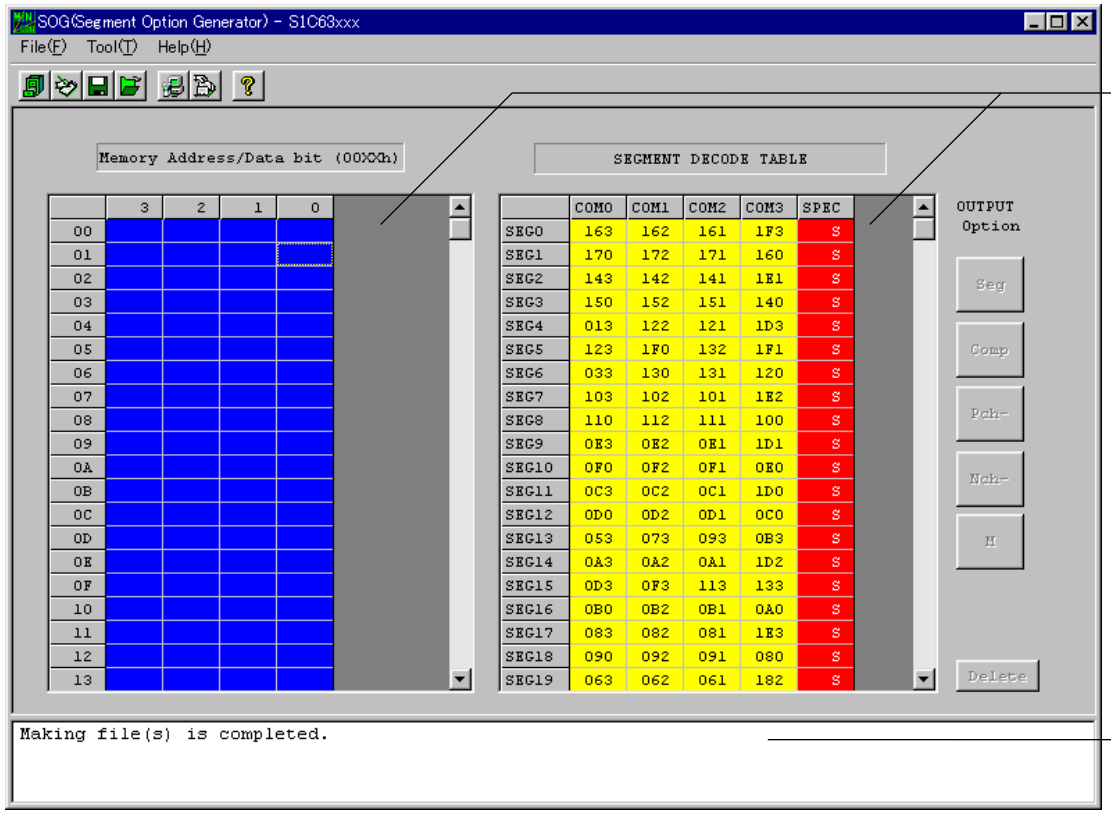
ワーニングメッセージ

Are you file update? xxxx is already exist	上書き確認メッセージ (指定したファイルは既に存在する。)

概要

セグメントオプションジェネレータwinsogは、LCD出力端子の出力仕様、表示メモリとLCD出力端子割り付けのマスクパターン生成のためのファイルを作成するソフトウェアツールです。
また、ICEを用いてデバッグを行う際に必要なマスクオプション設定用ファイルも同時に作成できます。

ウィンドウ



オプション設定領域

表示メモリマップとセグメントデコードテーブル、端子の仕様を選択するボタンで構成されています。表示メモリマップとセグメントデコードテーブルのセルをクリックすることで、表示メモリアドレス/ビットの割り付けが行えます。

Seg	LCDセグメント出力を選択します。
Comp	DC-コンプリメンタリ出力を選択します。
Pch-	DC-Pchオープンドレイン出力を選択します。
Nch-	DC-Nchオープンドレイン出力を選択します。
M	セグメント/コモン共有出力を選択します。
Delete	選択したセグメント割り付けをクリアします。

メッセージ領域

[Tool]メニューから[Generate]を選択、あるいは[Generate]ボタンをクリックしてファイルを作成した際に、その結果を示すメッセージを表示します。

ボタン

ツールバー


[Open]ボタン
セグメントオプションドキュメントファイルを開きます。


[Save]ボタン
現在のオプション設定内容をファイル(セグメント割り付けデータファイル)に保存します。


[Load]ボタン
セグメント割り付けデータファイルを読み込みます。


[Generate]ボタン
オプションリストの選択内容でファイルを作成します。

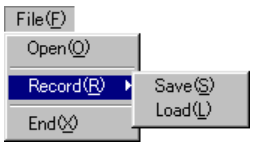

[Setup]ボタン
作成日や出力ファイル名、セグメントオプションドキュメントファイルに含めるコメントなどを設定します。


[Device INI Select]ボタン
機種情報定義ファイル(s1c63xxx.ini)をロードします。


[Help]ボタン
winsogのバージョンを表示します。

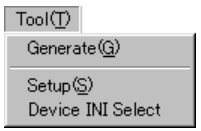
メニュー

[File]メニュー



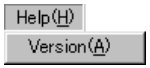
Open
セグメントオプションドキュメントファイルを開きます。
Record - Save
現在のオプション設定内容をファイル(セグメント割り付けデータファイル)に保存します。
Record - Load
セグメント割り付けデータファイルを読み込みます。
End
winsogを終了します。

[Tool]メニュー



Generate
オプションリストの選択内容でファイルを作成します。
Setup
作成日や出力ファイル名、ファンクションオプションドキュメントファイルに含めるコメントなどを設定します。
Device INI Select
機種情報定義ファイル(s1c63xxx.ini)をロードします。

[Help]メニュー



Version
winsogのバージョンを表示します。

エラーメッセージ

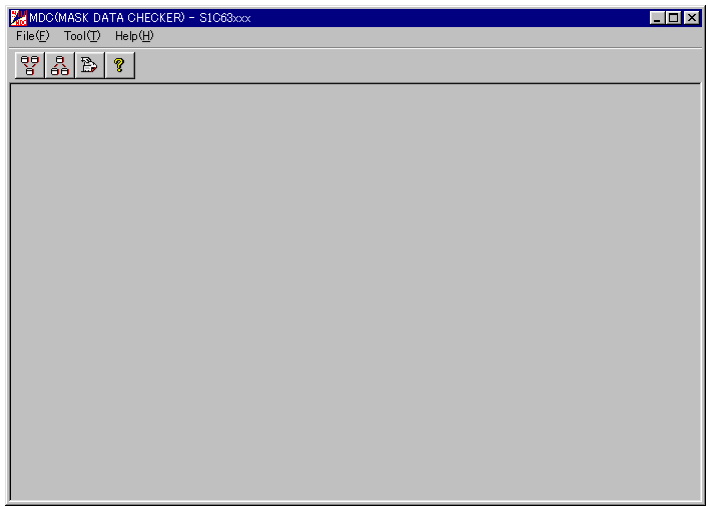
File name error	ファイル名または拡張子名の文字数が使用可能範囲を超えている。
Illegal character	入力禁止文字が入力された。
Please input file name	ファイル名が未入力。
Can't open File : xxxx	ファイル(xxxx)がオープンできない。
INI file is not found	指定した機種情報定義ファイル(.ini)が存在しない。
INI file does not include SOG information	指定した機種情報定義ファイル(.ini)にセグメントオプション情報が含まれていない。
Function Option document file is not found	指定したファンクションオプションドキュメントファイルが存在しない。
Function Option document file does not match INI file	指定したファンクションオプションドキュメントファイルの内容が機種情報定義ファイル(.ini)と異なる。
Segment Option document file is not found	指定したセグメントオプションドキュメントファイルが存在しない。
Segment Option document file does not match INI file	指定したセグメントオプションドキュメントファイルの内容が機種情報定義ファイル(.ini)と異なる。
Segment assignment data file is not found	指定したセグメント割り付けデータファイルが存在しない。
Segment assignment data file does not match INI file	指定したセグメント割り付けデータファイルの内容が機種情報定義ファイル(.ini)と異なる。
Can't open File: xxxx Making file(s) is not completed	Generate実行時、ファイル(xxxx)がオープンできない。
Can't write File: xxxx Making file(s) is not completed	Generate実行時、ファイル(xxxx)に書き込みができない。
ERROR: SPEC is not set Making file(s) is not completed	空白のSPECセルがある状態でGenerateを実行した。
ERROR: SEGMENT DECODE TABLE is not set. Making file(s) is not completed	選択したメモリアドレス/データビットセルがSEG/COM端子セルに割り付けられていない状態でGenerateを実行した。

ワーニングメッセージ

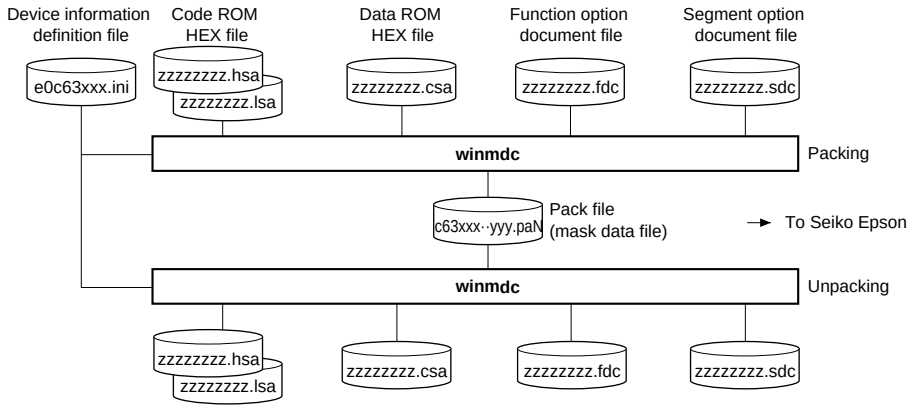
Are you file update? xxxx is already exist	上書き確認メッセージ (指定したファイルは既に存在する。)

概要

マスクデータチェッカwinmdcは、HEXコンバータhx63によって生成されたコードROMとデータROMのHEXファイル、ファンクションオプションジェネレータwinfog、セグメントオプションジェネレータwinsog、メロディアセンブラwinmlaによって生成されたオプションドキュメントファイルの各フォーマットをチェックし、マスクパターン生成のためのデータファイルを作成するソフトウェアツールです。
また、作成されたマスクデータファイルを元のファイル形式に復元する機能も持っています。



フローチャート



ボタン

ツールバー

[Pack]ボタン
ROMデータファイルとオプションドキュメントファイルをパックして、提出用のマスクデータファイルを作成します。

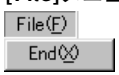
[Unpack]ボタン
パック後のファイルから元の形式のファイルを復元します。

[Device INI Select]ボタン
機種情報定義ファイル(s1c63xxx.ini)をロードします。

[Help]ボタン
winmdcのバージョンを表示します。

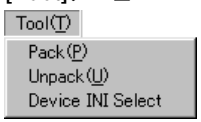
メニュー

[File]メニュー



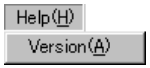
End
winmdcを終了します。

[Tool]メニュー



Pack
ROMデータファイルとオプションドキュメントファイルをパックして、提出用のマスクデータファイルを作成します。
Unpack
パック後のファイルから元の形式のファイルを復元します。
Device INI Select
機種情報定義ファイル(s1c63xxx.ini)をロードします。

[Help]メニュー



Version
winmdcのバージョンを表示します。

I/Oエラーメッセージ

File name error	ファイル名または拡張子名の文字数が使用可能範囲を超えている。
Illegal character	入力禁止文字が入力された。
Please input file name	ファイル名が未入力。
INI file is not found	指定した機種情報定義ファイル(.ini)が存在しない。
INI file does not include MDC information	指定した機種情報定義ファイル(.ini)にMDC情報が含まれていない。
Can't open file : xxxx	ファイル(xxxx)がオープンできない。
Can't write file: xxxx	ファイル(xxxx)に書き込みができない。

ROMデータエラーメッセージ

Hex data error: Not S record.	データが"S"で始まっていない。
Hex data error: Data is not sequential.	データが昇順に並んでいない。
Hex data error: Illegal data.	不当なキャラクタがある。
Hex data error: Too many data in one line.	1行中のデータ数が多すぎる。
Hex data error: Check sum error.	チェックサムが合わない。
Hex data error: ROM capacity over.	データ容量が大きい。(データサイズ>ROMサイズ)
Hex data error: Not enough the ROM data.	データ容量が少ない。(データサイズ<ROMサイズ)
Hex data error: Illegal start mark.	スタートマークが不当である。
Hex data error: Illegal end mark.	エンドマークが不当である。
Hex data error: Illegal comment.	データの最初の機種名表示が不当である。

ファンクションオプションデータエラーメッセージ

Option data error : Illegal model name.	機種名が不当である。
Option data error : Illegal version.	バージョンが不当である。
Option data error : Illegal option number.	オプションNo.が不当である。
Option data error : Illegal select number.	選択肢No.が不当である。
Option data error : Mask data is not enough.	マスクデータが充分でない。
Option data error : Illegal start mark.	スタートマークが不当である。
Option data error : Illegal end mark.	エンドマークが不当である。

セグメントオプションデータエラーメッセージ

LCD segment data error : Illegal model name.	機種名が不当である。
LCD segment data error : Illegal version.	バージョンが不当である。
LCD segment data error : Illegal segment No.	セグメントNo.が不当である。
LCD segment data error : Illegal segment area.	表示メモリのアドレスが範囲外である。
LCD segment data error : Illegal segment output specification.	出力仕様が不当である。
LCD segment data error : Illegal data in this line.	16進数と出力仕様以外の記述がある。
LCD segment data error : Data is not enough.	セグメントデータが充分でない。
LCD segment data error : Illegal start mark.	スタートマークが不当である。
LCD segment data error : Illegal end mark.	エンドマークが不当である。

セイコーエプソン株式会社

マイクロデバイス事業部　IC営業部

<IC 国内営業グループ>

東京　〒191-8501 東京都日野市日野421-8
TEL（042）587-5313（直通）　FAX（042）587-5116

大阪　〒541-0059 大阪市中央区博労町3-5-1 エプソン大阪ビル15F
TEL（06）6120-6000（代表）　FAX（06）6120-6100

ドキュメントコード：404569309
1996 年 6 月 作成Ⓛ B
2014 年 5 月 改訂Ⓗ