HighTec 社コンパイラとローターバッハ社デバッガチュートリアル

このチュートリアルでは、HighTec 社統合開発環境を用いてプロジェクトのコンパイル・リンクを行って実行形式ファイルを生成することと、Lauterbach 社 TRACE32 の最も重要なデバッグ機能やデバッガの使用に役立つ多くのヒントとコツを紹介します。

第一部 HighTec IDE チュートリアルでは、C 言語で作成されたプログラムを TriCore 用の HighTec Development Platform でビルドすることや、IDE(Eclipse)の初歩的な使用方法について説明します。

第二部 TRACE32 デバッガチュートリアルでは、第一部で生成した実行形式ファイルを用いて主要なデバッグ機能を説明します。TRACE32 シミュレータ(TRACE32 TriCore Instruction Set Simulator)を使用することで、ターゲットハードウェアが無くても動作を確認することができます。

＊第一部をパスして、第二部に直接進むことも可能です。

・TRACE32 シミュレータ評価版は、以下のリンクからダウンロードできます。ダウンロードした zip ファイル (simtc.zip) を任意のフォルダに解凍してください。TRACE32 シミュレータをインストールする必要はありません。

https://www.lauterbach.com/frames.html?download_demo.html

・HighTec Development Platform 評価版は、以下のリンクからダウンロードできます。

https://hightec-rt.com/en/downloads/evaluation-version.html

こちらから TriCore/AURIX Development Platform (C/C++ compiler)を選択ください。

各社ツールの詳細な説明と使用法につきましては、それぞれ各社からの説明文書を併せてご参照ください。

目次

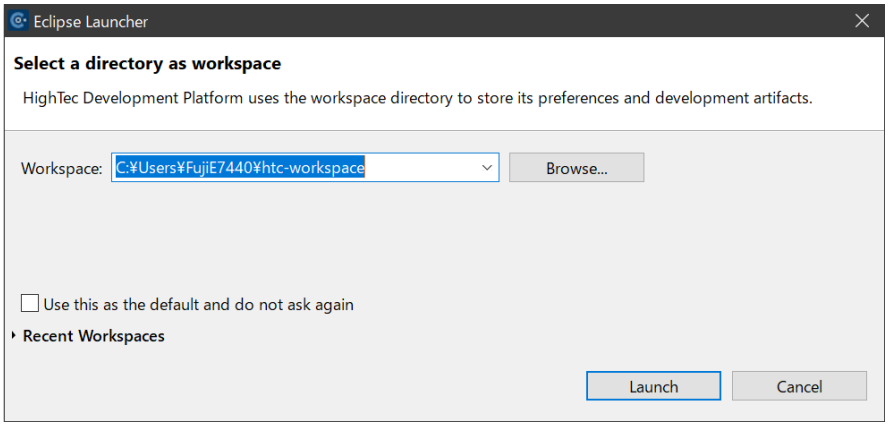
第一部　HighTec IDE チュートリアル

この HighTec IDE チュートリアル[1]では、IDE（統合開発環境）を用いてプログラムを作成し、コンパイル・リンクを行って実行形式ファイルを生成します。IDE の起動・終了、プロジェクト[2]の作成、ビルドの順に説明します。

HighTec IDE（Eclipse）の起動

HighTec IDE は、HighTec Development Platform をインストールすると自動的にインストールされます。

1. Windows のスタートメニューから TriCore Development Platform ～ > HighTec IDE を選択するか、あるいはデスクトップにある HighTec IDE のアイコン  をダブルクリックして IDE（Eclipse）を起動します
2. Eclipse Launcher ダイアログが表示されます。ここでプロジェクトのリソースや出力ファイルを格納するワークスペース[3]を指定します



3. ワークスペースのディレクトリを例えば`C:¥Users¥FujiE7440¥htc-workspace`などのように指定します。そのディレクトリが存在しない場合には新しく作成され、既にある場合はそれが使用されます

---

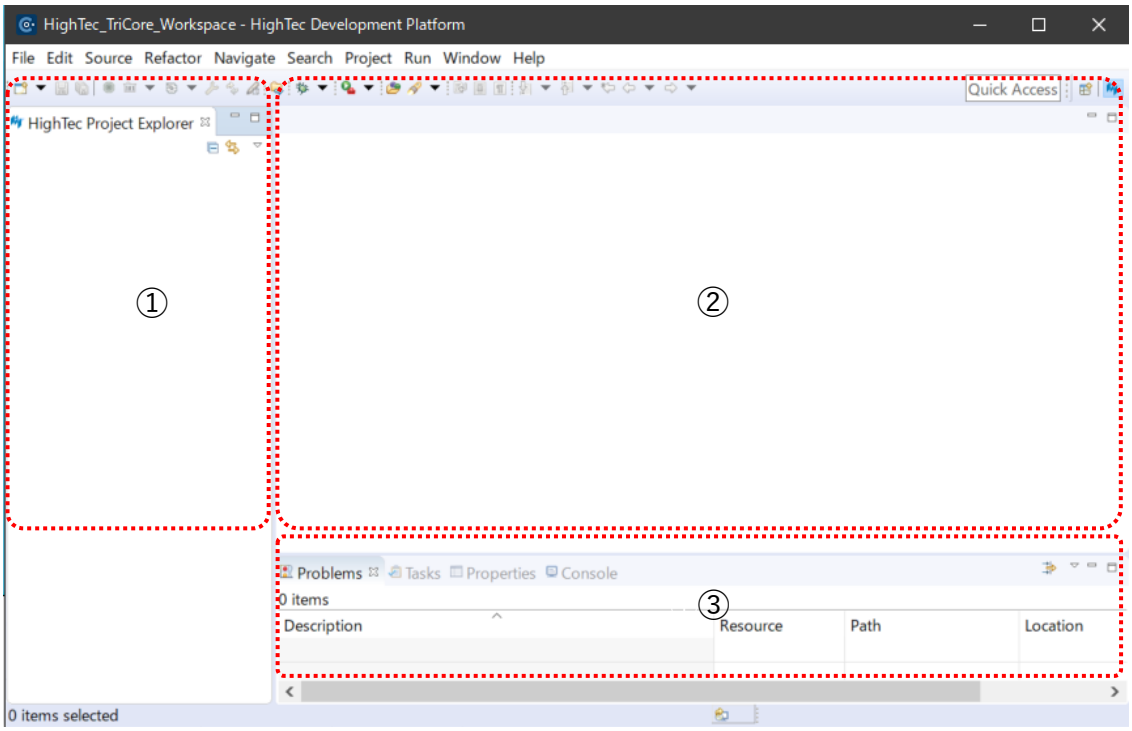
[1] 本チュートリアルは、TriCore/AURIX Development Platform パッケージ内にある HighTec IDE Quick Start Guide を基にした日本語版の説明です。

[2] 一つのプログラムの作成には、ソースコードやヘッダファイル、リンカスクリプト、設定ファイルなど、複数のファイルが必要になります。IDE では、これらをまとめて「プロジェクト」として扱い、一括して管理できるようにしています。

[3] 作成したプログラムなどのリソースは専用のフォルダ（ディレクトリ）に格納されます。このフォルダのことを「ワークスペース」と呼びます。一つのワークスペースには複数のプロジェクトを置くことができます。

4. Use this as the default... をチェックすることで、次回以降このダイアログを出さないようにできます。その場合、最後に使用したワークスペースが使用されます
   ワークスペースを変更するには File メニューの Switch Workspace から選択します
5. Launch で次に進みます

下図のような HighTec IDE のメイン画面が表示されるか、最後に保存されたワークベンチ[4]のレイアウトが表示されます[5]。



① Project Explorer は、ワークスペースで現在有効な HighTec IDE プロジェクトを保持し、ここからソースファイル編集やビルド操作などのアクションを行います。

② エディタビューはソースを編集する場所で、Project Explorer で選択してここにタブ付けしてオープンされます。

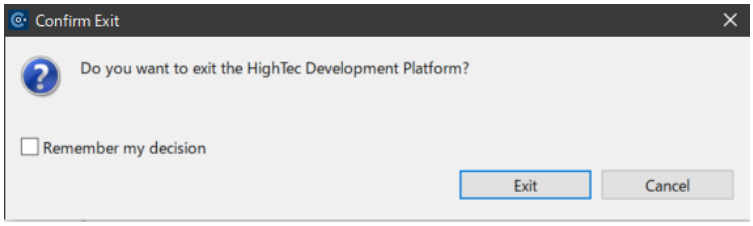
③ エラーやワーニングメッセージを出力するプロブレムビュー（Problems)、選択されたファイルの一般情報を表示するプロパティビュー（Properties)、ビルドなどプロセスからの出力が表示されるコンソールビュー（Console）などがタブ付けされて表示されます。

[4] エディタやビュー、次で説明する「パースペクティブ」から構成される画面全体（ウィンドウ）のことを「ワークベンチ」と呼びます。

[5] 最初の起動時や新たにワークスペースを作成した場合には「Welcome view」が表示されます。タブのクローズボタンで閉じてください。

IDE の終了

IDE を終了するには、File メニューから Exit を選択します。次のダイアログが表示された場合、終了するには Exit を、終了を取り消すには Cancel をクリックします。

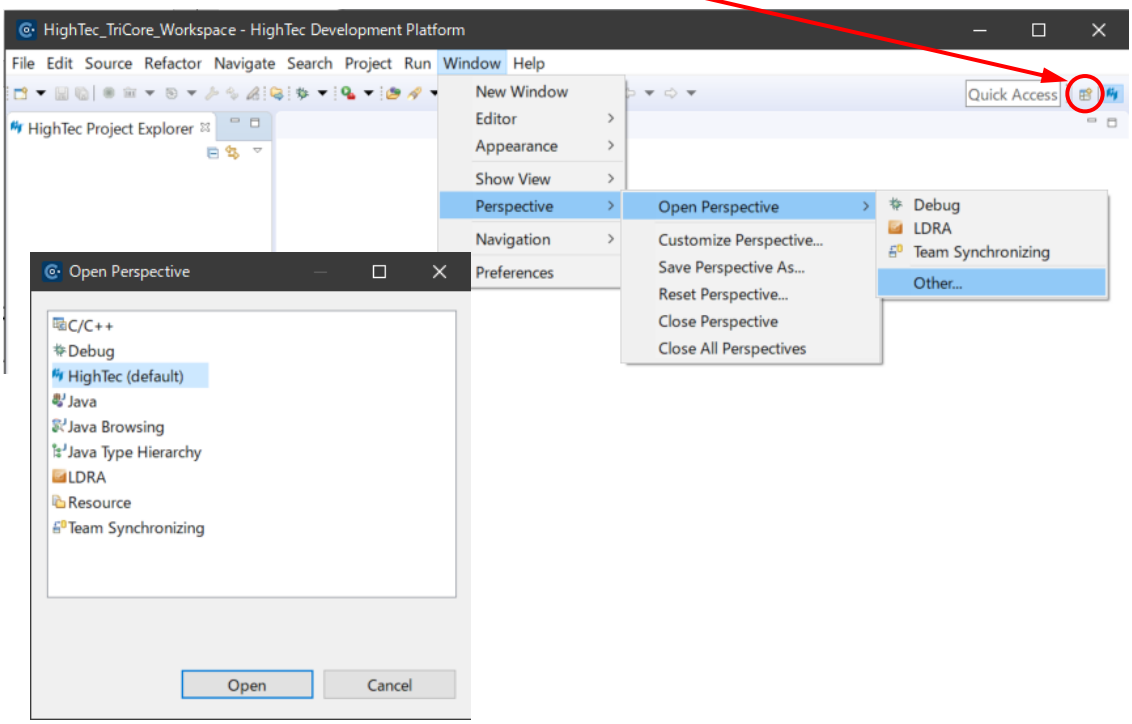


終了時にワークベンチのレイアウトが保存され、それが次回の開始時に使用されます。

プロジェクトの作成

プロジェクトを作成する前に、ワークベンチを HighTec パースペクティブ[6]にすることが必要です。Eclipse の開始時にデフォルトでそうなっているはずですが、もしそうでないならば、次のようにして HighTec パースペクティブを開いてください。

Window メニューから Perspective > Open Perspective > Other... と選択して Open Perspective ダイアログで HighTec を選択します。あるいはアイコンをクリックしてこのダイアログを開きます。



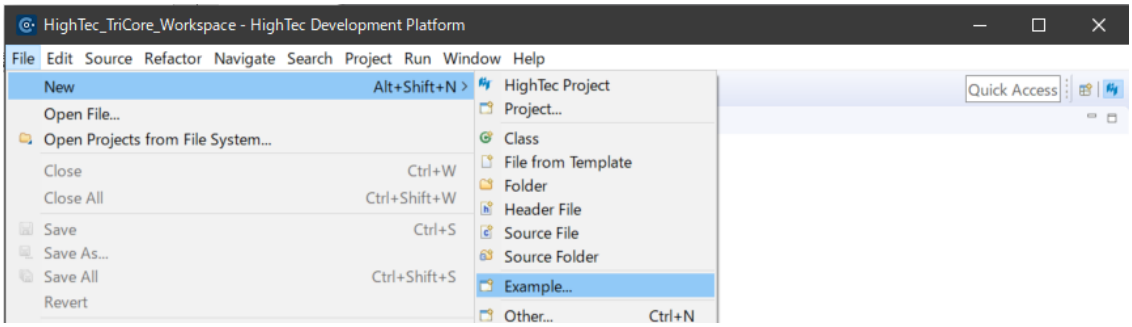
6 Eclipse では作業で必要な機能を選んでワークベンチに表示するようになっています。どの機能を表示するかを選び、どのように配置するかを決めたエディタやビューの画面構成を「パースペクティブ」と呼びます。

新規のプロジェクトは「既存のプロジェクトをインポートして流用する」、あるいは「プロジェクトを空の状態から作成する」ことで作成します。本チュートリアルは、HighTec Development Platform 内にある既存のプロジェクトをインポートして流用する方法を実施します。

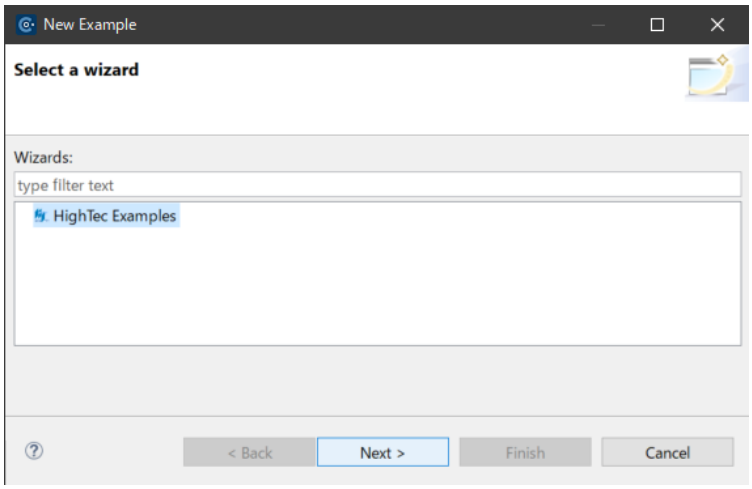
＊プロジェクトを空の状態から作成する方法は、本編末尾の補足 3 を参照してください。

Step1：既存のプロジェクト(HighTec Examples)をインポートする

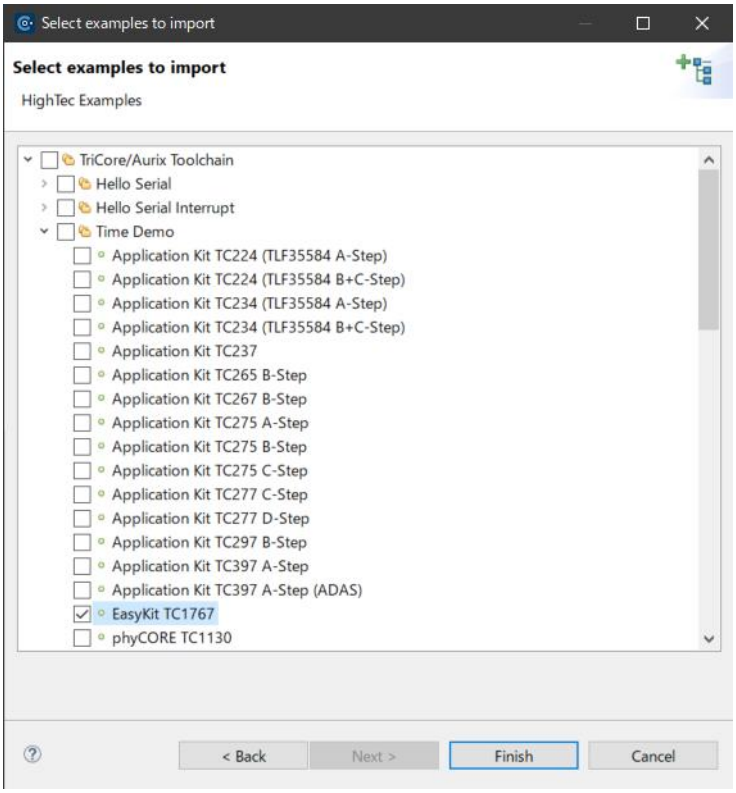
1. File メニューから New > Example... を選択します



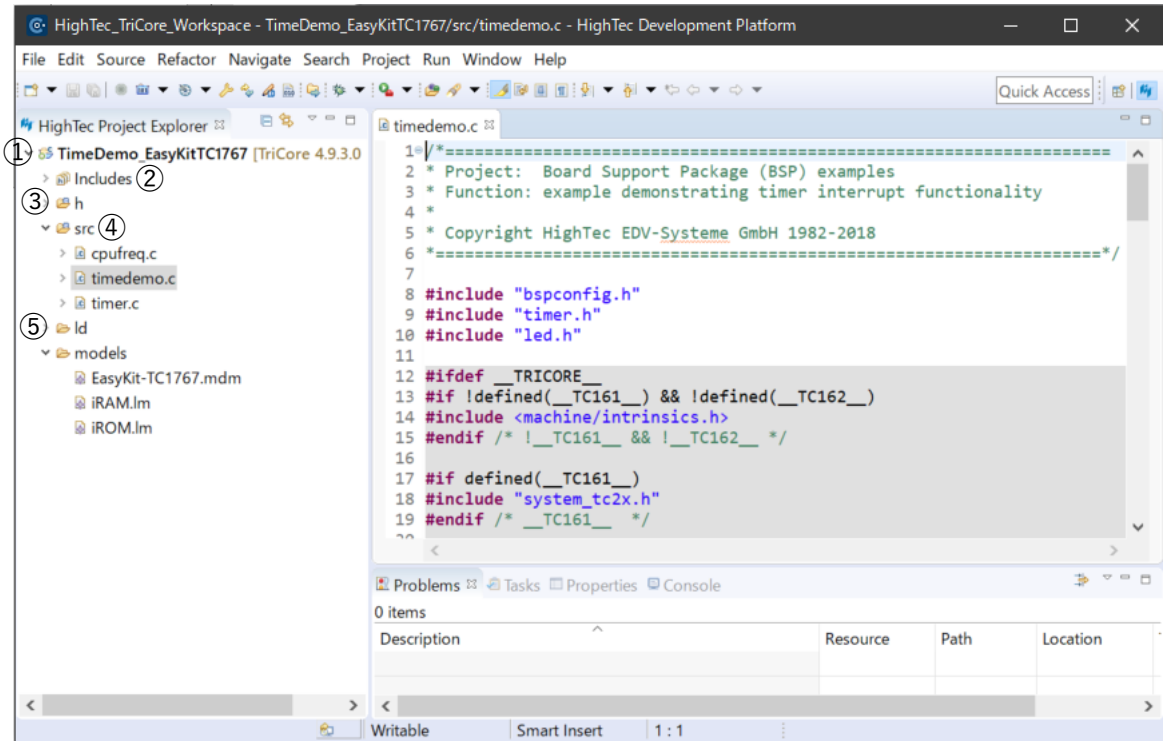
New Example ダイアログで HighTec Examples を選択し、Next > をクリックします。



2. Select examples to import ダイアログに示される、このツールチェーンがサポートする TriCore ボードの種類から必要なもの(例えば Time Demo 以下)を選択します
   ここでは EasyKit TC1767 を選んでいます



Finish をクリックするとダイアログが終了して、プロジェクトがインポートされます。



上図は、インポートされたプロジェクトと HighTec パースペクティブです。

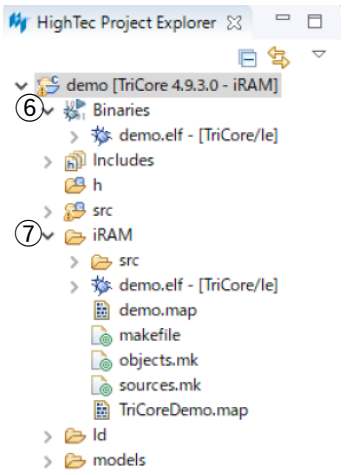
Project Explorer に表示されるフォルダは以下の意味をもっています。

番号	内容
①	プロジェクト名
②	設定されたインクルードパス用の仮想フォルダ[7]
③	プロジェクトのヘッダファイル
④	プロジェクトのソースファイル
⑤	プロジェクトのリンカファイル

プロジェクトのファイルを見るには、Project Explorer でプロジェクトの構造を展開することが必要になることがあります。生成されたファイルをオープンするには、Project Explorer で src フォルダ内のファイルをダブルクリックします。(エディタビューに表示されます)

＊後の作業でプロジェクトをビルドすることで、次のフォルダも生成されます

番号	内容
⑥	出力されるバイナリファイルの仮想フォルダ
⑦	現在有効なビルドコンフィグレーション(後述)の名前の出力フォルダで、ビルドプロセスで生成されるすべてのファイル（*.o や*.map など)が含まれます



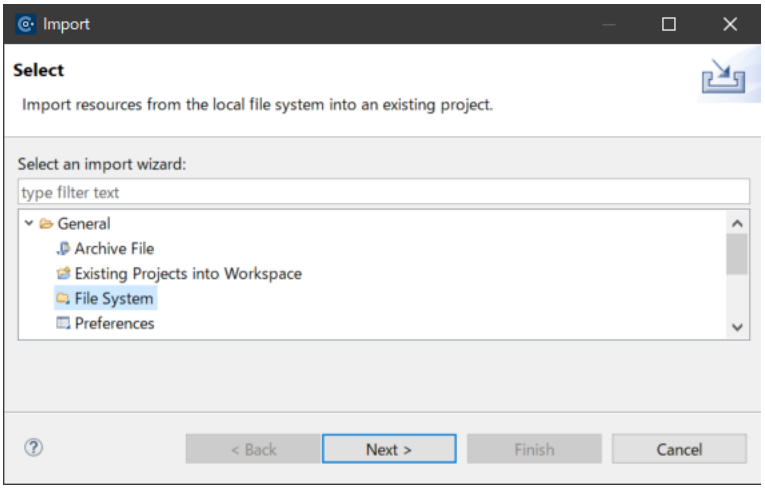
Step2：インポートした例題プロジェクトを修正する

次に、このインポートしたプロジェクトを修正して、新しいプロジェクトを作成します。そうすることで、既存プロジェクトと共通のファイルや設定情報を流用できます。

1. 「src」フォルダと「h」フォルダから全てのファイルを削除します(ファイルを選択して Delete やメニューから行います)。そして、以下の手順でソースファイルをインポートします。

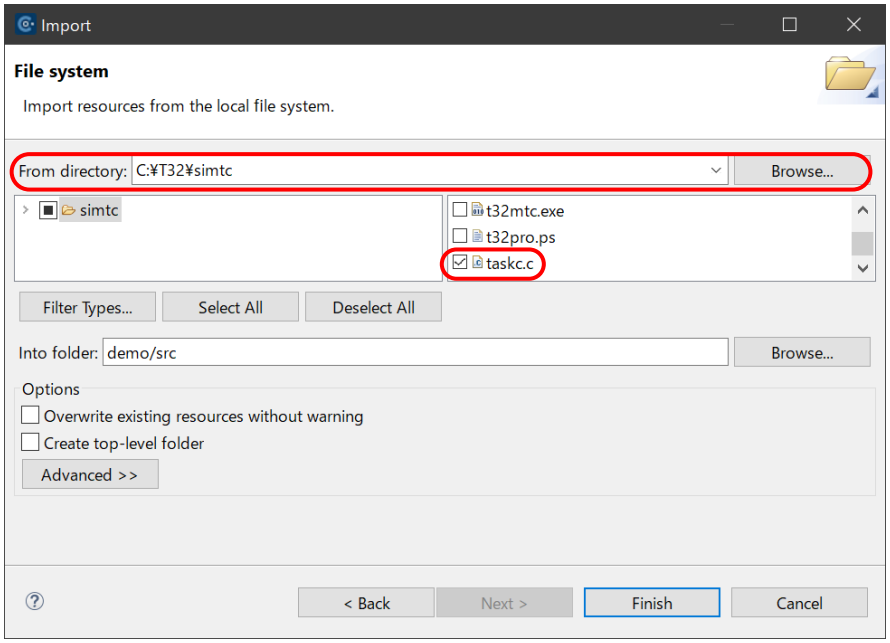
2. Project Explorer ビューからプロジェクト内の「src」をクリックして、コンテキストメニューから Import... を選択する。(File メニューの Import... からも選択可能です)

[7] 仮想フォルダは実際の保存場所に関係なく、ワークスペースのツリー構造にまとめて管理するフォルダです。



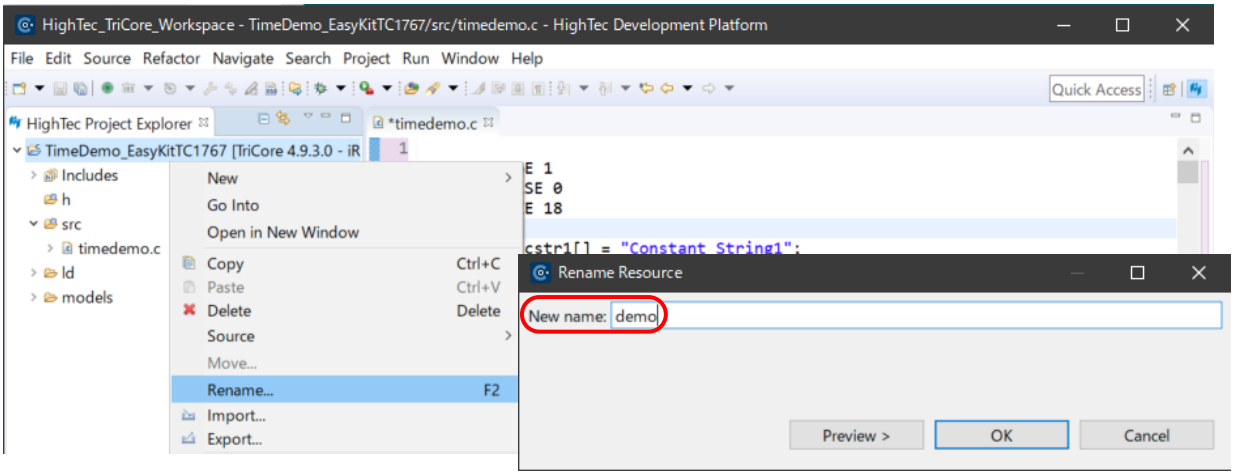
Import ダイアログの Select で General から File System を選択し、Next > をクリックします。

3. ソースファイルが存在するフォルダ（Lauterbach 社の評価版を解凍してできる simtc）を入力するかブラウズして指定し、一覧から `taskc.c` をチェックして Finish します。



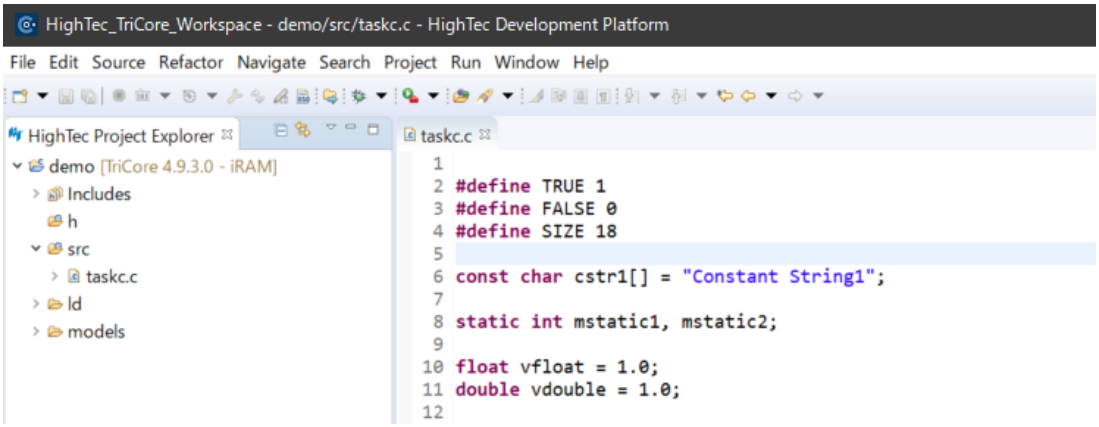
Step3：プロジェクト名称を変更する

プロジェクト名を右クリックして Rename... を選択し、Rename Resource ダイアログの New name に新しいプロジェクト名(demo)を入れて OK します。



(File メニューの Rename... からも選択可能です)

完成したプロジェクトは、下図のようになります。



プロジェクトのビルド

プロジェクトをビルドすると、HighTec TriCore コンパイラとアセンブラ、リンカが使用されてすべてのソースコードとプロジェクトに関係するライブラリがコンパイル・リンクされます。

ウィザードによって iRAM や iROM のようなビルドコンフィグレーション[8]をそれぞれ生成します。本例の場合、コードは iRAM(内蔵 RAM)に配置されます。

ビルド中に出力されるメッセージは Console ウィンドウに表示されます。ビルドプロセスはエラーやワーニングを出さずに終了すべきです。

ビルドには次の三つの型があります。

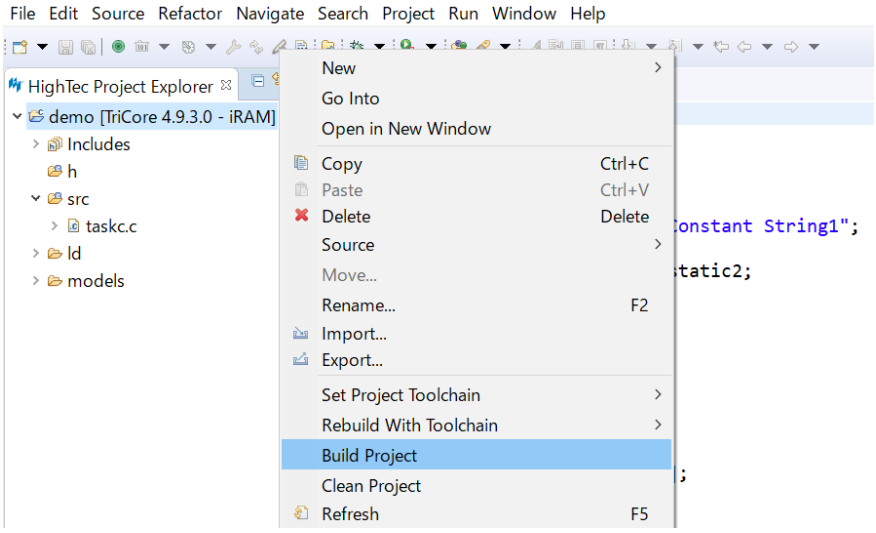
Build　選択したプロジェクトをインクリメンタルにビルドする
最後のビルド以降変更されていないファイルはコンパイルをスキップされ、ビルドが高速になります。

Clean　以前のビルドで生成された中間ファイルを削除する

Rebuild　アクティブなプロジェクトを再ビルドする
プロジェクト内のファイルは最後のビルド以降に変更があるなしにかかわらず、すべてビルドの対象になります。Rebuild は Clean の後に Build を行うものです。

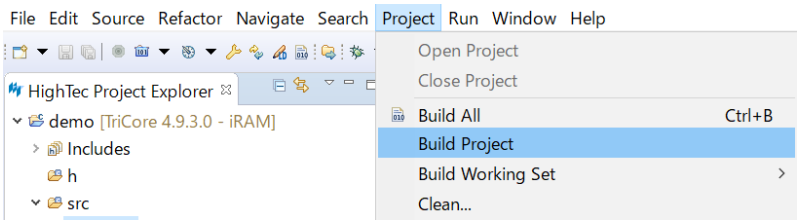
以下 3 種類のいずれかの方法でプロジェクトをビルドしましょう。

方法 1　Project Explore でプロジェクト名を右クリックし、Build Project を選択する

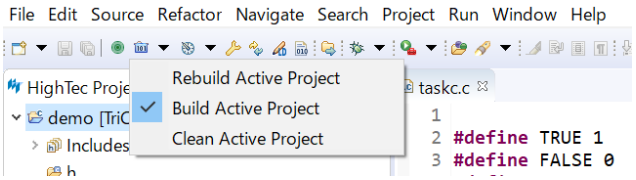


方法 2　Project メニューから Build Project を選択する

[8] ビルドコンフィグレーションはプロジェクトの「バリアント」に対応するもので、プロジェクトを複製することなく、RAM や ROM などのターゲット構成に向けたビルドを行うものです。後述するように、プロジェクトを右クリックして Build Configuration > Manage... から管理できます。

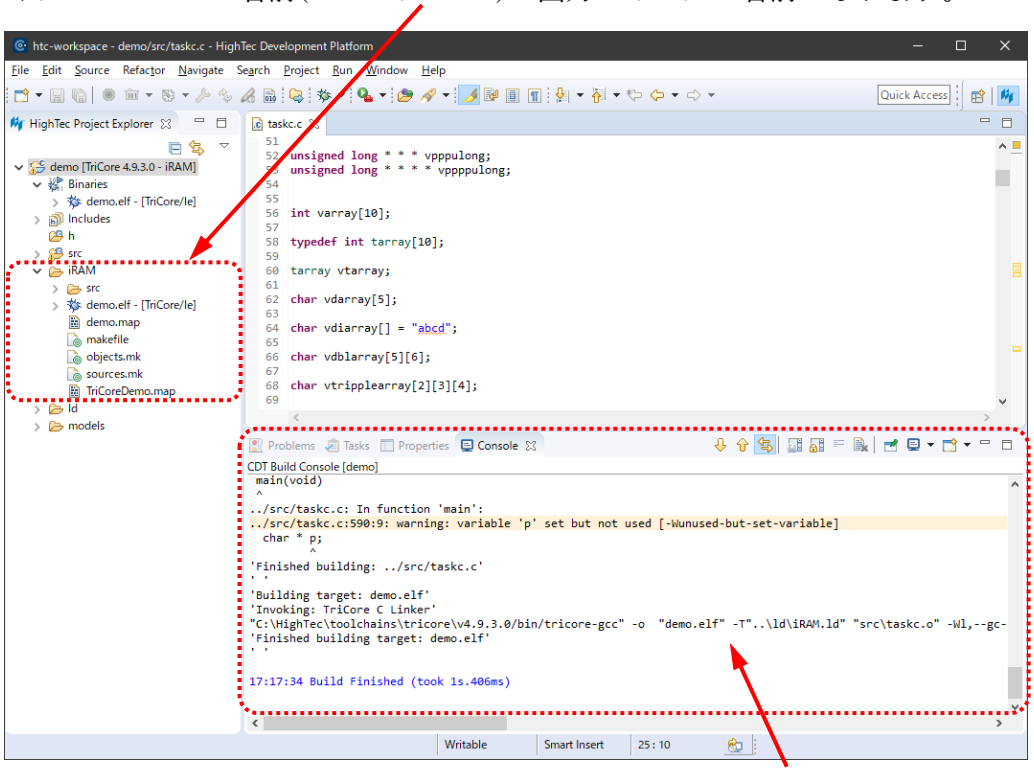


方法 3　ツールバーの　(Run/Set Active Project Build Action)ボタンをクリックし、Build Active Project を選択する



ビルドに成功すると、結果は以下のようになります。

コンフィグレーションの名前(ここでは iRAM)が出力フォルダの名前になります。



コンパイル・リンクでのコンソール出力メッセージ

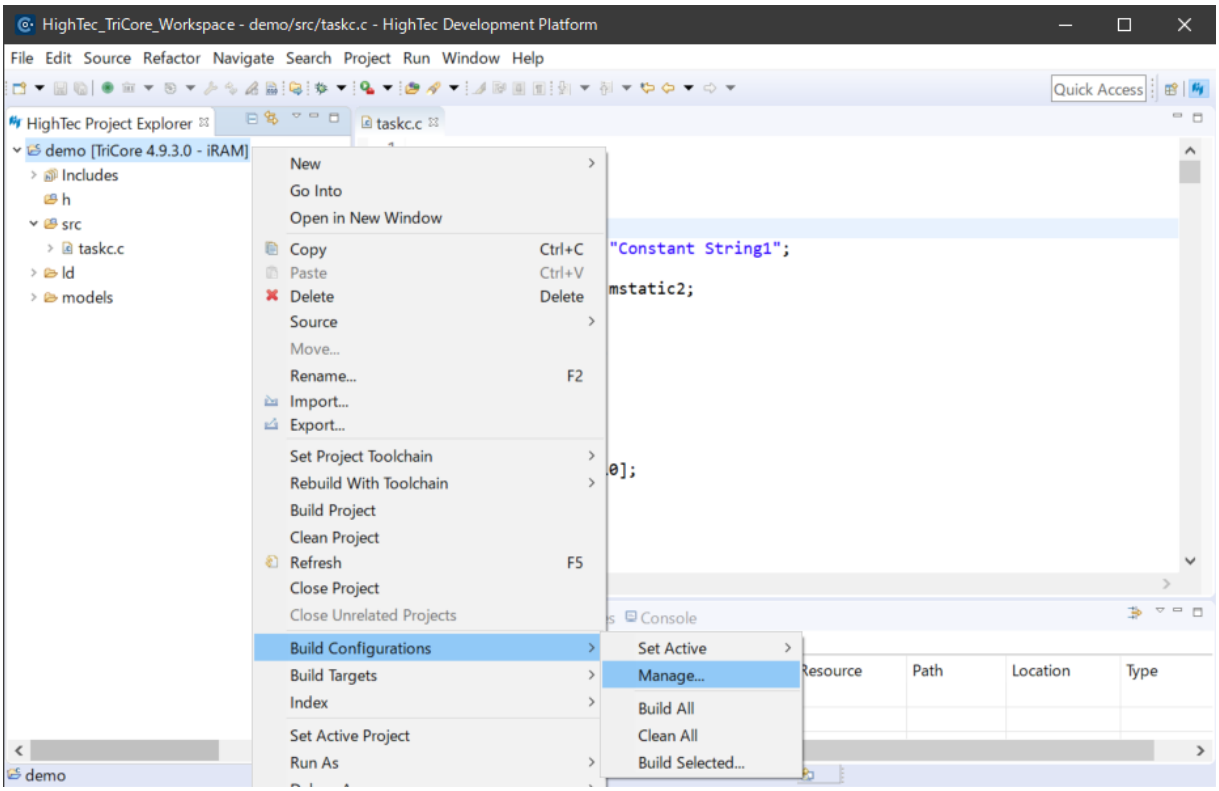
ビルドに成功して実行形式ファイルが生成されたら、「**第二部　TRACE32 デバッガチュートリアル**」に進みます。

そのために、ワークスペースの demo フォルダに生成された demo.elf(iRAM¥demo.elf)を、Lauterbach 社の評価版を解凍してできる simtc フォルダに上書きコピーして使用します。(simtc 内の demo.elf は事前に名称変更して保存してください)

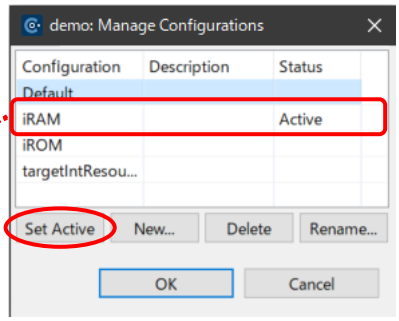
補足１：ビルドコンフィグレーションの設定

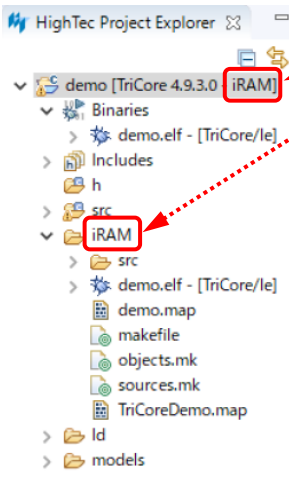
異なるオプションをもつ複数のビルドコンフィグレーションを設定することができ、それぞれ別の出力フォルダが生成されます。

ビルドコンフィグレーションは、プロジェクトを右クリックしたコンテキストメニューから Build Configuration > Manage... を選択し、Manage Configuration ダイアログで設定できます。



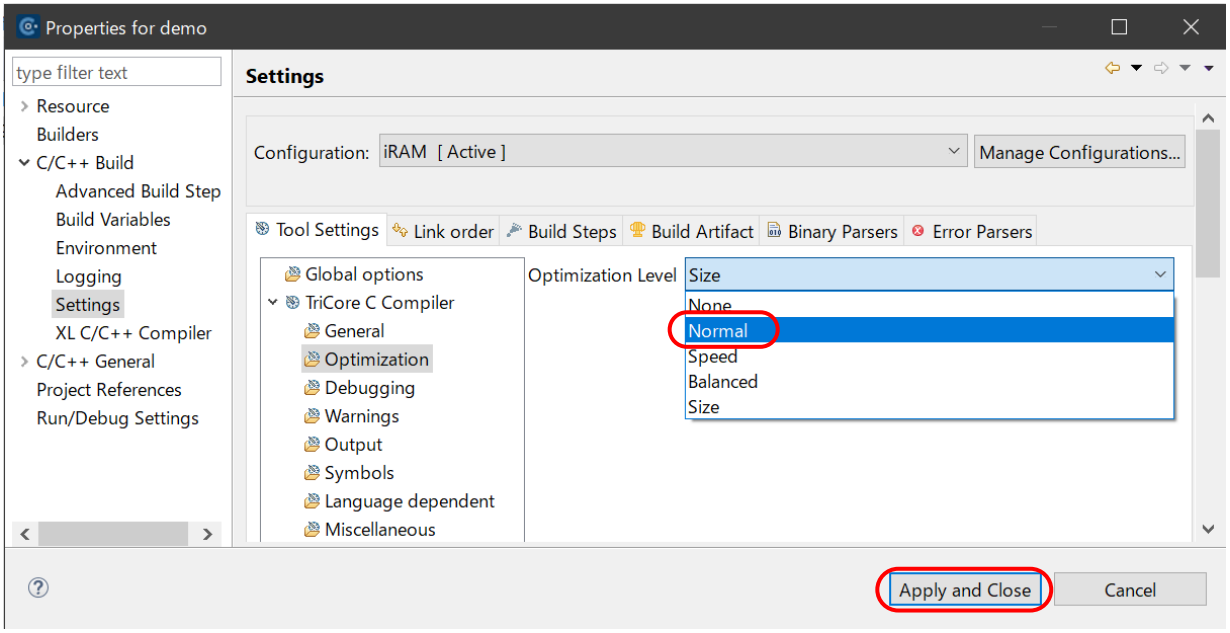
常に一つのコンフィグレーションがアクティブであり、Manage Configurations ダイアログでコンフィグレーションを選択して Set Active をクリックすることで変更できます。





補足２：プロパティの設定

上記の手順でのプロジェクト作成によって、ほとんどのプロパティ[9]が自動的に設定されます。また、Project Explore でプロジェクト名を右クリックし、メニューから Properties を選択して開くダイアログから Settings ページなどを操作することで、他に必要な設定を行うことができます。



本チュートリアルのプログラムは、インポートしたプロジェクトから引き継いだ設定の変更なしで動作しますが、ここでは設定変更の例として、最適化のレベルを Size[10]から Normal[11]に変更します。上図において、Optimization Level のリストで Normal を選択し、Size から変更して Apply and Close します。

プロパティ設定の詳細については、ダウンロードパッケージにある HighTec IDE Quick Start Guide をご覧ください。

[9] ターゲットに合わせてプロジェクトをビルドするためのパスやオプションなど各種設定情報

[10] コードサイズを小さくするように最適化を行います。(コマンドラインオプションでは -Os になります)

[11] 実行時間を小さくする基本的な最適化を行います。(コマンドラインオプションでは -O1 になります)

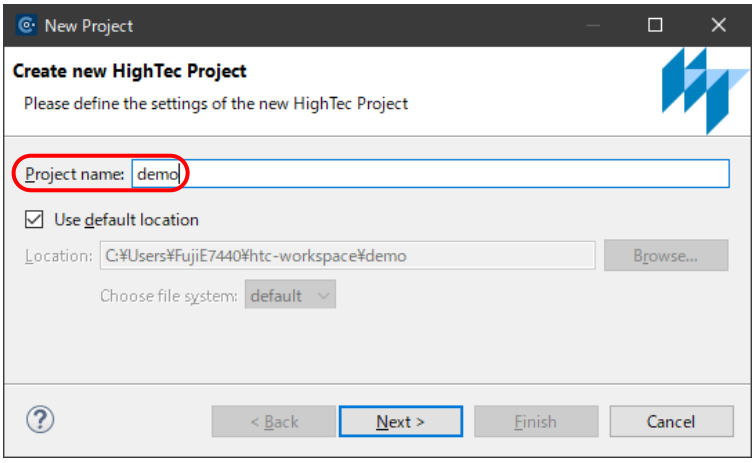
補足３：プロジェクトを空の状態から作成する方法

空の状態からターゲットのアーキテクチャや各種の設定情報を入力します。

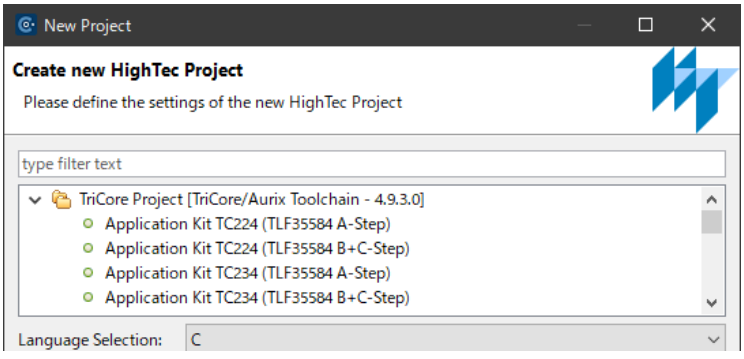
HighTec Project のウィザードに従って処理を進めます。

1. File メニューから New > HighTec Project を選択する（Project Explorer のビュー内で右クリックして New > HighTec Project を選択することでもできます）
   New Project ダイアログが開きます。

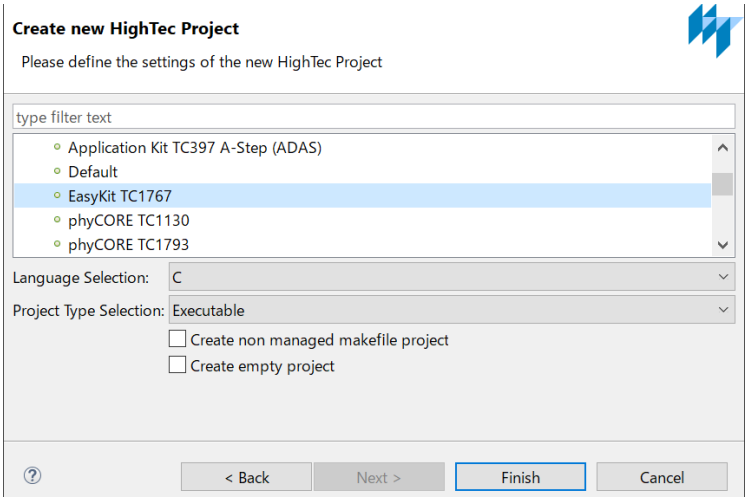
2. Project name を入力する
   ワークスペースフォルダに保存するには、Use default location をチェックしたままにしてください。



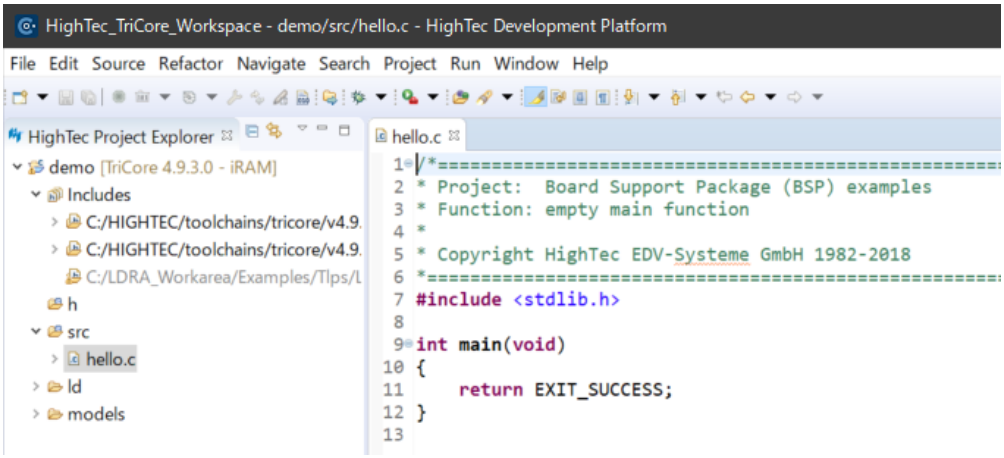
Next > でコンフィグレーションに進みます。



3. リストからアーキテクチャを選択し、Finish で終了する
   本例では、EasyKit TC1767 を選択します。



4. 以下のように新規プロジェクトが作成されます。



`hello.c` が仮のプログラムとして生成されています。

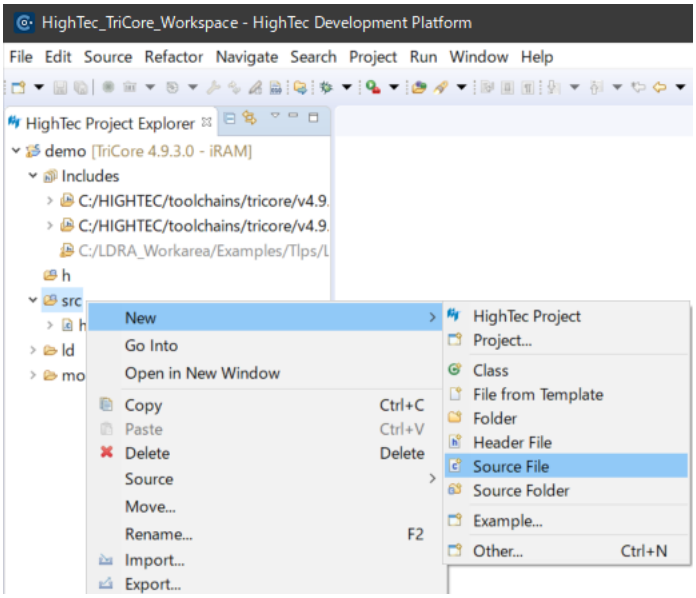
これを必要なファイルで上書きするか、以下の手順で必要なファイルを追加で作成し、`hello.c` を削除します。

本例の demo プロジェクトは `taskc.c` ソースファイル 1 つだけですが、他のソースファイルやヘッダファイルなどを追加することもあります。

以下の手順で、ソース/ヘッダファイルを新規作成します。

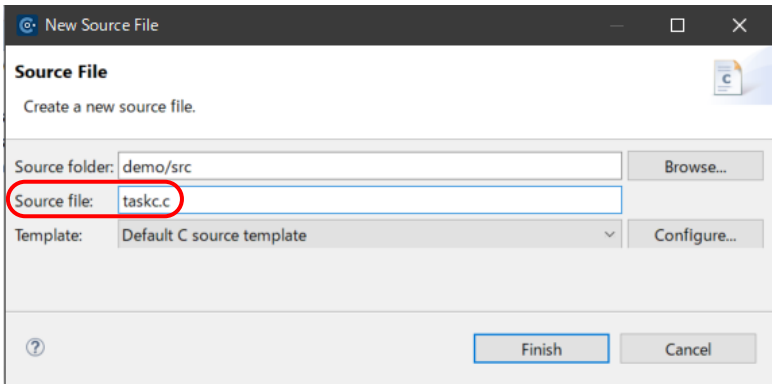
1. Project Explorer ビューからプロジェクト demo の src を右クリックし、コンテキストメニューから New > Source File と選択する。(File メニューの New からも選択可能です)

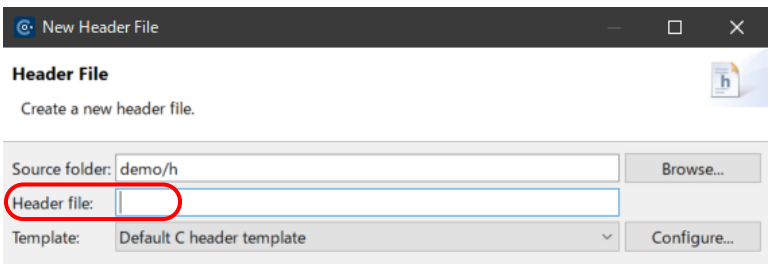
   ヘッダファイルを作成する場合は、プロジェクト demo の h を選択して、コンテキストメニューから New > Header File を選択します。



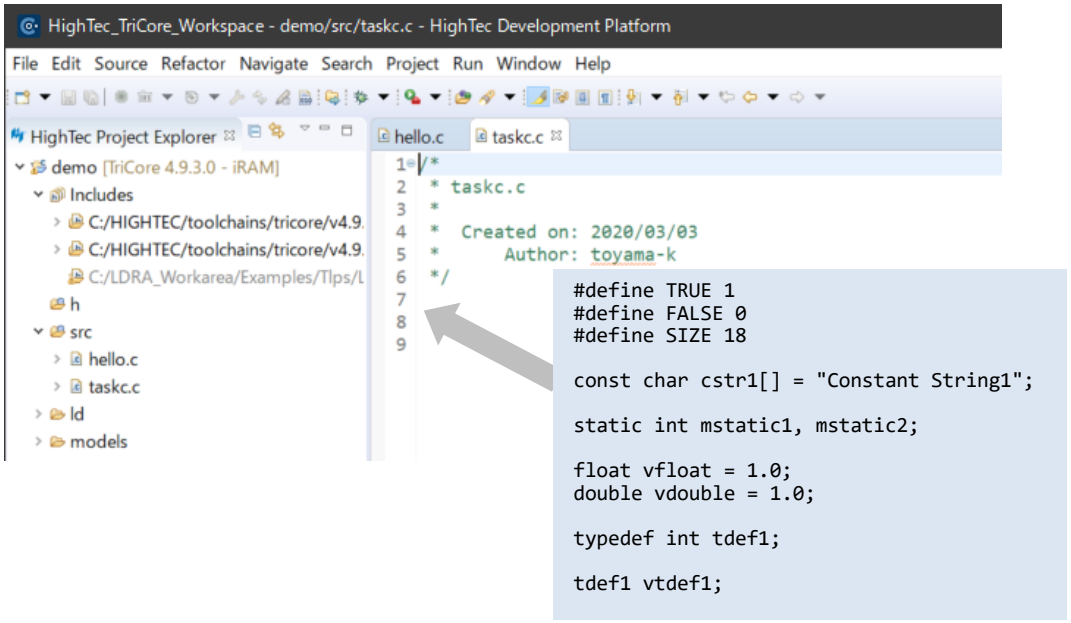
New Source File（または New Header File）ダイアログボックスが開きます。

2. Source file（Header file）に作成するソースファイル名（ヘッダファイル名）を入力し、Finish をクリックする





選択または指定したフォルダ内に指定名称のファイルが生成されます。また、エディタのタブに作成したファイルの名称が表示され、テキストの入力が可能な状態になります。



ここでは、新規作成した `taskc.c` の領域内に提供されているデモプログラム `taskc.c` の内容をコピー・ペーストし、`hello.c` を削除（ファイルの削除は、単に選択して Delete やメニューから行います）することで、プロジェクトを作成しています。

第二部　TRACE32 デバッガチュートリアル

この TRACE32 デバッガチュートリアル[12]では、C 言語の簡単な例題プログラムを用いて最も重要なデバッグ機能を説明すると同時に、デバッガの使用に役立つ多くのヒントとコツを紹介します。

所要時間は 0.5～1 時間で、以下の章を順番に作業します。

より詳しい情報は、以下のページで公開される「日本語トレーニングマニュアル」をご参考ください。

https://repo.lauterbach.com/manual.html#_H3

＜第一部で生成したオブジェクトコードを使用する場合＞

HighTec 社の Development Platform でビルドしたプログラム(`taskc.c`)をデバッグします。

＊HighTec ワークスペースの demo フォルダに生成された demo.elf (iRAM¥demo.elf) を Lauterbach 社の評価版を解凍してできる simtc フォルダに上書きコピーします。

＜第二部のみ実施する場合＞

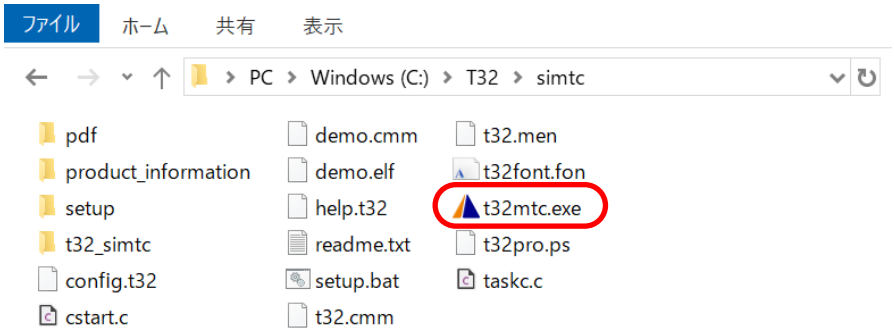
評価版の TRACE32 シミュレータをダウンロードして、その中にあるサンプルプロジェクトを使用します。

＊以下のチュートリアルでは、HighTec 社の TriCore 用開発環境を使用してサンプルプロジェクトと同じソースプログラからビルドした実行形式をロードしてシミュレータで実行しています。
TRACE32 シミュレータ評価版のサンプルプロジェクトでも同等の動作を確認いただけますが、プロセッサの型式や環境の違いなどにより、表示される項目や内容になどに多少の違いがあります。

[12] 本チュートリアルは、評価版 TRACE32 シミュレータのパッケージ内にある説明書 demo.pdf を基にした日本語版の説明です。

TRACE32 シミュレータの起動

1. ダウンロードした zip ファイル（simtc.zip）を任意のフォルダに解凍します。TRACE32 シミュレータをインストールする必要はありません。
2. t32m<アーキテクチャ>.exe ファイル（ここでは t32mtc.exe）をダブルクリックしてデバッグセッションを開始します。

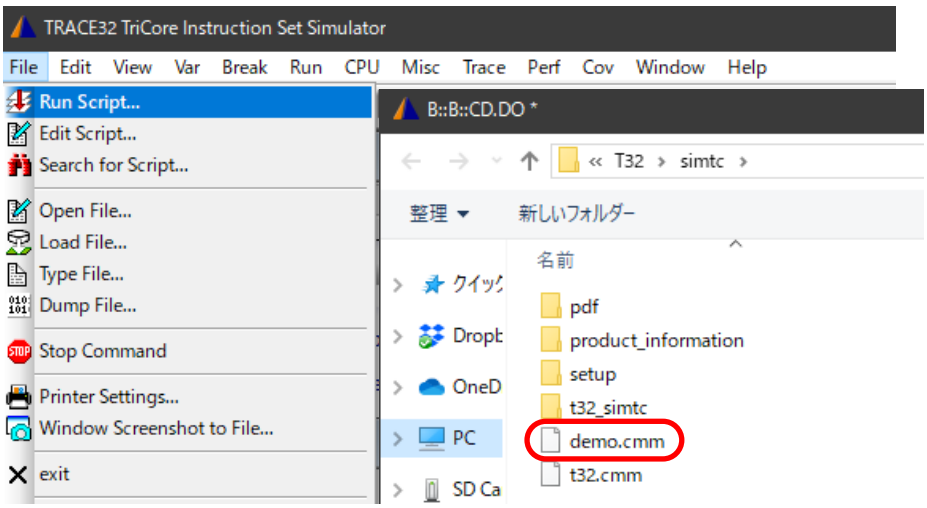


simtc.zip の内容

TRACE32 命令セットシミュレータが起動すると、デバッグセッションを設定する起動 PRACTICE スクリプトが自動的に実行されます。

以下のように File メニューから Run Script...を選択して、同じ起動 PRACTICE スクリプト（demo.cmm）を手動で実行することもできます。

PRACTICE は Lauterbach のスクリプト言語で、テストの自動化、TRACE32 GUI の設定、およびデバッグ環境の設定に使用されます。



このデモデバッグセッションでは、PRACTICE スクリプト demo.cmm がアプリケーションプログラムの実行形式 demo.elf をロードし、ロードされた情報から TRACE32 内部シンボルデータベースを生成します。

```
; Script file for TriCore TC17x7
;
; Load the Sieve-Demo into memory.
;
; $LastChangedDate: 2009-04-01 13:26:12 +0200 (Mi, 01 Apr 2009) $
; $LastChangedRevision: 2930 $
; $LastChangedBy: sphilipp $

;=============================================================
; initialize and start the debugger

   RESet
  SYStem.CPU tc1736
  SYStem.Up

;=================================
; load demo program
  Data.LOAD.ELF demo.elf
  Go main

; open some windows
  WinCLEAR
  WinPOS 0% 0% 100% 50%
  Data.List
  WinPOS 0% 50% 50% 50%
  Var.Frame /Locals /Caller
  WinPOS 50% 50% 50% 50%
  Var.Watch
  Var.AddWatch ast flags

ENDDO
```

demo.cmm

```
; TRACE32 Simulator Demos for Infineon TriCore CPUs.

; ================================================

  WinCLEAR
  AREA.RESet

  do demo.cmm

  HELP.FILTER.RESet
  HELP.FILTER.ADD    demo
  HELP.FILTER ON

  ENDDO
```

t32.cmm

TRACE32 は起動時にデフォルトで t32.cmm スクリプトを実行します。本例では、t32.cmm から demo.cmm を呼び出すようになっています。

ユーザーインターフェース – TRACE32 PowerView

TRACE32 のグラフィカルユーザーインターフェース(GUI)は、TRACE32 PowerView と呼ばれます。下のスクリーンショットを参照してください。例として Data.List コマンドと Data.List ウィンドウを使用して GUI について簡単に説明します。

プログラムのリストとプログラムカウンタ

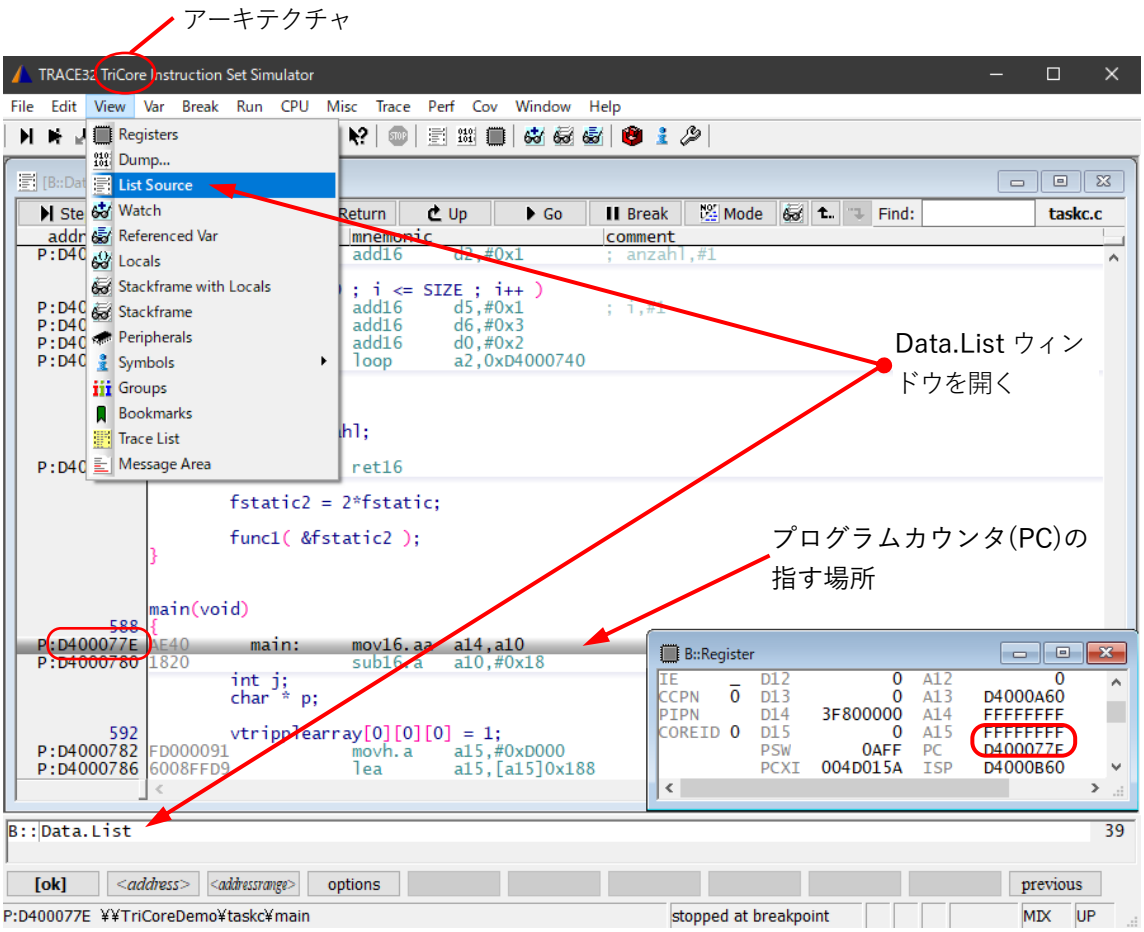
以下のいずれかの操作を行って、Data.List ウィンドウを開きます。[13]

- View メニューから List Source を選択する
- TRACE32 コマンドラインで Data.List（または D.L または List）と入力する

Data.List(= List)ウィンドウに、アセンブラニーモニックと HLL[14]のコードが表示されます。

[13] 図では Data.List ウィンドウと Register ウィンドウが表示されています。

[14] ソースプログラムを記述している C 言語など高水準言語（High-level Language）



Data.List ウィンドウで、灰色のバーはプログラムカウンタ(PC)の位置を示します。ここで現在、それはラベル main のシンボリックアドレスにあります。これは、プログラムカウンタ(PC)が、起動時に実行された PRACTICE スクリプト(demo.cmm)で次のように「Go main」と指示されているためです。

```
;===================================================================
; load demo program
  Data.LOAD.ELF demo.elf
  Go main
```

TRACE32 コマンド

TRACE32 PowerView では、以下のような様々な方法でコマンドを実行できます。

1. メニューバーのメニュー
2. メインツールバーのボタンと TRACE32 ウィンドウのツールバーのボタン
3. TRACE32 ウィンドウのコンテキストメニュー

さらに TRACE32 コマンドラインとその下部にあるソフトキーを使用してコマンドを実行できます。

TRACE32 コマンドラインとソフトキー

TRACE32 コマンドでは大文字と小文字が区別されません。register.view は Register.view と同じです。コマンド説明で表示されている長形式中の大文字は、そのコマンドの短形式を示すもので、省略できません(小文字で書くことはできます)。小文字はすべて省略できます。これにより、コマンドラインから頻繁に使用されるコマンドを入力するとき、短縮形が効率的な時間節約になります。

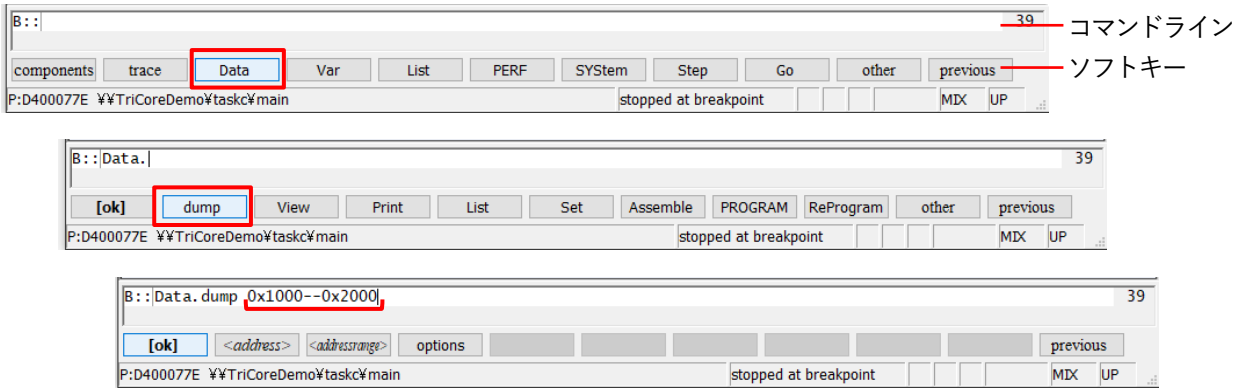
例

- 長形式の Register.view の代わりに短形式の r または R を入力する
- 長形式の Data.List の代わりに短形式の d.l または D.L を入力する[15]

ソフトキーはコマンドライン領域の下部にあります。ソフトキーのキャメルケース(大文字と小文字が混在する表示)は、長形式のコマンドを示しています。ソフトキーはコマンド入力をガイドし、すべての可能なコマンドとパラメータを表示します。

例 - ソフトキーを使用して Data.dump コマンドを作成する

1. Data をクリックする
2. dump をクリックする
3. ダンプするアドレス範囲(<*addressrange*>)またはアドレス(<*address*>)を入力する
   アドレス範囲の場合、例えば 0x1000--0x2000
4. **[ok]**をクリックしてコマンドを実行する
   Data.dump ウィンドウが開きます



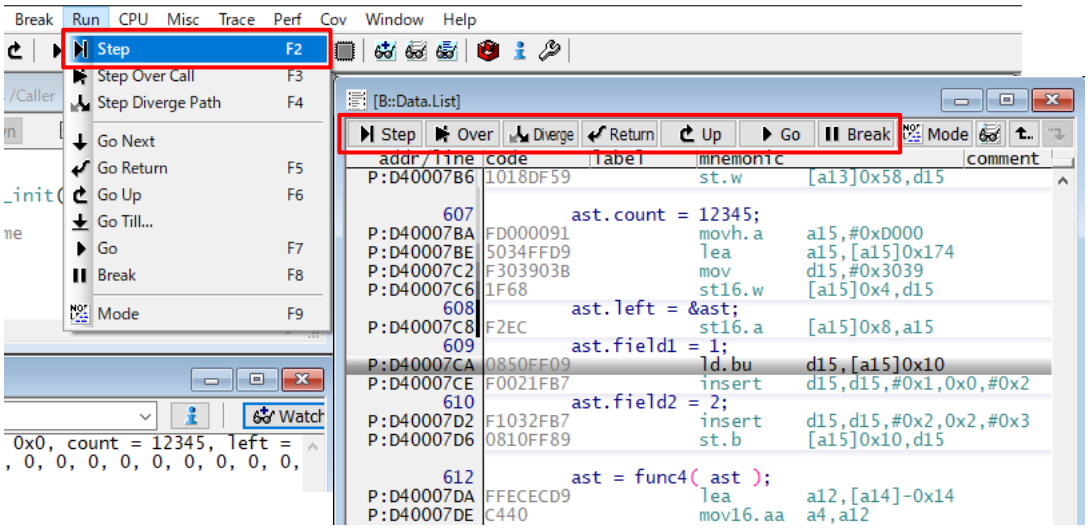
シングルステップ実行

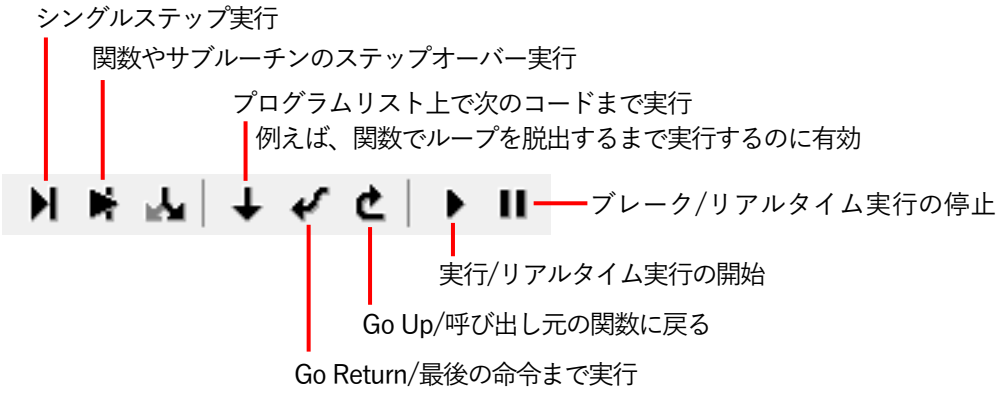
基本的なデバッグコマンドは、Run メニュー、Data.List ウィンドウのツールバー、メインツールバー、そして TRACE32 コマンドラインから利用できます。

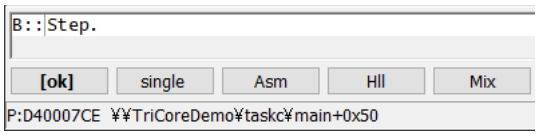
シングルステップ実行  は基本的なデバッグコマンドの 1 つで、命令ごと、もしくは 1 ステー

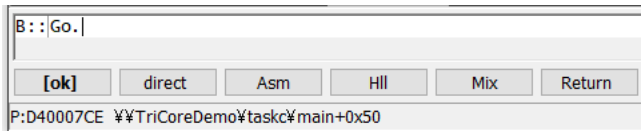
[15] 省略できない文字の直前の「.」(ドット)も省略できません。

トメントごとに実行させることです。









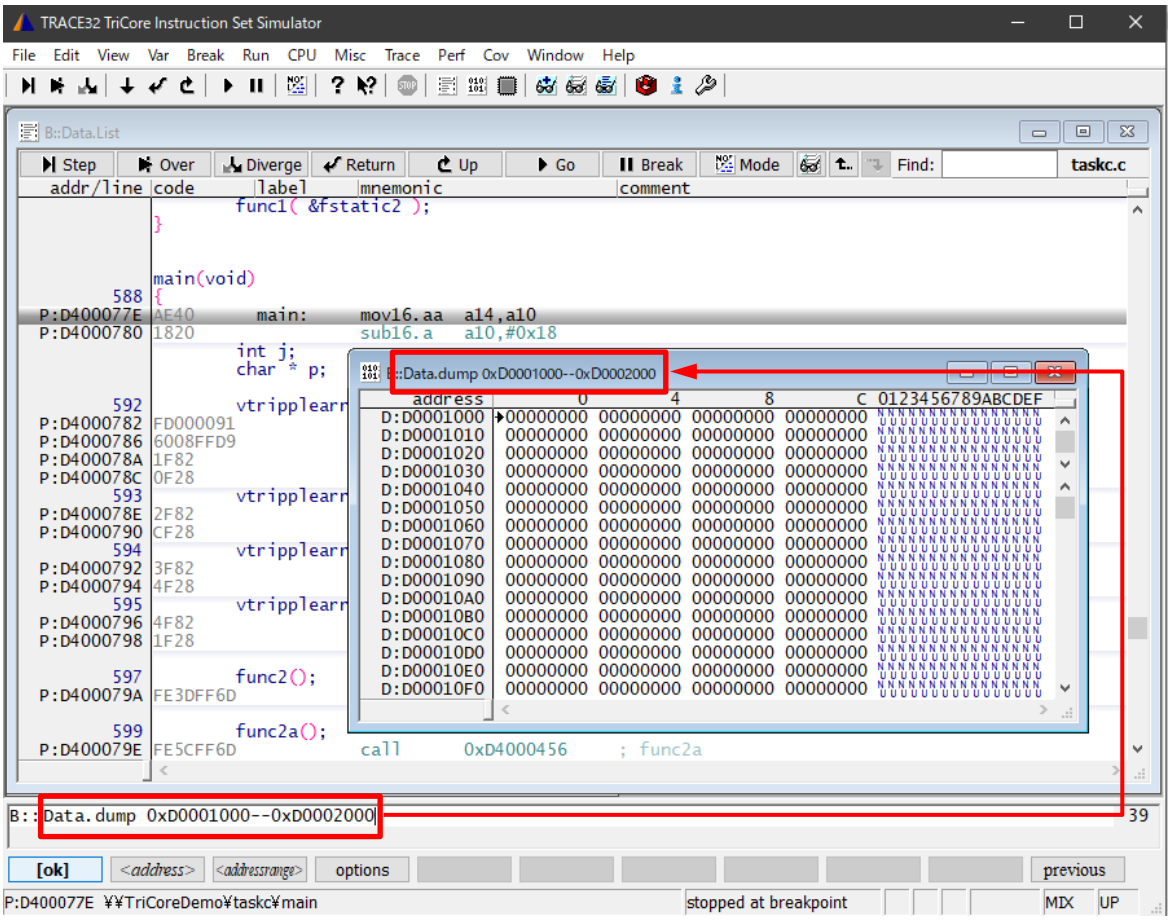
TRACE32 は、より複雑なデバッグ制御コマンドも提供します。例えば、式が変更されるか `true` になるまで実行またはシングルステップ実行できます。

例

`Var.Step.Till vchar>11.` 変数 `vchar` が `11` より大きくなるまでプログラムをシングルステップ実行します。末尾のドットは非常に重要で、これにより `11` が 10 進数としてみなされます。(注：`vchar>11.`は、`vchar>0xB` と同じです)

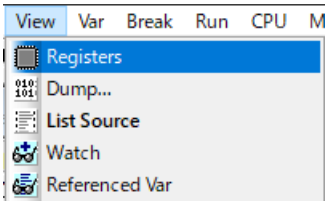
ウィンドウキャプション－TRACE32 での特別な点

ウィンドウを開くときのコマンドはウィンドウキャプションとして表示されます。パラメータとオプションもウィンドウキャプションに含まれます。

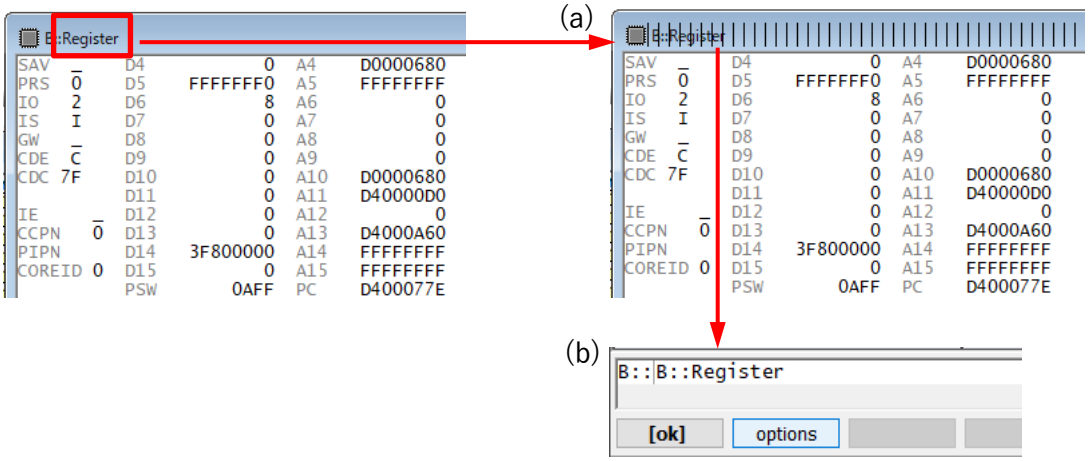


コマンドを修正するために、ウィンドウキャプションからコマンドラインにコマンドを再挿入できます。以下、Register ウィンドウによってこの操作を示します。

1. View メニューから Registers を選択します。



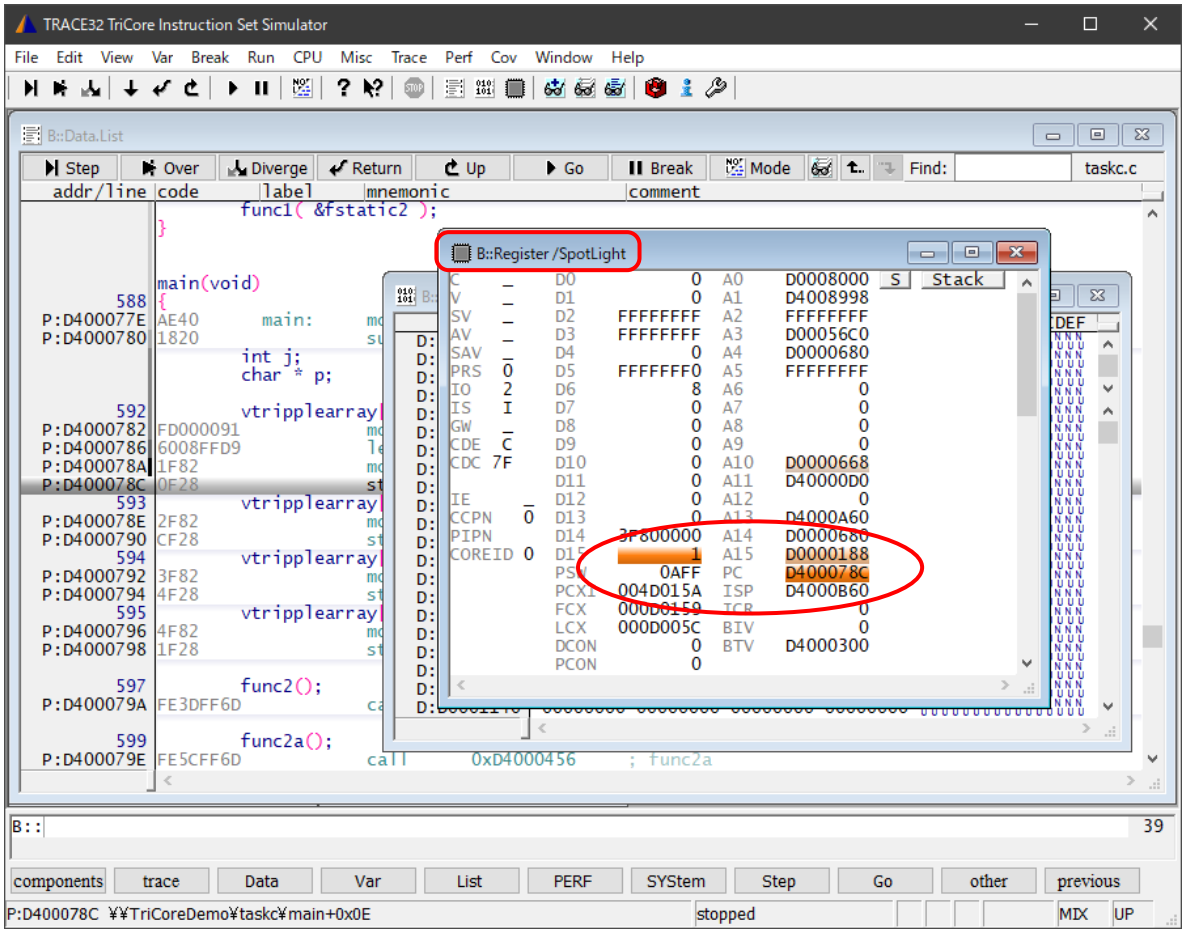
2. ウィンドウキャプションを右クリックします。(a) のようになります。このときコマンドラインは (b) のようになります。



3. コマンドを変更します。例：SpotLight ソフトキーによって /SpotLight オプションを追加します。このオプションにより変更されたレジスタが強調表示されるようになります。

4. **[ok]**をクリックして、変更したコマンドを実行します。

5. TRACE32 ツールバーのシングルステップ をクリックします。変更されたレジスタはすぐに強調表示されます。



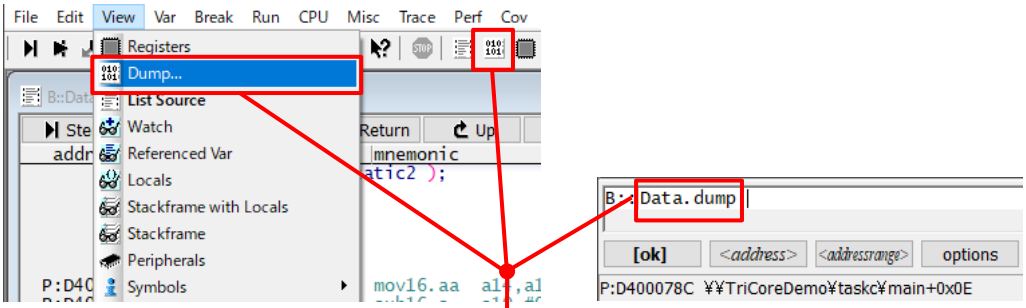
メモリ

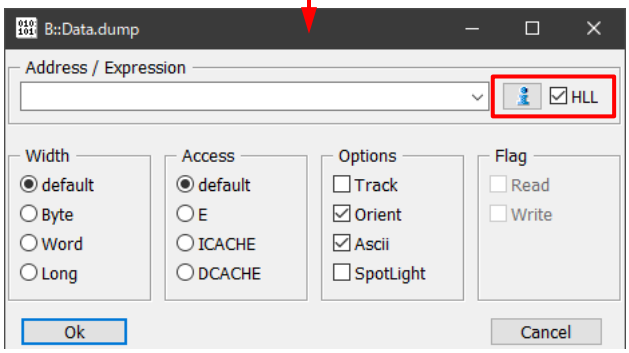
メモリの表示

1. Data.dump ウィンドウにメモリダンプを表示するために、以下のいずれかを実行します。
   - View メニューから Dump...を選択する
   - ツールバーの (Memory Dump)をクリックする
   - TRACE32 コマンドラインで Data.dump と入力する
     アドレスやシンボルを直接指定することもできます。例：Data.dump main++0x30

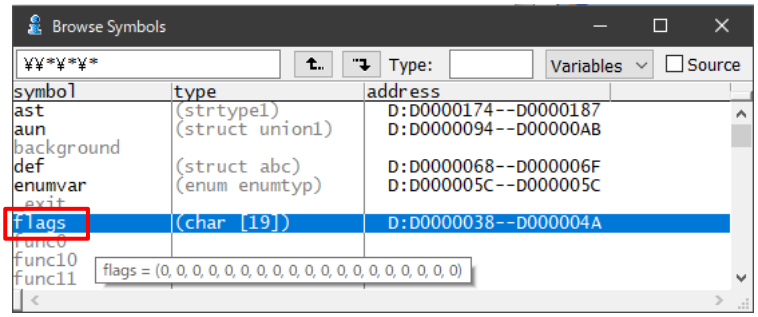
2. Data.dump ダイアログでデータ項目を入力します。例：main
   - または、HLL チェックボックスを「オン」として、 (シンボルデータベース)をクリックして参照します。

3. Browse Symbols ウィンドウでシンボル（例：`flags`）をダブルクリックして、Ok をクリックします。

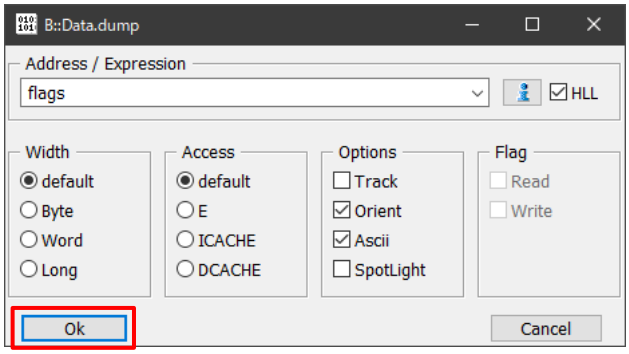




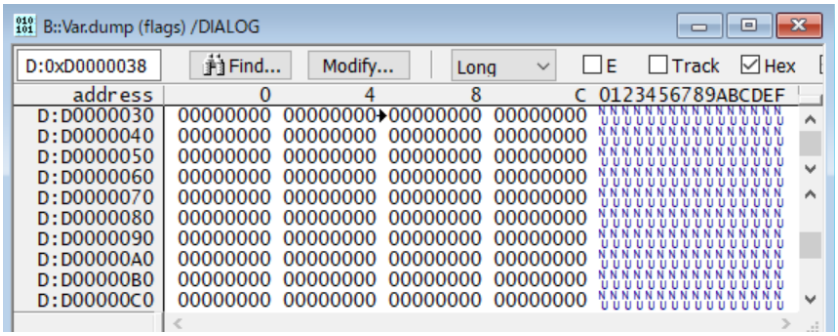
HLL チェックボックスを「オン」とし、 をクリック



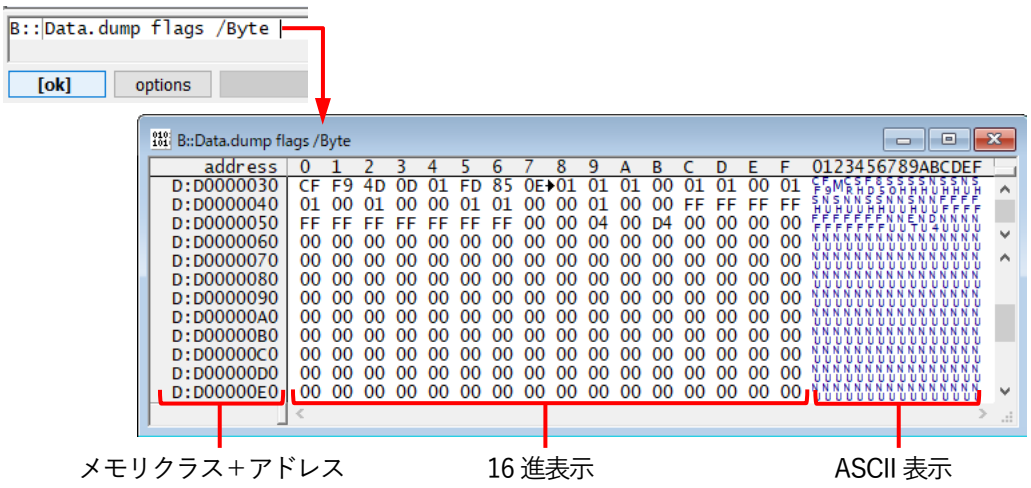
flags をダブルクリック



メモリダンプが表示されます。



下図は、Data.dump ウィンドウが TRACE32 コマンドラインから呼び出されている様子です。



アドレス範囲を定義する方法はいくつかあります。

- *<開始アドレス>--<終了アドレス >*

  `Data.dump D:0xD0000030--D:0xD00000AF` （D:はメモリクラスの修飾子です）

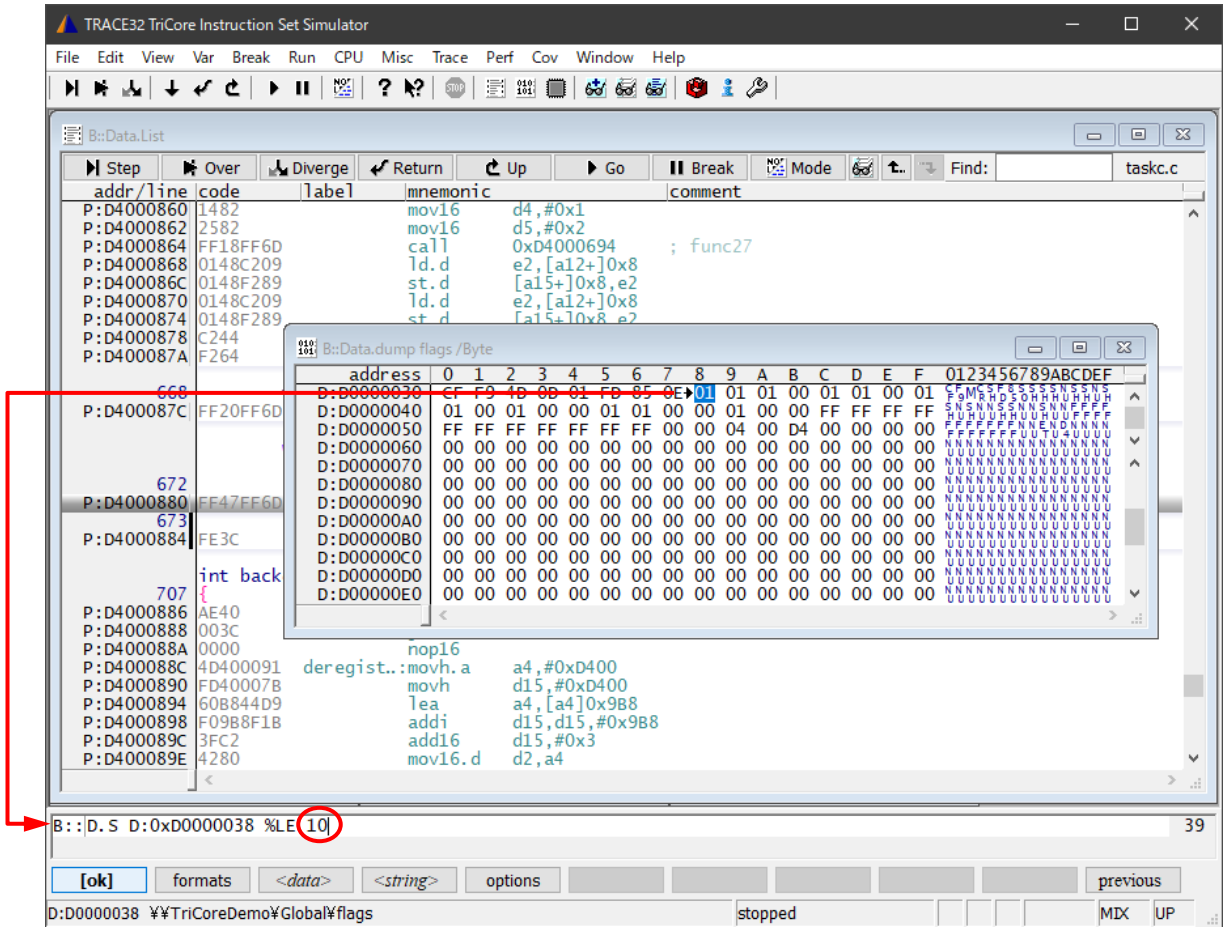
- *<開始アドレス>++<オフセット>*

  `Data.dump str2++20. /Byte` （str2 から始まって 20 バイト進む）

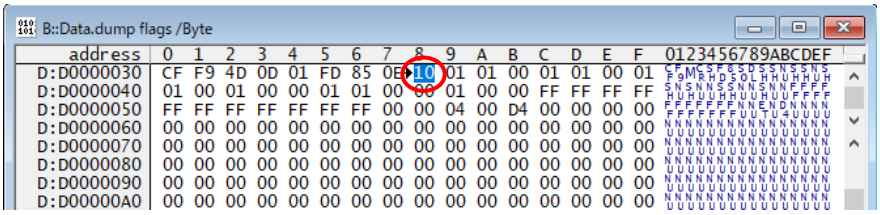
  `Data.dump str2--(str2+20.) /Byte` （上記と同じ：20 バイト先が終了アドレス）

メモリの変更

1. Data.dump ウィンドウで、変更したい値をダブルクリックします。
   選択したアドレスに対する Data.Set コマンドがコマンドラインに表示されます。コマンドの短形式は D.S または d.s です。

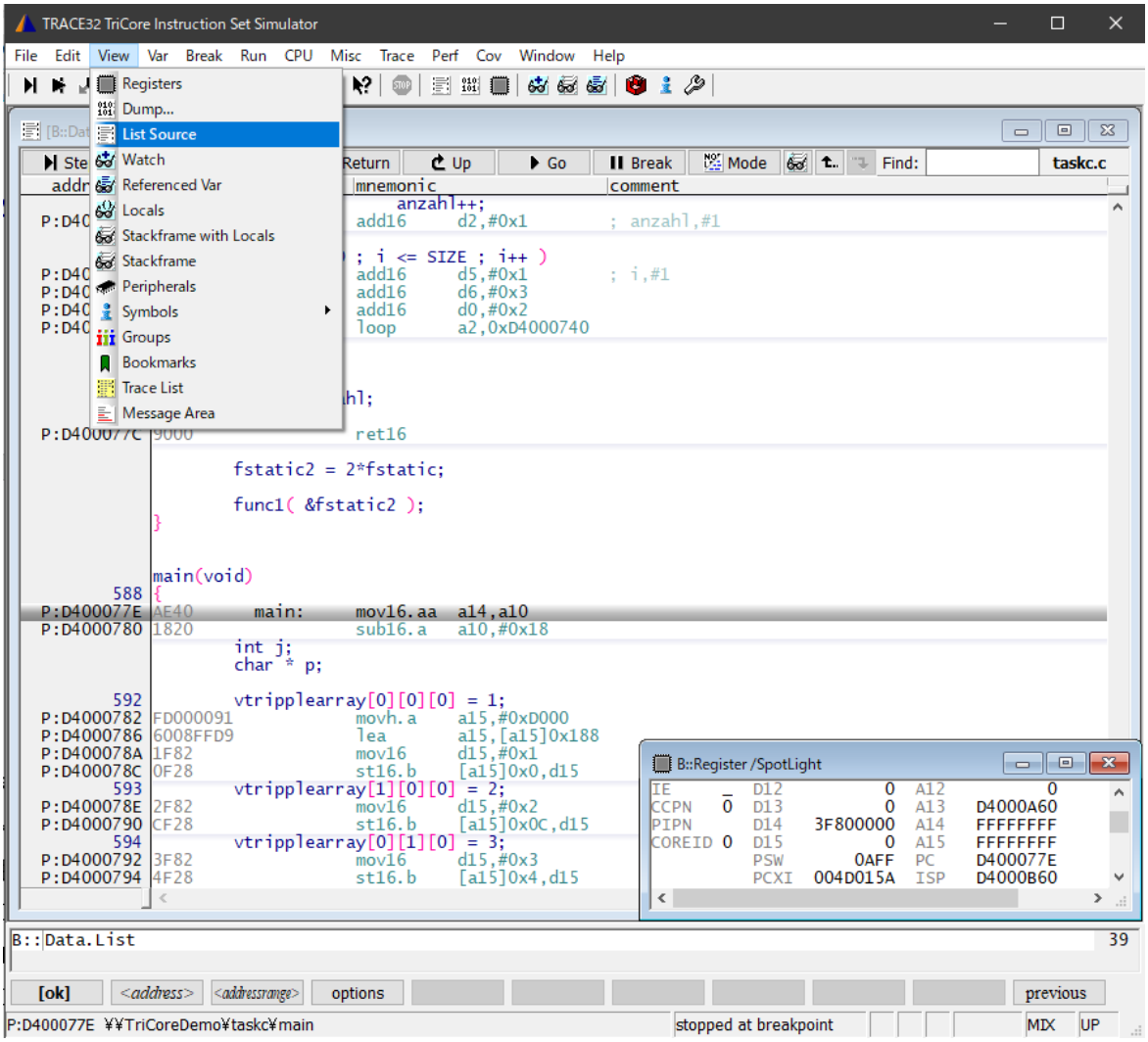


2. コマンドラインに表示された%LE の後に値を入力し、[ok]で確定します。値が変更されていることを確認してください。

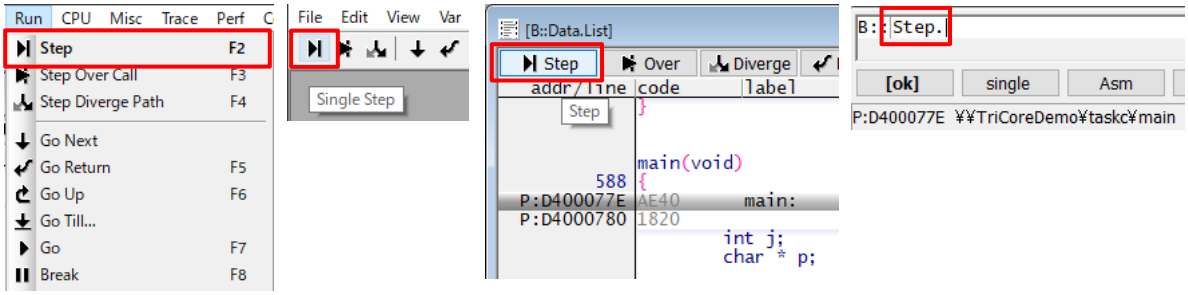


プログラムのデバッグ

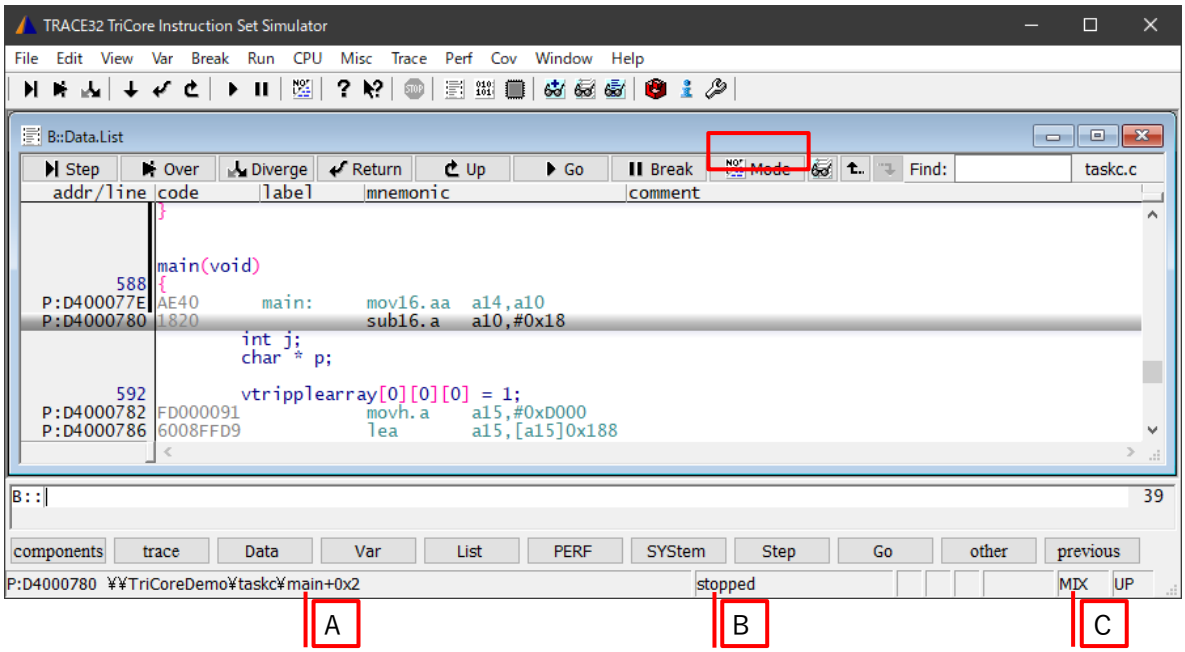
1. View メニューから List Source を選択します。
   Data.List ウィンドウが開き、現在のプログラムカウンタ周辺のプログラムリストが表示されます。



2. プログラムをシングルステップ実行するには、次のいずれかを実行します。
   - Run メニューから Step を選択する
   - F2 を押す（Run メニューで Step のファンクションキー欄が F2 となっています）
   - TRACE32 ツールバーの ⏭ をクリックする
   - Data.List ウィンドウの ⏭ Step をクリックする
   - TRACE32 コマンドラインで Step を入力する



3. TRACE32 メインウィンドウの下部にある状態行は以下のようになります。



状態行で表示される内容

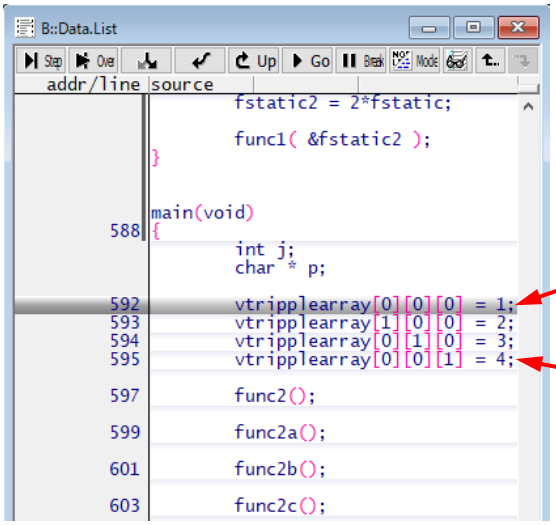
A アクティブウィンドウ内の現在のカーソル位置の(シンボリックな)アドレス： 現在のカーソル位置は青色で強調表示され、プログラムカウンタ(PC)は灰色で強調表示されています。

B デバッガの状態： stopped … アプリケーションプログラムが停止している状態であり、メモリの参照や変更などができます。

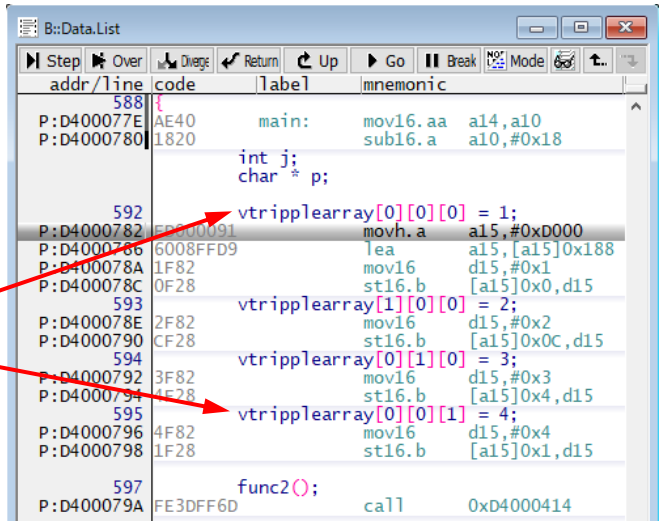
C 現在選択されているデバッグモード： コード表示は、HLL(高水準言語)、ASM(アセンブラ)、または HLL とそれに対応するアセンブラニーモニックの MIX モードになります。

4. Data.List ウィンドウのツールバーで、Mode をクリックしてデバッグモードを HLL に切り替えます。

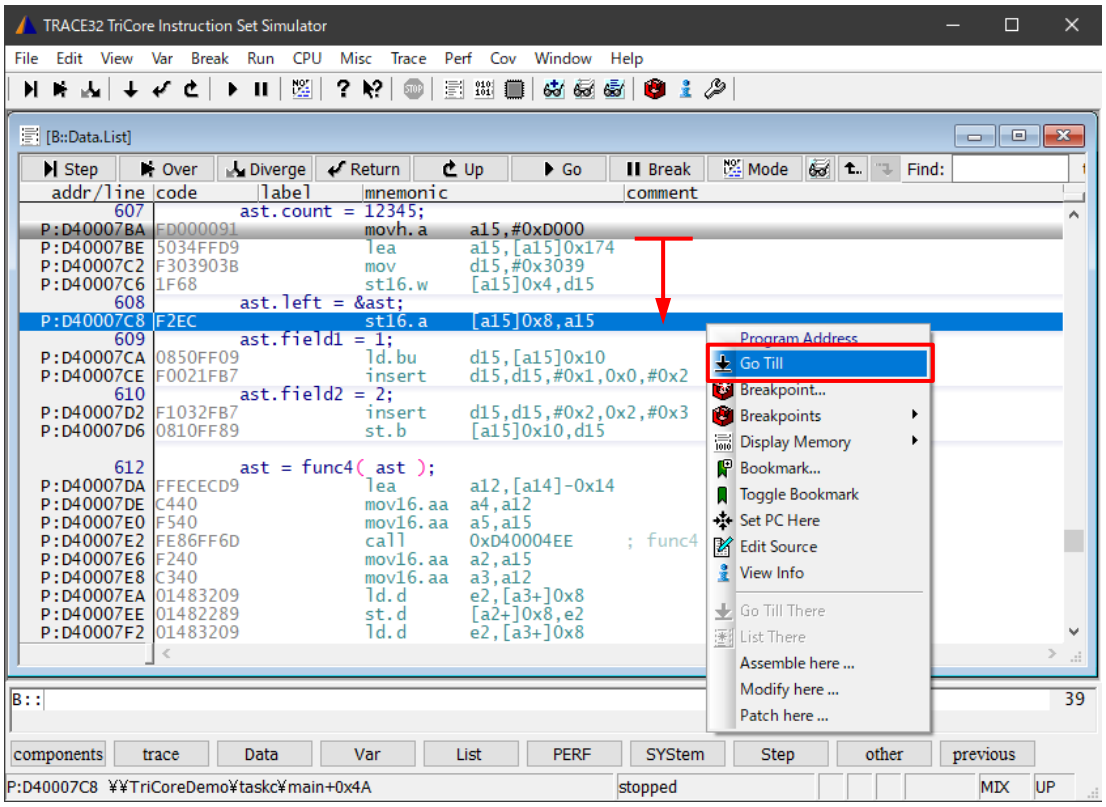
HLL デバッグモード



MIX デバッグモード

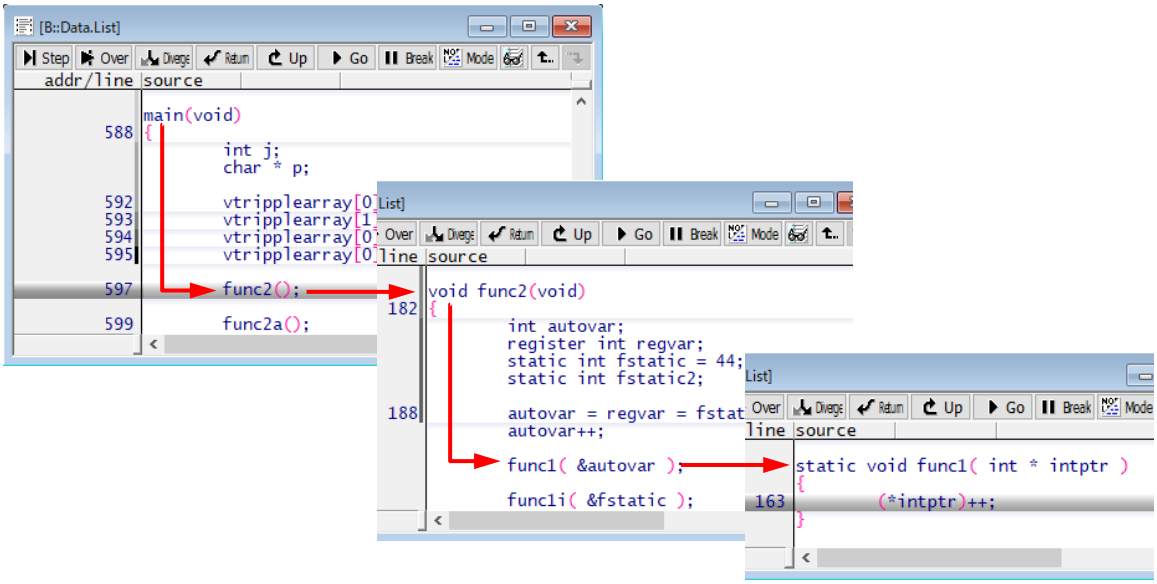


5. Step をクリックします
   ここで行っているのは、次の HLL 行への高水準言語レベルのステップ実行です
6. もう一度 Mode をクリックして、デバッグモードを MIX に切り替えます
7. Step をクリックします
   今回は、1 つのアセンブラ行が実行されます
8. コード行を右クリックして、Go Till を選択します
   プログラムの実行が始まります。プログラムが選択されたコード行に達すると停止します



入れ子になった関数の表示

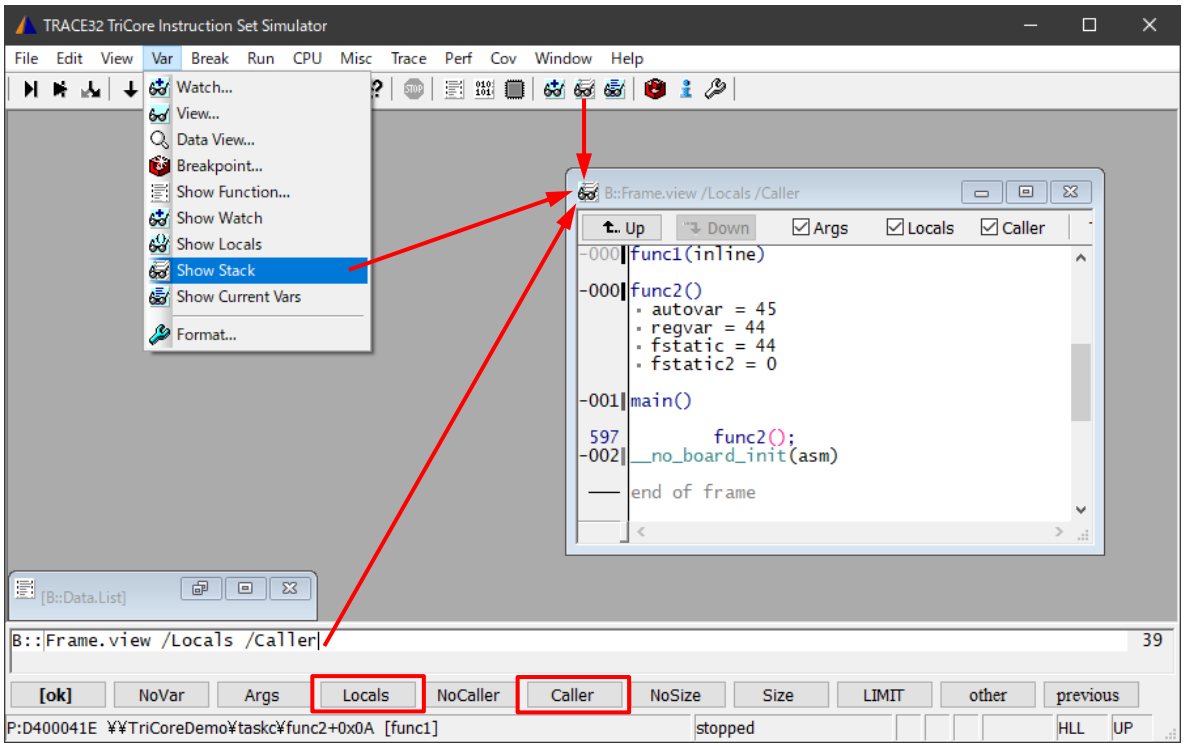
次の例では、関数のネストがあるとします。`main()` が `func2()` を呼び出し、`func2()` が `func1()`を呼び出します[16]。



Var メニューから Show Stack を選択します。Frame.view ウィンドウに関数の入れ子が表示されます。ここで、

- /Locals オプションは、各関数のローカル変数を表示します
- /Caller オプションは、関数が呼び出された場所を示すために C コードを数行表示します

上記のネストと呼び出しのシーケンスを下図に示します。



16 本例では、func1 がインライン展開されています。

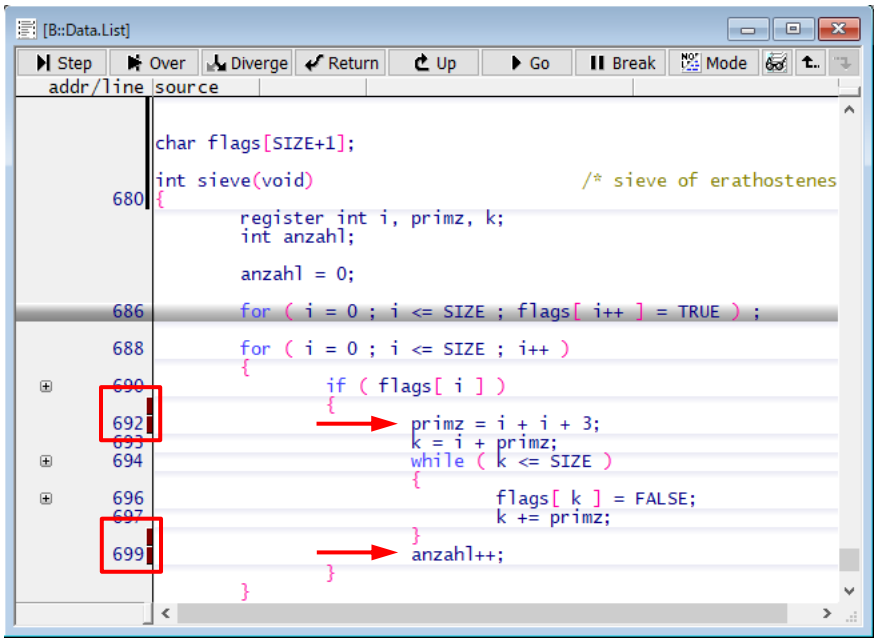
ブレークポイント

ブレークポイントは最も頻繁に使用されるデバッグ機能の 1 つです。

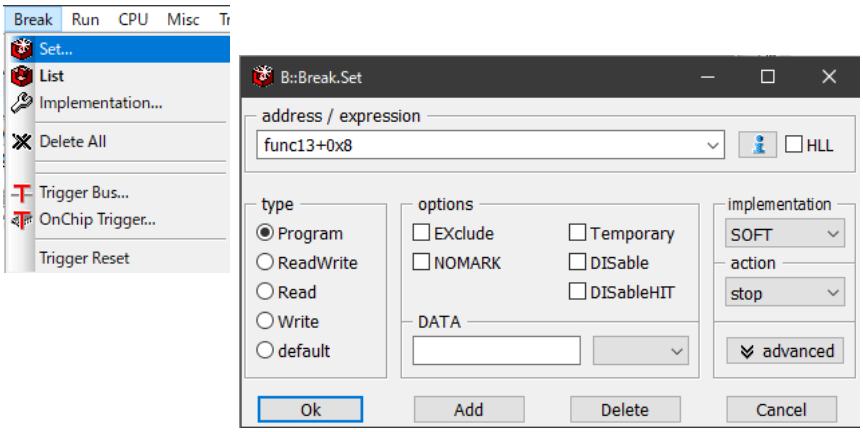
ソフトウェアブレークポイントの設定

以下のようにして `primz` の代入と `anzahl++`にブレークポイントを設定します。

1. コード行をダブルクリックしてプログラムブレークポイントを設定する
2. コード行の文字ではなく空白部分をクリックしてください
   プログラムブレークポイントを持つすべてのコード行は、茶色の縦線でマークされています。

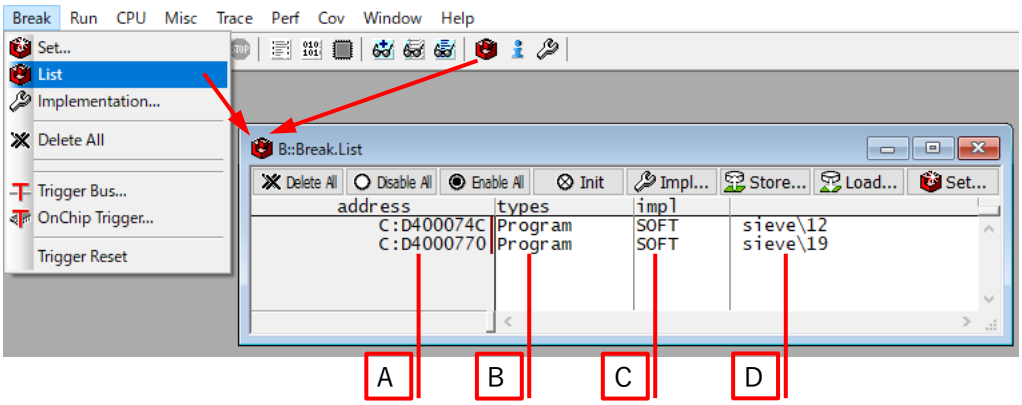


なお、ブレークポイントの設定は、Break メニューから set... を選択してもできます。



すべてのブレークポイントのリスト表示

1. Break メニューから List を選択して、すべてのブレークポイントを一覧表示します。
   Break.List ウィンドウが開き、設定されているブレークポイントの概要が表示されます。



A　ブレークポイントの 16 進表示アドレス

B　ブレークポイントのタイプ

C　ブレークポイントの実装：SOFTware、ONCHIP、または DISABLED

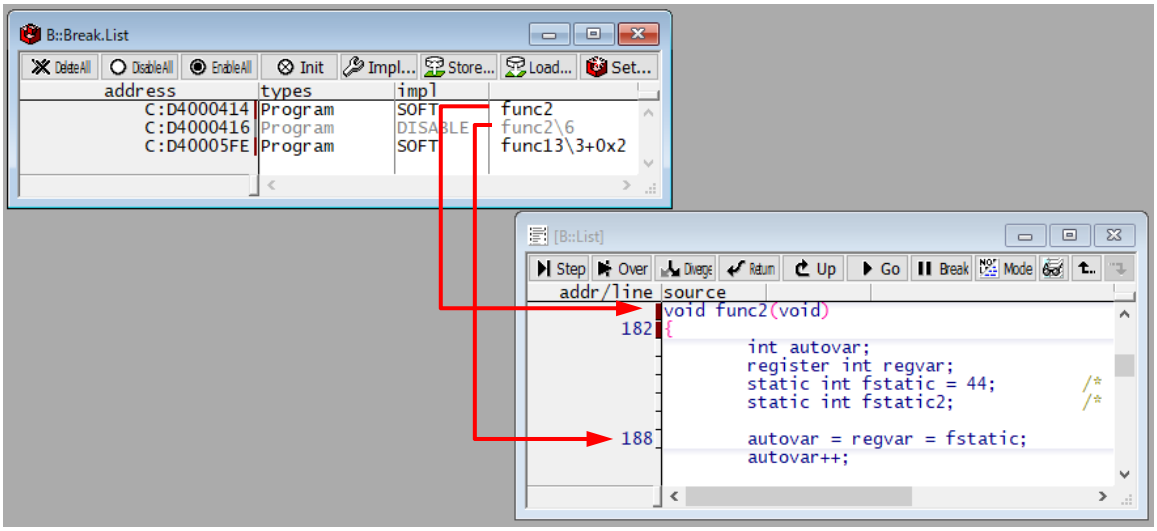
Data.List ウィンドウの灰色のバーは、DISABLED のブレークポイントを示します

D　ブレークポイントのコード行

例

- `func2¥6` は、`func2` の HLL での 6 行目を意味します
- `func2¥13+0x8` は、`func2` の HLL での 13 行目から 8 バイト先を意味します
  (デバッグモード MIX でのみ有用です)

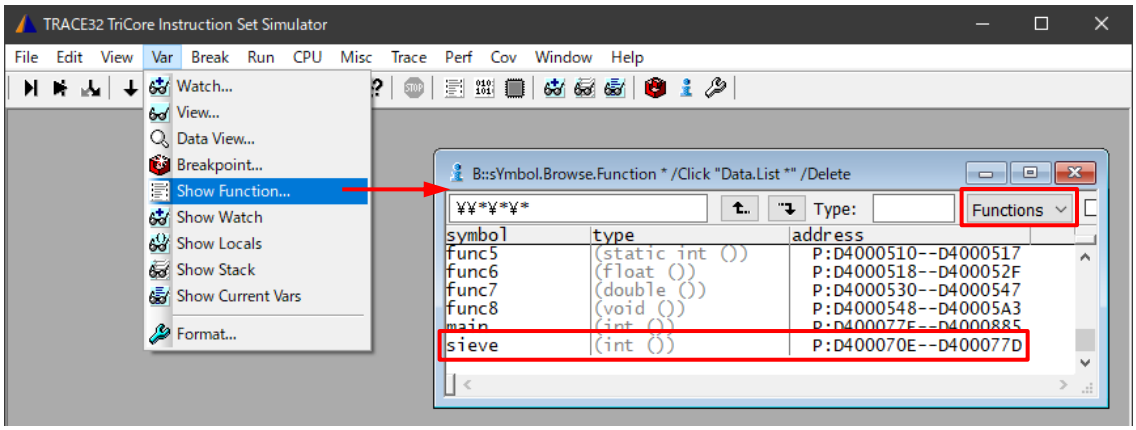
2. ツールバーの ▶ をクリックしてプログラムの実行を開始します。
3. プログラムがブレークポイントに達しない場合は、⏸をクリックしてプログラムの実行を停止します。



関数内で任意のコード行にプログラムブレークポイントを設定する

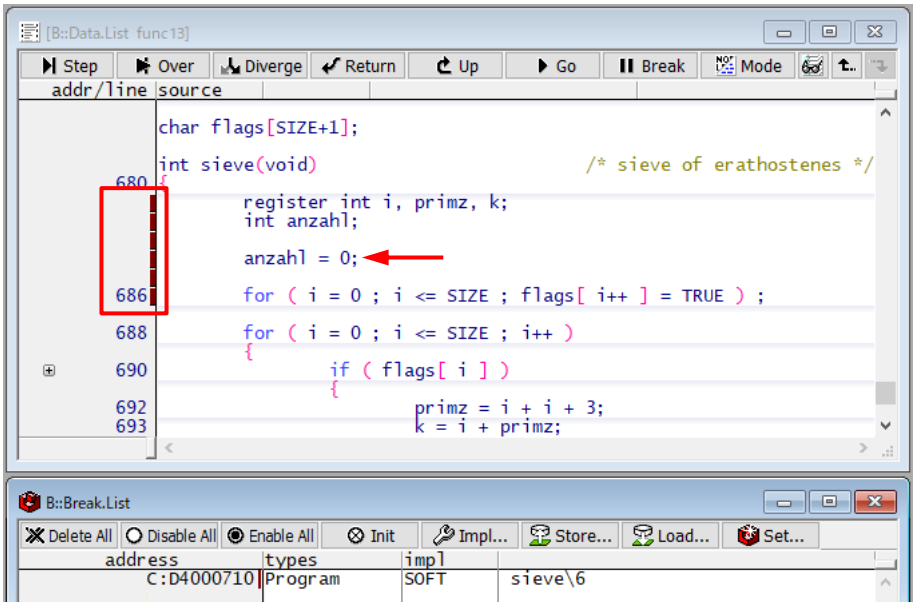
以下のように、関数 `sieve()`でブレークポイントを `anzahl = 0` に設定します。

1. Var メニューから Show Function を選択します
   sYmbol.Browse.Function ウィンドウが開きます



2. `sieve` をダブルクリックします
   Data.List ウィンドウが開き、`sieve` が表示されます

3. コード行の空白をダブルクリックしてブレークポイントを `anzahl = 0` に設定します
   ブレークポイントを示す茶色の縦線が Data.List ウィンドウに表示されます



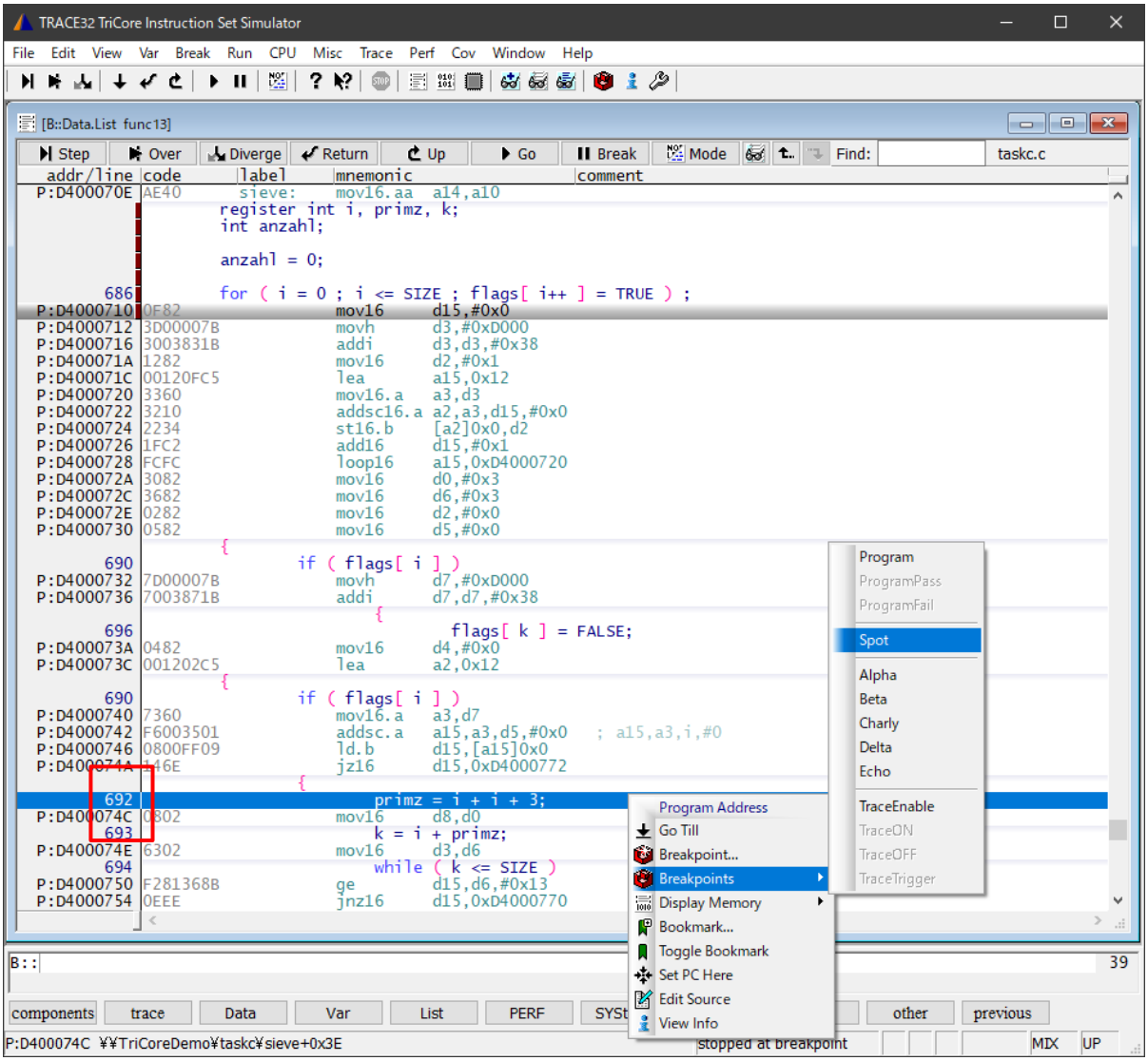
4. Break メニューから List を選択して、すべてのブレークポイントを一覧表示します
   Break.List ウィンドウが開き、既存のブレークポイントの概要が表示されます

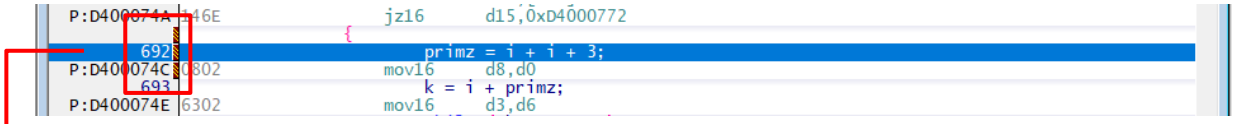
スポットブレークポイントの設定

スポットブレークポイントは、表示されているすべての情報を更新するためにプログラムの実行を短時間停止してからプログラムの実行を再開する監視ポイントです。次のページで紹介する機能と組み合わせて、プログラム実行中に変数の変更を監視することができます。まず、以下のようにして、変数 `primz` にスポットブレークポイントを設定します。

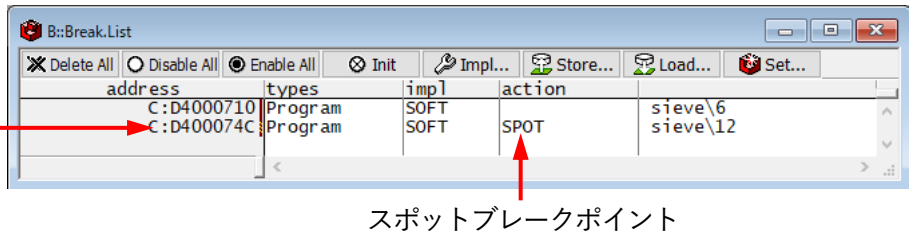
スポットブレークポイントを設定する

1. 変数 `primz` のコード行の空白部分を右クリックし、Breakpoints > Spot を選択します
   スポットブレークポイントが設定されると、黄色でハッチングされた茶色の縦線が Data.List ウィンドウに表示されます





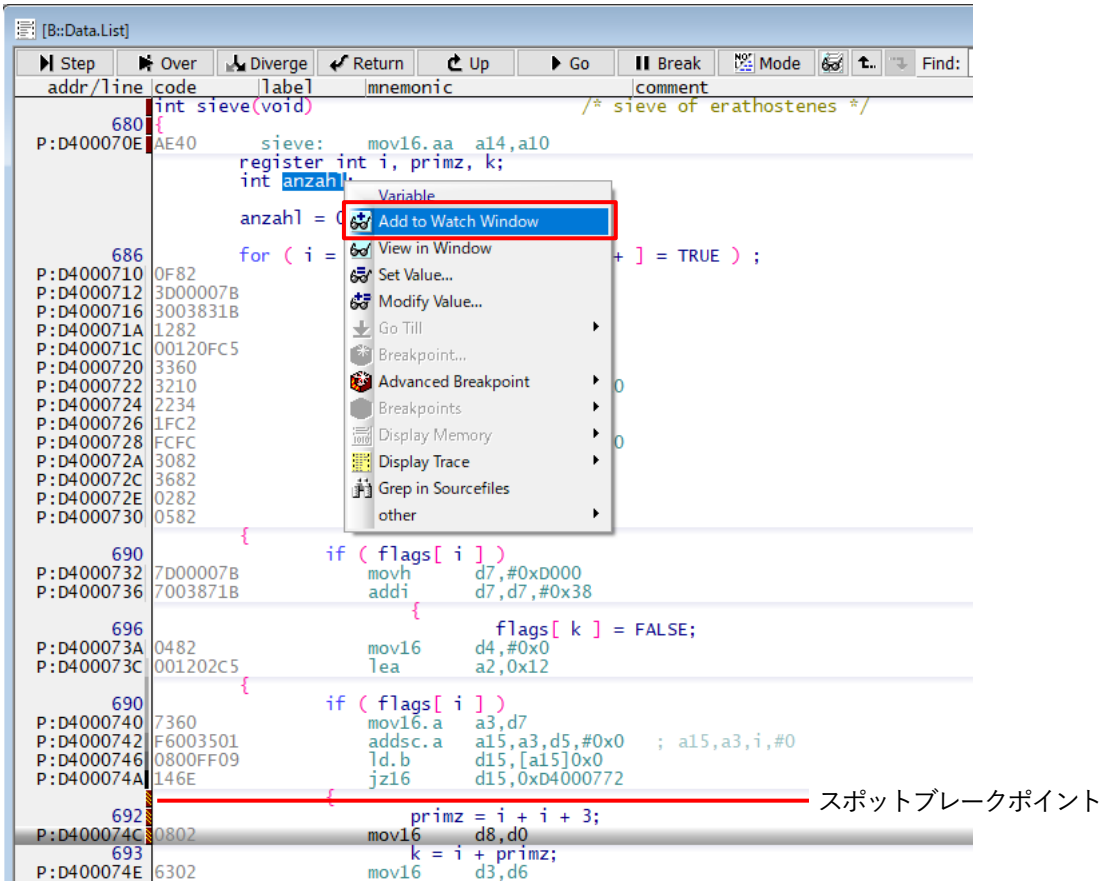
2. Break メニューから List を選択して、すべてのブレークポイントを一覧表示します
   Break.List ウィンドウが開き、設定されているブレークポイントの概要が表示されます



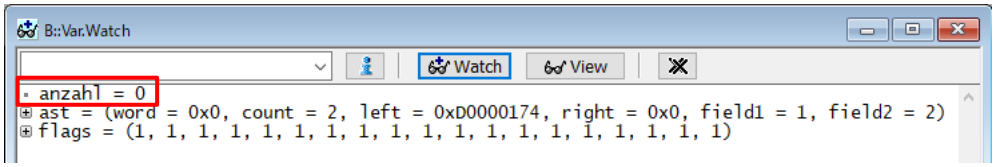
変数にウォッチを設定する

プログラムの実行中に、変数 anzahl のすべての変更を監視します。前提は、変数 primz にスポットブレークポイントを設定した状態です。

1. 変数 anzahl を右クリックし、Add to Watch Window を選択します

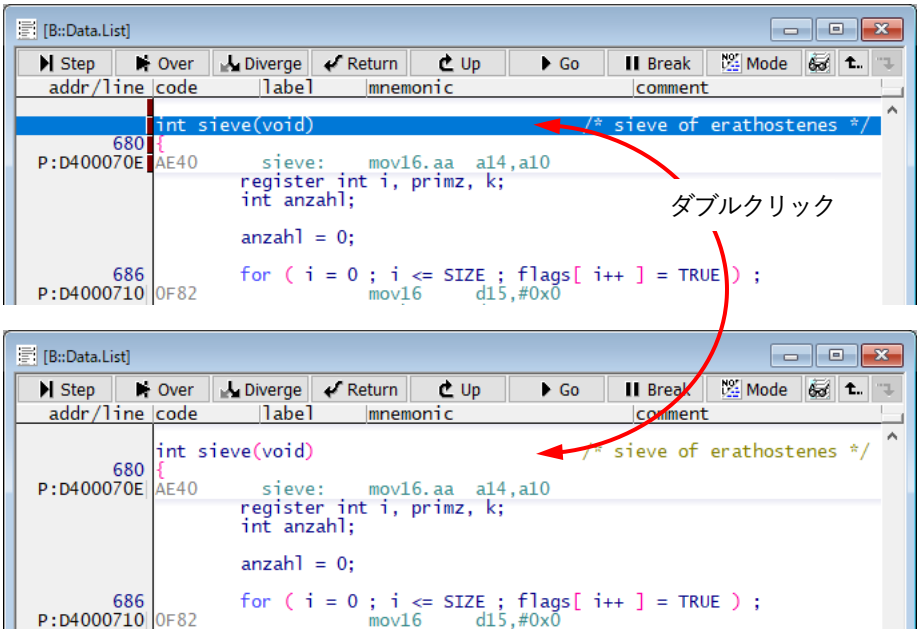


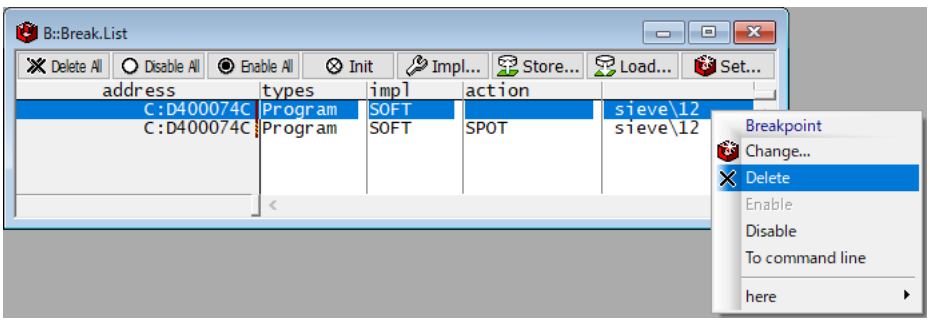
2. Data.List ウィンドウで、 Go をクリックしてプログラムの実行を開始します
   プログラム実行中に、Var.Watch ウィンドウで変数 anzahl が変更される様子を見ることができます



ブレークポイントの削除

ブレークポイントを削除するには、Data.List ウィンドウにおいて、マークされた行をダブルクリックする(ダブルクリックで設定と削除がトグルします)か、Break.List ウィンドウで右クリックして Delete を選択します。



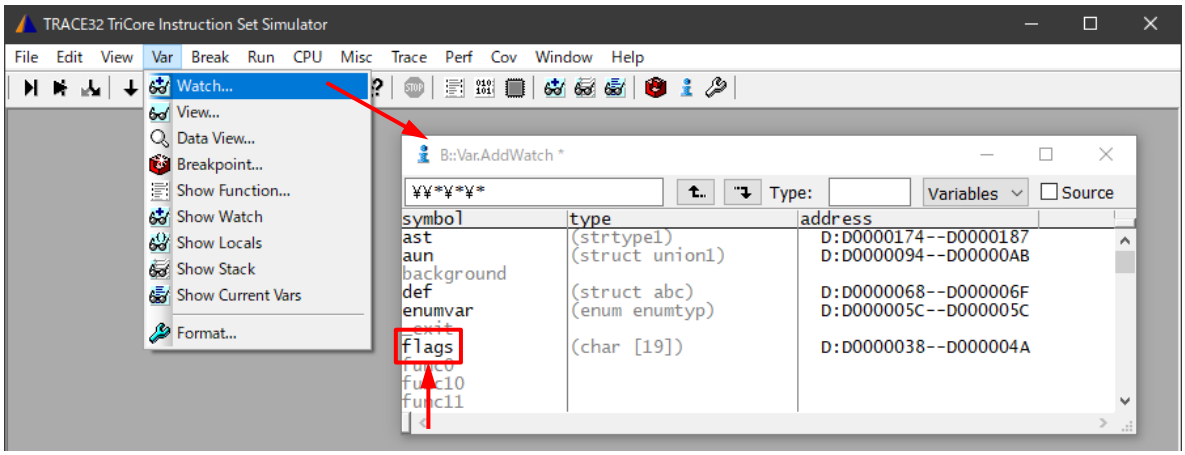


HLL 変数

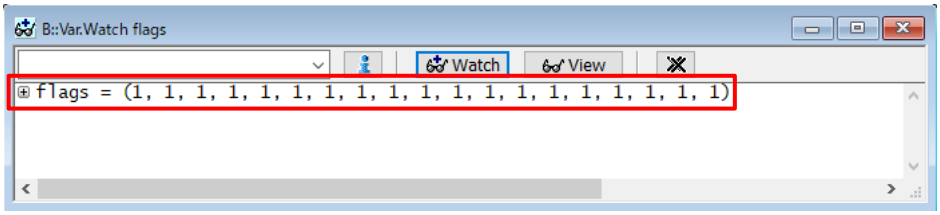
HLL 変数の表示

以下のようにして、変数 `flags`、`def`、および `ast` を表示します。

1. Var メニューから Watch...を選択します
   Var.AddWatch ウィンドウが開き、シンボルデータベースにロードされた HLL 変数が表示されます

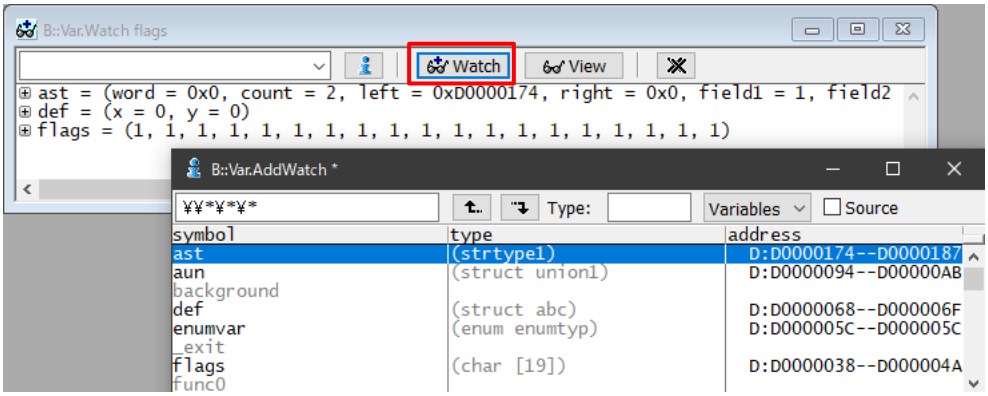


2. HLL 変数 `flags` をダブルクリックします
   Var.Watch ウィンドウが開き、選択した HLL 変数が表示されます



3. **変数表示の別法として、以下があります**
   - Var.Watch ウィンドウで、 

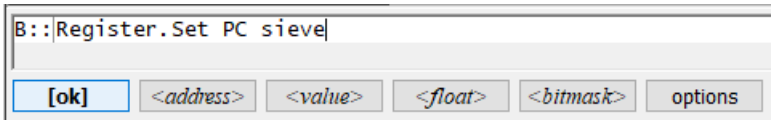
をクリックし、次に変数 `def` と `ast` をダブルクリックして Var.Watch ウィンドウに追加します



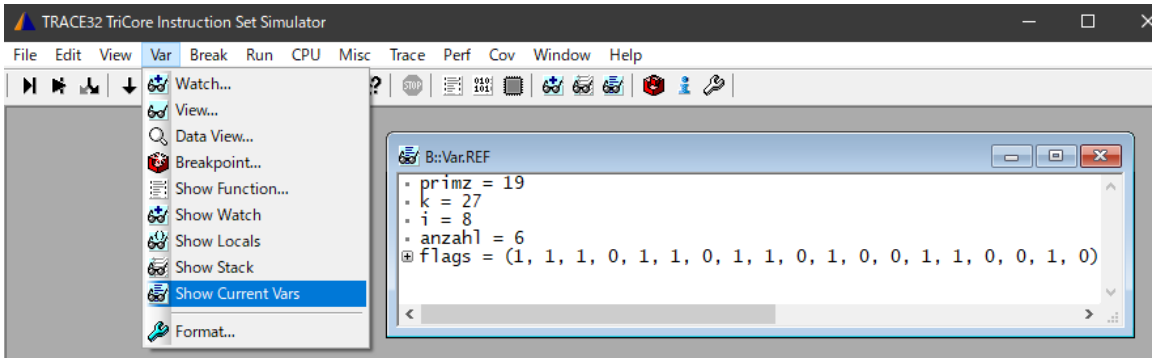
- Data.List ウィンドウから、Var.Watch ウィンドウに必要な変数をドラッグアンドドロップします
- Data.List ウィンドウで任意の変数を右クリックし、コンテキストメニューから Add to Watch window を選択します
- 複雑な構造や配列を別のウィンドウに表示する場合は、Var メニューから View を選択します

現在のプログラムコンテキストで HLL 変数を表示する

1. TRACE32 コマンドラインで、プログラムカウンタ(PC)を `sieve()`に設定します



2. Var メニューから Show Current Vars を選択します
   Var.REF ウィンドウが開き、現在のプログラムコンテキストによってアクセスされるすべての変数が表示されます。

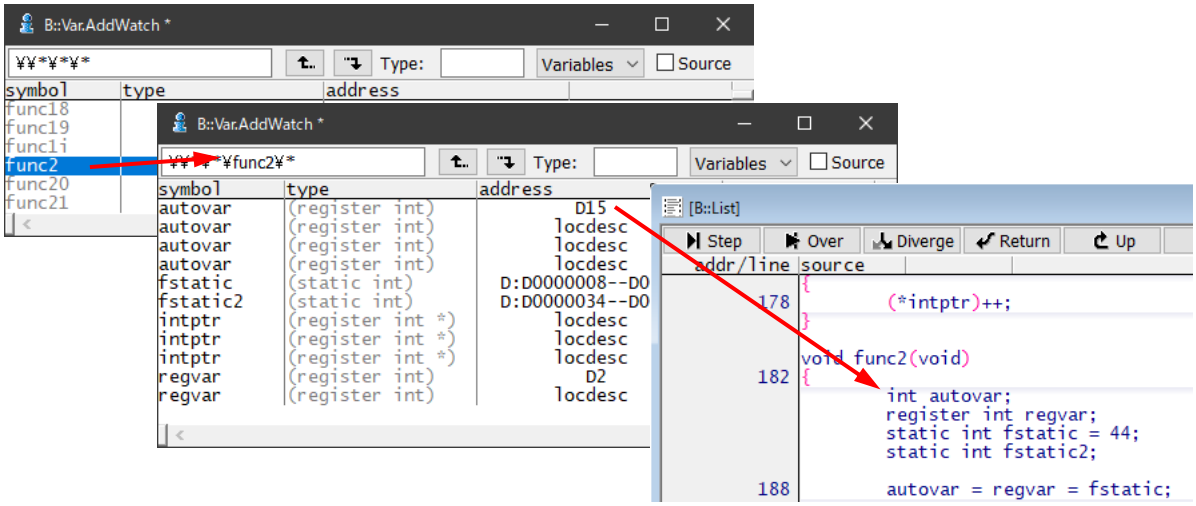


3. TRACE32 ツールバーの  をクリックして、いくつかステップ実行をします
   Var.REF ウィンドウは自動的に更新されます

シンボルブラウザを使う

シンボルブラウザは、現在シンボルデータベースに格納されている変数、関数、およびモジュールの概要を提供します。

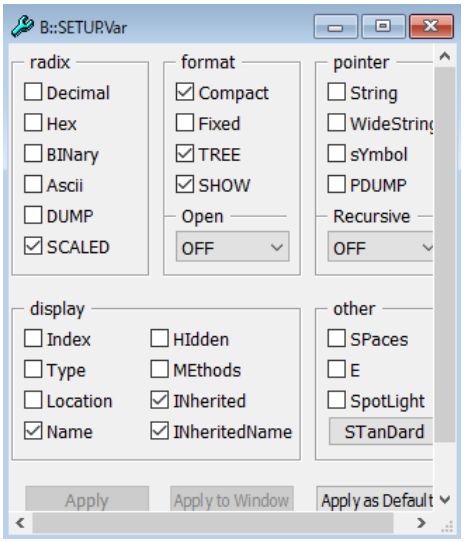
1. Var メニューから Watch を選択します
   Var.AddWatch ウィンドウでは、シンボルデータベースの内容を閲覧できます
2. Var.AddWatch ウィンドウで、`func2` をダブルクリックします



HLL 変数のフォーマット

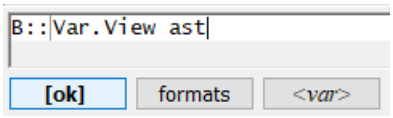
HLL 変数の表示フォーマットを設定する - グローバル設定

1. Var メニューから Format を選択します
2. SETUP.Var ウィンドウで設定を行います
   Decimal(10 進数)と Hex(16 進数)が便利なフォーマットです
   グローバル設定ですので、その後で開いたすべての Var.view ウィンドウにこの設定が適用されます



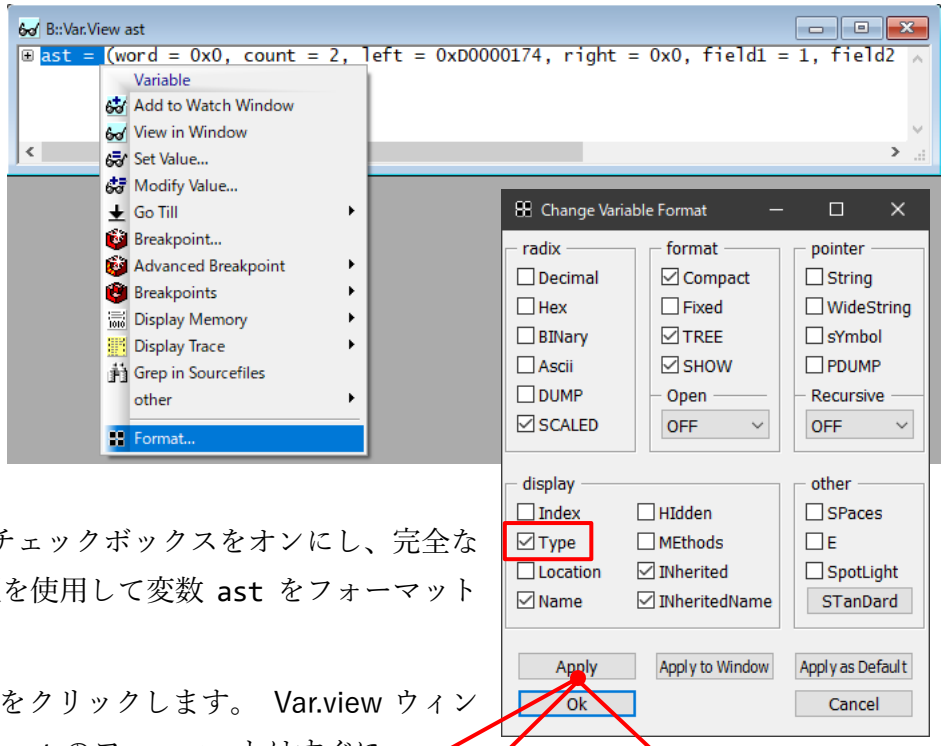
HLL 変数個々に表示をフォーマットするには以下のようにします。

1. コマンドラインで次のように入力します



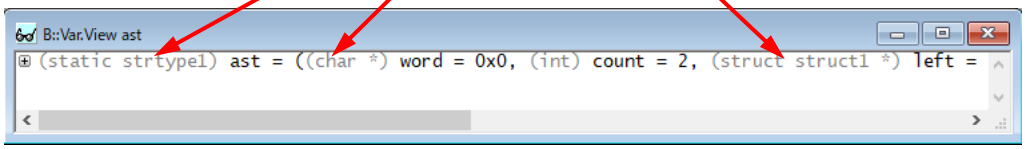
(変数 ast はこのデモに含まれています)

2. Var.view ウィンドウで ast を右クリックし、Format をクリックします
   Change Variable Format ダイアログが開きます



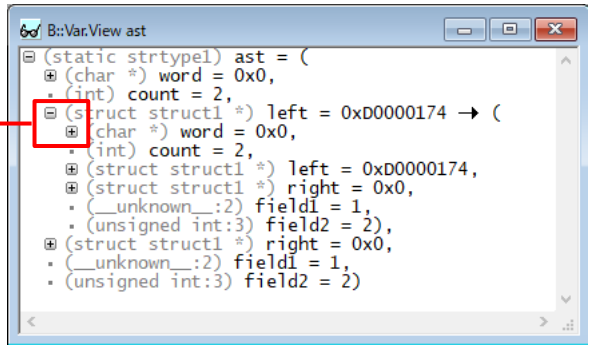
3. Type チェックボックスをオンにし、完全な型情報を使用して変数 ast をフォーマットします

4. Apply をクリックします。 Var.view ウィンドウの ast のフォーマットはすぐに更新されます



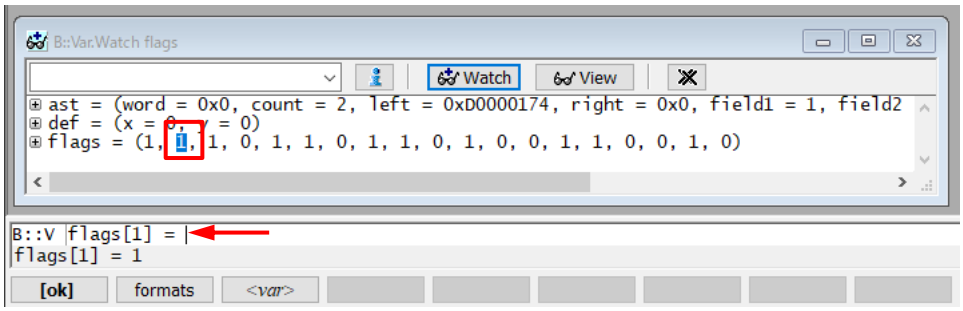
5. もっと複雑な HLL 構造の場合は、Change Variable Format ダイアログボックスの format で TREE を選択します

ウィンドウ内表示の[+]を
クリックして展開すると、
下位の構造が表示されます



変数値の変更

1. 変数値をダブルクリックして値を変更します
   Var.set コマンドがコマンドラインに表示されます
   コマンドの短縮形は V または v です



2. 等号(=)の直後に新しい値を入力し、[ok]で確定します

以上、このチュートリアルではプロジェクトをビルドしてデバッグするまでの作業を紹介しました。このプロジェクトを利用して、LDRA 社テストツールで静的解析や単体テストの自動化を紹介する資料も公開しています。

https://www.fuji-setsu.co.jp/files/TriCore_LDRA_TLP.pdf

またテストツールとの連携については、以下に動画やスライドも公開しています。

https://www.fuji-setsu.co.jp/products/LDRA/#TriCore