お客様各位

カタログ等資料中の旧社名の扱いについて

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

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

2010年4月1日
ルネサスエレクトロニクス株式会社

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

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

ご注意書き

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

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

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

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

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

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

User's Manual

M16C R8C FoUSB/UARTソフトウェア

ユーザーズマニュアル

ルネサスマイクロコンピュータ開発環境システム

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2005.02

安全設計に関するお願い

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

本資料ご利用に際しての留意事項

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

製品の内容及び本書についてのお問い合わせ先

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

¥SUPPORT¥製品名¥SUPPORT.TXT

株式会社ルネサス ソリューションズ

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

はじめに

High-performance Embedded Workshop は、ルネサスのマイクロコンピュータ用に、C/C++言語およびアセンブリ言語で書いたアプリケーションの開発およびデバッグを簡単に行うためのグラフィカルユーザインタフェースを提供します。 アプリケーションを実行するエミュレータやシミュレータへのアクセス、計測、および変更に関して、高機能でしかも直観的な手段を提供することを目的としています。

本ヘルプでは、High-performance Embedded Workshop の主に「デバッガ」としての機能について 説明しています。

対象システム

本デバッガは、FoUSB/UART, スタータキット上で動作します。

対象 CPU

本ヘルプは、以下の CPU に対応したデバッグ機能を説明しています。

- M16C/60, M16C/30, M16C/Tiny, R8C/Tiny シリーズ
  (注)この CPU に依存する情報については、本ヘルプでは「M16C/R8C 用」と記載しています。

このページは白紙です。

デバッガの起動/セットアップ編

このページは白紙です。

1. 機能概要

本デバッガは、以下の機能を持っています。

1.1 ブレーク機能

以下のブレーク機能をサポートしています。

1.1.1 ソフトウェアブレーク

指定したアドレスの命令を実行する直前でターゲットプログラムを停止する機能です。
設定可能なブレークポイント数は接続する MCU に依存します。 複数のソフトウェアブレークポイントを指定した場合、いずれかのブレークポイント到達でブレークします。

1.2 GUI 入出力機能

ユーザターゲットシステムのキー入力パネル(ボタン)や出力パネルをウィンドウ上で模擬する機能です。
入力パネルにはボタン、出力パネルにはラベル(文字列)および LED が使用できます。

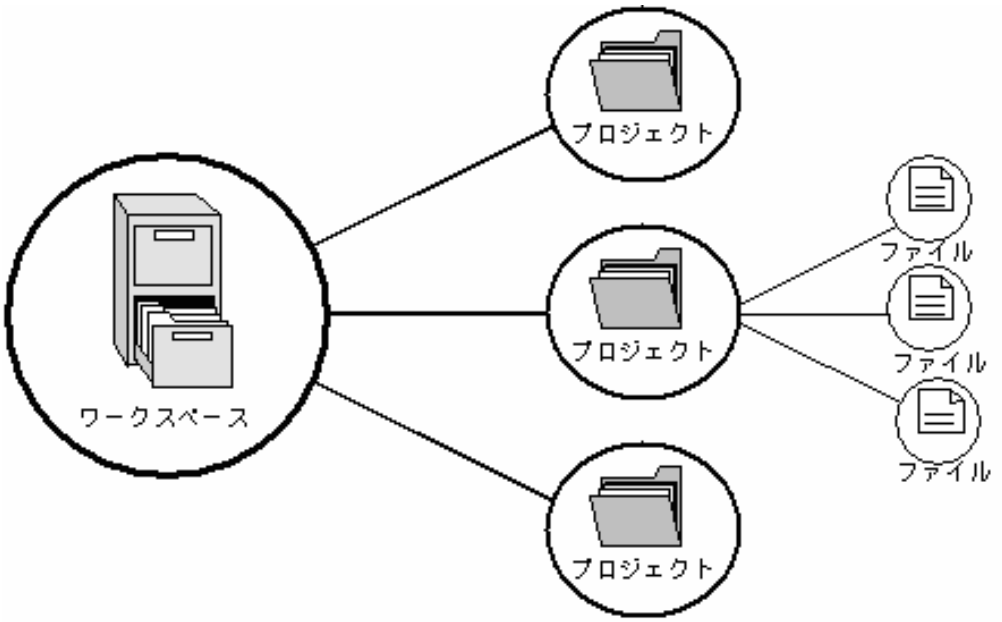
2. デバッグの準備

本製品を起動し、エミュレータ に接続してデバッグを開始します。
なお、本製品でデバッグを行うためには、ワークスペースを作成する必要があります。

2.1 ワークスペース，プロジェクト，ファイルについて

ワードプロセッサでドキュメントを作成、修正できるのと同じように、本製品ではワークスペースを作成、修正できます。

ワークスペースはプロジェクトを入れる箱と考えることができます。 同じように、プロジェクトはプロジェクトファイルを入れる箱と考えることができます。 したがって各ワークスペースにはプロジェクトが 1 つ以上あり、各プロジェクトにはファイルが 1 つ以上あります。

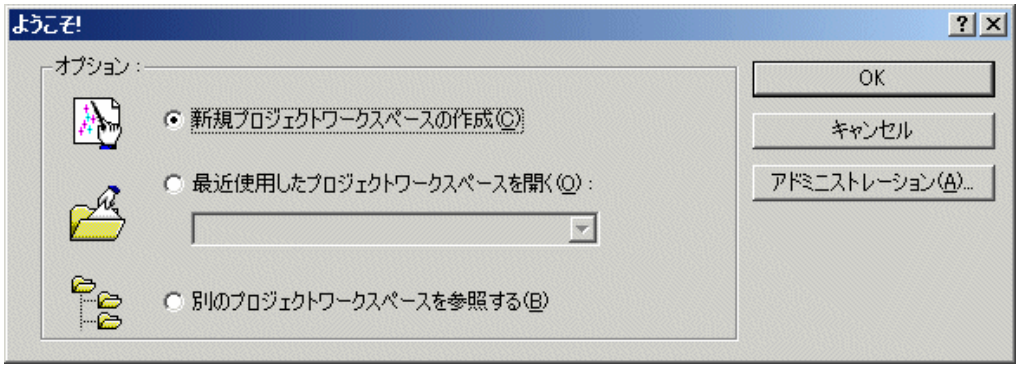


ワークスペースでは関連したプロジェクトを 1 つにまとめることができます。 例えば、異なるプロセッサに対して 1 つのアプリケーションを構築しなければならない場合、または、アプリケーションとライブラリを同時に開発している場合などに便利です。 さらに、ワークスペース内でプロジェクトを階層的に関連づけることができます。 つまり、1 つのプロジェクトを構築すると、その子プロジェクトを最初に構築します。

ワークスペースを活用するには、ユーザは、まずワークスペースにプロジェクトを追加して、そのプロジェクトにファイルを追加しなければなりません。

2.2 High-performance Embedded Workshop の起動

[スタート]メニューの[プログラム]から High-performance Embedded Workshop を起動してください。
[ようこそ!]ダイアログボックスが表示されます。



このダイアログで、ワークスペースを作成/表示します。

- **[新規プロジェクトワークスペースの作成]ラジオボタン**
  ワークスペースを新規作成する場合に選択します。

- **[最近使用したプロジェクトワークスペースを開く]ラジオボタン**
  既存のワークスペースを使用する場合に選択します。
  開いたワークスペースの履歴が表示されます。

- **[別のプロジェクトワークスペースを参照する]ラジオボタン**
  既存のワークスペースを使用する場合に選択します。
  開いた履歴が残っていない場合に使用します。

既存ワークスペースを指定する場合は、[最近使用したプロジェクトワークスペースを開く]または[別のプロジェクトワークスペースを参照する]ラジオボタンを選択し、ワークスペースファイル(拡張子.hws)を指定してください。

新規ワークスペースの作成方法については、以下を参照ください。

- 「2.2.1 新規にワークスペースを作成する（ツールチェイン使用）」

- 「2.2.2 新規にワークスペースを作成する場合（ツールチェイン未使用）」
  ※既存のロードモジュールファイルを本製品でデバッグする場合などは、この方法でワークスペースを作成します。

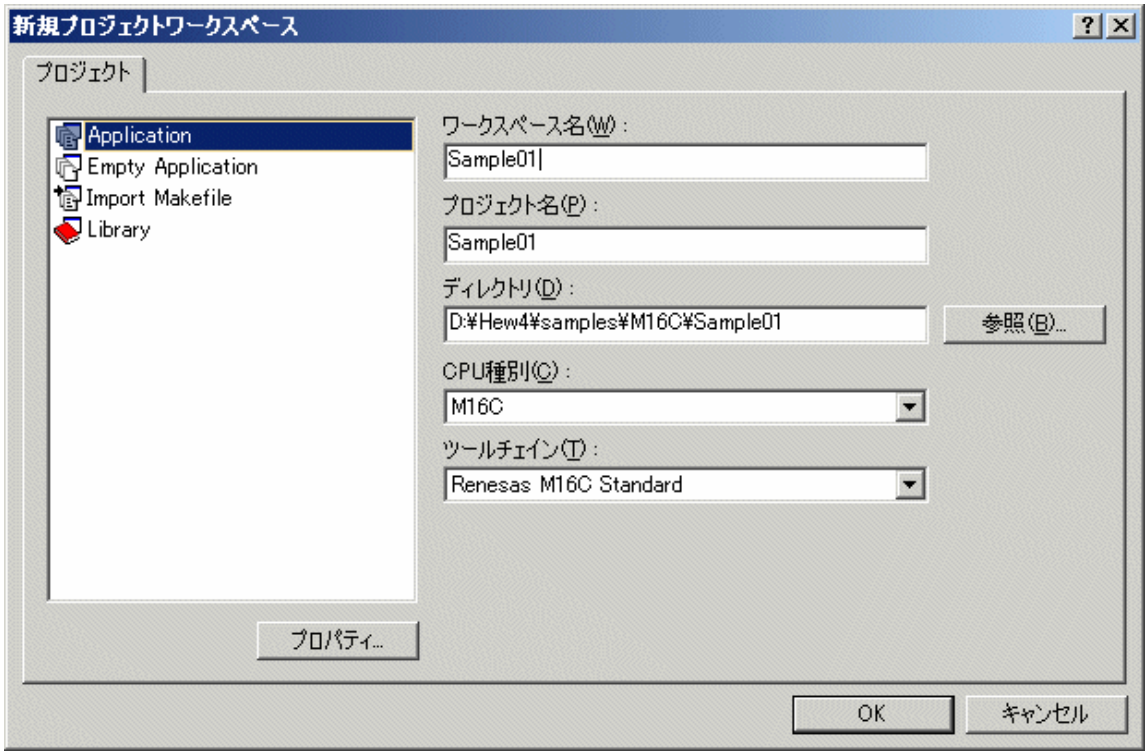
ツールチェインを使用する場合と使用しない場合では新規プロジェクトワークスペースの作成手順が異なります。本製品には、ツールチェインは含まれていません。ツールチェインはご使用の CPU に対応した C/C++コンパイラパッケージがインストールされている環境にて使用することができます。
ツールチェインを使用した新規プロジェクトワークスペースの作成についての詳細は、C/C++コンパイラパッケージ付属のマニュアルを参照してください。

2.2.1 新規にワークスペースを作成する（ツールチェイン使用）

2.2.1.1 Step1： 新規プロジェクトワークスペースの設定

High-performance Embedded Workshop 起動時に表示される、[ようこそ!]ダイアログボックスで、[新規プロジェクトワークスペースの作成]ラジオボタンを選択し、 [OK]ボタンをクリックしてください。

新規プロジェクトワークスペースの作成を開始します。
以下の画面が開きます。



1. CPU 種別を選択する

[CPU 種別]ドロップダウンリストボックスで、使用する CPU ファミリを選択してください。

2. ツールチェインを選択する

[ツールチェイン]ドロップダウンリストボックスで､該当するツールチェイン名を選択してください。

3. プロジェクトタイプを選択する

左の[プロジェクトタイプ]リストボックスで、使用したいプロジェクトタイプを選択します。
ここで、"Application" を選択してください。
（選択できるプロジェクトタイプの詳細については、C/C++コンパイラパッケージ付属のマニュアルを参照ください。）

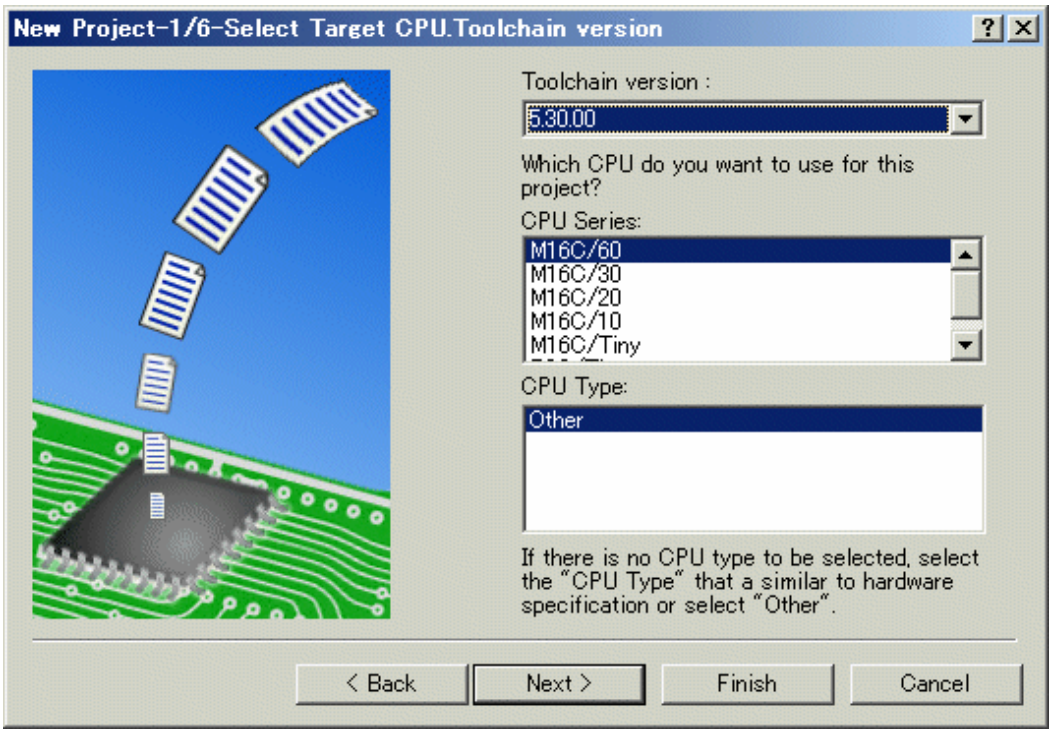
4. ワークスペース名、プロジェクト名を指定する

- [ワークスペース名]エディットボックスに、新規作成するワークスペース名を入力してください。
- [プロジェクト名]エディットボックスに､プロジェクト名を入力してください。 ワークスペース名と同じでよろしければ、入力する必要はありません。
- [ディレクトリ]エディットボックスに、ワークスペースを作成するディレクトリを入力してください。[参照...]ボタンをクリックしてワークスペースを作成するディレクトリを選択することもできます。

入力後、[OK]ボタンを押してください。

2.2.1.2 Step2： ツールチェインの設定

プロジェクト作成ウィザードが起動します。



ウィザードの最初のほうでは以下の設定を行います。

- ツールチェインの設定
- リアルタイム OS に関する設定（使用する場合）
- 生成ファイル、ヒープ領域、スタック領域等の設定

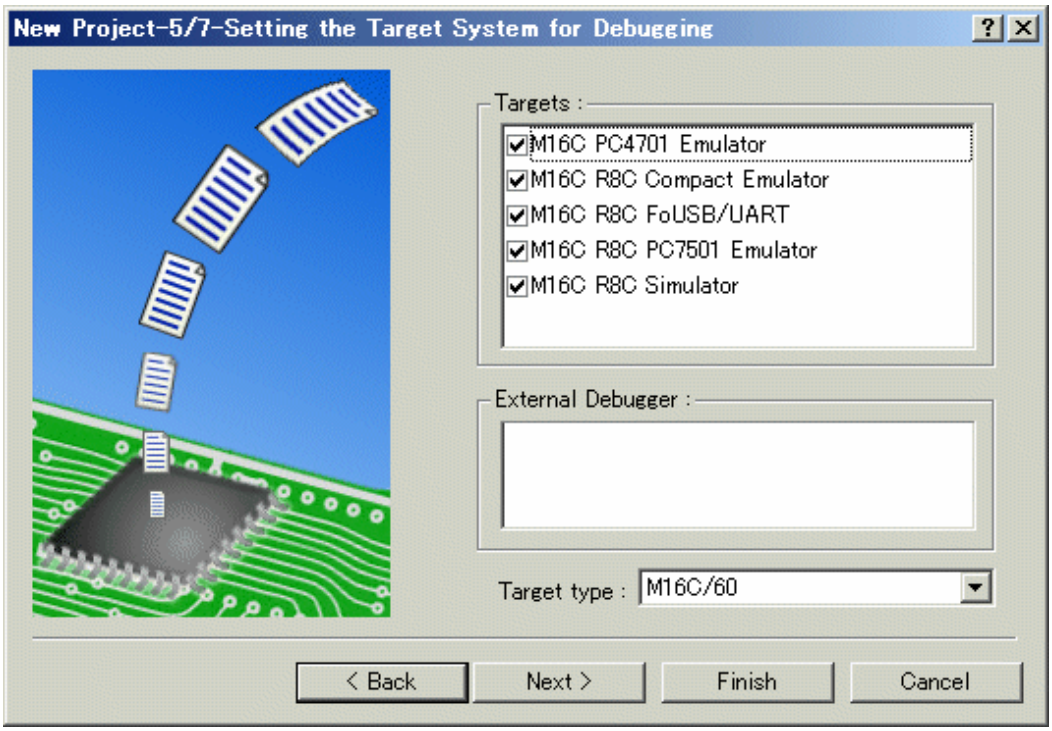
必要な情報を入力し、[次へ]ボタンを押していってください。

設定内容はご使用の C/C++コンパイラパッケージにより異なります。 設定内容の詳細については、C/C++コンパイラパッケージ付属のマニュアルを参照ください。

2.2.1.3 Step 3: ターゲットプラットフォームの選択

ウィザードの終盤で、使用するターゲット（エミュレータ，シミュレータ）の設定を行います。

ツールチェインの設定が終了したら、以下の画面が表示されます。



1. ターゲットタイプの選択

[Target type]ドロップダウンリストボックスで、使用するターゲットの CPU タイプを選択ください。

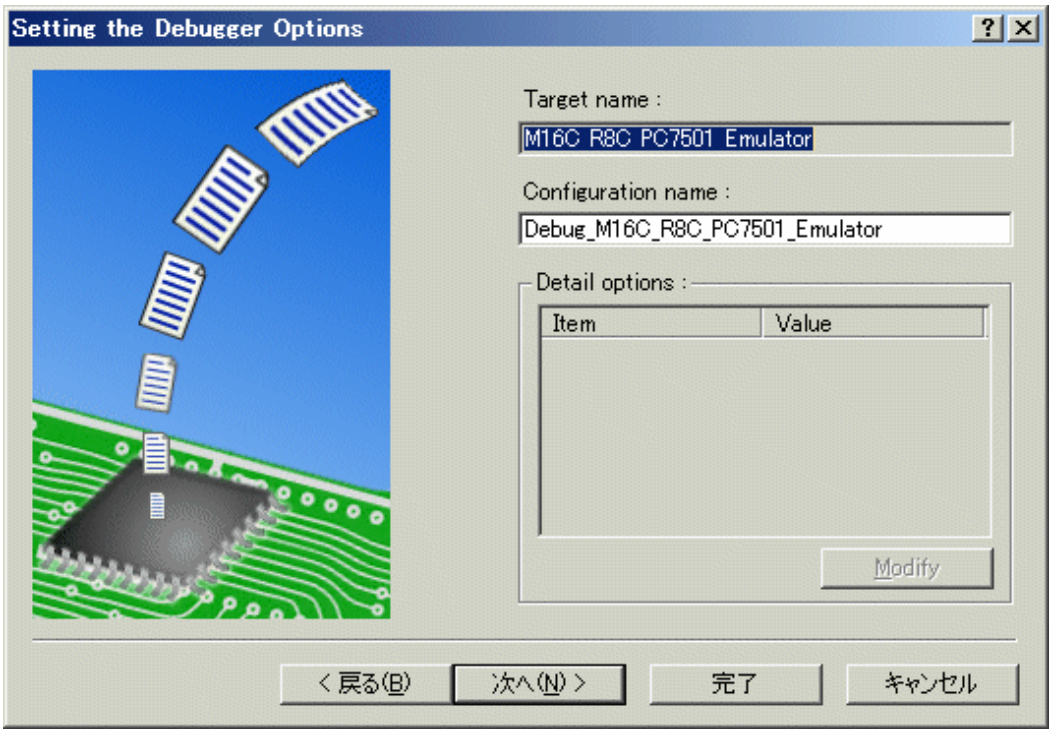
2. ターゲットプラットフォームの選択

[Targets]領域に、使用可能なターゲットが表示されます。
使用するターゲットをチェックしてください（複数指定可能）。

入力後、[次へ]ボタンを押してください。

2.2.1.4 Step4： コンフィグレーションファイル名の設定

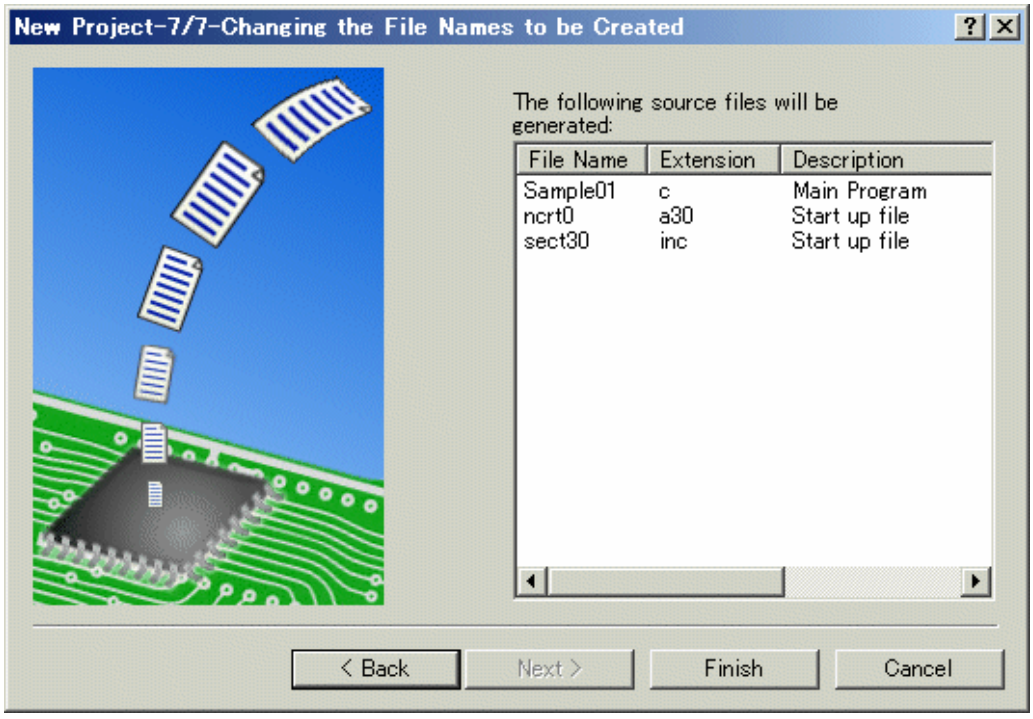
選択したターゲット毎にコンフィグレーションファイル名を設定します。
コンフィグレーションとは、ターゲット以外の High-performance Embedded Workshop の状態を保存するファイルです。



デフォルトの名前がすでに設定されていますので、変更する必要がなければそのまま[次へ]ボタンで進んでください。

2.2.1.5 Step5： 生成ファイルの確認

これまでの設定により本製品が生成するファイルが表示されます。 ファイル名を変更したい場合は、ファイル名を選択してクリック後、入力してください。



これで エミュレータ に関する設定は終了です。
画面の指示に従い、プロジェクト作成ウィザードを終了してください。

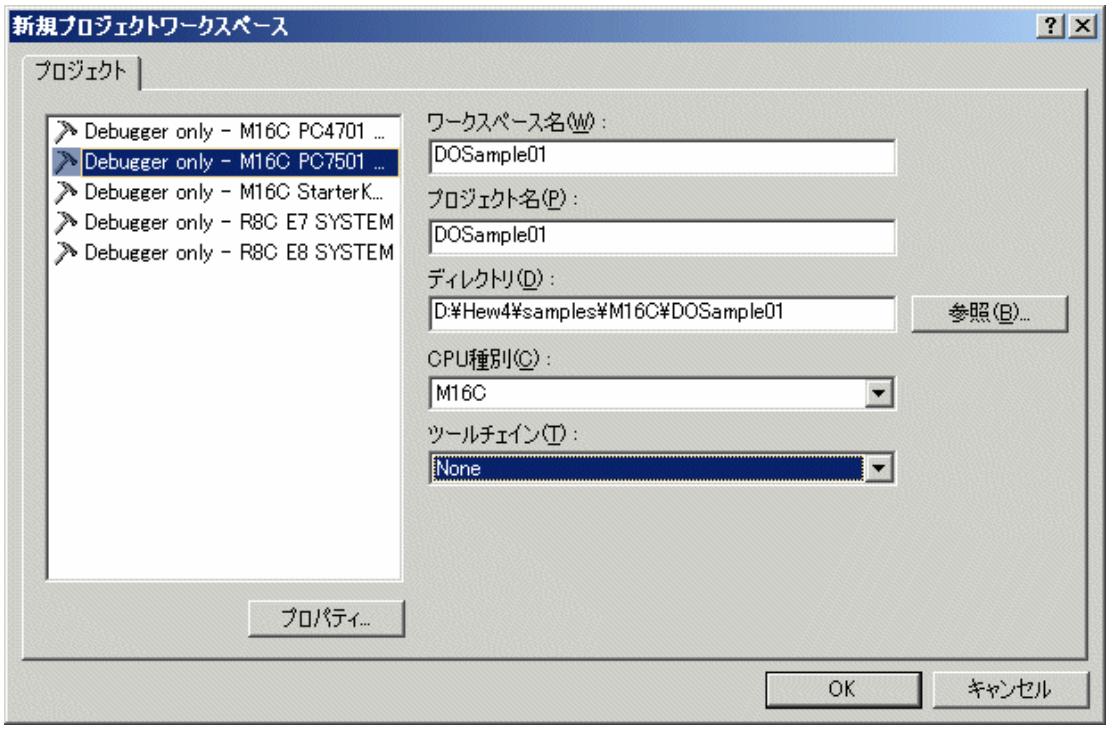
2.2.2 新規にワークスペースを作成する場合（ツールチェイン未使用）

既存のロードモジュールファイルを本製品でデバッグする場合などは、この方法でワークスペースを作成します。（ツールチェインがインストールされていなくても OK です。）

2.2.2.1 Step1： 新規プロジェクトワークスペースの設定

High-performance Embedded Workshop 起動時に表示される、[ようこそ!]ダイアログボックスで、[新規プロジェクトワークスペースの作成]ラジオボタンを選択し、 [OK]ボタンをクリックしてください。

新規プロジェクトワークスペースの作成を開始します。 以下の画面が開きます。



1. **CPU 種別を選択する**
   [CPU 種別]ドロップダウンリストボックスで、使用する CPU ファミリを選択してください。

2. **ツールチェインを選択する**
   ツールチェインは使用しませんので、[ツールチェイン]ドロップダウンリストボックスでは"None"を指定してください。
   （ツールチェインが登録されていない場合は、このドロップダウンリストは選択できません（選択不要）。）

3. **プロジェクトタイプを選択する**
   ツールチェインを使用しない場合、左の[プロジェクトタイプ]リストボックスには"Debugger only - ターゲット名" と表示されますので、 それを選択ください （複数のターゲットが表示される場合は、使用するプロジェクトタイプを 1 つ選択してください）。

4. **ワークスペース名、プロジェクト名を指定する**
   - [ワークスペース名]エディットボックスに、新規作成するワークスペース名を入力してください。
   - [プロジェクト名]エディットボックスに、プロジェクト名を入力してください。 ワークスペース名と同じでよろしければ、入力する必要はありません。

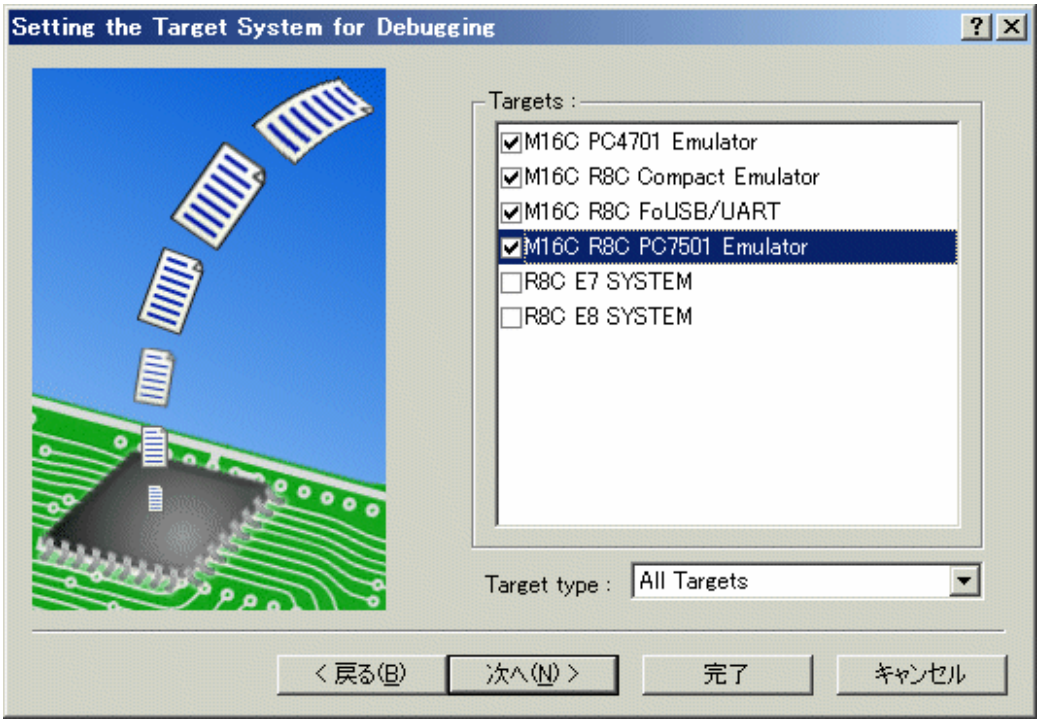
- [ディレクトリ]エディットボックスに、ワークスペースを作成するディレクトリを入力してください。[参照...]ボタンをクリックしてワークスペースを作成するディレクトリを選択することもできます。

入力後、[OK]ボタンを押してください。

2.2.2.2 Step 2: ターゲットプラットフォームの選択

使用するターゲット（エミュレータ，シミュレータ）の設定を行います。

プロジェクト作成ウィザードが起動し、以下の画面を表示します。



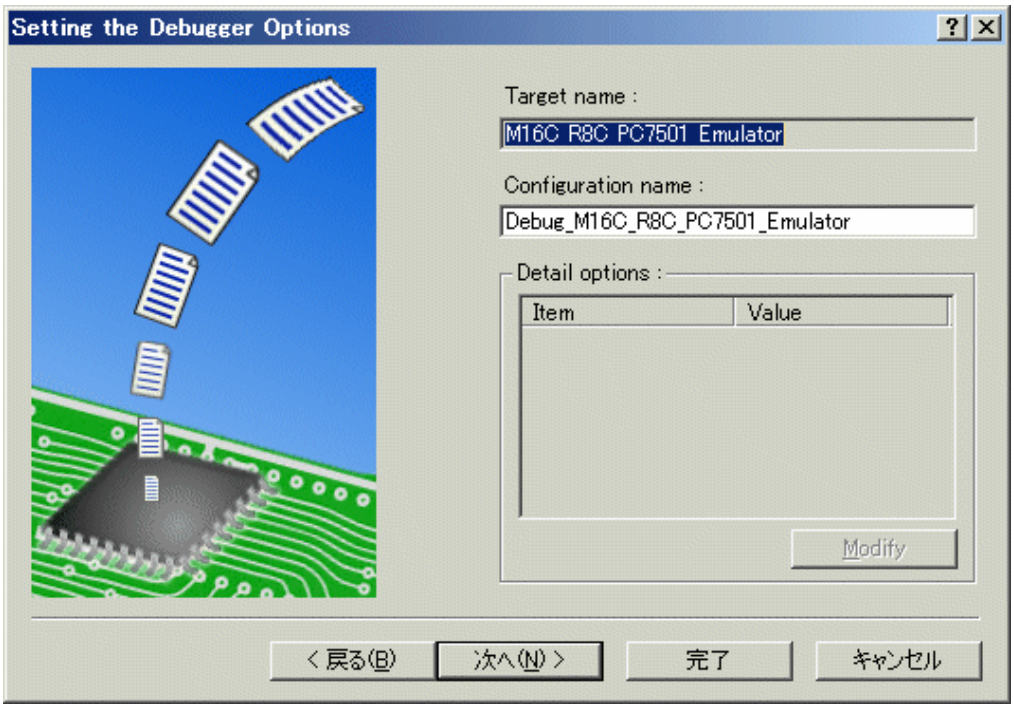
1. **ターゲットタイプの選択**
   [Target type]ドロップダウンリストボックスで、使用するターゲットの CPU タイプを選択ください。

2. **ターゲットプラットフォームの選択**
   [Targets]領域に、使用可能なターゲットが表示されます。
   使用するターゲットをチェックしてください（複数指定可能）。

入力後、[次へ]ボタンを押してください。

2.2.2.3 Step3：コンフィグレーションファイル名の設定

選択したターゲット毎にコンフィグレーションファイル名を設定します。
コンフィグレーションとは、ターゲット以外の High-performance Embedded Workshop の状態を保存するファイルです。



デフォルトの名前がすでに設定されていますので、変更する必要がなければそのまま[次へ]ボタンで進んでください。

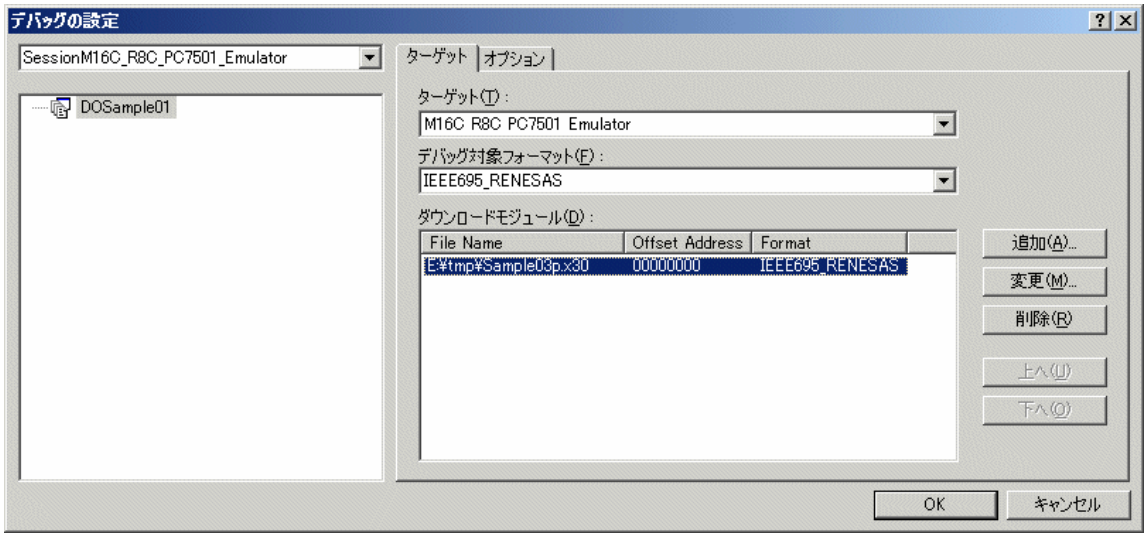
これで エミュレータ に関する設定は終了です。
画面の指示に従い、プロジェクト作成ウィザードを終了してください。

High-performance Embedded Workshop 起動と同時に、 デバッガのセットアップを開始するダイアログが表示されます（詳細は「3 デバッガのセットアップ」を参照ください）。 エミュレータ の準備が完了していれば、そのままセットアップを行い、 エミュレータ へ接続してください。

2.2.2.4 Step4： ダウンロードモジュールの登録

最後に、使用するロードモジュールファイルを登録します。

[デバッグ]メニューから[デバッグの設定...]を選択してください。 開いたダイアログボックスで、以下の設定を行います。

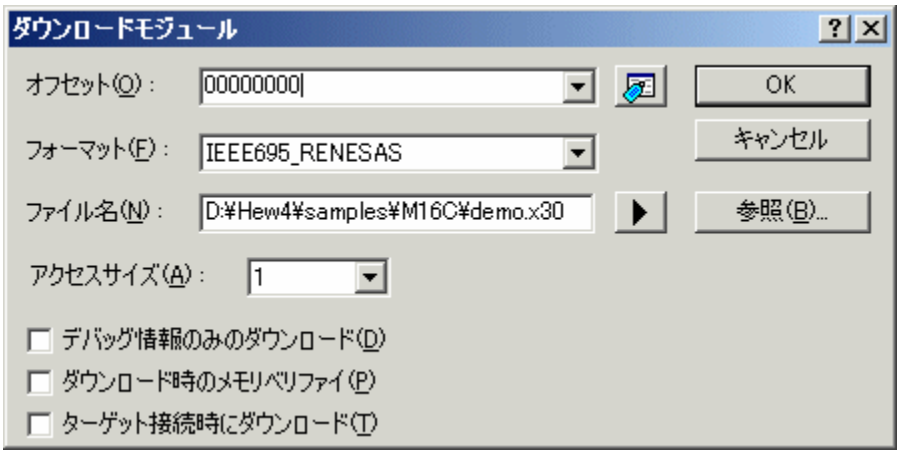


1. [ターゲット]ドロップダウンリストボックスで接続したい製品名を選択してください。

2. [デバッグ対象フォーマット] ドロップダウンリストボックスで、ダウンロードするロードモジュールの形式を選択してください。

フォーマット名	種類
IEEE695_RENESAS	IEEE-695 フォーマットファイル（弊社製クロスツール NCxx で生成）
IEEE695_IAR	IEEE-695 フォーマットファイル（IAR 社製クロスツールで生成）
IEEE695_TASKING	IEEE-695 フォーマットファイル（TASKING 社製クロスツールで生成）
ELF/DWARF2_IAR	ELF/DWARF2 フォーマットファイル（IAR 社製クロスツールで生成）
ELF/DWARF2_TASKING	ELF/DWARF2 フォーマットファイル（TASKING 社製クロスツールで生成）

3. [ダウンロードモジュール]リストボックスに、ダウンロードモジュールを登録してください。
   ダウンロードモジュールは、[追加]ボタンで開く以下のダイアログで指定できます。



- [オフセット]エディットボックスに、ダウンロードモジュールをロードするオフセットを指定してください。
- [フォーマット]エディットボックスに、ダウンロードモジュールの形式を指定してください。 形式名は、上記の一覧表を参照ください。
- [ファイル名]エディットボックスに、ダウンロードモジュールのフルパスとファイル名を入力してください。
- [アクセスサイズ]リストボックスに、ダウンロード時のメモリアクセスサイズを指定してください。

設定完了後、[OK]ボタンを押してください。

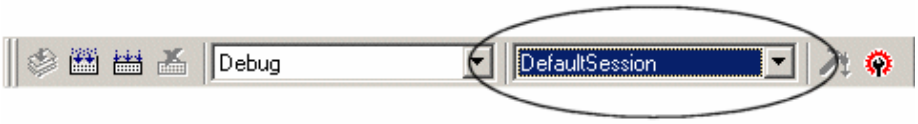
2.3 デバッガの起動

エミュレータ に接続することで、デバッグを開始できます。

2.3.1 エミュレータの接続

エミュレータ を使用する設定があらかじめ登録されているセッションファイルに切り替えることにより、エミュレータ を簡単に接続できます。

プロジェクト作成時にターゲットを選択すると、その選択したターゲットの個数分のセッションファイルがデフォルトで作成されています。
下記ツールバーのドロップダウンリストから、接続するターゲットに対応したセッションファイルを選択してください。



選択すると、デバッガのセットアップを行うためのダイアログが表示されます（詳細は「3 デバッガのセットアップ」を参照ください）。
このセットアップが終了すると、接続は完了です。

2.3.2 エミュレータの終了

以下の方法があります。

1. **セッションを"DefaultSession" に切り替える**
   エミュレータ 接続時に使用したドロップダウンリストで、"DefaultSession" を選択してください。

2. **High-performance Embedded Workshop 自体を終了する**
   [ファイル->アプリケーションの終了]を選択してください。High-performance Embedded Workshop は終了します。

セッション切り替え、および、High-performance Embedded Workshop 終了前には、セッション保存確認のメッセージボックスが表示されます。 セッション保存が必要な場合は、[はい]ボタンをクリックしてください。不要なら、[いいえ]ボタンをクリックしてください。

3. デバッガのセットアップ

3.1 Init ダイアログ

Init ダイアログにおいて、以下の内容をセットアップしてください。 Init ダイアログは、デバッガ起動時にオープンします。詳細については、表のタブ名をクリックしてください。

タブ名	内容
MCU	通信インタフェースを指定します。
デバッグ情報	使用 C コンパイラ、デバッグ情報の格納方式を指定します。
実行モード	ターゲットプログラム実行中の実行モードを指定します。
起動スクリプト	デバッガ起動時の動作を指定します。

次回デバッガ起動時に Init ダイアログをオープンしないようにするには、 Init ダイアログ下部の Next Hide をチェックしてください。

Init ダイアログは、以下のいずれかの方法で再表示できます。

- デバッガ起動後、メニュー[基本設定]→[エミュレータ]→[システム...]を選択する。
- Ctrl キーを押しながらデバッグセッションに切り替える。

3.1.1 MCU タブ

指定した内容は、次回起動時も有効となります。

3.1.1.1 MCU ファイルの指定

[参照]ボタンをクリックして下さい。
ファイルセレクションダイアログがオープンしますので、該当する MCU ファイルを指定してください。

- MCU ファイルは、ターゲット MCU の固有情報を格納したファイルです。
- 指定した MCU ファイルは、MCU タブの MCU 領域に表示されます。

3.1.1.2 通信インタフェースの指定

使用するインタフェースを選択してください。
使用可能な通信インタフェースは、エミュレータによって異なります。以下に通信インタフェースごとの設定を示します。

3.1.1.2.1. シリアル通信の設定

シリアル通信は、パーソナルコンピュータのシリアルインタフェース(RS-232C)を使用します。

<u>シリアル通信の設定</u>
シリアル通信の設定をする場合は、MCU タブ内のラジオボタン"Serial"をクリックして下さい。以下の表示になります。

Port 領域に使用するシリアルインタフェースの通信ポート、Baud Rate 領域にボーレートを指定して下さい。

3.1.1.2.2. USB 通信の設定

USB 通信は、パーソナルコンピュータの USB インタフェースを使用します。USB 1.1 に準拠しています。

<u>USBデバイスドライバのインストール</u>
USB 通信するには、あらかじめ専用のデバイスドライバがインストールされている必要があります。

Windows のプラグ&プレイ機能により USB デバイスが検出されます。 USB デバイスを検出するとデバイスドライバをインストールするためのウィザードが起動します。
以下の手順で USB デバイスドライバをインストールしてください。

1. ホストマシンとエミュレータを USB ケーブルで接続してください。

2. エミュレータの通信インタフェース設定スイッチを"USB"に設定し、電源を投入してください。

3. 以下のダイアログがオープンします。

そのままウィザードに従うとセットアップ情報ファイル(inf ファイル)を指定するためのダイアログがオープンします。本製品をインストールしたディレクトリ下の musbdrv.inf ファイルを指定してください。

注意事項

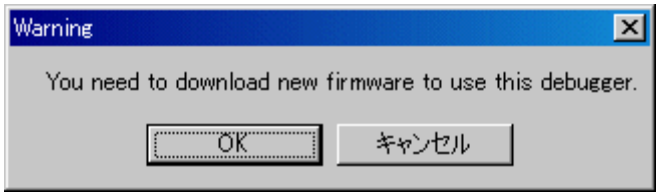
- USB デバイスドライバをインストールするには、あらかじめご使用になるエミュレータデバッガがインストールされている必要があります。先にエミュレータデバッガをインストールしてください。
- USB 通信は、Windows 98/Me/2000/XP 以外の OS では使用できません。
- Windows 2000/XP をご使用の場合、USB デバイスドライバのインストールは Administrator 権限を持つユーザが実施してください。
- インストール中にデバイスドライバ本体 musbdrv.sys が見つからないというメッセージが出る場合があります。 musbdrv.sys は、musbdrv.inf ファイルと同じディレクトリに格納されています。

USB通信の設定

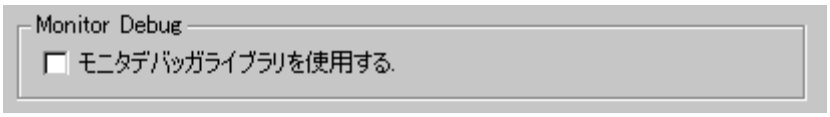
USB 通信で接続する場合は、MCU タブ内のラジオボタン"USB"をクリックして下さい。

モニタプログラムのダウンロード

MCU ファイルで指定した MCU に対応しているモニタプログラムと、現在ダウンロードされているモニタプログラムが異なる場合、モニタプログラムのダウンロードを促されます。
MCU ファイルに間違いが無ければ、OK を押してモニタプログラムをダウンロードしてください。



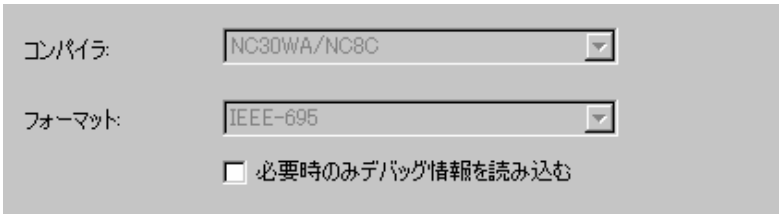
3.1.1.3 モニタデバッグ対応



モニタデバッグ用に本デバッガを起動する場合、上記チェックボックスをチェックしてください。
チェックされた状態で本デバッガを起動すると、本デバッガはデバッグシステムを初期化せずに起動します（モニタデバッグ対応のモニタが必要です）。

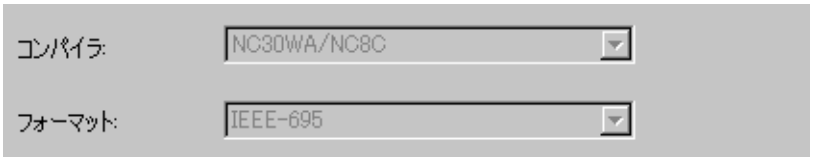
3.1.2 デバッグ情報 タブ

指定した内容は、次回ダウンロード時から有効です。



3.1.2.1 使用コンパイラ/オブジェクトフォーマットの参照

ご使用のコンパイラと、オブジェクトファイルのフォーマットを表示します。



本ダイアログで、現在の設定内容が確認できます。 設定は、メニュー[デバッグ]→[デバッグの設定...]で開くダイアログで行ってください。

3.1.2.2 デバッグ情報の格納方式指定

デバッグ情報の格納方式には、オンメモリ方式とオンデマンド方式があります。
デバッグ情報の格納方式を選択してください(デフォルトはオンメモリ方式です)。
オンデマンド方式を選択する場合、[必要時のみデバッグ情報を読み込む]チェックボックスをチェックします。

- **オンメモリ方式**
  デバッグ情報をパーソナルコンピュータのメモリ上に保持します。
  ロードモジュール（ターゲットプログラム）の規模が小さい場合に適します。

- **オンデマンド方式**
  デバッグ情報を再利用可能なテンポラリファイル上に保持します。
  同一ロードモジュールに対する二度目以降のダウンロードでは、 保持されたデバッグ情報を再利用するため、高速にダウンロード可能です。
  ロードモジュール（ターゲットプログラム）の規模が大きい場合に適します。

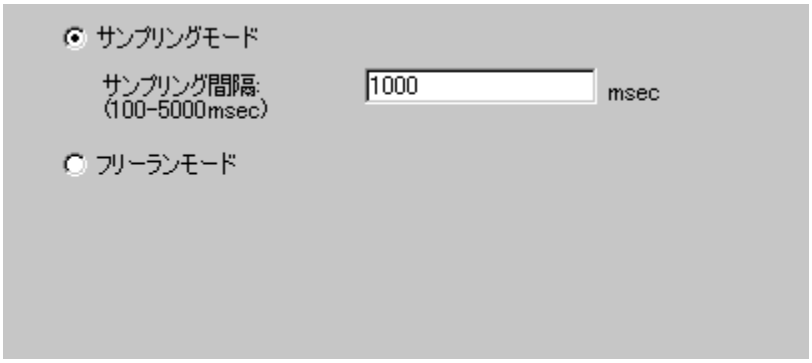
注意事項

- ロードモジュールの規模が大きい場合、オンメモリ方式では、 ダウンロード処理で非常に時間を要する場合があります。 この場合は、オンデマンド方式を選択してください。

- オンデマンド方式では、ダウンロードしたロードモジュールが位置するフォルダに、 再利用可能なテンポラリファイルを格納するフォルダを作成します。 フォルダ名は、"~INDEX_" にロードモジュール名を付加した名称です。 例えば、ロードモジュール名が "sample.abs" ならば、フォルダ名は "~INDEX_sample" です。 このフォルダは、デバッガを終了しても削除されません。

3.1.3 実行モード タブ

指定した内容は、次回起動時も有効となります。



3.1.3.1 実行モードの設定

Go 実行時および Come 実行時の、プログラムの実行モードを決定します。本デバッガでは、ソフトウェアブレークなどによるプログラムの停止を検出するため、定期的にモニタプログラムにプログラムの実行状態を問い合わせます。そのため、プログラム実行中にモニタプログラムが割り込むことになり、プログラムのリアルタイム性が失われてしまいます。そこで、定期的に実行状態を問い合わせる「サンプリングモード」と、問い合わせを行わずプログラム実行のリアルタイム性を維持する「フリーランモード」の 2 種類の実行モードが用意されています。

- サンプリングモード
  ターゲットプログラム実行中、モニタプログラムに実行状態を定期的に問い合わせます。 そのため、ソフトウェアブレークなどによるユーザープログラムの停止を検出することができます。[サンプリング間隔]に値を設定することで、問い合わせを行う間隔を指定することができます。 プログラムの実行に影響を与えない程度の間隔にしてください。
- フリーランモード
  ターゲットプログラム実行中、モニタプログラムに実行状態を問い合わせません。 そのため、プログラムのリアルタイム性は維持されますが、ソフトウェアブレークなどによるプログラムの停止を検出できません。 [プログラムの停止]ボタンまたは、Stop スクリプトコマンドで実行動作を停止させてください。

3.1.4 起動スクリプト タブ

指定した内容は、起動時のみ反映されます。起動後に Init ダイアログで再設定した内容は、有効になりません。

3.1.4.1 スクリプトコマンドの自動実行

デバッガ起動時にスクリプトコマンドを自動実行するには、"参照..."ボタンをクリックし、 実行するスクリプトファイルを指定してください。

"参照..."ボタンをクリックすることにより、ファイルセレクションダイアログがオープンします。
指定されたスクリプトファイルは、ファイル名:領域に表示されます。
スクリプトコマンドを自動実行しないようにするには、ファイル名:領域に表示された文字列を消去してください。

4. チュートリアル

4.1 はじめに

本デバッガの主な機能を紹介するために、チュートリアルプログラムを提供しています。 このプログラムを用いて各機能を説明します。

このチュートリアルプログラムは、C 言語で書かれており、10 個のランダムデータを昇順/降順にソートします。
チュートリアルプログラムでは、以下の処理を行います。

- tutorial 関数でソートするランダムデータを生成します。
- sort 関数では tutorial 関数で生成したランダムデータを格納した配列を入力し、昇順にソートします。
- change 関数では tutorial 関数で生成した配列を入力し、降順にソートします。

注意事項

再コンパイルを行った場合、本章で説明しているアドレスと異なることがあります。

4.2 使用方法

以下のステップに沿ってお進みください。

4.2.1 Step1： デバッガの起動

4.2.1.1 デバッグの準備

High-performance Embedded Workshop を起動し、エミュレータ に接続します。
詳細は「2 デバッグの準備」を参照ください。

4.2.1.2 デバッガのセットアップ

エミュレータ に接続すると、 デバッガをセットアップするためのダイアログが表示されます。 このダイアログでデバッガの初期設定を行います。
詳細は「3 デバッガのセットアップ」を参照ください。

デバッガのセットアップが終了すると、デバッグできる状態になります。

4.2.2 Step2：RAM の動作チェック

RAM が正常に動作することをチェックします。 [メモリ]ウィンドウでメモリ内容を表示、編集し、メモリが正常に動作することを確認します。

注意事項

マイコンによってはボード上にメモリをつけることができます。 この場合、メモリ動作チェックは上記だけでは不完全な場合があります。 メモリチェック用プログラムを作成し、チェックすることをお勧めします。

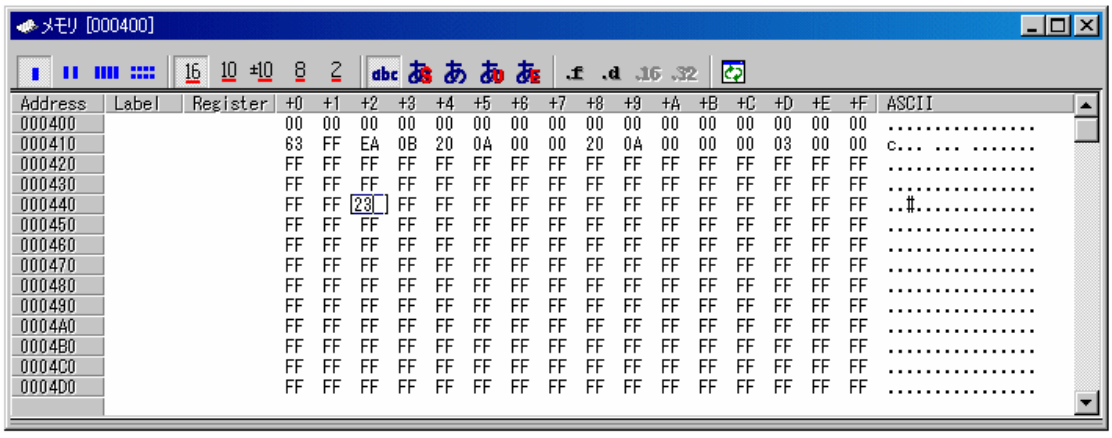
4.2.2.1 RAM の動作チェック

[表示]メニューの[CPU]サブメニューから[メモリ]を選択し、[表示開始アドレス]エディットボックスに RAM のアドレスを入力してください （ここでは"H'400"を入力しています）。 [スクロール開始アドレス][スクロール終了アドレス]エディットボックスはデフォルトの設定のままにしておきます （デフォルトの場合、メモリ全空間がスクロール領域になります）。

注意事項

各製品ごとに RAM 領域の設定は異なります。各製品のハードウェアマニュアルを参照してください。

[OK]ボタンをクリックしてください。指定されたメモリ領域を示す[メモリ]ウィンドウが表示されます。



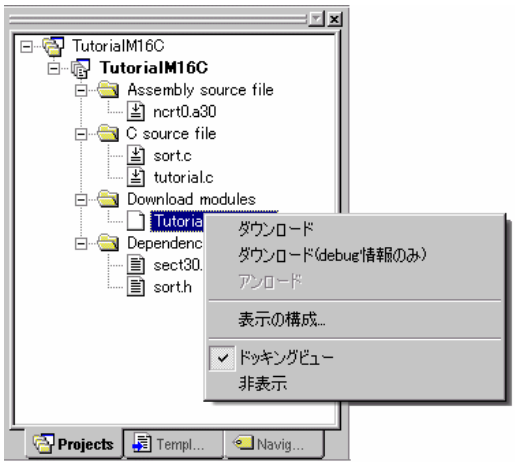
[メモリ]ウィンドウ上のデータ部分をダブルクリックすることにより、値が変更できます。

4.2.3 Step3： チュートリアルプログラムのダウンロード

4.2.3.1 チュートリアルプログラムをダウンロードする

デバッグしたいオブジェクトプログラムをダウンロードします。

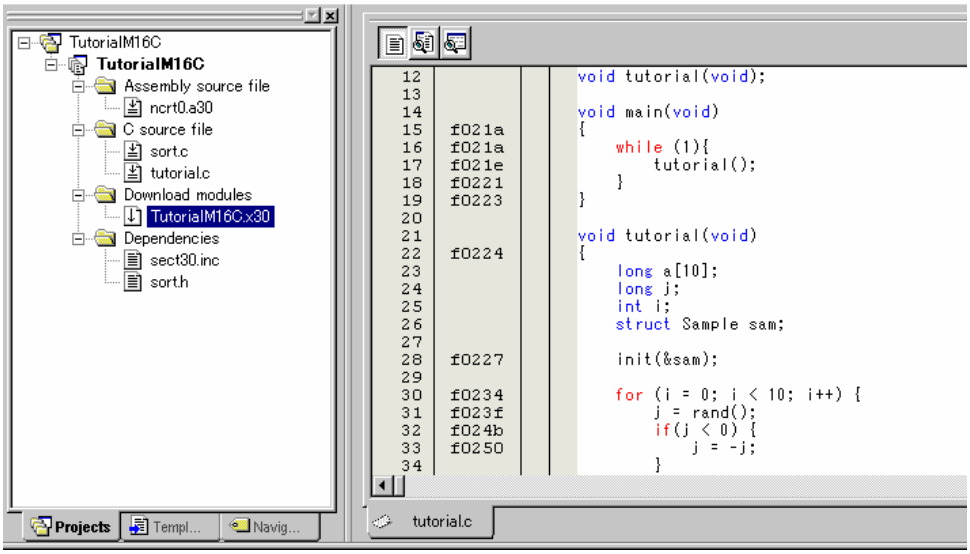
- **M16C/R8C 用デバッガの場合**
  [Download modules]の[TutorialM16C.x30]から[ダウンロード]を選択します。



4.2.3.2 ソースプログラムを表示する

本デバッガでは、ソースレベルでプログラムをデバッグできます。

[C source file]の[Tutorial.c]をダブルクリックしてください。 [エディタ(ソース)]ウィンドウが開き、"Tutorial.c"ファイルの内容を表示します。



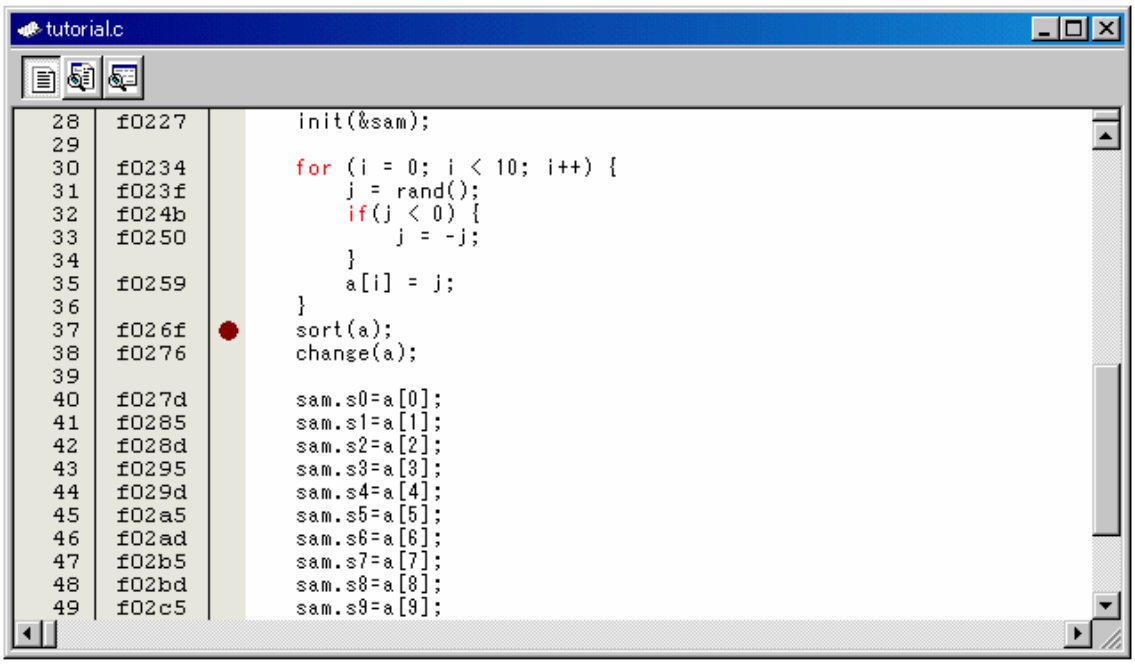
必要であれば、[基本設定]メニューから[表示の形式]オプションを選択し、見やすいフォントとサイズを選択してください。
[エディタ(ソース)]ウィンドウは、最初はプログラムの先頭を示しますが、スクロールバーを使って他の部分を見ることができます。

4.2.4 Step4： ブレークポイントの設定

基本的なデバッグ機能の 1 つにソフトウェアブレークポイントがあります。
[エディタ(ソース)]ウィンドウにおいて、ソフトウェアブレークポイントを簡単に設定できます。

4.2.4.1 ソフトウェアブレークポイントを設定する

例えば、sort 関数のコール箇所にソフトウェアブレークポイントを設定します。
sort 関数コールを含む行の[S/W ブレークポイント]カラムをダブルクリックしてください。



sort 関数を含む行に、赤色の印が表示されます。この表示によりソフトウェアブレークポイントが設定されたことを示しています。

4.2.5 Step5： プログラムの実行

プログラムの実行方法について説明します。

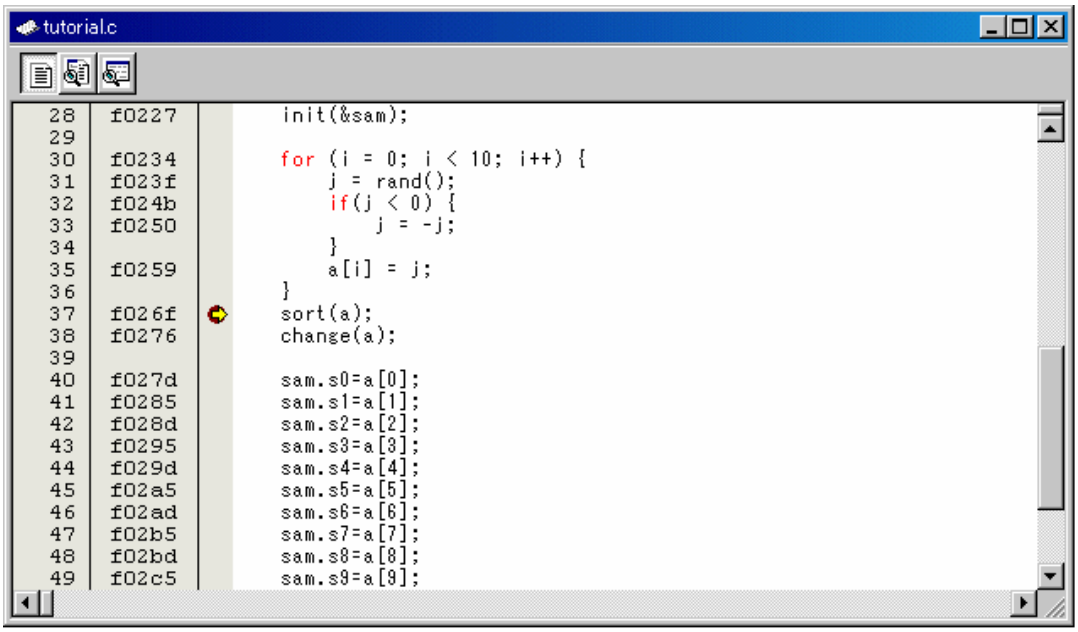
4.2.5.1 CPU のリセット

初期状態ではプログラムをダウンロード後、CPU はリセットされていません。
CPU をリセットする場合は、[デバッグ]メニューから[CPU のリセット]を選択するか、ツールバー上の[CPU のリセット]ボタン  を選択してください。

4.2.5.2 プログラムを実行する

プログラムを実行する場合は、[デバッグ]メニューから[実行]を選択するか、ツールバー上の[実行]ボタン  を選択してください。

プログラムはブレークポイントを設定したところまで実行されます。 プログラムが停止した位置を示すために[S/W ブレークポイント]カラム中に矢印が表示されます。

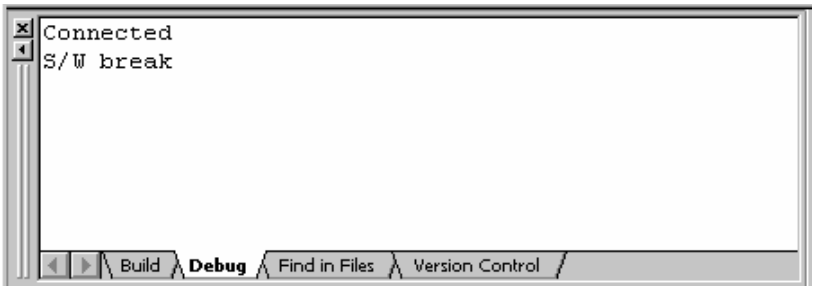


注意事項

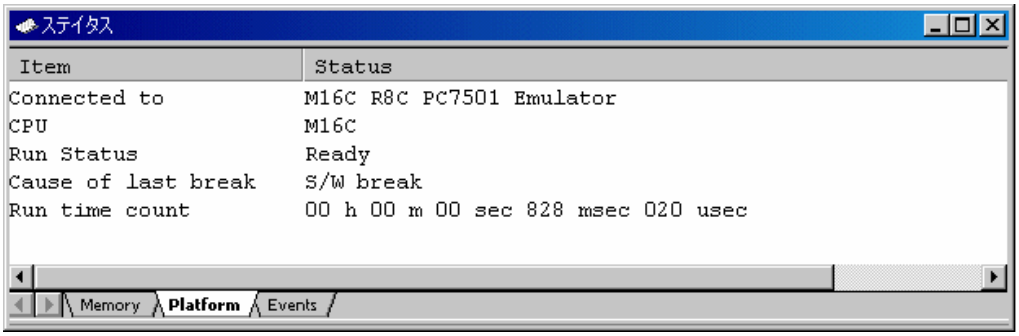
ブレーク後にソースファイルを表示する際に、ソースファイルパスを問い合わせる場合があります。その場合は、ソースファイルの場所を指定してください。

4.2.5.3 ブレーク要因を確認する

[アウトプット]ウィンドウにブレーク要因を表示します。



また、[ステイタス]ウィンドウでも、最後に発生したブレークの要因を確認できます。
[表示]メニューの[CPU]サブメニューから[ステイタス]を選択してください。 [ステイタス]ウィンドウが表示されますので、[Platform]シートを開いて Cause of last break の Status を確認してください。



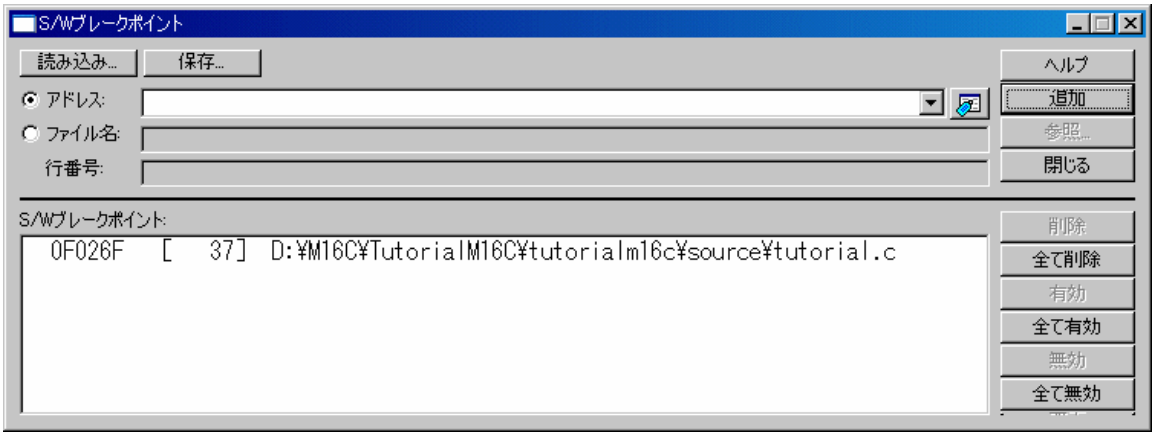
ブレーク要因の表記については、「9 プログラム停止要因の表示」を参照ください。

4.2.6 Step6： ブレークポイントの確認

設定した全てのソフトウェアブレークポイントは、[S/W ブレークポイント]ウィンドウで確認することができます。

4.2.6.1 ブレークポイントを確認する

[表示]メニューの[ブレーク]サブメニューから[S/W ブレークポイント]を選択してください。 [S/W ブレークポイント]ウィンドウが表示されます。



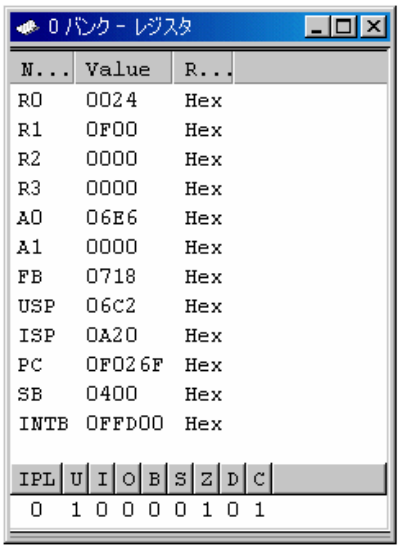
このウィンドウを使って、ブレークポイントの設定/変更、新しいブレークポイントの定義、 およびブレークポイントの削除、有効/無効の選択ができます。

4.2.7 Step7： レジスタ内容の確認

レジスタ内容は、[レジスタ]ウィンドウで確認することができます。

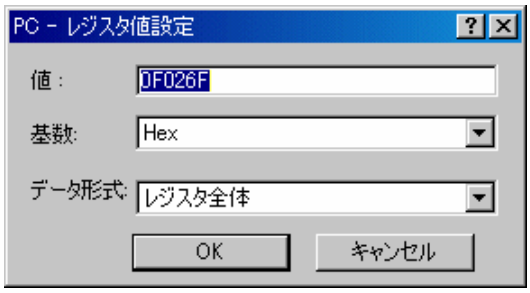
4.2.7.1 レジスタ内容を確認する

[表示]メニューの[CPU]サブメニューから[レジスタ]を選択してください。[レジスタ]ウィンドウが表示されます。



4.2.7.2 レジスタ内容を変更する

任意のレジスタの内容を変更することができます。
変更するレジスタ行をダブルクリックして下さい。ダイアログが表示しますので、変更する値を入力ください。



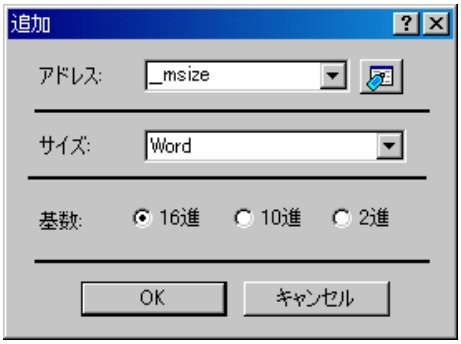
4.2.8 Step8： メモリ内容の確認

ラベル名を指定することによって、ラベルが登録されているメモリの内容を[ASM ウォッチ]ウィンドウで確認することができます。

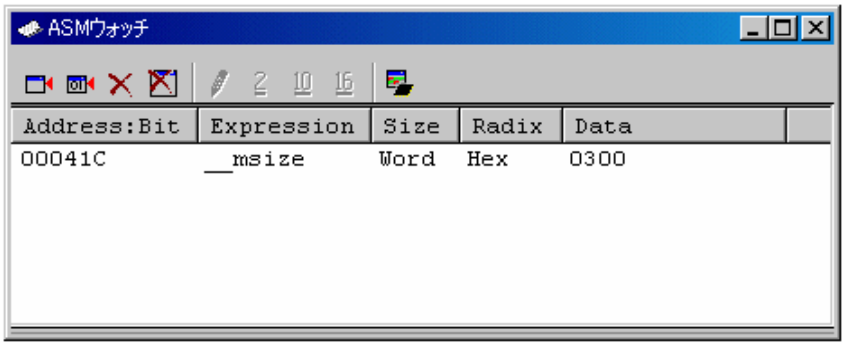
4.2.8.1 メモリ内容を確認する

例えば、以下のように、ワードサイズで__msize に対応するメモリ内容を確認します。

[表示]メニューの[シンボル]サブメニューから[ASM ウォッチ]を選択し、[ASM ウォッチ]ウィンドウを表示します。
[ASM ウォッチ]ウィンドウのポップアップメニュー[追加...]を選択し、 [アドレス]エディットボックスに"__msize"を入力し、[サイズ]コンボボックスを"Word"に設定してください。



[OK]ボタンをクリックすると、[ASM ウォッチ]ウィンドウに指定されたメモリ領域が表示されます。



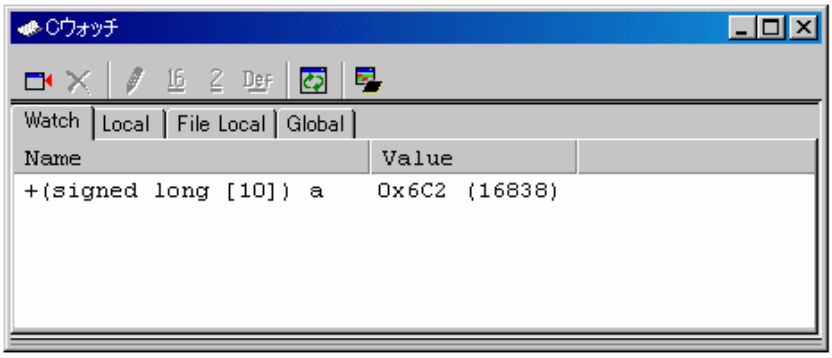
4.2.9 Step9： 変数の参照

プログラムをステップ処理するとき、プログラムで使われる変数の値が変化することを確認できます。

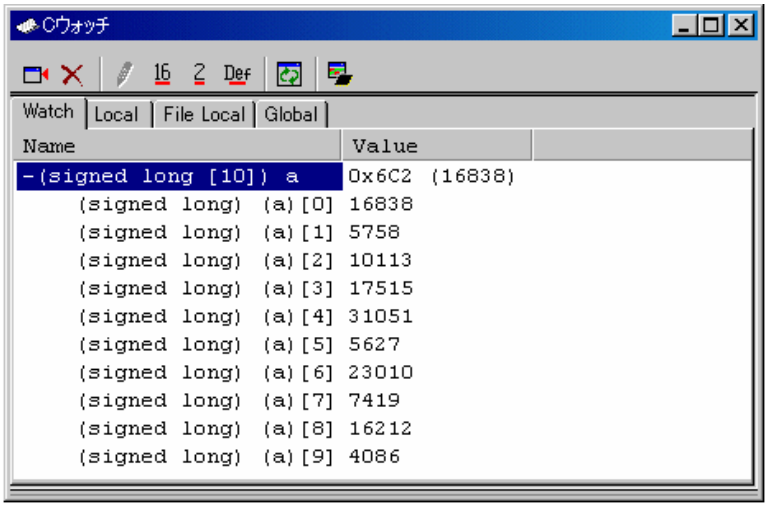
4.2.9.1 変数を参照する

例えば、以下の手順で、プログラムのはじめに宣言した long 型の配列 a を見ることができます。

[エディタ(ソース)]ウィンドウに表示されている配列 a を選択し、 マウスの右ボタンで表示するポップアップメニューの[C ウォッチウィンドウに追加]を選択してください。 [C ウォッチ]ウィンドウの[Watch]タブが開き、配列 a の内容を表示します。



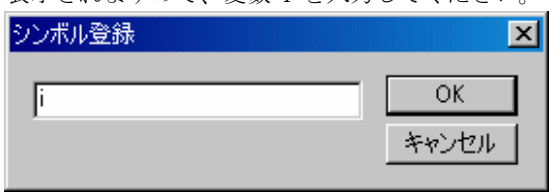
[C ウォッチ]ウィンドウの配列 a の左側にある"+"マークをクリックし、配列 a の各要素を参照することができます。



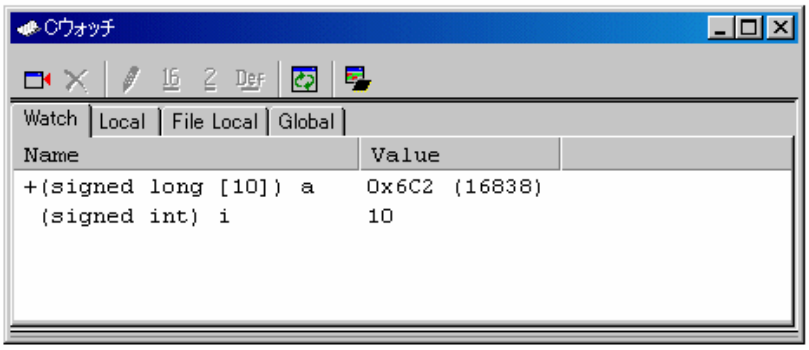
4.2.9.2 参照したい変数を登録する

また、変数名を指定して、[C ウォッチ]ウィンドウに変数を加えることもできます。

[C ウォッチ]ウィンドウのポップアップメニューから[シンボル登録...]を選択してください。
以下のダイアログボックスが表示されますので、変数 i を入力してください。



[OK]ボタンをクリックすると、[C ウォッチ]ウィンドウに、int 型の変数 i が表示されます。



4.2.10 Step10： プログラムのステップ実行

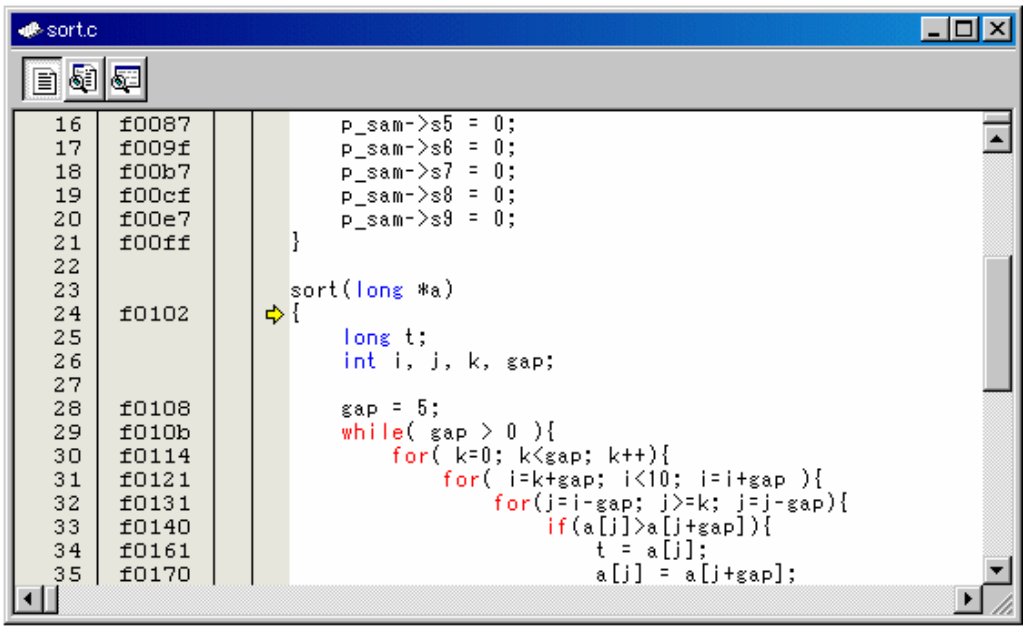
本デバッガは、プログラムのデバッグに有効な各種のステップコマンドを備えています。

1. **ステップイン**
   各ステートメントを実行します（関数内のステートメントを含む）。
2. **ステップアウト**
   関数を抜け出し、関数を呼び出したプログラムの次のステートメントで停止します。
3. **ステップオーバ**
   関数コールを 1 ステップとして、ステップ実行します。
4. **ステップ...**
   指定した速度で指定回数分ステップ実行します。

4.2.10.1 ステップインの実行

ステップイン機能はコール関数の中に入り、コール関数の先頭のステートメントで停止します。

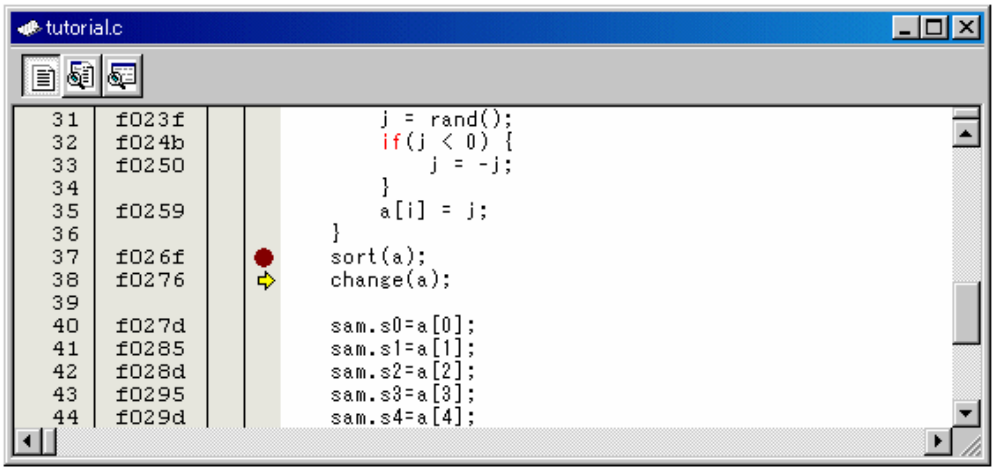
sort 関数の中に入るために、[デバッグ]メニューから[ステップイン]を選択するか、 またはツールバーの[ステップイン]ボタン  をクリックしてください。
[エディット(ソース)]ウィンドウの PC 位置を示すカーソルが、sort 関数の先頭のステートメントに移動します。



4.2.10.2 ステップアウトの実行

ステップアウト機能はコール関数の中から抜け出し、コール元プログラムの次のステートメントで停止します。

sort 関数の中から抜け出すために、[デバッグ]メニューから[ステップアウト]を選択するか、 またはツールバーの[ステップアウト]ボタン  をクリックしてください。
[エディット(ソース)]ウィンドウの PC 位置を示すカーソルが、sort 関数を抜け出し、change 関数の手前に移動します。



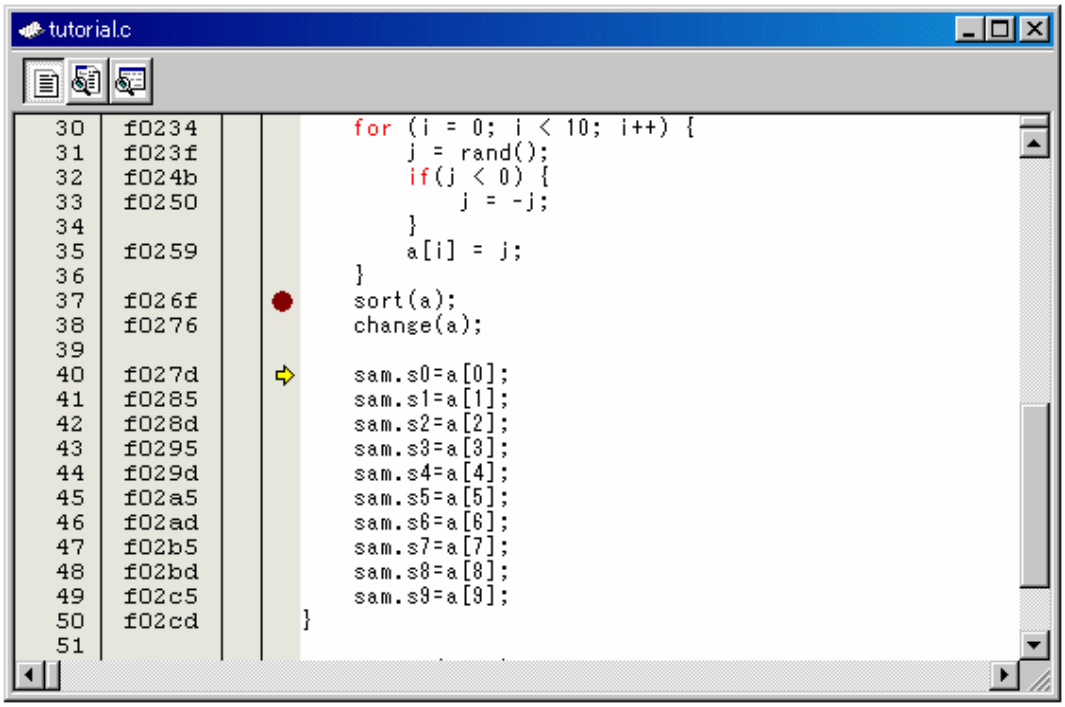
注意事項

本機能は処理時間がかかります。コール元が分かっている場合は、[カーソル位置まで実行]をご使用ください。

4.2.10.3 ステップオーバの実行

ステップオーバ機能は関数コールを 1 ステップとして実行して、メインプログラムの次のステートメントで停止します。

change 関数中のステートメントを一度にステップ実行するために、[デバッグ]メニューから[ステップオーバ]を選択するか、 またはツールバーの[ステップオーバ]ボタン  をクリックしてください。
[エディット(ソース)]ウィンドウの PC 位置を示すカーソルが、change 関数の次の位置に移動します。



4.2.11 Step11： プログラムの強制ブレーク

本デバッガは、プログラムを強制的にブレークすることができます。

4.2.11.1 プログラムを強制ブレークする

ブレークをすべて解除してください。
main 関数の残り部分を実行するために､[デバッグ]メニューから[実行]を選択するか、ツールバー上の[実行]ボタン  を選択してください。

プログラムは無限ループ処理を実行していますので、強制ブレークするために、[デバッグ]メニューから[プログラムの停止]を選択するか、 ツールバー上の[停止]ボタン  を選択してください。

4.2.12 Step12： ローカル変数の表示

[C ウォッチ]ウィンドウを使って関数内のローカル変数を表示させることができます。

4.2.12.1 ローカル変数を表示する

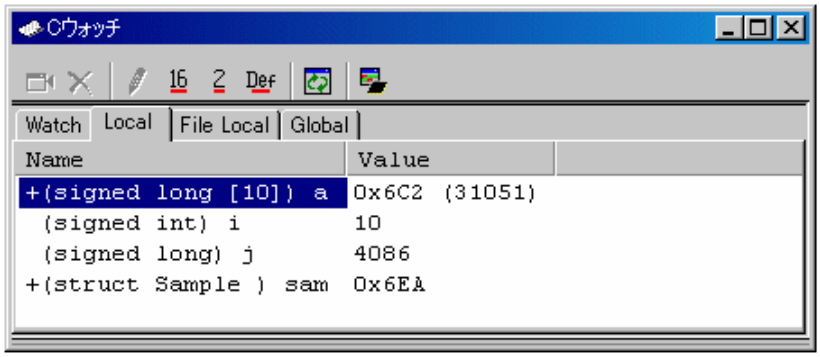
例として、tutorial 関数のローカル変数を調べます。
この関数は、4 つのローカル変数 a, j, i, sam を宣言しています。

[表示]メニューの[シンボル]サブメニューから[C ウォッチ]を選択し､[C ウォッチ]ウィンドウを表示します。
[C ウォッチ]ウィンドウには、デフォルトで以下の 4 つのタブが存在します。

- **[Watch]タブ**
  ユーザが登録した変数のみを表示します。
- **[Local]タブ**
  現在 PC が存在しているブロックで参照可能なローカル変数がすべて表示されます。 プログラム実行によりスコープが変更されると、[Local]タブの内容も切り替わります。

- **[File Local]タブ**
  現在 PC が存在しているファイルのファイルローカル変数がすべて表示されます。プログラム実行によりスコープが変更されると、[File Local]タブの内容も切り替わります。
- **[Global]タブ**
  ダウンロードしたプログラムで使用しているグローバル変数がすべて表示されます。

ローカル変数を表示する場合は、[Local]タブを選択してください。



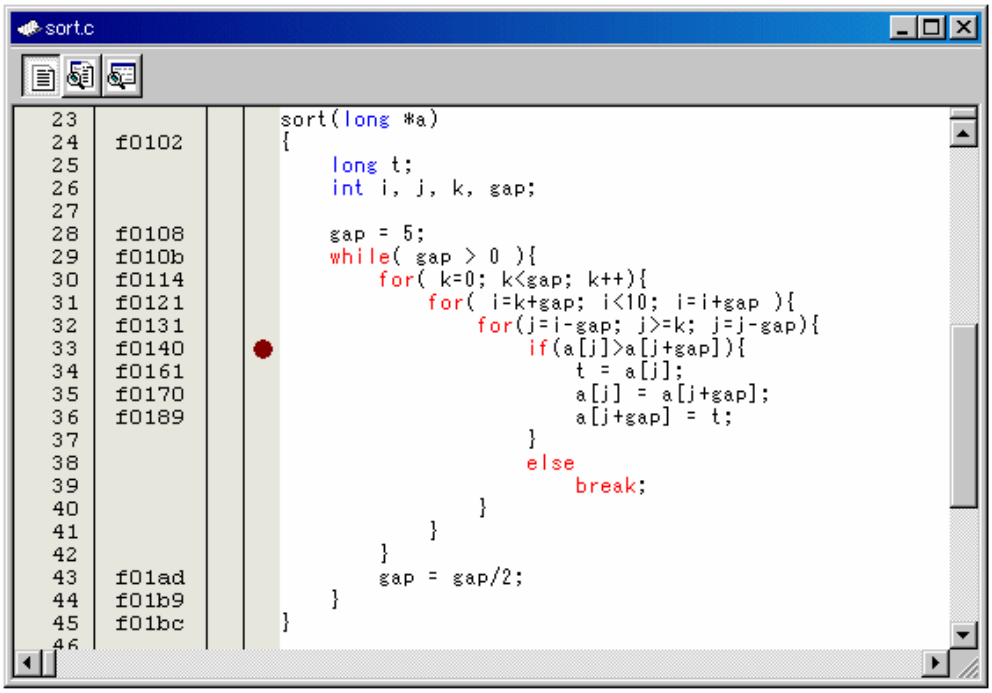
配列 a の左側にある"+"マークをクリックし、配列 a の構成要素を表示させてください。
sort 関数実行前と実行後の配列 a の要素を参照すると、ランダムデータが降順にソートされていることがわかります。

4.2.13 Step13： スタックトレース

本デバッガでは、スタック情報を用いて、現在の PC がある関数がどの関数からコールされているかを表示できます。

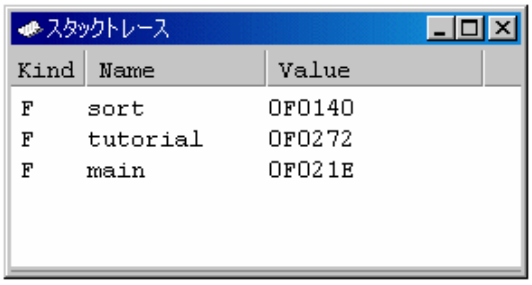
4.2.13.1 関数呼び出し状況を参照する

sort 関数内の行の[S/W ブレークポイント]カラムをダブルクリックして、ソフトウェアブレークポイントを設定してください。



プログラムを一旦リセットし、再実行します。 [デバッグ]メニューから[リセット後実行]を選択するか、ツールバー上の[リセット後実行]ボタン  を選択してください。

プログラムブレーク後、[表示]メニューの[コード]サブメニューから[スタックトレース]を選択し [スタックトレース]ウィンドウを開いてください。



現在 PC が sort()関数内にあり、sort()関数は tutorial()関数からコールされていることがわかります。

4.2.14 さて次は？

このチュートリアルでは、本デバッガの主な使い方を紹介しました。

ご使用のエミュレータで提供されるエミュレーション機能を使用することによって、さらに高度なデバッグを行うこともできます。 それによって、ハードウェアとソフトウェアの問題が発生する条件を正確に分離し、識別すると、それらの問題点を効果的に調査することができます。

リファレンス編

このページは白紙です。

5. ウィンドウ一覧

本デバッガ用のウィンドウを以下に示します
ウィンドウ名をクリックするとそのリファレンスを表示します。

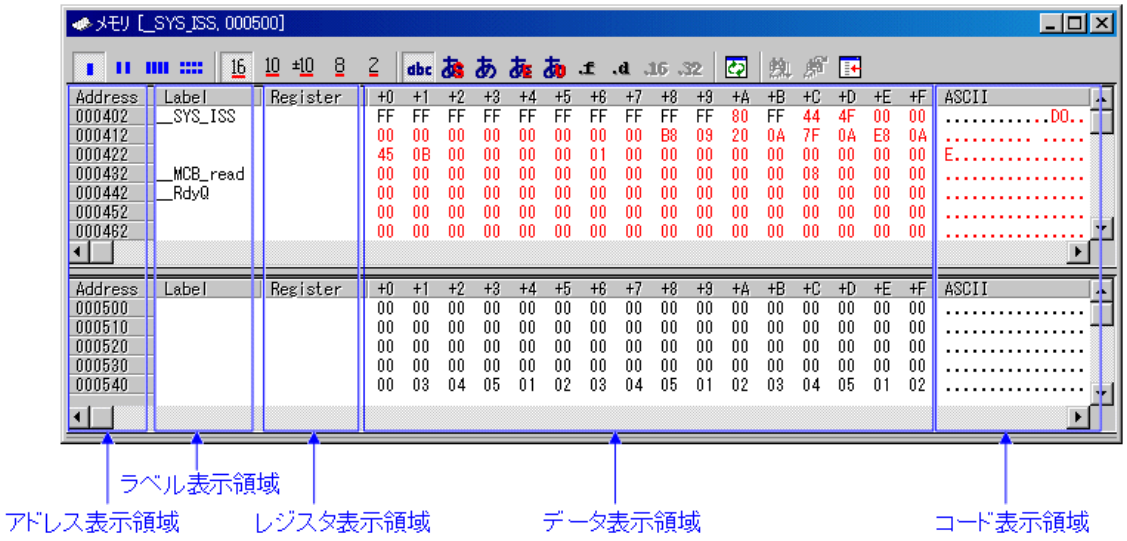
ウィンドウ名	表示用メニュー
RAM モニタウィンドウ	[表示]→[CPU]→[RAM モニタ]
ASM ウォッチウィンドウ	[表示]→[シンボル]→[ASM ウォッチ]
C ウォッチウィンドウ	[表示]→[シンボル]→[C ウォッチ]
スクリプトウィンドウ	[表示]→[スクリプト]
S/W ブレークポイント設定ウィンドウ	[表示]→[ブレーク]→[S/W ブレークポイント]
GUI 入出力ウィンドウ	[表示]→[グラフィック]→[GUI I/O]

なお、以下のウィンドウのリファレンスは High-performance Embedded Workshop 本体のヘルプに記載されていますので、 そちらをご参照ください。

- 差分ウィンドウ
- マップウィンドウ
- コマンドラインウィンドウ
- ワークスペースウィンドウ
- アウトプットウィンドウ
- 逆アセンブリウィンドウ
- メモリウィンドウ
- IO ウィンドウ
- ステイタスウィンドウ
- レジスタウィンドウ
- 画像ウィンドウ
- 波形ウィンドウ
- スタックトレースウィンドウ

5.1 RAM モニタウィンドウ

RAM モニタウィンドウは、ターゲットプログラム実行中のメモリの変化を表示するウィンドウです。
表示内容は、ターゲットプログラム実行中に一定間隔で更新されます。



- 1K バイトの RAM モニタ領域を備えています。この RAM モニタ領域は、任意の連続アドレスに配置できます。
- RAM モニタ領域は、任意のアドレス範囲に変更できます。
  RAM モニタ領域の変更方法については、「5.1.2 RAM モニタ領域を設定する」を参照してください。
  デフォルトの RAM モニタ領域は、内部 RAM 領域の先頭から 1K バイトの領域に割り当てられています。
- 表示内容の更新間隔はウィンドウごとに設定できます。
  ターゲットプログラム実行中の実際の更新間隔は、Address 表示領域のタイトル部分に表示されます。

注意事項

- 表示の更新間隔は、動作状況(以下の要因)によって指定した更新間隔より長くなる場合があります。
  - ホストマシンの性能/負荷状況
  - 通信インタフェース
  - ウィンドウのサイズ(メモリ表示範囲)や表示枚数

5.1.1 オプションメニュー

ウィンドウ内でマウスの右ボタンをクリックすると以下のポップアップメニューを表示します。これらのメニューの主要な機能は、ツールバーのボタンにも割り付けられています。

メニュー名		機能
RAM モニタ領域設定...		RAM モニタ領域を設定します。
サンプリング周期...		サンプリング周期を設定します。
アクセス履歴の消去		アクセス履歴を消去します。
前方に移動		前方（アドレスが小さい方）の RAM モニタ領域に表示位置を移動します。
後方に移動		後方（アドレスが大きい方）の RAM モニタ領域に表示位置を移動します。
表示開始アドレス...		表示開始アドレスを変更します。
スクロール範囲...		スクロール範囲を設定します。
データ長	1byte	1Byte 単位で表示します。
	2byte	2Byte 単位で表示します。
	4byte	4Byte 単位で表示します。
	8byte	8Byte 単位で表示します。
基数	16 進数表示	16 進数で表示します。
	10 進数表示	10 進数で表示します。
	符号付 10 進数表示	符号付 10 進数で表示します。
	8 進数表示	8 進数で表示します。
	2 進数表示	2 進数で表示します。
表示コード	ASCII	ASCII コードで表示します。
	SJIS	SJIS コードで表示します。
	JIS	JIS コードで表示します。
	UNICODE	UNICODE コードで表示します。
	EUC	EUC コードで表示します。
	Float	Float 型で表示します。
	Double	Double 型で表示します。
レイアウト	ラベル	ラベル表示領域の表示/非表示を切り替えます。
	レジスタ	レジスタ表示領域の表示/非表示を切り替えます。
	コード	コード領域の表示/非表示を切り替えます。
カラム...		表示カラム数を変更します。
分割		ウィンドウを分割表示します。
ツールバー表示		ツールバーの表示/非表示を切り換えます。
ツールバーのカスタマイズ...		ツールバーをカスタマイズします。
ドッキングビュー		ウィンドウをドッキングします。
非表示		ウィンドウを非表示にします。

5.1.2 RAM モニタ領域を設定する

RAM モニタウィンドウのポップアップメニュー[RAM モニタ領域設定...]を選択してください。

以下のダイアログがオープンします。
[開始アドレス]領域には現在設定されている RAM モニタ領域の先頭アドレス、[RAM モニタ領域]領域には RAM モニタ領域の範囲が表示されています（[サイズ]領域は入力不可）。

このウィンドウを使用して、RAM モニタ領域を追加、削除、変更します。

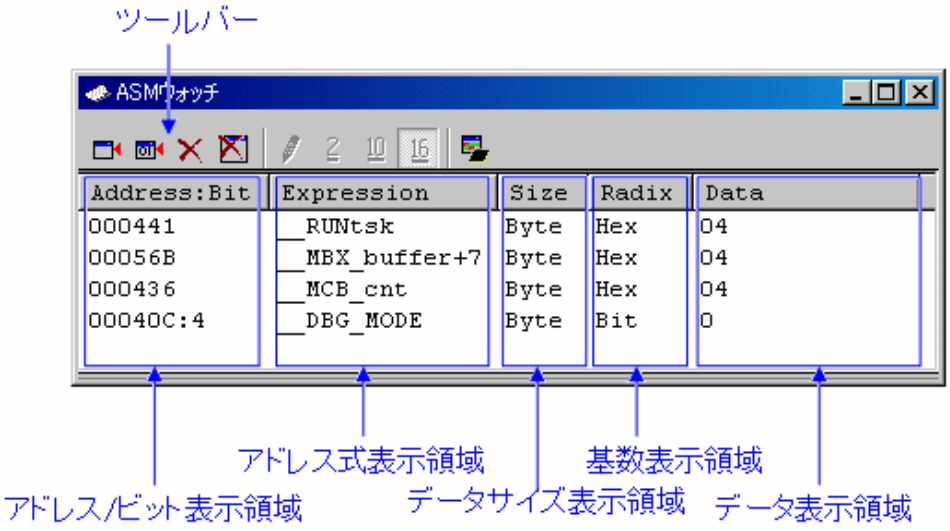
- RAM モニタ領域は、先頭アドレスで指定します。サイズは変更できません（1K バイト固定）。
- 先頭アドレスは、0x10 バイト単位で指定できます。
  端数のアドレス値を指定した場合は、0x10 バイト単位で丸め込まれた値が設定されます。

5.1.2.1 RAM モニタ領域を変更する

RAM モニタ領域の先頭アドレスを変更できます。
表示したダイアログの[開始アドレス]領域に、先頭アドレスを指定してください（[サイズ]領域は入力不可）。

5.2 ASM ウォッチウィンドウ

ASM ウォッチウィンドウは、ウォッチポイントとして特定のアドレスを登録し、メモリ内容を参照することができるウィンドウです。



- 登録するアドレスをウォッチポイントと呼びます。以下のいずれかを登録することができます。
  - アドレス(シンボルでの指定可)
  - アドレス+ビット番号
  - ビットシンボル
- 登録したウォッチポイントは、ASM ウォッチウィンドウクローズ時に保存され、再オープン時に自動登録されます。
- ウォッチポイントにシンボル/ビットシンボルを指定した場合、ウォッチポイントのアドレスはターゲットプログラムのダウンロード時に再計算されます。
- 無効なウォッチポイントは"--<not active>--"と表示します。
- (ドラッグ&ドロップ機能により)ウォッチポイントの並び順を変更することができます。
- ウォッチポイントのアドレス式、サイズ、基数、データはインプレイス編集により変更可能です。

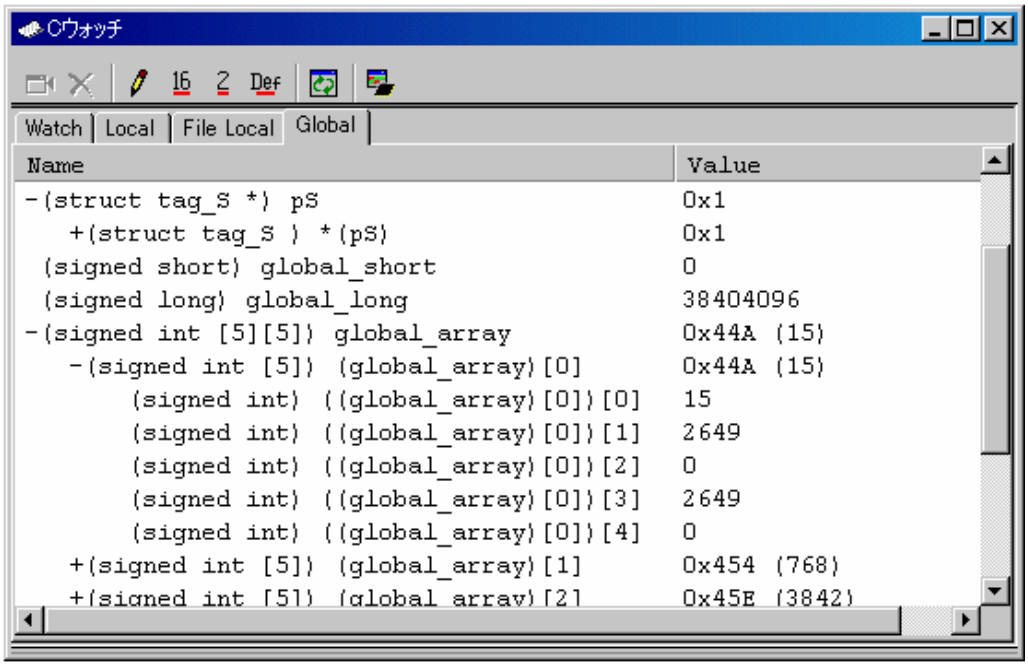
5.2.1 オプションメニュー

ウィンドウ内でマウスの右ボタンをクリックすると以下のポップアップメニューを表示します。 これらのメニューの主要な機能は、ツールバーのボタンにも割り付けられています。

メニュー名		機能
追加...		ウォッチポイントを追加します。
ビットの追加...		ビット形式のウォッチポイントを追加します。
削除		選択したウォッチポイントを削除します。
全て削除		全てのウォッチポイントを削除します。
値の編集...		選択したウォッチポイントの値を編集します。
基数	2 進数表示	2 進数で表示します。
	10 進数表示	10 進数で表示します。
	16 進数表示	16 進数で表示します。
最新の情報に更新		メモリをリフレッシュします。
レイアウト	アドレス	アドレスの表示/非表示を切り替えます。
	サイズ	サイズの表示/非表示を切り替えます。
RAM モニタ	RAM モニタ有効化	RAM モニタ機能を有効にします。
	サンプリング周期...	サンプリング周期を設定します。
ツールバー表示		ツールバーの表示/非表示を切り換えます。
ツールバーのカスタマイズ...		ツールバーをカスタマイズします。
ドッキングビュー		ウィンドウをドッキングします。
非表示		ウィンドウを非表示にします。

5.3 C ウォッチウィンドウ

C ウォッチウィンドウは、C 言語または C++言語で作成されたプログラムで使用されている変数を参照するウィンドウです。 表示されている変数を C ウォッチポイントと呼びます。



- 変数をスコープ別（ローカル、ファイルローカル、グローバル）に参照することができます。
- PC 値の変化に応じて、表示が自動的に更新されます。
- 変数値を変更することができます。
- 変数ごとに表示基数を変更できます。
- 任意の変数を Watch タブに登録し、常時表示することができます。
  - 登録した内容は、プロジェクトごとに保存されます。
  - C ウォッチウィンドウを複数オープンした場合、Watch タブの登録内容は全ウィンドウで共有されます。
- Watch タブを追加し、C ウォッチポイントの登録先を分けることができます。
- ドラッグ＆ドロップにより、他のウィンドウやエディタから変数を登録できます。
- 名前順、アドレス順にソートできます。

注意事項

- 以下に示す C ウォッチポイントは、値を変更できません。
  - ビットフィールド型変数
  - レジスタ変数
  - メモリの実体(アドレスとサイズ)を示さない C/C++言語式
- C/C++言語式が正しく計算できない場合(C シンボル未定義等)、無効な C ウォッチポイントとして登録されます。
- Local, File Local, Global タブの表示設定は保存されません。Watch タブ、および、新規に追加したタブの内容は保存されます。

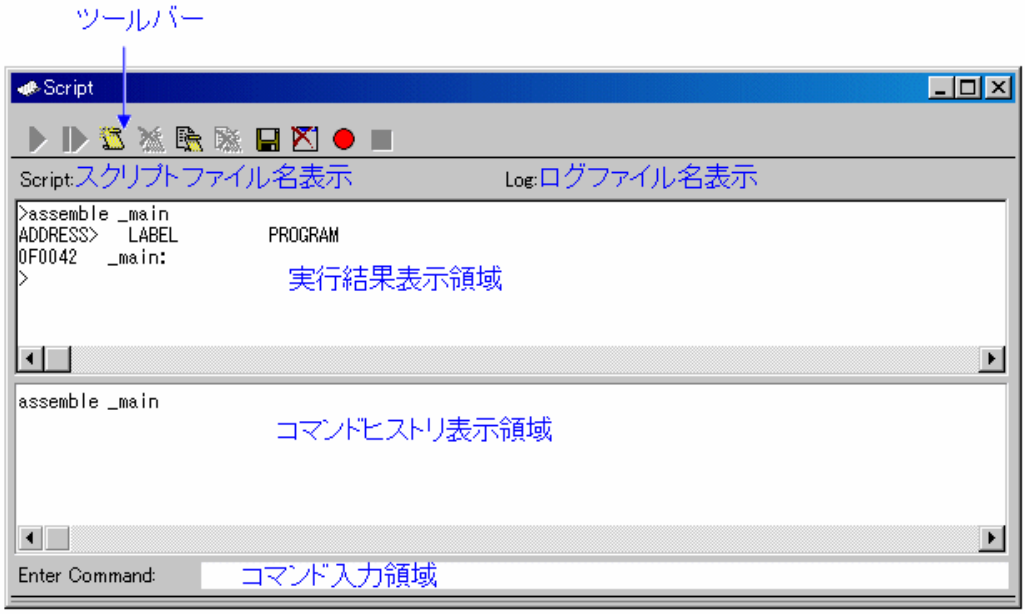
5.3.1 オプションメニュー

ウィンドウ内でマウスの右ボタンをクリックすると以下のポップアップメニューを表示します。 これらのメニューの主要な機能は、ツールバーのボタンにも割り付けられています。

メニュー名		機能
シンボル登録...		シンボルを追加します。
シンボル削除		選択したシンボルを削除します。
初期化		選択したシンボルを再評価します。
値の編集...		選択したシンボルの値を編集します。
基数	16 進数表示	16 進数で表示します。
	2 進数表示	2 進数で表示します。
	デフォルト	デフォルト基数で表示します。
	トグル(全シンボル)	表示基数を変更します(トグル)。
最新の情報に更新		メモリをリフレッシュします。
型名の非表示		型名を非表示にします。
char*の文字列表示		char*の文字列を表示します。
ソート	名前順	シンボルを名前順に並び替えます。
	アドレス順	シンボルをアドレス順に並び替えます。
RAM モニタ	RAM モニタ有効化	RAM モニタ機能を有効にします。
	サンプリング周期...	サンプリング周期を設定します。
タブの追加...		タブを追加します。
タブの削除		タブを削除します。
ツールバー表示		ツールバーの表示/非表示を切り換えます。
ツールバーのカスタマイズ...		ツールバーをカスタマイズします。
ドッキングビュー		ウィンドウをドッキングします。
非表示		ウィンドウを非表示にします。

5.4 スクリプトウィンドウ

スクリプトウィンドウは、スクリプトコマンドを実行するためのウィンドウです。
スクリプトコマンドは、ウィンドウ下部のコマンド入力領域から入力します。 コマンドの実行結果は、実行結果表示領域に表示します。
主要な操作は、ツールバーのボタンに割り付けています。



- 実行するスクリプトコマンドをあらかじめファイル(スクリプトファイル)に記述することにより、一括実行することができます。
- スクリプトコマンドの実行結果は、あらかじめ指定したファイル(ログファイル)に保存することができます。
- スクリプトウィンドウは、最新 1000 行分の実行結果を保存したバッファ(ビューバッファ)を持っています。 ログファイルの指定を忘れた場合でもスクリプトコマンドの実行結果をファイル(ビューファイル)に保存することができます。
- 実行するコマンドは、あらかじめ指定したファイルに保存することができます(スクリプトファイルとして再使用できます)。

5.4.1 オプションメニュー

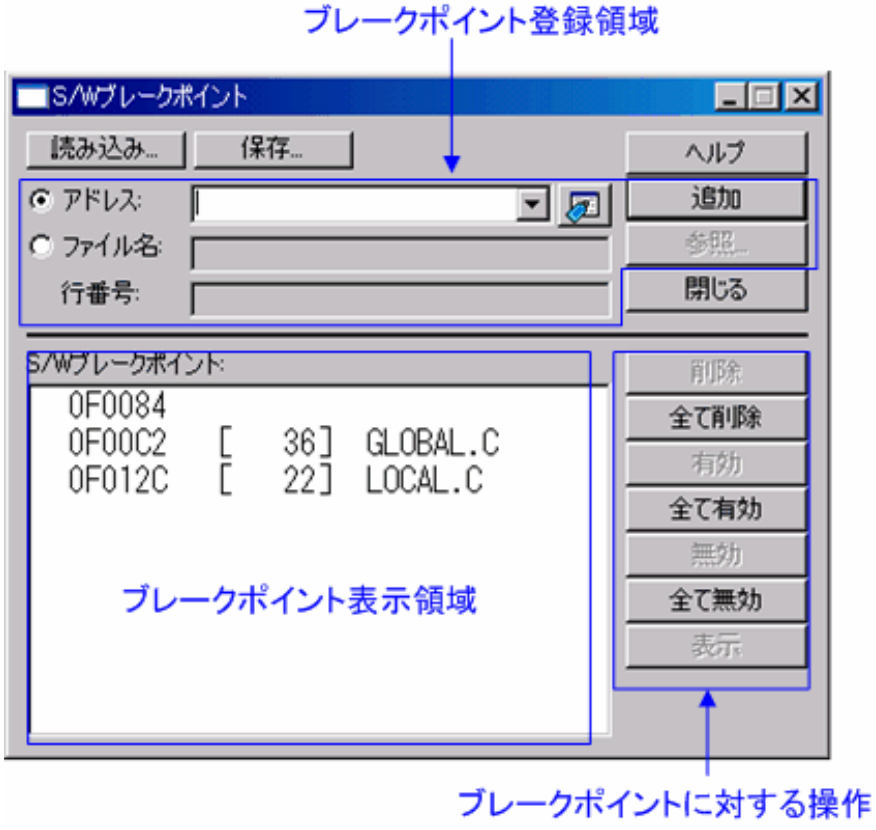
ウィンドウ内でマウスの右ボタンをクリックすると以下のポップアップメニューを表示します。 これらのメニューの主要な機能は、ツールバーのボタンにも割り付けられています。

メニュー名		機能
スクリプト	開く...	スクリプトファイルを開きます。
	実行	スクリプトファイルを実行します。
	ステップ実行	スクリプトファイルをステップ実行します。
	閉じる	スクリプトファイルを閉じます。
表示	保存...	実行結果表示をファイルに保存します。
	消去	実行結果表示を消去します。
ログ	開始...	ログファイルを開き出力を開始します。
	停止	出力を終了しログファイルを閉じます。
コマンドの記録	開始...	コマンドのファイルへの記録を開始します。
	停止	コマンドのファイルへの記録を停止します。
ツールバー表示		ツールバーの表示/非表示を切り換えます。
ツールバーのカスタマイズ...		ツールバーをカスタマイズします。
ドッキングビュー		ウィンドウをドッキングします。
非表示		ウィンドウを非表示にします。

5.5 S/W ブレークポイント設定ウィンドウ

S/W ブレークポイント設定ウィンドウは、ソフトウェアブレークポイントを設定するためのウィンドウです。

ソフトウェアブレークは、指定アドレスの命令を実行する手前でブレークします。



- ブレークポイントは、"アドレス"または"ファイル名+行番号"で指定できます。
- ブレークポイントを複数設定した場合、いずれか１点の ブレークポイントに到達するとターゲットプログラムを停止します(OR 条件)。
- 各ブレークポイントに対して、削除、無効/有効を切り換えることができます。
- ブレークポイント情報は、ファイルに保存することができます。保存したブレークポイント情報を読み込むことも可能です。

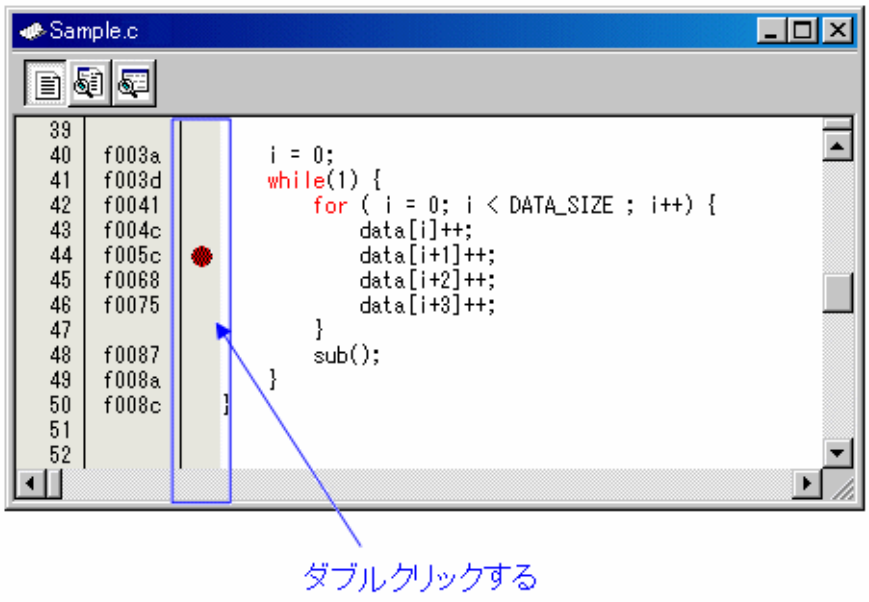
5.5.1 コマンドボタン

ウィンドウ上の各ボタンは、以下の意味を持っています。

ボタン名	機能
読み込み...	ファイルに保存した設定内容を読み込みます。
保存...	ウィンドウで設定した内容をファイルに保存します。
ヘルプ	ヘルプを表示します。
追加	ソフトウェアブレークポイントを設定します。
参照...	ソースファイルを指定します。
閉じる	ウィンドウを閉じます。
削除	選択したソフトウェアブレークポイントを解除します。
全て削除	全てのソフトウェアブレークポイントを解除します。
有効	選択したソフトウェアブレークポイントを有効にします。
全て有効	全てのソフトウェアブレークポイントを有効にします。
無効	選択したソフトウェアブレークポイントを無効にします。
全て無効	全てのソフトウェアブレークポイントを無効にします。
表示	選択したソフトウェアブレークポイントの位置をエディタ(ソース)ウィンドウに表示します。

5.5.2 エディタ(ソース)ウィンドウからブレークポイントを設定/解除する

エディタ(ソース)ウィンドウの S/W ブレークポイント設定用カラム上で、ブレークポイントを設定する行をダブルクリックして下さい (設定行に赤丸が表示されます)。

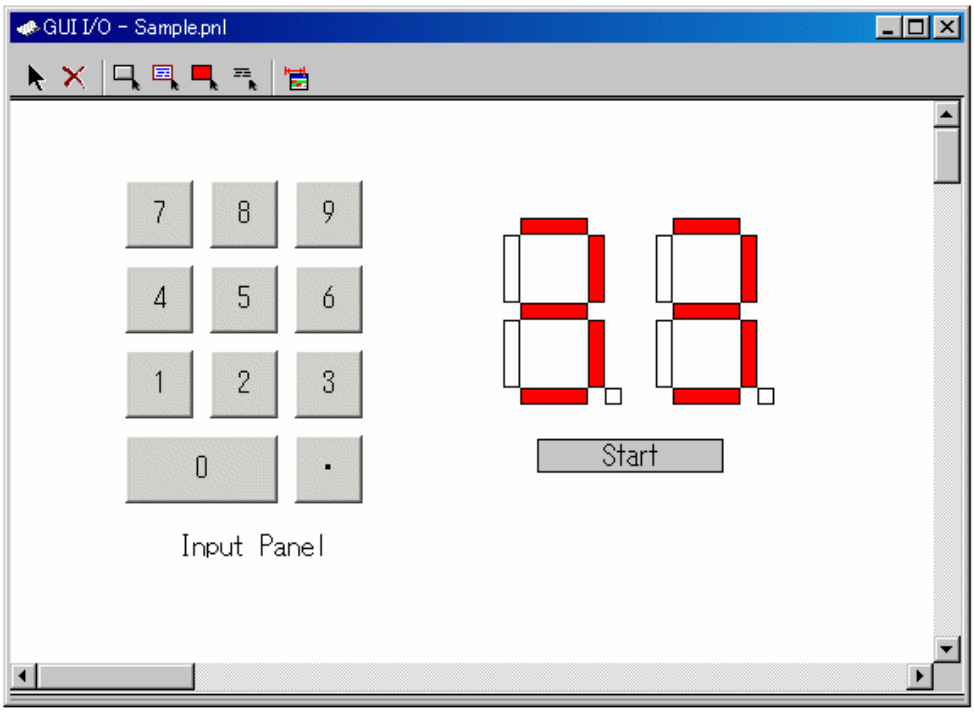


もう一度ダブルクリックするとブレークポイントの設定解除となります(赤丸の表示が消えます)。

エディタ(ソース)ウィンドウには､S/W ブレークポイント設定用カラムがデフォルトで表示されています。非表示にするには、メニュー[編集]→[表示カラムの設定...]で開くダイアログで、 [S/W ブレークポイント]チェックボックスをオフにしてください。 全てのエディタ(ソース)ウィンドウの、S/W ブレークポイント設定用のカラムが非表示になります。また、エディタ(ソース)ウィンドウのポップアップメニュー[カラム]→[S/W ブレークポイント]を選択することで、 個々のエディタ(ソース)ウィンドウ毎にカラムを設定することもできます。

5.6 GUI 入出力ウィンドウ

GUI 入出力ウィンドウは、仮想的な入出力パネルを作成できるウィンドウです。 ウィンドウ上に仮想のボタンを配置して入力したり、仮想 LED を配置してそこに出力したりできます。



- ウィンドウ上には、次のアイテムが配置できます。
- ラベル
  指定したアドレス(もしくはビット)に指定した値が書き込まれた際に、文字列を表示/消去します。
- LED
  指定したアドレス(もしくはビット)に指定した値が書き込まれた際に、指定した色で表示します(LED 点灯の代用)。
- ボタン
  押下することにより、仮想ポートへの入力が行えます。
- テキスト
  テキスト文字列を表示します。
- 作成した入出力パネルをファイル(入出力パネルファイル)に保存し、再読み込みすることもできます。
- 作成したアイテムに設定できるアドレスは、最大 200 点です。 各アイテムに設定したアドレスがすべて異なる場合、配置できるアイテム数は 200 個になります。

5.6.1 オプションメニュー

ウィンドウ内でマウスの右ボタンをクリックすると以下のポップアップメニューを表示します。 これらのメニューの主要な機能は、ツールバーのボタンにも割り付けられています。

メニュー名	機能
アイテムの選択	クリックしたアイテムを選択状態にします。
削除	クリックしたアイテムを削除します。
コピー	クリックしたアイテムをコピーします。
貼り付け	コピーしたアイテムを貼り付けます。
ボタンの作成	新規にボタンを作成します。
ラベルの作成	新規にラベルを作成します。
LED の作成	新規に LED を作成します。
テキストの作成	新規にテキストを作成します。
グリッドの表示	グリッドを表示します。
保存...	入出力パネルファイルを保存します。
読み込み...	入出力パネルファイルを読み込みます。
サンプリング周期...	表示更新間隔を設定します。
ツールバー表示	ツールバーの表示/非表示を切り換えます。
ツールバーのカスタマイズ...	ツールバーをカスタマイズします。
ドッキングビュー	ウィンドウをドッキングします。
非表示	ウィンドウを非表示にします。

6. スクリプトコマンド一覧

本デバッガでは、以下のスクリプトコマンドが使用できます。
網掛け表示しているスクリプトコマンドは、ランタイム実行可能です。
後ろに*の付いたコマンドは、製品によってはサポートしていません。

なお、各コマンドの詳細な説明は、本デバッガのヘルプをご参照下さい。

6.1 スクリプトコマンド一覧（機能順）

6.1.1 実行関連

コマンド名	短縮名	内容
Go	G	ターゲットプログラムの実行
GoFree	GF	ターゲットプログラムのフリーラン実行
Stop	–	ターゲットプログラムの停止
Status	–	ターゲットプログラムの実行状態表示
Step	S	ソースレベルのステップ実行
StepInstruction	SI	機械語レベルのステップ実行
OverStep	O	ソースレベルのオーバーステップ実行
OverStepInstruaction	OI	機械語レベルのオーバーステップ実行
Return	RET	ソースレベルのリターン実行
ReturnInstruction	RETI	機械語レベルのリターン実行
Reset	–	ターゲットプログラムのリセット
Time	–	実行時間表示の設定

6.1.2 ダウンロード関連

コマンド名	短縮名	内容
Load	L	ターゲットプログラムの一括ダウンロード
LoadHex	LH	機械語情報(インテルHEXフォーマットファイル)のダウンロード
LoadMot*	LM	機械語情報(モトローラSフォーマットファイル)のダウンロード
LoadSymbol	LS	ソース行/アセンブラシンボル情報のダウンロード
Reload	–	ターゲットプログラムの再ダウンロード
UploadHex	UH	機械語情報のインテルHEXフォーマットファイルへのアップロード
UploadMot*	UM	機械語情報のモトローラSフォーマットファイルへのアップロード

6.1.3 レジスタ操作関連

コマンド名	短縮名	内容
Register	R	指定レジスタの値を参照

6.1.4 メモリ操作関連

コマンド名	短縮名	内容
DumpByte	DB	メモリ内容の1バイト単位表示
DumpWord*	DW	メモリ内容の2バイト単位表示
DumpLword*	DL	メモリ内容の4バイト単位表示
SetMemoryByte	MB	メモリ内容の1バイト単位変更
SetMemoryWord*	MW	メモリ内容の2バイト単位変更
SetMemoryLword*	ML	メモリ内容の4バイト単位変更
FillByte	FB	メモリ内容の1バイト単位充填
FillWord*	FW	メモリ内容の2バイト単位充填
FillLword*	FL	メモリ内容の4バイト単位充填
Move	–	メモリ内容の1バイト単位転送
MoveWord*	MOVEW	メモリ内容の2バイト単位転送

6.1.5 アセンブル/逆アセンブル関連

コマンド名	短縮名	内容
Assemble	A	指定したアドレスから1行単位でアセンブル
DisAssemble	DA	指定した範囲の逆アセンブル結果を表示
Module	MOD	全モジュール(オブジェクト名)を表示
Scope	–	現在のスコープ表示/スコープの変更
Section	SEC	セクション情報を表示
Bit*	–	ビットシンボルの参照/設定
Symbol	SYM	シンボルの表示
Label	–	ラベルの表示
Express	EXP	指定したアセンブラ式の値を表示

6.1.6 ソフトウェアブレーク設定関連

コマンド名	短縮名	内容
SoftwareBreak	SB	ソフトウェアブレークポイントの表示/設定
SoftwareBreakClear	SBC	ソフトウェアブレークポイントの削除
SoftwareBreakClearAll	SBCA	全ソフトウェアブレークポイントの削除
SoftwareBreakDisable	SBD	ソフトウェアブレークポイントの無効化
SoftwareBreakDisableAll	SBDA	全ソフトウェアブレークポイントの無効化
SoftwareBreakEnable	SBE	ソフトウェアブレークポイントの有効化
SoftwareBreakEnableAll	SBEA	全ソフトウェアブレークポイントの有効化
BreakAt	–	行番号レベルのソフトウェアブレークポイント指定
BreakIn	–	関数の先頭にソフトウェアブレークポイントを指定

6.1.7 スクリプト/ログファイル関連

コマンド名	短縮名	内容
Script	–	スクリプトファイルのオープン
Exit	–	スクリプトファイルのクローズ
Wait	–	コマンド入力待機
Pause	–	指定メッセージを表示し、ボタン入力待ち
Sleep	–	指定秒数のコマンド入力待機
Logon	–	ログファイルのオープン
Logoff	–	ログファイルのクローズ

6.1.8 プログラム表示関連

コマンド名	短縮名	内容
Func	–	関数名の参照/関数内容の表示
Up*	–	呼び出し元関数の表示
Down*	–	呼び出し先関数の表示
Where*	–	関数の呼び出し状況の表示
Path	–	ソースファイルのパス指定
AddPath	–	ソースファイルのパス指定の追加
File	–	指定ソースファイルの表示

6.1.9 マップ関連

コマンド名	短縮名	内容
Map	–	マップの参照/設定

6.1.10 供給クロック関連

コマンド名	短縮名	内容
Clock	CLK	MCUの供給クロック設定/参照

6.1.11 C 言語関連

コマンド名	短縮名	内容
Print	–	C言語変数式の参照
Set	–	C言語変数式へのデータ指定

6.1.12 リアルタイム OS 関連

コマンド名	短縮名	内容
MR*	–	リアルタイムOS(MRxx)の状態表示

6.1.13 ユーティリティ関連

コマンド名	短縮名	内容
Radix	–	定数の既定値設定/参照
Alias	–	コマンドの別名定義/定義状況の参照
UnAlias	–	コマンドの別名定義削除
UnAliasAll	–	全コマンドの別名定義削除
Help	H	スクリプトコマンドのヘルプ表示
Version	VER	デバッガのバージョン表示
Date	–	現在の日時表示
Echo	–	メッセージの表示
CD	–	カレントディレクトリの設定/参照

6.2 スクリプトコマンド一覧（アルファベット順）

コマンド名	短縮名	内容
AddPath	–	ソースファイルのパス指定の追加
Alias	–	コマンドの別名定義/定義状況の参照
Assemble	A	指定したアドレスから1行単位でアセンブル
Bit*	–	ビットシンボルの参照/設定
BreakAt	–	行番号レベルのソフトウェアブレークポイント指定
BreakIn	–	関数の先頭にソフトウェアブレークポイントを指定
CD	–	カレントディレクトリの設定/参照
Clock	CLK	MCUの供給クロック設定/参照
Date	–	現在の日時表示
DisAssemble	DA	指定した範囲の逆アセンブル結果を表示
Down*	–	呼び出し先関数の表示
DumpByte	DB	メモリ内容の1バイト単位表示
DumpLword*	DL	メモリ内容の4バイト単位表示
DumpWord*	DW	メモリ内容の2バイト単位表示
Echo	–	メッセージの表示
Exit	–	スクリプトファイルのクローズ
Express	EXP	指定したアセンブラ式の値を表示
File	–	指定ソースファイルの表示
FillByte	FB	メモリ内容の1バイト単位充填
FillLword*	FL	メモリ内容の4バイト単位充填
FillWord*	FW	メモリ内容の2バイト単位充填
Func	–	関数名の参照/関数内容の表示
Go	G	ターゲットプログラムの実行
GoFree	GF	ターゲットプログラムのフリーラン実行
Help	H	スクリプトコマンドのヘルプ表示
Label	–	**ラベルの表示**
Load	L	ターゲットプログラムの一括ダウンロード
LoadHex	LH	機械語情報(インテルHEXフォーマットファイル)のダウンロード
LoadMot*	LM	機械語情報(モトローラSフォーマットファイル)のダウンロード
LoadSymbol	LS	ソース行/アセンブラシンボル情報のダウンロード
Logoff	–	ログファイルのクローズ
Logon	–	ログファイルのオープン
Map	–	マップの参照/設定
Module	MOD	全モジュール(オブジェクト名)を表示
Move	–	メモリ内容の1バイト単位転送
MoveWord*	MOVEW	メモリ内容の2バイト単位転送
MR*	–	リアルタイムOS状態表示
OverStep	O	ソースレベルのオーバーステップ実行
OverStepInstruaction	OI	機械語レベルのオーバーステップ実行
Path	–	ソースファイルのパス指定
Pause	–	指定メッセージを表示し、ボタン入力待ち
Print	–	C言語変数式の参照

Radix	–	定数の既定値設定/参照
Register	R	指定レジスタの値を参照
Reload	–	ターゲットプログラムの再ダウンロード
Reset	–	ターゲットプログラムのリセット
Return	RET	ソースレベルのリターン実行
ReturnInstruction	RETI	機械語レベルのリターン実行
Scope	–	現在のスコープ表示/スコープの変更
Script	–	スクリプトファイルのオープン
Section	SEC	セクション情報を表示
Set	–	C言語変数式へのデータ指定
SetMemoryByte	MB	メモリ内容の1バイト単位変更
SetMemoryLword*	ML	メモリ内容の4バイト単位変更
SetMemoryWord*	MW	メモリ内容の2バイト単位変更
Sleep	–	指定秒数のコマンド入力待機
SoftwareBreak	SB	ソフトウェアブレークポイントの表示/設定
SoftwareBreakClear	SBC	ソフトウェアブレークポイントの削除
SoftwareBreakClearAll	SBCA	全ソフトウェアブレークポイントの削除
SoftwareBreakDisable	SBD	ソフトウェアブレークポイントの無効化
SoftwareBreakDisableAll	SBDA	全ソフトウェアブレークポイントの無効化
SoftwareBreakEnable	SBE	ソフトウェアブレークポイントの有効化
SoftwareBreakEnableAll	SBEA	全ソフトウェアブレークポイントの有効化
Status	–	ターゲットプログラムの実行状態表示
Step	S	ソースレベルのステップ実行
StepInstruction	SI	機械語レベルのステップ実行
Stop	–	ターゲットプログラムの停止
Symbol	SYM	シンボルの表示
Time	–	実行時間表示の設定
UnAlias	–	コマンドの別名定義削除
UnAliasAll	–	全コマンドの別名定義削除
Up*	–	呼び出し元関数の表示
UploadHex	UH	機械語情報のインテルHEXフォーマットファイルへのアップロード
UploadMot*	UM	機械語情報のモトローラSフォーマットファイルへのアップロード
Version	VER	デバッガのバージョン表示
Wait	–	コマンド入力待機
Where*	–	関数の呼び出し状況の表示

7. スクリプトファイルの記述

スクリプトファイルは、スクリプトコマンドを自動実行するために、その制御文などを記述したファイルです。
スクリプトファイルは、スクリプトウィンドウで実行します。

7.1 スクリプトファイルの構成要素

スクリプトファイルには、以下の文が記述できます。

- **スクリプトコマンド**
- **代入文**
- **判断文(if,else,endi)**
  式の結果を判断して、実行する文を分岐します。
- **繰り返し文(while,endw)**
  式の結果を判断して、文を繰り返し実行します。
- **break 文**
  最も内側の繰り返し実行から抜けます。
- **コメント文**
  スクリプトファイルにコメント(注釈)を記述できます。スクリプトコマンド実行の際、コメント文は無視されます。

スクリプトファイルには、一行につき 1 つの文を記述してください。一行に複数の文を記述したり、1 つの文を複数行にまたがって記述することはできません。

注意事項

- スクリプトコマンドのコメントとして同一行に記述することはできません。
- スクリプトファイルのネストは 10 段までです。
- if 文と while 文のネストはそれぞれ 32 段までです。
- 一つのスクリプトファイルで if と endi 文､while と endw が対になっていなければいけません。
- スクリプトファイルに記述する式は、unsigned 型で計算します。したがって、if 文、while 文の式で負の値を比較した場合の動作は不定になります。
- 1 行に記述できる文字数は、4096 文字までです。これを越える行を実行した場合、エラーになります。
- 不適当な記述のあるスクリプトファイルを自動実行した場合、スクリプト行自身が読み込めない場合を除いて、エラー検出後もスクリプトファイルの終わりまで実行処理は続けられます。ただしこの場合、エラー検出後の動作は不定であり、したがってエラー検出後の実行結果は信頼性がありません。

スクリプトコマンド
スクリプトウィンドウで入力するコマンドを、そのまま記述することができます。
またスクリプトファイルからスクリプトファイルを呼び出すこともできます(ネストは 10 段まで)。

代入文

代入文は、マクロ変数の定義や初期化、および代入を行います。以下に記述書式を示します。

```
%マクロ変数名 = 式
```

- マクロ変数名には、英数字と'_'が使用できます。ただし、マクロ変数名の先頭には、数字を記述することはできません。
- マクロ変数に代入する式が扱える値の範囲は、0h から FFFFFFFFh までの整数です。
- 負の数を指定した場合は 2 の補数として扱います。
- マクロ変数は、式の中で使用することができます。
- マクロ変数は、先頭に'%'を付加して使用します。

判断文

判断文は、式の結果を判断し、実行する文を分岐します。以下に記述書式を示します。

```
if ( 式 )
      文 1
else
      文 2
endi
```

- 式が真（0 以外）のとき文 1 を実行します。式が偽（0）のとき文 2 を実行します。
- else 文は省略することができます。else 文を省略時に式が偽の場合、endi 文の次の行から実行します。
- if 文は、32 段までネストすることができます。

繰り返し文(while,endw)とbreak文

繰り返し文は、式の結果を判断し、文を繰り返し実行します。以下に記述書式を示します。

```
while ( 式 )
      文
endw
```

- 式が真の場合、文を繰り返し実行します。式が偽の場合、ループから抜けます（endw の次の文から実行します）。
- while 文は、32 段までネストすることができます。
- while 文を強制的の抜ける場合は、break 文を使用します。while 文がネストしている場合は、最も内側のループから抜けます。

コメント文

コメント文は、スクリプトファイルにコメント（注釈）を記述する場合に使用します。以下に記述書式を示します。

```
; 文字列
```

- セミコロン (';') から文を記述します。セミコロンの前には、空白文字とタブのみ記述可能です。
- コメント文の行は、スクリプトファイル実行時に無視されます。

7.2 式の記述方法

アドレス、データ、通過回数などの指定に式を記述することができます。
以下に式を使用したコマンド例を示します。

```
>DumpByte TABLE1
>DumpByte TABLE1+20
```

式の構成要素としては、以下のものが使用できます。

- 定数
- シンボル、ラベル
- マクロ変数
- レジスタ変数
- メモリ変数
- 行番号
- 文字定数
- 演算子

定数

2 進数、8 進数、10 進数、16 進数が入力可能です。数値の基数は、数値の先頭または、末尾に基数を示す記号を付けて区別します。

<M32C 用デバッガ、M16C/R8C 用デバッガの場合>

	16 進数	10 進数	8 進数	2 進数*
先頭	0x,0X	@	なし	%
末尾	h , H	なし	o, O	b ,B
例	0xAB24 AB24h	@1234	1234o	%10010 10010b

* 基数の既定値が 16 進数のときは、'%'のみ指定可能

- 既定値と同じ基数で入力する場合は、基数を示す記号は省略可能です（2 進数は除く）。
- 基数の既定値は、RADIX コマンドで設定します。ただし、以下のデータに関する入力を行う場合は、RADIX コマンドの設定に関係なく、基数は固定です。

種別	基数
アドレス	16 進
行番号 実行回数 通過回数	10 進

シンボル、ラベル

ターゲットプログラムで定義しているシンボル/ラベル、および Assemble コマンドで定義したシンボル/ラベルが使用できます。

- シンボル/ラベル名には、英数字、アンダスコア('_')、ピリオド('.')、クエスチョンマーク('?')が使用可能です。ただし、先頭文字に数字は使用できません。
- シンボル／ラベル名は、255 文字まで記述できます。
- 大文字／小文字は区別します。

製品名	注意事項
M32R 用デバッガ, M32C 用デバッガ, M16C/R8C 用デバッガ	• アセンブラの構造化命令、擬似命令、マクロ命令、オペコード、予約語は使用できません。 （.SECTION, .BYTE, switch, if など） • ".."で始まる文字列は、シンボル／ラベル名には使用できません。

ローカルラベルシンボルとスコープ

プログラムの全領域から参照可能なグローバルラベルシンボルと、宣言したファイル内でのみ参照可能なローカルラベルシンボルの 2 種類をサポートしています。
ローカルラベルシンボルの有効範囲をスコープといいます。スコープの単位は、オブジェクト(リロケータブル)ファイルです。
下記の場合に応じて、スコープを切り替えます。

- **コマンド入力時**
  プログラムカウンタが示すアドレスを含むオブジェクトファイルが、現在のスコープとなります。また SCOPE コマンドでスコープを設定した場合、設定したスコープが有効になります。
- **コマンド実行中**
  コマンドが扱うプログラムアドレスによって現在のスコープを自動的に切り替えます。

ラベル/シンボルの優先順位

値からラベル／シンボルへの変換、ラベル／シンボルから値への変換は、下記の優先順位で行います。

- アドレス値を変換する場合
  1. ローカルラベル
  2. グローバルラベル
  3. ローカルシンボル
  4. グローバルシンボル
  5. スコープ範囲外のローカルラベル
  6. スコープ範囲外のローカルシンボル

- データ値を変換する場合
  1. ローカルシンボル
  2. グローバルシンボル
  3. ローカルラベル
  4. グローバルラベル
  5. スコープ範囲外のローカルシンボル
  6. スコープ範囲外のローカルラベル

- ビット値を変換する場合
  1. ローカルビットシンボル
  2. グローバルビットシンボル
  3. スコープ範囲外のローカルビットシンボル（M16C/R8C 用デバッガを除く）

マクロ変数

マクロ変数は、スクリプトファイル中の代入文で定義します。マクロ変数は、変数名の先頭に' % 'を付加して使用します。
詳細については、「7.1 スクリプトファイルの構成要素」の「代入文」を参照してください。

- パーセント文字（' % '）の後の変数名には、英数字と'_'が使用可能です。ただし、マクロ変数名の先頭には、数字を記述することはできません。
- 変数名には、レジスタ名は使用できません。
- 変数名の大文字／小文字を区別します。
- マクロ変数は、255 個まで定義できます。一度定義したマクロ変数は、デバッガを終了するまで有効です。

マクロ変数は、while 文の繰り返し回数を指定する際に利用すると便利です。

レジスタ変数

レジスタの値を式中で利用する場合に使用します。レジスタ変数は、レジスタ名の前に'%'を付加します。
以下に使用できるレジスタ名を示します。

製品名	レジスタ名
M16C/R8C 用デバッガ	PC, USP, ISP, SB, INTB, FLG 0R0, 0R1, 0R2, 0R3, 0A0, 0A1, 0FB←レジスタバンク 0 1R0, 1R1, 1R2, 1R3, 1A0, 1A1, 1FB←レジスタバンク 1

レジスタ名の大文字／小文字は区別しません。どちらで指定しても結果は同じです

メモリ変数

メモリの値を式中で利用する際に使用します。メモリ変数の書式を以下に示します。

［アドレス］.データサイズ

- アドレスには、式が記述できます（メモリ変数も指定可能）。
- データサイズは、以下のように指定します 。

データ長	対応デバッガ	指定
1 バイト	すべて	B または b
2 バイト	M32R 用デバッガ	H または h
	その他	W または w
4 バイト	M32R 用デバッガ	W または w
	M32C 用デバッガ、M16C/R8C 用デバッガ	L または l

例：8000h 番地のメモリ内容を 2 バイト長で参照する場合
[0x8000].w

- データサイズの指定を省略した場合、ワード長を指定したことになります。

行番号

ソースファイルの行番号です。行番号の書式を以下に示します。

```
#行番号
#行番号."ソースファイル名"
```

- 行番号は、10 進数で指定します。
- 行番号に指定できるのは、ソフトウェアブレークが設定できる行だけです。コメント行や空白行などのアセンブラの命令が生成されない行を指定することはできません。
- ソースファイル名を省略した場合、現在フォーカスがあるエディタ(ソース)ウィンドウに表示しているソースファイルの行番号になります。
- ソースファイル名は、ファイル属性も指定してください。
- 行番号とソースファイル名の間に空白文字を挿入することはできません。

文字定数

指定された文字または文字列を ASCII コードに変換し、定数として扱います。

- 文字は、シングルクォーテーションで囲みます。
- 文字列は、ダブルクォーテーションで囲みます。
- 文字列は 2 文字以内（16 ビット長）でなければなりません。 2 文字を越えた場合も、記述した文字列の最後の 2 文字が処理の対象となります。例えば、"ABCD" と記入した場合、文字列の最後の 2 文字 "CD" が処理対象となり、値は 4344h となります。

演算子

式に記述可能な演算子を以下に示します。

- 演算子の優先度は、レベル 1 が最も高く、レベル 8 が最も低くなります。優先順位が同じ場合は、式の左から順番に計算します。

演算子	意味	優先度
( )	括弧	レベル 1
+, -, ~	単項正、単項負、単項論理否定	レベル 2
*, /	二項乗算、二項除算	レベル 3
+, -	二項加算、二項減算	レベル 4
>>,	右シフト、左シフト	レベル 5
&	二項論理積	レベル 6
|, ^	二項論理和、二項排他的論理和	レベル 7
<, <=, >, >=, ==, !=	二項比較	レベル 8

8. C/C++言語式の記述

8.1 C/C++言語式の記述方法

C ウォッチポイントの登録、及び C ウォッチポイントに代入する値の指定には、以下の字句(トークン)で構成された C/C++言語式が使用できます。

字句(トークン)	例
即値	10, 0x0a, 012, 1.12, 1.0E+3
スコープ解決	::name, classname::member
四則演算子	+, -, *, /
ポインタ	*, **, ...
参照	&
符号反転	-
"."演算子によるメンバ参照	Object.Member
"->"演算子によるメンバ参照	Pointer->Member, this->Member
メンバへのポインタ参照	Object.*var, Pointer->*var
括弧	(,)
配列	Array[2], DArray[2] [3] , ...
基本型へのキャスト	(int), (char*), (unsigned long *), ...
typedef された型へのキャスト	(DWORD), (ENUM), ...
変数名および関数名	var, i, j, func, ...
文字定数	'A', 'b', ...
文字列リテラル	"abcdef", "I am a boy.", ...

即値

即値としては、16 進数、10 進数、および 8 進数が使用できます。 0x で始まれば 16 進数、0 で始まれば 8 進数として認識します。それ以外の数値は、10 進数として認識します。また、変数に値を代入する場合、浮動小数点数値も使用できます。

注意

- 即値を C ウォッチポイントとして登録することはできません。
- 即値は、C ウォッチポイントを指定する C 言語式の中に用いる場合、および代入する値を指定する場合にのみ有効です。浮動小数点数値を使用する場合、1.0+2.0 等の演算はできません。

スコープ解決

スコープ解決演算子(::)が使用できます。以下に使用例を示します。

大域スコープ： ::変数名

```
::x, ::val
```

クラス指定： クラス名::メンバ名、クラス名::クラス名::メンバ名 等

```
T::member, A::B::member
```

四則演算子

四則演算子は、加算(+), 減算(-), 乗算(*), 除算(/)が使用できます。以下に、計算の優先順位を示します。

(*), (/), (+), (-)

注意

- 浮動小数点に対する四則計算は、現在サポートしておりません。

ポインタ

ポインタは、*で表され、ポインタのポインタ**、ポインタのポインタのポインタ ***、・・・が使用できます。
「*変数名」、「**変数名」、・・・という記述で使用します。

注意

- 即値をポインタとして扱うことはできません。つまり、*0xE000 などは、使用することができません。

参照

参照は、&で表され、「&変数名」のみが使用できます。「&&変数名」等は使用することができません。

符号反転

符号反転は、-で表され、「-即値」、「-変数名」のみが使用できます。 -を 2 つ以上偶数個続けた場合には、符号反転は行なわれません。

注意

- 浮動小数点変数に対する符号反転は、現在サポートしておりません。

"."演算子によるメンバ参照

"."演算子によるクラス、構造体、共用体のメンバ参照は、「変数名.メンバ名」のみが使用できます。

(例)

```
class T {
public:
int member1;
char member2;
};
class T t_cls;
class T *pt_cls = &t_cls;
```

この場合、t_cls.member1、(*pt_cls).member2 は、正しくメンバを参照することができます。

メンバへのポインタ

".*"演算子や"->*"演算子によるメンバへのポインタ参照は、「変数名.*メンバ名」、「変数名->*メンバ名」のみが使用できます。

(例)

```
class T {
public:
int member;
};
class T t_cls;
class T *pt_cls = &t_cls;

int T::*mp = &T::member;
```

この場合、t_cls.*mp、pt_cls->*mp は、正しくメンバを参照することができます。

注意

print *mp という記述では、メンバへのポインタ変数を正しく参照できません。

括弧

式の途中に、計算の優先順位を指定する括弧として、 '('と')'を使用することができます。

配列

配列の要素を指定する表現に '['と']'を使用することができます。配列は、「変数名[(要素番号または変数)]」、「変数名[(要素番号または変数)][(要素番号または変数)]」、・・・という記述で使用します。

基本型へのキャスト

C の基本型のうち、char 型、short 型、int 型、long 型へのキャスト、およびこれらの基本型へのポインタ型へのキャスト演算が使用できます。ポインタ型へのキャストは、ポインタのポインタ、ポインタのポインタのポインタ、・・・なども使用できます。なお、signed 、unsigned の指定がない場合のデフォルトは、以下のとおりです。

基本型	デフォルト
char	unsigned
short	signed
int	signed
long	signed

注意

- C++の基本型のうち、bool 型、wchar_t 型、浮動小数点型(float、double 型)へのキャストは使用できません。
- レジスタ変数に対するキャストは使用できません。

typedefされた型へのキャスト

typedef された型(C/C++の基本型以外の型)、およびそれらへのポインタ型へのキャスト演算が使用できます。ポインタ型へのキャストは、ポインタのポインタ、ポインタのポインタのポインタ、・・・なども使用できます。

注意

- class 型、struct 型、union 型、およびそれらのポインタ型へのキャストは使用できません。

変数名

変数名は、C/C++の規約通りアルファベットで始まる文字列が使用できます。最大文字数は、255 文字です。また、this ポインタ変数を使用することができます。

関数名

関数名は、C の規約通りアルファベットで始まる文字列が使用できます。C++の場合、関数名は使用できません。

文字定数

文字定数として、シングルクォーテーション(')で囲まれた文字が使用できます。例えば、'A'、'b'等です。これらは、ASCII コードに変換され、1 バイトの即値として使用されます。

注意

- 文字定数を C ウォッチポイントとして登録することはできません。
- C ウォッチポイントを指定する C/C++言語式の中に用いる場合、および代入する値を指定する場合にのみ有効です（文字定数は即値と同じ扱いになります）。

文字列リテラル

文字列リテラルとして、ダブルクォーテーション(")で囲まれた文字列が使用できます。例えば、"abcde"、"I am a boy."等です。

注意

- 文字列リテラルは、右辺式(代入演算子の右辺)にのみ記述することができ、左辺式(代入演算 子の左辺)が char 配列、または char ポインタ型の場合にのみ使用することができます。 それ以外の場合には、文法エラーとなります。

8.2 C/C++言語式の表示形式

C ウォッチウィンドウのデータ表示領域における C/C++言語式の表示は、その型名、C/C++言語式(変数名)、計算結果(値)から構成されています。 以下に、型別に表示形式を説明します。

列挙型の場合

- 計算結果の値が定義されているものであれば、その名前で表示します。

```
(DATE) date = Sunday                    (全Radix)
```

- 計算結果の値が定義されているものでなかった場合には、以下のように表示します。

```
(DATE) date = 16                        (Radixが初期状態の場合)
(DATE) date = 0x10                      (Radixが16進数の場合)
(DATE) date = 0000000000010000B         (Radixが2進数の場合)
```

基本型の場合

- 計算結果が char 型および浮動小数点以外の基本型の場合には、以下のように表示します。

```
(unsigned int) i = 65280                    (Radixが初期状態の場合)
(unsigned int) i = 0xFF00                   (Radixが16進数の場合)
(unsigned int) i = 1111111100000000B        (Radixが2進数の場合)
```

- 計算結果が char 型の場合には、以下のように表示します。

```
(unsigned char) c = 'J'                     (Radixが初期状態の場合)
(unsigned char) c = 0x4A                    (Radixが16進数の場合)
(unsigned char) c = 10100100B               (Radixが2進数の場合)
```

- 計算結果が浮動小数点の場合には、以下のように表示します。

```
(double) d = 8.207880399131839E-304         (Radixが初期状態の場合)
(double) d = 0x10203045060708               (Radixが16進数の場合)
(double) d = 0000000010.....1000B           (Radixが2進数の場合)
(...は省略を表す)
```

ポインタ型の場合

- 計算結果が char*型以外のポインタ型の場合には、以下のように内容を 16 進数表示します。

```
(unsigned int *) p = 0x1234                 (全Radix)
```

- 計算結果が char*型の場合には、C ウォッチウィンドウのメニュー［char*の文字列表示］で文字列／文字の表示が指定できます。表示例を以下に示します。

文字列表示の場合

```
(unsigned char *) str = 0x1234 "Japan"          (全Radix)
```

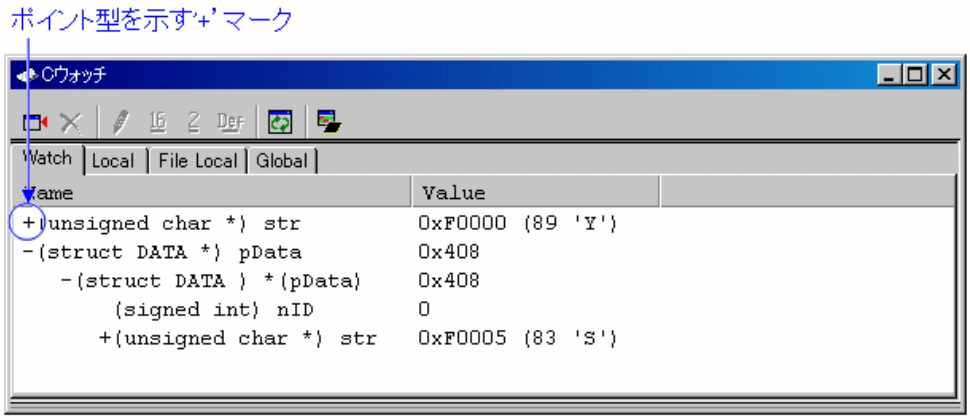
文字表示の場合

```
(unsigned char *) str = 0x1234 (74 'J')         (全Radix)
```

文字列表示の場合、文字列の終わりを表すコード(0)までに、文字表示できないコードが格納されていた場合には、以下のように、閉じ(")を出力しません。

```
(unsigned char *) str = 0x1234 "Jap
```

また、文字列の長さが 80 文字を越えた場合も同様に、閉じ(")を出力しません。
なお、C/C++言語式がポインタ型の場合は、以下に示すように、型名の左側に'+'マークが現れます。



この'+'マークが表示されている行をダブルクリックすると、そのポインタのオブジェクトが現れます。オブジェクトを表示すると、'+'マークは'-'マークにかわります。なお、'-'マークが表示されている行をダブルクリックすると、もとの状態に戻ります。このようにして、リスト構造やツリー構造等のデータも参照することができます。

配列型の場合

- 計算結果が char[]型以外の配列型の場合には、以下のように先頭アドレスを 16 進数表示します。

```
(signed int [10]) z = 0x1234                    (全Radix)
```

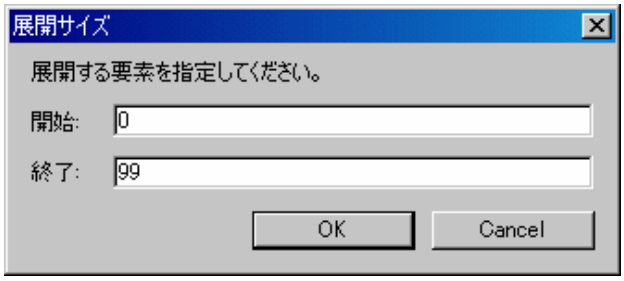
- 計算結果が char[]型の場合には、以下のように表示します。

```
(unsigned char [10]) str = 0x1234 "Japan"   (全Radix)
```

文字列の終わりを表すコード(0)までに、文字表示できないコードが格納されていた場合には、以下のように、閉じ(")を出力しません。

```
(unsigned char [10]) str = 0x1234 "Jap              (全Radix)
```

また、文字列の長さが 80 文字を越えた場合も同様に、閉じ(")を出力しません。
なお、C/C++言語式が配列型の場合は、ポインタ型と同様、型名の左側に'+'マークが現れます。展開方法は、ポインタ型と同じです。詳細な説明については、「ポインタ型の場合」をご参照下さい。
配列のサイズが 100 以上の場合、下記ダイアログがオープンするので、展開する要素数を指定してください。



Start で指定した要素から End で指定した要素までを表示します。
配列の要素数の最大値を超える値を指定した場合は、配列の最大値を指定した事になります。
なお、Cancel ボタンを押下した場合、配列は展開しません。

関数型の場合

- 計算結果が関数型の場合には、以下のように関数の開始アドレスを 16 進数表示します。

```
(void()) main = 0xF000                          (全Radix)
```

参照型の場合

- 計算結果が参照型の場合には、以下のように参照するアドレスを 16 進数表示します。

```
(signed int &) ref = 0xD038                     (全Radix)
```

ビットフィールド型の場合

- 計算結果がビットフィールド型の場合には、以下のように表示します。

```
(unsigned int :13) s.f = 8191                   (Radixが初期状態の場合)
(unsigned int :13) s.f = 0x1FFF                         (Radixが16進数の場合)
(unsigned int :13) s.f = 1111111111111B                 (Radixが2進数の場合)
```

Cシンボルが見つからなかった場合

- 計算した式の中に発見できなかった C シンボルがあった場合には、以下のように表示します。

```
() x = <not active>                             (全Radix)
```

文法エラーの場合

- 計算した式が文法的に間違っていた場合には、以下のように表示します。

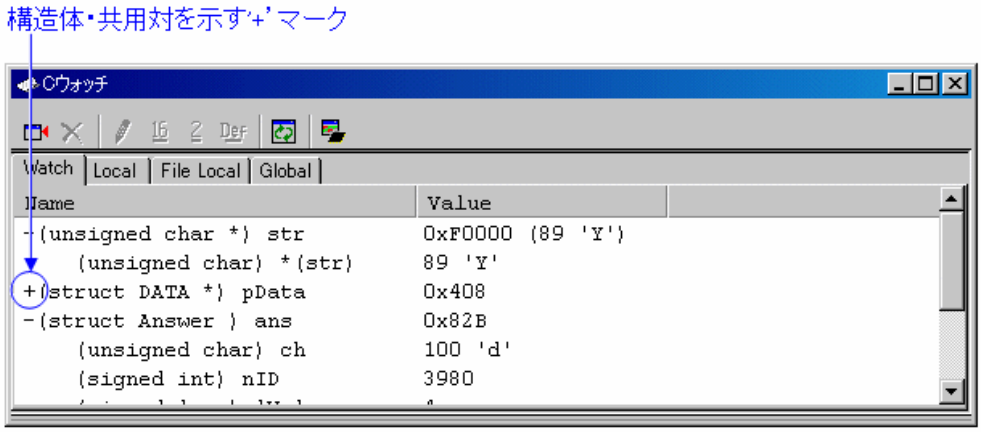
```
() str*(p = <syntax error>                    (全Radix)
(str*(p は間違った記述)
```

構造体・共用体型の場合

- 計算結果が構造体・共用体型の場合には、以下のようにアドレスを 16 進数表示します。

```
(Data) v = 0x1234                    (全Radix)
```

なお、C/C++言語式が構造体・共用体型のようにメンバを持つ場合は、以下に示すように、型名(タグ名)の左側に'+'マークが現れます。



この'+'マークが表示されている行をダブルクリックすると、その構造体(または共用体)のメンバが現れます。
メンバを表示すると、'+'マークは'-'マークにかわります。なお、'-'マークが表示されている行をダブルクリックすると、もとの状態に戻ります。このようにして、メンバを参照することができます。

注意

typedef で宣言された型定義名と同一名の変数を宣言した場合、その変数を参照することはできません。

レジスタ変数の場合

- 計算結果がレジスタ変数の場合には、以下のように型名の先頭に"register"と表示します。

```
(register signed int) j = 100
```

9. プログラム停止要因の表示

デバッグ機能によりプログラムが停止した場合、その停止要因はアウトプットウィンドウ、および、ステイタスウィンドウ（[Platform]シート）に表示されます。

停止要因の表示内容とその意味は、以下のとおりです。

表示	停止要因
Halt	[プログラムの停止]ボタン/メニューによる停止
S/W break	ソフトウェアブレーク
Address match interrupt break	アドレス一致ブレーク
H/W event, Combination	ハードウェアブレーク､論理組合せ And 条件または同時 And 条件成立
H/W event, Combination, Ax	ハードウェアブレーク、論理組合せ Or 条件成立 (Ax：成立したイベント番号)
H/W event, State transition, from xx	ハードウェアブレーク、状態遷移 State Transition 条件成立 (from xx：直前の状態 (start, state1, state2))
H/W event, State transition, Timeout	ハードウェアブレーク、状態遷移 タイムアウト成立
H/W event, Access protect error	プロテクトブレーク

注意事項

停止要因を表示可能かどうかは、接続しているターゲットに依存します。ターゲットによっては、常に "Halt" と表示されたり "---" と表示される場合があります。

10. 注意事項

10.1 製品共通の注意事項

10.1.1 Windows 上でのファイル操作

Windows 上でのファイル操作については、以下の点に注意してください。

1. ファイル名、及びディレクトリ名
   - 空白文字を含むファイル名、ディレクトリ名は使用できません。
   - 漢字のファイル名、ディレクトリ名は使用できません。
   - .(ピリオド)が 2 つ以上ついたファイルは使用できません。

2. ファイル指定、及びディレクトリ指定
   - "..."(2 つ上のディレクトリ指定)は使用できません。
   - ネットワークパス名は使用できません。 ネットワークパス名を使用する場合は、ドライブに割り当てて使用してください。

10.1.2 ソフトウェアブレークポイントの設定可能領域

ソフトウェアブレークポイントに設定可能な領域は、MCU によって異なります。

FoUSB/UARTデバッガの場合
内部 RAM 領域、および、内部 ROM 領域にソフトウェアブレークポイントが設定可能です。

10.1.3 C 変数の参照・設定

- typedef で宣言された型定義名と同一名の変数を宣言した場合、その変数を参照することはできません。
- レジスタ変数、ビットフィールド型変数への代入はできません。
- 64 ビット長の変数（long long 型や double 型など）への代入はできません。
- メモリの実体（アドレスとサイズ）を示さない変数への代入はできません。
- 複数のローカル変数が、コンパイラの最適化により同一領域に割り当てられている場合、その変数の値を正しく表示できない場合があります。
- リテラルな文字列を、char 配列あるいは char ポインタ型の変数以外に代入することはできません。
- 浮動小数点に対する四則演算はできません。
- 浮動小数点型変数に対する符号反転はできません。
- 浮動小数点型へのキャストはできません。
- レジスタ変数に対するキャストはできません。
- 構造体型、共用体型、及びそれらの型へのポインタ型へのキャストはできません。
- 文字定数およびリテラルな文字列には、エスケープシーケンスは記述できません。

10.1.4 C++での関数名

- ブレークポイント設定などで関数名を使用してアドレスを設定する場合、クラスのメンバ関数、operator 関数、および、オーバーロード（多重定義）関数を使用できません。
- C/C++言語式の記述に関数名を使用できません。
- 引数に関数名を指定するスクリプトコマンド（breakin, func 等）は使用できません。
- アドレス値設定領域において、関数名を使用したアドレス指定はできません。

10.1.5 複数モジュールのデバッグ

一つのセッションに複数のアブソリュートモジュールファイルを登録し、同時にダウンロードすることはできません。ただし、一つのアブソリュートモジュールファイルと複数の機械語ファイルを同時にダウンロードすることは可能です。

10.1.6 同期デバッグ

同期デバッグには対応していません。

10.1.7 RAM モニタ機能

RAM モニタ機能はメモリダンプで実現しています。本機能を使用した場合、リアルタイム性は損なわれます。また、アクセス属性による表示色の設定はできません。

10.1.8 ラインアセンブル機能

ラインアセンブルで指定可能な領域は、00000H～0FFFFH 番地に含まれる内部 RAM 領域です。

10.1.9 使用できないデバッグ機能

本デバッガには、ハードウェアブレーク、プロテクト、カバレッジ、トレース、時間計測、その他エミュレータに依存する機能はありません。したがって、これらに関連する機能は使用できません。

10.1.10 ソフトウェアブレーク機能

- ソフトウェアブレークは MCU のアドレス一致割り込みを利用して実現しています。したがって、設定可能なブレークポイント数は、ご使用の MCU に依存します。
- パスカウント欄をダブルクリックすると、通過回数を 1 から 255 まで指定できます。

10.2 M16C/R8C 用デバッガの注意事項

M16C/R8C 用デバッガに関する注意事項を以下に示します。

10.2.1 TASKING 社製 C コンパイラ ビットフィールドメンバの参照

TASKING 社製 C コンパイラ CCM16 をご使用の場合、ビットフィールドのメンバは常に unsigned short int 型で表示されます。これは、CCM16 が出力するデバッグ情報によるものです。

10.2.2 R8C/Tiny シリーズを使用する際の注意事項

- R8C/12-17 グループをデバッグする場合、フラッシュメモリの FFFFh 番地のビット 0 に"0"が書かれているマイコンはご使用になれません。"0"を書いた場合は、シリアルライタでフラッシュメモリを消去してください。
- R8C/10-17 グループでは、ウォッチドッグタイマのデバッグはできません。

10.3 コンパイラ/アセンブラ/リンカのオプション

デバッグするには、コンパイル・リンク時のオプション設定を考慮する必要があります。
設定内容については、次節以降を参照して下さい。
これ以外の設定では動作チェックを行っておりません。 これ以外の設定は、推奨いたしかねますのでご了承ください。

10.3.1 弊社 C コンパイラ NCxx をご使用の場合

コンパイル時に-O, -OR, -OS オプションを指定した場合、最適化のためにソース行情報が正しく生成されず、ステップ実行等が正しく行われない場合があります。
この問題を回避するには、-O, -OR, -OS オプションと同時に-ONBSD(もしくは-Ono_Break_source_debug) オプションも指定してください。

10.3.2 IAR 社製 C コンパイラをワークベンチ(EW)でご使用の場合

以下の手順でプロジェクトを設定してください。

1. IAR Embedded Workbench でのプロジェクト設定
   メニュー[Project]→[Options...]を選択すると Options For Target"xxx"ダイアログが開きます。
   このダイアログの Category で XLINK を選択し、以下のように設定してください。

   - Output タブ
     Format 領域で Other をチェックし、Output format に ieee-695 を選びます。

   - Include タブ
     XCL file name 領域で、ご使用の XCL ファイル(例：lnkm16c.xcl)を指定してください。

2. XCL ファイルの編集
   ご使用の XCL ファイルに -y オプションを追記してください。"-y"オプションの指定は、製品によって異なります。

製品名	-y オプション
M16C/R8C 用デバッガ	-ylmb

3. プログラムのビルド
上記設定後、ターゲットプログラムをビルドしてください。

これ以外の設定では動作チェックを行っておりません。これ以外の設定は、推奨いたしかねますのでご了承ください。

10.3.3 IAR 社製 C コンパイラをコマンドラインでご使用の場合

10.3.3.1 オプション指定

以下の手順でコンパイル・リンクしてください。

コンパイル時
"-r"オプションを指定して下さい。

リンク前
リンク時に読み込むリンカのオプション定義ファイル(拡張子.xcl)をオープンし、 "-FIEEE695"及び"-y"オプションを追加してください。
"-y"オプションの指定は、製品によって異なります。

製品名	-y オプション
M16C/R8C 用デバッガ	-ylmb

リンク時
"-f"オプションでリンカのオプション定義ファイル名を指定して下さい。

これ以外の設定では動作チェックを行っておりません。これ以外の設定は、推奨いたしかねますのでご了承ください。

10.3.3.2 コマンド入力例

以下にコマンド入力例を示します。

M32C用デバッガの場合

```
>ICCMC80 -r file1.c <Enter>
>ICCMC80 -r file2. c <Enter>
>XLINK -o filename.695 -f lnkm80.xcl file1 file2 <Enter>
```

M16C/R8C用デバッガの場合

```
>ICCM16C -r file1.c <Enter>
>ICCM16C -r file2.c <Enter>
>XLINK -o filename.695 -f lnkm16c.xcl file1 file2 <Enter>
```

XCL ファイル名は、製品やメモリモデルによって異なります。詳細は、ICCxxxx のマニュアルを参照して下さい。

10.3.4 TASKING 社製 C コンパイラをワークベンチ(EDE)でご使用の場合

以下の手順でプロジェクトを設定してください。

1. メニュー[EDE]→[C Compiler Option]→[Project Options...]を選択して下さい。 "M16C C Compiler Options [プロジェクト名]"ダイアログが開きます。
   このダイアログで以下のように設定してください。

   - Optimeze タブ
     Optimization level に"No optimization"を指定して下さい。

   - Debug タブ
     "Enable generation of any debug information(including type checkeing)" と "Genarate symbolic debug information"のみをチェックして下さい。

2. メニュー[EDE]→[Linker/Locator Options...]を選択して下さい。 "M16C Linker/Locator Options [プロジェクト名]"ダイアログが開きます。
   このダイアログで以下のように設定してください。

   - Format タブ
     Output Format に"IEEE 695 for debuggers(abs)"を指定して下さい。

3. 上記設定後、ターゲットプログラムをビルドしてください。

これ以外の設定では動作チェックを行っておりません。これ以外の設定は、推奨いたしかねますのでご了承ください。

10.3.5 TASKING 社製 C コンパイラをコマンドラインでご使用の場合

10.3.5.1 オプション指定

コンパイル時に"-g"、"-O0"オプションを指定して下さい。
これ以外の設定では動作チェックを行っておりません。
これ以外の設定は、推奨いたしかねますのでご了承ください。

10.3.5.2 コマンド入力例

以下にコマンド入力例を示します。

M16C/R8C用デバッガの場合

```
        >CM16 -g -O0 file1.c<Enter>
```

10.3.6 IAR 社製 EC++コンパイラをワークベンチ(EW)でご使用の場合

以下の手順でプロジェクトを設定してください。

1. IAR Embedded Workbench でのプロジェクト設定
   メニュー[Project]→[Options...]を選択すると Options For Target"xxx"ダイアログが開きます。
   このダイアログの Category で XLINK を選択し、以下のように設定してください。

   - Output タブ
     Format 領域で Other をチェックし、Output format に elf/dwarf を選びます。

   - Include タブ
     XCL file name 領域で、ご使用の XCL ファイル(例：lnkm32c.xcl)を指定してください。

2. XCL ファイルの編集
   ご使用の XCL ファイルに -y オプションを追記してください。"-y"オプションの指定は、製品によって異なります。

製品名	-y オプション
M16C/R8C 用デバッガ	-yspc

3. プログラムのビルド
   上記設定後、ターゲットプログラムをビルドしてください。

これ以外の設定では動作チェックを行っておりません。これ以外の設定は、推奨いたしかねますのでご了承ください。

M16C R8C FoUSB/UARTソフトウェア
ユーザーズマニュアル

発行年月日　2005年02月01日　　Rev.1.00

発行　　株式会社 ルネサス テクノロジ　営業企画統括部
〒100-0004　東京都千代田区大手町2-6-2

編集　　株式会社 ルネサス ソリューションズ　ツール開発部

M16C R8C FoUSB/UART ソフトウェア
ユーザーズマニュアル

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

RJJ10J0847-0100Z