
CodeScape マニュアル
for 2.1.2 Build 100



Dreamcast™

はじめに

CodeScape とは

CodeScape は、高速で直感的に分かりやすい Windows ベースのデバッガです。Windows 95、Windows NT 3.51/4.0 で動作します。

CodeScape のデバッグ機能を使用すると、オリジナルのソースまたは逆アセンブルしたコード中のバグを検出し、原因を突き止め、それを修正することができます。

CodeScape により、次の作業が可能になります。

1. プロジェクトファイルの編集。
2. プロジェクトファイルのコンパイルとリンク。
3. プロジェクトのデバッグおよびエラーのテスト。
4. さらに、次のいずれかを行います。
 - ・ デバッグでエラーが発生した場合は、上記ステップの 1 から 3 を繰り返す。
 - ・ デバッグが正常に終了した場合は、プロジェクトをビルドする。

CodeScape と各コンポーネント

CodeScape は、次のようなコンポーネントとともに提供されます。

- ・ Dev.Box 一式
- ・ Dreamcast SDK (CD-ROM)、CodeScape デバッガ本体、ヘルプ、サンプルプログラム、その他のドキュメントが収録されています

ご注意

このマニュアルに記載されている事項は、予告なしに変更されることがあります。このマニュアルは「現状のまま (as is)」で提供され、特定用途に対する市場性または適合性に関する暗示的な保証や条件を含み、これらに限定されることなく、明示であるか暗黙であるかに関わらず、いかなる保証も行いません。また Cross Products 社は、それが保証、契約、または他の法理論に基いている場合であっても、本マニュアルにおける記述の誤り、または本資料の性能や使用に関連して発生した逸失利益を含め、二次的損害あるいは間接損害に対して、一切責任を負わないものとします。

このマニュアルには、著作権で保護された所有者固有の情報が含まれています。本マニュアルのいかなる部分も、Cross Products 社の事前の許可なしに、いかなる形式でも複製、データベース システムや検索システムへの格納、その他いかなる手段によっても、転送や配布、電子的機械的なコピー、録音を行うことはできません。

商標

- ・ MS、Microsoft、MS-DOS、Windows、Windows NT は、米国 Microsoft Corporation の米国およびその他の国における登録商標です。
- ・ CodeScape と SNASM は、英国および英国外における Cross Products Limited の登録商標です。
- ・ Brief は INPRISE の登録商標です。
- ・ CodeWright は Premia Corporation の登録商標です。
- ・ Multi-Edit は American Cybernetics , Inc の登録商標です。
- ・ その他、記載されている会社名、製品名は、各社の商標および登録です。
- ・ 本文中には、TM、(R) マークは明記しておりません。

改版履歴

CodeScape ユーザーズガイド for 2.1.2 Build 100 BETA

1998 年 8 月 31 日 初版発行

1998 年 11 月 30 日 第二版発行

Copyright (C) 1998 by Cross Products Limited. All rights reserved.

Copyright (C) 1998 株式会社セガ・エンタープライゼス

編集・製作 株式会社アスキー AAP 書籍編集部

本マニュアルについて

対象読者

本マニュアルでは、Windows 95 または Windows NT 3.51/4.0 上でプログラムを作成するプログラムを対象としています。

本マニュアルの構成

「第1章 インターフェイスの設定と使用方法」では、CodeScape のデバッグ環境と使用方法を説明します。メニューバー、ツールバーおよびショートカットメニューのコマンドを説明し、プロジェクトで使用するウィンドウとリージョンの設定法と使用方法を説明します。

「第3章 ターゲットプロセッサ」では、ターゲットプロセッサへの接続法、初期化法とリセット法を説明します。これには、プログラムの再起動、プログラムのバイナリ部分の保存とローディングが含まれます。

「第4章 セッション」では、セッションを使用するためのコマンドの使用法を説明します。これには、新規および既存のセッションのオープンの仕方、セッションの保存方法、名前を付けて保存する方法、セッションのクローズの仕方、最近使用したファイルリストの表示法と使用方法、および CodeScape の終了の方法が含まれます。

「第5章 プロジェクトの使用法」では、プロジェクトビルド環境および MS-DOS や Windows のエディタの設定法を説明します。さらに、CodeScape でのプロジェクトのビルド法と、コンパイル後に CodeScape からエラーが通知された場合の対処法も説明します。

「第6章 デバッグ」では、プロジェクトファイルをデバッグするいくつかの方法を説明します。これには、ソースコードのステップ実行、ウォッチとデータブレイクポイントの設定、およびアセンブラコードのシミュレーションが含まれます。

「第8章 コマンドラインの使用法」では、コマンドラインによる CodeScape の使用方法を説明します。

マニュアルの表記法

このマニュアルでは、次の表記法を使用しています。

表記	説明
[スペース]キー	キーボード上のキー名に使用しています。
[Alt] + [F4]キー	2 個以上のキーを同時に押す必要がある場合、プラス記号「+」で示します。
[Alt] [F] [X]キー	2 個以上のキーを連続して押す必要がある場合、「、」で示します。
input and output	このフォントは、ユーザーによる入力とエラーメッセージを含めプログラムからの出力を示すために使用されます。
command-line	このフォントは、コマンドラインオプションに使用されます。

CodeScape 付加情報

(平成10年10月26日版)

概要

現時点で CodeScape にはいくつかの使用制限があります。これらは早急に解除される予定ですが、制限事項を十分理解の上、使用ください。

CodeScape Debugger は、Dev.Box 上で動作するプログラムをデバッグするためのツールです。

特徴

- ・ PC 上で動作するツールである。
- ・ Dev.Box 内のデバッグアダプタと SCSI にて接続される。
- ・ Windows ベースのインターフェイス。

推奨環境

PC 環境

- ・ OS : Windows95 OSR 2.0/2.1
- ・ SCSI カード : ASPI ドライバが動作するカード

ASPI マネージャはバージョン 4.00 以降が必要です。

Windows95 に標準で付属する ASPI マネージャは使用できません。詳しくは SCSI カードに付属のマニュアルをご参照ください。

なお、最新版 ASPI マネージャは、

<http://www.adaptec.com/support/overview/ezscsi4x.html>

より、ダウンロードできます。

制限事項

パイプラインシミュレータ・プロファイラ(バージョン)

パイプラインシミュレータとプロファイラが実装されていますが、バージョンであるため、各種の制限事項があります。詳細は CodeScape ヘルプファイルを参照ください。

C のソース関連制限

以下はデバッグ時の推奨環境です。コンパイラの Optimize 方法、CodeScape の動作不具合を理解した上で使用するのであれば、この限りではありません。

デバッグする C ソースは、

- ・ コンパイル時のオプションは、optimize = 0 以外指定しない。optimize = 1 とした場合、変数の表示、プログラムのステップ実行が正常に行えなくなります。
- ・ inline 展開関数は使用しない

- ・ inline assembler は使用しない

の制限で開発することを推奨します。

不具合項目

Watch Local Watch	<ul style="list-style-type: none"> ・ 変数の表示内容が間違っていることがあります。 ・ スコープの外と内側に同じ名前の変数がある場合、表示が正確に行われません。
Watch	<ul style="list-style-type: none"> ・ optimize = 1 のとき、変数の値が正常に表示されないことがあります。
Local Watch	<ul style="list-style-type: none"> ・ optimize = 1 のとき、変数の値が正常に表示されないことがあります。
Break	<ul style="list-style-type: none"> ・ F5 での設定が効かなくなることがあります。(この場合でもメニューからの設定は可能です。) ・ Optimize = 1 のとき、ブレークポイントのコンディション設定が効かなくなります。 ・ Watch Conditions を指定すると 変数の値が変化しなくなることがあります。
Register window	<ul style="list-style-type: none"> ・ FR0-15 などが指定できるにも関わらず、Change Inc/DecValue に小数点を入力できません。
Step	<ul style="list-style-type: none"> ・ Optimize = 1 のとき、ソースコードの行の順番は保証されません。
Project	<ul style="list-style-type: none"> ・ ビルドの最中、ウインドウの表示更新がされません。
Source	<ul style="list-style-type: none"> ・ 行番号が二モニックのある行にのみ表示されます。
Call Stack	<ul style="list-style-type: none"> ・ 1 レベルまたは 2 レベルまでしかコールスタックが表示されないことがあります。
Editor	<ul style="list-style-type: none"> ・ 日本語の入力の一部問題があります。

注意事項

OS モードと CPU モードについて

DebugAdapter の動作モードには、OS モードと CPU モードがあります。それぞれの特徴は以下のとおりです。これらのモードの切り替えは、DACheck の終了時に表示されるダイアログで行ないます。

OS モード： DebugAdapter 起動後、BootROM に処理を移行します。GD-Workshop と組み合わせた自己起動プログラムのテストなどに使用します。

CPU モード： 標準的な動作モードです。DebugAdapter の起動後、Dev.Box は待機状態になります。

関連ファイルの更新

CodeScape 上で Local 変数の参照が出来ない場合、CodeScape フォルダ内の一部ファイルが古い可能性があります。CodeScape ディレクトリ内のファイル全てを、当 SDK に付属のものに更新してください。

動作に不具合が生じた場合

今まで動作していたものが急に動作しなくなった場合は、以下の項目を確認してください。

主な確認事項

- ・ 特定のプログラムのみ不具合が生じますか？(プログラムの不具合)
- ・ PC 上の他のアプリケーションは正常に動作していますか？(PC の不具合)
- ・ 他の環境では、同じプログラムが正常動作していますか？(PC または Dev.Box の不具合)

内容	対処例
プログラムの不具合	プログラムを修正する
PC の不具合	電源を入れ直す
PC の不具合	OS を再インストールする。ASPI マネージャをアップデートする。
PC の不具合	ハードウェアの変更
CodeScape の不具合	CodeScape を再起動する。
CodeScape に設定した項目が元に戻せなくなった	Codescape.ini ファイルを消去する。
CodeScape に設定した項目が元に戻せなくなった	Session を新たに生成し、再度設定し直す
Dev.Box に初期化が必要	CodeScape より、File Reset Target Hard reset で再度プログラムを読み込む
Dev.Box に初期化が必要	Dev.Box の電源を入れ直す。
Dev.Box の不具合	Dev.Box の変更。

codescape.ini について

PC でウィンドウズのフォルダにある codescape.ini ファイルを消去 (またはリネーム) してください (例 : c : ¥windows¥codescape.ini を消去) 。

これらはすべてを一度に行う必要はありません。症状に応じて適宜組み合わせて動作確認してください。再現性がある場合はお手数ですが、テクニカルサポートセンターまでご連絡ください。

変更履歴

10月19日版からの変更点

- ・ CodeScape Build100 は、Build99 の BugFix 版のため、制限事項変更無し。

9月21日版からの変更点

- ・ プロファイラ制限事項改定
- ・ その他修正

7月30日版からの変更点

- ・ 関連ファイルについて
- ・ Profiler 制限事項追加
- ・ その他詳細修正

7月15日版からの変更点

- ・ DALI.CFG について
- ・ プロジェクト設定について

6月30日版からの変更点

- ・ 各種制限の変更
- ・ プロファイラ・シミュレータの追加
- ・ OS モードと CPU モードについて

6月11日版からの変更点

- ・ ASPI マネージャに関する記述を追加。

5月07日版からの変更点

- ・ CodeScape のバージョンアップに伴い、制限事項項目が変わりました。

4月24日版からの変更点

- ・ CodeScape のバージョンアップに伴い、制限事項項目が変わりました。

contents

目次

はじめに	2
本マニュアルについて	4
CodeScape 付加情報 (平成 10 年 10 月 26 日版)	5

第1章

インターフェイスの設定と使用法 13

1.1	メニューバーのコマンド	13
1.2	ツールバーのコマンド	16
1.2.1	ツールバーの表示、非表示、ドッキング、移動	16
1.3	各ツールバーのコマンド	18

第2章

ウィンドウとリージョンの動作 27

2.1	ウィンドウの使用	28
2.2	リージョンの使用	29
2.2.1	リージョンの設定	32
2.3	【Target】ウィンドウ	36
2.3.1	Dev.Boxのプロセッサの表示	37
2.4	【Input/Output】ウィンドウ	37
2.5	【Source】リージョン	39
2.6	【Disassembly】リージョン	43
2.7	【Call Stack】リージョン	47

2.8	【 Watch リージョンと Local Watch リージョン ..	48
2.8.1	【 Watch リージョン	49
2.8.2	【 Local Watch リージョン	52
2.9	【 Memory リージョン	54
2.10	【 Register リージョン	62
2.11	【 Edit リージョン	67
2.11.1	ファイルのオープンと保存	68
2.11.2	検索と置換	69
2.11.3	テキストの切り取りと貼り付け	70
2.11.4	ブックマークの使用	70

第3章

プログラムの実行

73

3.1	Dev.Box への接続	73
3.2	プロセッサの更新レートの設定	74
3.3	プログラムのロード	75
3.4	プログラムのリスタート	77
3.5	バイナリの保存とロード	78

第4章

セッション

79

第5章

プロジェクトの使用法

83

5.1	プロジェクトビルド環境の準備	83
5.1.1	エディタの設定	85

第6章

デバッグ

89

6.1	プログラムの実行と停止	89
6.2	ブレークポイント	95
6.3	ブレークポイントの設定	99
6.3.1	ブレークポイント式のフォーマット	109
6.4	Dev.Box のシミュレーション	113
6.4.1	シミュレータによって生成される情報	113
6.4.2	パイプラインのストール	116
6.4.3	シミュレーションの結果を読む	116
6.5	プログラムファイルのプロファイル	117
6.5.1	プロファイラのショートカットメニューを使用する	118
6.5.2	[Profiler] ツールバーのオプション	119
6.5.3	プロファイルの表示形式	121

第7章

式

123

7.1	C/C++ 式	123
7.2	アセンブラ式	125
7.2.1	[Expression evaluator] ダイアログボックス	126
7.2.2	[Symbol Completion] ダイアログボックス	126

第8章

コマンドラインの使用法

127

8.1	コマンドラインからの CodeScape の起動	127
-----	--------------------------------	-----

一般的操作

131

A.1	メニューバー上のショートカットとアクセスキー	131
A.2	ショートカットメニュー上のショートカットと アクセスキー	136

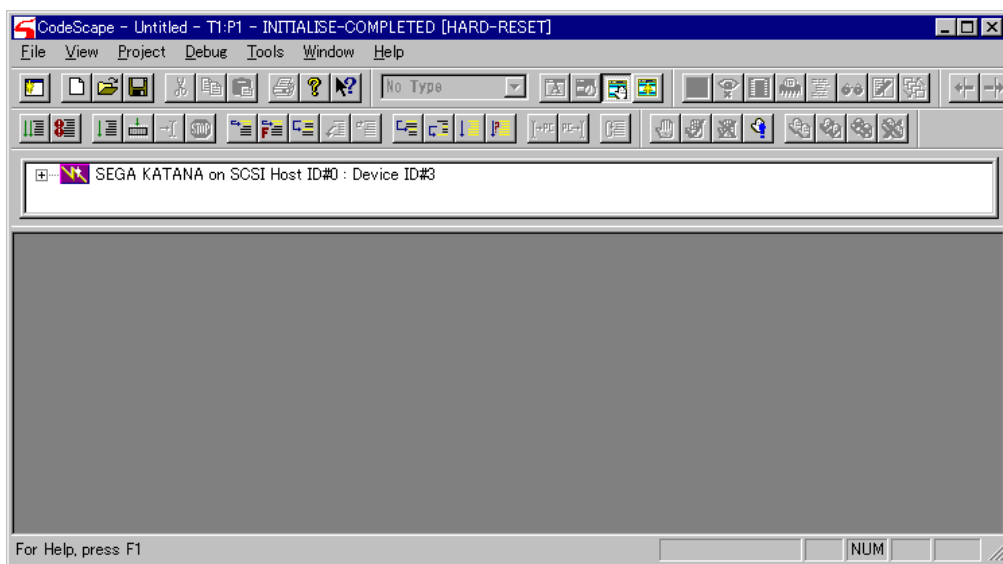
第 1 章

インターフェイスの設定と使用法

CodeScape はマウスやキーボードを使用して操作することができます。CodeScape には、ドッキング、フローティング、または非表示にできる多くの有用なツールバーがあります。リージョン固有のショートカットメニューが、頻繁に使用する機能に対し用意されています。ユーザーインターフェイスは、次のものから構成されています。

- ・メニューバー
- ・ツールバー
- ・ウィンドウ
- ・リージョン

1.1 メニューバーのコマンド



[File]メニュー [Alt] + [F]キー

[File]メニューのコマンドにより、セッションの使用、Dev.Box のリセット、プログラムファイルのロード、プログラムファイルの実行再開、バイナリ情報の保存とロードを行うことができます。バイナリの保存 (Save Binary) とバイナリのロード (Load Binary) を使用すると、メモリーイメージのロード / セーブができます。

[File]メニューにより、最近使用したセッションファイルの一覧を表示することもできます。この一覧を非表示にしたり、表示されるファイル数を変更することはできません。

[Edit]メニュー [Alt] + [E]キー

[Edit]メニューのコマンドにより、[Edit]リージョンの切り取りと[Edit]リージョンへの貼り付け、および検索をすることができます。ウィンドウを開いたり、リージョンを作成したり、またはセッションをロードすると、[Edit]メニューが使用可能になります。

[View]メニュー [Alt] + [V]キー

[View]メニューのコマンドにより、ツールバーの表示、非表示、およびリージョンの設定をすることができます。

[Project]メニュー [Alt] + [P]キー

[Project]メニューのコマンドにより、CodeScape から現在のプロジェクトの設定およびプロジェクトのビルドを行うことができます。

[Debug]メニュー [Alt] + [D]キー

[Debug]メニューのコマンドにより、プログラム実行のコントロール、コードのステップ実行、およびブレークポイントを利用することができます。

カーソル位置を PC (プログラムカウンタ) に設定することができます。その逆も可能です。デフォルトは、PC の値に設定されています。表示を、PC を含む式 (Expression)、レジスタ、またはメモリにロックすることもできます。

自分のコンピュータ上で、Dev.Box のプロセッサの動きをシミュレートすることもできます。

[Tools]メニュー [Alt] + [T]キー

[Tools]メニューのコマンドにより、ショートカットキーを作成してカスタマイズし、GD Workshop で CodeScape の中から走らせたいプログラムを指定することができます。CodeScape にプログラムを追加したときに、[Tools]メニューに、その名前が表示されます。

Dev.Box のプロセッサのオペレーションを、開発用コンピュータでシミュレートすることができます。CodeScape の全てのデバッグ機能は、シミュレータが動作しているときも利用可能です。プログラムファイルのランタイム動作を調べるために同じくプロファイラを走らせることができます。

[Region]メニュー [Alt] + [R]キー

[Region]メニューのコマンドにより、リージョンの分割 (新規作成) とリージョン種類の変更をすることができます。ウィンドウ、リージョン、またはセッションを開いたり、作成したりすると、リージョンメニューが使用可能になります。

[Window]メニュー [Alt] + [W]キー

[Window]メニューを使用すると、新規ウィンドウを開くことができます。新規ウィンドウを開くと、[Edit]メニューと[Region]メニューがメニューバーに表示されます。追加のコマンドが[Window]メニューに表示され、これにより、CodeScape 内の複数のウィンドウを調整することができます。

[Window]メニューには、リージョンタイプの一覧も表示され、現在アクティブなリージョンが強調表示されます。1つのウィンドウ フレームに2つ以上のリージョンがある場合は、そのフレーム内のアクティブなリージョンがその一覧に表示されます。この一覧を非表示にしたり、表示されるリージョン数を変更することはできません。

[Help]メニュー [F1]キー

[Help]メニューを使用すると、オンラインヘルプを取得したり、CodeScape のバージョン情報を表示することができます。

リージョン固有のショートカットメニュー

各リージョンには、次の2つのショートカットメニューがあります。

1. [Region Type]メニューには、リージョンのタイプを変更するためのコマンドがあります。メニューを表示するには、次のいずれかを行います。
 - ・ [Ctrl] + [Shift] + [F10]キーを押す。
 - ・ [Ctrl] + 右クリックで、リージョン内の適当な個所をクリックする。
2. [Region Actions]メニューには、リージョン固有のコマンドと、プログラムの実行をコントロールしたり、ブレークポイントを操作するためのグローバルなコマンドがあります。メニューを表示するには、次のいずれかを行います。
 - ・ [Shift] + [F10]キーを押す。
 - ・ リージョン内の適当な個所を右クリックする。

1.2 ツールバーのコマンド

ツールバーを使用すると、主要なデバッグ機能を使用することができます。ツールバーを表示、非表示にするために [Toolbar Configuration] チェックボックスを使用するには、次のいずれかを行います。

- ・ステータスバーを右クリックする。
- ・ [View] メニューの [Toolbar] をクリックする。

その次に、一覧からツールバーを選択するか、選択を解除します。

ツールバーの役割

名称	役割
[Breakpoint] ツールバー	一般的なブレークポイント操作に使用します。
[Debug] ツールバー	デバッグ操作に使用します。
[Processor Combo] ツールバー	Dev.Box のプロセッサの選択、Dev.Box のプロセッサの設定、プログラムファイルのロードに使用します。
[Project Build] ウィンドウ	進行中のプロジェクトビルドのステータス情報が表示されます。
[Region Combo] ツールバー	リージョンタイプの設定を行います。
[Region] ツールバー	リージョンタイプの設定と変更を行います。
[Splitter] ツールバー	マウスを使用して、リージョンを分割します。
[Standard] ツールバー	新規ウィンドウのオープン、およびセッションの作成、オープン、保存に使用します。
[Target] ウィンドウ	現在の Dev.Box のプロセッサを表示します。ツールバーにより、プログラムファイルのロードやプロセッサの設定に使用することもできます。
[Target Combo] ツールバー	現在の Dev.Box の表示と設定に使用します。
[Edit] ツールバー	編集操作に使用します。

1.2.1 ツールバーの表示、非表示、ドッキング、移動

ツールバーの表示

次のいずれかを行います。

- ・ステータスバーを右クリックする。
- ・ツールバーの空白部分を右クリックして、ツールバーを選択する。
- ・ [View] メニューをクリックし、[Toolbar...] を選択し、表示したいツールバーをチェックボックスで選択して、[OK] ボタンをクリックする。

ツールバーの非表示

1. ツールバーの空白部分を右クリックします。
2. 一覧からツールバーをクリアします。

ツールバーがドッキングされていない場合は、次のようにします。

- ・ ツールバーのタイトルバーを右クリックして、非表示をクリックする。
- ・ ツールバーのタイトルバーをクリックする。

ツールバーがドッキングされている場合は、次のようにします。

1. [View] メニューをクリックし、[Toolbar...] を選択します。
2. 非表示にするチェックボックスをクリアし、[OK] をクリックします。

ツールバーのドッキング

次のいずれかを行います。

- ・ ツールバーをメインウィンドウの端までドラッグする。
- ・ タイトルバーをダブルクリックします。ツールバーは、その最後にドッキングされた位置にドッキングされます。

ツールバーの移動

1. 次のいずれかを行います。
 - ・ ツールバーのタイトルバーを、右クリックして、移動をクリックする。
 - ・ ツールバーのタイトルバーをドラッグする。
 - ・ ツールバーの適当な個所をクリックする。
2. ツールバーを必要な位置までドラッグします。

注意

[Ctrl] キーを押し続けると、[Target] ウィンドウまたは [Build] ウィンドウをドッキングすることはできません。

1.3 各ツールバーのコマンド

ツールバーのコマンドにより、主要なデバッグ機能を使用することができます。ほとんどのデバッグ機能は、キーボードによるショートカットキーを利用することもできます。

[Breakpoint] ツールバー



[Breakpoint] ツールバーにより、一般的なブレイクポイント機能をポイントし、クリックして使用することができます。

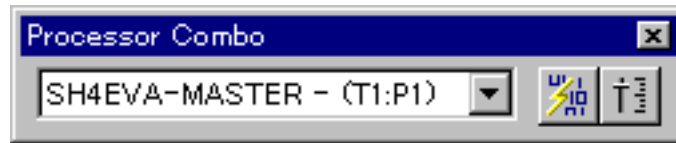
コマンド	アイコン	キー操作
ブレイクポイントのオン/オフ		[F5] キー
ブレイクポイントの有効化		なし
ブレイクポイントの無効化		なし
ブレイクポイントの設定		[Ctrl] + [F5] キー
すべてのブレイクポイントのリセット		[Alt] + [F5] キー
すべてのブレイクポイント有効化		[Ctrl] + [Shift] + [F5] キー
すべてのブレイクポイント無効化		[Ctrl] + [Alt] + [F5] キー
すべてのブレイクポイント削除		[Shift] + [F5] キー

[Debug] ツールバー



コマンド	アイコン	キー操作
すべてのプロセッサの実行		[Ctrl] + [F9] キー
すべてのプロセッサの停止		なし
実行		[F9] キー
カーソルまで実行		[Shift] + [F9] キー
アドレスまで実行		[Alt] + [F9] キー
停止		[F9] キー
シングルステップ (イントゥー)		[F7] キー
強制ステップ (イントゥー)		なし
ステップオーバー		[F8] キー
ステップアウト		[Ctrl] + [F8] キー
アンステップ		[Ctrl] + [F7] キー
ステップランイン		[Shift] + [F7] キー
ステップランアウト		[Shift] + [F8] キー
カーソル位置を PC にセット		[Ctrl] + [Shift] + [P] キー
PC をカーソル位置にセット		[Ctrl] + [Alt] + [P] キー
リスタート		[Ctrl] + [Shift] + [R] キー

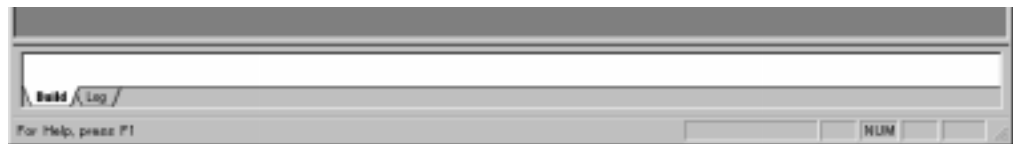
[Processor Combo] ツールバー



[Processor Combo] ツールバーには、現在の Dev.Box の現在のプロセッサが表示されます。このツールバーにより、プログラムファイルのロードと現在のプロセッサの設定を行うことができます。

コマンド	アイコン	キー操作
プログラムファイルのロード		[Ctrl] + [Shift] + [C] キー
プロセッサの設定		なし

[Input/Output] ウィンドウ



[Input/Output] ウィンドウは自動的に表示されます。

- ・ [Build] タブはプロジェクトのビルド時に、ユーティリティーの出力を表示します
すべてのエラーと警告は [Build] タブに表示されます。生成された情報をスクロールすることも可能ですし、[F4] キーを押してリスト表示されたエラー行に移動することもできます。
- ・ CodeScape の [Edit] リージョンは、プロジェクトファイル中のエラーや警告を含む最初の行を自動的に開きます。[Project Built] ウィンドウは他のエラーを表示するのにも使用できます。[Edit] リージョンが表示されていない場合には、自動的に表示されます。
- ・ [Project Build] ウィンドウをダブルクリックすると、エディタが起動してソースファイルを読み込み、エラーや警告を含む行に移動します。エディタの種類によっては、エラーや警告を含む行に移動しないものもあります。
- ・ [Log] タブは現在の Dev.Box の全てのメッセージを表示します。

注意

132 文字より大きいテキスト文字列は、[Log] リージョンに表示されるとき、切りつめられます。

注意

[Input/Output] ウィンドウは、メインウィンドウの上または下にドッキングしたり、フローティングにすることができます。

[Region Combo] ツールバー



[Region Combo] ツールバーを使用すると、リージョンのタイプの変更とリージョンの表示更新レートを設定することができます。

設定するコマンド	アイコン
リージョンの設定	
ウィンドウの更新レート	
すべてのウィンドウの更新停止	
すべてのリージョンの更新	

[Region] ツールバー



[Region] ツールバーにより、リージョンタイプの設定や変更ができます。

作成するリージョンのタイプ	アイコン	キー操作
Disassembly		[Alt] + [1] キー
Log		[Alt] + [2] キー
Local Watch		[Alt] + [3] キー
Memory		[Alt] + [4] キー
Register		[Alt] + [5] キー
Source		[Alt] + [6] キー
Watch		[Alt] + [7] キー
Edit		[Alt] + [8] キー
Call Stack		[Alt] + [9] キー

注意

すべてのリージョンで表示の更新をやめるには、[Ctrl] + [Shift] + [U] キーを押します。

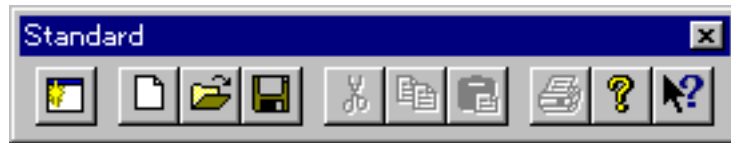
[Splitter] ツールバー



[Splitter] ツールバーにより、既存のリージョンを分割して新規リージョンを作成することができます。

コマンド	アイコン	キー操作
左に分割		[Ctrl] + [Shift] + [] キー
右に分割		[Ctrl] + [Shift] + [] キー
上に分割		[Ctrl] + [Shift] + [] キー
下に分割		[Ctrl] + [Shift] + [] キー
リージョンの削除		[Ctrl] + [D] キー

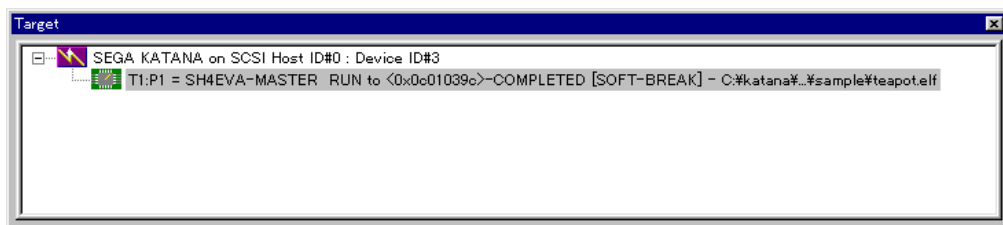
[Standard] ツールバー



[Standard] ツールバーを使用すると、新規ウィンドウの作成、およびセッションの作成、オープン、保存の各操作を簡単に行うことができます。

コマンド	アイコン	キー操作
新規ウィンドウ		[Ctrl] + [N] キー
新規セッション		[Ctrl] + [Shift] + [N] キー
セッションを開く		[Ctrl] + [O] キー
セッションを保存する		[Ctrl] + [S] キー
カット		[Ctrl] + [X] キー
コピー		[Ctrl] + [C] キー
ペースト		[Ctrl] + [V] キー
印刷		[Ctrl] + [P] キー
About Box		なし
ヘルプ		[F1] キー

[Target] ウィンドウ



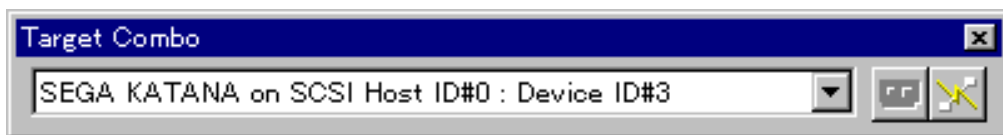
[Target] ウィンドウには、CodeScape が接続されているすべての Dev.Box と各 Dev.Box で使用可能なプロセッサが表示されます。各 Dev.Box に対するプロセッサのステータスは、ターゲットプロセッサに表示されます。

新規ウィンドウを作成するとき、CodeScape は Dev.Box 上の選択されたプロセッサからのターゲット情報を使用します。[Target] ウィンドウにより、Dev.Box のプロセッサの選択を行うことができます。

注意

[Target] ウィンドウは、メインウィンドウの上または下にドッキングするか、フローティング状態にすることができます。

[Target Combo] ツールバー



[Target Combo] ツールバーは現在の Dev.Box を表示し、現在の Dev.Box を設定することができます。

コマンド	アイコン
Dev.Box の設定	
シリアルの設定	

注意

[シリアルの設定] ボタンは、Dev.Box がシリアルポートを通じて接続されているときだけ表示されます。

[Edit] ツールバー

[Edit] ツールバーは編集操作に使用します。

コマンド	アイコン	キー操作
新規編集ファイルの作成		なし
編集ファイルのオープン		なし
現在の編集ファイルの保存		なし
最後の操作のアンドゥー		[Ctrl] + [Z] キー
最後の操作のリドゥー		なし
文字列の検索		[Ctrl] + [F] キー
現在選択しているものを置換		なし
ブックマークのオン / オフ (トグル)		なし
ファイル中の次のブックマークに移動		なし
ファイル中の前のブックマークに移動		なし
すべてのブックマークの削除		なし

ウィンドウとはフレームで、それを 1 つのリージョンとして設定したり、分割して複数のリージョンを作成したりすることができます。

リージョンは、プロジェクトの情報を表示するものです。

次の方法で、プロジェクトの複数のリージョンを同時に表示することができます。

- ・複数のウィンドウのオープンとクローズ
- ・最小化と最大化
- ・重ねたり整列
- ・ウィンドウを複数のリージョンに分割し、メモリ内容やソースコードなどの異なる情報を表示
- ・ウィンドウ サイズを変更
- ・分割バーを移動して、リージョン サイズを変更
- ・ウィンドウのリージョンの縦横比を維持してサイズを変更

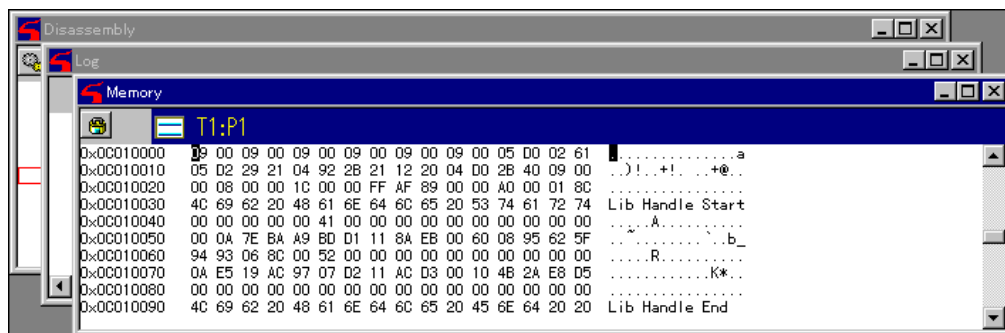
リージョンの設定

[Region Configuration] ダイアログボックスにより各リージョンの種類を、また個々のリージョンおよび各プロセッサにしたがって、フォントやカラーを設定することができます。これにより、各プロセッサを区別し、関連するリージョンを表示してメモリの変化を知ることができます。


注意

[Source] リージョンと [Disassembly] リージョン内のカーソルを同期して、ソースコードとそのコンパイルしたアセンブルコードを比較することができます。


2.1 ウィンドウの使用




新規ウィンドウを開く

- ・ [Window] メニューをクリックしてから、[New window] をクリックする。
- ・ 標準ツールバーの  をクリックする。
- ・ [Ctrl] + [N] キーを押す。


ウィンドウの最小化

- ・ ウィンドウタイトルバーの  をクリックする。
- ・ システムメニューの、[最小化] をクリックする。

ウィンドウの最大化

- ・ ウィンドウタイトルバーの  をクリックする。
- ・ システムメニューの、[最大化] をクリックする。

ウィンドウを閉じる

- ・ ウィンドウタイトルバーの  をクリックする。
- ・ システムメニューの、[閉じる] をクリックする。
- ・ [Ctrl] + [F4] キーを押す。

注意 ウィンドウ内にひとつしかないリージョンを削除すると、ウィンドウも同様に削除されます。

すべてのウィンドウを閉じる

- ・ [Window] メニューをクリックし、[Close all Windows] をクリックする。

ウィンドウの移動

次の手順で行います。

1. ウィンドウのタイトルバーをクリックする。
2. ウィンドウを必要な位置までドラッグする。

ウィンドウ間の移動

- ・ [Ctrl] + [Tab] キーを押す。

ウィンドウサイズの変更

次の手順で行います。

1. ウィンドウの端をポイントする。
2. ウィンドウの外枠をクリックして、必要なサイズになるまでドラッグする。

ウィンドウ中のリージョンを縦横比を変えずにサイズを変更するには、次のようにします。

1. [Window] メニューをクリックし、[Proportional resizing] をチェックする。
2. ウィンドウの境界をポイントする。
3. ウィンドウをクリックして、必要なサイズになるまでドラッグする。

CodeScape の再起動時に、現在のセッションをロード

- ・ [Window] メニューをクリックし、[Load last session on startup] をクリックして選択する。

すべてのウィンドウを重ねて並べる

- ・ [Window] メニューをクリックし、[Cascade] をクリックする。

すべてのウィンドウの整列

- ・ [Window] メニューをクリックし、[Tile] をクリックする。

アイコンの整列

セッションウィンドウの下部に、最小化したすべてのリージョンウィンドウを整列します。

- ・ [Window] メニューをクリックし、[Arrange Icons] をクリックする。

2.2 リージョンの使用



リージョンの移動

リージョンをスクロールするには、次のいずれかを行います。

- ・ [] キーと [] キーを使用して、カーソルを一度に 1 文字動かす。
- ・ [] キーと [] キーを使用して、カーソルを一度に 1 行上下する。
- ・ [PageUp] キーと [PageDown] キーを使用して、一度に 1 ページ上下する。
- ・ [Home] キーと [End] キーを使用して、ファイルの最初の行や最終行に移動する。

リージョンのフィールド間を移動するには、次の操作を行います。

- ・ カーソルを次のフィールドに移動するには、[Tab] キーを押す。
- ・ カーソルを前のフィールドに移動するには、[Shift] + [Tab] キーを押す。

注意

フィールドの最後では、カーソルは次のフィールドに移動します。最後のフィールドの最後では、カーソルは次の行に移動します。

リージョン間の移動

左のリージョンに移動するには、次のいずれかを行います。

- ・ 左のリージョンの適当な個所をクリック。
- ・ [Ctrl] + [] キーを押す。

右のリージョンに移動するには、次のいずれかを行います。

- ・ 右のリージョンの適当な個所をクリック。
- ・ [Ctrl] + [] キーを押す。

上のリージョンに移動するには、次のいずれかを行います。

- ・ 上のリージョンの適当な個所をクリック。
- ・ [Ctrl] + [] キーを押す。

下のリージョンに移動するには、次のいずれかを行います。

- ・下のリージョンの適当な個所をクリック。
- ・ [Ctrl] + [] キーを押す。


新規リージョンの作成

新規リージョンを作成するには、次のいずれかを行います。


- ・新規ウィンドウを開く ([Ctrl] + [N] キー)
- ・既存のリージョンを分割する。

リージョンの分割


リージョンを左に分割するには、次のいずれかを行います。

- ・ [Region] メニューをクリックして、 [Split] をポイントし、 [Left] をクリック。
- ・ [Splitter] ツールバーの  をクリック。
- ・ [Ctrl] + [Shift] + [] キーを押す。


リージョンを右に分割するには、次のいずれかを行います。

- ・ [Region] メニューをクリックして、 [Split] をポイントし、 [Right] をクリック。
- ・ [Splitter] ツールバーの  をクリック。
- ・ [Ctrl] + [Shift] + [] キーを押す。

リージョンを上分割するには、次のいずれかを行います。


- ・ [Region] メニューをクリックして、 [Split] をポイントし、 [Up] をクリック。
- ・ [Splitter] ツールバーの  をクリック。
- ・ [Ctrl] + [Shift] + [] キーを押す。

リージョンを下分割するには、次のいずれかを行います。

- ・ [Region] メニューをクリックして、 [Split] をポイントし、 [Down] をクリック。
- ・ [Splitter] ツールバーの  をクリック。
- ・ [Ctrl] + [Shift] + [] キーを押す。

リージョンの削除

リージョンをアクティブにしてから、次のいずれかを行います。

- ・ [Region] メニューをクリックして、 [Delete] をクリック。
- ・ [Splitter] ツールバーの  をクリック。
- ・リージョンで、 [Ctrl] + 右クリックして、 [Delete Region] をクリック。
- ・ [Ctrl] + [D] キーを押す。

注意 ウィンドウ内に、ひとつしかないリージョンを削除すると、ウィンドウも同様に削除されます。

リージョンのタイプを変更

リージョンのタイプを変更するには、次のいずれかを行います。

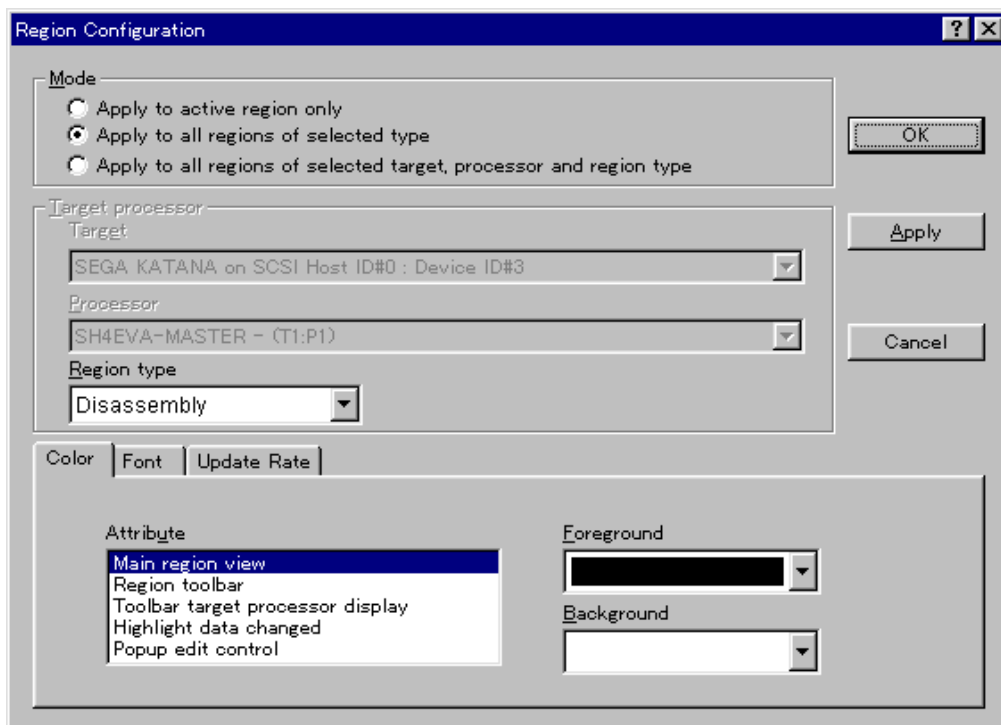
- ・ [Region] メニューをクリックし、[Type] をポイントして、リージョンのタイプをクリック。
- ・ [Region] コンボボックスで、ドロップダウンリストからリージョンのタイプを選択。
- ・ [Region] ツールバーで、[Region Type] アイコンをクリック。
- ・ [Ctrl] キーを押しながら右クリックして、リージョンのタイプをポイントしクリック。

リージョンの更新

- ・ リージョンを右クリックし、[Update all regions now] をクリックする。
- ・ [Ctrl] + [U] キーを押す


2.2.1 リージョンの設定

[Region Configuration] ダイアログボックス



[Region Configuration] ダイアログボックスには、フォントとカラーの設定、リージョンごと、リージョンタイプごと、および各プロセッサについての更新レートを設定するためのタブコマンドがあります。

アクティブなリージョンの設定をするには、次のいずれかを行います。

- ・ 右クリックして、[Properties...] をクリック。
- ・ アクティブリージョンのタイトルバーで  をダブルクリック。

[Region Configuration] ダイアログボックスが表示されます。

次の操作を行います。

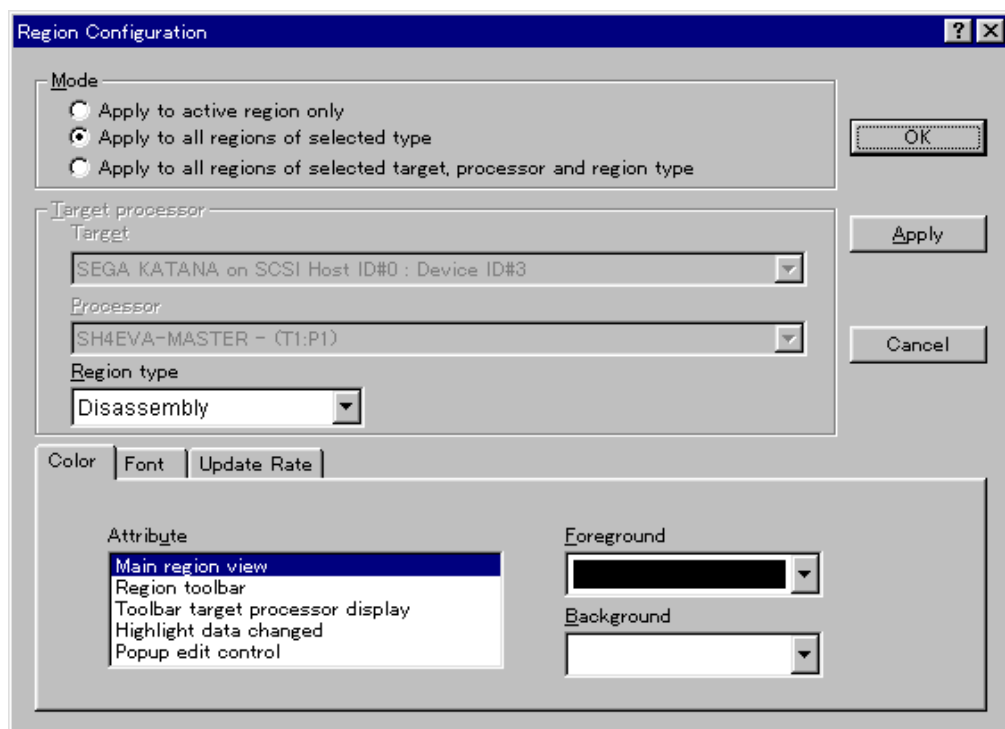
1. [Mode] を設定して、[Target processor] テキストボックスにコマンドを指定します。
2. [Target] [Processor] または [Region] タイプリストにオプションを指定します。
3. それから、次のいずれかの操作を行います。
 - ・ [Apply] ボタンをクリックして、ダイアログボックスを終了せずに設定の変更内容を表示する。
 - ・ [OK] ボタンをクリックして、プロジェクトに対する変更内容を設定する。

Mode と Dev.Box のプロセッサの設定

設定対象	[Mode] を以下に設定
プロジェクトの現在アクティブなリージョン	アクティブリージョンだけに適用
すべてのプロセッサの選択されたタイプのすべてのリージョン	選択したタイプのすべてのリージョンに適用 リージョンのタイプを選択
特定の Dev.Box のプロセッサとリージョンタイプ	選択した Dev.Box のすべてのリージョンに適用 Dev.Box のプロセッサとリージョンタイプを選択

カラーとフォントの設定

[Color] および [Font] のタブコマンドを使用すると、プロセッサを区別したり、関連したリージョンやメモリの変化を区別して表示することができます。



指定された [Mode] のカラー属性を設定するには、次の操作を行います。

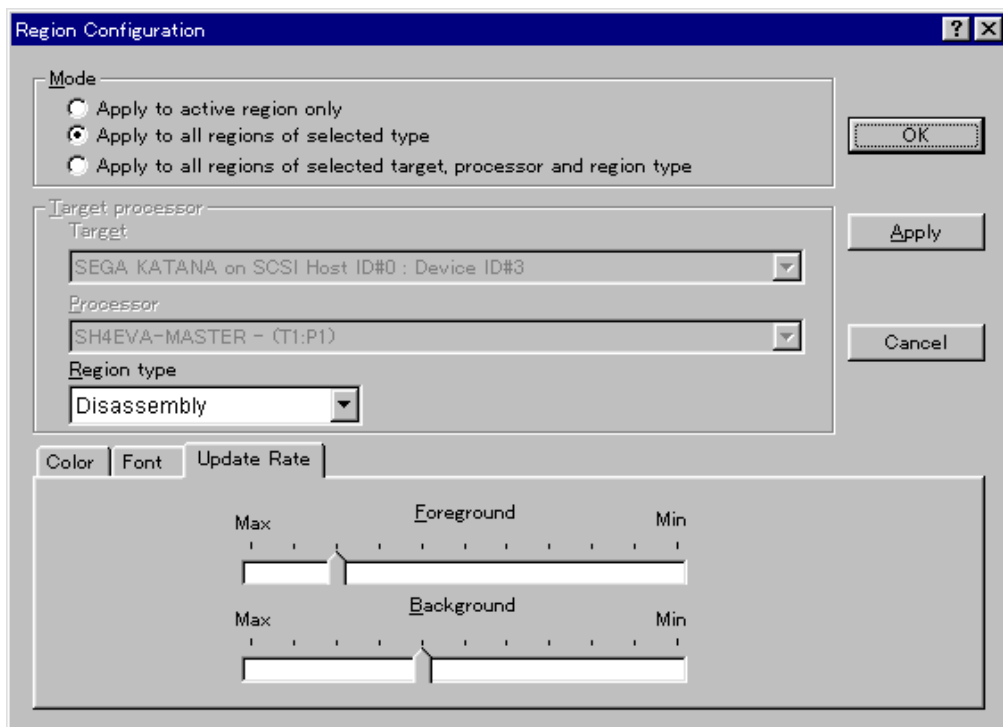
1. [Color] タブを選択します。
2. 色を変更したい属性を選択します。
3. リージョンの前景色を設定します。
4. リージョンの背景色を設定します。
5. [OK] ボタンをクリックします。

指定された [Mode] のフォントタイプとサイズを設定するには、次のことを行います。

1. [Font] タブを選択します。
2. [Change font] をクリックします。
3. リージョンのフォント、フォントスタイル、サイズを設定します。
4. 必要な文字飾り (Effects) を設定します。
5. [OK] ボタンをクリックします。

リージョンの更新レートの設定

[Update Rate] タブを使用して、CodeScape が各リージョンの情報を更新する周期を指定します。

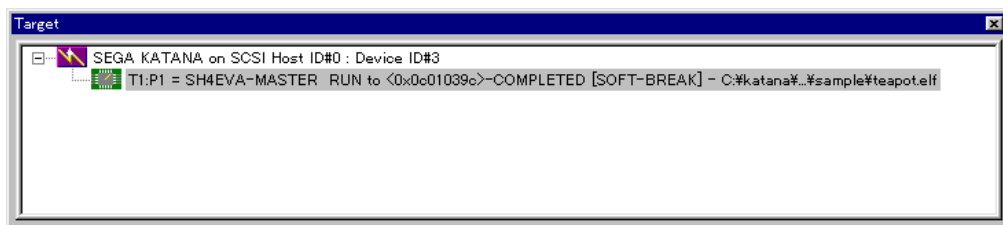


リージョン表示の更新レートにより Dev.Box がインタラプトされ、プログラムの実行が不安定になる場合は、[Foreground] と [Background] のスライダを [Min] に設定します。CodeScape がリージョンの情報を更新する周期を指定するには、次の操作を行います。

1. [Update] タブを選択します。
2. リージョンにフォーカスがある場合の更新レートを設定するには、[Foreground] スライダをドラッグします。表示を継続的に更新するには、スライダを [Max] に設定します（約 14Hz）。スライダを [Min] に設定すると、[Max] の約 1/10 で表示を更新します。
3. リージョンにフォーカスがない場合の更新レートを設定するには、[Background] スライダをドラッグします。表示を継続的に更新するには、スライダを [Max] に設定します。表示の更新をしないようにするには、スライダを [Min] に設定します。

2.3 [Target] ウィンドウ

CodeScape を実行すると、CodeScape は有効な Dev.Box を探して、それらを [Target] ウィンドウに表示します。[Target] ウィンドウには、CodeScape が接続されているすべての Dev.Box と、各 Dev.Box で使用可能なプロセッサが表示されます。各 Dev.Box に対するプロセッサステータスは、[Target] ウィンドウのプロセッサ表示に現れます。



新規のウィンドウを作成するとき、CodeScape は Dev.Box の選択されたプロセッサからのターゲット情報を使用します。

注意

[Target] ウィンドウは、メインウィンドウの上か下にドッキングしたり、フローティングすることができます。


[Target] ウィンドウのショートカットメニューの使用

Dev.Box のプロセッサの実行をコントロールするには、右クリックしてから、次の操作を行います。


メニュー	内容
Configure Processor...	現在のプロセッサに対する更新レートを設定。
Simulate Processor	シミュレータを実行。
Execution	プログラムの実行、停止、およびリスタート。プログラムは指定のアドレスまで実行。プログラムファイルのすべてを同時実行。同時実行中のすべてのプログラムを停止。シングルステップコマンドを使用するか、ステップコマンドを実行。
Breakpoints	ブレークポイントの追加、有効化、無効化、設定、リセット、または削除。
Reset Target	ソフトリセットまたはハードリセットの実行。Dev.Box をリセットすると、プログラムファイルの再ロードを要求されます。
Load Program File	Dev.Box の選択したプロセッサに、プログラムファイルをダウンロード。

2.3.1 Dev.Box のプロセッサの表示

Dev.Box のプロセッサを表示するには、次のいずれかを行います。

- ・ Dev.Box のステータス行をダブルクリック。
- ・  をクリック。

Dev.Box のプロセッサを非表示するには、次のいずれかを行います。

- ・ Dev.Box の表示行をダブルクリック。
- ・  をクリック。

2.4 [Input/Output] ウィンドウ

[Input/Output] ウィンドウは自動的に表示されます。



エラーと警告があれば、[Input/Output] ウィンドウに表示されます。

- ・ ビルドエラーが発生した場合は、そのエントリをダブルクリックするとエディタが起動され、ソースファイルをオープンしてエラーや警告を含む行が表示されます。

すべてのエラーと警告は [Input/Output] ウィンドウに表示されます。生成された情報をスクロールすることも可能ですし、[F4] キーを押してリスト表示されたエラー行に移動することもできます。

- ・ CodeScape の [Edit] リージョンは、プロジェクトファイル中のエラーや警告を含む最初の行を自動的に開きます。[Input/Output] ウィンドウは他のエラーを表示するのにも使用できます。[Edit] リージョンが表示されていない場合には、自動的に表示されます。
- ・ [Input/Output] ウィンドウをダブルクリックすると、エディタが起動してソースファイルを読み込み、エラーや警告を含む行に移動します。エディタの種類によっては、エラーや警告を含む行に移動しないものもあります。

注意

132 文字より大きいテキスト文字列は、[Log] リージョンに表示されるとき、切りつめられます。

注意

[Project Build] ウィンドウは、メインウィンドウの上または下にドッキングしたり、フローティングにすることができます。

[Build] タブでのショートカットメニューの使用

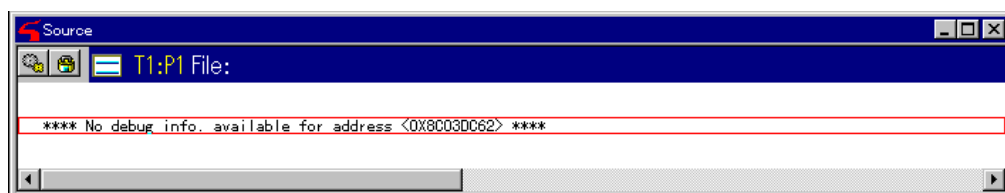
メニュー	内容
Setup Project...	プロジェクトを作成し、ビルドするためのファイルの位置を指定します。
Setup Editor...	プロジェクトで使用するエディタを指定します。
Make	プロジェクトをビルドして、それを現在のプロジェクトにセットします。
Stop Make	プロジェクトのビルドを中止します。
Next Error	リストの次のエラー行に移動します。
Clear	[Project Build] ウィンドウの内容を消去します。
Allow Docking	ウィンドウのオン / オフのドッキングをトグル操作します。
Hide	ウィンドウを隠します。

[Log] タブでのショートカットメニューの使用

メニュー	動作
Configure Log	[Log] タブを設定します。
Print	[Log] タブの内容を印刷します。
Save To File	[Log] タブファイルの内容を印刷します。
Execution	プログラムの実行、停止、再実行を行います。カーソル行まで、または指定したアドレスまでプログラムを実行実行します。すべてのプログラムを同時に実行します。すべてのプログラムを同時に停止します。シングルステップコマンドか、ステップコマンドを使用してください。
Breakpoints	ブレークポイントのオン / オフをトグル切り替えます。ブレークポイントを有効、無効、設定、リセット、削除します。
Reset Log	[Log] タブの内容を消去します。

2.5 [Source]リージョン

[Source]リージョンには、元のソースコードをデバックするコマンドがあります。



[Source]リージョンでのショートカットメニューの使用


ショートカットメニューのコマンドは、リージョンでのデバッグとソース表示の設定に使用します。リージョンの適当な個所を右クリックすると、ショートカットメニューを使用することができます。

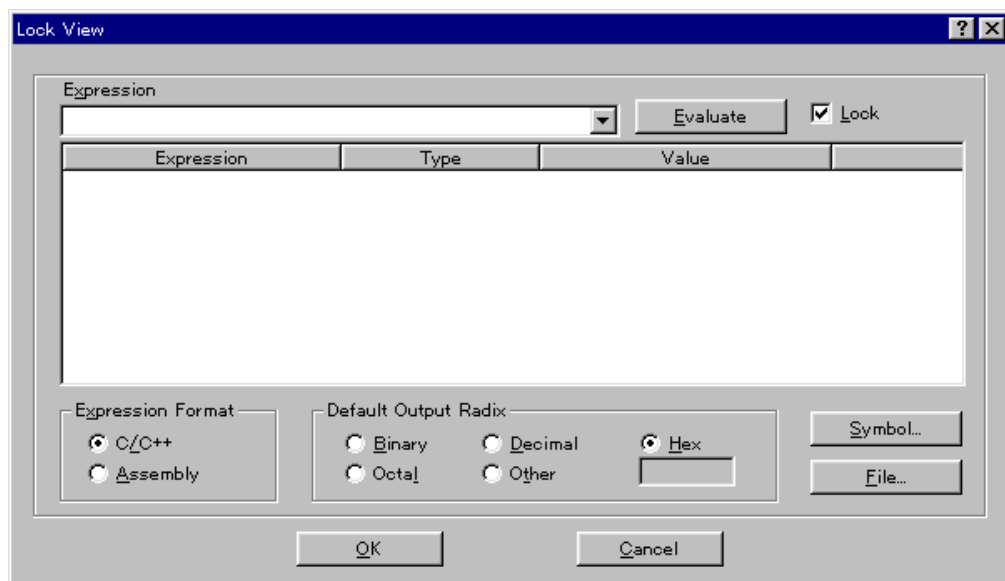
[Source]リージョンでのコピー

1. [Source]リージョンで、コピーするテキストを強調表示して選択します。
2. 右クリックし、[Copy]をクリックします。

選択した内容がコピーされ、クリップボードに貼り付けられます。

表示開始点を式にロック

1. [Source]リージョンのタイトルバーの  をクリックします。[Lock View] ダイアログボックスが表示されます。



2. リージョンの開始点の式を入力します。
3. [OK] ボタンをクリックします。

ソース表示の設定

ショートカットメニューからソース表示を設定するには、右クリックしてから、次のいずれかをクリックします。

メニュー	内容
Show Address	ソースコード行から生成された、最初のコード行に対応するアドレスを表示。
Show Line Nos.	左側にソースの各行の行番号を表示。

[Source] リージョンでのデバッグオプションの設定

[Source] リージョンでデバッグオプションを設定するには、右クリックしてから、次のいずれかをクリックします。



メニュー	内容
Execution	プログラムの実行、停止、リスタート。カーソル位置、または指定アドレスまでプログラムを実行。プログラムファイルのすべてを同時実行。同時実行中のすべてのプログラムを停止。シングルステップコマンドを使用するか、ステップコマンドを実行。
Breakpoints	ブレークポイントのオン、オフを切り替え。ブレークポイントの有効化、無効化、設定、および削除。

[Source] リージョンでのカーソル表示の設定

[Source] リージョンでカーソル表示の設定を行うには、右クリックしてから、次のいずれかをクリックを行います。

メニュー	内容
Set Cursor to PC	PC の値が指すソースコードを表示。
Set PC to Cursor	PC を現在のカーソル位置に変更。
Goto Address...	リージョンの開始点を示す式を入力。
Goto Source File...	必要なソースファイルを選択。
Tools	[Source] リージョンで検索。次の項目を検索。
Tab Width...	タブの幅を設定する値をスペースで入力。
Properties	フォントおよびカラーを設定。1 つのリージョン、リージョンタイプ、およびプロセッサについて更新レートを設定。タブ設定を変更。

[Source] リージョンと [Disassembly] リージョンのカーソルを同期

1. [Source] リージョンで、次のいずれかを行います。
 - ・ 右クリックして、[Synchronize Cursor] をクリック。
 - ・ [Source] リージョンタイトルバーで  をクリック。
2. [Disassembly] リージョンで、次のいずれかを行います。
 - ・ 右クリックして、[Synchronize Cursor] をクリック。
 - ・ [Disassembly] リージョンタイトルバーで  をクリック。

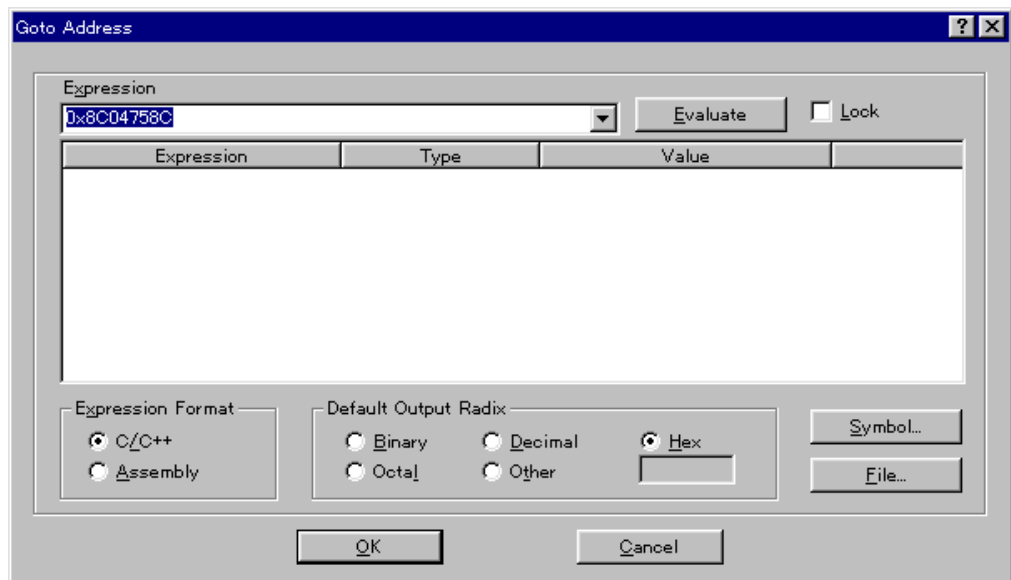
[Disassembly] リージョンと [Source] リージョンのカーソルは、これで同期します。フォーカスのあるリージョンでカーソルを移動すると、同期されたリージョンのカーソルは、対応するコード行に移動します。

注意

同一のウィンドウにあり、同一の Dev.Box のプロセッサに接続されているリージョンだけが、同期化をとることができます。

指定アドレスに進む

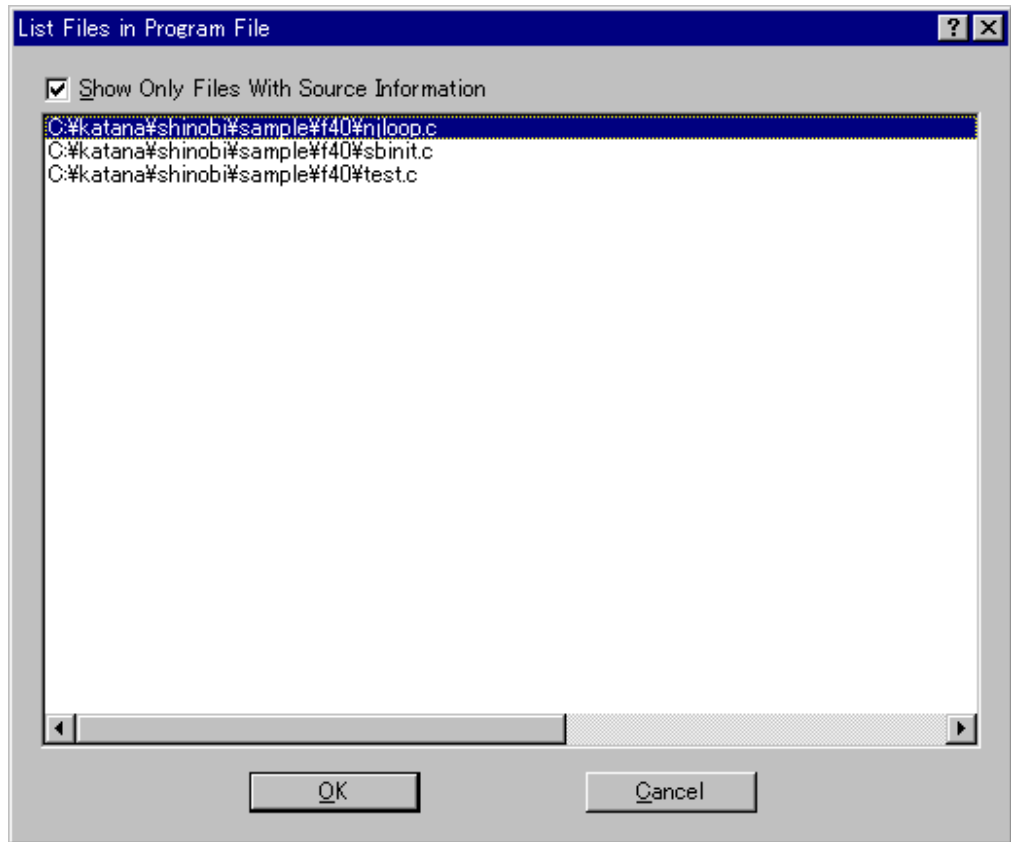
1. 次のいずれかを行います。
 - ・ [Edit] メニューをクリックして、[Go To] をクリック ([Ctrl] + [G] キー)。
 - ・ 右クリックして、[Goto Address] をクリック。
- [Goto Address] ダイアログボックスが表示されます (このダイアログボックスは、式エディタと同様に機能します)。



2. 指定アドレスを表す式を入力します。
3. [OK] ボタンをクリックします。

プログラムファイルで参照されるソースファイルに進む

1. 右クリックして、[Goto Source File] をクリックします。
[List Files in Program File] ダイアログボックスが表示されます。



2. もしソースファイルを参照可能にしたいなら、ソースの情報から選択します。
3. 必要なソースファイルを選択します。
4. [OK] ボタンをクリックします。

パスが正しくない場合は、エラーメッセージが [Source] リージョンに表示されます。
[Project] メニューをクリックし、[Edit Source Path] をクリックして、ソースファイルへの正しいパスを入力します。

注意

コードは、データ専用ファイルには生成されません。また「-g」コマンドがコンパイル中に設定されていない場合は生成されません。コードが生成されない場合は、エラーメッセージが表示されます。

指定した式の評価

1. リージョンで式を選択します。
2. 右クリックして、[Evaluate...] をクリックします。

タブ設定の変更

1. 右クリックして、[Properties] をポイントしてから、[Tab Width...] をクリックします。
[Change Tab Size] ダイアログボックスが表示されます。
2. タブを表すスペースの数を入力します。
3. [OK] ボタンをクリックします。

[Source] リージョンでの検索

1. [Source] リージョンで右クリックし、[Tools] をクリックしてから、[Find] をクリックします。
2. 検索する文字列テキストボックスに、検索文字列を入力します。
3. 単語の一部ではなく、単語全体を検索するには、単語単位で探すチェックボックスを選択します。
4. 検索で大文字と小文字を区別する場合は、大文字と小文字を区別するチェックボックスを選択します。
5. [OK] ボタンをクリックします。

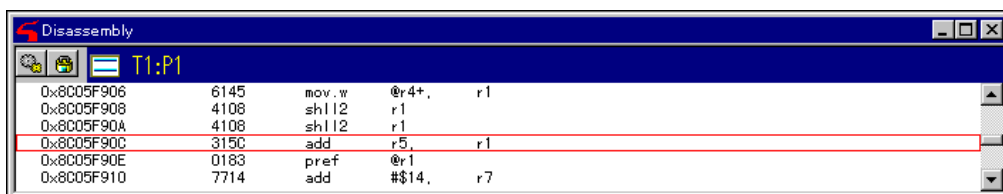
検索は現在のカーソル位置から始まり、ファイルの最後まで検索します。

注意

右クリックしてから、[Find next] をクリックすると、同一項目の検索を続けます。

2.6 [Disassembly] リージョン

[Disassembly] リージョンには、インストラクションレベル (アセンブルコード) でプログラムをデバッグするためのコマンドがあります。



[Disassembly] リージョンでのショートカットメニューの使用

ショートカットメニューのコマンドには、リージョンでのデバッグに使用するものと、逆アセンブルの表示を設定するものがあります。ショートカットメニューを使用するには、リージョン内で右クリックします。


[Disassembly] リージョンでのコピー

1. [Disassembly] リージョンで、コピーしたいテキストを強調表示して選択します。
2. 右クリックしてから、[Copy] をクリックします。

選択内容がコピーされ、クリップボードに貼り付けられます。

[Disassembly] リージョンのロック

表示開始点は PC、レジスタ、メモリのアドレスにロックすることができます。

1. [Disassembly] リージョンタイトルバーで  をクリックします。[Goto Address] ダイアログボックスが表示されます。
2. [Expression] テキストボックスに、次のいずれかの式を入力します。
 - ・表示を PC に固定する PC の値。[OK] ボタンをクリックします。
 - ・表示をレジスタに固定するレジスタ名。[OK] ボタンをクリックします。
 - ・ [Expression] テキストボックスで、表示をメモリアドレスに固定するそのメモリのアドレスを入力します。[OK] ボタンをクリックします。

注意

表示開始点のロックを解除するには  を再度クリックします。

逆アセンブル表示の設定

ショートカットメニューから逆アセンブルの表示法を設定するには、右クリックしてから、次の操作を行います。

メニュー	表示
Show Address	逆アセンブルコードのロケーションアドレス
Show Labels	逆アセンブルコードのシンボルラベル
Show Opcode Words	逆アセンブルリージョンの OP コードワード
Show Hexadecimal	16 進によるオペランドの値
Show Uppercase	大文字によるインストラクション
Show Symbols	シンボルのオペランドの値
Show EAs & Lits	実効アドレスとリテラル

[Disassembly] リージョンでのデバッグオプションの設定

[Disassembly] リージョンでデバッグコマンドを使用するには、右クリックしてから、次のメニューをクリックします。



メニュー	内容
Execution	プログラムの実行、停止、リスタート。プログラムをカーソルまで、または指定アドレスまで実行。すべてのプログラムを同時実行。同時実行中のすべてのプログラムの停止。シングルステップコマンドの使用またはステップコマンドの実行。
Breakpoints	ブレークポイントをオン、オフに切り替え。ブレークポイントの有効化、無効化、設定、およびリセット。

[Disassembly] リージョンでのカーソル表示の設定

[Disassembly] リージョンで、カーソル表示の設定をするには、右クリックしてから、次のメニューをクリックします。

メニュー	内容
Set Cursor to PC	PC の値に対応するソースコードを表示。
Set PC to Cursor	PC を現在のカーソルの位置に変更。
Goto Address...	目的のリージョン開始点を表す式を入力。
Tools	[Disassembly] リージョンでの検索。最後に行った検索の繰り返し。アドレスを指定して、ファイルに逆アセンブル。
Properties	フォントとカラーの設定。1 つのリージョン、リージョンタイプごと、および各プロセッサごとに、更新レートの設定。タブ設定の変更。

[Disassembly] リージョンと [Source] リージョンでカーソルを同期

- [Source] リージョンで、次のいずれかを行います。
 - 右クリックして、[Synchronize Cursor] をクリック。
 - [Source] リージョンのタイトルバーで  をクリック。
- [Disassembly] リージョンで、次のいずれかを行います。
 - 右クリックして、[Synchronize Cursor] をクリック。
 - [Disassembly] リージョンのタイトルバーで  をクリック。

[Disassembly] リージョンと [Source] リージョンのカーソルは、これで同期します。フォーカスのあるリージョン中でカーソルを移動すると、同期したリージョンのカーソルは、対応するコード行に移動します。

注意

同一ウィンドウ中の同一の Dev.Box のプロセッサに接続されているリージョンだけが同期をとることができます。

アドレスにジャンプ

1. 次のいずれかを行います。
 - ・ [Edit] メニューをクリックして、[Go To] をクリック ([Ctrl] + [G] キー)
 - ・ 右クリックして、[Goto Address] をクリック。[Goto Address] ダイアログボックスが表示されます。
2. アドレスを入力します。
3. [OK] ボタンをクリックします。

指定した式の評価

1. リージョンで式を選択します。
2. 右クリックして、[Evaluate...] をクリックします。

[Disassembly] リージョンでの検索

1. [Source] リージョンで右クリックして、[Tools] をポイントしてから、[Find] をクリックします。
2. [Find for] テキストボックスに、検索文字列を入力します。
3. [Start Address] テキストボックスに、開始アドレスを入力します。
4. 検索で大文字と小文字を区別する場合は、[Case sensitive] チェックボックスを選択します。
5. 次のオプション ボタンのいずれかを選択します。
 - ・ [Length] (データ量)
 - ・ [End Address]
6. 下のテキストボックスに、検索項目を入力します。
7. 次のオプション ボタンのいずれかを選択します。
 - ・ [All Fields] (デフォルト) [Words] [Opcode] [OpSrc] [OpDest] または [Label] (アドレス)
8. [OK] ボタンをクリックします。

注意

同一項目の検索を続けるには、右クリックして、[Find Next] をクリックします。

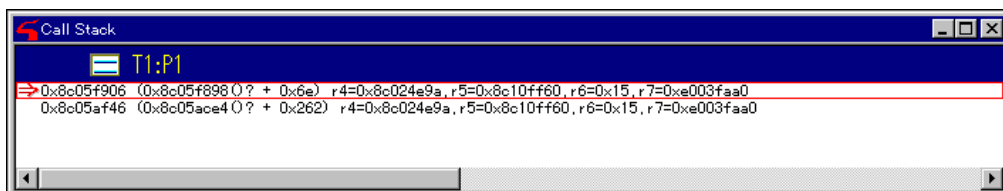
アドレスを指定して、ファイルに逆アセンブル

この汎用ダイアログボックスは、メモリブロックまたは逆アセンブル結果を 16 進数でファイルに書き込むためのものです。

1. [Disassembly] リージョンで右クリックし、[Tools] をポイントしてから、[Disassemble to File] をクリックします。
2. [Destination filename] テキストボックスに、書き込むファイルのファイル名を入力します。

3. [Start Address] テキストボックスに、開始アドレスを 16 進数で入力します。
4. 次のいずれかを行います。
 - ・ [Length] を選択して、長さを 16 進数で入力する。
 - ・ [End Address] を選択して、終了アドレスを 16 進数で入力する。
5. [OK] ボタンをクリックします。

2.7 [Call Stack] リージョン



[Call Stack] リージョンを使用して、アクティブな関数コールのリストを表示することができます。Call Stack を表示することにより、関数実行のトレースを容易に行うことができます。例えば、Dev.Box が停止しているときにブレークポイントが発生すると、CodeScape は [Call Stack] リージョンのリストの一番上に現在の関数の名前、ラベルまたはアドレスを表示します。現在の関数の下に実行トレースの履歴が表示されます。開始点はリストの一番下になります。

アクティブな [Source]、[Disassembly]、[Watch]、および [Local Watch] リージョンで特定の関数コールに移動するためにコールスタックを使用するには、以下の操作を行います。

- ・ [Call Stack] リージョンで関数をダブルクリックします。CodeScape は、アクティブリージョン内でその関数が発生すると、その関数を強調表示します。

[Call Stack] リージョンでのショートカットメニューの使用

メニュー	内容
Show Parameter Names	関数パラメータの表示のオン、オフ。
Show Parameter Types	関数パラメータ タイプの表示のオン、オフ。
Show Parameter Values	関数パラメータの値の表示のオン、オフ。
Show Parameter Registers	関数パラメータのレジスタの表示のオン、オフ。
Show Octal	関数の値を 8 進数で表示。
Show Decimal	関数の値を 10 進数で表示。
Show Hexadecimal	関数の値を 16 進数で表示。
Execution	プログラムの実行、停止、リスタート。プログラムを指定アドレスまで実行。すべてのプログラムファイルを同時実行。同時実行中のすべてのプログラムの停止。シングルステップコマンドの使用またはステップコマンドの実行。
Breakpoints	ブレークポイントのオン、オフの切り替え。ブレークポイントの有効化、無効化、設定、およびリセット。
Properties	フォントおよびカラーの設定。1 つのリージョン、リージョンタイプごと、および各プロセッサごとに更新レートを設定。

2.8 [Watch]リージョンと[Local Watch]リージョン

[Watch]リージョンと[Local Watch]リージョンには、変数と式を1行ごとに表示します。

各行には4列あり、式は3番目の列に、その値(場合による)は4番目の列に表示されます。

- ・最初の列に「.」があると、式にウォッチポイントが指定できます。
- ・2番目の列に「+」があると、式が展開できます。
- ・3番目の列に「-」があると、式が折り畳めます。

[Watch]リージョンまたは[Local Watch]リージョンでは、以下を行うことができます。

- ・実行ステップごとのデータ値の変化を強調表示。
- ・[Watch]リージョンと[Local Watch]リージョンでの式の値の編集。

注意 実在の式だけを編集することができます。

[Watch]リージョンまたは[Local Watch]リージョンにおける、C++名のディマングル

C++名のディマングルをすべての変数名について実行することができます。これは、元のソースに出現する通りの名前の記号を入力できることを意味します。次の目的でデータを見ることができます。

- ・構造体の階層表示の展開と折り畳み
- ・構造体のメモリ内の位置の表示
- ・変数値の編集

以下を含めて、Cのすべてのタイプがサポートされています。

- ・structs
- ・unions
- ・arrays
- ・enumeration (enum)
- ・floats/double

式の展開

式を展開すると、各子式はインデントされ、親式の下に表示されます。
たとえば、次の通りです。

```
parent
  child
  child
```

式は、展開された次のものに追加されます。

- ・ポインタ (Pointers) 。ディリファレンス項目を示します
- ・配列 (Arrays) 。式は配列の各要素に追加されます
- ・構造体 (Structures) 。式は各メンバに追加されます

2.8.1 [Watch] リージョン

[Watch] リージョンで、変数と式を入力することができます。[Watch] リージョンにおける変数のスコープは、グローバルです。式がプログラム実行中にスコープを超えると、メッセージが表示されます。



- ・スタティック変数の場合のように、式の値を決定できる場合は、その値がリージョンに表示されます。
- ・式の値が決定できない場合は、値は表示されません。
- ・式がスコープ内に戻ると、その値が表示されます。

[Watch] リージョンのショートカットメニューの使用

メニュー	内容
Cut	編集ファイルの現在の選択内容を切り取り、クリップボードに貼り付け。
Copy	編集ファイルの現在の選択内容をコピーし、クリップボードに貼り付け。
Paste	クリップボードの内容を現在のカーソルの位置に挿入。
Delete	構造体の一部を削除。
Open/Close	構造体や配列を展開 / 折り畳む。
Insert	新規のウォッチ式を挿入。
Append	現在のリストの最後に変数を挿入。
Show Octal	ウォッチ式を 8 進数で表示。
Show Decimal	ウォッチ式を 10 進数で表示。
Show Hexadecimal	ウォッチ式を 16 進数で表示。
Edit Watch Value...	変数またはウォッチ式の値を変更。
Execution	プログラムの実行、停止、リスタート。プログラムを指定アドレスまで実行。すべてのプログラムファイルを同時実行。同時実行中のすべてのプログラムの停止。シングルステップコマンドの使用またはステップコマンドの実行。
Breakpoints	ブレークポイントのオン、オフの切り替え。ブレークポイントの有効化、無効化、設定、およびリセット。
Highlight Changes	変化したメモリの位置を表示。
Properties	フォントおよびカラーの設定。1 つのリージョン、リージョンタイプごと、各プロセッサごとに更新レートの設定。

[Watch] リージョンでデータのブラウズ

記号または変数の追加

[Watch] リージョンで、次のいずれかを行います。

- ・ 右クリックし、[Insert] をクリックして、記号または変数を現在のカーソル位置に追加。
- ・ 右クリックし、[Append] をクリックして、記号または変数を、変数の現在のリストの最後に追加。
- ・ [Enter] キーを押して、新規の記号または変数を、現在のカーソル位置に入力。

構造体または配列の展開

展開する構造体または配列を選択してから、次のいずれかを行います。

- ・ 「 + 」をクリックする。
- ・ [スペース] キーを押す。
- ・ 右クリックして、[Open/Close] をクリックする。

構造体または配列の折り畳み

折り畳む構造体または配列を選択してから、次のいずれかを行います。

- ・「 - 」をクリックする。
- ・ [スペース] を押す。
- ・ 右クリックして、[Open/Close] をクリックする。

[Watch] リージョンで変数の編集

変数またはウォッチ式の値を変更します。

次のいずれかを行います。

- ・ 変更する値を選択して、[Enter] キーを押す。
- ・ [Ctrl] + [Alt] + [E] キーを押す。

Expression Evaluator ダイアログボックスが表示されます。式を入力して、[OK] ボタンをクリックします。

変数のデータ値を編集（構造体、配列、または結合）

1. 編集する値をダブルクリックします。
2. 値を編集します。
3. 次のいずれかを行います。
 - ・ [Enter] キーを押す。
 - ・ [Ctrl] + [Alt] + [E] キーを押す、[Expression Evaluator] ダイアログボックスを表示する。

親式と子式を削除するには

1. 構造体または配列を展開します。
2. 削除するコンポーネントを選択してから、次のいずれかを行います。
 - ・ 右クリックして、[Delete] をクリックする。
 - ・ [Delete] キーを押す。

注意

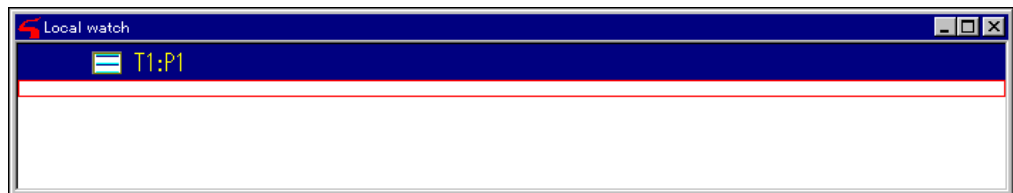
親式を削除すると、子式もリージョンから削除されます。

親式を削除して、子式をすべて 1 レベル上げるには

1. 構造体または配列を展開します。
2. 削除するコンポーネントを選択します。
3. [Shift] + [Delete] キーを押します。

2.8.2 [Local Watch] リージョン

[Local Watch] リージョンには、PC (プログラムカウンタ) の現在の位置からビュー内のすべてのローカル変数が自動的に表示されます。関数のスコープ内であれば、これらの変数は自動的にディスプレイに追加されます。



[Local Watch] リージョンでのショートカットメニューの使用

メニュー	内容
Copy	Editor ファイルで現在の選択内容をコピーし、クリップボードに貼り付け。
Delete	式または式構造体の一部を削除。
Open/Close	構造体や配列を展開 / 折り畳む。
Show Octal	ローカルウォッチ式を 8 進数で表示。
Show Decimal	ローカルウォッチ式を 10 進数で表示。
Show Hexadecimal	ローカルウォッチ式を 16 進数で表示。
Edit Local Value	変数またはウォッチ式の値を変更。
Execution	プログラムの実行、停止、リスタート。プログラムを指定アドレスまで実行。すべてのプログラムファイルを同時実行。同時実行中のすべてのプログラムの停止。シングルステップコマンドの使用またはステップコマンドの実行。
Breakpoints	ブレークポイントのオン、オフの切り替え。ブレークポイントの有効化、無効化、設定、およびリセット。
Highlight Changes	変化したメモリの位置を表示。
Cache Expanded Symbols	拡張された関数の状態をセッションファイルに保存。次回その関数がアクセスされたときに、自動的に保存時の拡張された状態に戻る。
Properties	フォントおよびカラーの設定。1 つのリージョン、リージョンタイプごと、および各プロセッサごとの更新レートの設定。

[Local Watch] リージョンでデータをブラウズ

構造体または配列の展開

展開する構造体または配列を選択してから、次のいずれかを行います。

- ・ 「 + 」をクリックする。
- ・ [スペース] キーを押す。
- ・ 右クリックして、[Open/Close] をクリックする。

構造体または配列を折り畳む

折り畳む構造体または配列を選択してから、次のいずれかを行います。

- ・ 「 - 」をクリックする。
- ・ [スペース] キーを押す。
- ・ 右クリックして、[Open/Close] をクリックする。

[Local Watch] リージョンで変数の編集

変数またはウォッチ式の値の変更

1. 次のいずれかを行います。

- ・ 変更する値を選択し、[Enter] キーを押す。
- ・ [Ctrl] + [Alt] + [E] キーを押す。

[Expression Evaluator] ダイアログボックスが表示されます。

2. 式を入力します。

3. [OK] ボタンをクリックします。

親式と子式をすべて削除するには

1. 構造体または配列を展開します。

2. 削除するコンポーネントを選択してから、次のいずれかを行います。

- ・ 右クリックして、[Delete] をクリックする。
- ・ [Delete] キーを押す。

注意

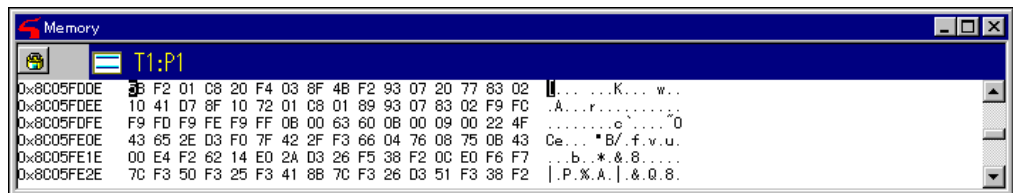
親式を削除すると、すべての子式もリージョンから削除されます。

親式を削除して、子式をすべて 1 レベル上げるには

1. 構造体または配列を展開します。
2. 削除するコンポーネントを選択します。
3. [Shift] + [Delete] キーを押します。

2.9 [Memory] リージョン

[Memory] リージョンを使用すると、Dev.Box の指定アドレスのメモリ内容を表示することができます。[Memory] リージョン内には、ASCII 文字、バイト、ワード、またはロングとして、メモリの内容を表示することができます。現在の [Memory] リージョンでメモリ内容が変更されないように、メモリ領域を書き込み保護することができます。



注意

[Memory] ウィンドウをスクロールするに従い、スライダの速度が速くなります。スライダの一番上 (または下) をクリックすると、中央位置に戻ります。


[Memory] リージョンでのショートカットメニューの使用

メニュー	内容
Display Bytes	バイトとしてメモリ内容を表示。
Display Words	ワードとしてメモリ内容を表示。
Display Longs	ロングとしてメモリ内容を表示。
Display Quadwords	クワッドワードとしてメモリ内容を表示。
Display ASCII	各バイト メモリの ASCII 値を表示。
Highlight Changes	Dev.Box のメモリの変更位置を表示。
Set Bytes Per Line...	ラインごとに指定バイト数を表示。
Edit ASCII	[Memory] リージョンの ASCII 値を変更。
Edit Memory Value	[Memory] リージョンの値を変更。
Follow Pointer	メモリ内のポインタに従う。
Goto Address...	オリジンを設定。
Write Protect	書き込み保護のオン、オフ。
Execution	プログラムの実行、停止、リスタート。カーソルの位置まで、または指定アドレスを実行するまで、プログラムを実行。プログラムファイルのすべてを同時実行。同時実行中のすべてのプログラムを停止。シングルステップコマンドを使用するか、ステップコマンドを実行。
Breakpoints	ブレークポイントのオン、オフの切り替え。ブレークポイントの有効化、無効化、設定、およびリセット。
Tools	メモリ内のパターンを検索。最後の検索を繰り返す。メモリ範囲をデータで埋める。メモリブロックを 16 進数でファイルに書き込む。
Properties	フォントとカラーの設定。1 つのリージョン、リージョンタイプごと、各プロセッサごとの更新レートを設定。

メモリ内容の表示

[Memory] リージョンの表示

次のいずれかを行います。

- ・ [Region] メニューをクリックして、[Type] をポイントし、[Memory] をクリック。
- ・ [Region] ツールバーの  をクリック。
- ・ リージョン内で、[Ctrl] + 右クリックして、[Memory] をクリック。


オリジンの設定

1. 右クリックして、[Goto Address...] をクリックします。
[Goto Address...] ダイアログボックスが表示されます。
2. 新規のオリジンに対するアドレスまたは記号を入力します。
3. 不正な記号を入力すると、警告が表示されます。
4. [OK] ボタンをクリックします。

注意

オリジンは、最初 PC の値に設定されています。オリジンを式で設定することもできます。

常に指定アドレスのメモリ内容を表示

1. 次のいずれかを行います。
 - ・ [Edit] メニューをクリックしてから、[Go To...] をクリック ([Ctrl] + [G] キー)。
 - ・ 右クリックして、[Goto Address...] をクリック。
 - ・ [Memory] リージョンタイトルバーの  をクリック。[Goto Address...] ダイアログボックスが表示されます。
2. 式のリストから、メモリアドレスまたは式を入力するか選択します。
3. [Lock] を選択して、[OK] ボタンをクリックします。

メモリ内のポインタに従う

1. 表示を [Display Longs] に変更します。
2. ポインタの値のメモリアドレスを選択します。
3. 右クリックして、[Follow Pointer] をクリックします ([Ctrl] + [T] キー)。
[Memory] リージョンのオリジンが変わり、ポインタの値によって指定されたアドレスのメモリ内容が表示されます。

メモリのブロックを 16 進数でファイルに書き込む

1. [Memory] リージョンで右クリックして、[Tools] をポイントしてから、[Hex Dump to File] をクリックします。[Hex Dump to File] ダイアログボックスが表示されます。
2. [Destination filename] テキストボックスで、書き込むべきファイルのファイル名を入力します。[Start Address] テキストボックスで、開始アドレスを 16 進数で入力します。
3. 次のいずれかを行います。
 - ・ [Length] を選択して、メモリのブロック長を 16 進数で入力。
 - ・ [End Address] を選択して、最終アドレスを 16 進数で入力。
4. [OK] ボタンをクリックします。これで、メモリの指定ブロックがファイルに書き込まれます。

メモリの編集

[Memory] リージョンで値を変更

1. [Memory] リージョンで、次のいずれかを行います。
 - ・ 「+」と「-」キーを使用して、現在の値をインクリメントしたり、デクリメントする。
 - ・ ダブルクリックするか、[Enter] キーを押してから、既存のバイト、ワード、またはロング値に上書き入力する。
 - ・ 右クリックして、[Edit Memory Value...] をクリックする。
 - ・ [Ctrl] + [Alt] + [E] キーを押す。[Enter a Value] ダイアログボックスが表示されます。
2. 式を入力します。
3. [OK] ボタンをクリックします。

注意

現在の基数に対する正しい値を入力してください。CodeScape では、メモリ値はすべて 16 進数で表示します。

メモリを指定のデータで埋める

メモリ範囲を指定のデータで埋めるには、次の操作を行います。

1. [Memory] リージョンで、右クリックして、[Tools] を選択して、[Fill..] をクリックします。
2. [Fill Expression] テキストボックスに、値を入力します。
3. [Start Address] テキストボックスに、値を入力します。
4. [End address] または [Length] を選択します。
5. 下のテキストボックスに値を入力します。
6. モード (Mode) をテキスト (ASCII)、Byte、Word、Long または Quad として選択します。
7. 次のいずれかを行います。
 - ・ [Convert Native Endian] チェックボックスを選択して、実メモリの値を表示。
 - ・ [Convert Native Endian] チェックボックスをオフにして、メモリをバイト順に格納。
8. [OK] ボタンをクリックします。

メモリの検索

メモリ領域を定義して、データパターンを指定して検索するには、次の操作を行います。

1. [Memory] リージョンで、右クリックして、[Tools] を選択します。
2. [Find] をクリックします。[Find In Memory] メモリダイアログボックスが表示されます。
3. [Find Pattern] テキストボックスに、検索文字列を入力します。

バイナリ、8 進数、10 進数、16 進数モードでは、検索パターンはカンマまたはセミコロン (等価) のいずれかで区切ります。

4. [Start Address] テキストボックスに、値を入力します。

デフォルト値は、リージョンの開始アドレスです。検索を実行したことがない場合は、現在のメモリブロックの開始アドレスが使用されます。

5. [End Address] または [Length] を選択します。

最終アドレスのデフォルト値は、[Memory] リージョンに最後に表示されたバイトです。

6. 下のテキストボックスに値を入力します。
7. データ長として、Byte、Word、Long を選択します。
8. [Forward] または [Reverse] を選択して、検索の方向を指定します。
9. [OK] ボタンをクリックします。

注意

指定した検索が不正な場合、「Invalid Address」がフィールドに表示されるので、これを編集します。

検索が一致した場合は、そのアドレスが表示されます。無効なメモリ領域、書き込み専用メモリ、モニタ用などの重要なメモリ上では、検索はスキップされます。

データ長

データ長を利用すると、検索パターンと Dev.Box のメモリ内のデータとの整列がとられます。これにより、検索パターンとメモリ内容の比較の方法を指定します。可能なデータ長は、選択するモードによって異なります。

有効なモードとデータ長の組み合わせ

モード	有効なデータ長
Binary	バイナリ、ワード、ロング
Decimal	バイト
Hex	バイナリ、ワード、ロング
Text	なし

16 進数とバイトの幅が設定されている場合、次のパターンは等価です。

FF , FF , FF , FF , 34 , DC

FF ; FF ; FF ; FF ; 34 ; DC

FFFFFFFF34DC

「¥」指定子

「¥」指定子を使用して、特殊文字を検索テキストに入れることができます。

注意

必ず検索テキストは引用符で囲ってください。

例

パターンの "How are you¥?" は、"How are you?" を検索します。

「？」ウィルドカード

ウィルドカード文字「？」は、バイナリ、16進数モードで使用することができます。常にマッチするように16進数で1つのニブル(4ビット)、バイナリモードで1ビットを指定するには、「？」を使用します。

例

16進数モード:

FF?F は、FF0F, FF1F, FF2F, ..., FFFF を検索します

バイナリモード:

????1111 は、00001111, 00011111, ..., 11111111 を検索します。

「@」ウィルドカード

ウィルドカード文字の「@」は、テキストモードで使用することができます。2バイト文字1文字を指定するには、「@」を使用します。

自動パディング

バイナリ、10進数、16進数モードでは、検索パターンは自動的に左側をパディングされます。パディングは、「0」または「？」で区切り文字に依存します。

カンマ区切り

検索パターンをカンマ(デフォルト)で区切ると、左側はゼロパディングされます。

例えば、16進数モードのバイト幅の場合、

FFFFFFFF34DC と FF, FF, FF, FF, 34, DC

は、同一の検索となります。

最初の検索パターンのカンマ区切りが暗示的に使用されます。

さらに別の例、16進数モードとワードの場合、

「f, 87d, a」は、自動的に、000F, 087D, 000A になります。

「f87da」は、自動的に、000F, 87DA になります。

「,」は、自動的に、0000 になります。

注意

カンマを単独で使用すると、パターン「0000」が作成されます。この機能の使用には注意が必要となります。例えば、「,7」は、「0000,0007」になります。

セミicolon区切り

検索パターンをセミcolonで区切ると、ワイルドカードの「?」でパディングされます。例えば、16進数モードでワード幅の場合、

「f;8d;a」は、自動的に、??? F,? 87D,??? A になります。

「f;87da」は、自動的に、??? F,87DA になります。

「;」は、自動的に、???? になります。

等価な検索パターン

カンマとセミcolonの区切り文字を使用すると、同じ検索パターンを異なる手段で検索することができます。次のパターンは、16進数とワードが設定されていると、同じになります。

FF,FF,FF,FF,34,DC

FF;FF;FF;FF;34;DC

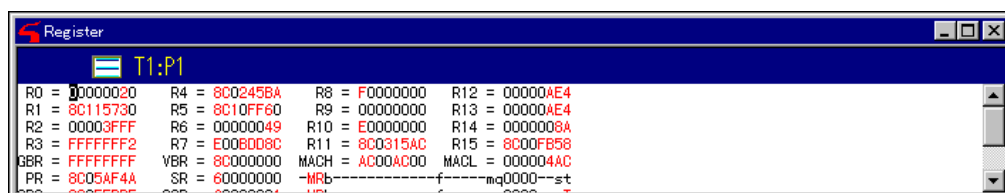
FFFFFFFF34DC

カンマとセミcolonの区切り文字を混ぜると、詳細な検索パターンを作成することができます。例えば、16進数とロング幅の場合、

f;f0f0f0f0,ffffff?,7は、??????? F,F0F0F0F0,FFFFFFFF?,??????? 7 になります。

2.10 [Register] リージョン

[Register] リージョンには、プロセッサのレジスタブロックとフラグの内容が表示されます。



[Register] リージョンでショートカットメニューを使用

メニュー	内容
Increment Register	現在の増分値 (1 がデフォルト) をレジスタの内容に適用
Decrement Register	現在の減分値 (1 がデフォルト) をレジスタの内容に適用。
Change Inc/Dec Value...	増分値 / 減分値を変更。
Highlight Changes	最後の操作で変化した位置を表示。
Write Protect	現在アクティブな [Register] リージョンにデータが書き込まれないように保護。
Edit Register	選択したレジスタ値を変更。
Column Format	レジスタを 2 列または 4 列で表示。[Auto] を選択すると、CodeScape が自動的に選択。
Show Banked Register	バンク化されたレジスタ表示のオン、オフ。
Show Float Register	フローティングポイントレジスタ表示のオン、オフ。
Execution	プログラムの実行、停止、リスタート。カーソルの位置まで、または指定のアドレスを実行するまで、プログラムを実行。プログラムファイルのすべてを同時実行。同時実行中のすべてのプログラムを停止。シングルステップコマンドを使用するか、ステップコマンドを実行。
Breakpoints	ブレークポイントのオン、オフの切り替え。ブレークポイントの有効化、無効化、設定、およびリセット。
Tools	現在のレジスタブロックを保存。最後に保存したレジスタブロックを回復。
Properties	フォントとカラーの設定。1 つのリージョン、リージョンタイプ、各プロセッサに対する更新レートを設定。

[Register] リージョンの表示

[Region] をクリックして、[Type] をポイントし、[Register] をクリックします。

表示形式の変更

デフォルトでは、レジスタは使用可能な領域に表示されます。2 列または 4 列で表示するように、設定することができます。

レジスタを 2 列で表示

次のいずれかを行います。

- ・ 右クリックして、[Column Format] をポイントしてから、[2 Columns] をクリックする。
- ・ [Ctrl] + [2] キーを押す。

レジスタを 4 列で表示

次のいずれかを行います。

- ・ 右クリックして、[Column Format] をポイントしてから、[4 Columns] をクリックする。
- ・ [Ctrl] + [4] キーを押す。

レジスタを使用可能な領域に表示

次のいずれかを行います。

- ・ 右クリックして、[Column Format] をポイントしてから、[Auto Format] をクリックする。
- ・ [Ctrl] + [0] キーを押す。

レジスタ値の編集

レジスタ値の変更

1. 挿入ポイントを、変更するレジスタの値に移動します。
2. 次のいずれかを行います。
 - ・ 「 + 」または「 - 」を使用して、現在値を増加または減小させる。
 - ・ 新規の値を挿入ポイントに入力する。
 - ・ レジスタをダブルクリックして値または式を入力し、[Enter] キーを押す。[Expression Evaluator] ダイアログボックスが表示される。
 - ・ [Ctrl] + [Alt] + [E] キーを押し、[Expression Evaluator] ダイアログボックスを表示する。

注意 英数字は、大文字で表示されます。

増分値 / 減分値の変更

1. 右クリックして、[Change Inc/Dec Value...] をクリックします。
[Register Increment/Decrement] ダイアログボックスが表示されます。
2. レジスタを増分するか減分する値を入力して、[OK] ボタンをクリックします。

レジスタのライトプロテクト

1. リージョンで右クリックします。
2. 次のいずれかを行います。
 - ・ [Write Protect] が選択されていないければ、[Write Protect] をチェックする。
 - ・ [Write Protect] が選択されていれば、ショートカットメニューを終了する。

注意

ライトプロテクトは、プロテクトされているリージョン内でのみレジスタ値が変更されないようにします。他のリージョンでは変更ができ、リージョンに反映されます。

現在の PC のインストラクションに式を入力

1. レジスタ値をダブルクリックしてから、次のいずれかを行います。
 - ・ 式を入力して、[Enter] キーを押す。
 - ・ 右クリックして、[Edit Register...] をクリックする([Ctrl] + [Alt] + [E] キー)。[Register Evaluation] ダイアログボックスが表示されます。
2. 式を入力します。
3. [OK] ボタンをクリックします。

注意

不正な式を入力すると、[Register Evaluation] ダイアログボックスは Invalid Register Expression を表示します。

レジスタの状態を保存

右クリックして、[Tools] をポイントしてから、[Save Register] をクリックします。

注意

各 Dev.Box のプロセッサのレジスタの状態は、一度に 1 つの状態だけ保存され、スタックされません。また、レジスタの状態は CodeScape に内部的に保存され、ファイルには保存されません。

レジスタの状態の読み出し

右クリックして、[Tools] をポイントしてから、[Restore Register] をクリックします。

日立プロセッサの [Register] リージョンの表示

汎用レジスタ

[Register] リージョンには、R0 から R15 の 16 個の汎用レジスタ (Rn) の値が表示されます。

R0 : 一部のインストラクションでは、特定のソースレジスタまたはデスティネーションレジスタとして機能します。R0 は、次のモードではインデックスレジスタとして機能します。

- ・間接インデックスレジスタアドレスモード
- ・間接インデックス GBR アドレスモード

R15 : スタックポインタ (SP) として機能します。

制御レジスタ

[Register] リージョンには、SR (ステータスレジスタ)、GBR (グローバルベースレジスタ)、および VBR (ベクトルベースレジスタ) の値が表示されます。

システムレジスタ

[Register] リージョンには、MACH と MACL (high and low multiply and accumulate register)、PR (プロシージャレジスタ)、および PC (プログラムカウンタ) が表示されます。MACH と MACL レジスタには、乗算とそのアキュムレーション操作の結果が格納されます。PR には、サブルーチンからの戻りアドレスが格納されます。

ステータスレジスタの値の変更

1. 変更したいレジスタの値に挿入ポインタを移動します。
2. 次のいずれかの操作を行います。
 - ・「 + 」, 「 - 」で現在の値を増減します。
 - ・新しく値を入力します。
 - ・[Ctrl] + [Alt] + [E] キーを押して [Expression Evaluation] ダイアログボックスを表示します。

注意

16 進数の「A」～「F」は大文字で表示されます。

ステータスレジスタ (SR、SSR、FPSR) フラグの設定

1. 挿入ポイントを、設定するフラグに移動します。
2. 次のいずれかを行います。
 - ・「1」を押し、フラグを設定。ビットフラグは小文字で表示される。
 - ・「0」を押し、フラグをクリア。ビットフラグは大文字で表示される。

注意

[スペース]キーはステータスレジスタのトグル切り替えに使用します。

フラグは、[Register] リージョンに表示され、Dev.Box のプロセッサをリセットした後の値が表示されます。

フラグ	表示内容
T ビット	MOVT、CMP/cond、TAS、TST、BT (IBT/S)、BF (BF/S)、SETT および CLRT インストラクションは T ビットを使用して、真 (1) または偽 (0) を表示。ADDV/C、SUBV/C、DIV0U/S、DIV1、NEGC、SHAR/L、SHLR/L、ROTR/L および ROTCR/L インストラクションも T ビットを使用し、carry/borrow またはオーバーフロー / アンダーフローを表示。
S ビット	乗算 / アキュムレーション命令が使用。
ビット 2、3 および 10-31 (予約ビット)	常に、0 として読み込み、0 として書き込み。
ビット 3 - 10	インタラプトマスクビット
M および Q ビット	DIV0U/S および DIV1 インストラクションが使用。

SEA および DEA

[Register] リージョンには、SEA (Souce Effective Address) と DEA (Destination Effective Address) の値が表示されます。SEA と DEA は、左側で SEA (読み取り元) と右側でそのアドレス (書き込み先) を示しています。

最近変更された属性を強調表示

最近変更された属性は、赤で表示されます (デフォルト)。

別の色を使用して変更された属性を強調表示するには、次の操作を行います。

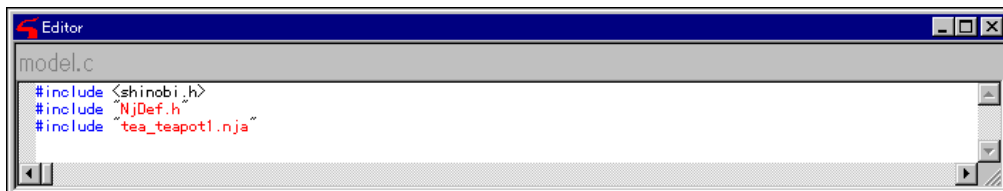
1. [View] メニューをクリックしてから、[Properties] をクリックします。
2. [Mode] を指定します。
3. [Color] タブを選択します。
4. [Attribute] リージョンで、[Highlight data changed] を選択します。
5. [Effects] リージョンで、新規の前景 (Foreground) カラーを選択します。
6. [OK] ボタンをクリックします。

注意

異なる背景 (Background) カラーを設定して、変更された属性を強調表示することはできません。

2.11 [Edit]リージョン

デフォルトのエディタは、CodeScape の[Edit]リージョンです。ここで既存ファイルの編集や新規ファイルの作成を行うことができますが、外部のエディタを使用するように CodeScape を設定することもできます。

**注意**

CodeScape でプロジェクトのコンパイルやビルドに使用する Make オプションを使用する前に、外部エディタで編集したファイルは、すべての変更内容を必ず保存してください。ビルドエラーが発生した場合、[Project Build] ウィンドウにその旨表示されます。その場合は、エントリをクリックすると、エディタが起動され、ソースファイルを開きエラーや警告を含む行を表示します。外部エディタにはこのオプションをサポートしないものもあり、その場合はエラーが起きた行を表示せずファイルを開きます。

[Edit] リージョンでのショートカットメニューの使用

メニュー	内容
New	新規の編集ファイルを作成。
Open...	既存の編集ファイルを開く。
Recent	最近使用した 10 個の編集ファイルをリスト表示する。
Save	現在の編集ファイルを保存。
Save As...	名前を付けて現在のエディタファイルを保存。
Cut	エディタファイルの現在の選択内容を切り取り、クリップボードに貼り付け。
Copy	エディタファイルの現在の選択内容をコピーし、クリップボードに貼り付け。
Paste	クリップボードの内容を、現在のカーソルの位置に挿入。
Tabs...	新規のタブ値を入力。
Undo...	最後の操作を元に戻す (アンドゥー)
Redo...	アンドゥーを取り消す。
Find...	文字列を検索。
Replace...	現在の選択内容を置き換え。
Go To...	オリジンアドレスの行番号を変更。
Book Marks	ブックマークの ON/OFF をトグル切り替え、次または直前のブックマークに移動、すべてのブックマークの削除をする。
Properties	リージョンのフォントとカラーを設定。
Syntax Highlighting	ハイライト表示の ON/OFF を切り替える。コード中のキーワード、引用符、コメント、デフォルトテキスト、背景の色を明確にする。

2.11.1 ファイルのオープンと保存

既存ファイルのオープン

1. 右クリックして、[Open] をクリックします。
ファイルを開くダイアログボックスが表示されます。
2. 必要なファイルを選択し、[開く] をクリックします。

新規ファイルのオープン

右クリックして、[New] をクリックします。

ファイルの保存

右クリックして、[Save] をクリックします。
名前を付けていないファイルに対し、[Save] コマンドを使用すると、ファイル名を付けて保存ダイアログボックスが表示されます。

新しい名前を付けてファイルを保存

1. 右クリックして、[Save As] をクリックします。
ファイル名を付けて保存ダイアログボックスが表示されます。
2. ファイルの新しい名前を入力して、保存をクリックします。

2.11.2 検索と置換

検索

1. 挿入ポイントを、検索を開始するところに動かします。
2. 次のいずれかを行います。
 - ・ [Edit] メニューをクリックして、[Find...] をクリックする。
 - ・ 右クリックして、[Find...] をクリックする。[Find] ダイアログボックスが表示されます。
3. [Find What] テキストボックスに検索する文字列を入力します。
4. [Direction] フィールドで [Up] または [Down] を選択します。
5. 次のオプションのいずれかを選択します。
 - ・ [Match Case] は大文字 / 小文字の判別を有効にします。
 - ・ [Regular expression] は正規検索を有効にします。
 - ・ [Wrap around search] はドキュメントの末尾に到達しても検索を続けます。
6. 次のいずれかを行います。
 - ・ [Find Next] をクリックすると、検索文字列を変更せずに検索を続けます。
 - ・ [Mark All] をクリックすると、検索した文字列のあるすべての行をブックマークに追加します。

検索と置換

1. 挿入ポイントを、置換を始めるところに動かします。
2. 次のいずれかを行います。
 - ・ [Edit] メニューをクリックして、[Replace...] をクリックする。
 - ・ 右クリックして、[Replace...] をクリックする。[Replace] ダイアログボックスが表示されます。
3. [Find What] テキストボックスに検索する文字列を入力します。
4. [Replace With] テキストボックスに新しい文字列を入力します。
5. [Direction] フィールドで [Up] または [Down] を選択します。
6. 次のオプションのいずれかを選択します。
 - ・ [Regular expression] は正規検索を有効にします。
 - ・ [Wrap around search] はドキュメントの末尾に到達しても検索を続けます。
7. 次のいずれかを行います。
 - ・ [Find Next] をクリックすると、置換を行わずに検索を続けます。
 - ・ [Replace] をクリックすると、最初に見つけた文字列を置換します。
 - ・ [Replace All] をクリックすると、すべての単語を置換します。

2.11.3 テキストの切り取りと貼り付け

テキストの移動

1. 移動するテキストを選択して反転表示します。
2. [Edit] をクリックしてから、[Cut] をクリックします ([Ctrl] + [X] キー)。
3. 挿入ポイントを、貼り付ける位置に置きます。
4. [Edit] をクリックしてから、[Paste] をクリックします ([Ctrl] + [V] キー)。

テキストは元の位置から削除され、新しい位置に移されます。

テキストのコピー

1. コピーするテキストを選択して反転表示します。
2. [Edit] をクリックしてから、[Copy] をクリックします ([Ctrl] + [C] キー)。
3. 挿入ポイントを、貼り付ける位置に置きます。
4. [Edit] をクリックしてから、[Paste] をクリックします ([Ctrl] + [V] キー)。

元の位置から情報がコピーされ、新しい位置に表示されます。

2.11.4 ブックマークの使用

使用しているソースファイルの頻繁に使用する行に、ブックマークをセットすることができます。ブックマークはファイルが閉じられたりリロードされると消えます。ブックマークは行だけを記録し、カーソルのある列は記録しません。ブックマークを含む行が削除されると、ブックマークも消えます。

ブックマークのセット

1. ブックマークをセットしたい行に挿入ポインタを移動します。
2. 右クリックして、[Book Marks] を選択します (トグルになっています) 。 端にインジケータが表示されます。

特定の文字列を含むすべての行にブックマークをセット

1. 右クリックして、[Find] をクリックします。
 2. [Find What] テキストボックスに、文字列を入力します。
 3. [Mark All] をクリックします。
- 特定の文字列を含む行の端にインジケータが表示されます。

ブックマーク間の移動

現在の位置より後のブックマークに移動するには

- ・ 右クリックして、[Book Marks] を選択して、[Next] をクリックします。

現在の位置より前のブックマークに移動するには

- ・ 右クリックして、[Book Marks] を選択して、[Previous] をクリックします。

ブックマークの削除

1. 削除したいブックマークのある行に挿入ポインタを移動します。
2. 右クリックして、[Book Marks] を選択します (トグルになっています)。
端に表示されていたインジケータが削除されます。

すべてのブックマークの削除

- ・ 右クリックして、[Book Marks] を選択して、[Delete] をクリックします。

次のコマンドがあります。

- ・ Dev.Box への接続
- ・ プロセッサの更新レートの設定
- ・ プロジェクトへのファイルの追加
- ・ プログラムのリスタート
- ・ バイナリの保存とロード

3.1 Dev.Box への接続

Dev.Box の初期化

CodeScape を実行すると、CodeScape は自動的にすべての検出された Dev.Box に接続します (必要に応じてモニタをロードするように要求します)。

Dev.Box のリセット

Dev.Box をリセットするよう促されることがあります。この場合は、ソフトリセットを行います。これで、Dev.Box の状態は回復され、モニタは再び初期化されます。

注意

Dev.Box のリセット後に、プログラムファイルを再度ロードするように要求されることがあります。

ソフトリセットを行うには、次のいずれかを行います。

- ・ [File] メニューをクリックし、[Reset Target] を選択してから、[Soft Reset] をクリックする。
- ・ [Target] ウィンドウで右クリックし、[Target Reset] を選択してから、[Soft Reset] をクリックする。

ソフトリセットが失敗した場合は、ハードリセットを行うように促されます。これにより、Dev.Box がリセットされモニタが再ロードされます。ハードリセットの後、プロジェクトの再ロードが要求されます。

ハードリセットするには、次のいずれかを行います。


- ・ [File] メニューをクリックし、[Reset Target] を選択してから、[Hard Reset] をクリックする。

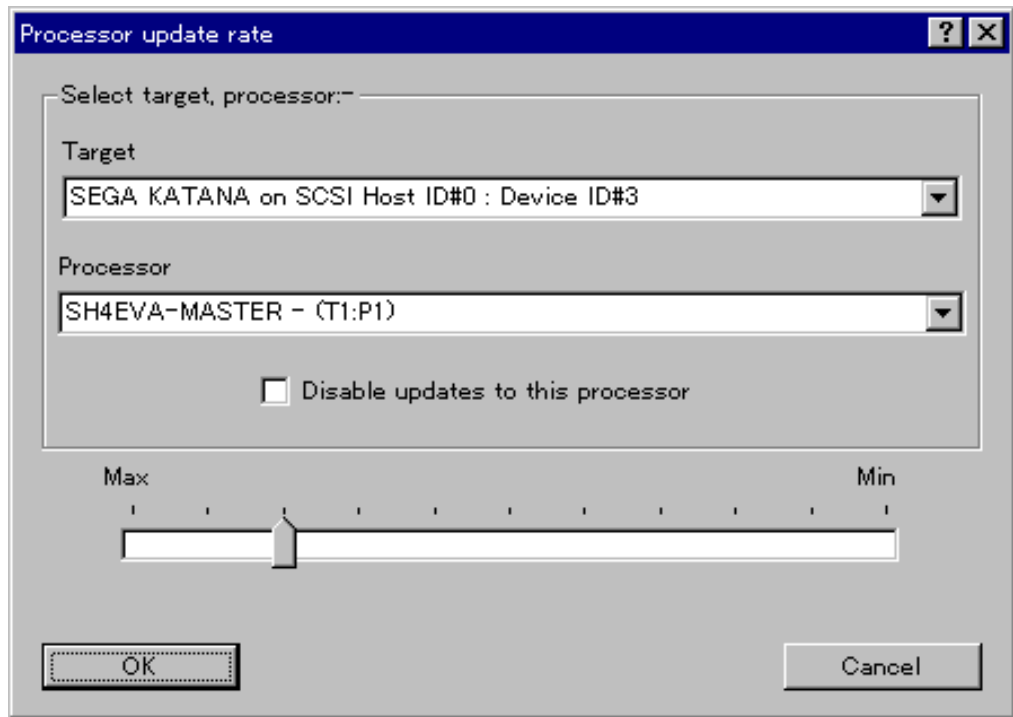
- ・ [Target] ウィンドウで右クリックし、[Target Reset] を選択してから [Hard Reset] をクリックする。

注意 ハードリセットの後、モニタを再ロードするように要求されます。

3.2 プロセッサの更新レートの設定

プロセッサの更新レートを設定すると、CodeScape は Dev.Box の情報を自動的に更新します。レートの変更により Dev.Box の動作が不安定になった場合は、次の手順に従います。

1. [Processor Combo] ツールバーで、 をクリックします。
[Processor Update Rate] ダイアログボックスが表示されます。



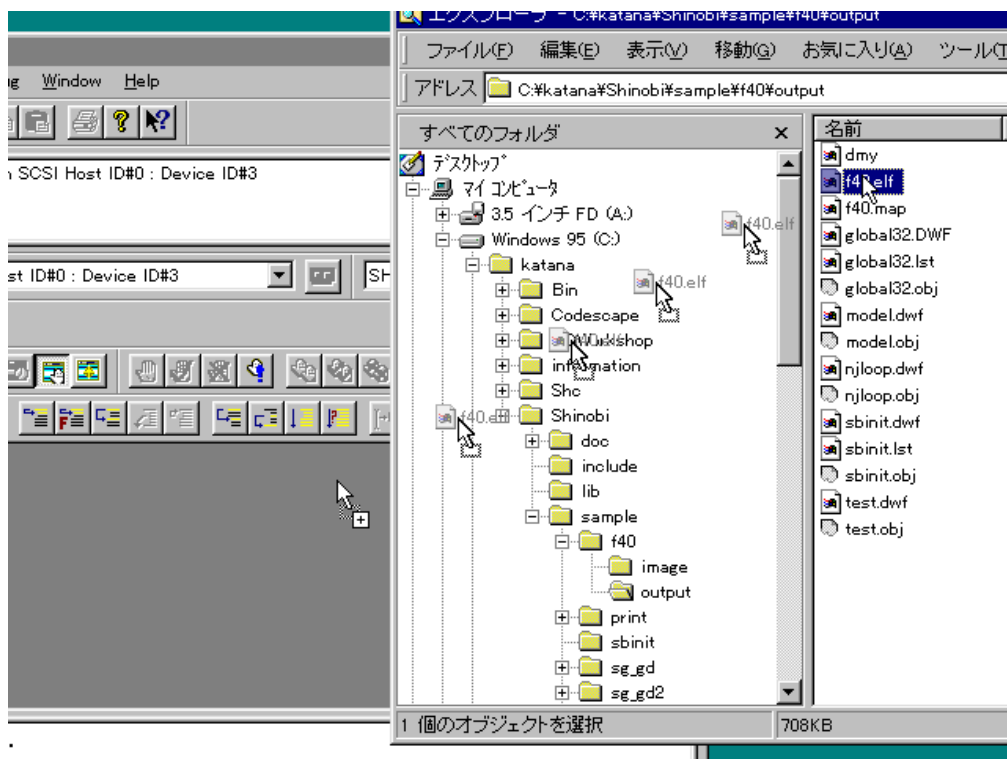
2. 更新レートを設定する [Target] を選択します。
3. プログラムをロードするプロセッサを選択します。
4. 次のいずれかを行います。
 - ・ スライダーを [Min] に設定する。
 - ・ [Disable updates to this processor] を選択して、リージョンのすべての表示を更新しないようにする。
5. [OK] ボタンをクリックします。

3.3 プログラムのロード

プログラムファイルのロード(ドラッグ&ドロップ)

プログラムファイルをロードするには、ELF ファイル(Executable Linkable Format : デバッグ情報付き実行ファイル)をドラッグ&ドロップする方法と、[File]メニューの[Load Program File]をクリックする方法があります。

次の図のようにエクスプローラから CodeScape のウィンドウにドラッグ&ドロップすると、[Load Program File]ダイアログボックスが表示されます。その後の手順は『プログラムファイルのロード(メニューから)』の2番以降と同様です。

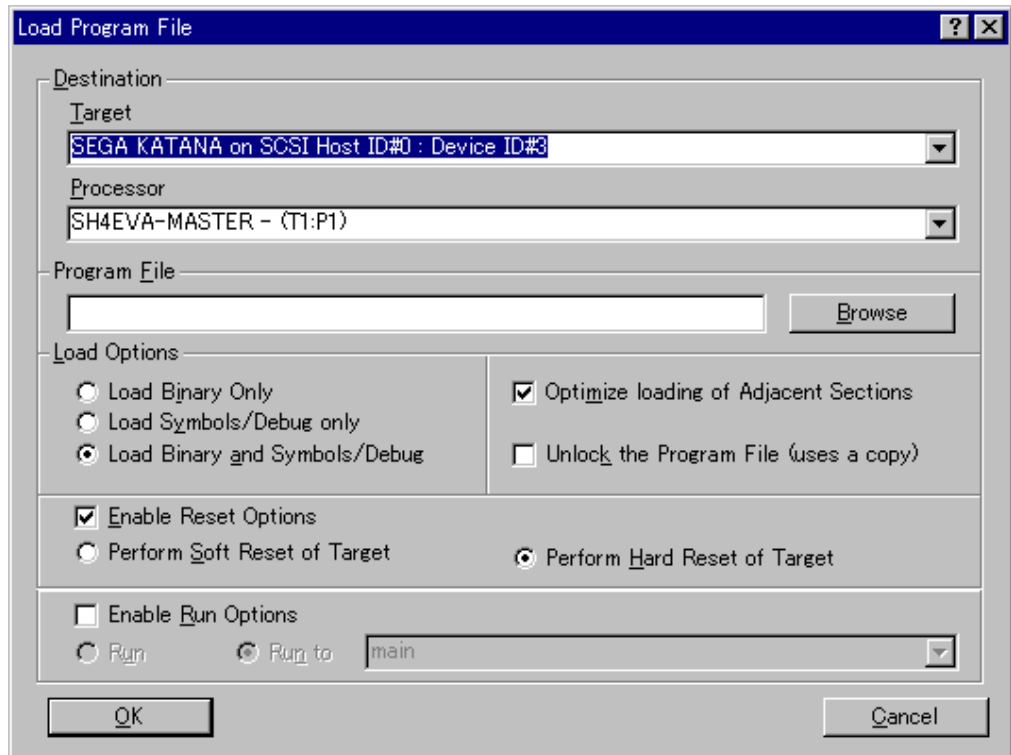


プログラムファイルのロード (メニューから)

1. 次のいずれかを行います。

- ・ [File] メニューをクリックしてから、[Load Program File] をクリックする ([Ctrl] + [Shift] + [C] キー)
- ・ [Target] ウィンドウで、右クリックして、[Load Program File] を選択する。

[Load Program File] ダイアログボックスが表示されます。



2. [Target] リストボックスから、[Target] を選択します。

3. [Processor] リストボックスから、プロセッサを選択します。

4. 次のいずれかを行います。

- ・ [Browse] をクリックし、ファイルを選択する。
- ・ [Program File] テキストボックスに、パスとファイル名を入力する。

5. [Load Options] ラジオボタンで、次のオプションボタンのいずれかを選択します。

[Load Binary Only]

[Load Symbolic/Debug only]

[Load Binary and Symbolic/Debug]

6. Dev.Box にプログラムファイルをリロードするときには使用したいオプションを [Enable Reset Option] から選択します。

7. [Enable Run Options] を選択します。

- ・ [Run] を選択すると、ファイルをロードして実行します。
- ・ [Run to] を選択すると、ファイルをロードして特定のアドレスから実行します。

8. [OK] ボタンをクリックします。

プログラムファイルのバイナリ部分のロード

1. 次のいずれかを行います。
 - ・ [File]メニューをクリックしてから、[Load Program File...]をクリックする([Ctrl] + [Shift] + [C]キー)
 - ・ [Target]ウィンドウで、右クリックして、[Load Program File...]を選択する。
 [Load Program File] ダイアログボックスが表示されます。
2. [Target] リストボックスから、[Target]を選択します。
3. [Processor] テキストボックスから、プロセッサを選択します。
4. [Program File] テキストボックスに、プログラムファイルの位置を入力します。
5. [Load Binary Only] をクリックして、[OK] ボタンをクリックします。


プログラムファイルのシンボリックデバッグ情報のロード

1. 次のいずれかを行います。
 - ・ [File]メニューをクリックしてから、[Load Program File]をクリックする([Ctrl] + [Shift] + [C]キー)
 - ・ [Target]ウィンドウで、右クリックして [Load Program File...]を選択する。
 [Load Program File] ダイアログボックスが表示されます。
2. [Target] リストボックスから、[Target]を選択します。
3. 次のいずれかを行います：
 - ・ [Processor] リストボックスから、[Processor]を選択する。[Program File] テキストボックスに、プログラムファイルの名前とそれへのパスを入力します。
 - ・ [Browse] をクリックし、必要なファイルを見つける。
4. [Load Symbolic/Debug only] をクリックして、[OK] ボタンをクリックします。

3.4 プログラムのリスタート

リスタートにより、現在のプログラムファイルのバイナリ部分がロードされ、PC はエントリー ポイントにリセットされます(分かっている場合)。プログラムファイルの最後の変更時間が違えば、シンボリック情報がロードされます。

現在のプログラムファイルのバイナリ部分のロード

- ・ [Debug]メニューをクリックし、[Execution]を選択してから、[Restart]をクリックする([Ctrl] + [Shift] + [R]キー)
- ・ [Target]ウィンドウで、右クリックし [Execution]を選択してから、[Restart]をクリックする。
- ・ [Debug] ツールバーで  をクリックする。
- ・ 適当なリージョンで右クリックし、[Execution]を選択してから、[Restart]をクリックする。

3.5 バイナリの保存とロード

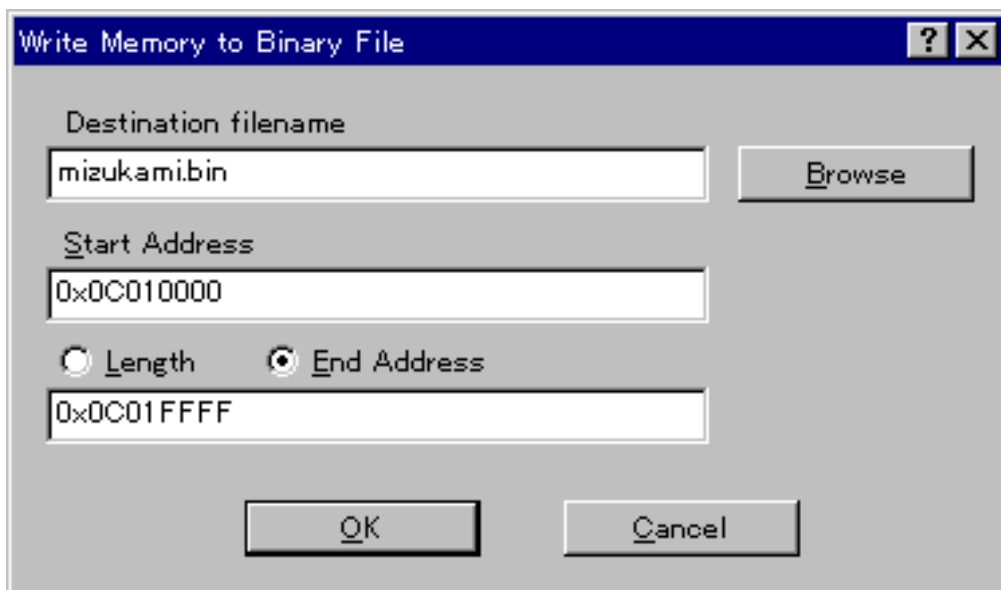
大きなメモリイメージをロード/セーブします。

[Save binary] および [Load binary] を使用すると、メモリイメージをロード/セーブすることができます。この機能は、ビットマップやプロセッサ固有のコードをメモリの選択領域にロードや保存する場合に役立ちます。

1. 次のいずれかを行います。:

- ・ [File] メニューをクリックしてから、[Load Binary] をクリックする。
- ・ [File] メニューをクリックしてから、[Save Binary] をクリックする。

[Write Binary File to Memory] ダイアログボックスが表示されます。



2. ソースファイル名を入力します。

3. 開始アドレスを指定します。

4. 次のいずれかを行います。

- ・ [End Address] を選択してから、下のテキストボックスで終了アドレスを指定する。
- ・ [Length] を選択してから、下のテキストボックスでアドレス長を指定する。

5. [OK] ボタンをクリックします。

注意

無効メモリ領域、読み取り専用メモリ、モニタ用の予約メモリなどの重要なメモリ領域に、バイナリ情報を書き込むことはできません。指定領域内に要注意のメモリ領域がある場合には、そのメモリ領域がスキップされたというメッセージが表示されます。

セッションを使用するためのコマンドは、[File] メニューにあります。
セッションを保存すると、CodeScape は次のデバッグ情報を自動的に保存します。

- ・ ウィンドウとリージョンの設定
- ・ プロジェクトビルド情報
- ・ ソースファイルのパス
- ・ CodeScape のソースファイルのディレクトリ
- ・ 各 Dev.Box のプロセッサに、どのプログラムファイルをロードするかの情報
- ・ 設定されているブレークポイント

注意 次にこのセッションをオープンするとき、CodeScape は以上の情報を自動的にロードします。

[File] メニューのコマンド

新規のセッションをオープンするには、次の操作を行います。

1. [File] メニューをクリックして、[Session New] をクリックする。
[New] ダイアログボックスが表示されます。
2. 拡張子を「.SSN」として、ファイル名を入力する。
3. [OK] ボタンをクリックする。

既存のセッションをオープンするには、次の操作を行います。

1. [File] メニューをクリックして、[Session Open...] をクリックする。
[ファイルを開く] ダイアログボックスが表示されます。
2. [ファイルの場所] テキストボックスでファイルの場所を選択する。
3. [ファイル名] テキストボックスでファイルを選択する。
4. [開く] ボタンをクリックする。

注意 新規のセッションをオープンする場合は、プログラムファイルをロードしなければなりません。

セッションを保存するには、次の操作を行います。

- ・ [File] メニューをクリックし、[Session Save] をクリックする。

新しい名前を付けてセッションを保存するには、次の操作を行います。

1. [File] メニューをクリックし、[Session Save As...] をクリックする。
[ファイル名を付けて保存] ダイアログボックスが表示されます。
2. [保存する場所] テキストボックスに、ファイルの場所を入力する。
3. [ファイル名] テキストボックスに、新規セッションのファイル名を入力する。
4. [保存] ボタンをクリックする。

セッションをクローズするには、次の操作を行います。

- ・ [File] ボタンをクリックして、[Session Close] をクリックする。

注意

CodeScape が新規セッションをオープンする前に、現在のセッション情報を保存するように促されることがあります。

最近使用されたファイルの一覧

[File] メニューには、最近使用されたセッションファイルがリスト表示されます。

最近使用したセッションファイルをオープンするには、次の操作を行います。

1. [File] メニューをクリックする。
2. リストからオープンするセッションをクリックする。

注意

このリストを非表示にしたり、表示するファイル数を変更することはできません。

CodeScape を終了するには、次の操作を行います。

- ・ [File] メニューをクリックして、[Exit] をクリックする。

CodeScape を終了するときに、次のデバッグ情報を保存するかどうか、確認を求められます。

- ・ ウィンドウとリージョンの設定
- ・ プロジェクトビルド情報
- ・ ソースファイルのパス
- ・ CodeScape のソースファイルのディレクトリ
- ・ 各 Dev.Box のプロセッサに、どのプログラムファイルをロードするかという情報
- ・ 設定されているブレークポイント

注意

次にこのセッションをオープンするとき、CodeScape はこの情報をロードします。

[Project]メニューはプロジェクトを設定し、メイクし、ビルドするために使用します。
[Project]メニューは次のことを行います。

- ・プロジェクトビルド環境
- ・エディタ
- ・ソースファイルの場所
- ・CodeScape のソースファイルのディレクトリ
- ・ファイルサーバーのリフレッシュレート

注意

CodeScape はコンパイルされたソースファイルをリンクするためにプロジェクトファイル (GNU C では makefile に相当) を使用します。

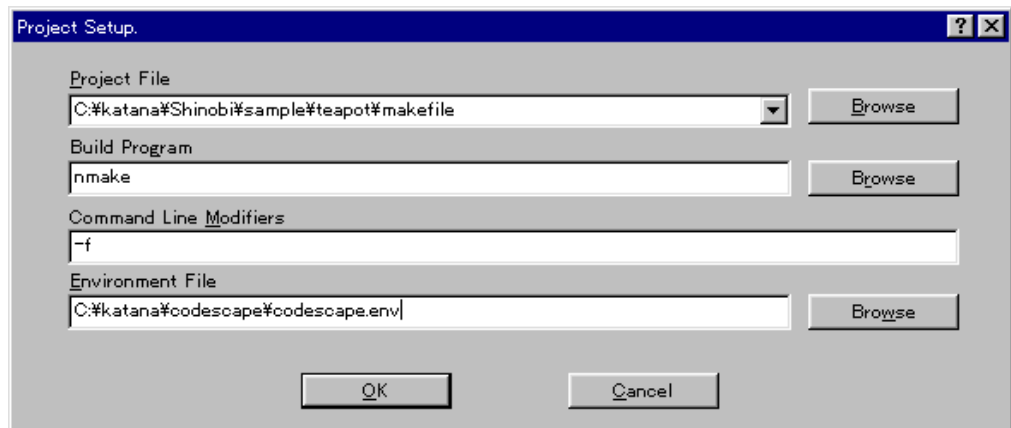
5.1 プロジェクトビルド環境の準備

プロジェクトを設定するには、次の手順に従います。

1. [Project]メニューをクリックし、次に [Setup Project]をクリックします。
2. 次のいずれかを行います。
 - ・プロジェクトの makefile をフルパスで [Project File]テキストボックスに入力する。
 - ・プロジェクトの名前とパスを [Project File]テキストボックスに入力する。
 - ・ [Project File]リストから最近のプロジェクトを選択する。
 - ・ [Browse]をクリックして、プロジェクトを検索する。
3. プログラムビルドファイルの名前とパスを [Build Program]テキストボックスに入力します (たとえば、nmake)。
4. コマンドラインに渡すコマンドラインパラメータがあれば、[Command Line Modifiers]テキストボックスに入力します (nmake なら、- f)。
5. プロジェクト環境ファイルの名前とパスを [Environment File]テキストボックスに入力します。このディレクトリはプログラムビルドファイルと同じディレクトリのこともあります。
6. [OK]ボタンをクリックすると、プロジェクトの準備が完了します。

例：

1. MS-DOS プロンプトを起動し、c : ¥katana¥codescape フォルダに移動します。
C : ¥> cd ¥katana¥codescape
2. ¥katana¥bin¥set_kt.bat を実行します。
C : ¥katana¥codescape > ¥katana¥bin¥set_kt.bat
3. CodeScape のプロジェクト用環境ファイルを作成します。
C : ¥katana¥codescape > set > codescape.env
4. 次の図のようにテキストボックスに入力し、[OK] ボタンをクリックします。



プロジェクトをメイクしビルドするには

現在のプロジェクトをビルドするには、次のいずれかの方法に従います。

- ・ [Project] をクリックし、次に [Make] をクリックする。
- ・ [Project Build] ウィンドウで、右クリックし、[Make] をクリックする。
- ・ [Ctrl] + [M] キーを押す。

プロジェクトのビルドを行うと [Project Build] ウィンドウが表示され、そのビルドに関するステータス情報が表示されます。エラーや警告があれば、それらは [Project Build] ウィンドウに表示されます。ビルドエラーが起こった場合は、そのエントリをダブルクリックすると、エディタが起動され、ソースファイルが開かれ、エラーか警告を含んだ行が表示されます。

注意

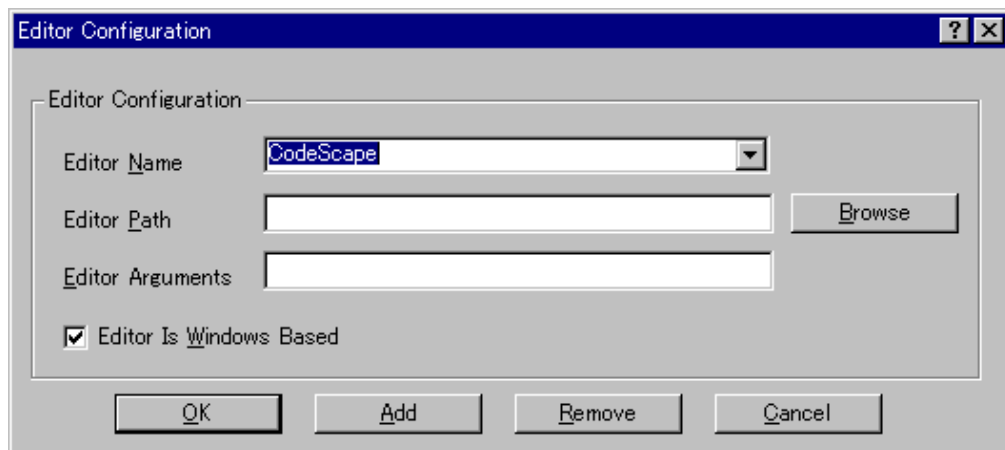
[Target] と [Project Build] ウィンドウは、メインウィンドウの上か下にドッキングさせたり、フロートしたままにしておくことができます。

5.1.1 エディタの設定

デフォルトのエディタは CodeScape の [Edit] リージョンです。ここで既存のファイルを編集したり新しいファイルを作成したりできます。

外部エディタの設定

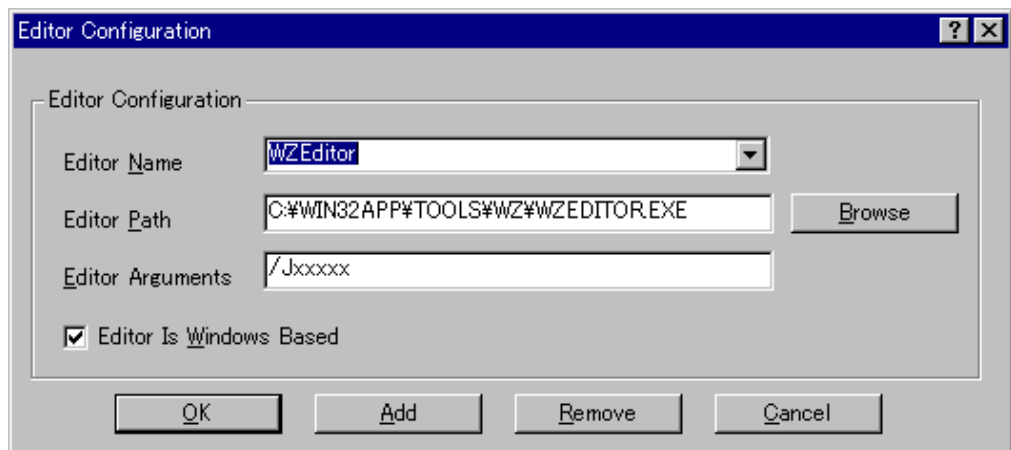
1. [Project] メニューをクリックし、次に [Setup Editor] をクリックします。



2. [Editor Name] テキストボックスにエディタの名前を入力します。
3. [Editor Path] テキストボックスにエディタのパスとコマンドを入力します。
4. [Editor Arguments] テキストボックスに、特定の行にジャンプするためのコマンドラインパラメータを入力します。
5. 次のいずれかを行います。
 - ・ Windows 用のエディタを選択した場合は、[Is Editor Windows Based] チェックボックスを選択し、[OK] ボタンをクリックする。
 - ・ MS-DOS エディタを選択した場合は、[Is Editor Windows Based] チェックボックスをクリアして、[OK] ボタンをクリックする。

例：

次の図は、外部エディタとして WZ Editor を利用した例です。



注意

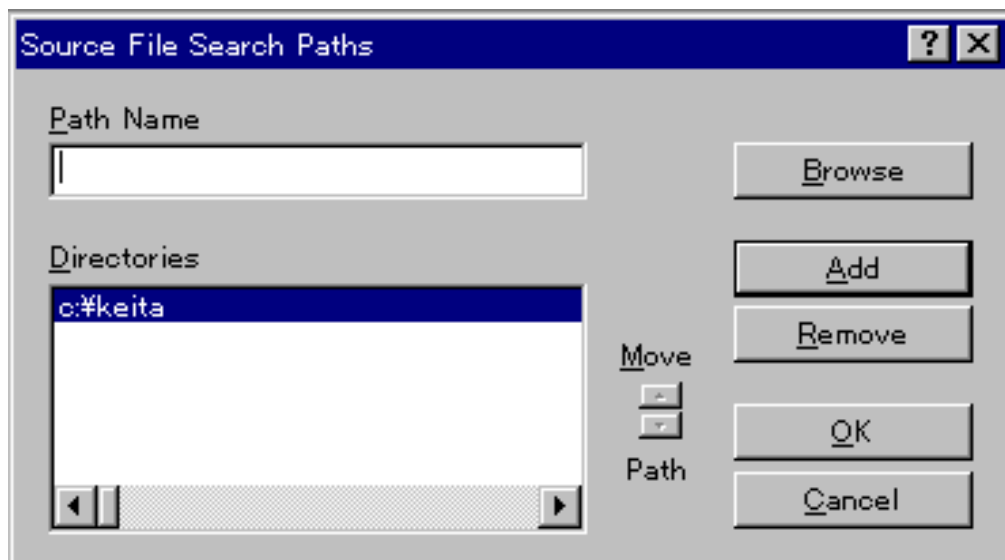
外部エディタを使用して行ったファイルへの変更は、CodeScape で Make オプションを使用してコンパイルし、プロジェクトをビルドする前に必ず保存してください。
ビルドエラーがあった場合は、[Project Build] ウィンドウにそれが表示されます。エントリをクリックするとエディタが起動され、ソースファイルが開かれ、エラーまたは警告が含まれている行を表示します。外部エディタの中には、このオプションがサポートされていないものがあり、その場合はエラーのあった行を表示せずにファイルが開かれます。

プロジェクトコマンドの設定

ソースファイルを検索するパスの設定

[Source File Search Path] ダイアログで、CodeScape がプログラムのプロジェクトファイルを探すディレクトリのパス名を設定、変更、削除することができます。

1. [Project] メニューをクリックし、次に [Edit Source Path...] をクリックします。
[Source File Search Paths] ダイアログボックスが表示されます。



2. [Path Name] に次のいずれかを行います。
 - ・ソースファイルのパスを入力する。
 - ・ [Browse] をクリックして、目的のディレクトリを選択する。
3. [Add] ボタンをクリックします。

[Source File Search Path] からのパスの削除

1. [Project] メニューをクリックし、次に [Edit Source Path...] をクリックします。
[Source File Search Path] ダイアログボックスが表示されます。
2. [Path Name] に次のいずれかを行います。
 - ・ソースファイルのパスを入力する。
 - ・ [Browse] をクリックして目的のディレクトリを選択する。
3. [Remove] ボタンをクリックします。

CodeScape のソースファイルディレクトリの設定

[Set Fileserver] ディレクトリのダイアログボックスを使用して、プログラムファイルのディレクトリの相対パス名を設定したり変更します。これは [Source File Search Paths] に設定したディレクトリと同じでも構いません。

次の手順に従います。

1. [Project] メニューをクリックし、次に [Set FileServer Root Directory] をクリックします。
2. 目的のディレクトリのパスを入力します。
3. [OK] ボタンをクリックします。

注意

使用中のコンピュータのファイルサーバーディレクトリに情報をロード中に、ディスプレイ更新によって Dev.Box がインタラプトされる場合は、[Project] メニューをクリックしてから [Enable Fileserver Optimization] を選択します。
ソフトウェアブレークポイントとウォッチ、プログラム更新レートなどを含め、使用した設定コマンドは、閉じる際にセッションファイルに保存されます。

ファイルサーバーのリフレッシュレートの最適化を有効にする

使用中のコンピュータのファイルサーバーディレクトリに情報をロード中に、ディスプレイ更新によって Dev.Box がインタラプトされる場合は、[Project] メニューをクリックしてから [Enable Fileserver Optimization] を選択します。

これにより、データが Dev.Box から使用中のコンピュータに転送されている間、ディスプレイの更新が中止されます。たとえば大きなビットマップを Dev.Box からファイルサーバーディレクトリにロードする際などに、これは有効です。

注意

このオプションはオン / オフいずれにも切り替えられます。

デバッグ操作には次のものがあります


- ・プログラムのコードをトレースする。
- ・変数や制御構造をチェックする。
- ・ブレークポイントを設定してプログラム実行を制御する。
- ・Dev.Box プロセッサの動作をシミュレートして、プログラム中のアセンブラのクリティカルなループを最適化する。

注意

ツールバーを使用すると、デバッグの主な機能にアクセスできます。ツールバーの表示と非表示を切り替えるには、Configuration チェックボックスを使用します。

6.1 プログラムの実行と停止

プログラムの実行 (Run)


1. [Target] リージョンで、プログラムがロードされたプロセッサを選択します。
2. 次のいずれかを行います。
 - ・ [Debug] メニューをクリックし、[Execution] をポイントして、次に [Run] をクリックする ([F9] キー)。
 - ・ 右クリックし、[Execution] をポイントして、次に [Run] をクリックする。
 - ・ [Debug] ツールバーの  をクリックする。

注意

プログラムは、停止されるか、エラーが起こるまで実行されます。プログラム実行中は [F9] キーを押すと停止します。

すべてのプロセッサを同時に実行 (Run All)

次のいずれかを行います。


- ・ [Debug] をクリックし、[Execution] をポイントし、次に [Run All] ([Ctrl] + [F9] キー) をクリックする。
- ・ 右クリックし、[Execution] をポイントして、次に [Run All] をクリックする。
- ・ [Debug] ツールバーの  をクリックする。

注意

プログラムの実行は、ブレークポイントまたはエラーの発生で停止します。

カーソル位置までプログラム実行 (Run to Cursor)


アクティブな [Source] リージョンまたは [Disassembly] リージョンで、次のいずれかを行います。

- ・ [Debug] メニューをクリックし、[Execution] をポイントし、次に [Run to Cursor] ([Alt] + [F9] キー) をクリックする。
- ・ 右クリックし、[Execution] をポイントし、次に [Run to Cursor] をクリックする。
- ・ [Debug] ツールバーの  をクリックする。

プログラムの実行中に [Run to Cursor] を使用して、いつでもブレークポイントを追加できます。プログラムはブレークポイントまで実行されると停止します。プログラムの実行はカーソル位置かブレークポイントで停止するか、またはエラーが起こると停止します。

指定アドレスを実行するまでプログラムを実行 (Run to Address)

1. 次のいずれかを行います：

- ・ [Debug] メニューをクリックし、[Execution] をポイントし、次に [Run to Address] ([Shift] + [F9]) をクリックする。
- ・ 右クリックし、[Execution] をポイントし、次に [Run to Address] をクリックする。
- ・ [Debug] ツールバーの  をクリックする。

2. [Run to Address/Instructions] ダイアログボックスの [Expression] テキストボックスにアドレスを入力します。


3. [OK] ボタンをクリックします。

アドレスとは、関数名またはアドレスとして解決される式です。

プログラムの実行中 [Run to Address] を使用して、いつでもブレークポイントを追加できます。プログラムはブレークポイントまで実行されると停止します。プログラムの実行は指定アドレスかブレークポイントで停止するか、またはエラーが起こると停止します。

プログラムの停止 (Stop)

プログラムの実行を停止するには、[Target] リージョンでプログラムがロードされたプロセッサを選択し、次のいずれかを行います。


- ・ [Target] ウィンドウ中で右クリックし、[Execution] をポイントしてから右クリックし、そして [Stop] をクリックする。
- ・ [Debug] ツールバー上で  をクリックする。
- ・ [Debug] メニューをクリックし、[Execution] をポイントし、次に [Stop] をクリックする。
- ・ [F9] キーを押す。

注意

プロセッサは直ちに停止します。

すべてのプログラムを同時に停止 (Stop All)

実行中のすべてのプログラムを同時に停止するには、次のいずれかを行います。


- ・ [Target] ウィンドウ中で右クリックし、[Execution] をポイントし、そして [Stop All] をクリックする。
- ・ [Debug] ツールバー上で  をクリックする。
- ・ [Debug] メニューをクリックし、[Execution] をポイントし、次に [Stop All] をクリックする。

注意

すべてのプロセッサが直ちに停止します。

ソースコードを 1 行ずつシングルステップ (Step)

次のいずれかを行います。

- ・ [Debug] メニューをクリックし、次に [Step] ([F7] キー) をクリックする。
- ・ [Debug] ツールバー上で  をクリックする。

アクティブな [Disassembly] リージョン (またはソース以外のリージョン) では、Dev.Box は PC の指す命令を実行します。

アクティブな [Source] リージョンでは、Dev.Box は PC の指す命令を実行します。ソースの命令 1 行から生成された低レベルのアセンブラ命令の実行がすべて終了してから停止します。これには次のものが含まれます。

- ・ ソースの 1 つのマクロ命令に対応するすべての命令。
- ・ 複数のアセンブラ命令を生成するすべての C 命令

後続くソース行(現在のソース行により変わる)は、実行フローにしたがって決定され、別の関数や別のファイル内のこともあります。

注意

シングルステップを実行すると、実行トレースの履歴が生成されます。

トラップ命令はサブルーチン(BSR または JSR)として扱われます。トラップ 32 は CodeScape により予約され、単一命令として扱われます。トラップ 32 のルーチンに Step into するには、[Forced Step Into]を使用します。トラップ 32 に Step into すると、モニターが失敗することがあります。

ステップ実行のアニメーション(Animate Step Run)


各命令の実行ごとにすべてのリージョンを更新するには、[Animate Step Run](デフォルト)を選択します。

- ・ [Debug]メニューをクリックし、[Execution]をポイントし、次に[Animate Step Run]をクリックします。

CodeScape は命令をトレースし、ブレークポイントか start/stop などの別なコマンドが発行されるまで、アクティブなウィンドウに更新された情報を表示します。

特定条件まで実行(Step Run Until...)

1. 次のいずれかを行います。

- ・ [Debug]メニューをクリックし、[Execution]をポイントし、次に[Step Run Until...]をクリックする。
- ・ [Debug]ツールバー上で  をクリックする。

2. 実行を停止する条件式を [Expression Evaluator]に入力します。


CodeScape は命令をトレースし、ブレークポイントか start/stop などの別なコマンドが発行されるまで、アクティブなウィンドウに更新された情報を表示します。

注意

式の評価結果が 0 となった時は、トレースは継続されます。

逆アセンブルレベルでソースコードを 1 行ずつ強制実行(Forced Step Into)

次のいずれかを行います。

- ・ [Debug]メニューをクリックし、[Execution]をポイントし、次に[Forced Step Into]をクリックする([Shift] + [F7]キー)
- ・ [Debug]ツールバー上で  をクリックする。


[Forced Step Into]を使用すると、トラップ 32 に Step into するなど、通常は許されな

い個々のアセンブラ命令を、[Disassembly] リージョンで 1 ステップずつトレースすることができます。

[Source] リージョンでは、Dev.Box は現在のレジスタ値を使用して PC の命令を実行し、次に生成された個々のアセンブラ命令で停止します。個々のアセンブラ命令は、ソース レベルのシングルステップの代わりに、逆アセンブルレベルのシングルステップを使用してトレースされます。

Step into のメカニズム、つまり Trap、Line-A、Line-F、またはサブルーチンの内部に入り、プログラム実行は内部で停止します。1 つのソース命令が多くのアセンブラ命令を生成する場合には、次のソース命令に進む前に、ソース命令上で [Shift] + [F7] キーを数回押す必要があります。

ステップ実行を戻す (Unstep)

- ・ [Debug] メニューをクリックし、次に [Unstep] をクリックする ([Ctrl] + [F7] キー)
- ・ [Debug] ツールバー上で  をクリックする。

CodeScape はトレース実行の履歴を保持しています。トレース履歴は自動的に蓄積され廃棄されます。

Unstep すると、現在のプロセッサとメモリの状況だけが逆にトレースされます。次の内容を Unstep できます。


- ・ 個々の逆アセンブリ命令の一組として実行された命令。
- ・ トレース履歴が残っている限りについて、[Source] と [Disassembly] リージョン中のトレース結果。

注意

一群の命令を Step over した場合は、[Unstep] は使用できません。
コードを Step over すると、それまでのトレース履歴はすべて失われます。

ソースコードを 1 行ずつ Step over (Step Over)

次のいずれかを行います。

- ・ [Debug] メニューをクリックし、次に [Step Over] ([F8] キー) をクリックする。
- ・ [Debug] ツールバー上で  をクリックする。

逆アセンブルされたコードでは、Dev.Box は PC の指す命令を実行して停止します。トラップ、JSR または BSR、は単一の命令として扱われ、プログラム実行はルーチンが終了したときにメモリ内の次の命令で停止します。

ソースコードでは、Dev.Box は PC の指す命令を実行し、ソースファイルの参照が変わった時に停止します。関数コールを Step over する場合は、その関数全体が実行されます。次のソース行で実行は停止します。

注意


条件ブランチは Step over できません。
Step over 実行時にはトレース履歴が生成されます。

コードのステップインとステップアウト (Step Run In、 Step Run Out)

連続的にネストした各関数コールの開始点まで実行しそこで停止するには、[Step Run In] を使用します。連続的にネストした各関数コールの終了時点まで実行し、そこで停止するには [Step Run Out] を使用します。

コードのステップ・ラン・イン (Step Run In)


次のいずれかを行います。

- ・ [Debug] メニューをクリックし、[Execution] をポイントし、次に [Step Run In] をクリックする ([Shift] + [F7] キー)。
- ・ リージョンの中を右クリックし、[Execution] をポイントし、次に [Step Run In] をクリックする。
- ・ [Debug] ツールバー上で  をクリックする。

CodeScape は次の関数の開始点まで実行し、そこで停止します。

コードのステップ・ラン・アウト (Step Run Out)


次のいずれかを行います。

- ・ [Debug] メニューをクリックし、[Execution] をポイントし、次に [Step Run Out] をクリックする ([Shift] + [F8] キー)。
- ・ リージョンの中をクリックし、[Execution] をポイントし、次に [Step Run Out] をクリックする。
- ・ [Debug] ツールバー上で  をクリックする。

CodeScape は現在の関数の終わりまで実行し、そこで停止します。

実行中のプログラムの停止 (Stop)

次のいずれかを行います。

- ・ [Target] ウィンドウで右クリックし、[Execution] をポイントし、[Stop] をクリックする。
- ・ [Debug] ツールバー上で  をクリックする。
- ・ リージョンの中をクリックし、[Execution] をポイントし、次に [Run to Address] をクリックする。
- ・ リージョンの中をクリックし、[Execution] をポイントし、次に [Run to Cursor] をクリックする。

注意 何らかのトレース操作を行うと、直ちにプログラム実行が停止します。

6.2 ブレークポイント


CodeScape には、メモリ範囲へのデータアクセスや外部周辺機器へのアクセスによるブレークなど、ソフトウェアとハードウェアの強力なデバッグ機能があります。

注意 すべてのブレークポイント操作は、[Configure breakpoint (s)] ダイアログボックスを使用していつでも行えます。

ブレークポイントの追加

ブレークポイントを [Source] [Disassembly] [Memory] [Watch] の各リージョンに追加できます。プログラムが実行中でも、いつでもブレークポイントを追加できます。プログラムがブレークポイントに達すると停止します。ブレークポイントの追加はメニュー、ショートカットメニューまたはツールバーを使用して行います。


コードブレークポイントは次の手順で追加します。



1. 右クリックし、[Goto Address] をクリックします。
2. ブレークポイント ツールバーで、 をクリックしてブレークポイントをセットします。
3. 右クリックし、[Execution] をポイントし、[Run] をクリックします。プログラムはブレークポイントに達するまで実行されます。

現在のカーソル位置にブレークポイントをセット

[Source] または [Disassembly] リージョンで、コードブレークポイントを追加することができます。[Watch] または [Memory] リージョンで、データブレークポイントを追加することができます。

どのリージョン中でも目的の位置にカーソルを置いて、次のようにします：

- ・ [Debug] メニューをクリックし、ブレークポイントをポイントし、次に [Toggle Breakpoint] ([F5] キー) をクリックする。
- ・ 右クリックし、ブレークポイントをポイントし、次に [Toggle Breakpoint] をクリックする。
- ・ ブレークポイントツールバーで  をクリックする。

ブレークポイントがセットされ [Source] または [Disassembly] リージョンで有効にされると、ブレークポイントセットアイコン  が最初のカラムに表示されます。ブレークポイントが無効にされると、ブレークポイント無効アイコン  が最初のカラムに表示されます。

ウォッチ変数が[Watch]リージョンに表示されると、ウォッチ変数アイコンが表示されます。[Memory]リージョンでは、指定されたアドレスの背景色が変化します。

ブレークポイントはデフォルトで、次の動作によりセットされます。

- ・ウォッチブレークポイントは、ハードウェアに対するデータの読み出し、書き込みアクセスによってトリガされます。デフォルトでは、ブレークポイントがトリガされ、すべての条件が満たされるとメッセージが表示されます。

すべてのブレークポイント位置の設定は、それが設定される前にチェックされます。問題が発見されるとメッセージが表示され、ブレークポイントの再設定を求められます。


注意

ブレークポイントのデフォルトの動作を変更するには、「[ブレークポイントの設定](#)」を参照してください。

ブレークポイントは、コードを生成する行にのみ追加できます。(これは[Source]リージョンまたは[Watch]リージョンでは第一カラム、また[Disassembly]リージョンでは任意の位置の「。」で示されます)

ブレークポイントの削除


どのリージョンでも、目的のブレークポイントにカーソルを置き、次のいずれかを行います。

- ・[Debug]メニューをクリックし、[Breakpoint]をポイントし、次に[Toggle Breakpoint]をクリックする([F5]キー)。
- ・右クリックし、[Breakpoint]をポイントし、次に[Toggle Breakpoint]をクリックする。
- ・ブレークポイントツールバーで  をクリックする。
- ・[Configure breakpoint (s)]ダイアログボックス([Ctrl] + [F5]キー)で、無効にしたいブレークポイントを選択して、次に[Remove]をクリックする。

ブレークポイントセットアイコン  がコードウインドウに表示されなくなります。

すべてのブレークポイントの削除


どのリージョンでも、目的のブレークポイントにカーソルを置いて、次のいずれかを行います。



- ・[Debug]メニューをクリックし、[Breakpoints]をポイントし、次に[Remove all Breakpoints]をクリックする([Shift] + [F5]キー)。
- ・[Breakpoint]ツールバーで  をクリックする。
- ・右クリックし、[Breakpoints]をポイントし、次に[Remove all Breakpoints]をクリックする。
- ・[Configure breakpoint (s)]ダイアログボックスで、[Remove All]をクリックする。

ブレークポイントの有効化と無効化

無効ブレークポイントの有効化


現在無効なブレークポイントは、どのリージョンでも、目的のブレークポイントにカーソルを置いて、次にいずれかの操作を行うことで再度有効になります。

- ・ [Debug]メニューをクリックし、[Breakpoints]をポイントし、次に[Enable Breakpoint]をクリックする。
- ・ 右クリックし、[Breakpoints]をポイントし、次に[Enable Breakpoint]をクリックする。
- ・ [Breakpoint] ツールバーで  をクリックする。
- ・ [Configure breakpoint (s)] ダイアログボックス ([Ctrl] + [F5] キー) で、[Code Settings] をクリックし、[Breakpoint Enabled] を選択する。

ブレークポイント設定アイコンが  から  に変化して、ブレークポイントが有効になったことを示します。

有効ブレークポイントの無効化


どのリージョンでも、目的のブレークポイントにカーソルを置いて、次のいずれかを行います。

- ・ [Debug]メニューをクリックし、[Breakpoints]をポイントし、次に[Disable Breakpoint]をクリックする。
- ・ 右クリックし、[Breakpoints]をポイントし、次に[Disable Breakpoint]をクリックする。ブレークポイントツールバーで  をクリックする。
- ・ [Configure breakpoint (s)] ダイアログボックス ([Ctrl] + [F5] キー) で、[Code Settings] をクリックし、[Breakpoint is Enabled] をクリアする。

ブレークポイントセットアイコンは、  から  に変わってブレークポイントが無効なことを示します。


すべてのブレークポイントの有効化

次のいずれかを行います。

- ・ [Debug]メニューをクリックし、[Breakpoints]をポイントし、次に[Enable all Breakpoints]をクリックする ([Ctrl] + [Shift] + [F5] キー)
- ・ 右クリックし、[Breakpoints]をポイントし、次に[Enable all Breakpoints]をクリックする。
- ・ [Breakpoint] ツールバーで  をクリックする。

すべてのブレークポイントの無効化


次のいずれかを行います。

- ・ [Debug]メニューをクリックし、[Breakpoints]をポイントし、次に[Disable all Break... points]をクリックする([Ctrl] + [Alt] + [F5]キー)
- ・ 右クリックし、[Breakpoints]をポイントし、次に[Disable all Breakpoints]をクリックする。
- ・ [Breakpoint] ツールバーで  をクリックする。

ブレークポイントのリセット

すべてのブレークポイントのリセット

次のいずれかを行います。

- ・ [Debug]メニューをクリックし、[Breakpoints]をポイントし、次に[Reset all Break... points]をクリックする([Alt] + [F5]キー)
- ・ 右クリックし、[Breakpoints]をポイントし、次に[Reset all Breakpoints]をクリックする([Alt] + [F5]キー)
- ・ [Breakpoint] ツールバーで  をクリックする。

注意

すべてのブレークポイントをリセットすると、現在のカウントを含めてすべての条件値が初期条件にセットされます。

ブレークポイントのトリガカウントをリセット

- ・ [Configure Breakpoint (s)] ダイアログボックスで [Breakpoint] を選択し、[Reset] をクリックします。


ブレークポイントのカウントの現在値のみをリセット

- ・ [Configure Breakpoint (s)] ダイアログボックスで、[Breakpoint] を選択し [General Conditions] をクリックし [Reset Current] をクリックします。

6.3 ブレークポイントの設定

CodeScape では、メモリ範囲へのデータアクセスや外部周辺機器でのブレークポイントなどを含め、ブレークポイントの設定が可能です。

ブレークポイントを設定するには次の操作を行います。

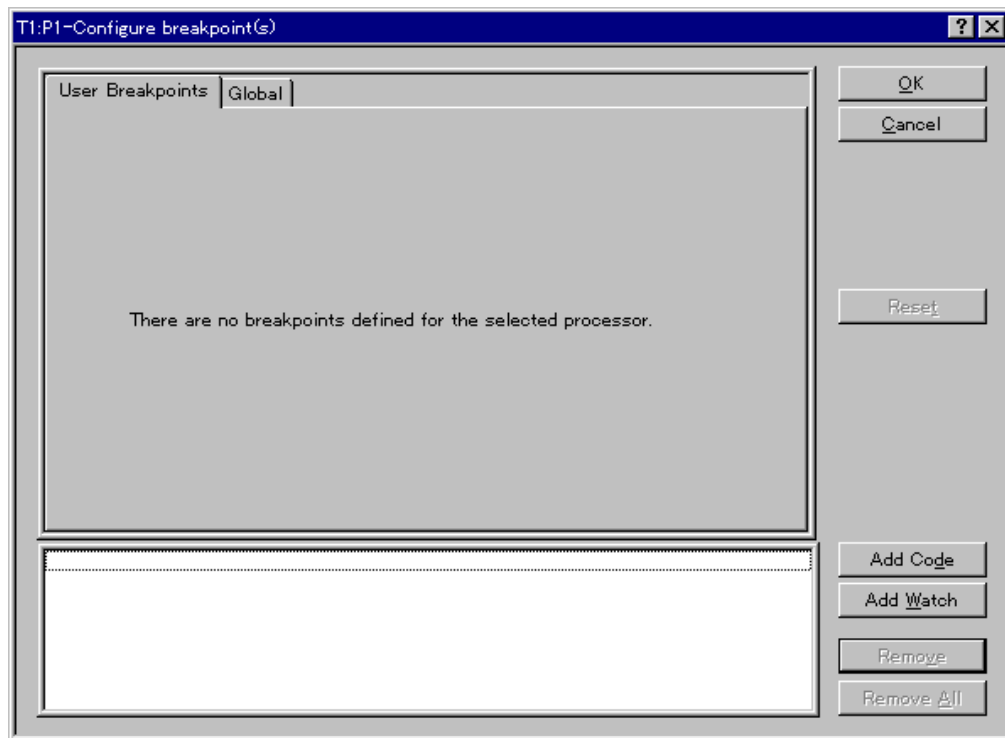
- ・ [Debug] メニューをクリックし、[Breakpoints] をポイントし、次に [Configure Breakpoint (s) ...] をクリックする ([Ctrl] + [F5] キー)
- ・ 右クリックし、[Breakpoints] をポイントし、次に [Configure Breakpoint (s) ...] をクリックする。
- ・ [Breakpoint] ツールバーで  をクリックする。

注意

ブレークポイントを追加して、それを [Configure Breakpoint (s)] ダイアログボックスを使用して、手操作で設定することもできます。

ウォッチブレークポイントは、データアクセスでトリガされ、コードブレークポイントは、命令サイクルのフェッチ - 実行フェーズでトリガされます。

[Configure Breakpoint (s)] ダイアログボックス

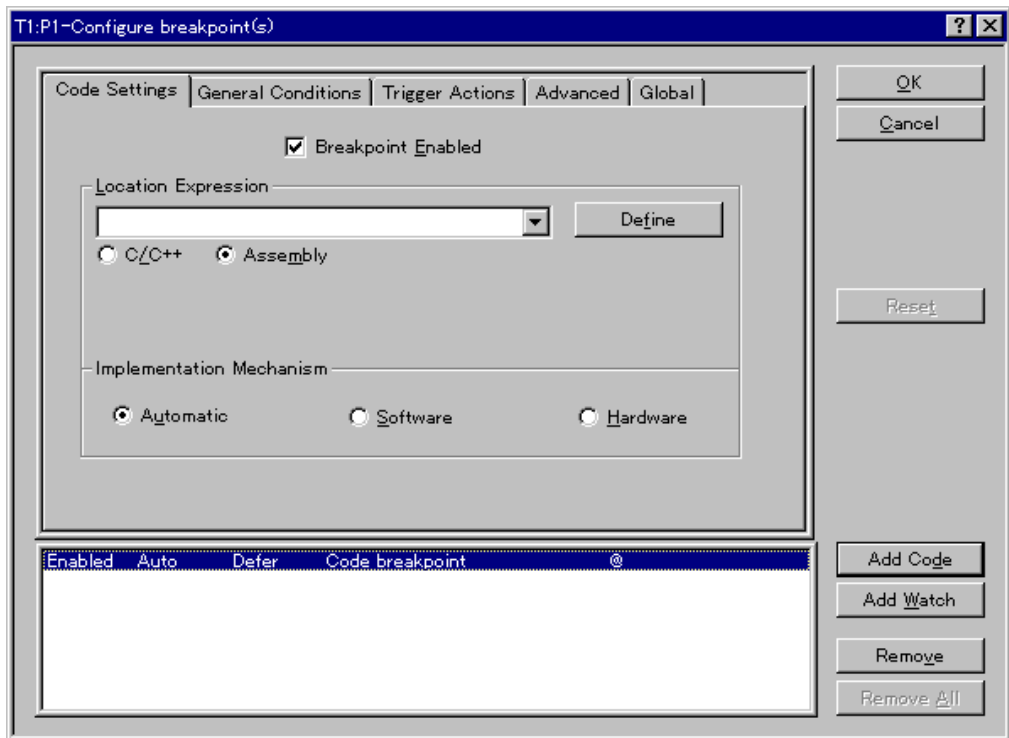


[Configure Breakpoint (s)] ダイアログボックスでは次のことができます。

- ・コードブレークポイントとウォッチブレークポイントの追加、削除、設定
- ・ブレークポイントの有効化と無効化、ロケーションと使用するリソースのセット
- ・ブレークポイントが発生する時点の指定
- ・ブレークポイントが発生したときのプロンプトの設定

[Code Settings] タブの使用

コードブレークポイントは、命令の実行によってトリガされます。コードブレークポイントがトリガされる時、PC はパイプライン中の命令と同一の命令を指します。設定するコードブレークポイントを追加または設定すると、[Code Settings] タブが使用可能になります。



1. 次のいずれかを行います。

- ・リストから設定するコードブレークポイントを選択する。
- ・コードブレークポイントを追加するコードをクリックして設定する。

2. [Breakpoint Enabled] (デフォルト) を選択して、ブレークポイントを有効にします。

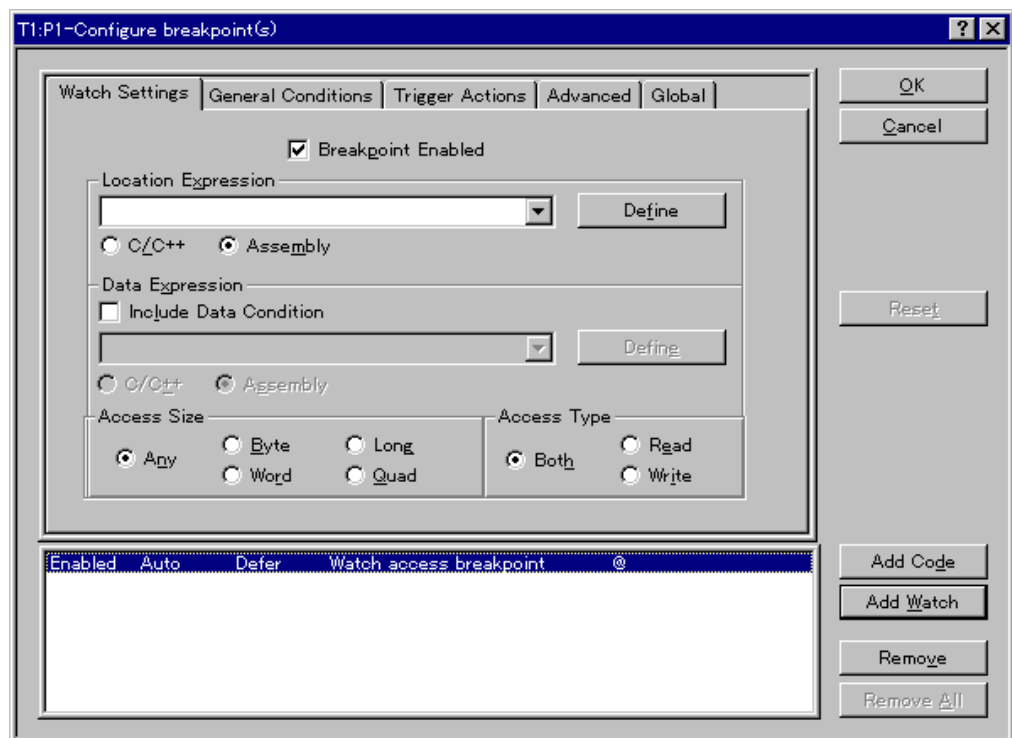
無効化されたブレークポイントを再設定するようにプロンプトがでることがあります。これはコード実行中、セッションの回復中、またはこのダイアログボックス中でコマンドを設定中に属性の有効性が確認できないときに起こります。無効化されたブレークポイントはコード実行に影響を与えません。またハードウェア リソースを消耗することもあります。

3. 実行を中止するコードのメモリ中の位置を指定します。[Location Expression] テキストボックスで、次のいずれかを行います。
 - ・ 式を入力する。
 - ・ [Define] をクリックする。[Breakpoint location expression] ダイアログボックスが表示されます。ここでは式を評価してアドレスをセットします。
4. 次に次のいずれかを行います。
 - ・ C/C++ の式構文を使用するには、[C/C++] を選択する。
 - ・ SHx アセンブラ言語構文を使用するには、[Assembly] を選択する。
5. [Implementation mechanism] グループボックスで、次のいずれかを行います。
 - ・ [Automatic] を選択すると CodeScape がブレークポイントのリソースを管理します。デフォルトでは、ブレークポイントはソフトウェアでインプリメントされます。これが不可能な場合は、ハードウェアのリソースが使用されます。
 - ・ ソフトウェア ブレークポイントを指定するには、[Software] を選択する。
 - ・ Dev.Box のプロセッサに固有のブレークポイントをセットするには、[Hardware] を選択する。

[Watch Settings] タブの使用

ウォッチ (データ) ブレークポイントは、メモリのデータアクセスによりトリガされます。ウォッチブレークポイントがトリガされると、PC はパイプライン中で、ブレークポイントの数命令先に進んでいます。

設定するウォッチブレークポイントを選択すると、[Watch Settings] タブが使用可能となります。

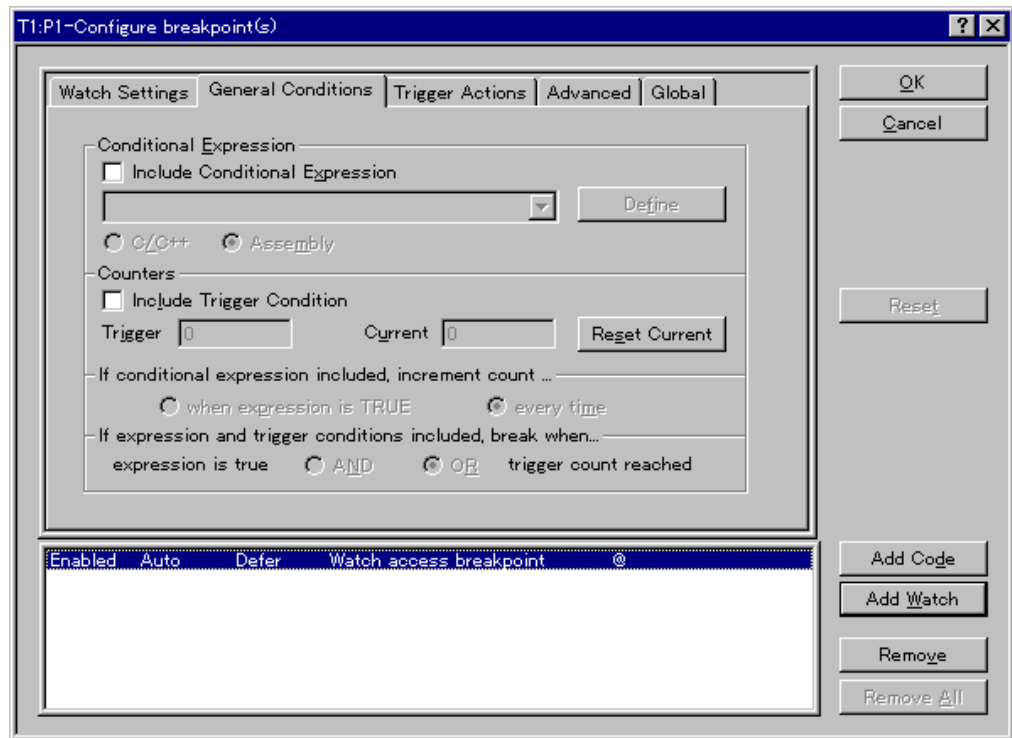


1. 次のいずれかを行います。
 - ・ リストから設定するウォッチブレークポイントを選択する。
 - ・ 設定するウォッチブレークポイントを追加するには Watch をクリックする。
2. [Breakpoint Enabled](デフォルト) を選択してブレークポイントを有効にします。

無効化されたブレークポイントを再設定するようにプロンプトがでることがあります。これはコード実行中、セッションの回復中、またはこのダイアログボックス中でコマンドを設定中に属性の有効性が確認できないときに起こります。無効化されたブレークポイントはコード実行に影響を与えません。またハードウェア リソースを消耗することもあります。
3. ブレークポイントがアクセスされるメモリ中の位置を指定します。[Location Expression] テキストボックスに、次のいずれかを入力します。
 - ・ 式を入力する。
 - ・ [Define] をクリックする。[Breakpoint location expression] ダイアログボックスが表示されます。ここでは式を評価してアドレスをセットします。
4. ウォッチアクセスブレークポイントを、UBC (User Break Controller) の機能を使用するウォッチデータブレークポイントに変更するには、[Include Data Condition] を選択します。必要な [Data Expression] を入力し、[Define] をクリックします。[Breakpoint watch data Expression] ダイアログボックスが表示されます。式を評価して、ロケーションアドレスを設定します。
5. 次のいずれかを行います。
 - ・ C/C ++ の式構文を使用するには、[C/C ++] を選択する。
 - ・ アセンブラ式構文を使用するには、[Assembly] を選択する。
6. [Implementation Mechanism] に、必要なインプリメンテーション メカニズム (デフォルトは any) を入力します。このオプションは SH4 プロセッサにのみ有効です。
7. [Access Size] に必要なアクセスサイズ (デフォルトは [Any]) を入力します。ウォッチブレークポイントを追加するときに [Toggle] を使用し、そのサイズが分かっている場合は、[Any] の代わりにそれを使用します。
8. [Access Type] で、必要なアクセスタイプを選択します。デフォルトは [Both] で読み出し、書き込みの両アクセスです。

[General Conditions] タブの使用

[General Conditions] タブを使用して、ブレークポイントがトリガされる条件を定義します。ブレークポイントに条件を付け、メモリアクセスのタイプとデータの値、またトリガカウントを指定することができます。



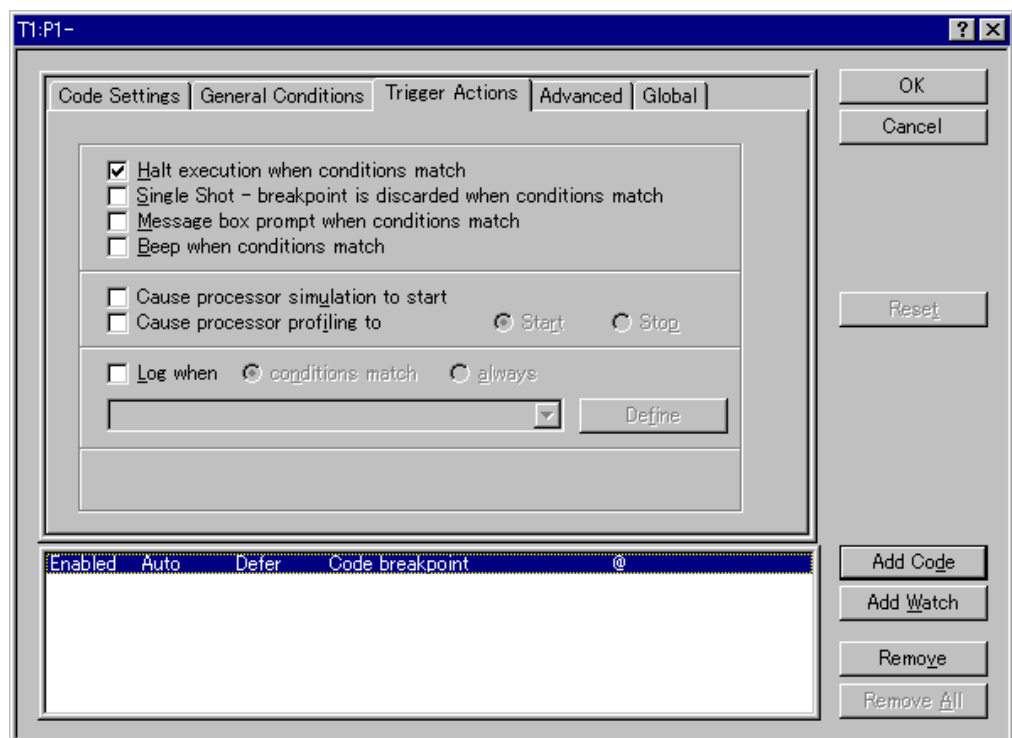
1. 条件式を使用するには、[Include Conditional Expression] を選択します。
2. 次のいずれかを行います。
 - ・ [Include Conditional Expression] テキストボックスに式を入力する。
 - ・ [Define] をクリックして [Breakpoint condition expression] ダイアログボックスを開き、式を定義する。
3. 次のいずれかを行います。
 - ・ C/C++ の式構文を使用するには、[C/C++] を選択する。
 - ・ トリガカウント (SHx アセンブラ言語構文) を使用するには、[Assembly] を選択する。

式は論理式として評価され、値 0 は偽 (false) を、0 以外の値は真 (true) を表します。

4. トリガ条件を含めるには、[Include Trigger Condition] 選択します。現行カウントが指定された [Trigger Count] 値に達した時に条件が真になります。
5. [Trigger Count Condition] が真となるために必要な現行カウントの値を入力します。
6. [Counters] で、[Current] ボックスの値がセットした [Trigger] ボックスの値と一致することを確認します。[Reset Current] をクリックすると、現行カウントがリセットされます。
7. カウンタをインクリメントする条件を選択します。デフォルトではブレークポイントが発生するか評価された時点に毎回現行カウントがインクリメントされます。
8. 式とカウント条件の両方が含まれている場合は、ブレークする条件を選択します。デフォルトは [OR] です。

[Trigger Actions] タブの使用

ブレークポイントがトリガされたときに CodeScape の動作を指定するには、[Trigger Actions] タブのチェックボックスを使用します。



- [Halt execution when conditions match]
ブレークポイントの条件を満足したときに、プログラムの実行を停止します。他の指定されたすべての動作が実行された後にプログラムの実行を再開するには、このチェックボックスをクリアします。
- [Single Shot - breakpoint is discarded when conditions match]
ブレークポイントがすべてトリガされ、条件を満足した後に廃棄します。
- [Message box prompt when conditions match]
ブレークポイントがトリガされ、すべての条件を満足したときに、メッセージを表示します。
- [Beep when conditions match]
ブレークポイントがトリガされ、すべての条件を満足したときに、PC のビーブ音が鳴ります。
- [Cause processor profiling to]
ブレークポイントがトリガされたときに、シミュレータの開始または停止を指定します。
- [Log when]
ブレークポイントがトリガされたとき、または毎回、ログを生成します。有効なログ式を入力します。

注意

Dev.Box のプロセッサ用の [Log] リージョンがない場合は、CodeScape が作成します。

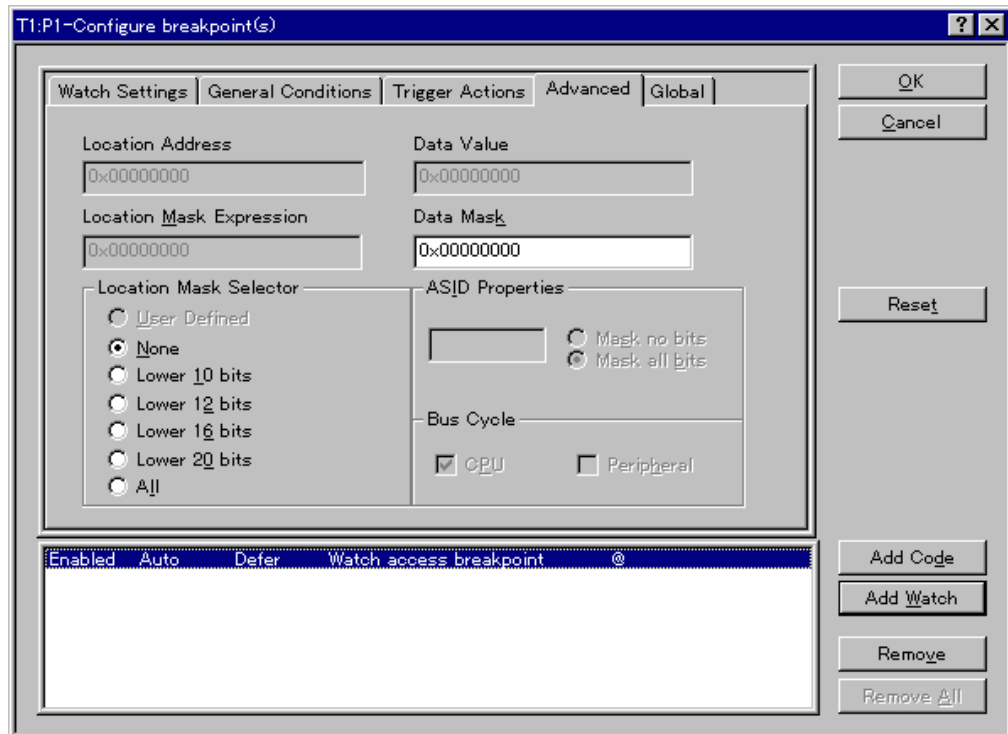
[Advanced] タブを使用してコードブレイクポイントのオプションを指定

注意

[Location Address] テキストボックスは読み出し専用です。ロケーションをセットするには [Code Settings] タブをクリックします。

[ASID] プロパティは、現在はデフォルトの設定になっていて変更できません。将来のリリースで有効になる予定です。

[Advanced] タブには、ハードウェア インプリメンテーションメカニズムを使用するコマンドがあります。これらのコマンドはウォッチブレイクポイントとコードブレイクポイントのみに適用されます。



1. マスクして無視する [Location Address] のビットを指定するには [Location Mask] を選択します。[Location Address] の対応するビットを無視するには [Location Mask] を 1 にセットし、そうでなければ 0 にします。
2. [Break Mode] テキストボックスで次のいずれかを選択します。
 - [Before Execution.]
 - [After Execution.]

注意

[Location Address] テキストボックスは読み出し専用です。ロケーションをセットするには [Code Settings] タブをクリックします。

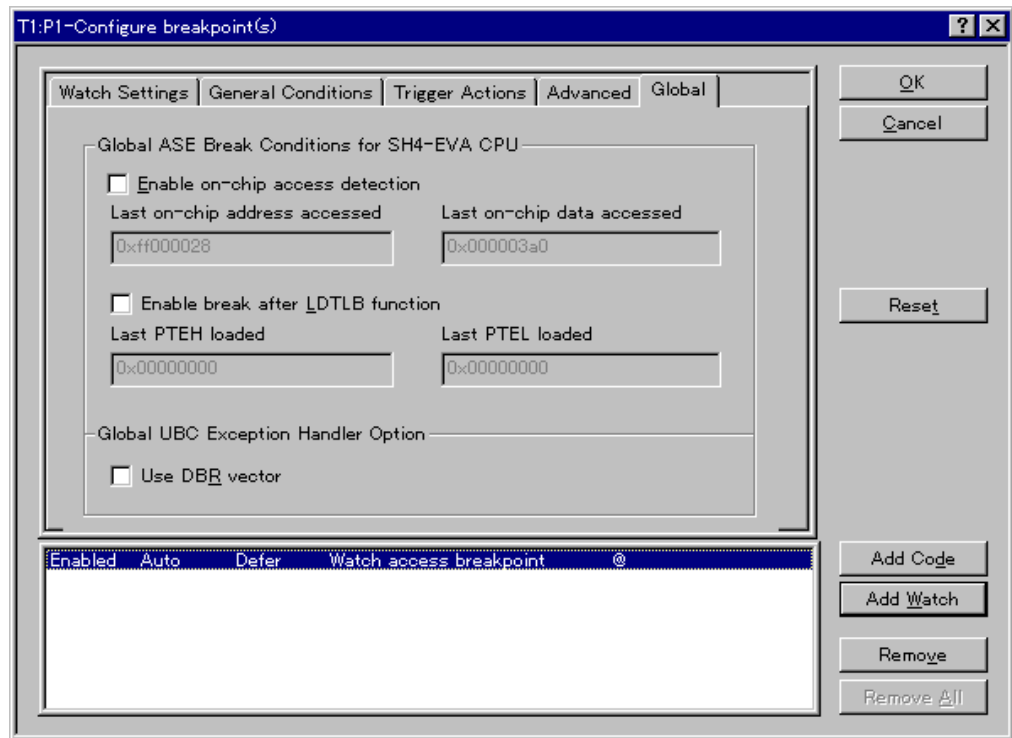
[ASID] プロパティは、現在はそのデフォルトの設定になっていて変更できません。将来のリリースで有効になる予定です。

[Advanced] タブには、ハードウェアインプリメンテーションメカニズムを使用するコマンドがあります。これらのコマンドはウォッチブレークポイントとコードブレークポイントのみに適用されます。

1. マスクして無視する [Location Address] のビットを指定するには [Location Mask] を選択します。[Location Address] の対応するビットを無視するには [Location Mask] を 1 にセットし、そうでなければ 0 にセットします。
2. [Data Mask] テキストボックスで、対応する [Data Address] のビットを無視するには [Data Mask] のビットを 1 にセットし、そうでなければ 0 にセットします。
3. [Bus cycle] フィールドで、含めるバスサイクルを [CPU] [Peripheral (DMA)] または両方の中から選択します。

[Global] タブを使用して Hitachi SH4-EVA プロセッサ用のデバッグ環境を指定

[Global] タブには、Dev.Box のプロセッサのデバッグ環境をセットするコマンドがあります。[Global] タブは、[SH4-EVA] プロセッサに接続したときに表示され、必要なオプションを指定することができます。



1. [Global ASE Break Conditions for SH4-EVA CPU] フィールド。

- ・ [Enable on-chip access detection] を選択すると、CodeScape はオンチップの I/O 例外を生成します。

表示される値は、例外が発生したときに最後にアクセスされたオンチップアドレス、そして最後にアクセスされたオンチップ データです。

- ・ [Enable break after LDTLB function] を選択すると、CodeScape は LDTLB 命令ブレイクを生成します。

表示される値は、最後にロードされた PTEH、そして最後に MMU にロードされた PTEL です。

2. [Global UBC Exception Handler Option] フィールドで、[Use DBR vector] (デフォルト) を選択します。

CodeScape は UBC の debug stub default 例外ハンドラを使用します。これにより、UBC ブレイクポイントの動作を変えずに、自分のプログラム中に例外処理ルーチンを定義し、VBR を変更することが可能になります。

6.3.1 ブレークポイント式のフォーマット

CodeScape には、式を [Log] ウィンドウ中表示する方法を制御できる強力な式フォーマット機能があります。フォーマット定義は C の `printf` 関数と同様に使用できるフォーマット式によって管理されます。式はフォーマット文字列に続く、任意数のカンマ区切りの式によって構成されます。式は 0 から番号づけられ、レジスタ名やメモリロケーションを参照する任意のスロット式です。フォーマット式の構文は次の通りです。

```
[ "FormattingString " | FormattingString ] [ Expression ] ...
```

フォーマット指定 (Specification)

指定 (Specification) は「 % 」記号で開始し、その後に 1 つ以上のコマンド修飾子 (modifier) を続けます。指定はフォーマット指示子 (Specifier) で終わります。フォーマット文字列は 1 つ以上のフォーマット指定からなります。複数の指定を区切るには次のようにします。

- ・スペースを使用する。
- ・指定を引用符で囲み、その各々をカンマで区切る。

フォーマット指定の構文は次の通りです。

```
% [ Pointer ] [ Width ] [ Repeat ] Specifier
```

% はフォーマット指定の開始を意味します。指示子 (Specifier) は、式の表示に影響を与えるオプションの修飾子 (modifier) を制御します。

指示子 (Specifier) によって制御されるコマンド修飾子 (modifier)

式項目	コマンド修飾子の意味
Pointer	パラメータポインタを変更
Width	式の表示幅を指定
Repeat	表示する項目数を指定

各々の式には、フォーマット指定 (「 % 」文字) と同じ数のオペランドが必要です。オペランドが不足していると、CodeScape はエラーを発生します。すべての式オペランドは、評価できなければなりません。オペランドの評価結果がセクション相対アドレスとなった場合は、「 SectionName : Value 」という文字列が表示されます。

フォーマット指示子に使用する文字

フォーマット指示文字は、ポインタの制御と式の表示に影響するフォーマット命令に使用されます。

フォーマット指示文字とその意味

文字	意味
D, d	符号付き 10 進整数
C, c	ASCII 文字
U, u	符号なし 10 進整数
O, o	符号なし 8 進整数
X, H	符号なし 16 進整数 (A-F を使用)
x, h	符号なし 16 進整数 (a-f を使用)
S, s	ヌル終了の ASCII 文字列へのポインタ
T, t	HH/MM/SS 形式で時間を表示
!	パラメータ式を文字列として表示
l, i	逆アセンブルする命令へのポインタ

ポインタパラメータの使用例

文字	意味
% d	符号付き 10 進整数としてのフォーマットパラメータ
% u	符号なし 10 進整数としてのフォーマットパラメータ
% H	符号なし 16 進整数 (「A」～「F」を使用) としてのフォーマットパラメータ

ポインタ修飾子 (modifier)

パラメータポインタは、位置 0 から始めて、現在の式の位置を指示します。コマンドポインタ修飾子はパラメータポインタの位置を変更し、直接「%」記号の後に続きます。修飾子は、オプションの「+」または「-」記号を付した 10 進数で構成され、「#」記号で終わります。

構文

[+ | -] Number #

ポインタ修飾子の記号の意味

文字	意味
+ -	パラメータポインタの位置を、現在の位置から相対的に変更
Number	パラメータポインタの絶対値、または「+」「-」記号と併せて使用された場合は、相対移動量

注意

ポインタを最初のポインタパラメータの前にある値にセットすると、CodeScape はそれを最初のパラメータにセットします。ポインタを最後のポインタパラメータの後にある値にセットすると、その後の指定子は無効となり、表示文字列にそのままコピーされます。

例

3つのパラメータを符号付き10進整数で逆順に表示するためのフォーマット文字列。

```
% 2 # d % l # d % 0 # d
```

パラメータを最初に16進で、次に10進で表示するためのフォーマット文字列。

```
"% 0 # x , % - l # d "
```

幅修飾子(modifier)

オプションの幅修飾子(modifier)は、式の表示幅を指定します。その後に#修飾子が続きます(ポインタ修飾子が指定されない場合は「%」)。幅は、10進数か、後続のパラメータ式の値です。

構文

```
[ - ][ Number | * ]
```

幅修飾子(modifier)の記号の意味

記号	フィールド幅の意味
-	左揃え。「-」記号がなければフィールドは右揃え。
Number	10進数。表示フィールドを0で埋める場合は、数字に0を前置する。
*	次のパラメータ式の値にセット。

注意

「%s」フォーマットでは、幅は表示する最大文字数を意味します。

幅修飾子の使用例

記号	フィールド幅の意味
% 4x	符号なし16進整数(a-fを使用) 右揃え4桁
% - 8s	左揃え8文字の文字列
% 08X	8桁符号なし16進整数(A-Fを使用) 0を詰める
% 3 # - 15S	4番目のパラメータを15文字の左揃え文字列
% * s	次のパラメータの値に従った文字列
% 4 # * d	次のパラメータの値に従った4桁の右揃え符号付き10進整数

繰り返し修飾子 (modifier)

オプションの繰り返し修飾子 (modifier) は、表示する項目数を制御し、指定する場合はポインタ修飾子と幅修飾子がそれに続きます。修飾子は「@」記号にオプションのサイズ修飾子が続き、表示する項目数で終了します。

表示項目の区切りは次の通りです。

- ・フォーマット指定子が 10 進または 8 進の場合は、カンマ
- ・指定子が 16 進の場合は、スペース
- ・指定子が文字の場合は、カンマやスペースなし

複数バイトの項目にフェッチする場合、Dev.Box のプロセッサのエンディアンは保持されます。

注意

フォーマット指定子が文字列や命令の場合は、繰り返し修飾子は無効です。

構文

@ [Size] Number

繰り返し修飾子の記号

記号	意味
@	繰り返し修飾子の開始
Size	メモリからフェッチされる項目のサイズ
Number	表示される項目数を意味する 10 進数

オプションのコマンドライン文字

文字	意味
b	バイト
w	ワード
t	トリプル
l	ロング

6.4 Dev.Box のシミュレーション

シミュレータは、Dev.Box の SH4 プロセッサの動作を自分のコンピュータ上でシミュレーションするものです。[Memory] リージョンと [Register] リージョンについては実際の Dev.Box を使用します。

[Simulator] リージョンでシングルステップを行うと、カーソルはパイプライン中で現在実行されている命令を指します。シミュレーションの間、PC は現在の命令の先にある命令をフェッチします。

プログラムのフローの変化によって実行されない命令もあります。たとえば、ブランチの後にある命令などです。

注意

CodeScape の他のリージョンでシングルステップすると、カーソルは PC (program counter) の位置に表示されます。

6.4.1 シミュレータによって生成される情報

プロジェクトをシミュレーションすると、各命令がシミュレーション スロット (時間) 中に実行されます。シミュレータが進むに従い、このパイプライン動作の次の情報がそのリージョンに表示されます。

- ・ソースコードのメモリ中のアドレス
- ・各命令の Op コード
- ・メモリのあるアドレスの各命令を実行するのに必要な CPU 時間
- ・各命令の Op コードの逆アセンブル
- ・パイプライン動作の各ステージのスロット情報
 1. 縦のカーソルは、各スロットの命令を実行するのにプロセッサが必要とする時間を表す。
 2. 水平のカーソルは、アクティブなスロット中で時間を割り当てられている命令を表す。
- ・プロセッサのステータス情報

プロセッサのステータス情報

アクティブなスロットに関する情報はステータスバーに表示されます。

ステータス	内容
Diagnosis :	ストールの原因とそのタイプ
Cache :	読み出し / 書き込みのミスがあったときに起こるキャッシュメモリのストール
System clock :	現在のカーソル位置までに必要としたプロセッサ動作の全時間

注意

シミュレータによって生成された情報は、*.sim という拡張子でコンフィギュレーション ファイルに保存できます。

実行時間

実行時間は、ある命令の「ex (実行)」フェーズの開始から次の命令の「ex」フェーズの開始までに積算された CPU 時間です。

パイプラインのやりとり

シミュレータは、パイプラインのあるステージで命令の機能を評価します。

次の命令は、r0 に保存されたアドレスから 32 ビット読み出し、次にその結果を r3 にセットするように、プロセッサに指示します。

```
mov. L @ r0 , r3
```

命令がシミュレートされたスロット (時間) 中に実行されると、次の命令ステージがシミュレートされたパイプラインに表示されます：

IF | ID | EX | MA | WB

この命令の実行中に次の操作が行われます。

- ・ IF ステージに命令の op コードがメモリから読み出されます。
- ・ ID ステージに命令がデコードされます。
- ・ EX ステージに命令の実行が開始され、r0 の内容が読み出されます。
- ・ MA ステージに r0 の指すメモリがアクセスされ、その値がデータ バスに保存されます。
- ・ WB ステージにデータ バスに保存された値が、r3 の指すメモリに書き込まれます。

ニーモニック	意味	*.sim フォーマット
IF	命令がフェッチされた	F
if	命令のダミーフェッチ (前の実際のフェッチでフェッチされたデータのワードアクセスによる)	f
ID	命令デコード	D
EX	命令の実行	E
EX -	命令の実行 (命令が生成したストールを伴う)	E
MA	メモリアクセス	M
MA -	メモリアクセス (命令が生成したストールを伴う)	M
MAM	オンチップのマルチプライヤへのアクセスを伴うメモリアクセス (場合によっては前の命令の最後の「mm」とオーバーラップする)	M
mm	マルチプライヤの動作中 (ビジー)	m
WB	メモリアクセスの後に、データがレジスタにストアされた (メモリアクセス後のレジスタへのデータの書き戻し)	W

プロセッサの動作

各スロットによって蓄積された CPU 時間は、動作終了時にプロセッサの状態を表示するためにハイライトで表示されます。種々の色とニーモニックによって、各フェーズのパイプラインのやりとりが示されます。

[Pipeline] リージョンのプロセッサ動作

色	プロセッサの状態
黒	OK
赤	ストール
青	キャッシュのミス
ピンク	ストールとキャッシュのミス

6.4.2 パイプラインのストール

ある命令からのステージが、前または後の命令のステージと競合状態にある時にストールが発生します。これによりパイプラインの動作が遅くなります。シミュレーションによりプロセッサのパイプラインにストールが表示されることがあります。次のいずれかの方法でストールを除去します。

- ・ 命令順によるストールを取り除くには、命令の順序を変えます。
- ・ 命令の整列によるストールを取り除くには、メモリの命令アドレスを移動します。

注意

命令ステージと実行状態の競合に関する詳細については、日立 7091 シリーズのプログラミングマニュアルを参照してください。

シミュレータが識別できるパイプライン命令ストール

記号	ストール種類	可能な場合のパイプラインスピード改善法
-	自動	自動ストールについては何もできません。例: TAS、TRAP A、RTE
=	マルチプライヤアクセス (命令順によるストール)	マルチプライヤを使用する命令を連続して配置しない
~	メモリアクセスに伴うレジスタライトバック (命令順によるストール)	メモリからのロードを伴う命令に続く命令がロード命令と同じデスティネーションレジスタを使用しないようにする。
>	メモリアクセスに伴う命令フェッチ (命令の整列によるストール)	MA ステージを持つ命令をオンチップメモリのロングワード境界から開始するように配置する (可能な場合)。

注意

パイプライン命令のストールに関する詳細については、日立 7091 シリーズのプログラミングマニュアルを参照してください。

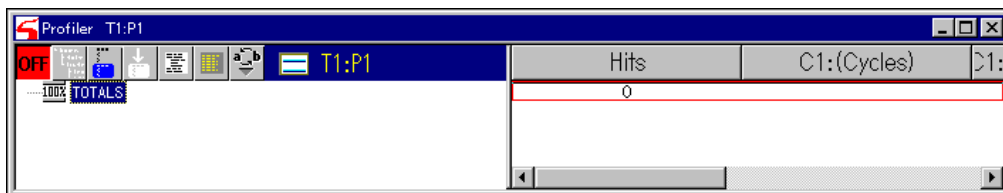
6.4.3 シミュレーションの結果を読む

シミュレータは、プログラムシミュレーション中の各アセンブラ命令のパイプライン動作に関する情報を生成します。プロセッサパフォーマンスの低下は、すべて [Simulator] リージョンに示す結果に現れます。シミュレーション結果を印刷することもできます (シミュレーションの結果の読み方の例については、シミュレータのチュートリアルを参照してください)。

6.5 プログラムファイルのプロファイル

プロファイラは日立 SH シリーズのプロセッサ用に書かれたプログラムファイルの実行時の振る舞いを調べる強力な解析ツールです。

プロファイラを使うことによって、プログラムにどれだけ時間がかかっているか、関数が実行時にどう呼ばれているかがわかります。難解なコードを解析し、プロファイラが生成する情報を使用することができます。



注意

正確な結果を得るためには、あらかじめ DACHECK を実行してキャッシュをオフにします。プロファイラとシミュレータは同時に実行してはいけません。

プロファイラを開く

[Target] リージョンで欲しいプロファイラのプロセッサを選択します。

- ・ [Debug] メニューをクリックして、[Profiler] をクリックします。
- ・ [Targer] リージョンで右クリックし、[Profiler] をクリックします。
- ・ ショートカットキーを利用します ([Ctrl] + [Alt] + [X])。

6.5.1 プロファイラのショートカットメニューを使用する

メニュー	内容
File	プロファイル記録のロード/セーブを行う。
Enable Profiler	プログラムファイルでプロファイルを開始、終了する。
Trace Tree Profile Display	呼び出す関数、呼ばれる関数の検索。このオプションは将来のリリースで実行できます。
Function Profile Display	呼び出す関数、呼ばれる関数の検索。また、関数の呼び出しにどれだけ時間がかかり、関数が何回呼ばれたか。
Function Profile Filter	すべての関数、すべてのタグ付き関数、すべてのタグなし関数の表示方法を変更する。
Untag All	すべてのタグ付き関数のタグをはずす。
Sort	関数名、関数のヒット数、関数の実行、最小サンプル周期、最大サンプル周期でプログラムのプロファイルを閲覧する。実行時に最小順、最大順に並び換えが可能。
Source Display	[Source] リージョンを表示する。
Disassembly Display	[Disassembly] リージョンを表示する。
Rename fundtion	関数の名前を変更する。
Profile Display Setup	[Profile] リージョン内の表示項目を設定する。
Setup	[Profiler Setup] ダイアログでプロファイル情報をどのように生成するかを設定する。1 秒間あたりに何回、どれぐらいの長さサンプルをプロファイルするかというオプションを設定する。

注意

【 Trace Tree Profile Display 】を選択する場合には、プログラムの実行前にプロファイラを実行しなければなりません。

特定の関数にタグを付けるには、プロファイル中でそのエントリをダブルクリックします。

6.5.2 [Profiler] ツールバーのオプション

オプション	アイコン
プロファイルの開始	
プロファイルの終了	
Trace Tree Profile と Function Profile の切り替え	
全関数からタグ付き関数への表示スイッチ	
タグ付き関数からタグ無し関数への表示スイッチ	
タグ無し関数から全関数への表示スイッチ	
一覧中で次のタグ付き関数への移動	
プログラムファイルのオリジナルソースコード表示	
インストラクションレベル (アセンブラコード) でのプログラムファイルの表示	
ソートオプションの切り替え	

注意

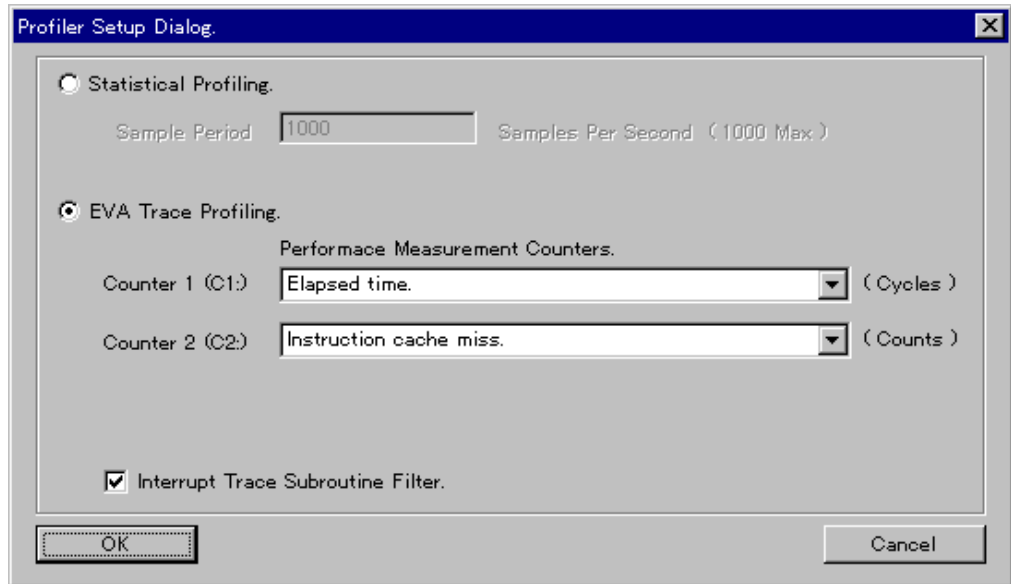
正確な結果を得るためには、あらかじめ DACHECK を実行してキャッシュをオフにします。
[Trace Profile] を選択する場合には、プログラムの実行前にプロファイラを実行しなければなりません

プロファイラの設定

[Profiler Setup] ダイアログのオプションは、どのようなプロファイル情報を生成するかを設定します。プロファイラを設定するには次のようにします。

1. プロファイラで右クリックし、[Setup] をクリックします。

[Profiler Setup] ダイアログが表示されます。



2. 次のいずれかを行います。

- ・ [Statistical Profiling] を選択し、プロファイラ実行時のサンプル周期、長さを指定する
- ・ [EVA Trace Profiling] を選択する

注意

正確な結果を得るためには、あらかじめ DACHECK を実行してスタブルーチンでキャッシュを使用しないようにします。

プロファイラ中のデバッグ識別子

すべての CodeScape のデバッグ機能はプロファイラの実行中に有効です (デバッグ機能は、プログラムの実行制御、コードのステップ実行、ブレークポイント、PC へのカーソルのセットといったコマンドを含みます)

6.5.3 プロファイルの表示形式

プログラムの使用時に、次のプロファイルができます。

- ・ [The Trace Tree Profile Display]
- ・ [The Function Profile Display]

[The Trace Tree Profile Display]

[The Trace Tree Profile Display] はそれぞれの呼ばれる関数、呼び出す関数を表示します。

関数がヒットすると、それぞれの関数がプログラムの実行中に何回ヒットしたかが表示されます。

注意

正確な結果を得るためには、あらかじめ DACHECK を実行してスタブルーチンでキャッシュを使用しないようにします。

[The Function Profile Display]

[The Function Profile Display] はそれぞれの呼ばれる関数、呼び出す関数を表示します。[Function Profile] は、それぞれの関数の実行中に合計どれだけの時間がかかったのか、またそれぞれの関数にかかった時間、その関数の子に相当する関数にかかった時間を表示します。

[Result] フィールド	内容
Hits : Num/Percent	それぞれの関数の呼び出し時間。それぞれの関数の呼ばれた回数、それぞれの関数による関数の呼び出し割合を表示。
Exe. Time : Clocks/Percent	それぞれの関数の実行時間。それぞれの関数の合計時間、それぞれの関数の実行時間の割合を表示。

関数の検索

関数を検索するには、次の操作を行います。

1. 検索を開始したい箇所に挿入ポインタを移動します。
2. 現在のプロファイルで検索したい文字列を入力します。
プロファイラが自動的に検索し、もっとも近いものを表示します。
3. 検索を続けるなら、[Enter] キーをクリックします。

注意

検索は文字列、または文字列の一部も可能です。
プロファイラは照合するものを見つけ、降順に表示します。

第 7 章

式

CodeScape には 2 つの式エバリュエータがあります。1 つは C/C++ の式、もう 1 つはアセンブラ式の評価に使用されます。

7.1 C/C++ 式

C/C++ 式エバリュエータは C 形式の式を受け付けます。

演算子の優先順位

演算子	タイプ	使用法	説明
()	基本		カッコ
[]	基本	pointer [expr]	サブスクリプト
.	2 項演算子	object.member	メンバー選択
- >	2 項演算子	pointer - > member	メンバー選択
sizeof ()	単項演算子	sizeof (expr)	オブジェクトのサイズ
sizeof ()	単項演算子	sizeof (type)	型のサイズ
-	単項演算子	- expr	単項マイナス
+	単項演算子	+ expr	単項プラス
~	単項演算子	~expr	ビット単位否定
!	単項演算子	! expr	論理否定
*	単項演算子	* expr	間接参照
&	単項演算子	& lvalue	... のアドレス
*	2 項演算子	expr * expr	乗算
/	2 項演算子	expr/expr	除算
%	2 項演算子	expr % expr	モジュロ (余り)
+	2 項演算子	expr + expr	加算
-	2 項演算子	expr - expr	減算
< <	2 項演算子	expr < < expr	左シフト
> >	2 項演算子	expr > > expr	右シフト
<	2 項演算子	expr < expr	より小
< =	2 項演算子	expr < = expr	より小か等しい
>	2 項演算子	expr > expr	より大
> =	2 項演算子	expr > = expr	より大か等しい
= =	2 項演算子	expr = = expr	等しい
! =	2 項演算子	expr ! = expr	等しくない
&	2 項演算子	expr & expr	ビット単位 AND
^	2 項演算子	expr ^ expr	ビット単位 XOR
	2 項演算子	expr expr	ビット単位 OR
& &	2 項演算子	expr & & expr	論理 AND
	2 項演算子	expr expr	論理 OR

前記の演算子は、次のオペランドを操作対象とします。

オペランド	定義
定数（浮動小数点または整数）	定数：「0x」を前置した 16 進数、「0」を前置した 8 進数、「U」を後置した無符号数。たとえば「A」は無効。
レジスタ	有効なレジスタ名。
シンボル	シンボル名はそのタイプが影響する。たとえば（char chr = 'A'）と定義された変数は、評価されると「A」を返す。オブジェクトのアドレスを取得するには、「& chr」とすることが必要。

演算子の制限

- ・タイプキャスト（typecast）。整数、浮動小数点、無符号整数、int *、char * などの基本的なタイプのタイプキャストは有効です。struct basic * などのユーザー定義タイプに対するタイプキャストは無効です。
- ・スコープ演算子「::」。クラス エLEMENT 名の一部としてスコープ演算子は有効です。例：
c_basic::print
- ・このリリースでは、「=、+=、*=、++、--」などの代入演算子はインプリメントされていません。
- ・このリリースでは、file/line number のフォーマットはインプリメントされていません。

7.2 アセンブラ式

演算子の優先順位

演算子	タイプ	使用法	説明
()	基本	(expr)	カッコ
[]	基本	[expr]	... のアドレス
-	単項演算子	- expr	負の expr
+	単項演算子	+ expr	正の expr
~	単項演算子	~expr	ビット単位否定
< <	2 項演算子	expr < < expr	左シフト
> >	2 項演算子	expr > > expr	右シフト
&	2 項演算子	expr & expr	論理積 (AND)
!	単項演算子	! expr	論理否定 (NOT)
	2 項演算子	expr expr	論理和 (OR)
^	2 項演算子	expr ^ expr	排他的論理和 (XOR)
*	2 項演算子	expr * expr	乗算
/	2 項演算子	expr / expr	除算
%	2 項演算子	expr % expr	モジュロ (余り)
+	2 項演算子	expr + expr	加算
-	2 項演算子	expr - expr	減算
=	2 項演算子	expr = expr	等しい
< >	2 項演算子	expr < > expr	等しくない
<	2 項演算子	expr < expr	より小
< =	2 項演算子	expr < = expr	より小か等しい
>	2 項演算子	expr > expr	より大
> =	2 項演算子	expr > = expr	より大か等しい

前記の演算子は次のオペランドを操作対象とします。

オペランド	定義		
定数 (整数)	定数は、いくつかの演算子に種々の基数で定義できます。		
	任意 16 進 10 進 2 進	X_ < number >	X は一桁の基数 「 \$ 」または「 Ox 」を前置、「 h 」を後置 「 # 」を前置、「 d 」を後置 「 % 」を前置、「 b 」を後置
レジスタ	有効なレジスタ名		
シンボル	シンボルはラベルに評価されます。タイプ (char chr = 'A ') の変数が評価されると、変数「 A 」のアドレス (のラベル) を返します。ラベルの「 b_ 」 「 w_ 」 「 l 」 は、それぞれバイト、ワード、ロングを意味します。 [symbol] @ b、[symbol] @ w、[symbol] @ l ファイル名の : < number > は行番号		

7.2.1 [Expression evaluator] ダイアログボックス

[Expression evaluator] は、レジスタの編集やアドレスへのジャンプなどを含め、いくつかの操作に使用できるダイアログボックスです。

[Expression evaluator] のオプション

項目	目的
[Expression] コンボ ボックス	既存の式を編集。またはヒストリ リストから選択。
[Result] ボックス	エラーメッセージを含め、式評価の結果を表示。
[Expression Format] ラジオ ボタン	式フォーマットとして [C/C ++] または [Assembly] を選択。
[Default radix] ラジオボタン	式に使用する基数を 2 進、8 進 10 進、16 進から選択するか、[Other] テキストボックスにその他の基数を指定。C の式では、出力の基数を指定するのみ。
[Evaluate] ボタン	[Expression] コンボテキストボックス中の式を評価。
[Symbol] ボタン	[Symbol Completion] ダイアログボックスを使用して、プログラムファイル中で使用できるシンボルを検索。
[File] ボタン	プログラムをビルドするのに使用されるファイルの一覧を、[List Files in Program File] ダイアログボックスに表示。このダイアログボックスは「file : line number」情報のアドレスへのアクセスも提供。
[Lock] チェックボックス	現在の式をファイルまたはシンボルにロック。

7.2.2 [Symbol Completion] ダイアログボックス

プログラムファイル中のシンボルを検索するには、[Symbol Completion] ダイアログボックスを使用します。

項目	目的
[Search String] テキストボックス	検索するシンボルの最初の数文字を入力。
[Only Search For Symbols Within Scope] チェックボックス	スコープ内でシンボルの検索 (チェックボックスをオン) プログラム全体で検索 (チェックボックスをオフ)
[Possible Completions] テキストボックス	現在の検索文字列にマッチするすべてのシンボルを一覧表示。
[Lookup] ボタン	検索を再開するには [Lookup] をクリック。
[OK] ボタン	現在の検索文字列を受け入れる。
[Cancel] ボタン	現在の検索文字列を無視する。

CodeScape の実行方法を指定するには、コマンドラインのオプションを使用します。たとえば、CodeScape は Codewright エディタなどのアプリケーションやバッチファイルから起動できます。

8.1 コマンドラインからの CodeScape の起動

コマンドラインから CodeScape を起動するには、CodeScape と入力し、その後にオプションのスイッチを入力します。スイッチは必ずスペースで区切りますが、スイッチの引数の中でスペースを使用してはいけません。

コマンドライン構文は次の通りです。

```
codescape [ Switch ] ...
```

コマンドラインスイッチ

[- | /] ?

コマンドラインのヘルプ。

[- | /] nologo

スプラッシュ スクリーン (splash screen) を表示せずに CodeScape を実行。

[- | /] c

Crosslib Verbose モード。このスイッチは Fileserver ライブラリ (LIBCROSS.A) のコマンドを実行するに従い、付加的な情報を [Log] ウィンドウに表示します。

[- | /] i = Session

Project Info. を使用。このスイッチは、SessionFile によって指定されたセッションファイルを使用して CodeScape を起動します。このセッションファイルには Dev.Box への接続方法、各 Dev.Box に使用するオブジェクトファイル、更新レート、ブレークポイント、ウォッチ式、ログ式、ウィンドウの位置が含まれます。セッションファイルにメモリ範囲が指定されない場合、CodeScape は DEFAULT.SSN の中からそれを探します。

[- | /] t # p # [b] [n] [ProgramFile]

このスイッチを使用して、使用する Dev.Box (t #) とプロセッサ (p #) をロードするプログラムファイルを指定します。プロセッサはプロセッサ ID # (0-7) により識別されます。ここで 1 = Master、2 = Slave です。プログラムファイルは ProgramFile で指定します。このスイッチを使用して Dev.Box のオブジェクトファイルを指定すると、セッションファイルでの指定が無視されます。

コマンドはオブジェクトファイル (b) からバイナリをロードし、デバッグ情報を抑止 (n) します。

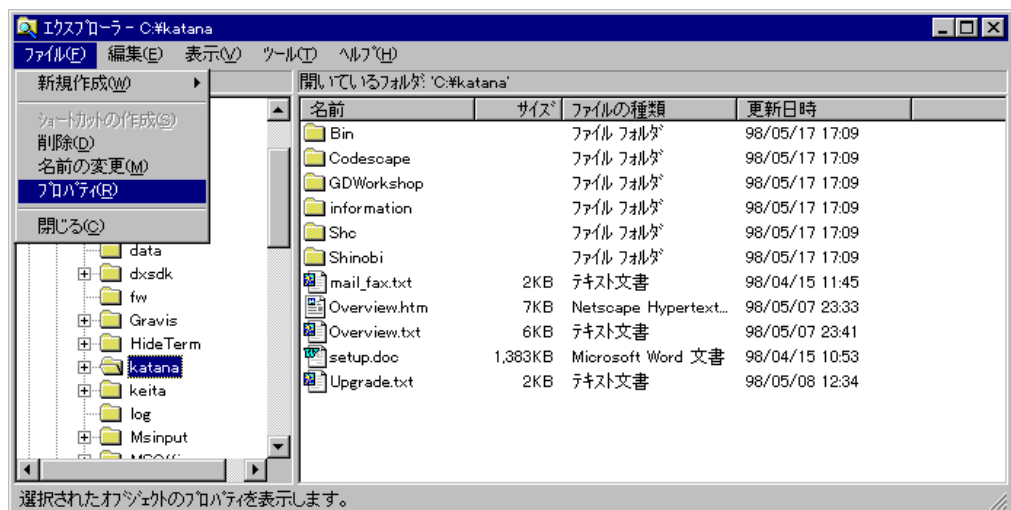
注意

n コマンドにはコードやシンボルは不要です。b コマンドなしで n コマンドを使用すると、無視されます。

CodeScape が使用するファイル

ファイル名	内容
SessionFile	以前のデバッグセッションを回復するための情報が含まれます。
ProgramFile	オブジェクトファイル。アセンブラやコンパイラによって生成されたバイナリ、そしてオプションでソースレベルのデバッグ情報、シンボルテーブル情報が含まれます。

ファイルやフォルダのプロパティを変更するには次のようにします。



1. プロパティを変更するファイルやフォルダをクリックします。
2. [ファイル] メニューの [プロパティ] をクリックします。

注意

ファイルのアイコンをドキュメントにドラッグしたり、ショートカットアイコンをドラッグすることもできます。

コマンドラインの使用例

例 1

CodeScape を起動してセッションファイル SESSION.SSN に含まれる情報にしたがってデバッグセッションを回復するには、次のように入力します。

```
codescape -i = session
```

例 2

この例は CodeScape を起動してデバッグセッションを回復します。ファイル TEST.COF からのバイナリ (...b...) は Dev.Box7 (- t7... : test) にダウンロードされますが、シンボリック情報はロードされません (...n...)。

```
codescape / t7p1bn : test
```

例 3

n コマンドにはコードやシンボルは不要です。n コマンドを b コマンドなしで使用すると無視されます。次の例では CodeScape は起動されません。

```
codescape - t1p1n : test
```


一般的操作

頻繁に使用するデバッグ操作には、キーボードショートカットが用意されています。すべての操作には、インターフェイスに下線文字で示されるアクセスキーが使用できます。

A.1 メニューバー上のショートカットとアクセスキー

メニューバー上の項目にアクセスするためにキーボードを使用するには次のようにします。

- ・ [F10] キーを押し、カーソルキーで項目を選択してから [Enter] キーを押す。
- ・ メニューのキーボードショートカットを押して、カーソルキーで項目を選択してから [Enter] キーを押す。

[File] メニュー [Alt] + [F] キー

[File] メニューのコマンドにキーボードを使用する場合

目的	キーボードショートカット	アクセスキー
新規セッションを開く	[Ctrl] + [Shift] + [N]	[F10] [F] [N]
既存セッションを開く	[Ctrl] + [O]	[F10] [F] [O]
セッションを閉じる	なし	[F10] [F] [C]
開いているセッションを保存	[Ctrl] + [S]	[F10] [F] [S]
セッションに名前を付けて保存	なし	[F10] [F] [A]
ソフトリセットで Dev.Box をリセット	なし	[F10] [F] [T] [S]
ハードリセットで Dev.Box をリセット	なし	[F10] [F] [T] [H]
プログラムファイルのロード	[Ctrl] + [Shift] + [C]	[Alt] [F] [L]
セッションのリスタート	[Ctrl] + [Shift] + [R]	[Alt] [F] [E]
ファイルのバイナリ部分を保存	なし	[Alt] [F] [B]
ファイルのバイナリ部分をロード	なし	[Alt] [F] [D]
アクティブリージョンの内容を印刷	[Ctrl] + [P]	[Alt] [F] [P]
プリンターの設定	なし	[Alt] [F] [R]
CodeScape を終了する	[Alt] + [F4]	[Alt] [F] [X]

[Edit]メニュー [Alt] + [E]キー

[Edit]メニューのコマンドにキーボードを使用する場合

目的	キーボードショートカット	アクセスキー
最後の操作を元に戻す	[Ctrl] + [Z]	[Alt] [E] [U]
選択領域を切り取る	[Ctrl] + [X]	[Alt] [E] [T]
選択領域をコピー	[Ctrl] + [C]	[Alt] [E] [C]
クリップボードから貼り付け	[Ctrl] + [V]	[Alt] [E] [P]
項目を検索	[Ctrl] + [F]	[Alt] [E] [F]
次項目を検索	[F3]	[Alt] [E] [N]
検索項目を置換	なし	[Alt] [E] [L]
指定アドレスにジャンプ	[Ctrl] + [G]	[Alt] [E] [G]

[View]メニュー [Alt] + [V]キー

[View]メニューのコマンドにキーボードを使用する場合

目的	アクセスキー
ツールバーを表示リストに追加、削除	[Alt] [V] [T]
ステータスバーの表示 / 非表示	[Alt] [V] [S]
アクティブリージョンのプロパティを変更	[Alt] [V] [P]

[Project]メニュー [Alt] + [P]キー

[Project]メニューのコマンドにキーボードを使用する場合

目的	キーボードショートカット	アクセスキー
現在のプロジェクトコマンドを設定	なし	[Alt] [P] [P]
エディタコマンドを設定	なし	[Alt] [P] [E]
現在のプロジェクトをビルド	[Ctrl] + [M]	[Alt] [P] [M]
プロジェクトソースファイルへのパスを設定	なし	[Alt] [P] [S]
ファイルサーバーディレクトリの設定	なし	[Alt] [P] [F]
ファイルサーバー最適化の ON/OFF	[Alt] + [P]	[Alt] [P] [O]

[Debug] メニュー [Alt] + [D] キー

[Debug] メニューのコマンドにキーボードを使用する場合

目的	キーボードショートカット	アクセスキー
プロセッサ実行コマンドにアクセス	なし	[Alt] [D] [E]
全プロセッサを同時に実行	[Ctrl] + [F9]	[Alt] [D] [E] [A]
全プロセッサを停止	なし	[Alt] [D] [E] [L]
ひとつのプロセッサを実行	[F9]	[Alt] [D] [E] [R]
指定アドレスまでプロセッサを実行	[Shift] + [F9]	[Alt] [D] [E] [S]
カーソル位置までプロセッサを実行	[Alt] + [F9]	[Alt] [D] [E] [C]
プロセッサの実行を停止	[F9]	[Alt] [D] [E] [O]
コード 1 行をシングルステップ	[F7]	[Alt] [D] [E] [G]
コード 1 行を強制 Step into	なし	[Alt] [D] [E] [F]
コード 1 行を Step over	[F8]	[Alt] [D] [E] [V]
アンステップ	[Ctrl] + [F7]	[Alt] [D] [E] [N]
ステップ実行のアニメーション	なし	[Alt] [D] [E] [M]
コード 1 行を Step into	[Shift] + [F7]	[Alt] [D] [E] [I]
コード 1 行を Step Run Out	[Shift] + [F8]	[Alt] [D] [E] [U]
コード 1 行をステップ実行	[Alt] + [F7]	[Alt] [D] [E] [T]
特定位置までステップ実行	[Alt] + [F8]	[Alt] [D] [E] [P]
プロセッサ実行のリスタート	[Ctrl] + [Shift] + [R]	[Alt] [D] [E] [E]
ブレークポイント操作にアクセス	なし	[Alt] [D] [B]
ブレークポイントのオン / オフ	[F5]	[Alt] [D] [B] [K]
ブレークポイントの有効化	なし	[Alt] [D] [B] [N]
ブレークポイントの無効化	なし	[Alt] [D] [B] [I]
ブレークポイントの設定	[Ctrl] + [F5]	[Alt] [D] [B] [F]
全ブレークポイントのリセット	[Alt] + [F5]	[Alt] [D] [B] [L]
全ブレークポイントの有効化	[Ctrl] + [Shift] + [F5]	[Alt] [D] [B] [E]
全ブレークポイント無効化	[Ctrl] + [Alt] + [F5]	[Alt] [D] [B] [D]
全ブレークポイントの削除	[Shift] + [F5]	[Alt] [D] [B] [M]
カーソル位置を PC にセット	[Ctrl] + [Shift] + [P]	[Alt] [D] [C]
PC をカーソル位置にセット	[Ctrl] + [Alt] + [P]	[Alt] [D] [P]
指定アドレスにジャンプ	[Ctrl] + [G]	[Alt] [D] [G]
シミュレータの起動	[Ctrl] + [Alt] + [Z]	[Alt] [D] [S]
プロファイラの開始	[Ctrl] + [Alt] + [X]	

[Region]メニュー [Alt] + [R]キー

[Region]メニューのコマンドにキーボードを使用する場合

目的	キーボードショートカット	アクセスキー
リージョン分割コマンドの使用	なし	[Alt] [R] [S]
アクティブリージョンを左に分割	[Ctrl] + [Shift] + []	[Alt] [R] [S] [L]
アクティブリージョンを右に分割	[Ctrl] + [Shift] + []	[Alt] [R] [S] [R]
アクティブリージョンを上分割	[Ctrl] + [Shift] + []	[Alt] [R] [S] [U]
アクティブリージョンを下分割	[Ctrl] + [Shift] + []	[Alt] [R] [S] [D]
アクティブリージョンの削除	[Ctrl] + [D]	[Alt] [R] [D]
リージョンタイプ一覧にアクセス	なし	[Alt] [R] [T]
[Disassembly]リージョンを作成	[Alt] + [1]	[Alt] [R] [T] [D]
[Log]リージョンを作成	[Alt] + [2]	[Alt] [R] [T] [O]
[Local Watch]リージョンを作成	[Alt] + [3]	[Alt] [R] [T] [L]
[Memory]リージョンを作成	[Alt] + [4]	[Alt] [R] [T] [M]
[Register]リージョンを作成	[Alt] + [5]	[Alt] [R] [T] [R]
[Source]リージョンを作成	[Alt] + [6]	[Alt] [R] [T] [S]
[Watch]リージョンを作成	[Alt] + [7]	[Alt] [R] [T] [W]
[Edit]リージョンを作成	[Alt] + [8]	[Alt] [R] [T] [E]
[Call Stack]リージョンを作成	[Alt] + [9]	[Alt] [R] [C]
リージョンの表示を更新	[Ctrl] + [U]	[Alt] [R] [U]

[Tools]メニュー [Alt] + [T]キー

[Tools]メニューのコマンドにキーボードを使用する場合

目的	キーボードショートカット	アクセスキー
プロセッサのシミュレート	[Ctrl] + [Alt] + [Z]	[Alt] [T] [S]
プロファイラ	[Ctrl] + [Alt] + [X]	[Alt] [T] [P]
キーボードショートカットのカスタマイズ	なし	[Alt] [T] [C] [K]
メニューの追加	なし	[Alt] [T] [C] [T]

[Window]メニュー [Alt] + [W]キー

[Window]メニューのコマンドにキーボードを使用する場合

目的	キーボードショートカット	アクセスキー
新規ウインドウを作成	[Ctrl] + [N]	[Alt] [W] [N]
ウインドウを重ねて表示	なし	[Alt] [W] [C]
ウインドウを並べて表示	なし	[Alt] [W] [T]
アイコン化したリージョンウインドウをすべてセッションウインドウの下に並べる	なし	[Alt] [W] [I]
ウインドウの比率を変えずにサイズを変更	なし	[Alt] [W] [P]
次回 CodeScape 起動時に最後にロードしたセッションファイルをロードする	なし	[Alt] [W] [L]
全ウインドウを閉じる	なし	[Alt] [W] [A]

[Help]メニュー [Alt] + [H]キーまたは[F1]キー

[Help]メニューのコマンドにキーボードを使用する場合

表示対象	キーボードショートカット	アクセスキー
ヘルプの内容	[F1]	[Alt] [H] [C]
ヘルプのトピック一覧	[F1]	[Alt] [H] [H]
CodeScape バージョンと著作権情報	なし	[Alt] [H] [A]

A.2 ショートカットメニュー上のショートカットとアクセスキー

[Tab] キーを使用してリージョンに移動しアクティブにしてから、[Shift] + [F10] キーを押す、カーソルキーで項目を選択し、次に [Enter] キーを押します。

[Source] リージョン

[Source] リージョンでキーボードコマンドを使用する場合

目的	キーボードショートカット	アクセスキー
カーソルの同期	なし	[Shift] + [F10] [N]
ソースコードから生成された最初の行に対応するアドレスを表示	[Ctrl] + [A]	[Shift] + [F10] [A]
各コード行の行番号を表示	[Ctrl] + [L]	[Shift] + [F10] [L]
プロセッサ実行コマンドの使用	なし	[Shift] + [F10] [E]
全プロセッサの実行	[Ctrl] + [F9]	[Shift] + [F10] [E] [A]
全プロセッサの実行停止	なし	[Shift] + [F10] [E] [L]
ひとつのプロセッサの実行	[F9]	[Shift] + [F10] [E] [R]
指定アドレスまで実行	[Shift] + [F9]	[Shift] + [F10] [E] [S]
プロセッサを現在のカーソル位置まで実行	[Alt] + [F9]	[Shift] + [F10] [E] [C]
プロセッサの実行停止	[F9]	[Shift] + [F10] [E] [O]
1 コード行をシングルステップ	[F7]	[Shift] + [F10] [E] [G]
1 コード行を強制ステップオーバー	[Shift] + [F7]	[Shift] + [F10] [E] [F]
1 コード行をステップオーバー	[F8]	[Shift] + [F10] [E] [V]
1 コード行をアンステップ	[Ctrl] + [F7]	[Shift] + [F10] [E] [N]
1 コード行へステップランイン	[Shift] + [F7]	[Shift] + [F10] [E] [I]
1 コード行からステップランアウト	[Shift] + [F8]	[Shift] + [F10] [E] [U]
1 コード行をステップ実行	[Alt] + [F7]	[Shift] + [F10] [E] [T]
特定位置までステップ実行	[Alt] + [F8]	[Shift] + [F10] [E] [P]
プロセッサ実行のリスタート	[Ctrl] + [Shift] + [R]	[Shift] + [F10] [E] [E]
ブレークポイント操作を使用	なし	[Shift] + [F10] [B]
ブレークポイントのオン / オフ	[F5]	[Shift] + [F10] [B] [K]
ブレークポイントの有効化	なし	[Shift] + [F10] [B] [N]
ブレークポイントの無効化	なし	[Shift] + [F10] [B] [I]
ブレークポイントの設定	[Ctrl] + [F5]	[Shift] + [F10] [B] [L]
全ブレークポイントのリセット	[Alt] + [F5]	[Shift] + [F10] [B] [E]
全ブレークポイントの有効化	[Ctrl] + [Shift] + [F5]	[Shift] + [F10] [B] [D]
全ブレークポイントの無効化	[Ctrl] + [Alt] + [F5]	[Shift] + [F10] [B] [K]
全ブレークポイントの削除	[Shift] + [F5]	[Shift] + [F10] [B] [M]
カーソル位置を PC にセット	[Ctrl] + [Shift] + [P]	[Shift] + [F10] [C]
PC をカーソル位置にセット	[Ctrl] + [Alt] + [P]	[Shift] + [F10] [P]
指定アドレスにジャンプ	[Ctrl] + [G]	[Shift] + [F10] [G]
指定ファイルにジャンプ	[Ctrl] + [Shift] + [F5]	[Shift] + [F10] [F]
リージョンツールを使用	なし	[Shift] + [F10] [T] []
アクティブリージョンのプロパティの変更	[Alt] + [V] + [P]	
アクティブリージョンの色とフォントのセット	なし	
アクティブリージョンのタブ幅のセット	[Ctrl] + [T]	
アクティブリージョンの更新レートのセット	なし	

[Disassembly] リージョン

[Disassembly] リージョンでキーボードコマンドを使用する場合

目的	キーボードショートカット
カーソルの同期	なし
逆アセンブルコードのロケーションアドレスの表示	[Ctrl] + [A]
逆アセンブルコードのシンボルラベル置換の表示	[Ctrl] + [B]
逆アセンブルされたリージョンの opcode ワードの表示	[Ctrl] + [W]
オペランドの値を 16 進で表示	[Ctrl] + [H]
大文字 / 小文字表示を切り替え	[Ctrl] + [U]
各コード行のシンボルを表示	[Ctrl] + [Y]
実効アドレスとリテラルの表示	[Ctrl] + [I]
プロセッサ実行コマンドの使用	なし
全プロセッサの実行	[Ctrl] + [F9]
全プロセッサの停止	なし
ひとつのプロセッサの実行	[F9]
ひとつのプロセッサを指定アドレスまで実行	[Shift] + [F9]
ひとつのプロセッサを現在のカーソル位置まで実行	[Alt] + [F9]
プロセッサ実行の停止	[F9]
1 コード行をシングルステップ	[F7]
1 コード行を強制 Step into	[Shift] + [F7]
1 コード行を Step over	[F8]
1 ステップ元に戻す	[Ctrl] + [F7]
プロセッサ実行のリスタート	[Ctrl] + [Shift] + [R]
ブレークポイント	なし
ブレークポイントのオン / オフ	[F5]
ブレークポイントの有効化	なし
ブレークポイントの無効化	なし
ブレークポイントの設定	[Ctrl] + [F5]
全ブレークポイントのリセット	[Alt] + [F5]
全ブレークポイントの有効化	[Ctrl] + [Shift] + [F5]
全ブレークポイントの無効化	[Ctrl] + [Alt] + [F5]
全ブレークポイントの削除	[Shift] + [F5]
カーソル位置を PC にセット	[Ctrl] + [Shift] + [P]
PC をカーソル位置にセット	[Ctrl] + [Alt] + [G]
指定アドレスにジャンプ	[Ctrl] + [G]

[Watch] リージョン

[Watch] リージョンでキーボードコマンドを使用する場合

目的	キーボードショートカット	アクセスキー
構造の一部を削除	[Delete]	[Shift] + [F10] [D]
構造または配列の展開 / 折り畳み	[スペース]	[Shift] + [F10] [O]
新規ウォッチ式の挿入	[Ctrl] + [I]	[Shift] + [F10] [I]
現行リストの最後に変数を挿入	[Ctrl] + [A]	[Shift] + [F10] [A]
変数またはウォッチ式の値を変更	[Ctrl] + [Alt] + [E]	[Shift] + [F10] [V]
プロセッサ実行コマンドの使用	なし	[Shift] + [F10] [E]
全プロセッサの実行	[Ctrl] + [F9]	[Shift] + [F10] [A]
全プロセッサの停止	なし	[Shift] + [F10] [L]
ひとつのプロセッサの実行	[F9]	[Shift] + [F10] [R]
ひとつのプロセッサを指定アドレスまで実行	[Shift] + [F9]	[Shift] + [F10] [S]
プロセッサの実行停止	[F9]	[Shift] + [F10] [O]
プロセッサの実行をリスタート	[Ctrl] + [Shift] + [R]	[Shift] + [F10] [E]
ブレークポイント	なし	[Shift] + [F10] [B]
ブレークポイントのオン / オフ	[F5]	[Shift] + [F10] [B] [K]
ブレークポイントの有効化	なし	[Shift] + [F10] [B] [N]
ブレークポイントの無効化	なし	[Shift] + [F10] [B] [I]
ブレークポイントの設定	[Ctrl] + [F5]	[Shift] + [F10] [B] [F]
全ブレークポイントのリセット	[Alt] + [F5]	[Shift] + [F10] [B] [L]
全ブレークポイントの有効化	[Ctrl] + [Shift] + [F5]	[Shift] + [F10] [B] [E]
全ブレークポイントの無効化	[Ctrl] + [Alt] + [F5]	[Shift] + [F10] [B] [D]
全ブレークポイントの削除	[Shift] + [F5]	[Shift] + [F10] [B] [M]
変化をハイライト	なし	[Shift] + [F10] [H]
アクティブリージョンのプロパティを変更	なし	[Shift] + [F10] [R]

[Local Watch] リージョン

[Local Watch] リージョンでキーボードコマンドを使用する場合

目的	キーボードショートカット	アクセスキー
式または式構造の一部を削除	[Delete]	[Shift] + [F10] [D]
構造または配列の展開 / 折り畳み	[スペース]	[Shift] + [F10] [O]
変数またはウォッチ式の値を変更	[Ctrl] + [Alt] + [E]	[Shift] + [F10] [V]
プロセッサ実行コマンドの使用	なし	[Shift] + [F10] [E]
全プロセッサの実行	[Ctrl] + [F9]	[Shift] + [F10] [A]
全プロセッサの停止	なし	[Shift] + [F10] [L]
ひとつのプロセッサの実行	[F9]	[Shift] + [F10] [R]
ひとつのプロセッサを指定されたアドレスまで実行	[Shift] + [F9]	[Shift] + [F10] [S]
プロセッサの実行停止	[F9]	[Shift] + [F10] [E]、 [O]
プロセッサの実行をリスタート	[Ctrl] + [Shift] + [R]	[Shift] + [F10] [E]、 [E]
ブレークポイント操作の使用	なし	[Shift] + [F10] [B]
ブレークポイントの設定	[Ctrl] + [F5]	[Shift] + [F10] [B]、 [F]
全ブレークポイントのリセット	[Alt] + [F5]	[Shift] + [F10] [B]、 [L]
全ブレークポイントの有効化	[Ctrl] + [Shift] + [F5]	[Shift] + [F10] [B]、 [E]
全ブレークポイントの無効化	[Ctrl] + [Alt] + [F5]	[Shift] + [F10] [B]、 [D]
全ブレークポイントの削除	[Shift] + [F5]	[Shift] + [F10] [B]、 [M]
変化をハイライト	なし	[Shift] + [F10] [H]
アクティブリージョンのプロパティを変更	なし	[Shift] + [F10] [R]

[Memory] リージョン

[Memory] リージョンでキーボードコマンドを使用する場合

目的	キーボードショートカット	アクセスキー
メモリをバイトで表示	[Ctrl] + [B]	[Shift] + [F10] [B]
メモリをワードで表示	[Ctrl] + [W]	[Shift] + [F10] [W]
メモリをロングで表示	[Ctrl] + [L]	[Shift] + [F10] [L]
メモリをクワッドで表示	[Ctrl] + [Q]	[Shift] + [F10] [Q]
メモリの各バイトを ASCII 値で表示	[Ctrl] + [A]	[Shift] + [F10] [A]
1 行あたりのバイト数をセット	[Ctrl] + [Shift] + [L]	[Shift] + [F10] [S]
[Memory] リージョンの ASCII 値を変更	[Ctrl] + [Alt] + [A]	[Shift] + [F10] [D]
[Memory] リージョンの値を変更	[Ctrl] + [Alt] + [E]	[Shift] + [F10] [V]
メモリ中のポインタを追跡	[Ctrl] + [T]	[Shift] + [F10] [F]
特定のメモリ位置にジャンプ	[Ctrl] + [G]	[Shift] + [F10] [G]
書き込み保護のオン / オフ	[Ctrl] + [Shift] + [W]	[Shift] + [F10] [P]
プロセッサ実行コマンドの使用	なし	[Shift] + [F10] [E]
全プロセッサの実行	[Ctrl] + [F9]	[Shift] + [F10] [A]
全プロセッサの停止	なし	[Shift] + [F10] [L]
選択したプロセッサを実行	[F9]	[Shift] + [F10] [R]
選択したプロセッサを指定アドレスまで実行	[Shift] + [F9]	[Shift] + [F10] [S]
選択したプロセッサの実行を停止	[F9]	[Shift] + [F10] [O]
選択したプロセッサをリスタート	[Ctrl] + [Shift] + [R]	[Shift] + [F10] [E]
ブレークポイント操作の使用	なし	[Shift] + [F10] [B]
ブレークポイントのオン / オフ	[F5]	[Shift] + [F10] [B] [K]
ブレークポイントの有効化	なし	[Shift] + [F10] [B] [N]
ブレークポイントの無効化	なし	[Shift] + [F10] [B] [I]
ブレークポイントの設定	[Ctrl] + [F5]	[Shift] + [F10] [B] [F]
全ブレークポイントのリセット	[Alt] + [F5]	[Shift] + [F10] [B] [L]
全ブレークポイントの有効化	[Ctrl] + [Shift] + [F5]	[Shift] + [F10] [B] [E]
全ブレークポイントの無効化	[Ctrl] + [Alt] + [F5]	[Shift] + [F10] [B] [D]
全ブレークポイントの削除	[Shift] + [F5]	[Shift] + [F10] [B] [M]
リージョンのツールを使用	なし	[Shift] + [F10] [T]
メモリ中のパターンを検索	[Ctrl] + [F]	[Shift] + [F10] [T] [F]
1 ブロックのメモリを 16 進でファイルに書き出す	なし	[Shift] + [F10] [T] [H]
アクティブリージョンのプロパティの変更	なし	[Shift] + [F10] [R]

[Register] リージョン

[Register] リージョンでキーボードコマンドを使用する場合

目的	キーボードショートカット	アクセスキー
現在のインクリメント値(デフォルトは1)をレジスタの内容に適用	+	[Shift] + [F10] [I]
現在のデクリメント値(デフォルトは1)をレジスタの内容に適用	-	[Shift] + [F10] [D]
インクリメント値/デクリメント値の変更	なし	[Shift] + [F10] [C]
最後の操作による変化を表示	なし	[Shift] + [F10] [H]
現在アクティブな [Register] リージョンへのデータの書き込み禁止	[Ctrl] + [Shift] + [W]	[Shift] + [F10] [P]
選択されたレジスタ値の変更	[Ctrl] + [Alt] + [E]	[Shift] + [F10] [E]
カラムフォーマットコマンドの使用	なし	[Shift] + [F10] [F]
レジスタを2カラムで表示	[Ctrl] + [2]	[Shift] + [F10] [F] [2]
レジスタを4カラムで表示	[Ctrl] + [4]	[Shift] + [F10] [F] [4]
CodeScope にカラムフォーマットを指定	[Ctrl] + [0]	[Shift] + [F10] [F] [A]
バンク化レジスタの表示	[Ctrl] + [B]	[Shift] + [F10] [S]
フロートレジスタの表示	[Ctrl] + [L]	[Shift] + [F10] [O]
全プロセッサの実行	[Ctrl] + [F9]	[Shift] + [F10] [E] [E] [A]
全プロセッサの停止	なし	[Shift] + [F10] [E] [E] [L]
選択したプロセッサを実行	[F9]	[Shift] + [F10] [E] [E] [R]
選択したプロセッサを指定アドレスまで実行	[Shift] + [F9]	[Shift] + [F10] [E] [E] [S]
選択したプロセッサを停止	[F9]	[Shift] + [F10] [E] [E] [O]
選択したプロセッサをリスタート	[Ctrl] + [Shift] + [R]	[Shift] + [F10] [E] [E] [E]
ブレークポイント操作の使用	なし	[Shift] + [F10] [B]
ブレークポイントの設定	[Ctrl] + [F5]	[Shift] + [F10] [B] [F]
全ブレークポイントのリセット	[Alt] + [F5]	[Shift] + [F10] [B] [L]
全ブレークポイントの有効化	[Ctrl] + [Shift] + [F5]	[Shift] + [F10] [B] [E]
全ブレークポイントの無効化	[Ctrl] + [Alt] + [F5]	[Shift] + [F10] [B] [D]
全ブレークポイントの削除	[Shift] + [F5]	[Shift] + [F10] [B] [M]
リージョンのツールを使用	なし	[Shift] + [F10] [T]
現在のレジスタブロックを保存	なし	[Shift] + [F10] [T] [S]
現在のレジスタブロックを回復	なし	[Shift] + [F10] [T] [R]
アクティブリージョンのプロパティを変更	なし	[Shift] + [F10] [R]

[Log] リージョン

[Log] リージョンでキーボードコマンドを使用する場合

目的	キーボードショートカット	アクセスキー
現在のログウインドウの設定	[Ctrl] + [R]	なし
現在のログウインドウの内容を印刷	[Ctrl] + [P]	なし
現在のログウインドウの内容をファイルに保存	[Ctrl] + [S]	なし
プロセッサ実行操作を使用	なし	[Shift] + [F10] [E]
全プロセッサの実行	[Ctrl] + [F9]	[Shift] + [F10] [E]、 [A]
全プロセッサの停止	なし	[Shift] + [F10] [E]、 [L]
選択したプロセッサを実行	[F9]	[Shift] + [F10] [E]、 [R]
選択したプロセッサを指定アドレスまで実行	[Shift] + [F9]	[Shift] + [F10] [E]、 [S]
選択したプロセッサの停止	[F9]	[Shift] + [F10] [E]、 [O]
選択したプロセッサをリスタート	[Ctrl] + [Shift] + [R]	[Shift] + [F10] [E]、 [E]
ブレークポイント操作の使用	なし	[Shift] + [F10] [B]
ブレークポイントの設定	[Ctrl] + [F5]	[Shift] + [F10] [B]、 [F]
全ブレークポイントのリセット	[Alt] + [F5]	[Shift] + [F10] [B]、 [L]
全ブレークポイントの有効化	[Ctrl] + [Shift] + [F5]	[Shift] + [F10] [B]、 [E]
全ブレークポイントの無効化	[Ctrl] + [Alt] + [F5]	[Shift] + [F10] [B]、 [D]
全ブレークポイントの削除	[Shift] + [F5]	[Shift] + [F10] [B]、 [M]
アクティブリージョンのプロパティを変更	なし	[Shift] + [F10] [R]

[Edit] リージョン

[Edit] リージョンでキーボードコマンドを使用する場合

目的	キーボードショートカット	アクセスキー
新規エディタファイルの作成	なし	[Shift] + [F10] [N]
既存エディタファイルを開く	なし	[Shift] + [F10] [O]
現在のエディタファイルの保存	なし	[Shift] + [F10] [S]
現在のエディタファイルに名前を付けて保存	なし	なし
エディタファイルの現在の選択領域を切り取りクリップボードに貼り付け	[Ctrl] + [X]	[Shift] + [F10] [T]
エディタファイルの現在の選択領域をコピーしクリップボードに貼り付け	[Ctrl] + [C]	[Shift] + [F10] [C]
クリップボードの内容を現在のカーソル位置に挿入	[Ctrl] + [V]	[Shift] + [F10] [P]
新しいタブの値を入力	なし	[Shift] + [F10] [T]
操作を元に戻す	なし	[Shift] + [F10] [U]
文字列を検索	なし	[Shift] + [F10] [F]
現在の選択領域を置換	なし	[Shift] + [F10] [L]
行番号を変更	なし	[Shift] + [F10] [G]
アクティブリージョンのプロパティを変更	なし	[Shift] + [F10] [R]