

<b>O przewodniku</b>	<b>2</b>
<b>Praca z debugger'em</b>	<b>3</b>
Konfiguracja środowiska	3
Uruchomienie TRACE32-ICD	4
<b>Główne okno TRACE32</b>	<b>5</b>
O TRACE32	6
Online Help	7
Ustawienia środowiska Debug	9
Pliki wsadowe	14
Interfejs użytkownika	16
Podgląd i modyfikacja pamięci	18
Debugowanie programu	20
Jak ustawić punkty przerwań?	25
Programowe punkty przerwań	25
Punkty przerwań w pamięci ROM, Flash i EEPROM	30
Punkty przerwań na dostępie do danych	32
Punkty przerwań on-chip na różnych architekturach procesorów	34
Procesory RISC/CISC	35
Procesory DSP	39
Softcores	41
Rdzenie konfigurowalne	41
Podgląd i modyfikacja zmiennych HLL	42
Format zmiennych HLL	45
Kończenie pracy TRACE32	47

## O przewodniku

---

### O czym jest ten przewodnik?

Przewodnik dotyczy wszystkich debugger'ów In-Circuit (TRACE32-ICD), które używają interfejsu typu on-chip (np. BDM, JTAG i ONCE).

### Założenia

Przewodnik zakłada, iż oprogramowanie TRACE32 jest aktualnie zainstalowane. Pomocna jest podstawowa wiedza na temat oprogramowania debugującego oraz znajomość języka programowania C. Warunki te są konieczne w zrozumieniu przykładowych kodów źródłowych znajdujących się w tym dokumencie. Dodatkowo, zakładana jest elementarna znajomość systemu operacyjnego Windows. Wiedza na temat docelowych układów procesorowych oraz assemblerów i kompilatorów jest niezbędna do uruchomienia oraz pracy z oprogramowaniem TRACE32.

### Przeznaczenie tego przewodnika

Celem niniejszego dokumentu jest opisanie podstawowych czynności konfigurujących środowisko pracy. Oznacza to, iż opisane są tu sposoby konfiguracji oprogramowania TRACE32 na komputerze typu host, uruchomienia debugger'a oraz tworzenia plików wsadowych w celu automatyzacji pracy. Dodatkowym aspektem przewodnika jest zaznajomienie Cię z podstawowymi cechami i właściwościami debugger'aa In-Circuit.

### Jak używać przewodnika?

Przewodnik zawiera opis przykładowej sesji debugowania. Wykorzystuje on prosty program napisany w języku C w celu pokazania najważniejszych właściwości oprogramowania TRACE32. Wskazane jest, abyś przeprowadził kilka ćwiczeń czytając ten dokument. Rekomendowane jest również, dokładne i kompletne przeczytanie wszystkich zawartych tu rozdziałów, zarówno treści podstawowej (pisanej normalnym tekstem) jak i wskazówek (pisanych kursywą), które nie są powtarzane w innych rozdziałach.

### Gdzie mogę znaleźć więcej informacji?

Interfejs użytkownika programu TRACE32 zawiera szczegółową pomoc (Online Help), która oferuje aktualne opisy wszystkich cech i właściwości przedstawionego tu środowiska. Rozdział 'Online Help' opisuje sposób pracy z systemem pomocy.

### Ile czasu to zajmuje?

60 minut.

## Konfiguracja środowiska

Po instalacji przeprowadzonej zgodnie z dokumentem 'ICD Szybka instalacja' pliki oprogramowania TRACE32 znajdują się w katalogu systemowych TRACE32. Dodatkowo, dostęp do programu możliwy jest przez menu Start. Proces instalacji ustawia wszystkie domyślne zmienne środowiskowe. Przedstawiona tu konfiguracja powinna być dostosowana do Twojego środowiska debugowania na komputerze PC typu host.

Uwaga: Skrót PC w kontekście oprogramowania TRACE32 zazwyczaj używany jest jako licznik programu (ang. Program Counter), dlatego też będziemy używać terminu host zamiast PC, oznaczającego urządzenie nadrzędne, które uruchamia środowisko TRACE32.

W celu dostosowania środowiska do swoich potrzeb, zapoznaj się z poniższymi terminami:

1. Definicja pliku konfiguracyjnego użytkownika.

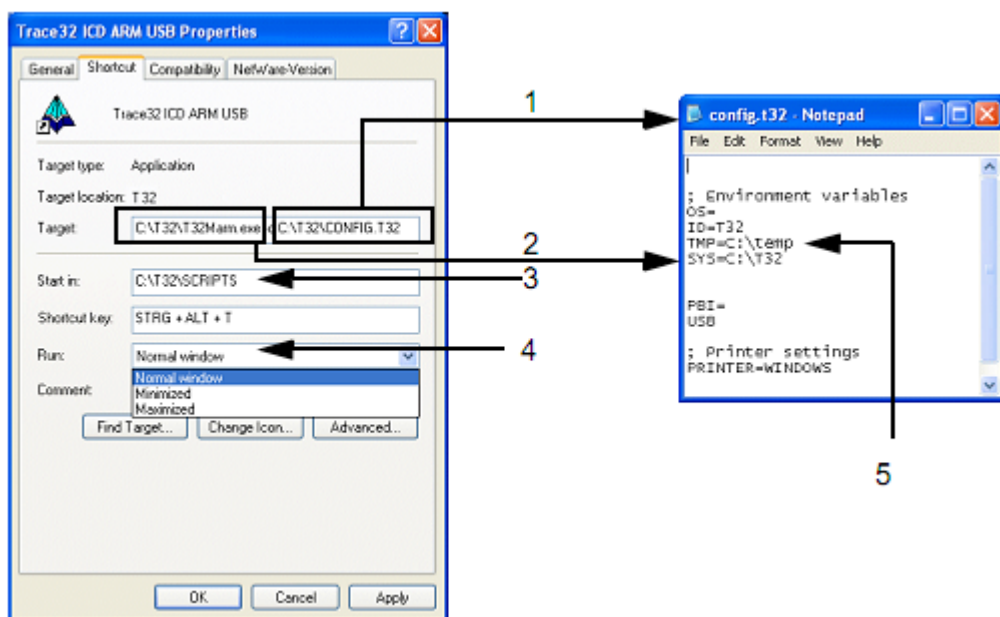
Domyślnie używany jest plik konfiguracyjny config.t32 znajdujący się w folderze systemowym. Parametr -c umożliwia zdefiniowanie innej lokalizacji oraz nazwy dla tego pliku. Więcej informacji na temat plików konfiguracyjnych znajdziesz w dokumencie 'ICD Szybka instalacja'.

2. Folder systemowy. Jest on podawany podczas procesu instalacyjnego. Zazwyczaj nie musisz dokonywać tu jakichkolwiek modyfikacji.

3. Definicja twojego folderu roboczego. Zalecane jest wpisanie tu ścieżki folderu w którym będziesz przechowywał swoje projekty.


4. Rozmiar okna programu.


5. Przechowywane tu są tymczasowe pliki debugger'a. Zazwyczaj nie musisz dokonywać tu jakichkolwiek modyfikacji.



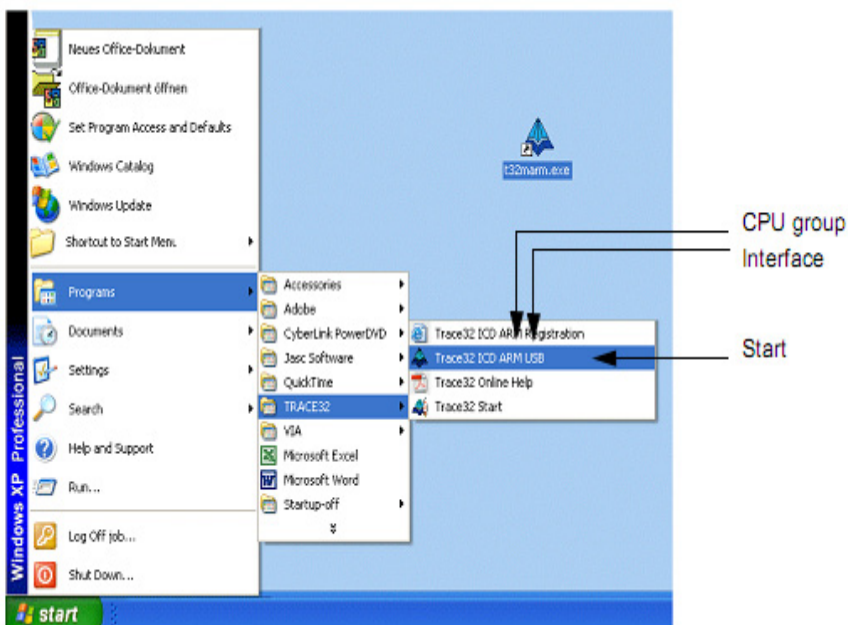
# Uruchomienie TRACE32-ICD

Na samym wstępie uruchom system debugowania, a następnie swój układ docelowy.

	Przy pracy z debugger'em In-Circuit (ICD) niezbędny jest uruchomiony system docelowy!
---	---

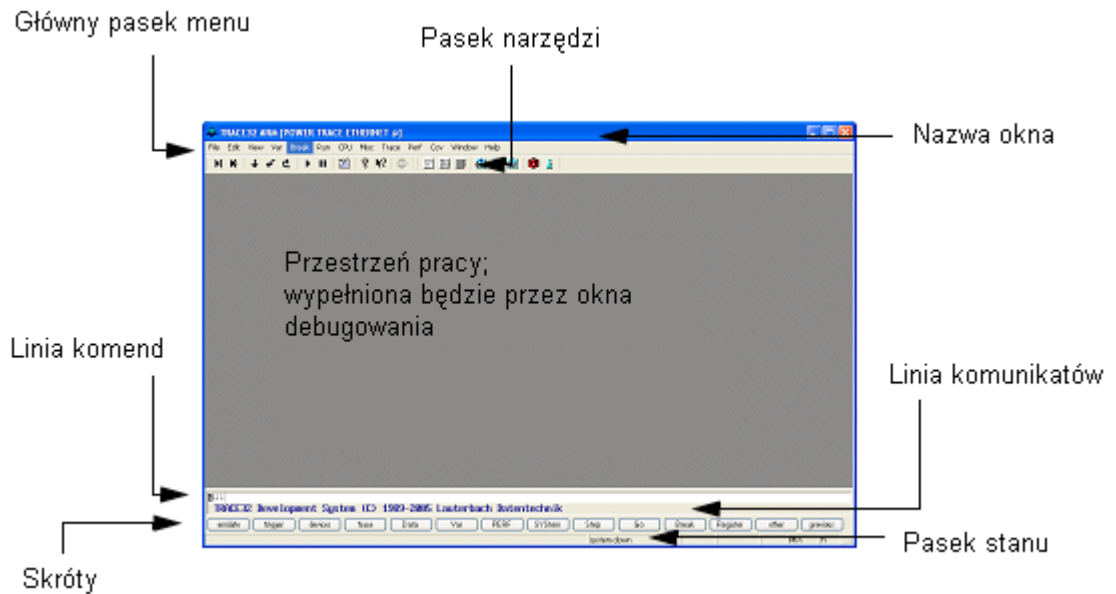
	Zwróć uwagę na prawidłowy przebieg sekwencji włączenia / wyłączenia: <ul style="list-style-type: none"><li>• Włączenie: debugger – układ docelowy</li><li>• Wyłączenie – układ docelowy - debugger</li></ul>
---	--

W celu uruchomienia oprogramowania debugującego na komputerze host, otwórz folder TRACE32 w menu Start i uruchom interfejs użytkownika TRACE32. Jeśli utworzyłeś ikonę programu na pulpicie, kliknij ją dwa razy. Dla poniższego przykładu, zainstalowane zostało oprogramowanie dla procesorów z rodziny ARM.



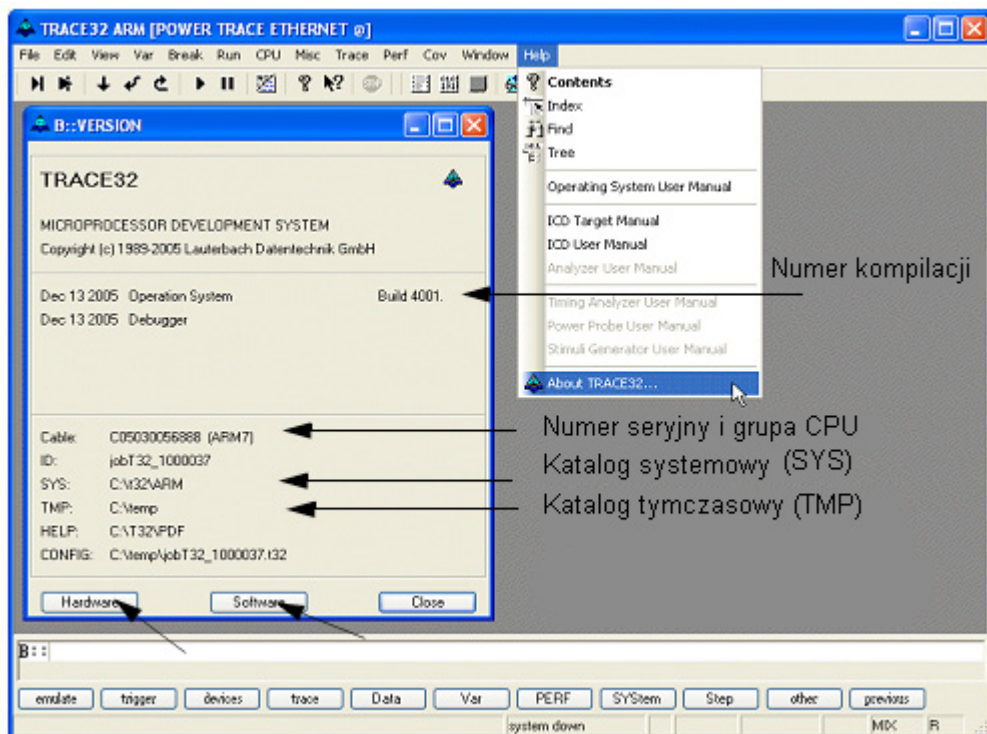
# Główne okno TRACE32

Po uruchomieniu oprogramowania TRACE32 zostanie wyświetlone główne okno debugger'a.



Program TRACE32 jest teraz uruchomiony i gotowy do pracy.

Polecenie About TRACE32 w menu Help dostarcza informacji na temat wersji wszystkich modułów wchodzących w skład oprogramowania TRACE32-ICD.



Naciśnięcie przycisków Hardware lub Software znajdujących się na dole okna spowoduje wyświetlenie dalszych informacji odnośnie zainstalowanego sprzętu i oprogramowania, które są niezbędne przy kontakcie z naszym serwisem. Ponadto, można zauważyć, iż oprogramowanie zostało skonfigurowane zgodnie z rozdziałem 'Konfiguracja środowiska'.

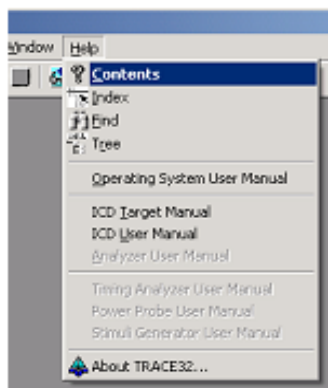
## Online Help

System Online Help zawiera kilkadziesiąt dokumentów, które są dostępne w formacie PDF z poziomu oprogramowania TRACE32 jak i z folderu Help. Specyficzne informacje dla konkretnych CPU umieszczone są w plikach postaci debugger\_<cpu>.PDF. Istnieją trzy dostępne sposoby na uruchomienie systemu Online Help:

- Przycisk Help Topice na pasku narzędzi
- Pozycja Help Contents na pasku menu
- Polecenie HELP wpisane w linii komend



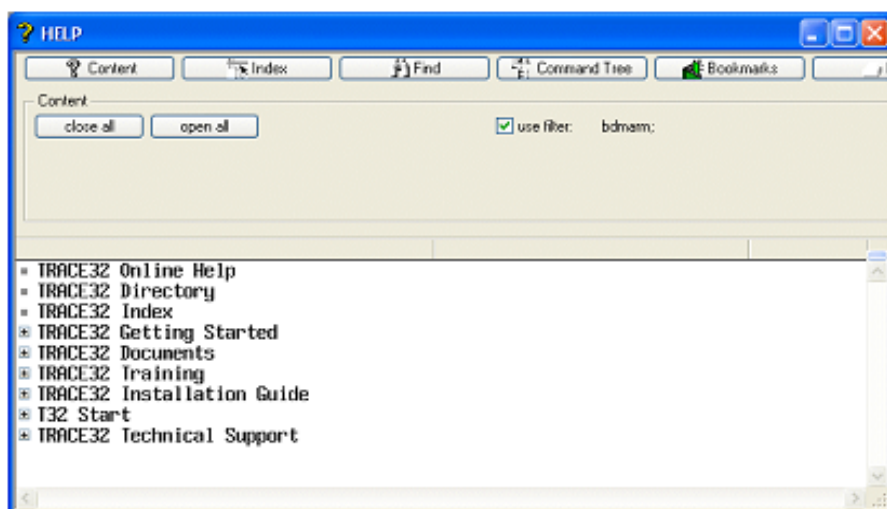
przycisk na pasku narzędzi



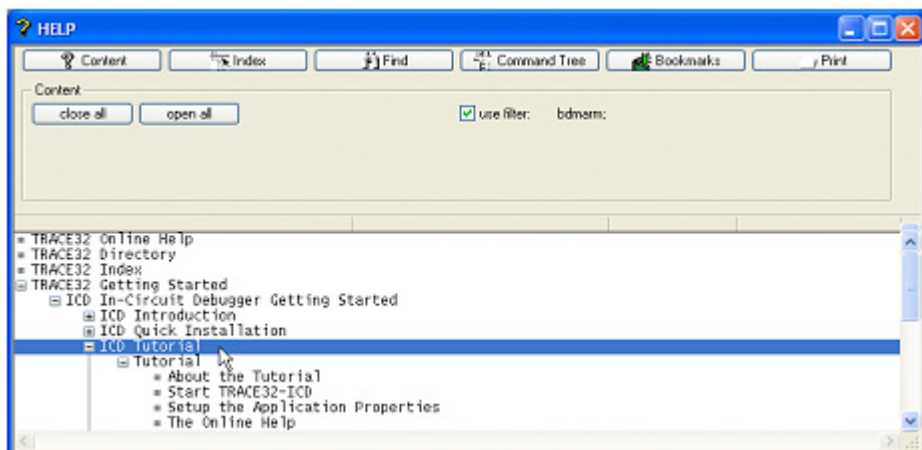
pozycja w pasku menu



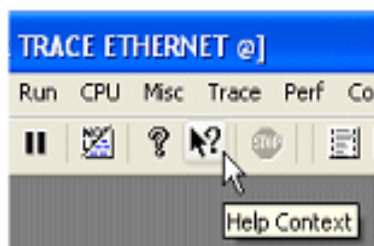
polecenie wpisane w linii komend



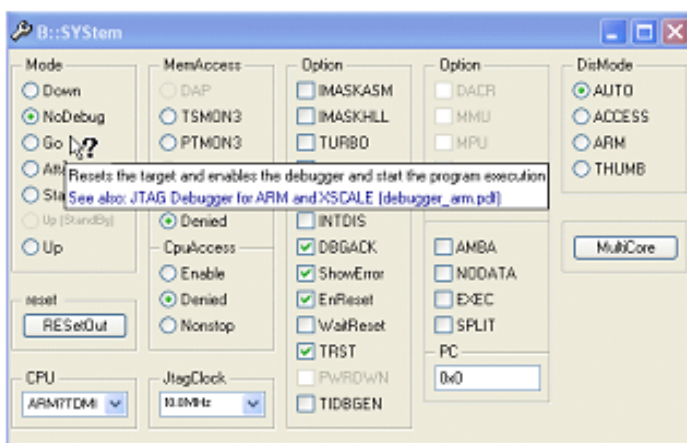
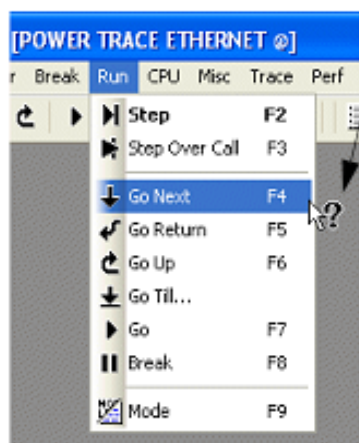
System pomocy zorganizowany jest w postaci wielopoziomowej struktury. Ilustracja poniżej przedstawia sposób dostępu do niniejszego przewodnika.



Oprócz pomocy składającej się z dokumentacji, dostępna jest również pomoc kontekstowa. Dostarcza ona informacji odnośnie specyficznego aspektu środowiska. Dostęp do pomocy kontekstowej składa się z dwóch etapów. Na samym początku należy ją aktywować poprzez naciśnięcie poniższego przycisku:



Kursor zmieni się na znak zapytania. Przesuń teraz kursor na interesujący Cię element, po czym otworzy się ramka z opisem wskazanego obiektu.



Like the command **Go** with a temporary breakpoint set to the next assembler command or next HLL line. This command can be used to overstep a subroutine call instruction or to leave a loop. See also the command **Step.Over**.

See also [Go.Next Continue program](#)



# Ustawienia środowiska Debug

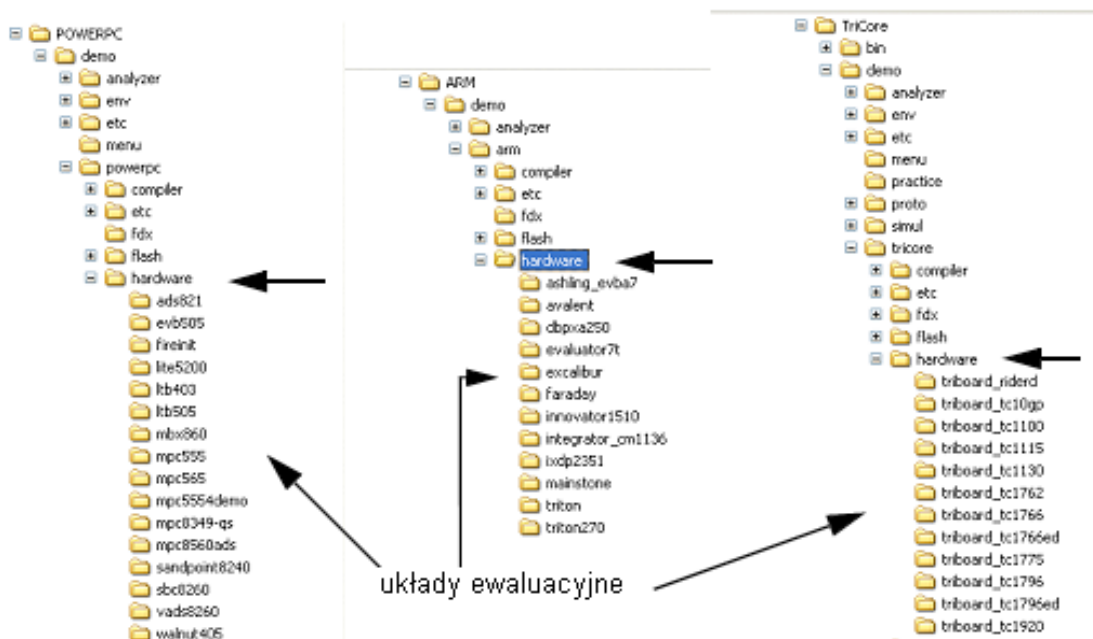
W celu skonfigurowania debugger'a musisz posiadać podstawową wiedzę na temat Twojego procesora oraz konfiguracji układu docelowego. Aby wgrywać do układu docelowego programy zawierające wszystkie symbole i informacje dla debugger'a, musisz także umieć posługiwać się kompilatorem.

Podstawowa procedura konfiguracji i opis specyficznych ustawień CPU dla debugger'a ICD opisana została w dokumencie 'ICD Target Manual'.

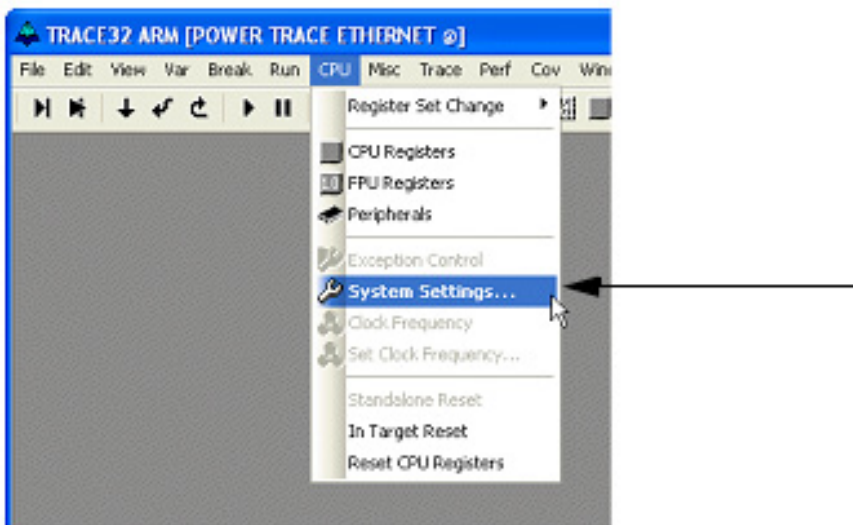
'ICD Target Manual' w szybki sposób daje Ci dostęp do opisu ustawień oraz dodatkowych możliwości Twojego procesora docelowego.

Oprócz tego, wszystkie komendy debugger'ów szczegółowo opisane są w dokumencie 'Referen-ce ICE/FIRE/ICD'.

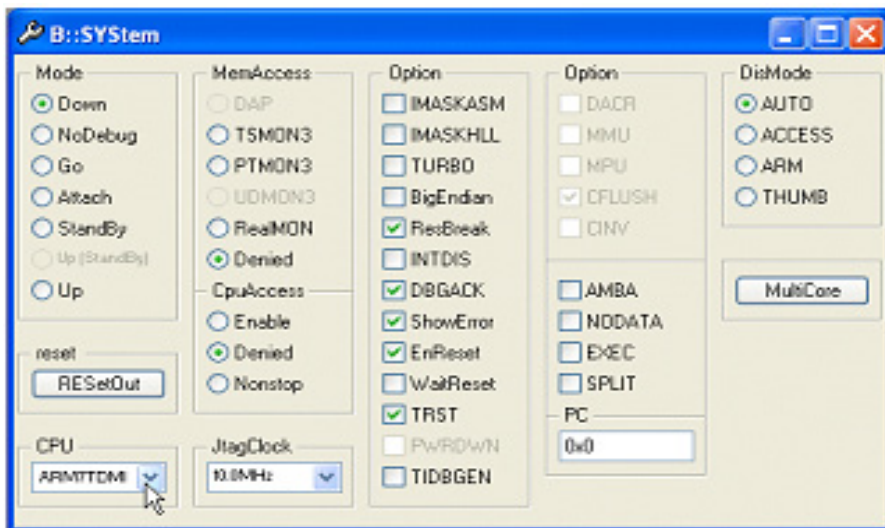
Jeśli używasz układu ewaluacyjnego, konfigurację jego peryferii możesz znaleźć w katalogu demo. Poniższa ilustracja pokazuje kilka przykładów dla procesorów PowerPC, ARM i TriCore.



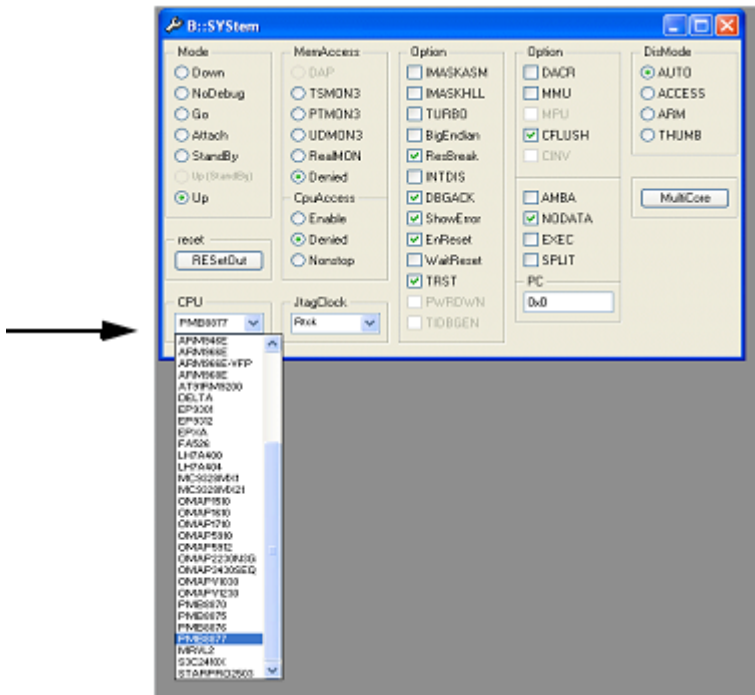
Kolejne kroki przedstawiają typową procedurę konfiguracji debugger'a. W celu demonstracji niezbędnych kroków, przeprowadzimy ręczną konfigurację środowiska, a następnie zaprezentujemy znacznie szybszą metodą polegającą na wykorzystaniu plików wsadowych. Okno SYStem dostarcza wszystkich specyficznych ustawień dla konkretnego CPU. Otwórz okno System Settings w sposób pokazany poniżej:



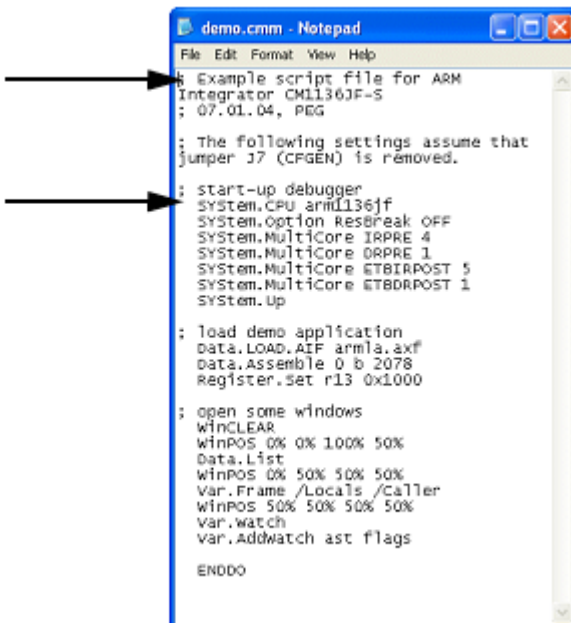
Procesory są zgrupowane w rodziny (np. ARM, TriCore lub PowerPC). Okno System pokazuje wszystkie parametry, które są specyficzne dla konkretnej grupy procesorów. Każdy procesor posiada ponadto zestaw własnych parametrów i ustawień. Poniżej widać okno SYStem dla rodziny procesorów ARM.



1. Poinformuj program TRACE32 o używanym przez Ciebie typie procesora w układzie docelowym, jeśli automatyczne wykrycie CPU nie jest możliwe. Wybierz poprawny typ z listy CPU w oknie SYStem. (Alternatywnie możesz użyć komendy `SYStem.CPU <CPU type>`)



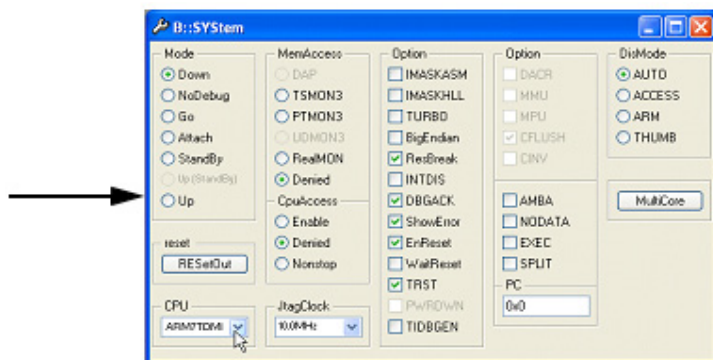
Standardowo, wszystkie dostępne opcje są ustawione w domyślnej konfiguracji. W katalogu demo, który jest dostarczony wraz z oprogramowaniem TRACE32, możesz znaleźć przykłady konfiguracji parametrów różnych systemów.



## 2. Wykonaj inicjalizację uruchomienia i podaj tryb debugowania.

W celu zresetowania procesora z włączonym trybem debug należy wybrać przycisk Up w sekcji Mode w oknie SYStem. (komenda: SYStem.UP)

W tej chwili TRACE32 nawiązuje połączenie z docelowym mikroprocesorem. Czynność ta powoduje reset docelowego CPU, uruchomienie go w trybie debugowania oraz zatrzymanie na wektorze resetu (ang. reset vector). Jeśli podczas tego procesu wystąpi jakiś błąd, zapoznaj się z dokumentacją 'ICD Target Manual'. W przypadku gdy wszystko przebiegnie pomyślnie, powinieneś mieć dostęp do układu docelowego, np. jego procesora lub pamięci.



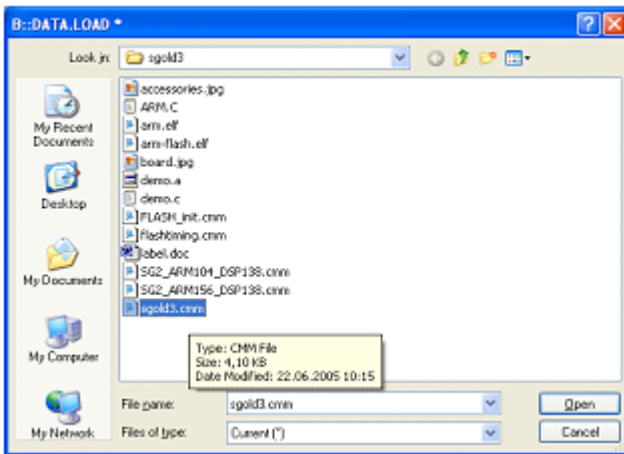
## 3. Wykonaj specyficzną konfigurację układu docelowego.

Procesor jest aktywny lecz zatrzymany. Rejestry są ustawione na wartości domyślne. W kolejnych krokach należy skonfigurować pamięć, poprzez zapisanie rejestrów o specjalnym przeznaczeniu (ang. special function registers) używając komendy PER.Set. Przykładowo, niektóre procesory potrzebują wybrania odpowiedniego chipu w celu dostępu do pamięci. Jeśli używasz płyty ewaluacyjnej, jej firmware może inicjalizować tego typu moduły.

## 4. Wgraj aplikację do układu docelowego.

Następnym etapem jest załadowanie Twojej aplikacji do pamięci układu docelowego. Upewnij się koniecznie czy Twój system ma możliwość dostępu do pamięci, gdyż jest to czynność bardzo specyficzna dla każdego rodzaju CPU. W celu sprawdzenia, spróbuj zmienić wartość jakiejś komórki pamięci korzystając z komendy Data.Set.

załaduj swoją aplikację poprzez komendę Data.LOAD (Data.LOAD.<option> <file\_name>). Opcje, które są wymagane przez Twój kompilator mogą być znalezione w dokumentacji 'ICD Target Manual' w sekcji 'Support/Compilers'. Alternatywnie, możesz użyć komendy Data.LOAD \* i wybrać plik z poniższego okna:

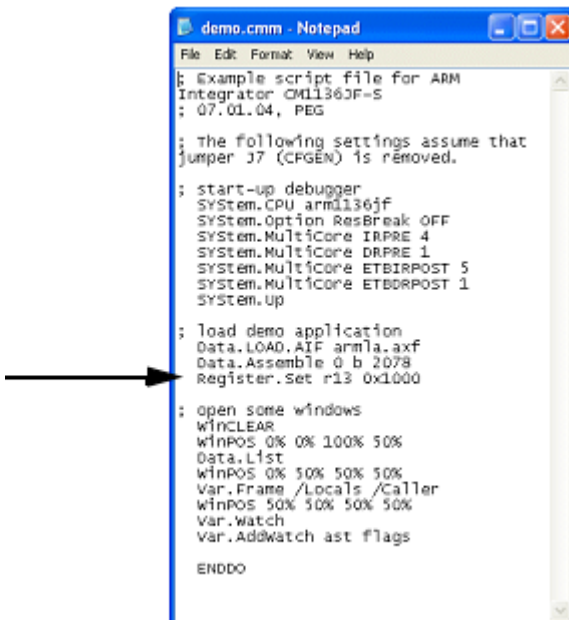


W celu zaprogramowania pamięci typu Flash skorzystaj z grupy komend FLASH oraz dokumentu 'TRACE32 Training / Training ICD In-Circuit Debugger / Training IDC Basics / Flash Programming'.

Aby wyświetlić kod źródłowy skompilowanego programu, musi on być dostarczony wraz z informacjami dla debugger'a (zazwyczaj: opcja kompilatora debug). W tym momencie TRACE32 będzie mógł bezpośrednio pracować z kodem aplikacji.

5. Zainicjuj licznik programu i wskaźnik stosu za pomocą komendy `Register.Set`.

Wiele kompilatorów dodaje automatycznie poniższe ustawienia kodu startowego do programów użytkownika.



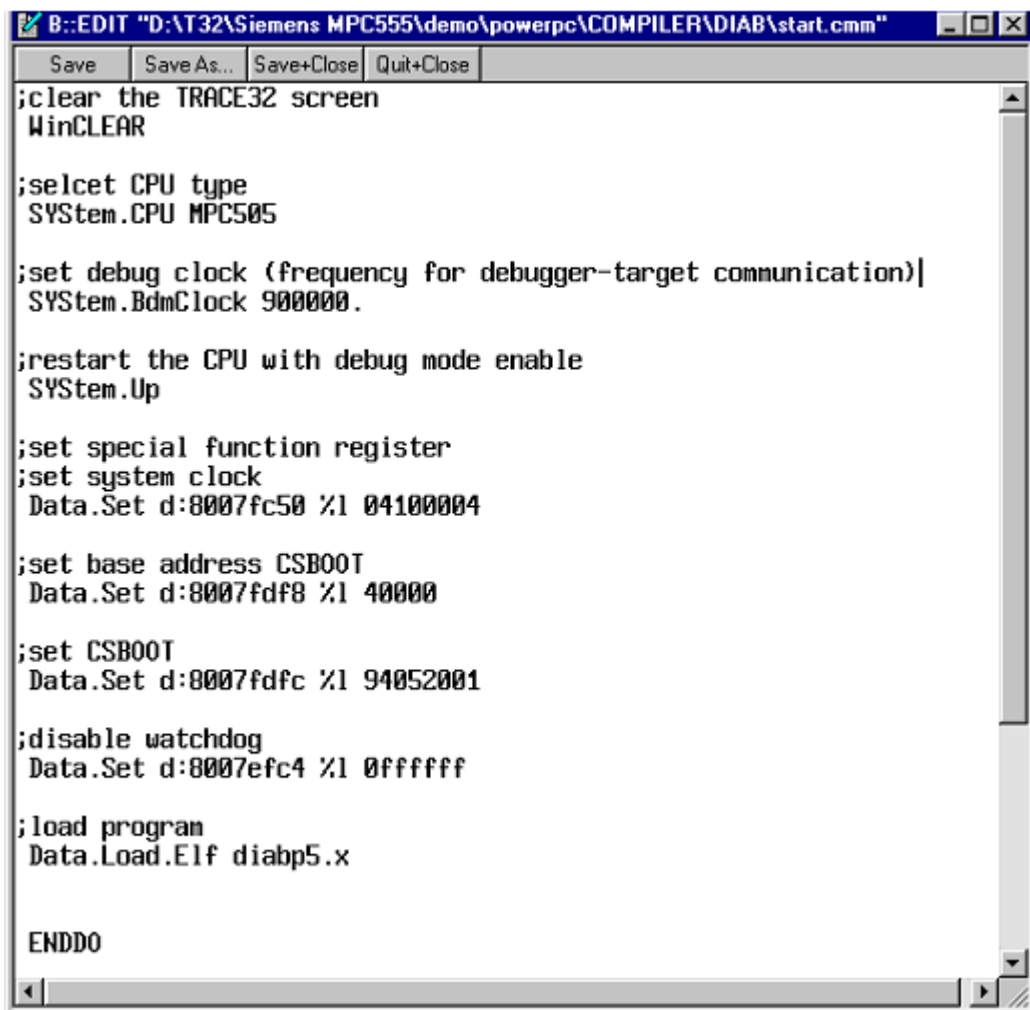
Powyższa sekwencja demonstruje ręczny sposób konfiguracji środowiska debugger'a. Zdecydowanie szybszą i praktyczniejszą metodą jest wykorzystanie plików wsadowych, które wykonują te czynności w sposób automatyczny. W celu zagwarantowania poprawnej procedury konfiguracyjnej, zalecane jest napisanie tego typu pliku. W następnym rozdziale pokażemy jak można tego dokonać.

## Pliki wsadowe

Utwórz nowy plik wsadowy `start.cmm` w katalogu roboczym, poprzez użycie komendy `PEDIT start.cmm`.

TRACE32 posiada własny język skryptowy do wykonywania zadań wsadowych. Nazywa się on `PRACTICE` i posiada bardzo duże możliwości (zobacz 'PRACTICE User's Guide' i 'PRACTICE Reference'). Dozwolone są wszystkie polecenia oprogramowania TRACE32, komendy sterujące wykonywaniem aplikacji, polecenia warunkowe oraz komendy I/O. Domyślnym rozszerzeniem plików wsadowych jest `.cmm`.

Dodatkowo, istnieje możliwość debugowania programów napisanych w języku `PRACTICE`. W celu zasięgnięcia obszerniejszych informacji, przeczytaj dokumenty 'PRACTICE User's Guide' oraz 'PRACTICE Reference' (komendy: `PLIST`, `PEDIT`, `PBREAK`).



```
B:::EDIT "D:\T32\Siemens MPC555\demo\powerpc\COMPILER\DIAB\start.cmm"
Save Save As... Save+Close Quit+Close
;clear the TRACE32 screen
WinCLEAR

;set CPU type
SYStem.CPU MPC505

;set debug clock (frequency for debugger-target communication)
SYStem.BdmClock 900000.

;restart the CPU with debug mode enable
SYStem.Up

;set special function register
;set system clock
Data.Set d:8007fc50 %1 04100004

;set base address CSBOOT
Data.Set d:8007fdf8 %1 40000

;set CSBOOT
Data.Set d:8007fdfc %1 94052001

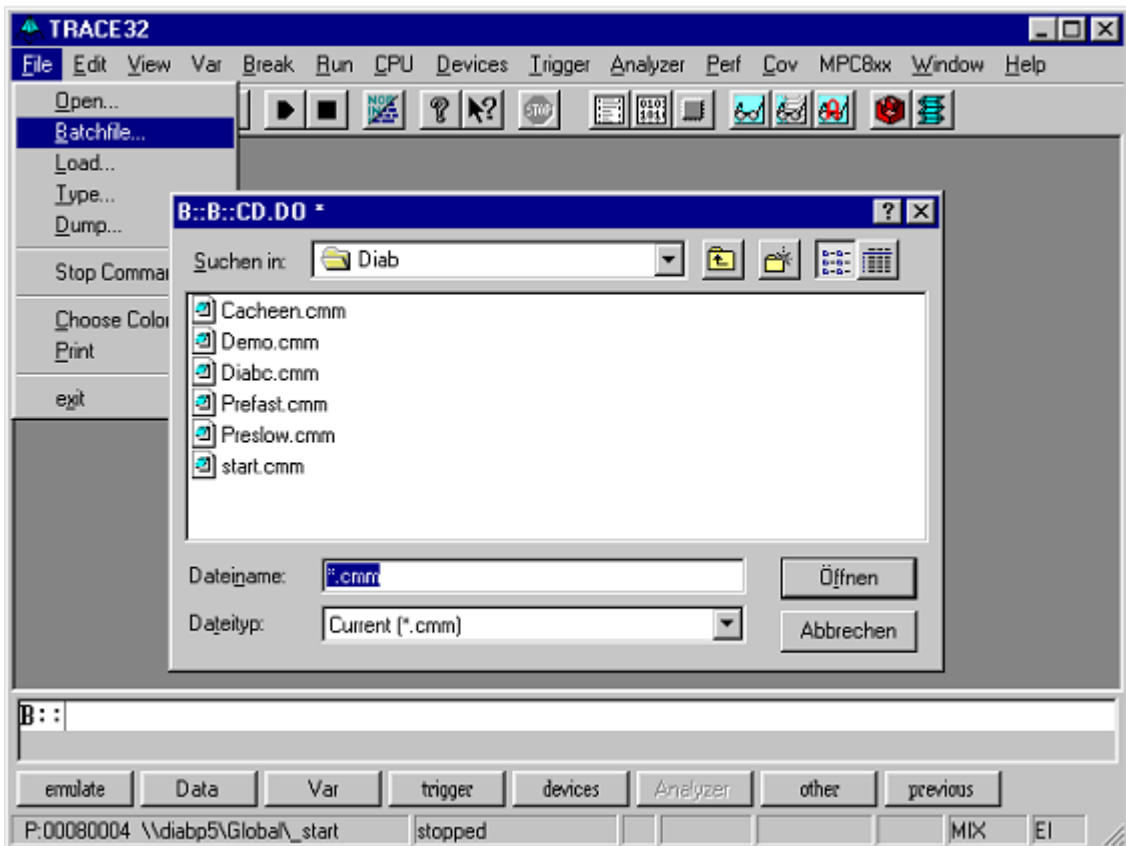
;disable watchdog
Data.Set d:8007efc4 %1 0ffffff

;load program
Data.Load.Elf diabp5.x

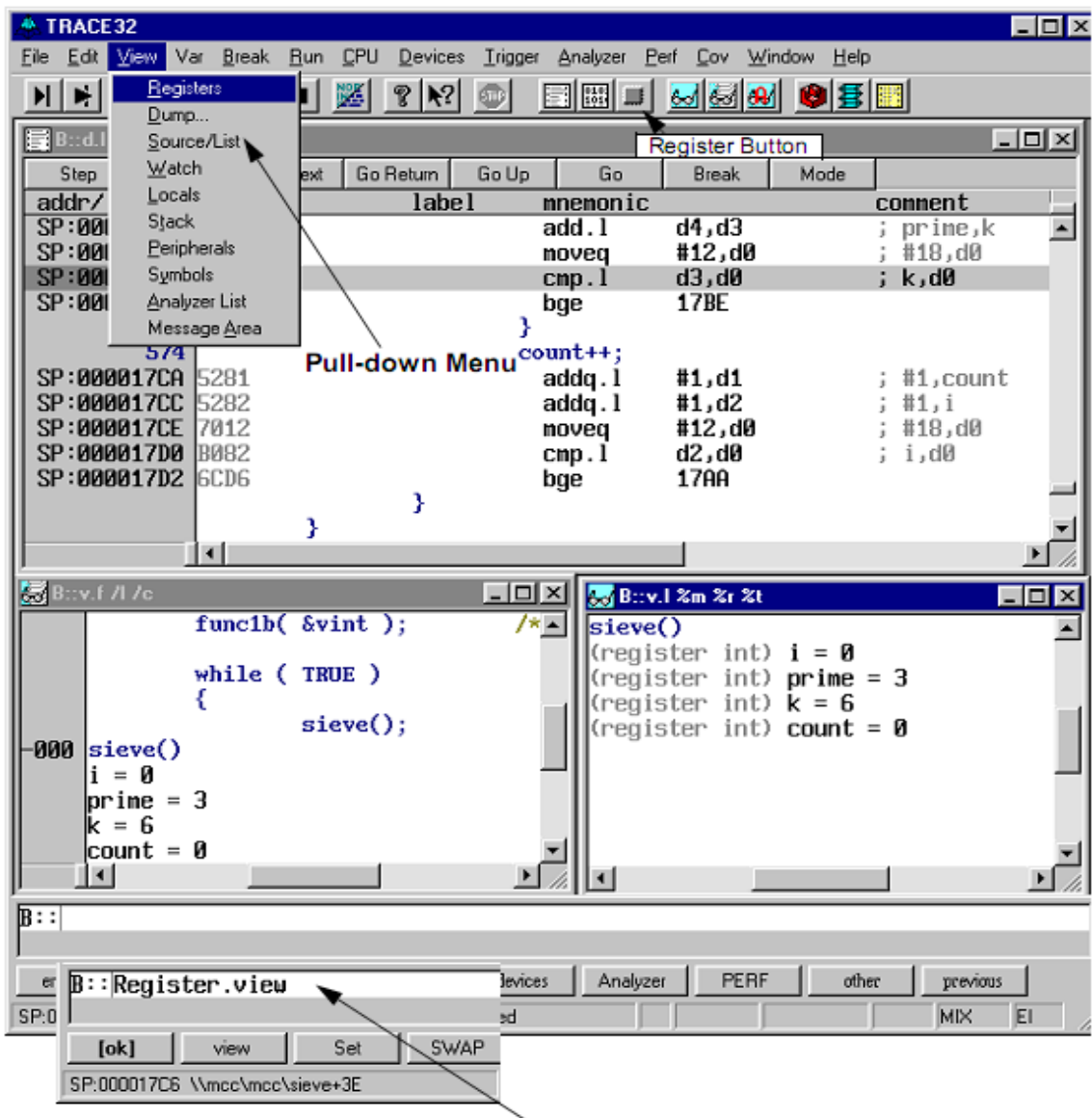
ENDDO
```

Wpisz niezbędne komendy i zakończ skrypt poprzez wpisanie polecenia `ENDDO` i kliknięcie przycisku `Save`. Rysunek powyżej pokazuje przykładową procedurę startową procesora `PowerPC505`.

Rozpoczęcie wykonywania skryptu dokonuje się poprzez wybranie pozycji `Batchfile...` w menu `File`, paska menu.



W celu kontynuowania naszego przewodnika, wybierz jeden przykładowy plik, który możesz znaleźć w katalogu systemowym TRACE32 pod folderem `\demo\ np. \demo\powerpc\compiler\Diab\Diabc.cmm lub wykorzystaj swój własny plik skryptowy, jeśli już taki przygotowałeś.`



Otwórz okno w celu wyświetlenia rejestrów CPU. Alternatywnie, możesz wybrać Register z menu View, nacisnąć przycisk Register lub wpisać Register.view za znakiem zachęty B:: w linii poleceń.

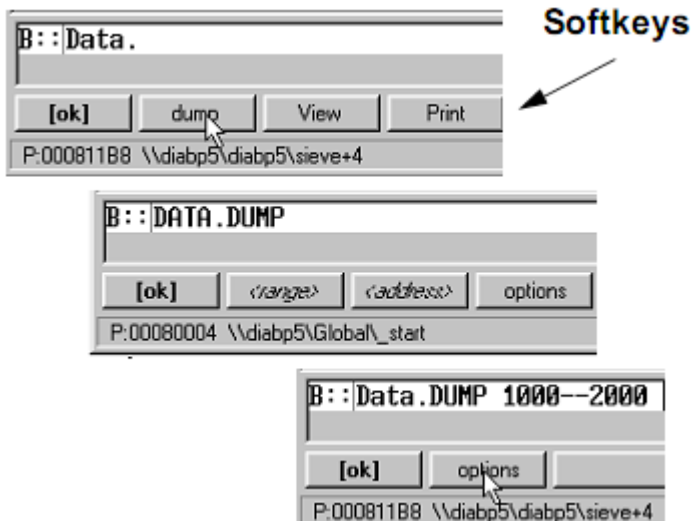
Większość właściwości oraz narzędzi dostępna jest w menu kontekstowym, w głównym pasku narzędzi lub za pomocą wiersza poleceń. Pamiętaj o tych metodach, nawet jeśli w dalszej części tego przewodnika będziemy używać tylko jednej z nich.

Wiersz poleceń TRACE32 nie jest wrażliwy na wielkość znaków. W dostarczonej dokumentacji używamy dużych liter dla znaków, które są znaczące dla wprowadzanych poleceń np. Register.view może być skrócony do formy r. Następnym przykładem, pokazującym typową strukturę poleceń TRACE32 <command\_family>. <subcommand> jest Data.List, który może być skrócony do d.l.



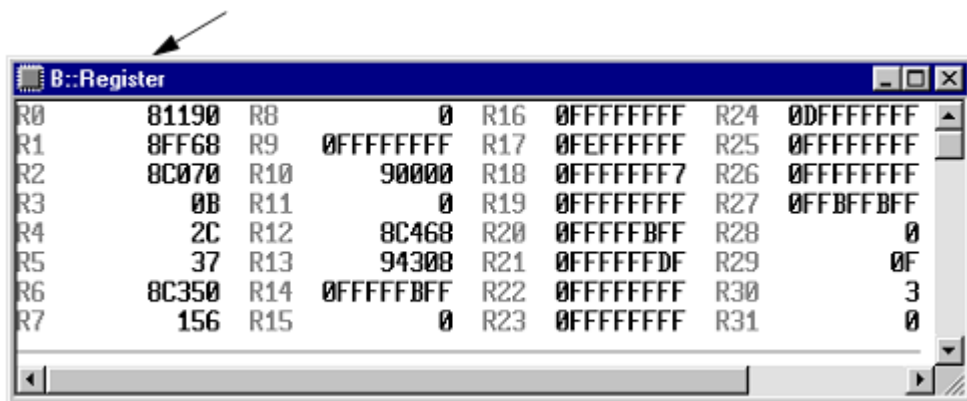
Bardzo poręczną funkcją jest możliwość zastosowania przycisków skrótów (ang. softkeys). Dostarczają one wskazówek dla wpisywanych komend w linii poleceń, prezentując wszystkie możliwe komendy oraz ich parametry. Zamiast wpisywać całe polecenie, możesz zbudować je za pomocą klikania na odpowiednie przyciski.

Przykład: Konstrukcja komendy `Data . dump` za pomocą przycisków skrótów.



Więcej informacji odnośnie interfejsu użytkownika można znaleźć w dokumencie 'Operating System User's Guide'.

Nagłówek okna zawiera nazwę komendy, która wywołała aktualne okno.

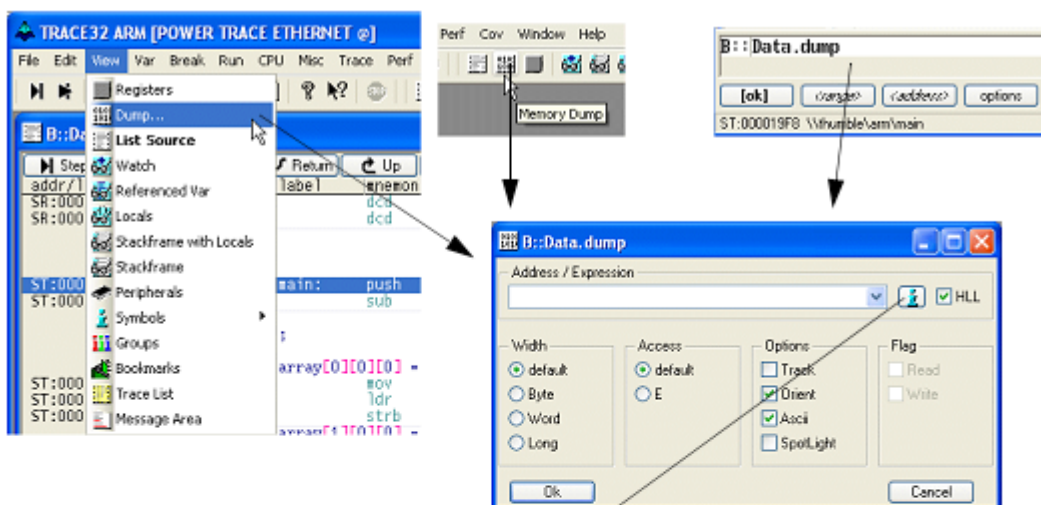


## Podgląd i modyfikacja pamięci

W celu sprawdzenia pamięci o zadanym adresie, należy użyć okna Data.dump w sposób jaki został pokazany w poprzednim rozdziale. Wpisz polecenie `data.dump <zakres adresu>` lub wybierz:

- Select Dump... z menu View.
- Naciśnij ikonę na pasku narzędzi.
- Wpisz polecenie `Data.dump`.

Okno podglądu pamięci zostanie otwarte. Wypełnij pozycje danych w poniższym oknie. Naciśnięcie przycisku Browse umożliwi przeglądanie danych za pomocą symboli. Wybierz etykietkę poprzez podwójne kliknięcie oraz potwierdź czynność przyciskiem OK. Jeśli używasz linii poleceń, możesz bezpośrednio wprowadzić interesujący Cię adres lub symbol.



W celu wyświetlenia informacji HLL wybierz opcję HLL i naciśnij przycisk Browse

Dane zostaną wyświetlone w oddzielnym oknie.

Poniższe zdjęcie prezentuje sposób wyświetlenia danych w pamięci za pomocą linii poleceń.



address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
SD:00006770	47	62	57	E4	A0	70	99	82	6D	09	3D	90	05	38	2F	38	GbW50p33m4=888/8
SD:00006780	87	0C	FF	00	3D	10	FF	40	FF	10	F7	04	EB	10	FF	40	5FFU-1F0F1FEE1F0
SD:00006790	BF	60	FB	00	EF	A8	FF	80	FF	04	FB	00	FD	11	FF	01	5'FNEAF8FEEWF1F5
SD:000067A0	67	B6	BB	30	E9	81	DF	73	BD	58	FD	E2	FA	02	C7	18	g80338=5X02A X03
SD:000067B0	FD	82	FF	00	FE	68	57	00	97	44	FB	52	3F	BC	EE	C4	52FNFHW3DER7EE4
SD:000067C0	FF	24	7F	00	FF	10	BD	00	FE	00	DF	00	EE	D0	FF	02	52FNFHW3DER7EE4
SD:000067D0	BE	00	FF	00	FB	00	FF	00	FD	01	EF	00	FF	20	FF	20	5NFNFNFNFNFNFNFNF
SD:000067E0	2E	14	18	54	BB	00	BF	32	F6	E0	E3	80	5B	EC	9F	04	.11TEU2E556LCE5F
SD:000067F0	9B	33	D9	80	BD	20	7F	10	96	9A	8F	31	27	54	FF	CC	83088-21221' TEE

Klasa pamięci + adres

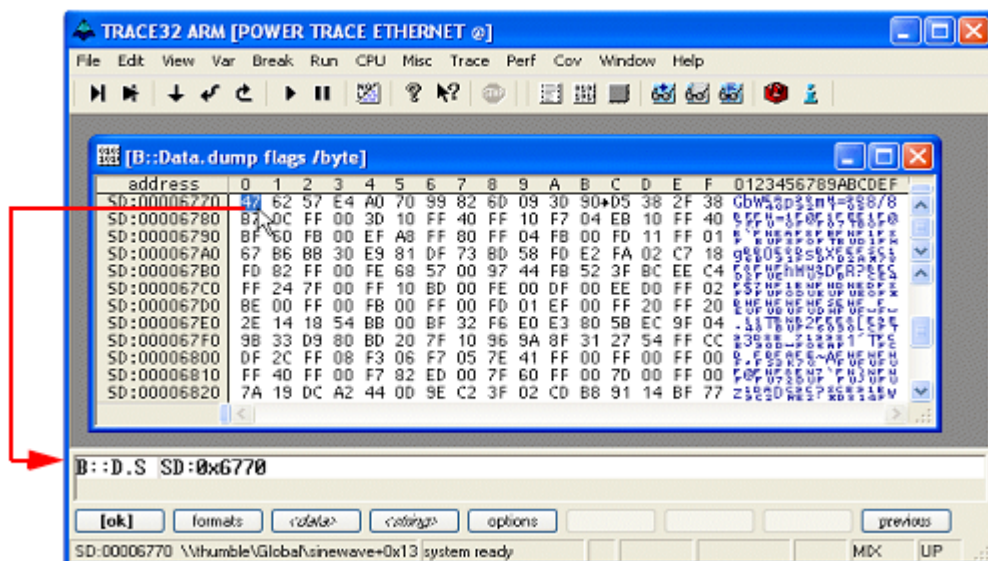
Wartość hex

Wartość ASCII

Istnieją różne sposoby wyświetlenia danych i definiowania zakresu adresów:

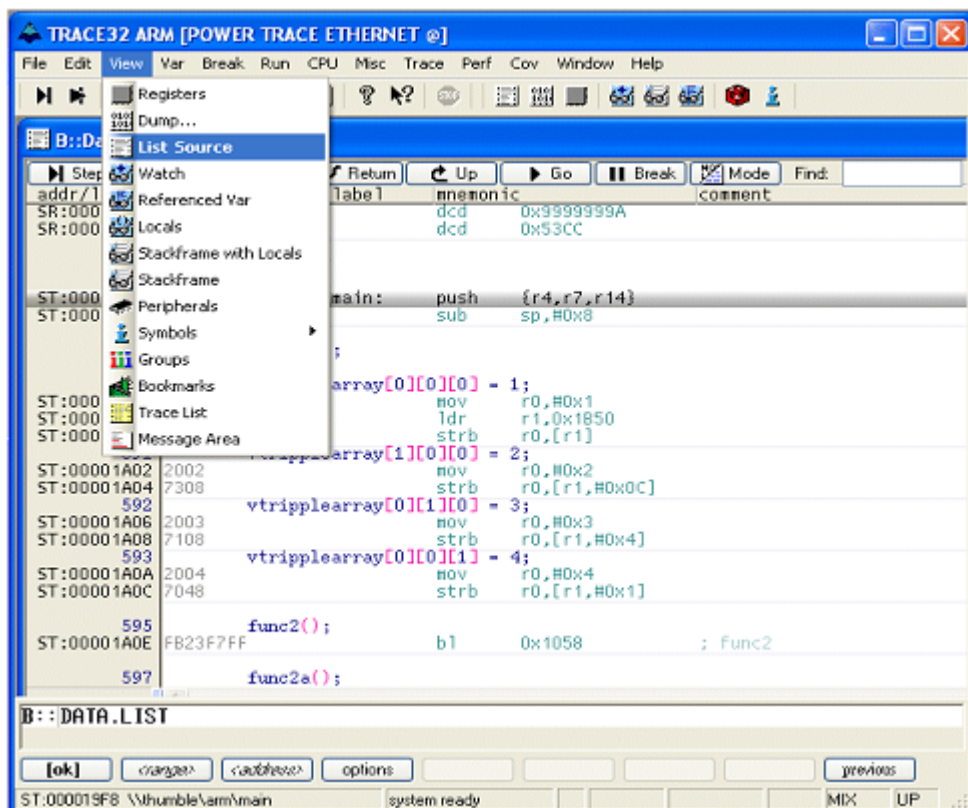
- <adres początku>---<adres końca>
- <adres początku>++<offset>

Wartość pamięci spod zadanego adresu może być zmodyfikowana poprzez podwójne jej kliknięcie. Komenda `Data.Set` dla wybranego adresu wyświetlana jest w wierszu poleceń. Wprowadź nową wartość i potwierdź ją enterem.



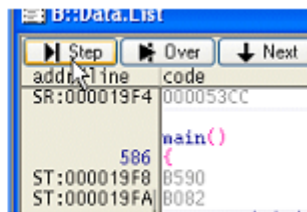
# Debugowanie programu

Otwórz okno Data.List poprzez wybranie pozycji List Source w menu View. Listing kodu źródłowego w obrębie licznika programu zostanie wyświetlony.



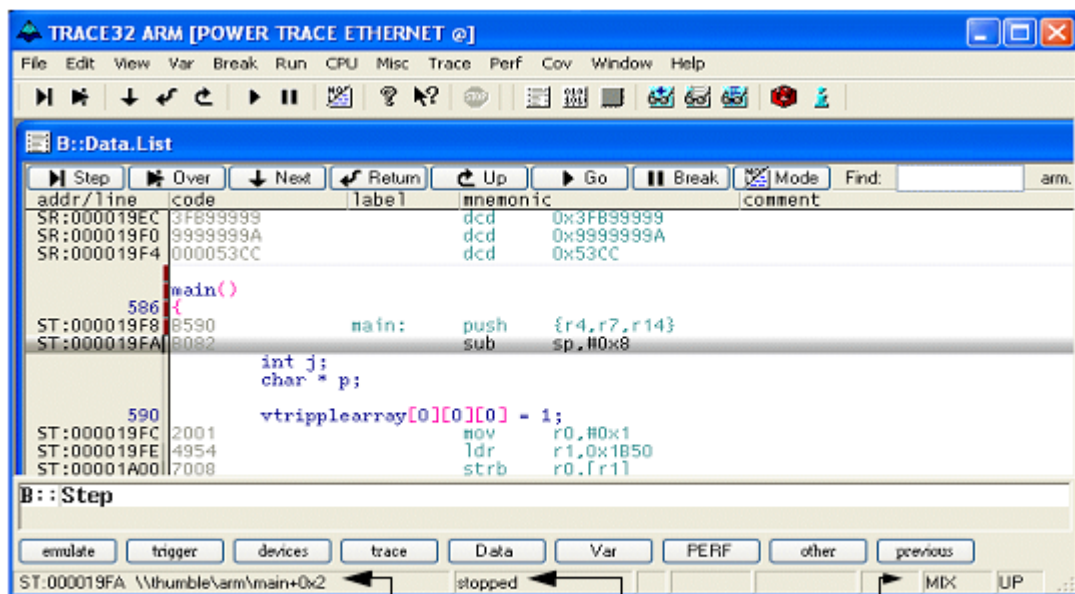
Wykonaj pojedynczy krok, klikając na jeden z następujących elementów:

- pozycja Step w menu Run
- <F2>
- przycisk Step na pasku narzędzi
- przycisk Step w oknie Data.List
- komenda Step wprowadzona w wierszu poleceń



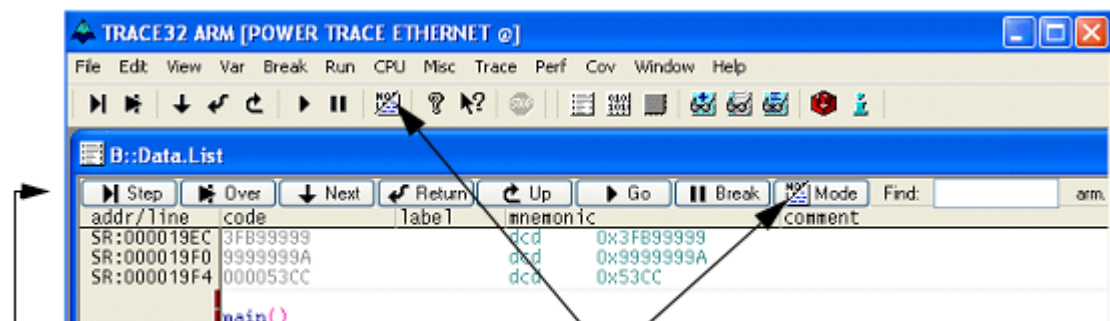
Teraz spojrzysz na pasek statusu. Adres aktualnej pozycji kursora (szary pasek w aktywnym oknie) jest wyświetlony.

Następne pole prezentuje obecny stan debugger'a: stopped oznacza, że Twój program jest zatrzymany. W tym momencie możesz np. podejrzeć lub zmienić jego pamięć.



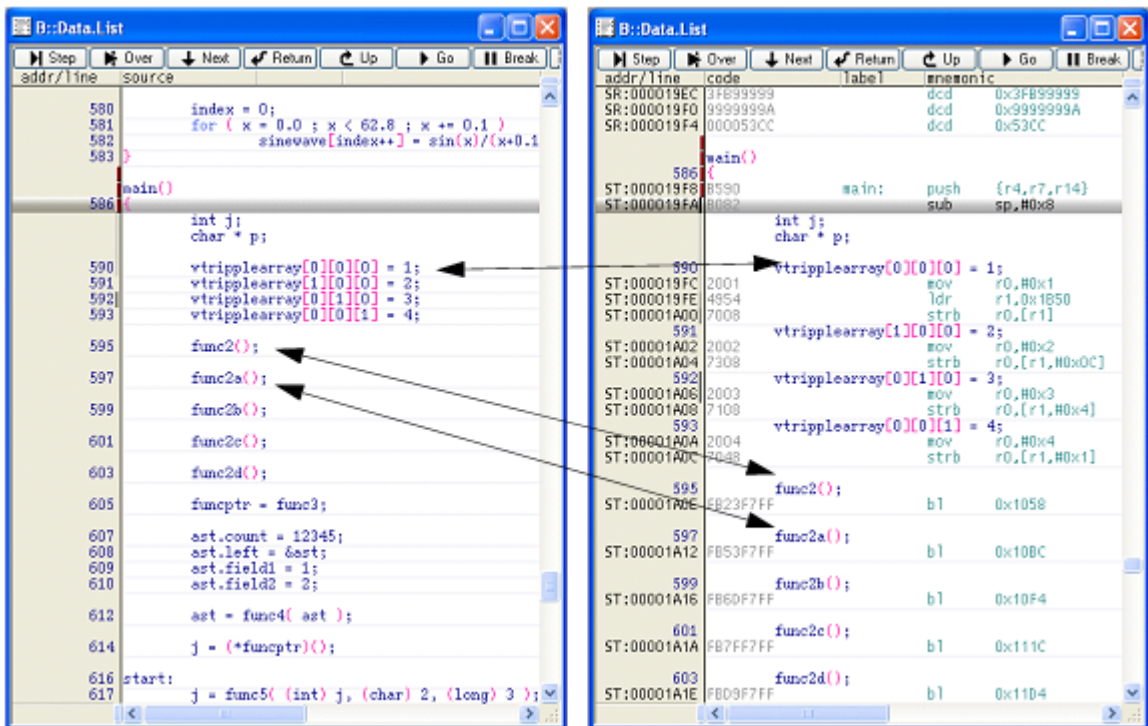
(symboliczny) adres pozycji kursora      stan debugger'a      tryb debug

W zależności od użytego skryptu startowego, wyświetlanie kodu źródłowego będzie w postaci HLL (High Level Language) lub mieszanej (HLL i odpowiadające jemu mnemoniki asemblera). Pole trybu debug na pasku stanu sygnalizuje aktualnie używany tryb. Naciśnięcie przycisku u góry okna Data.List zmienia sposób wyświetlania kodu z HLL na mieszany i odwrotnie. Pasek stanu zawsze pokazuje tryb debug.

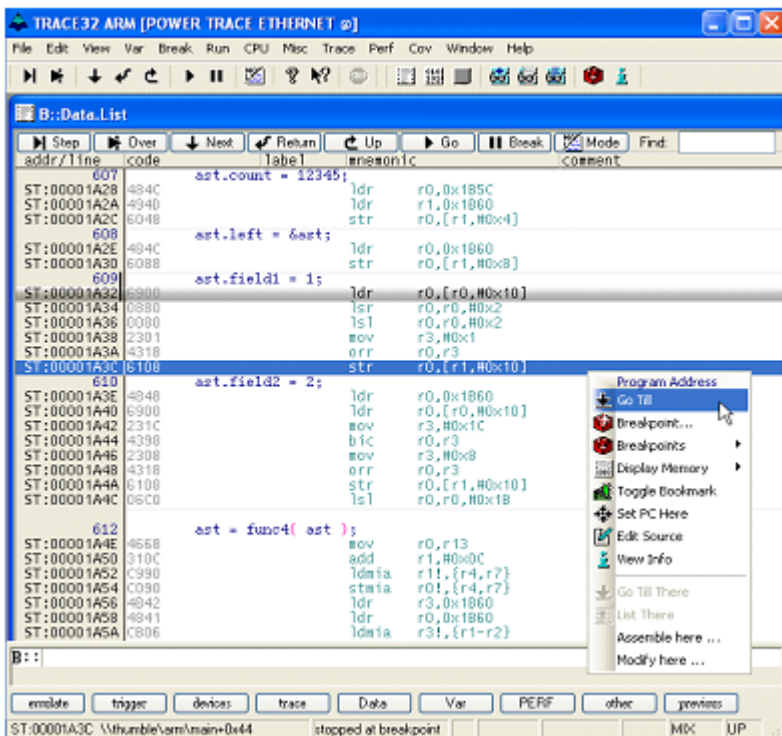


Przyciski lokalne dla okna Data.List      Przełącznik trybu debug

Podgląd w trybie HLL i mieszanym:

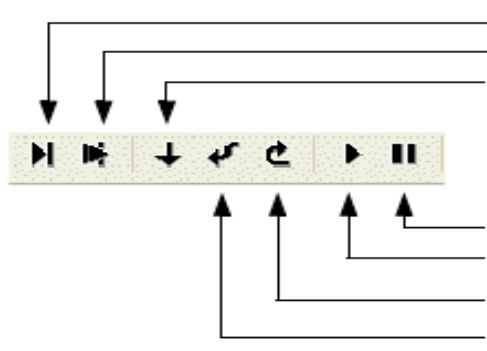
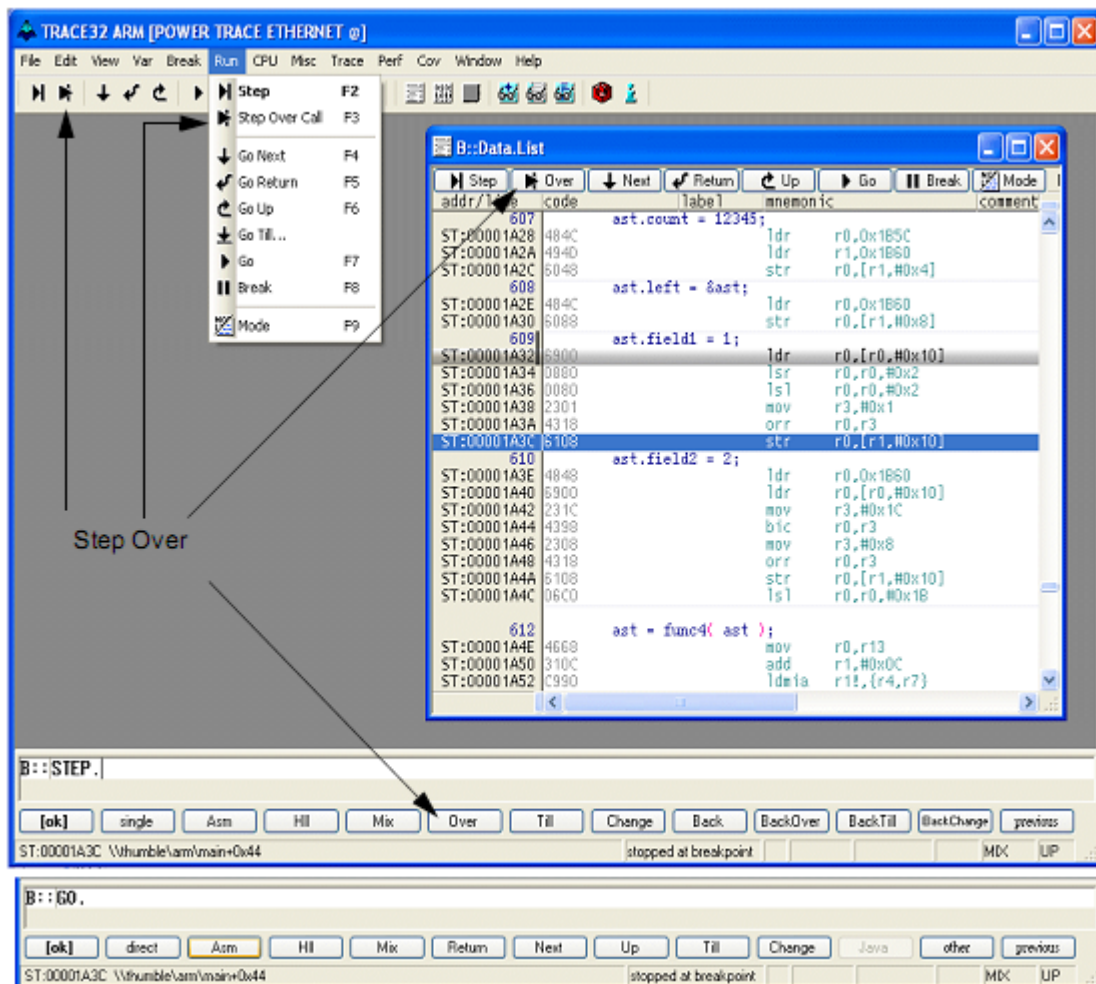


Przełącz tryb debug na HLL i wykonaj następny krok. Krok który właśnie wykonałeś został przeprowadzony w języku wysokiego poziomu, aż do następnej jego instrukcji. Jeśli przełączysz się znowu na tryb mieszany i naciśniesz przycisk Step to wykona się jedna instrukcja asemblera.



Wybierz linię kodu i naciśnij prawy przycisk myszy. Jeśli wybierzesz pozycję Go Till, wykonywanie programu zostanie wznowione, aż do momentu, kiedy program osiągnie wskazaną przez Ciebie linię kodu.

Praca krokowa jest jedną z podstawowych metod analizy i debugowania kodu. W celu zapoznania się z innymi metodami uruchamiania programu, przyjrzyj się pozycjom w menu Run, przyciskom w oknie Data.List oraz głównemu pasku narzędzi.

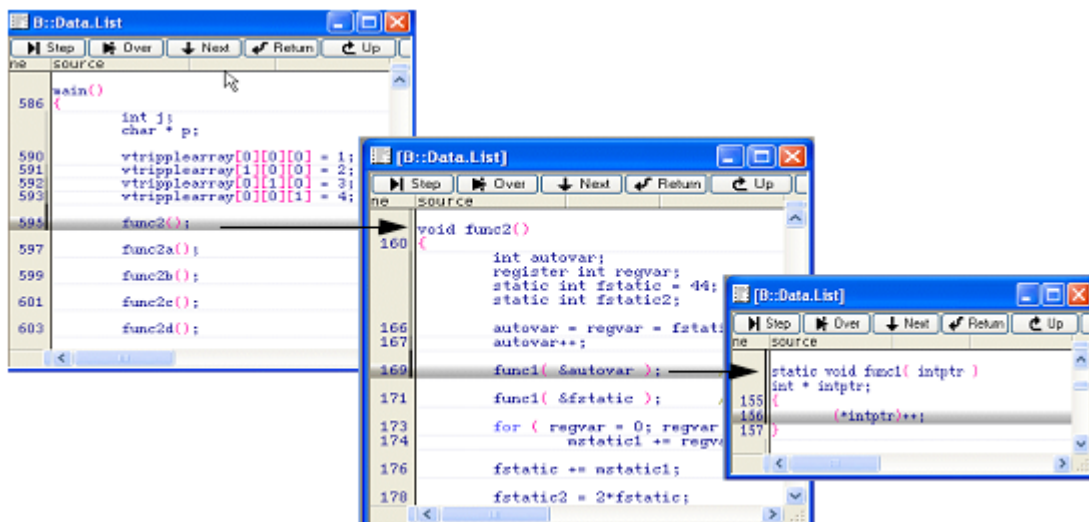


Pojedynczy krok  
Krok ponad wywołanie funkcji  
Idź do następnej linii kodu w listingu programu. Użyteczne np. do opuszczania pętli

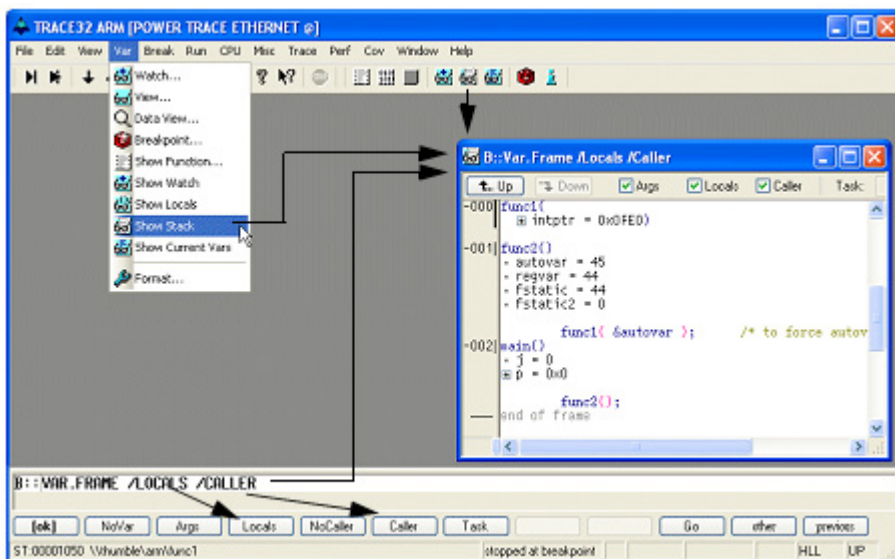
Przerwij / Zatrzymaj wykonywanie w czasie rzeczywistym  
Idź / Rozpocznij wykonywanie w czasie rzeczywistym  
Idź w górę / wróć do funkcji wywołującej  
Wróć / Idź do ostatniej instrukcji

Komendy Go next, Go Return i Go Up są dostępne tylko jeśli program jest uruchomiony w pamięci RAM lub jeśli procesor udostępnia punkty przerwań typu on-chip.

Na potrzeby poniższego przykładu, założmy, że posiadamy zagnieżdżenie funkcji gdzie funkcja main wywołuje func2(), a ta wywołuje funkcję func1().



Okno Var.Frame wyświetla zagnieżdżone funkcje w programie. Wykorzystując opcję LOCAL, lokalne zmienne każdej z funkcji będą możliwe do podejrzenia. Kiedy opcja CALLER jest ustawiona, kilka linii kodu w języku C będzie wyświetlone w celu zaznaczenia gdzie dana funkcja została wywołana. Poniższy rysunek prezentuje zagnieżdżenie i sekwencję wywołania funkcji opisanych w tym akapicie.



TRACE32-ICD dostarcza bardziej złożonych poleceń służących do kontrolowania pracy programu. Możliwa jest praca lub wykonywanie krokowe, dopóki jakiś z warunków nie zostanie spełniony. Przykładowo: `Var.Step.Till j>9` wykona poszczególne kroki programu dopóki zmienna `j` nie będzie większa niż 9. Więcej szczegółowych informacji można znaleźć w dokumencie 'Reference ICE/FIRE/ICD' w opisie takich funkcji jak `Step.Change`, `Step.Till`, `Go.Change`, `Go.Till`, `Var.Step.Change`, `Var.Step.Till`, `Var.Go.Change` oraz `Var.Go.Till`.



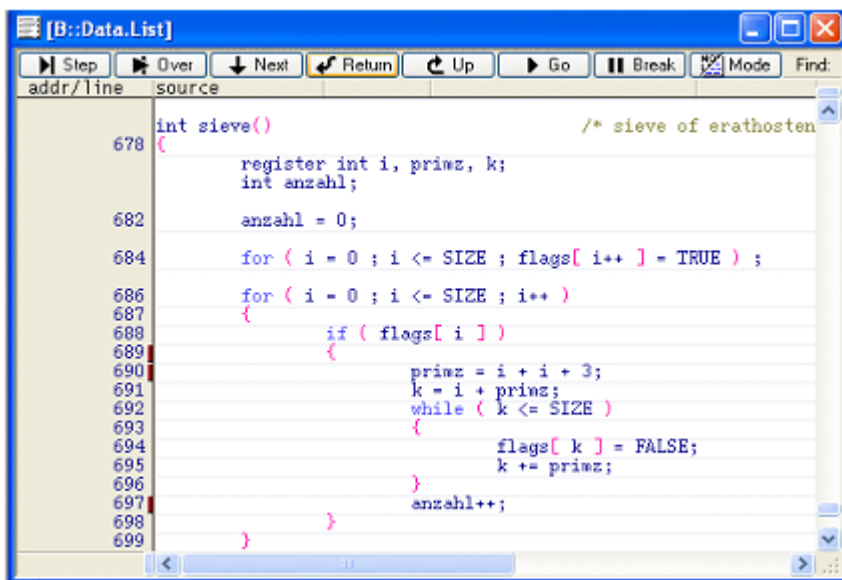
# Jak ustawić punkty przerwań?

## Programowe punkty przerwań

Debugger ICD domyślnie używa programowych punktów przerwań (ang. breakpoint). W momencie kiedy programowy punkt przerwania jest ustawiony na jakimś wyrażeniu, kod w tym miejscu zastępowany jest specjalną instrukcją np. TRAP, która zatrzymuje wykonywany w czasie rzeczywistym program i przekazuje kontrolę do systemu debugującego on-chip. Metoda ta wymaga wolnej pamięci RAM w miejscu zatrzymania!. Jeśli uruchamiasz program w pamięci RAM, ilość programowych punktów przerwań jest nieograniczona.

W przypadku gdy Twoja aplikacja nie jest uruchamiana w pamięci RAM, zapoznaj się z rozdziałem 'Punkty przerwań w pamięci ROM, Flash, EEPROM'.

Wróćmy do naszego przykładowego programu. Kliknij dwukrotnie na linii kodu w której chcesz ustawić punkt przerwań. Zwróć uwagę, aby nie zaznaczyć pustej linii. Wszystkie instrukcje, w których są ustawione punkty przerwań oznaczone są małym, czerwonym paskiem.



The screenshot shows a debugger window titled "[B::Data.List]". The window has a menu bar with "Step", "Over", "Next", "Return", "Up", "Go", "Break", "Mode", and "Find". Below the menu bar is a toolbar with icons for these actions. The main area is a list of source code lines with addresses on the left. The code is a sieve of Eratosthenes. Red vertical bars on the left side of the code indicate that breakpoints are set on lines 689, 690, 691, 692, 693, 694, 695, 696, 697, and 698. The code is as follows:

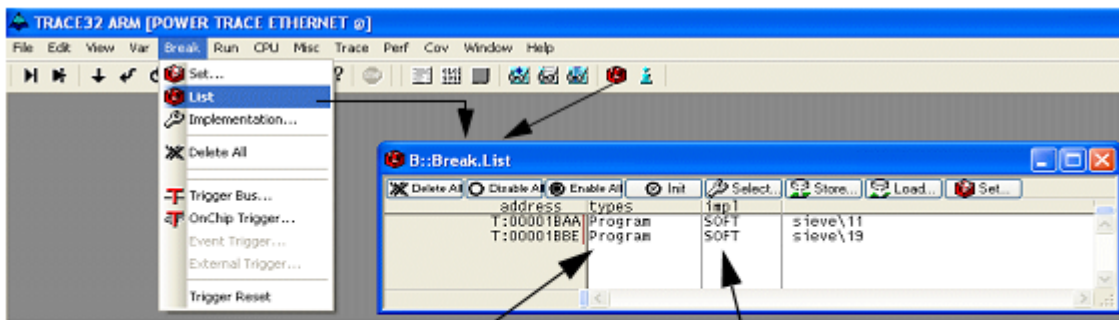
```
int sieve() /* sieve of erathosten
(
    register int i, primz, k;
    int anzahl;

    anzahl = 0;

    for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ;

    for ( i = 0 ; i <= SIZE ; i++ )
    {
        if ( flags[ i ] )
        {
            primz = i + i + 3;
            k = i + primz;
            while ( k <= SIZE )
            {
                flags[ k ] = FALSE;
                k += primz;
            }
            anzahl++;
        }
    }
}
```

Użyj pozycji List z menu Breakpoint w celu wyświetlenia informacji odnośnie wszystkich używanych punktów przerwań.

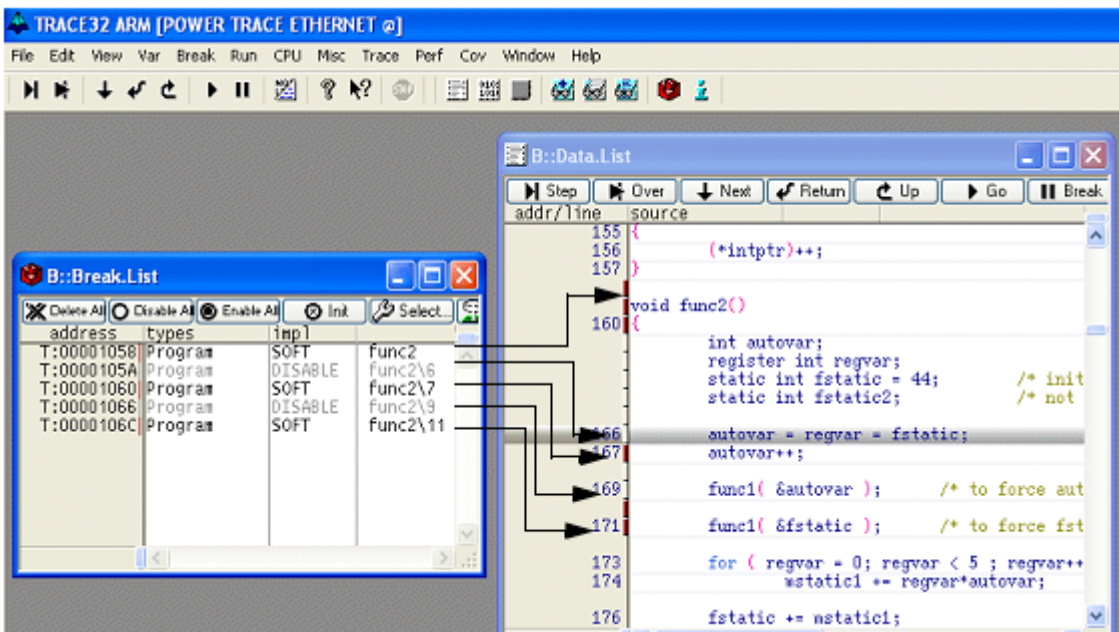


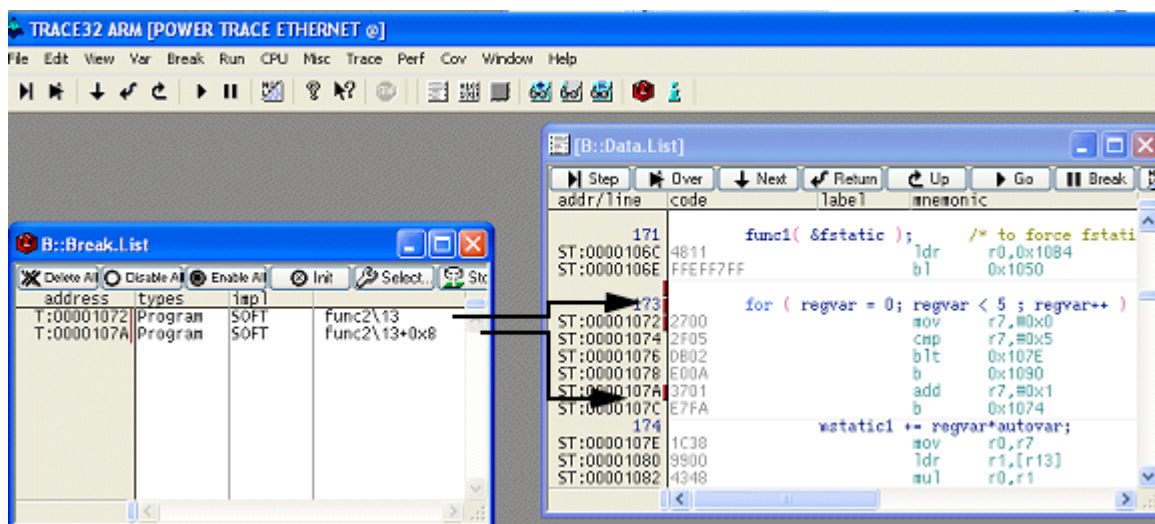
Typu punktu przerwania  
(programow punkt przerwania)

Punkt przerwań ustawiony  
jest jako programowy

Format listy punktów przerwań jest następujący:

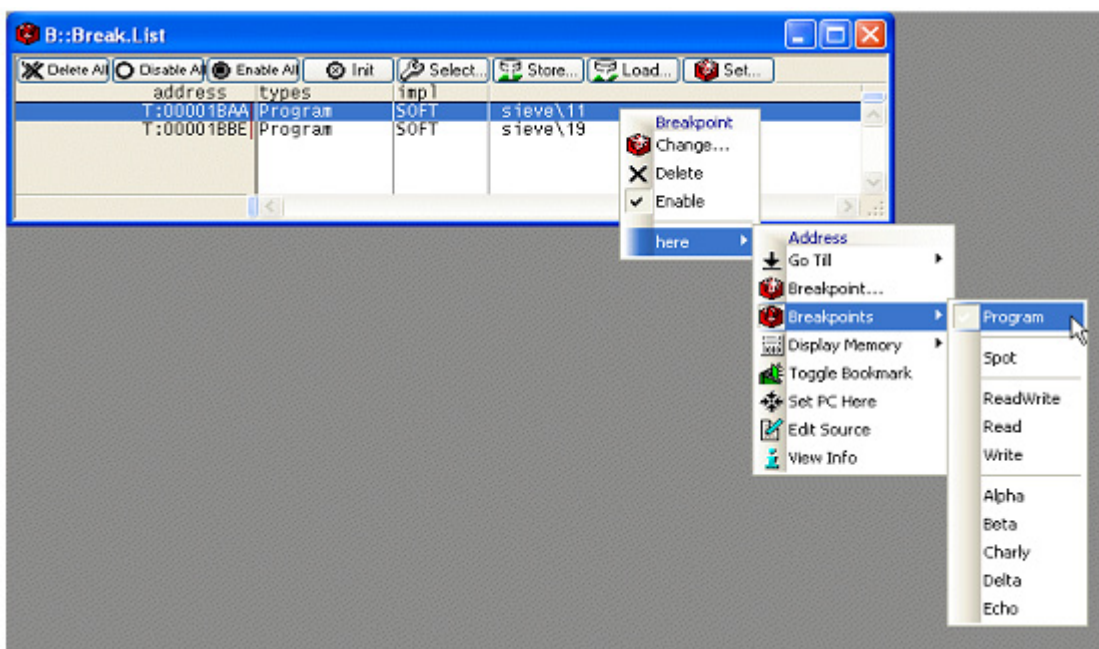
- Kolumna 1: Adres hex punktu przerwania
- Kolumna 2: Typ punktu przerwania
- Kolumna 3: Sygnalizuje sposób realizacji punktu przerwania – programowy (SOFT), ONCHIP, wyłączony (DISABLED).
- Kolumna 4: Przerwana instrukcja. Np. func2\6 oznacza linie 6 kodu HLL w funkcji func2; func2\13+0x8 oznacza linie 13 kodu HLL w funkcji func2 plus 8 bajtów (użyteczne tylko w trybie mieszanym (MIX)).



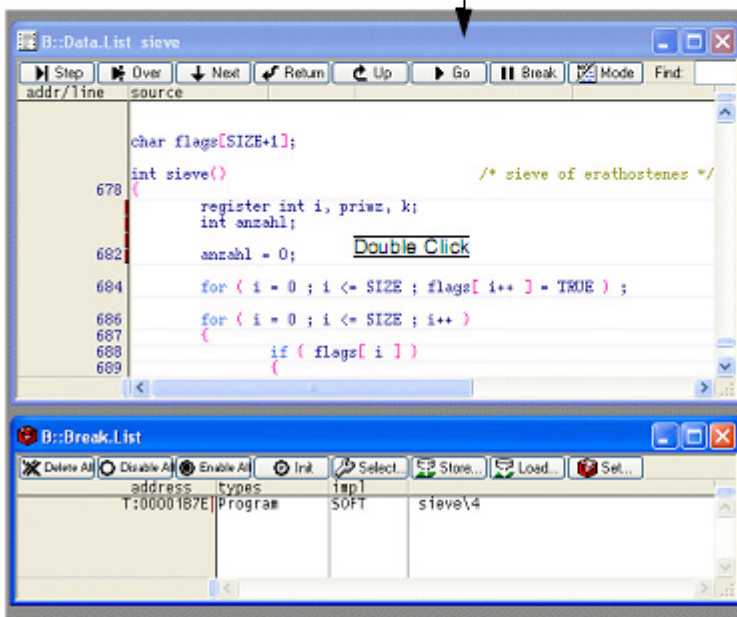
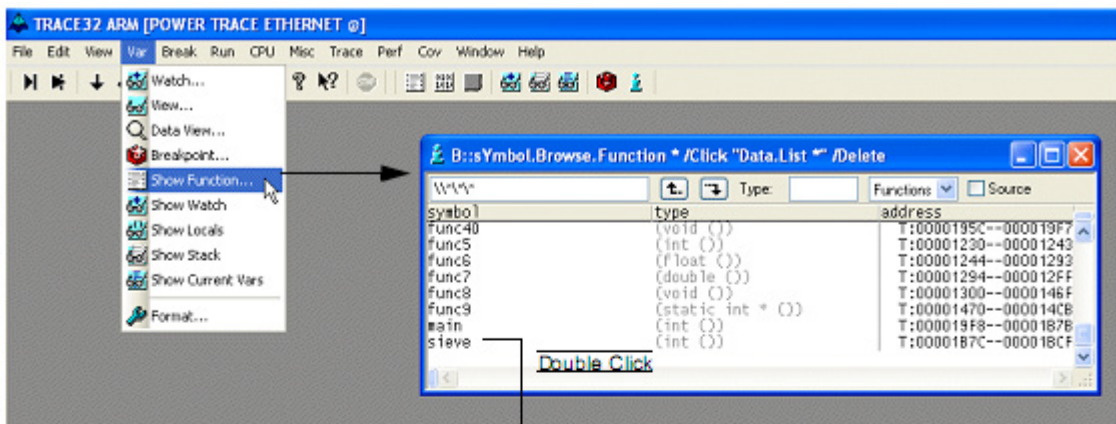


Rozpocznij wykonywanie programu wybierając funkcję Go. Jeśli program nie osiągnie ustawione- go przez Ciebie punktu przerwania, zawsze możesz zatrzymać program używając funkcji Break.

Aby usunąć punkt przerwania należy kliknąć dwukrotnie na oznaczonej linii kodu, lub wyłączyć go w oknie Break.List.

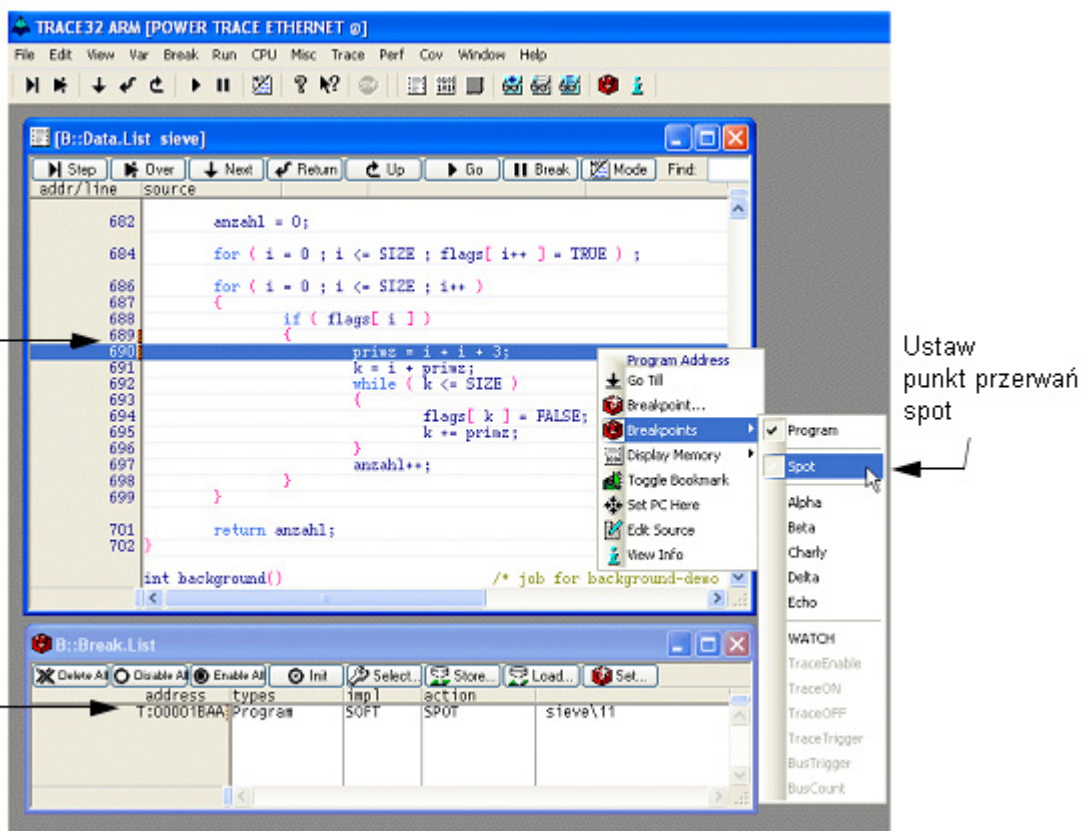


W celu ustawienia punktu przerwania na linii kodu, która nie jest aktualnie wyświetlona, można posłużyć się pozycją Show Function... w menu Var. Podwójne kliknięcie na nazwie funkcji spowoduje skok do jej kodu źródłowego, co umożliwi ustawienie punktu przerwania w interesującym nas miejscu.

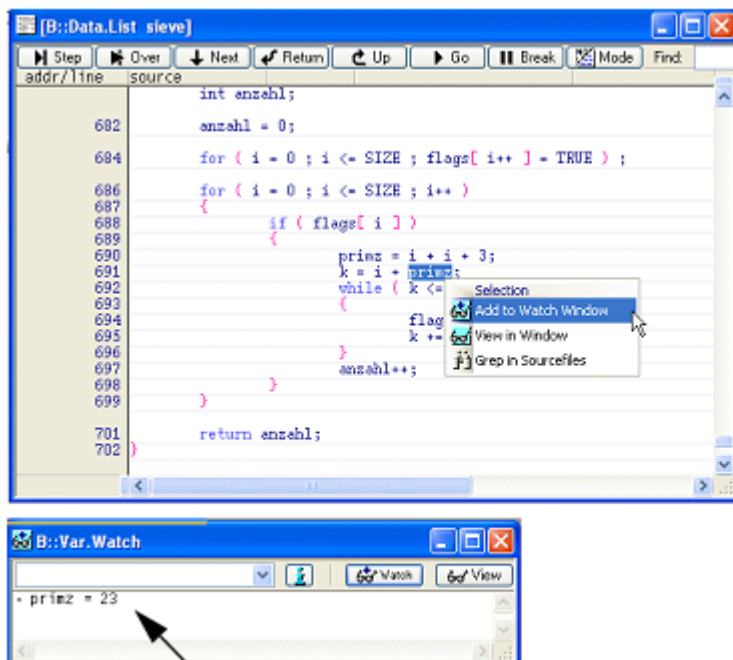


Drugim rodzajem typu punktów przerwań, które są dostępne w wersji programowej to punkty typu spot. Punkty przerwań spot są swoistego rodzaju punktami obserwacyjnymi, które powodują zatrzymanie programu w konkretnym miejscu na krótki odcinek czasu, w celu zaktualizowania wszystkich wyświetlanych informacji. Po odczytaniu danych wykonywanie programu zostaje wznowione.

Aby ustawić przerwanie typu spot, należy zaznaczyć instrukcję w pliku źródłowym, na której chcemy odświeżyć podglądane informacje oraz z menu kontekstowego wybrać pozycję Spotpoint.



W celu podejrzenia wszystkich zmian zmiennej primz, należy zaznaczyć ją, nacisnąć prawy przycisk myszy i wybrać pozycję Add to Watch Window z menu kontekstowego.



Jeśli wznowisz teraz wykonywanie programu za pomocą funkcji Go, w tym miejscu możesz śledzić wartość zmiennej primz

Większość typów procesorów (oprócz 6833x i 6834x) dostarcza kilku sprzętowych punktów przerwań on-chip. Używane są one przez oprogramowanie TRACE32 jako zwykłe punkty przerwań lub typu spot, w momencie gdy aplikacja nie jest uruchomiona w pamięci RAM. Więcej informacji na ten temat znajdziesz w podrozdziale 'Punkty przerwań on-chip'.



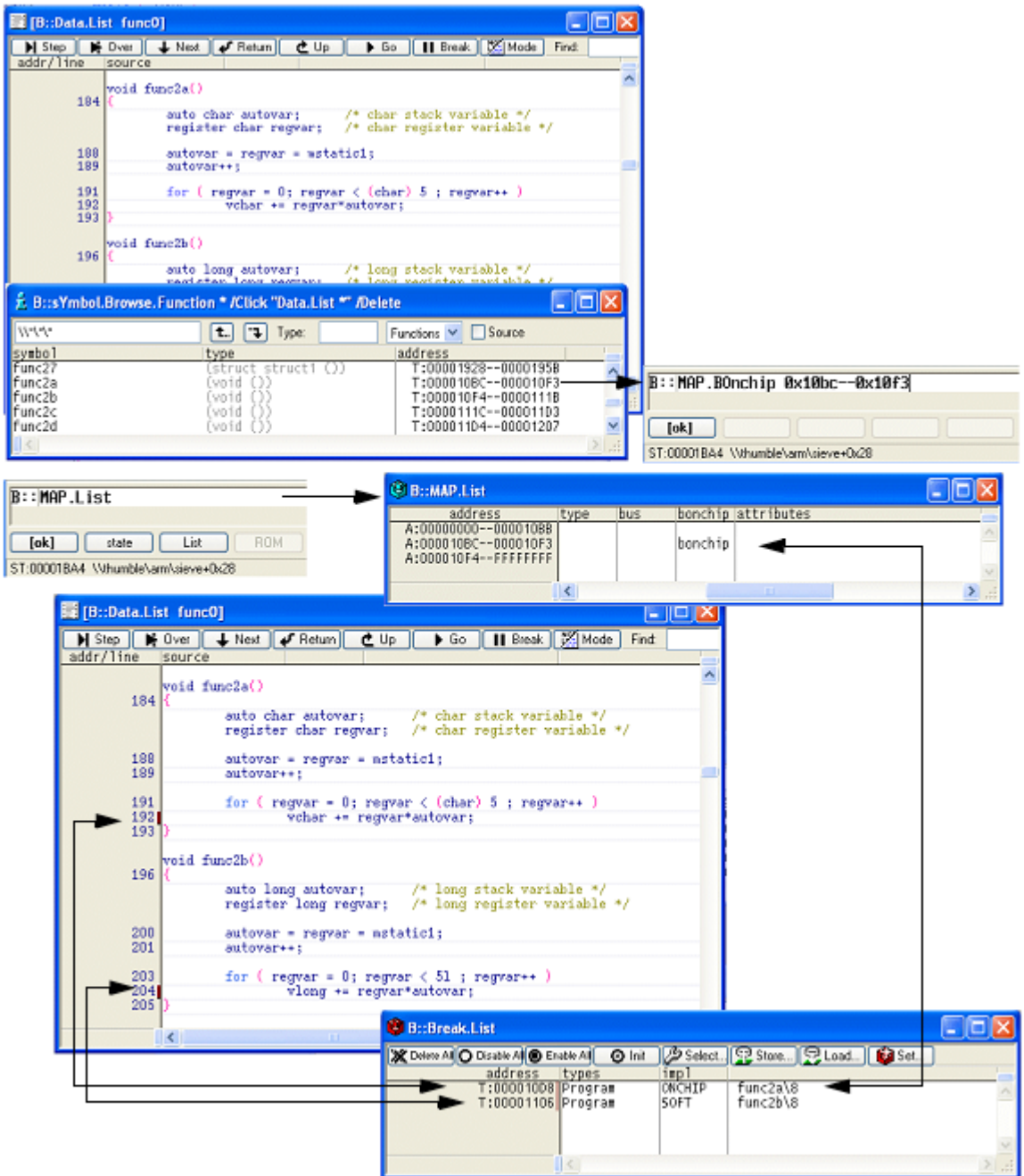
Debugger domyślnie używa programowych punktów przerwań, dlatego musisz zaznaczyć, że chciałbyś wykorzystać ich sprzętową wersję!

```
MAP.BOnchip <zakres_adresów>
```

Komenda `MAP.BOnchip` sygnalizuje, iż wszystkie ustawione punkty przerwań znajdujące się w podanej przestrzeni adresowej powinny być typu *on-chip*.

W poniższym przykładzie zademonstrujemy sposób ustawienia dwóch punktów przerwań - jednego typu on-chip, a drugiego programowego. Załóżmy, że funkcja `func2a` znajduje się w nieulotnej pamięci typu ROM, Flash lub EEPROM, a funkcja `func2b` rezyduje w pamięci RAM. Zgodnie z przedstawioną powyżej zależnością, `func2a` potrzebuje punktów przerwań typu on-chip, podczas gdy `func2b` może korzystać z przerwań programowych:

6. Pobierz przestrzeń adresową funkcji `func2a` i `func2b` (Show Function w menu Var)
7. Przypisz funkcji `func2a` nieulotną pamięć (`MAP.BOnchip`). Aby sprawdzić mapę pamięci użyj komendy `MAP.List`.
8. Ustaw punkty przerwań w funkcjach `func2a` i `func2b`.
9. Sprawdź punkty przerwań wywołując polecenie `Break.List`.



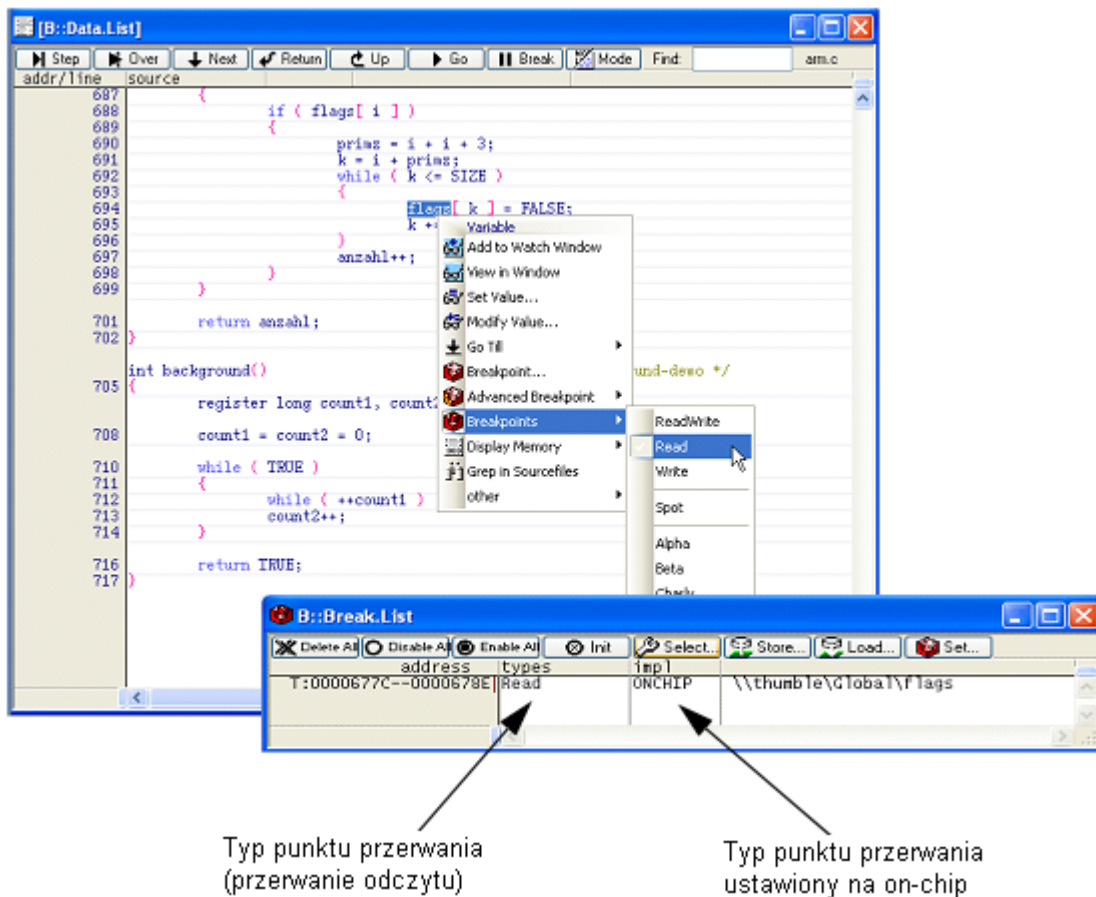
W momencie przekroczenia dopuszczalnej liczby punktów przerwań typu on-chip, zostanie wyświetlony poniższy komunikat. Aby kontynuować pracę, usuń nadmiarowe przerwania.



## Punkty przerwań na dostępie do danych

Dla większości procesorów, możliwe jest zastosowanie punktów przerwań typu on-chip do zatrzymania programu w momencie wystąpienia żądania zapisu lub odczytu danych spod określonego adresu. Więcej informacji odnośnie punktów przerwań on-chip Twojego procesora, znajdziesz w podrozdziale 'Punkty przerwań on-chip (zestawienie)'.

W celu zatrzymania wykonywania programu na odczycie wybranej zmiennej, należy zaznaczyć ją kursorem, kliknąć prawy przycisk myszy i wybrać pozycję Read z menu Breakpoint.

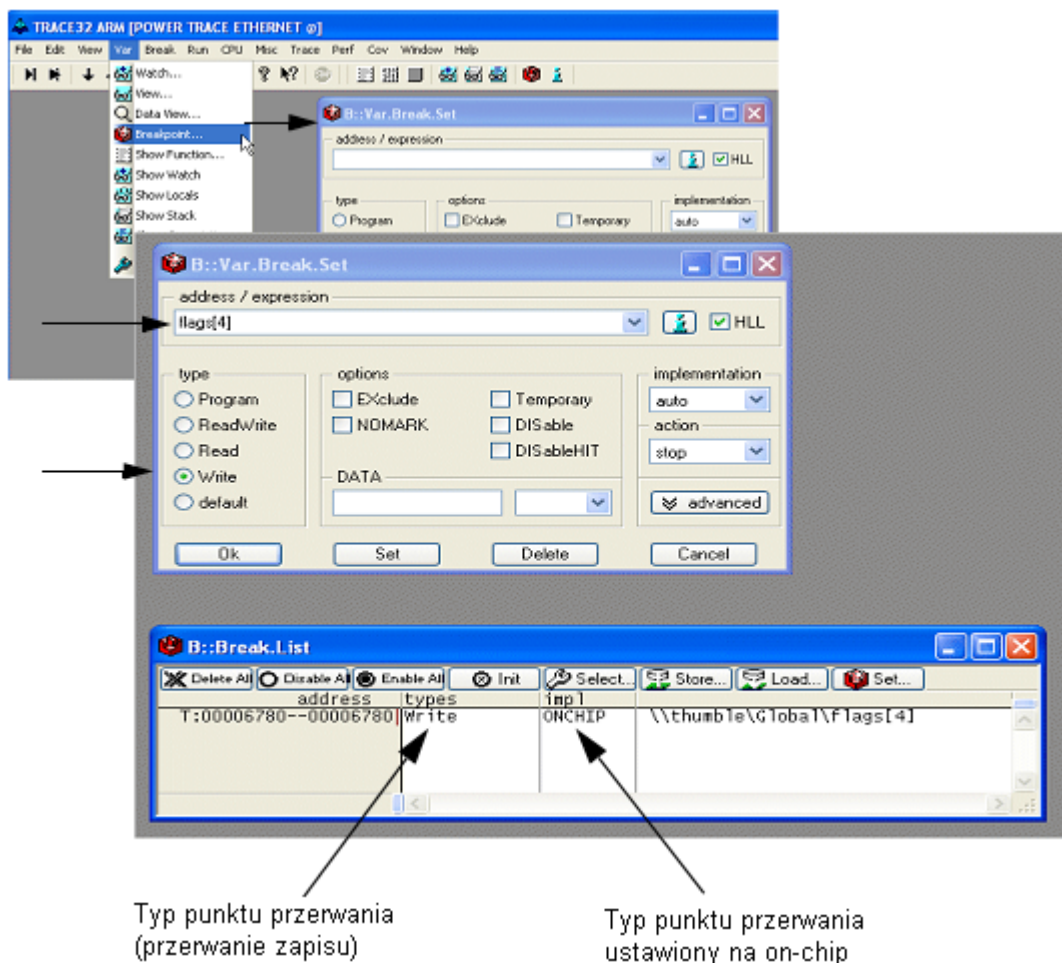


Rozpocznij działanie programu wybierając Go. Jeśli program nie osiągnie Twojego punktu przerwań, możesz go zatrzymać za pomocą funkcji Break.

W celu zatrzymania pracy programu na żądaniu zapisu zmiennej, możesz użyć pozycji Breakpoints... z menu Var.

- Klikając przycisk Browse będziesz miał możliwość wybrania interesującej Cię zmiennej spośród wszystkich występujących w aplikacji. Swoją decyzję potwierdź podwójnym kliknięciem i zaznaczeniem pola Write w oknie Variable Breakpoint Set. Zakończ klikając przycisk OK.
- Alternatywnie, możesz podać nazwę zmiennej lub wyrażenie zapisane w języku typu HLL w polu Expression znajdującym się w oknie Variable Breakpoint Set. Następnie wybierz Write i naciśnij przycisk OK.





Typ punktu przerwania  
(przerwanie zapisu)

Typ punktu przerwania  
ustawiony na on-chip

Rozpocznij działanie programu wybierając Go. Jeśli program nie osiągnie Twojego punktu przerwania, możesz go zatrzymać za pomocą funkcji Break.

Większość dostępnych procesorów dostarcza bardziej złożonych mechanizmów przerywania i wyzwalania pracy programu. Obsługa tych możliwości przez oprogramowanie TRACE32-ICD zależy od typu konkretnego CPU. Więcej informacji na ten temat znajdziesz w dokumentacji 'ICD Target Manual'.

## Punkty przerwań on-chip na różnych architekturach procesorów

Lista znajdująca się na następnym stronie stanowi zestawienie możliwości wykorzystania punktów przerwań *on-chip* dla poszczególnych rodzin procesorów.

Używana jest następująca notacja:

- Punkty przerwań on-chip: całkowita dopuszczalna liczba.
- Przerwania instrukcji: liczba punktów przerwań on-chip możliwa do wykorzystania jako punkty przerywające pracę programu w pamięci Flash/EEPROM/ROM.
- Przerwania dostępu do danych: liczba punktów przerwań on-chip możliwa do wykorzystania jako punkty przerwań w momencie czytania lub zapisu z/do poszczególnego adresu w pamięci.
- Przerwania danych: Liczba przerwań danych on-chip, które powodują przerwanie pracy programu w momencie zapisu lub odczytania konkretnej wartości spod ustalonego adresu pamięci.

### Pojedynczy adres/zakres adresów/maski bitowe

W niektórych architekturach procesora przerwania typu on-chip mogą obejmować tylko pojedyncze adresy (np. ARM11).

Większość rodzin procesorów pozwala na użycie punktów przerwań typu on-chip dla poszczególnego zakresu adresowego. W zależności od technologii użytej w module debugującym, dostępne są następujące możliwości:

- Jedno przerwanie *on-chip* może obejmować pojedynczy adres lub zakres adresów (np. Tri-Core).
- Dwa przerwania *on-chip* używane są w celu oznaczenia zakresu adresowego. Jedno przerwanie ustawiane jest na adresie początkowym, a drugie na adresie końcowym pożądanego zakresu (np. MPC55xx).

Znaczna ilość architektur dostarcza jedynie maski bitowej w celu założenia przerwania on-chip na zadanym zakresie adresowym. W wypadku tym, zakres adresowy jest zawsze powiększony do najmniejszej maski bitowej zawierającej pożądaną zakres. W celu kontrolowania aktualnie przerywanego adresu, zalecane jest użycie komendy `Data.View <address>`.

Punkt przerwania zapisu został ustawiony na zmiennej hll o nazwie 'flags'. Użyta architektura procesora umożliwia użycie jedynie maski bitowej, toteż większa przestrzeń adresowa oznaczona jest punktem przerwania zapisu.

breakpoint	address	data	value	symbol
W	SD:00007E79	05	'5'	\\varnAGlobal\ineuave+0x130D
W	SD:00007E7A	30	'='	\\varnAGlobal\ineuave+0x130E
W	SD:00007E7B	98	'8'	\\varnAGlobal\ineuave+0x130F
W	SD:00007E7C	39	'9'	\\varnAGlobal\flags
W	SD:00007E7D	09	'9'	\\varnAGlobal\flags+0x1
W	SD:00007E7E	44	'D'	\\varnAGlobal\flags+0x2
W	SD:00007E7F	01	'1'	\\varnAGlobal\flags+0x3
W	SD:00007E80	B1	'1'	\\varnAGlobal\flags+0x4
W	SD:00007E81	33	'3'	\\varnAGlobal\flags+0x5
W	SD:00007E82	D3	'3'	\\varnAGlobal\flags+0x6
W	SD:00007E83	03	'3'	\\varnAGlobal\flags+0x7
W	SD:00007E84	14	'4'	\\varnAGlobal\flags+0x8
W	SD:00007E85	43	'C'	\\varnAGlobal\flags+0x9
W	SD:00007E86	6A	'j'	\\varnAGlobal\flags+0x0A
W	SD:00007E87	B8	'8'	\\varnAGlobal\flags+0x0B
W	SD:00007E88	01	'1'	\\varnAGlobal\flags+0x0C
W	SD:00007E89	C7	'7'	\\varnAGlobal\flags+0x0D
W	SD:00007E8A	99	'9'	\\varnAGlobal\flags+0x0E
W	SD:00007E8B	31	'1'	\\varnAGlobal\flags+0x0F
W	SD:00007E8C	75	'u'	\\varnAGlobal\flags+0x10
W	SD:00007E8D	41	'A'	\\varnAGlobal\flags+0x11
W	SD:00007E8E	28	'8'	\\varnAGlobal\flags+0x12
W	SD:00007E8F	0C	'C'	
W	SD:00007E90	00		\\varnAGlobal\_signalvector
W	SD:00007E91	47	'G'	\\varnAGlobal\_signalvector+0x1
W	SD:00007E92	03	'3'	\\varnAGlobal\_signalvector+0x2

## Procesory RISC/CISC

Rodzina CPU	Punkty przerwań on-chip	Przerwania instrukcji	Przerwania dostępu do danych	Przerwania danych
68k				
6833x	-	-	-	-
6834x	-	-	-	-
68360	1	1	1	-
68HC12	do 2	do 2 pojedynczych adresów	do 2 pojedynczych adresów	1
68HC12A				
68HC16	-	-	-	-
Andes	0 ... 8	do 8	do 8 zakresów jako maska bitowa	do 8
ARM7	2 lub 1 (jeśli przerwania programowe są użyte)	do 2 zakresów jako maska bitowa	do 2 zakresów jako maska bitowa	2
ARM9				
Janus				
ARM10 /	2 ... 16 instrukcji	2 ... 16 pojedynczych adresów	2 ... 16 pojedynczych adresów	-
ARM11	2 ... 16 odczyt/zapis			

<b>C166CBC</b> <b>C166SV2</b>	4	do 4	do 4 zapis do 1 odczyt	do 4 zapis do 1 odczyt
<b>Cortex-A</b>	2 ... 16 instrukcji 1 ... 16 odczyt/ zapis	2 ... 16 zakre- sów jako maska bitowa	1 ... 16 zakresów jako maska bito- wa	-
<b>Cortex-M</b>	6 instrukcji 4 asynchroniczne (asynchroniczny punkt przerwań zatrzymuje pro- gram po wystą- pieniu przerwa- nia)	6 pojedynczych adresów i do 4 zakresów jako maska bitowa wykorzystująca asynchronicz- ność	do 4 asynchro- nicznych zakre- sów jako maska bitowa	1 (wykorzystuje 2 przerwania asyn- chroniczne)
<b>Cortex-R</b>	2 ... 8 instrukcji 1 ... 8 odczyt/za- pisy	2 ... 8 zakresów jako maska bito- wa	1 ... 8 zakresów jako maska bito- wa	-
<b>ColdFire</b>	4 instrukcje, 2 odczyt/zapis	3 pojedyncze adresy, 1 maska bitowa	2 pojedyncze ad- resy lub 2 zakresy	2
<b>eTPU</b>	2	do 2 pojedyn- czych adresów	do 2 zakresów od- czytu/zapisu jako maska bitowa	2 (tylko z przerwa- niami zapisu)
<b>H8S</b>	2	do 2	do 2 zakresów jako maska bito- wa	2
<b>H8SX</b>	4	do 4	do 4 zakresów jako maska bito- wa	1
<b>M32R</b>	4 instrukcje 2 odczyt/zapis	4 pojedyncze adresy	2 pojedyncze ad- resy lub 2 zakresy	2
<b>MCORE</b>	2	2 pojedyncze adresy lub 1 za- kres jako maska bitowa	2 zakresy jako maska bitowa	-
<b>MCS8</b>	2	do 2 pojedyn- czych adresów	do 2 pojedyn- czych adresów (zredukowane do 1 jeśli wykorzysta- ne jest z daną)	1
<b>MCS12</b> <b>MCS12C</b>	do 3	do 3 pojedyn- czych adresów	do 3 pojedyn- czych adresów	1

<b>MCS12X</b>	4	do 4 pojedynczych adresów lub 2 zakresy adresowe	do 4 pojedynczych adresów lub 2 zakresy adresowe	1
<b>MGT5100</b>	1 instrukcja (brak przerwania <i>on-chip</i> , jeśli użyte są przerwania programowe)  1 odczyt/zapis	1/0 pojedynczy adres	1 pojedynczy adres	-
<b>MIPS32</b> <b>MIPS64</b>	do 15 instrukcji  do 15 odczyt/zapis	do 15 zakresów jako maska bitowa	do 15 zakresów jako maska bitowa	do 15
<b>MPC500</b> <b>MPC800</b>	4 instrukcje,  2 odczyt/zapis	4 pojedyncze adresy lub 2 zakresy	2 pojedyncze adresy lub 1 zakres	2
<b>MPC5200</b>	2 instrukcje (zredukowane do 1 jeśli użyte są przerwania programowe)  2 odczyt/zapis	2/1  2 pojedyncze adresy lub 1 zakres	2  2 pojedyncze adresy lub jeden zakres	-
<b>MPC55xx</b>	4 instrukcje  2 odczyt/zapis	4 pojedyncze adresy lub 2 zakresy	2 pojedyncze adresy lub 1 zakres	-
<b>MPC74xx</b> <b>MPC86xx</b>	1 instrukcja (brak przerwania <i>on-chip</i> , jeśli użyte są przerwania programowe)  1 odczyt/zapis	1/0 pojedynczy adres	1 pojedynczy adres	-
<b>MPC8240</b> <b>MPC8245</b> <b>MPC825x</b> <b>MPC826x</b> (PQ2)	1 instrukcja (brak przerwania <i>on-chip</i> , jeśli użyte są przerwania programowe)	1/0 pojedynczy adres	-	-

<b>MPC8247</b> <b>MPC8248</b> <b>MPC827x</b> <b>MPC8280</b> <b>(PQ27)</b> <b>MPC83xx</b> <b>(PQ2 Pro)</b>	2 instrukcje (zredukowane do 1 jeśli użyte są przerwania programowe)  2 odczyt/zapis	2/1  2 pojedyncze adresy lub 1 zakres	2  2 pojedyncze adresy lub 1 zakres	-
<b>MPC85xx</b> <b>(PQ3)</b>	2 instrukcje (zredukowane do 1 jeśli użyte są przerwania programowe)  2 odczyt/zapis	2/1  2 pojedyncze adresy lub 1 zakres	2  2 pojedyncze adresy lub 1 zakres	-
<b>PPC401</b> <b>PPC403</b>	2 instrukcje  2 odczyt/zapis	2 pojedyncze adresy lub 2 zakresy	2 pojedyncze adresy lub 2 zakresy	-
<b>PPC405</b> <b>PPC44x</b>	4 instrukcje  2 odczyt/zapis	4 pojedyncze adresy lub 2 zakresy	2 pojedyncze adresy lub 1 zakres	2
<b>PPC600</b>	1 instrukcja (brak przerw <i>on-chip</i> , jeśli użyte są przerwania programowe)	1/0 pojedynczy adres	-	-
<b>PPC740</b> <b>PPC750</b>	1 instrukcja (brak przerw <i>on-chip</i> , jeśli użyte są przerwania programowe)  1 odczyt/zapis	1/0 pojedynczy zakres	1 pojedynczy adres	-
<b>PWR-ficient</b>	2 instrukcje  2 odczyt/zapis	2 pojedyncze adresy lub 1 zakres	2 pojedyncze adresy lub jeden zakres	-
<b>SH2A</b> <b>ST4A</b>	10	do 10	do 10 zakresów jako maski bitowe	2
<b>SH3</b>	2	do 2	do 2 zakresów jako maski bitowe	-
<b>SH4</b> <b>ST40</b>	6	do 6	do 6 zakresów jako maski bitowe	2

<b>SH7047</b> <b>SH7144/45</b>	1	do 1	do 1	-
<b>SH7058</b>	12	do 12	do 12 zakresów jako maski bitowe	do 12
<b>Super10</b>	do 8	do 8	do 8	8
<b>TriCore</b>	do 8 instrukcji do 4 odczyt/zapis	do 8 pojedynczych adresów lub/i do 4 zakresów	do 4 pojedynczych adresów lub zakresów	-
<b>XC800</b>	4	do 4 do 1 zakresu (potrzebne 2 pojedyncze)	do 1 pojedynczego adresu do odczytu do 1 pojedynczego adresu do odczytu lub zakresu	-
<b>XSCALE</b>	2 instrukcje 2 odczyt/zapis	2 pojedyncze adresy	2 pojedyncze adresy lub 1 zakres jako maska bitowa	-

## Procesory DSP

Rodzina CPU	Punkty prze- rwań on-chip	Przerwania in- strukcji	Przerwania do- stępu do danych	Przerwania da- nych
<b>Blackfin</b>	6 instrukcji 2 odczyt/zapis	6 pojedynczych adresów lub 3 zakresy	2 pojedyncze adresy lub 1 zakres	-
<b>CEVA-X</b>	4 instrukcje 4 odczyt/zapis	4 pojedyncze adresy	4 pojedyncze adresy lub zakresy	2
<b>DSP56K</b> <b>56k/56300/</b> <b>56800</b> <b>56100</b>	2 1	2 1	2 1	-

<b>DSP</b> <b>56300</b> <b>56800E</b>	2	do 2 pojedynczych adresów	do 1 pojedynczego adresu	-
<b>MMDSP</b>	2 instrukcje 1 odczyt/zapis	2 pojedyncze adresy	1 pojedynczy adres	1
<b>OAK</b> <b>TeakLite</b> <b>TeakLite II</b> <b>Teak</b>	3 instrukcje 1 odczyt/zapis	3 pojedyncze adresy	1 pojedynczy adres lub zakres jako maska bitowa	1
<b>StarCore</b>	12	do 12 pojedynczych adresów lub do 6 zakresów	do 6 pojedynczych adresów lub do 3 zakresów	1
<b>STN8810</b> <b>STN8815</b> <b>STN8820</b>	2	do 2	do 2	1
<b>TeakLite III</b>	2 instrukcje 1 odczyt/zapis	2 pojedyncze adresy	2 pojedyncze adresy lub 1 zakres	1
<b>TMS320</b> <b>C28x</b>	2	2 pojedyncze adresy	-	-
<b>TMS320</b> <b>C54x</b>	2	2 pojedyncze adresy	-	-
<b>TMS320</b> <b>C55x</b>	4	do 4 pojedynczych adresów	do 3 danych, 1 zakres i 2 maski bitowe	do 3
<b>TMS320</b> <b>C62x</b>	1	1 pojedynczy adres	-	-
<b>TMS320</b> <b>C64x</b>	do 4	do 4 pojedynczych adresów	-	-
<b>TMS320</b> <b>C67x</b>	1	1 pojedynczy adres	-	-
<b>ZSP400</b>	-	-	-	-
<b>ZSP500</b>	4	do 4 pojedynczych adresów	do 1 zakresu jako maska bitowa	1



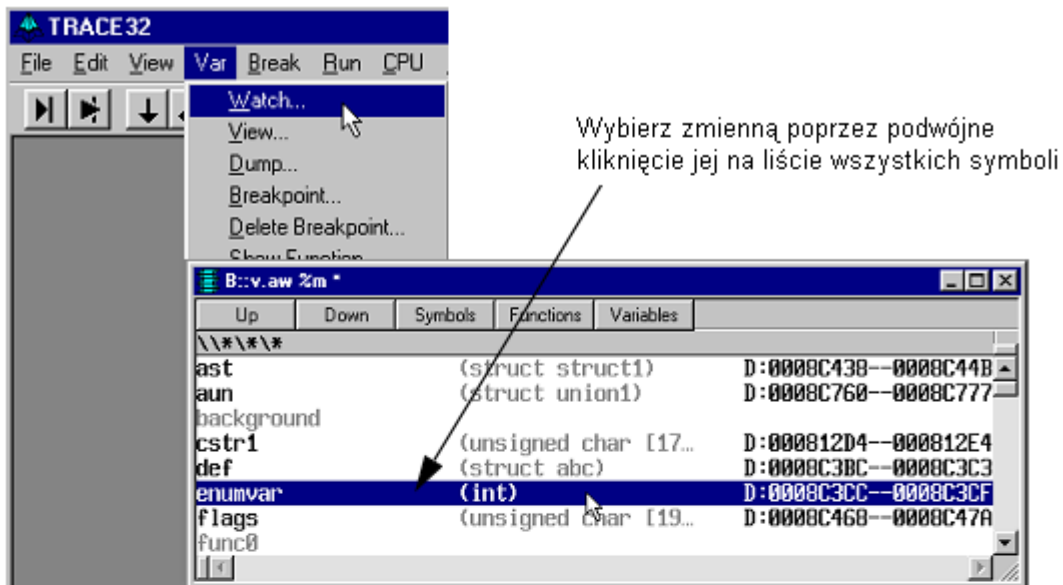
Rodzina CPU	Punkty przerwań on-chip	Przerwania instrukcji	Przerwania dostępu do danych	Przerwania danych
<b>Micro-Blaze</b>	0 ... 4 instrukcje 0 ... 4 odczyt/zapis	0 ... 4 zakresy jako maski bitowe	0 ... 4 zakresy jako maski bitowe	-
<b>NIOS2</b>	0/4/8 (konfigurowalne)	do 4	do 4 pojedynczych adresów lub 2 zakresy	do 4

### Rdzenie konfigurowalne

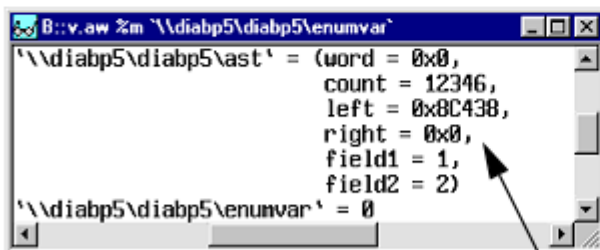
Rodzina CPU	Punkty przerwań on-chip	Przerwania instrukcji	Przerwania dostępu do danych	Przerwania danych
<b>ARC600</b> <b>ARC700</b>	0/2/4/8 (konfigurowalne)	do 0/2/4/8 zakresów jako maski bitowe	do 0/2/4/8 zakresów jako maski bitowe	do 0/1/2/4 (tylko zapisy, tylko w pełnym trybie)
<b>Diamond Cores</b>	2	do 2 zakresów jako maski bitowe	do 2 zakresów jako maski bitowe	2

## Podgląd i modyfikacja zmiennych HLL

Aby wyświetlić zawartość zmiennych typu HLL, należy wybrać opcję Watch z menu Var.



Wybrana zmienna wyświetlona jest na górze okna Watch



Za każdym razem kiedy dodasz nową zmienną do obserwowania, będzie się ona pojawiała na górze okna. Rozszerz okno, aby zobaczyć wszystkie wpisy.

Szybszą metodą podglądu zmiennej jest zaznaczenie jej kursorem w oknie Data.List i naciśnięcie prawego przycisku myszy. Z menu podręcznego Var należy wybrać Add to Watch Window.

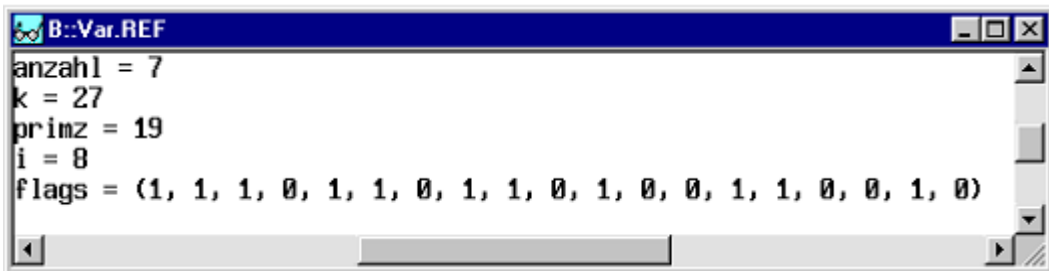
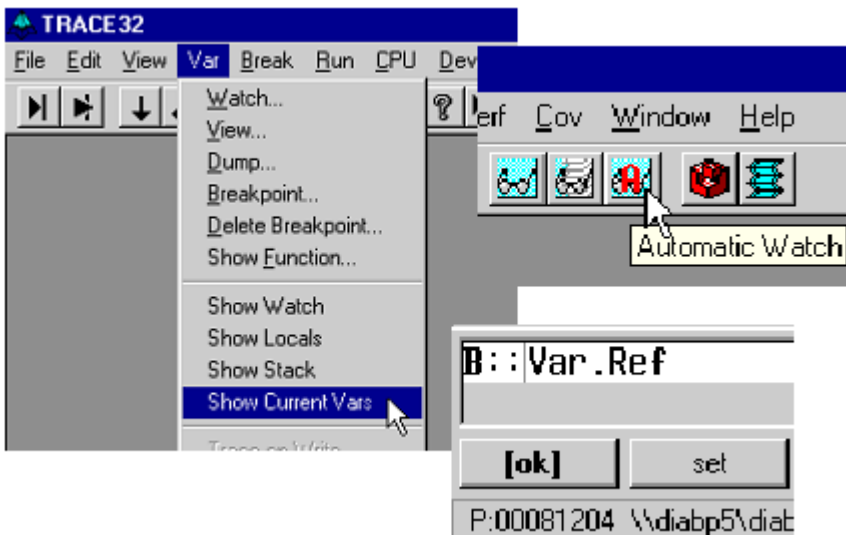
```

while ( k <= SIZE )
{
    flags[ k ] = FALSE;
    k +=
}
anzahl++;

```

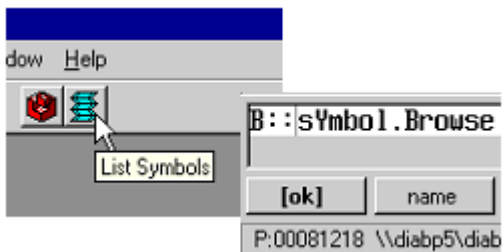
Jeśli chcesz wyświetlić bardziej złożoną strukturę lub tablicę w oddzielnym oknie, zawsze możesz skorzystać z pozycji View... w menu Var.

W wypadku kiedy potrzebujesz śledzić wartości wszystkich zmiennych występujących w aktualnym kontekście programu, powinieneś użyć pozycji Show Current Vars w menu Var i wykonać kilka pojedynczych kroków.

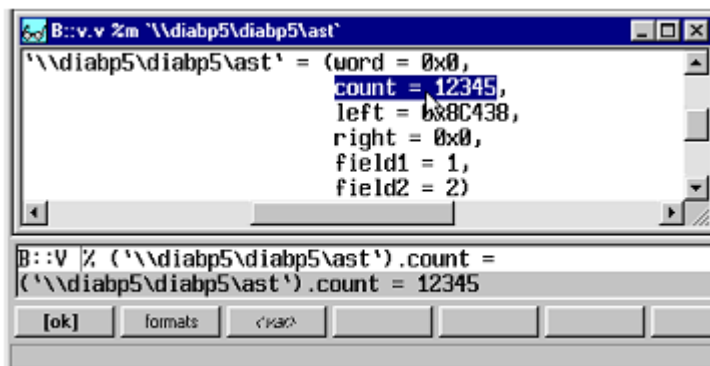


W większości okien, dostęp do menu podręcznego realizowany jest poprzez kliknięcie prawym przyciskiem myszy. Jeśli zaznaczysz zmienną, otrzymasz dostęp do menu Var, które umożliwi Ci jej podgląd oraz modyfikacje.

W przypadku kiedy chcesz podejrzeć zmienną, lecz nie jesteś dokładnie pewien jej nazwy, zawsze możesz skorzystać z przeglądarki symboli zawierającej bazę wszystkich używanych obiektów.

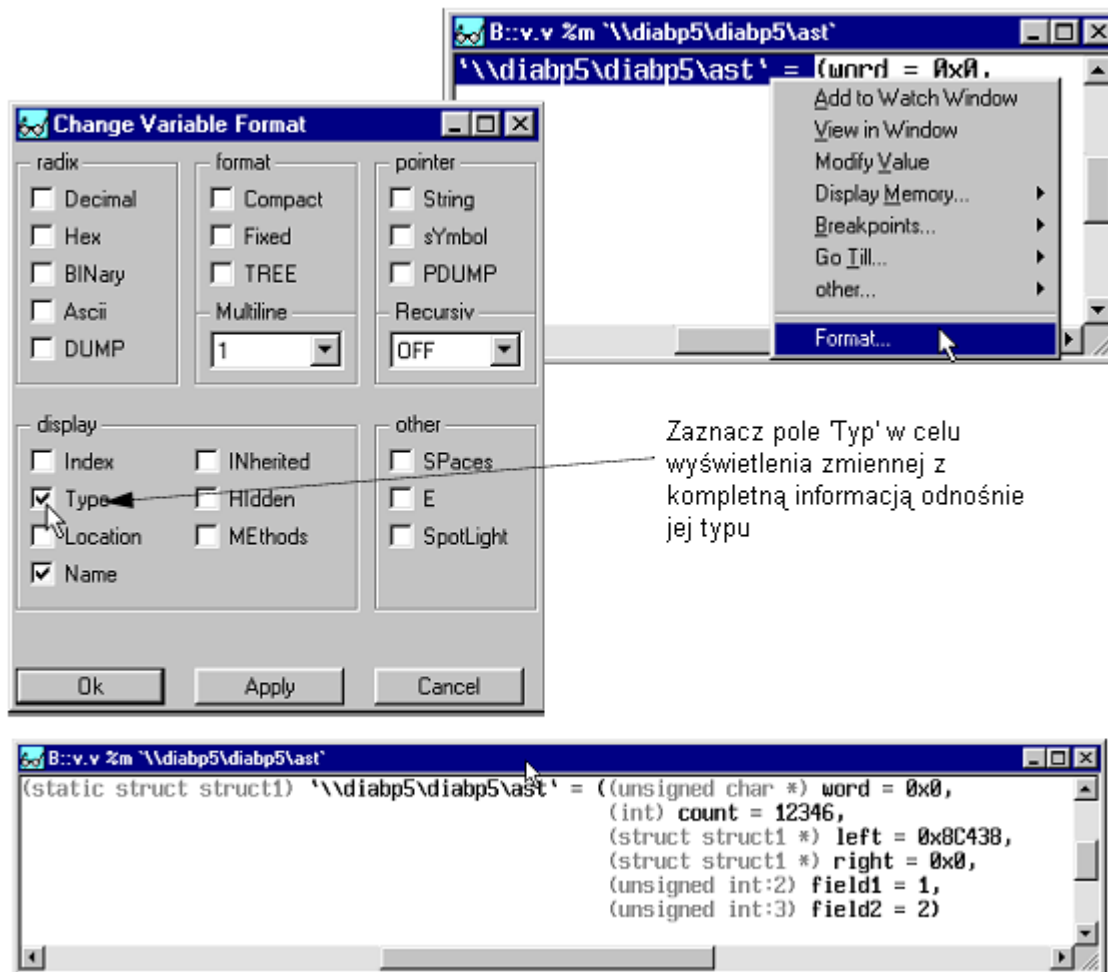


Jeśli chcesz zmodyfikować jakąś zmienną, kliknij dwa razy na jej wartość. Odpowiednia komenda `Var.Set` zostanie automatycznie wpisana w wierszu poleceń. Wprowadź nową wartość i zatwierdź ją enterem.

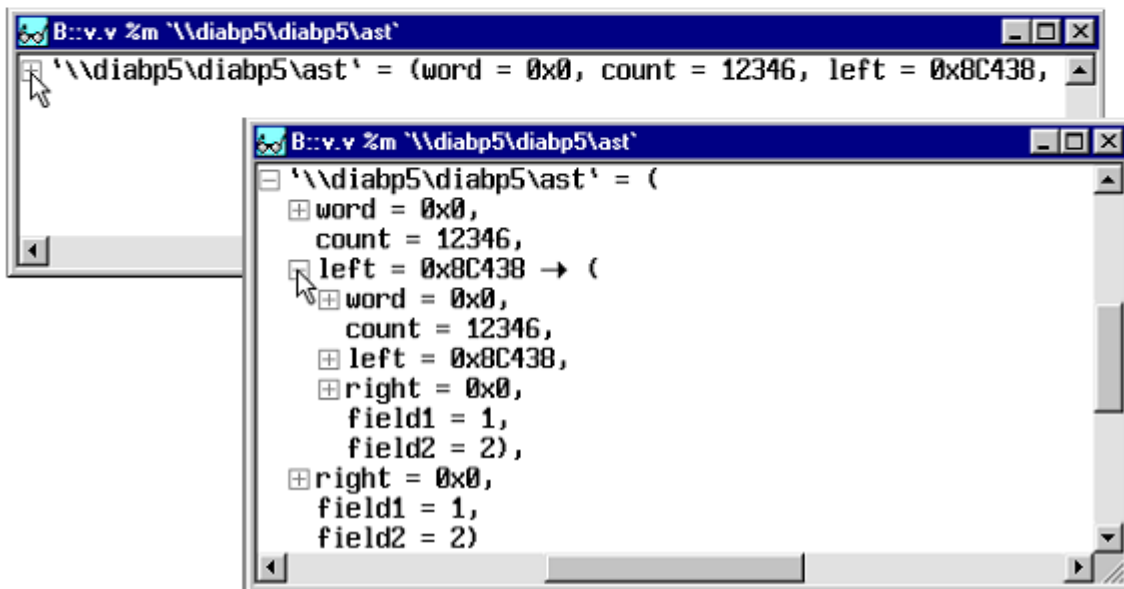
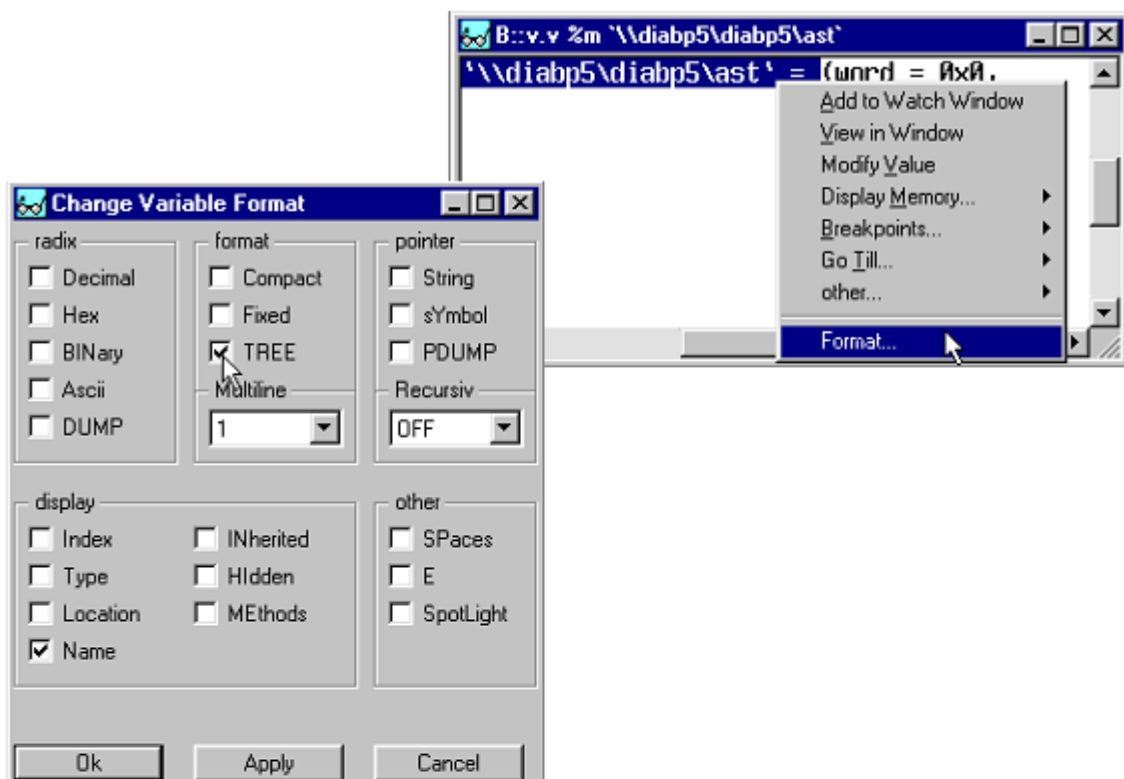


## Format zmiennych HLL

W celu dostosowania sposobu wyświetlania zmiennych do Twoich potrzeb, należy zaznaczyć nazwę zmiennej, nacisnąć prawy przycisk myszy i wybrać Format... z menu Var.

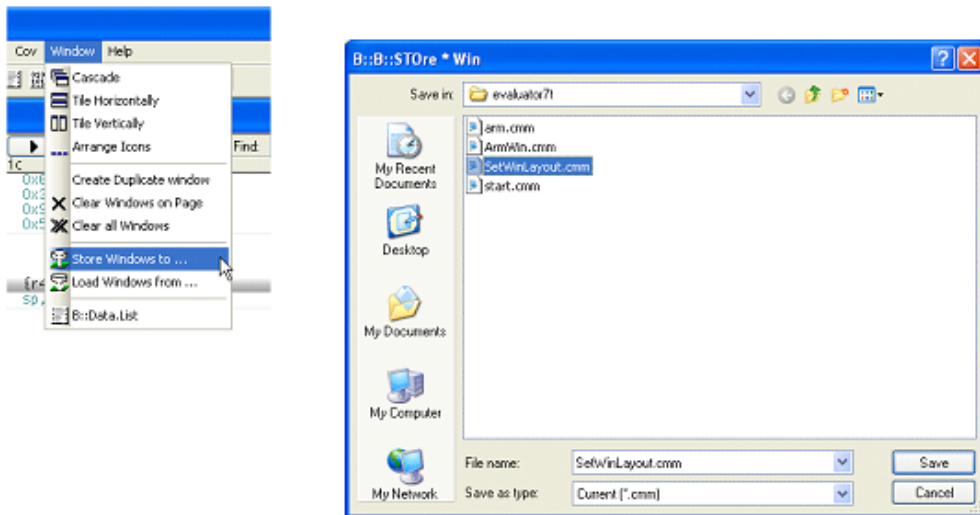


Jeśli wyświetlasz bardziej złożone struktury HLL, wybierz pole TREE w ramce Format w oknie Change Variable Format. Czynność ta pozwoli na sformatowanie osobnego sposobu wyświetlania każdego pola struktury poprzez kliknięcie + lub



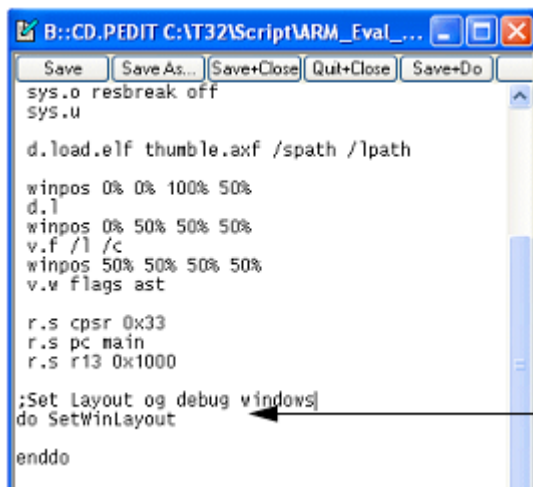
## Kończenie pracy TRACE32

W celu zapisania konfiguracji okien Twojego środowiska TRACE32-ICD, użyj pozycji Store Windows to... w menu Window. Funkcja ta wygeneruje plik skryptowy w języku PRACTICE zawierający wszystkie niezbędne polecenia do przywrócenia zapisanego środowiska. Podaj nazwę pliku w polu File name w oknie Store i naciśnij przycisk Save, aby zapisać ustawienia.



Zapisane ustawienia okien mogą być przywrócone z powrotem, w następnej sesji poprzez wybranie pozycji Load Windows from... w menu Window.

Dodatkowo, język PRACTICE obsługuje modułową strukturę programu, toteż możesz użyć wywołania funkcji w Twoim pliku startowym dokonującej automatycznego przywrócenia konfiguracji.



Aby zakończyć pracę programu wybierz pozycję Exit z menu File.



Zwróć szczególną uwagę na sekwencję włączania / wyłączenia systemu:

- Włączanie: debugger – układ docelowy
- Wyłączenie: układ docelowy - debugger