Lab 2 Working with AXD

Table of Contents

1. INTRODUCTION 1

2. RUNNING A PROGRAM 1

3. SETTING A BREAKPOINT 2

4. SETTING A WATCHPOINT 6

5. EXAMINING THE CONTENTS OF VARIABLES 7

5.1 CONTENTS OF VARIABLES 7
5.2 ADDRESSES AND CONTENTS OF VARIABLES 9

6. EXAMINING THE CONTENTS OF REGISTERS AND MEMORY 12

6.1 EXAMINING THE CONTENTS OF REGISTERS 12
6.2 EXAMINING THE CONTENTS OF MEMORY 13
6.3 LOCATING AND CHANGING VALUES AND VERIFYING CHANGES 15

IP Core Design
2002.9

1. Introduction

This Lab gives step-by-step instructions to perform a variety of debugging tasks. These contents are based on the Chapter 3 of "ARM Developer Suite Version 1.1 Debuggers Guide". Thought in this Lab the debugger target is ARMulator, but the skills can be applied to Multi-ICE/Angel with the ARM development board(s). The following instructions are based on the demonstration program that runs the **Dhrystone** test software, which is the same to that used in Lab 1. For details of the Dhrystone test program, please refer to the readme.txt file and the various source files in its subdirectory (e.g., **C:\Program Files\ARM\ADSv1_1\Examples\dhryansi**).

Skills you will learn

- Set breakpoints and watchpoints
- Locate, examine and change the contents of variables, registers and memory
- Using the command line interface to automate repetitive tasks

2. Running a Program

1. Make the directory C:\ARMSoC\Lab_02\AXD_skills\

2. Copy ARM Project file **My_Poeject.mcp** from C:\ARMSoC\Lab_01\My_Poeject\ to AXD_skills\ or copy **dhryansi.mcp** from the **dhryansi** directory in the C:\Program Files\ARM\ADSv1_1\Examples\dhryansi\ and rename it as **My_Poeject.mcp**.

3. Double click on **My_Poeject.mcp**.

4. Make **My_Poeject.mcp** and then Select ***Project →Debug*** (Ctrl + F5) to launch AXD.
   4.1 A **Disassembly processor view** of the image is displayed and a blue arrow indicates the current execution point.

5. Select ***Execute → Go*** from the menu (or press F5, or from toolbar) to begin execution on the target processor.
   5.1 Execution stops at the beginning of function main(), where a breakpoint is set by default. A red disc indicates the line where a breakpoint is set.
   5.2 Also, a Source processor view of the relevant few lines of the relevant file is displayed. If it is not, right-click in the **Disassembly processor view** and select ***Source*** from the pop-up menu. Again, a red disc indicates the line where a breakpoint is set, and a blue arrow indicates the current execution point.

6. Select ***Go*** again to continue execution.
   6.1 You are prompted, in the **Console processor view**, for the number of runs through the benchmark that you want performed.
   6.2 ***Enter 8000***. The program runs for a few seconds, displays some diagnostic messages, and shows the test results.

7. To repeat the execution of the program, select ***Reload*** Current Image from the File menu or toolbar shown in Figure 1, then repeat Steps 5 and 6.
   7.1 You do not have to open the Source process view again. Once opened, it remains displayed.

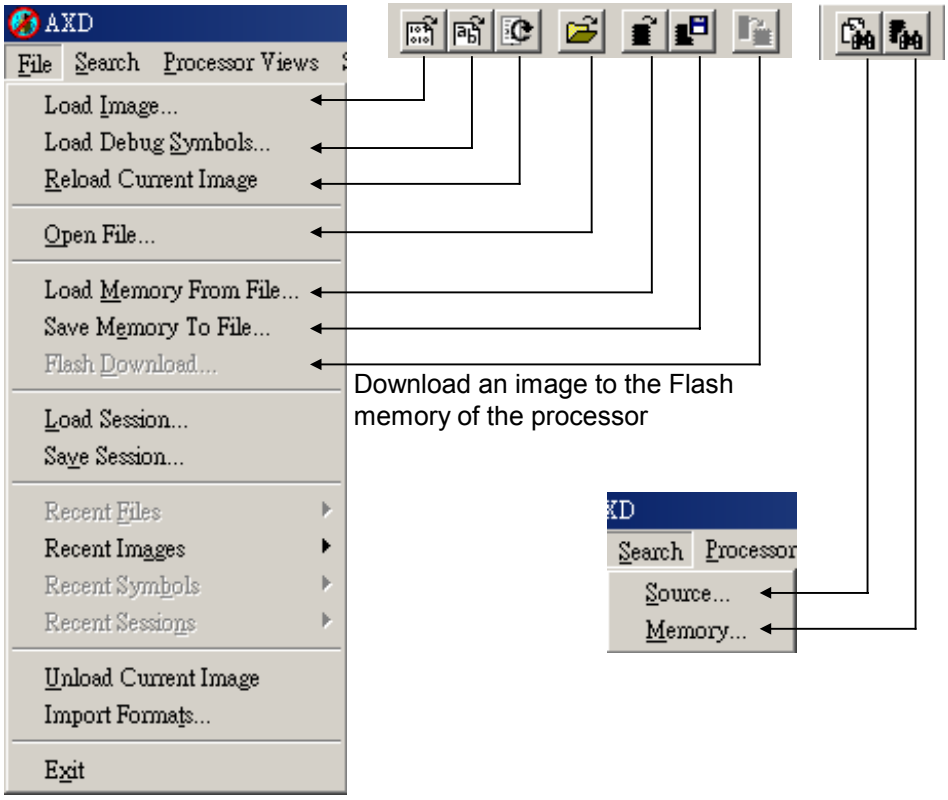

Figure 1. File and Search Menu, and their corresponding Toolbar.

3. Setting a breakpoint

1. Select ***Reload Current Image***

2. Select ***Go*** to reach the first breakpoint, set by default at the beginning of function main() and indicated by a red disc. You can see the source file dhry_1.c with a breakpoint and the current position indicated at line number ***87***.

3. Scroll down through the source file until line number ***159*** is visible. This is a call to **Proc_4()**, and is inside the loop to be executed the number of times you specify.
   3.1 Alternatively, you may use ***Search in Source*** by Search string set as "**Proc_4()**", as shown in Figure 2.

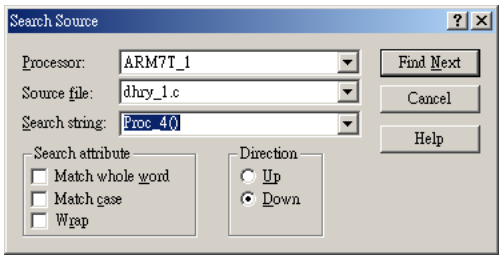

Figure 2. Search in Source.

4. ***Right-click*** on line ***159*** to position the cursor there and display the pop-up menu, and select ***Toggle Breakpoint*** (or left-click on the line and press ***F9***, or double-click in the margin next to the line).
   4.1 Another red disc indicates that you have set a second breakpoint.

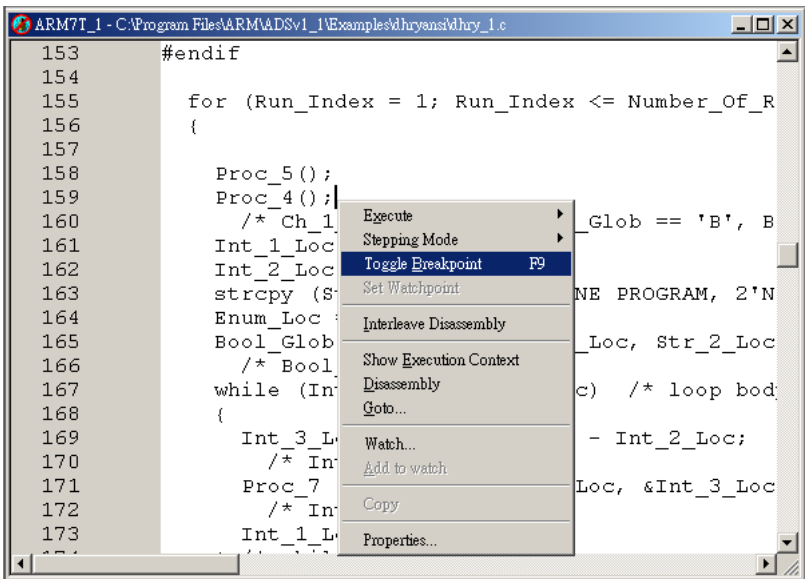

Figure 3. Set Breakpoint.

5. To edit the details of the new breakpoint, select ***Breakpoints*** from the ***System Views*** menu (Figure 4). The breakpoints pane is displayed (Figure 5).
   5.1 Double-click on the line in the breakpoints pane that describes the new breakpoint, or right-click on it and select ***Properties*** (Figure 5), to display a ***Breakpoint Properties*** dialog.
   5.2 Enter ***750*** in the ***out*** of field in the ***Conditions box,*** as shown in Figure 6. This is the number of times execution has to arrive at the breakpoint to trigger it.
   5.3 Click OK.

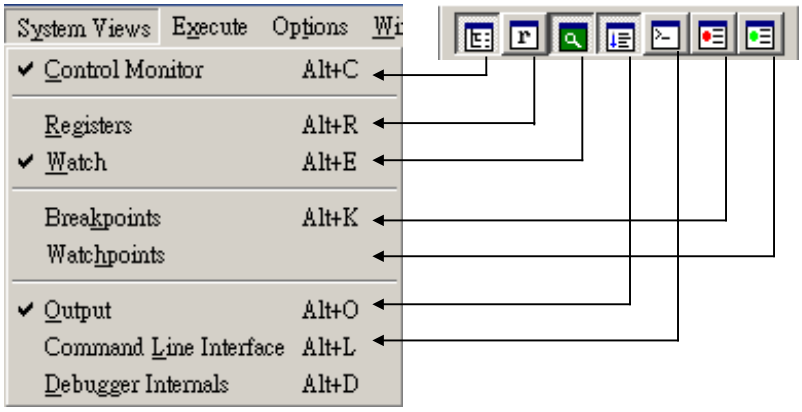

Figure 4. System View Menu and its corresponding Toolbar.

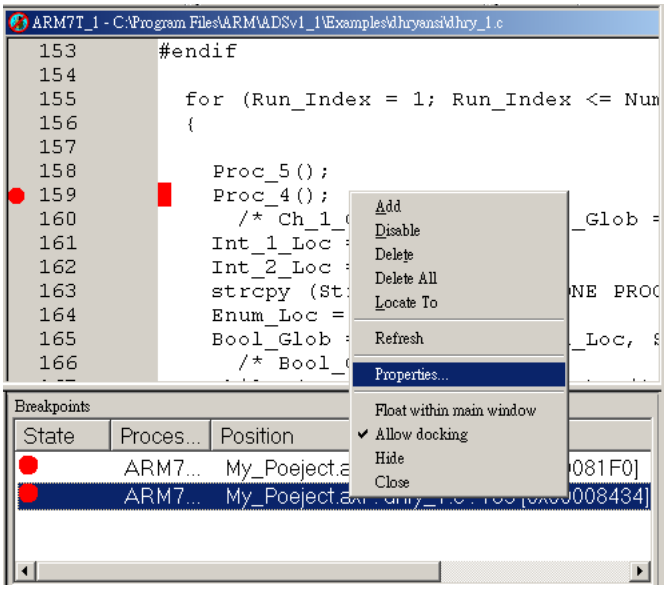

Figure 5. Breakpoints pane (the lower window).

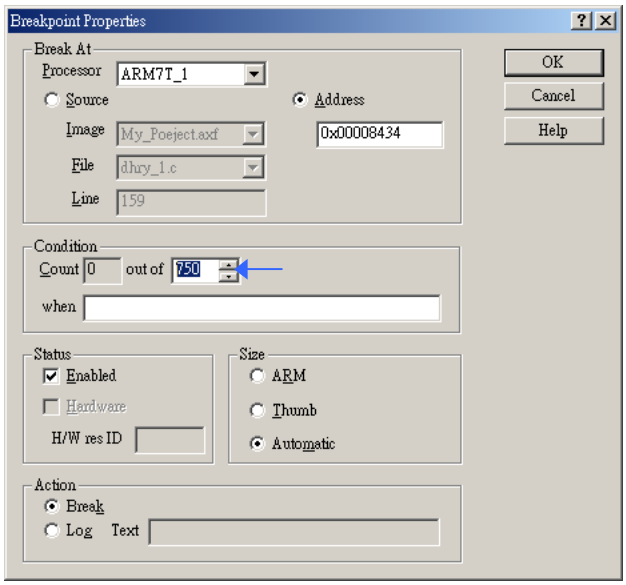

Figure 6. Setting breakpoint details.

6. Press ***F5*** (***Go***) to resume execution, and enter the smaller number of 5000 this time for the number of runs required.
   6.1 Execution stops when the ***750th*** time your new breakpoint is reached.

7. Select ***Variables*** from the ***Processor Views menu*** (Figure 7) to check progress. Reposition or resize the window if necessary.
   7.1 Click the ***Local tab*** and look for the ***Run_Index*** variable. Its value is shown as 0x2EE (hexadecimal).
   7.2 Right-click on the variable so that it is selected and a pop-up menu appears (Figure 8). Select ***Format → Decimal*** and the value is now displayed as 750 (decimal).

Note: For a description of the display formats available, see Data Formatting, Section 4.6 of Debuggers Guide.

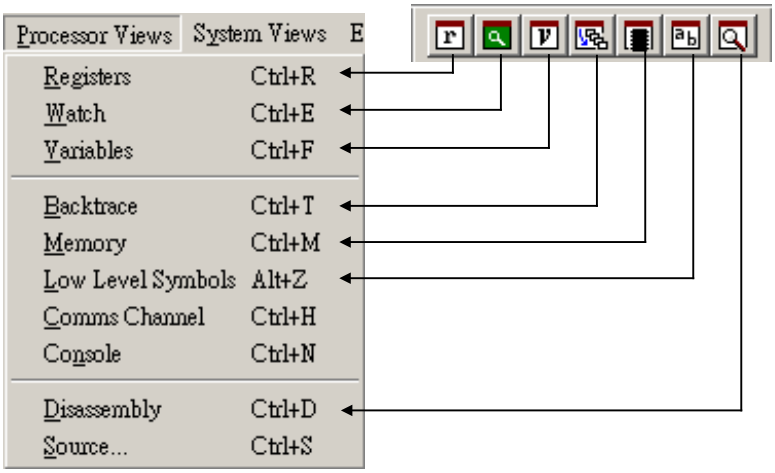

Figure 7. Processor View Menu and its corresponding Toolbar.

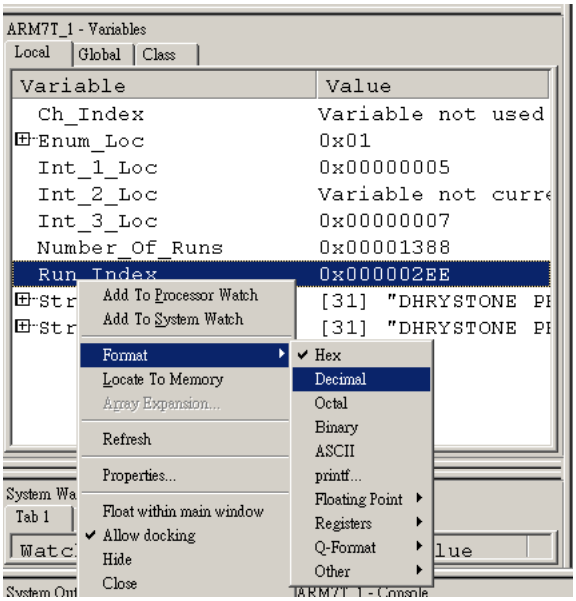

Figure 8. Pop-up menu for Local variable in Processor View.

8. Press ***F5*** to resume execution, and the value of the ***Run_Index*** local variable changes to ***1500***. It is now (red) colored to show that its value has changed since the previous display.

9. Press ***F5*** repeatedly until the value of ***Run_Index*** reaches the highest multiple of 750 (1500➔2250➔3000➔3750➔4500) before exceeding your specified number of runs, then once more to allow the program to complete execution.

Note: This time the Dhrystone test results are meaningless, because of the interruptions to the timing measurements, but the use of a breakpoint has been demonstrated.

10. Close down the Breakpoints system view, either by right-clicking and selecting Close or by clicking on the Close button in the title bar if the view is not docked.

4. Setting a watchpoint

1. Select ***Reload Current Image***

2. Select ***Go*** to reach the first breakpoint, set by default at the beginning of function main().

3. Select ***Go*** to continue execution.

4. When you are prompted for the number of runs to execute, enter ***770***. Execution continues until it reaches the breakpoint at line ***159*** for the 750th time. This is the breakpoint you defined in last section.

5. Select ***Watchpoints*** from the ***System Views*** menu/toolbar (Figure 4), right-click in the Watchpoints system view (Figure 9), and select ***Add*** to display the Watchpoint Properties dialog (Figure 10).
   5.1 Enter ***Run_Index*** in the ***Item*** field in the Watch box.
   5.2 Set the ***out of field in the Conditions*** box to a value of ***6***. This is the number of times the watched value has to change to trigger the watchpoint action.
   5.3 Click the ***OK*** button. Take a look at the changes in ***Watchpoints view***.

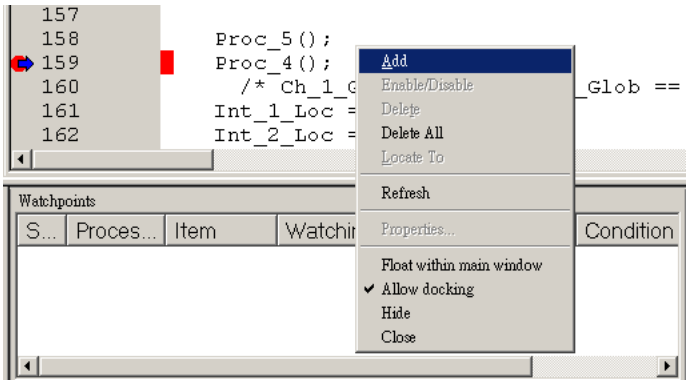

Figure 9. Watchpoints from the System Views.

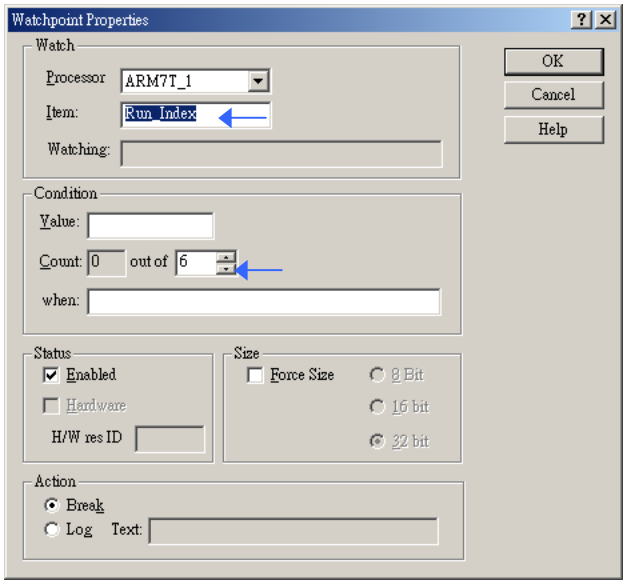

Figure 10. Watchpoint Properties dialog.

6. Show the ***Run_Index*** variable
   6.1 If the ***Variables processor view*** is not already displayed, select Variables from the Processor Views menu. Reposition or resize the window if necessary.
   6.2 Click the ***Local*** tab and look for the ***Run_Index*** variable. The value of ***Run_Index*** is currently ***750***.
   6.3 If it is displayed in hexadecimal notation, right-click on the value and select ***Format → Decimal*** to change the display format to decimal.

7. Press ***F5*** to resume execution.
   7.1 Soon the value of the ***Run_Index*** local variable changes to ***756***. It is now displayed in red to show that its value has changed since the previous display. Execution stops.

8. Examine any displayed values, then press ***F5*** again to resume execution and perform ***six*** more runs.
   8.1 When the value of ***Run_Index*** becomes greater than the number of runs you specified (***770*** at step four), the test results are displayed in console window and execution terminates.

Note: Again, the Dhrystone test results are meaningless, because of the interruptions to the timing measurements, but the use of a watchpoint has been demonstrated.

9. Delete the watchpoint by right-clicking on its line in the ***Watchpoints window*** and selecting ***Delete*** from the pop-up menu, then close down the Watchpoints system view.

5. Examining the contents of variables

Two methods of examining the contents of variables are described in this section:
- Contents of variables (***variable processor view***): This method is simpler and shows only the contents of the specified variables.
- Addresses and contents of variables (***watch processor view***): This method shows the addresses of the variables as well as their contents.

5.1 Contents of variables

To examine the contents of variables as simply as possible, use the Variables processor view.

1. Select ***Reload Current Image***

2. Select ***Go*** to reach the first breakpoint, set by default at the beginning of function main().

3. Select ***Go*** to continue execution.

4. When you are prompted for the number of runs to execute, enter ***760***. Execution continues until it reaches the breakpoint at line ***159*** for the 750th time. This is the breakpoint you defined in Setting a breakpoint at ***Section Three***.

5. If the ***Variables processor view*** is not already displayed, select Variables from the Processor Views menu. Reposition or resize the window if necessary. On the ***Local*** tabbed page look for the Run_Index variable. Other variables that you can see include ***Enum_Loc***, ***Int_1_Loc***, ***Int_2_Loc***, and ***Int_3_Loc***.
   5.1 Right-click in the window, select ***Properties... → Dec*** and click ***OK***. The display is now in decimal format and is similar to that shown in Figure 11.

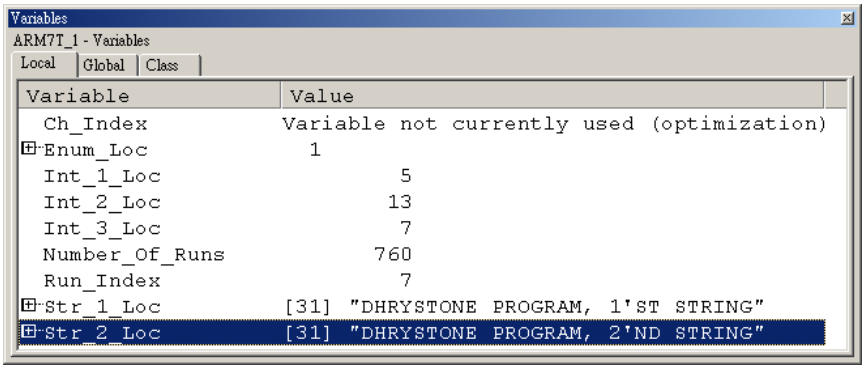

Figure 11. Examining the contents of variables.

6. Press ***F10***. This is equivalent to selecting ***Step*** from the ***Execute menu*** (Figure 12). The program executes a single instruction and stops. Any values that have changed in the ***Variables processor view*** are displayed in red.

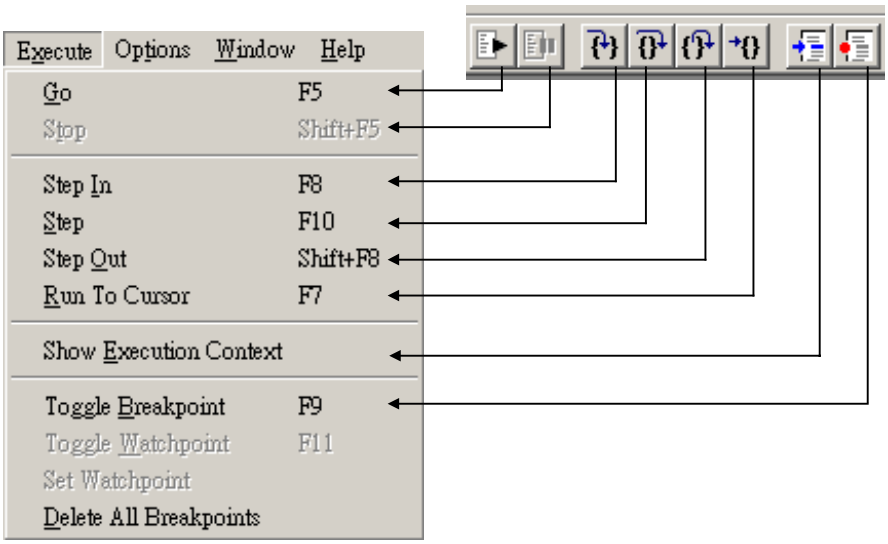

Figure 12. Execute Menu and its corresponding Toolbar.

7. Press ***F10*** repeatedly. As you execute the program, one instruction at a time, the values of several of the variables change. After you have allowed approximately ***30*** program instructions to execute, the value of ***Run_Index*** increases by ***1***. The program has now completed one further execution of the Dhrystone test.

8. Explore the various display options available from the pop-up menu (Figure 8). Try some other settings in both the ***Format*** submenu and the ***Default*** Display Options dialog displayed when you select ***Properties***....

9. Press ***F5*** to allow the program to complete its execution, then close down the ***Variables processor view***.

5.2 Addresses and contents of variables

An alternative method of examining a variable is to use a ***Watch processor view***. This allows you to see the memory address of the variable as well as its value.

1. Select ***Reload Current Image***

2. Select ***Go*** to reach the first breakpoint, set by default at the beginning of function main().

3. Select ***Go*** to continue execution.

4. When you are prompted for the number of runs to execute, enter ***760***. Execution continues until it reaches the breakpoint at line ***159*** for the 750th time. This is the breakpoint you defined in Setting a breakpoint at ***Section Three***.

5. Select ***Watch*** from the ***Processor Views menu*** (Figure 7) and reposition or resize the window if necessary. You can specify items to watch on several tabbed pages. In this example you examine a few variables using the first tabbed page only.

6. Right-click in the window, and select ***Add Watch*** from the pop-up menu (Figure 13). A ***Watch dialog*** appears, prompting you to enter an ***expression***. For this example you enter some valid variable names, most of them preceded by an ampersand (&).
   6.1 Enter the first expression in the Expression field (as shown in Figure 14) by typing:
   &Enum_Loc

Note:

- ***Enum_Loc*** is a global variable, so it is stored in RAM at the address ***&Enum_Loc***.
- These names are ***case-sensitive***.
- You can also add a variable to the Watch view by selecting it in the source view using ***right-clicking*** and selecting ***Watch***, as show in the following Figure. And then using the ***Add Watch*** pop-up menu command. ➔ Try this.

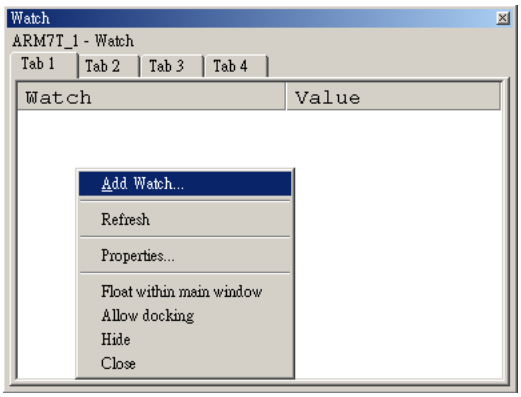

Figure 13. Add Watch to Processor Watch window.

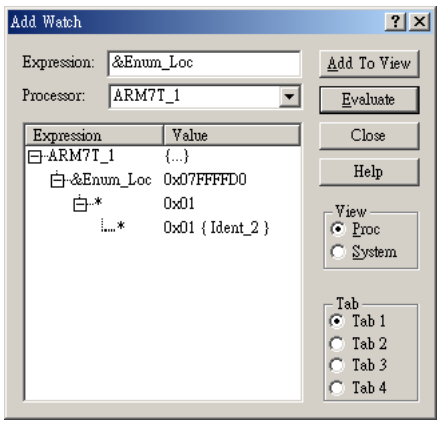

Figure 14. Watch dialog.

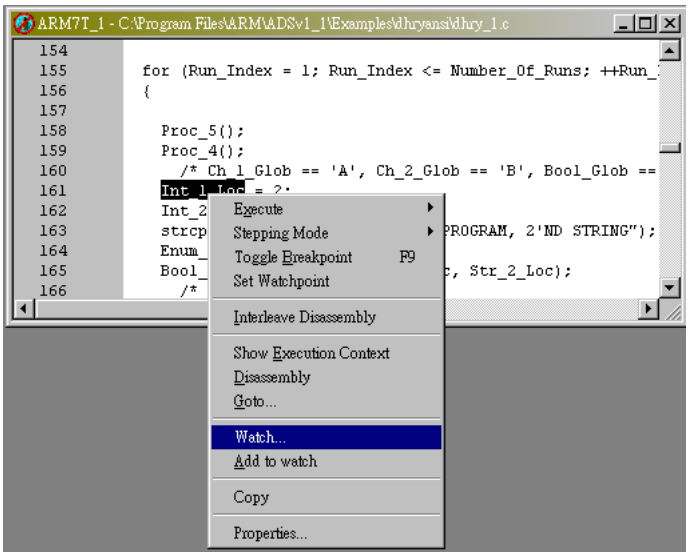

Figure 15. Add a variable to the Watch view by selecting it in the source view.

7. Press the ***Enter*** key or click on the ***Evaluate*** button. The expression you entered appears in the Expression column, and its value, being the address of the variable, appears in the Value column.
   7.1 Click on the + symbol to expand the display, and another line appears showing the contents of the variable in the Value column.
   7.2 Enter, in a similar way:
   &Int_1_Loc
   &Int_3_Loc
   Run_Index
   7.3 Expand these lines. The result is shown in Figure 16.

Note: The ***Run_Index*** variable name is not preceded by an ampersand because, in this program, the variable is stored in a ***processor register***. Having no memory address, it is inappropriate to ask for it to be displayed. Specifying the variable name without the ampersand shows its contents but not its address.

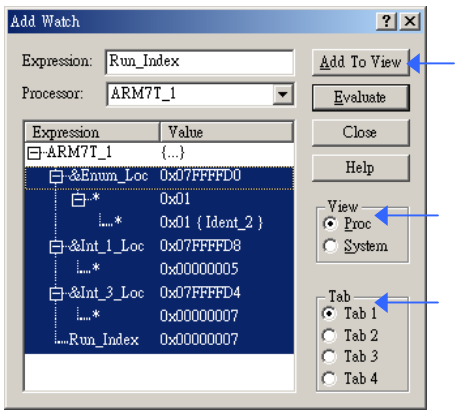

Figure 16. Specifying variables to watch.

8. Select all the lines you have entered, as shown in Figure 16, ensure that ***Proc*** is the selected View and ***Tab1*** the selected Tab, then click the ***Add To View*** button and the ***Close*** button.

9. The variables you have specified are now displayed in the ***Watch processor view*** (similar to that shown in Figure 17), and if you expand the lines you can see both the addresses and the contents of the variables.
   9.1 Point to the value displayed for the ***Run_Index*** variable and right-click to display the pop-up menu. Select ***Format → Decimal*** so that the value of ***Run_Index*** is displayed as a decimal number.

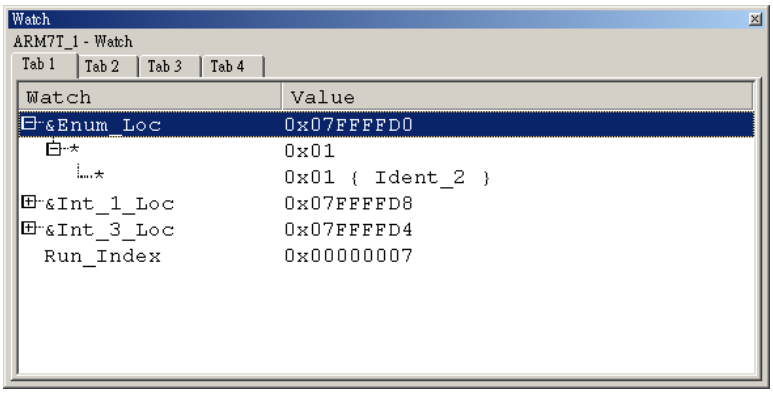

Figure 17. Watch processor view.

10. Press ***F10***. This is equivalent to selecting Step from the Execute menu. The program executes a single instruction and stops. Any values that have changed in the Watch processor view are displayed in color.

11. Press ***F10*** repeatedly. As you execute the program, one instruction at a time, the values of several of the variables change. After you have allowed approximately 30 program instructions to execute, the value of ***Run_Index*** increases by ***1***. The program has now completed one further execution of the Dhrystone test.

12. Explore the various display options available from the pop-up menu. Try some other settings in both the ***Format*** submenu and the ***Default*** Display Options dialog displayed when you select ***Properties***....

13. Press ***F5*** to allow the program to complete its execution, then close down the

Variables processor view.

6. Examining the contents of registers and memory

6.1 Examining the contents of registers

To examine the contents of registers used by the currently loaded program:

1. Select ***Reload Current Image***

2. Select ***Go*** to reach the first breakpoint, set by default at the beginning of function main().

3. Select ***Registers*** from the ***Processor Views menu*** (Figure 7) and reposition or resize the window if necessary.
   3.1 The registers are arranged in groups, with only the group names visible at first. Click on the + symbol of any group name to see the registers of that group displayed. An example is shown in Figure 18.

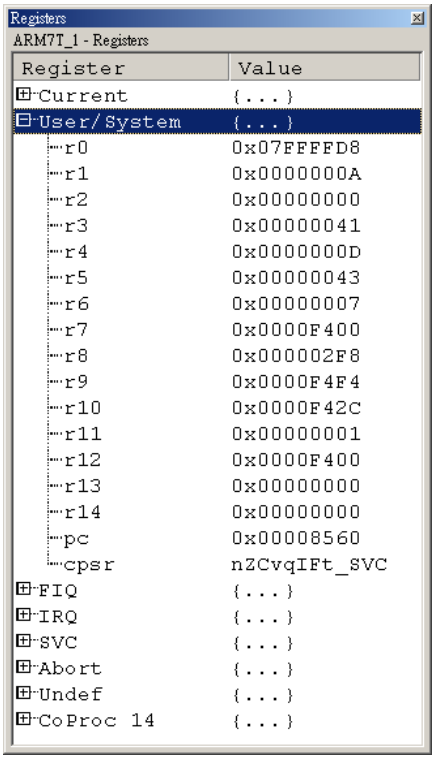

Figure 18. Examining contents of registers.

4. Press ***F10***. This is equivalent to selecting Step from the Execute menu/toolbar (Figure 12). The program executes a single instruction and stops. Any values that have changed in the Registers processor view are displayed in red.

5. Press ***F10*** a few more times. As you execute the program, one instruction at a time, you can see the values of several of the registers change.
   5.1 You soon reach the point when you are prompted, in the Console processor view, for the number of runs to perform. A very small number (e.g., ***300***) is sufficient this time.

6. Explore the format options available from the ***Registers processor view*** pop-up

menu.

6.1 If you position the mouse pointer on a selectable line when you right-click, the line is selected. You can change the display format of selected lines only.

6.2 You can select multiple lines by holding down the ***Shift*** or ***Ctrl*** keys while you click on the relevant lines, in the usual way.

6.3 If you select ***Add to System*** from the pop-up menu, the currently selected register is added to those that are displayed in the ***Registers system view***. This is particularly useful when your target has ***multiple processors*** and you want to examine the contents of some registers of each processor. Collecting the registers of interest into a single Registers system view avoids having to display many separate processor views.

6.4 You can also select ***Add Register*** from the pop-up menu of the ***Registers system view***. This allows you to select registers from any processor to add to those being displayed in the ***Registers processor view***.

7. Press ***F5*** to allow the program to complete its execution, then close down the Registers processor view.

6.2 Examining the contents of memory

To examine the contents of memory used by the currently loaded program:

1. Select ***Reload Current Image***

2. Select ***Go*** to reach the first breakpoint, set by default at the beginning of function main().

3. Select ***Go*** to continue execution.

4. When you are prompted for the number of runs to execute, enter ***760***. Execution continues until it reaches the breakpoint at line ***159*** for the 750th time.

5. Select ***Memory*** from the ***Processor Views menu*** and (Figure 7).

5.1 Addresses and contents of variables on Figure 19 shows that addresses of interest are in the region of ***0x07FFFFD0***, so set the ***Start address*** value to, say, ***0x07FFFF00***.

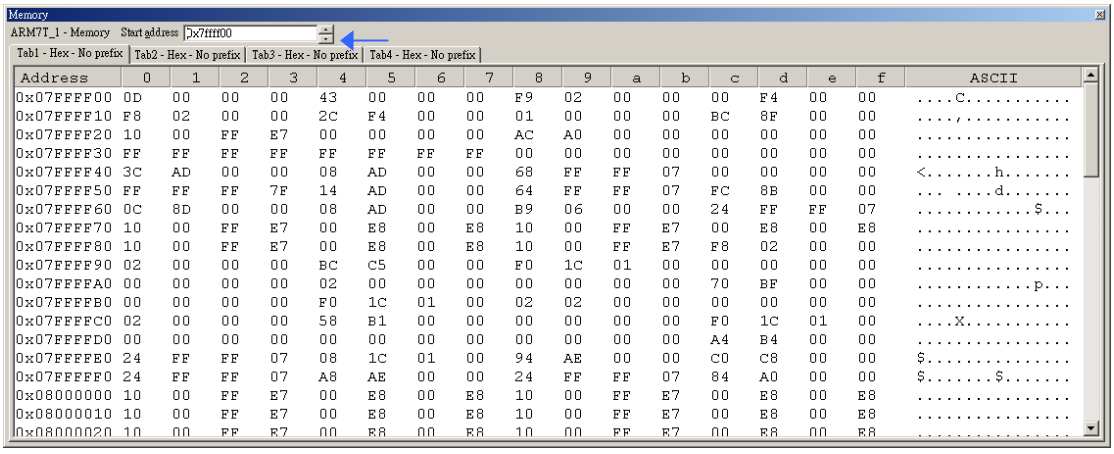

Figure 19. Examining contents of memory.

6. Press ***F10*** (or ***Step*** from the Execute menu). The program executes a single instruction and stops. Any values that have changed in the ***Memory processor view*** are displayed in red.

7. Press ***F10*** a few more times. As you execute the program, one instruction at a time, you can see the values stored in several of the memory addresses change.

8. Explore the format options available in the ***Memory processor view*** pop-up menu. Size settings appear both on the pop-up menu and in the dialog displayed when you select ***Properties...*** from the pop-up menu.

Note: For a description of the display formats available, see AXD Desktop, Chapter 5 of Debuggers Guide.

6.3 Locating and changing values and verifying changes

To locate a value (of a variable or string, for example) in memory and change it:

1. Select ***Reload Current Image***

2. Select ***Go*** to reach the first breakpoint, set by default at the beginning of function main().

3. Select ***Memory*** from the ***Search menu*** (Figure 1). A window shown in Figure 20 appears.
   3.1 Enter ***2'ND*** in the ***Search*** for field, set the ***In range*** and ***to*** addresses to ***0x0*** and ***0xFFFF***,
   3.2 Select ***ASCII*** for the ***Search string type***, and click the ***Find*** button. A ***Memory processor view*** opens and shows the contents of an area of memory, with the string you specified highlighted. Reposition, resize and/or adjust the resolution of the window if necessary.
   To see a display similar to that in Figure 21, You might have to right-click in the window to display the pop-up menu and set ***Size*** to ***8 bit*** and ***Format*** to ***Hex - No prefix***.
   3.3 Click ***Cancel*** to close Search Memory Window.

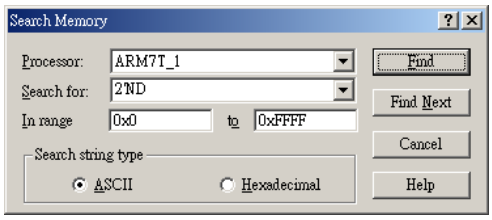

Figure 20. Search for a string in memory.

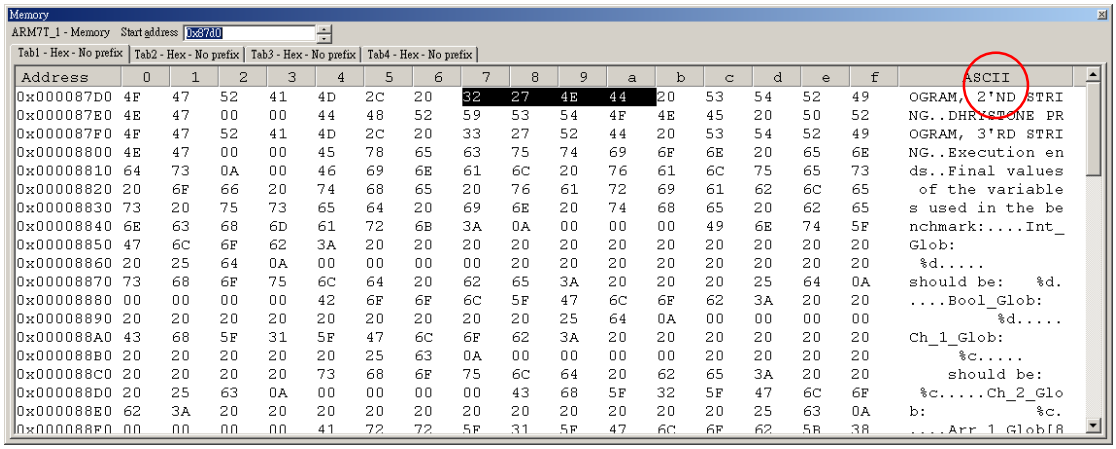

Figure 21. Search for string in memory (0x87d0).

4. In ***Memory processor view***, the four hexadecimal values highlighted are ***32 27 4E 44***.
   4.1 An example of entering a ***hexadecimal value***: Double-click on the value ***32*** and type ***0x4E*** and press ***Enter***. The corresponding change in ASCII column will be "***2'ND***" ➔ "***N'ND***".

4.2 An example of entering an ***ASCII value***: Double-click on the value ***27*** and type ***"o*** (a double quote followed by a lowercase letter o) and press ***Enter***. The corresponding change in ASCII column will be "***N'ND***" ➔ "***NoND***".

4.3 An example of entering a ***decimal value***: Double-click on the value ***4E*** (the one before ***44***) and type ***46*** and press ***Enter***. The corresponding change in ASCII column will be "***NoND***" ➔ "***No.D***".

4.4 An example of entering an ***octal value***: Double-click on the value ***44*** and type ***o62*** and press ***Enter***. The corresponding change in ASCII column will be "***No.D***" ➔ "***No.2***".

5. Press ***F5*** (or ***Go***) to continue execution, and enter a value of, say, ***100*** when you are prompted in the Console processor view for the number of runs to perform.

When the program displays its messages after completing its tests you can see that one of the lines that in earlier examples included the text ***2'ND*** STRING now has ***No.2*** STRING instead because of the change you made.

Note: In this example, the change you made was not permanent, because you did not alter the source code or the executable image stored in a disk file. You altered only the temporary copy of the image in the target memory.