

## Configuring Keil $\mu$ Vision3 and Raisonance RIDE C Compilers for Use with Ramtron's High Performance 8051 MCUs

### 1 Abstract

This technical bulletin addresses how to configure Keil  $\mu$ Vision3 and Raisonance RIDE C compilers for use with Ramtron's fast and flexible Versa 8051 microcontrollers (VRS51L2xxx/3xxx). It also demonstrates how to generate the necessary output files for the Versa Ware JTAG Debugger.

Commercial C compilers support an array of device derivatives with various features and memory sizes. For this reason they can be tailored to generate code in many specific conditions. They, therefore, include many configuration parameters, which not only impacts the generation of binary/Hex files, but also the content of the output files generated during the compilation process.

### 2 Hardware for Debugging

VRS51L2xxx/3xxx devices include a hardware debugging feature that enables real-time in-circuit debugging, whether the device is being used in the development system or the final application. Contrary to ROM monitors available for standard 8051 parts, the hardware debugging feature on these devices allows the user program to run at full-speed.

Some of the supported debugging functions include:

- The ability to set breakpoints
- Single-step operation
- SFR register content read/write
- Internal and external SRAM content read/write
- Code display in C, assembler, or both

The VRS51L2xxx and VRS51L3xxx incorporate a JTAG interface for in-circuit programming and in-circuit debugging.

### 3 Software for Debugging

Ramtron has developed a Windows™-based application that supports device programming and in-circuit debugging called the:

- Versa Ware JTAG for JTAG-based derivatives such as the VRS51L2xxx/3xxx

Versa Ware JTAG were developed for Windows XP™, however, they will also run on Windows 2000™, Windows Millennium™, Windows 98™ and Windows NT™ 4.0 operating systems, as well as on a PC that meets the following system requirements:

- Intel Pentium® III 400 MHz
- 128MB of RAM
- 10MB hard drive space
- A free IEEE 1284 parallel port\* (Versa Ware JTAG)
- A USB connector (for Versa Ware JTAG)
- A session of Windows™ running in administrator mode for installation (except on Windows 98™)

*Note: Only built-in motherboard parallel port and ISA parallel port cards are supported. The Versa JTAG interface must be connected on LPT1. Please see Section 9 of the Versa Ware JTAG User Guide for more details on parallel port setup and supported modes.*

In order to display source code, the Versa Ware Debugger interface relies on the .LST file and other files generated by the compiler. There is no standard list file format and each compiler generates files in its own manner. Versa Ware JTAG currently supports list files generated by the following compilers:

- Keil  $\mu$ Vision3
- Small Device C Compiler (SDCC)
- Raisonance Kit 6.1

For device programming, any compiler or assembler that can generate an Intel Hex output file should work with the Versa Ware JTAG software.

If you are not using one of the compilers listed above, please email us at [mcuinfo@ramtron.com](mailto:mcuinfo@ramtron.com) and we will verify whether your compiler is supported.

**Note:** In order for the Versa Ware JTAG interface to properly start and display the source code, it is important to ensure that all the compiler output files are in the same directory.

#### 4 Configuring the Keil $\mu$ Vision3 Compiler

Keil  $\mu$ Vision3 requires that a project be created in order to compile a source file. To create a new project, select menu item "Project  $\rightarrow$  New Project".

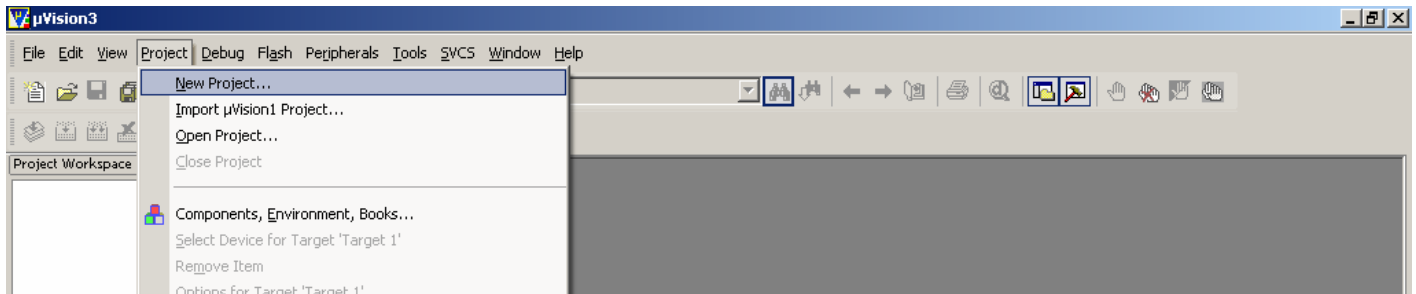


Figure 1: Creating a New Project in  $\mu$ Vision3

A file dialog box will appear and allow you to create a project file.

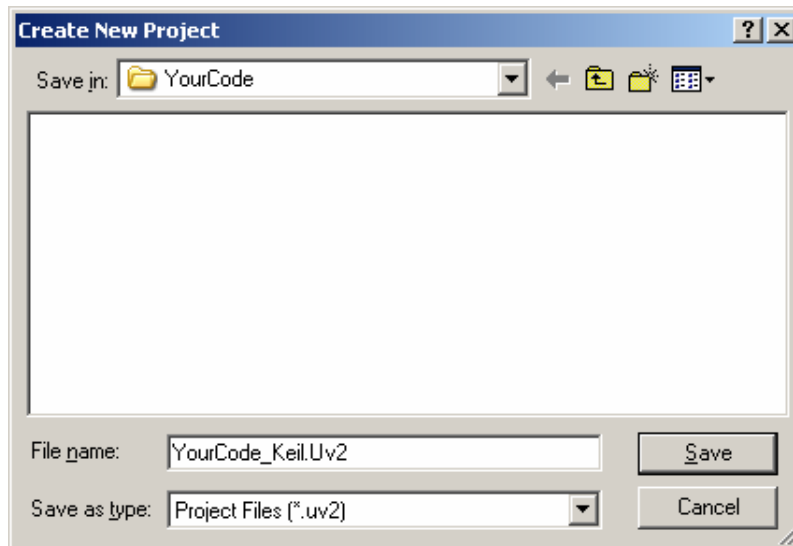


Figure 2: Create New Project Dialog Box

Once the project name and target directory are defined, click <Save>. Now you will be prompted to select the target device.

To develop code for the VRS51L2xxx/3xxx devices we recommend selecting Generic  $\rightarrow$  8052 as the Target CPU.

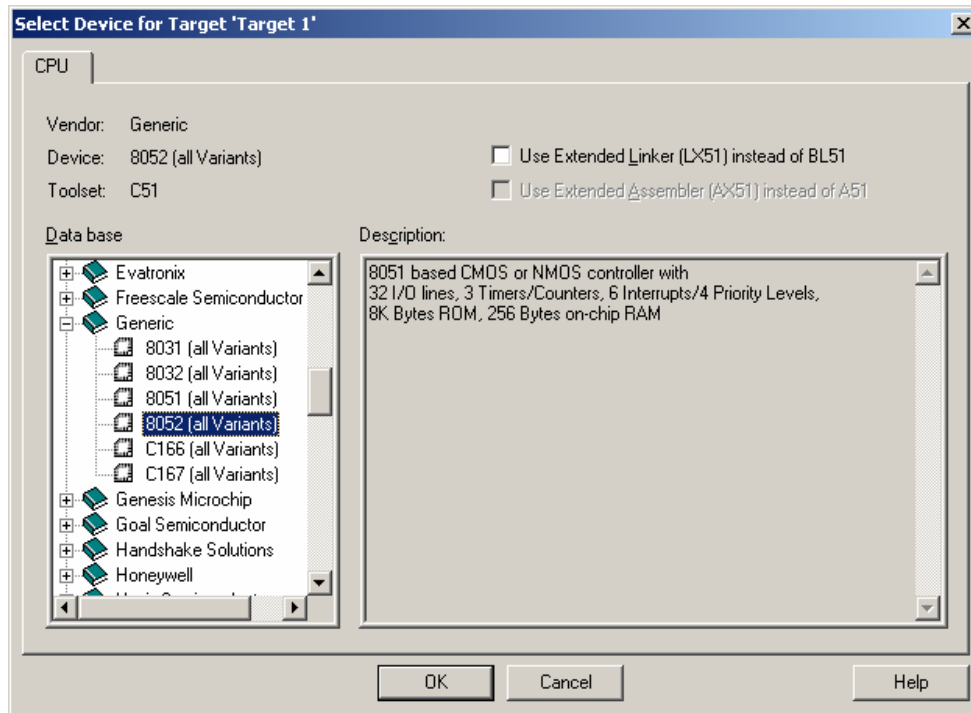


Figure 4: Device Selection for VRS51L2xxx/VRS51L3xxx

After you have selected the target device, click <OK>. You will now be asked if you want to copy the Standard 8051 Startup Code, which clears IRAM and XRAM to

0x00 upon program startup. This parameter does not affect debugger compliance or performance.

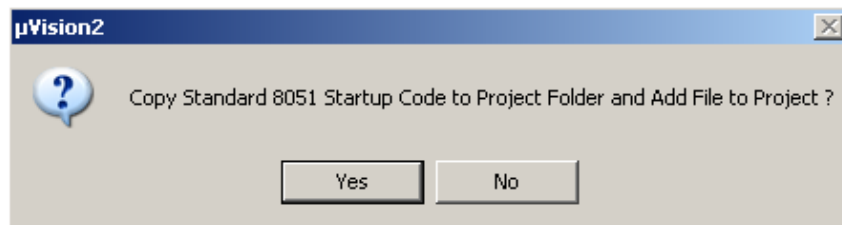


Figure 5: Copying Standard 8051 Startup Code

At this point µVision3 will return to the main window. To add your source file to the project, click “+” next to “Target 1” in “Project Workspace”, click “+” next to

“Source Group”, and right click “Source Group1”. An options menu will appear. Select “Add Files to Group ‘Source Group 1’”.

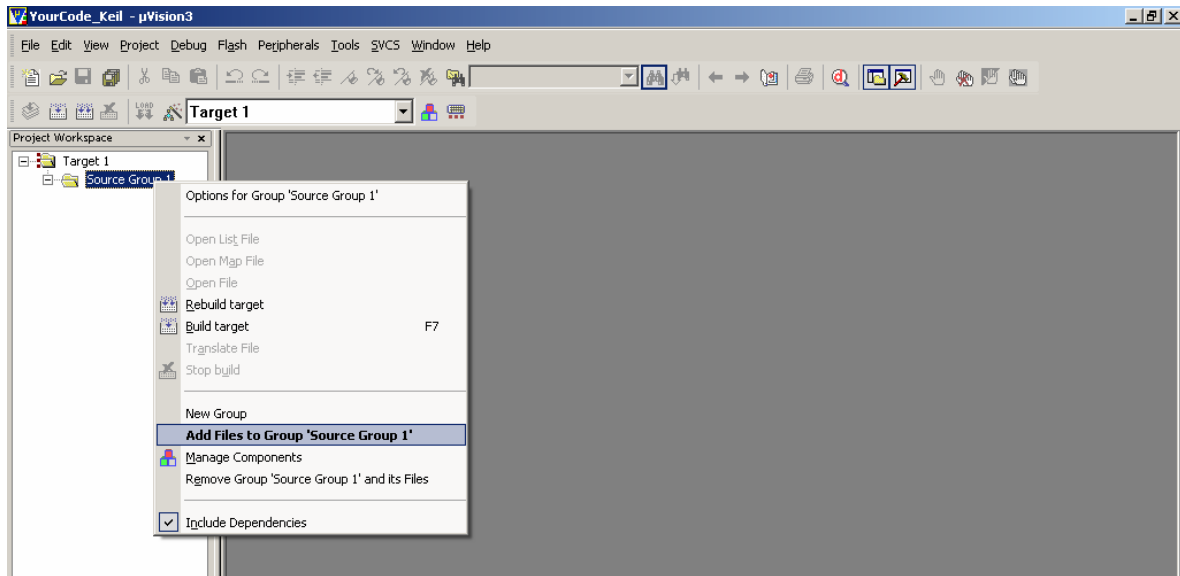


Figure 6: Adding Files to a Project

You will now be prompted to select the source file(s) for inclusion in the project.

Select the file(s) and click <Add>. Click on <Close> to complete the process.

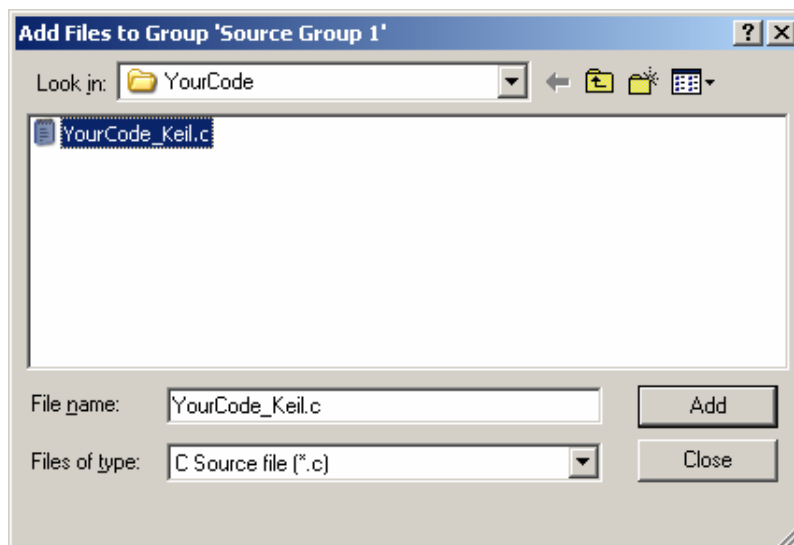


Figure 7: Adding Files to the Project Dialog Box

#### 4.1 Configuring $\mu$ Vision3 Compiling and Target Device Options

The following steps involve configuring the  $\mu$ Vision3 compiler to:

- Generate a Hex file output
- Generate appropriate LST files for the project

Select Project → Options for Target 'Target 1' to open the "Option for Target 1" configuration panel. Here you

can select the required memory model and code ROM size. Editing the "Xtal" parameters is optional. Leave the "Off chip code memory" section blank.

The VRS51L2xxx/3xxx includes XRAM memory and XDATA memory bus access. We recommend entering the XRAM parameters in the "Off-Chip Xdata memory" section. Figure 9 demonstrates the suggested configuration for these high performance Versa 8051 devices.

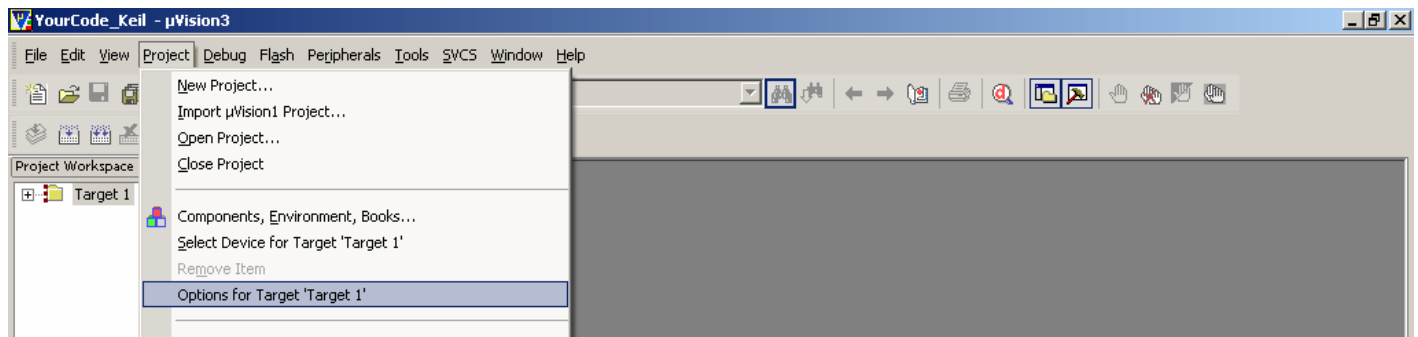


Figure 8: Options for the Target Device

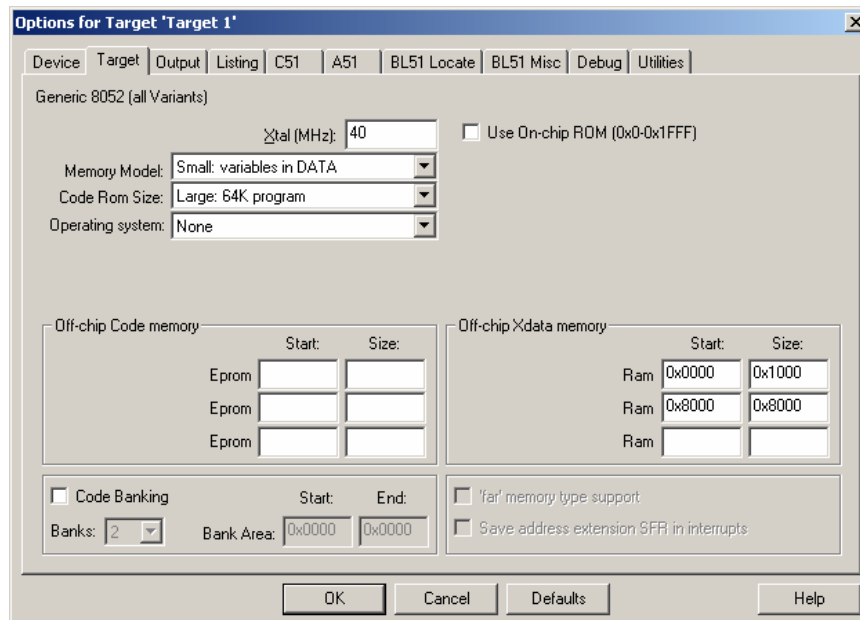


Figure 9: Configuring the Target Device Options

Open the “Output” tab in the “Options for Target...” window.

- Select the “Create Executable” option
- Check “Debug Information” and “Browse Information”
- Check “Create HEX File” and “Create HEX File”
- Position the “HEX Format” drop down menu at Hex-80

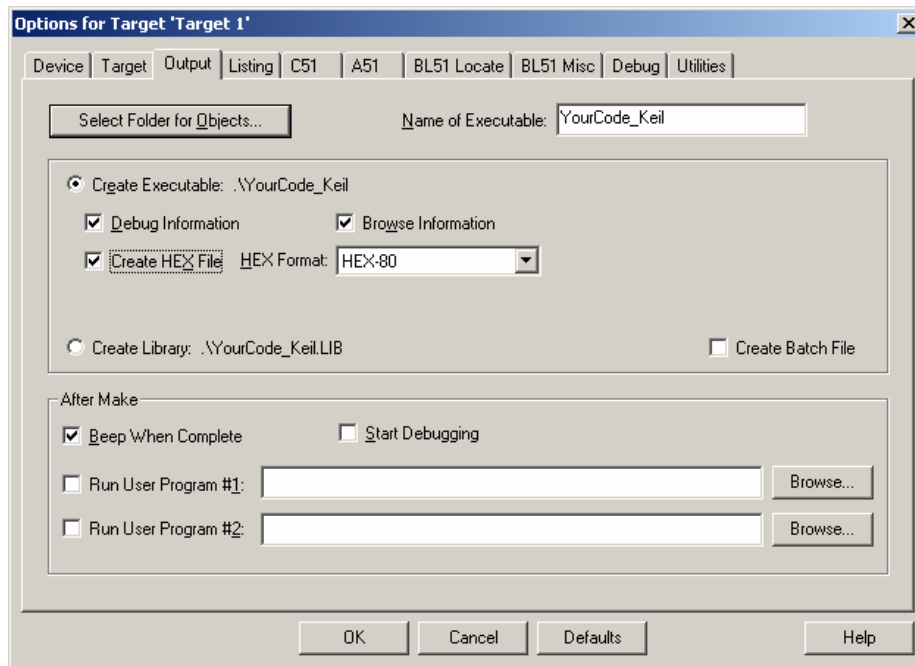


Figure 10: Output File Configuration

Keil  $\mu$ Vision3 also provides the ability to run a user program after the compilation process is complete. This feature can be used to upload the updated Hex file into the VRS51L2xxx/3xxx. We recommend using Versa Ware JTAG for device programming and in-circuit debugging because this software provides a convenient and user-friendly interface

Ramtron has also developed a command-line utility that can be used for device programming; VWJTAG is the command-line version of Versa Ware JTAG that

can be used to program the device with the new Hex file that has been created in the compilation process. This software can be called from  $\mu$ Vision3 after compilation. To enable this feature, place the software path into the “Run User Program” edit box in the “After Make” section, as shown in Figure 11.

VWJTAG provides a variety of configuration options that can be found in the Read Me file in the JTAG software folder (C:\Program Files\Ramtron\Versa Ware JTAG) or in the Start Menu.

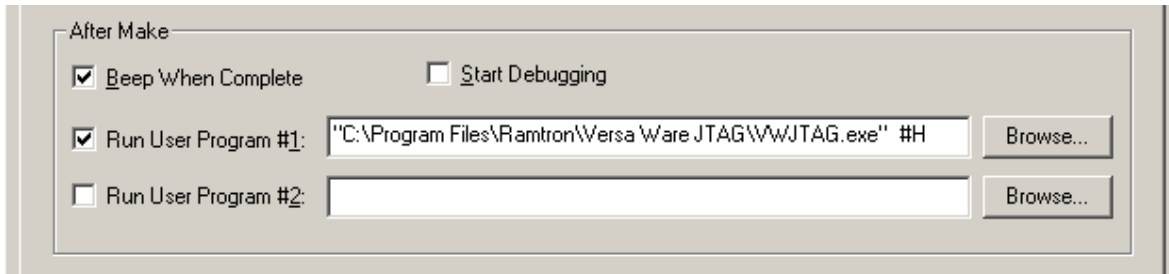


Figure 11: Performing Device Programming after Compilation

**Warning:**

VWJTAG.exe does not support in-circuit debugging. To use the debugger, it is recommended to deactivate automatic Flash programming using the VWJTAG.EXE after compiling, and run the Versa Ware JTAG software instead.

VWARE.EXE deactivates the debugging feature on the VRS51L2xxx/3xxx. It is, therefore, recommended to perform an Erase + Program before launching the Versa Ware JTAG Debugger to ensure that the debugging features on the VRS51L2xxx/3xxx are properly configured.

Attempting to launch the Versa Ware Debugger without performing an Erase + Program of the device will likely

cause the sudden termination of the Versa Ware JTAG software, since the VWARE.EXE deactivates the debugging feature on the VRS51L2xxx/3xxx.

Next, select the “Listing” tab in the “Options for Target” window and make sure the follow checkboxes are selected:

- “C Compiler Listing...”
- “Conditional”, “Symbol”
- “Assembly Code” boxes (if the “Assembly Code” option is left unchecked, there will be no code display in the debugger window)

Click <OK> to return to the main window of the Keil  $\mu$ Vision3 IDE. Begin editing and compiling the source file by selecting “Build Target” from the “Project” menu.

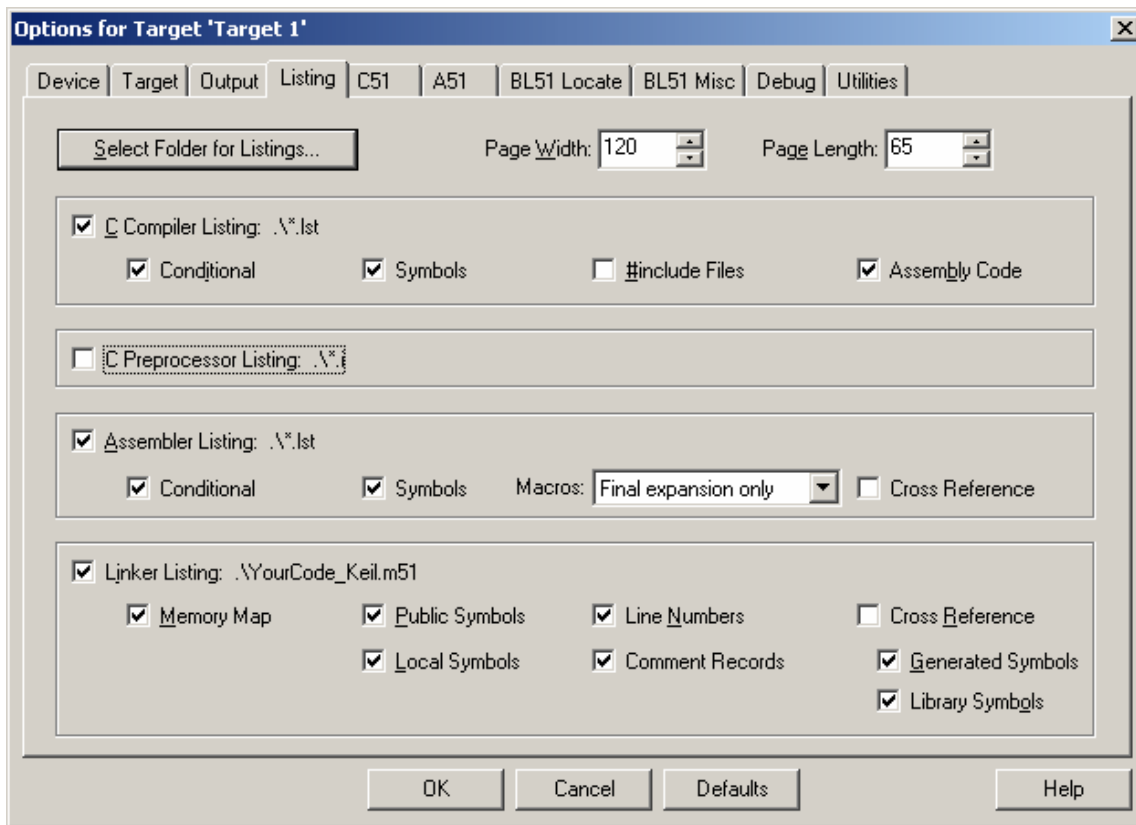


Figure 12: Configuring Output Files

#### 4.2 Compiling Code Using Keil $\mu$ Vision3

Once the compiling and output file options are configured. You may begin editing the source code. Double click on the previously added source file displayed in the Projects workspace pane of the

$\mu$ Vision3 main window (see Figure 13) to display the code in the editor panel.

To begin compiling the project, select menu item Project → Build Target or press <F7>. Compilation results will appear in the output window in the bottom section of the main window.

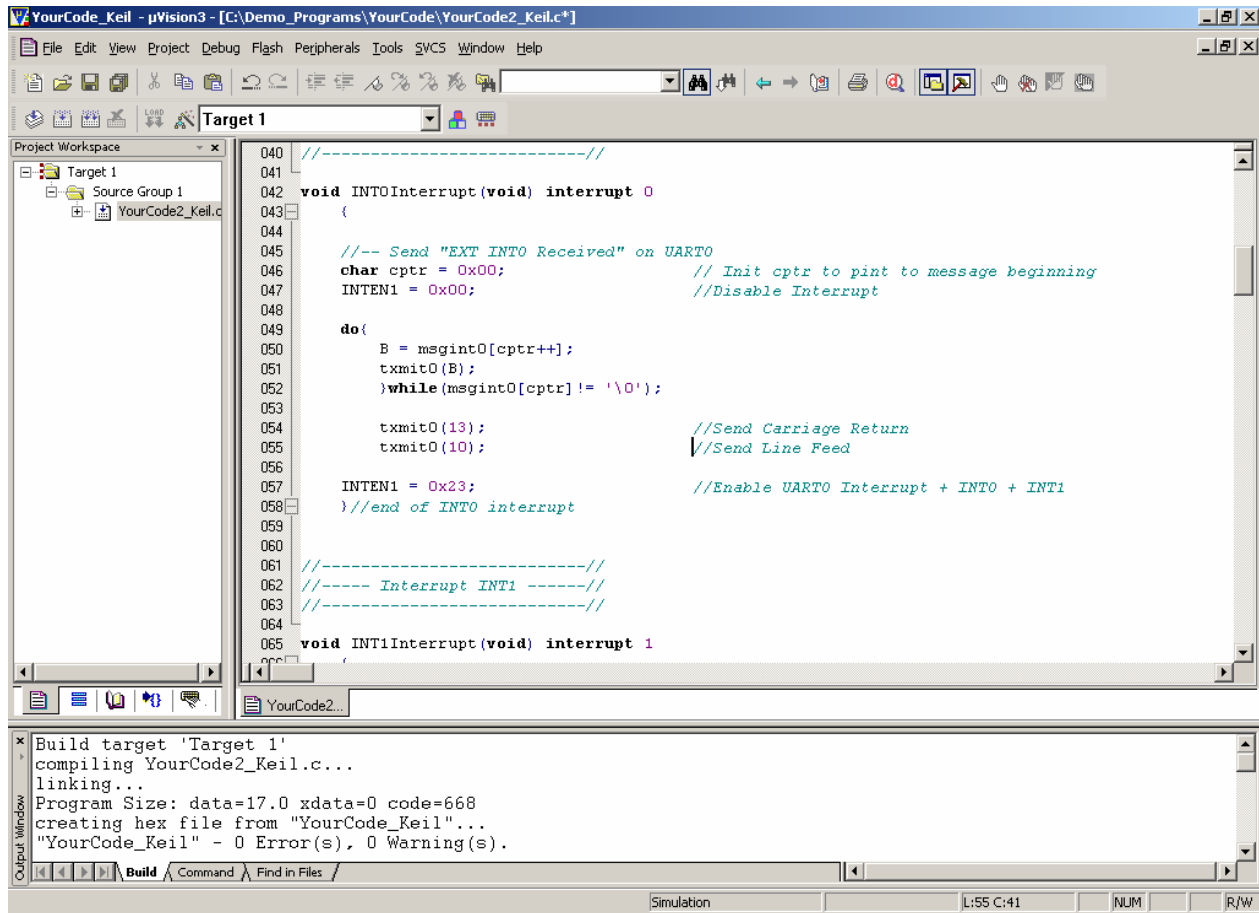


Figure 13: Compiling the Source File

If  $\mu$ Vision3 is configured to call the VWJTAG.EXE programming software after the file compilation is complete, the output window will display the status of

the device programming as shown in Figure 14. The programming status is displayed at the end of the programming process.

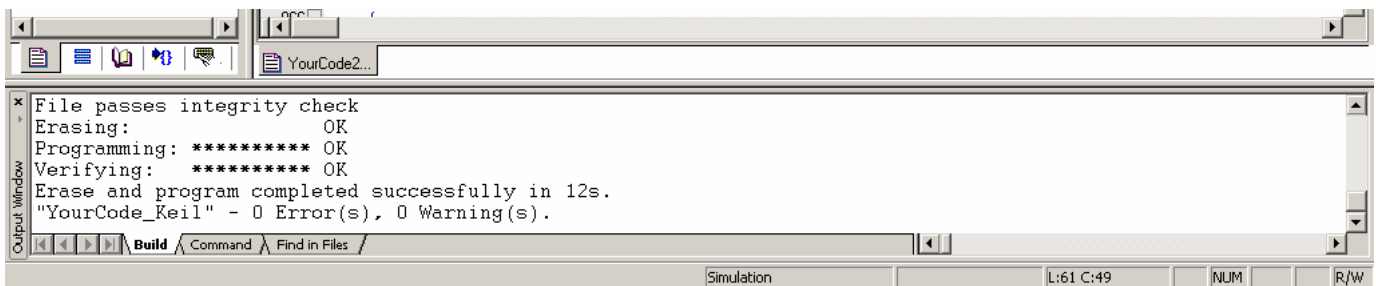


Figure 14: Compiling the Source File with the Device Programming Option Activated

## 5 Configuring Raisonance Kit 6.1

The Raisonance Kit (RIDE) compiler can compile files without having to create a project. The configuration steps are essentially the same whether a project is created or not.

### 5.1 Creating a Project File in RIDE

To create a project, select Project → New and enter the project parameters. Select 80C51 as the “Target family”. Click <Next> to open the “Target” window and select the project’s target device: Philips → P80C52.

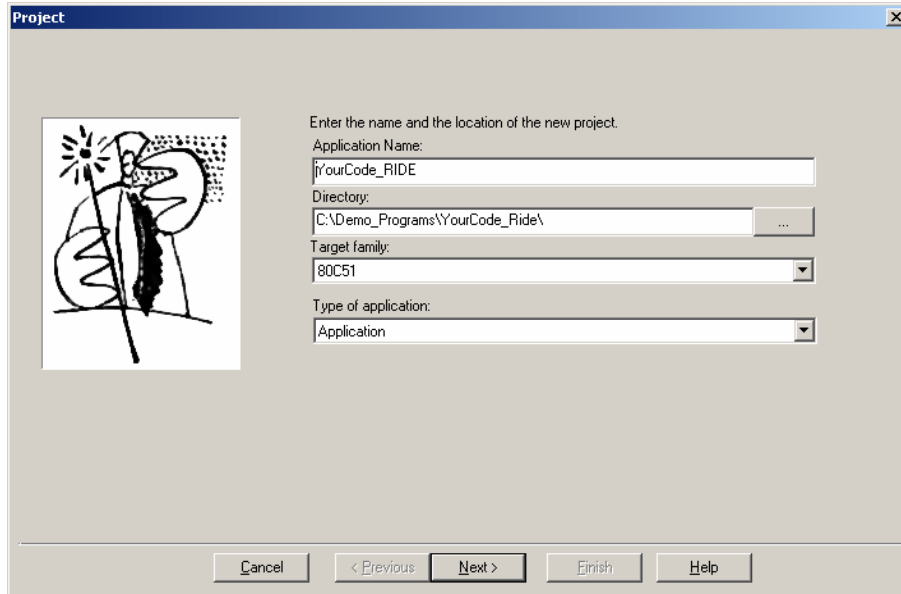


Figure 15: Creating a Project with RIDE

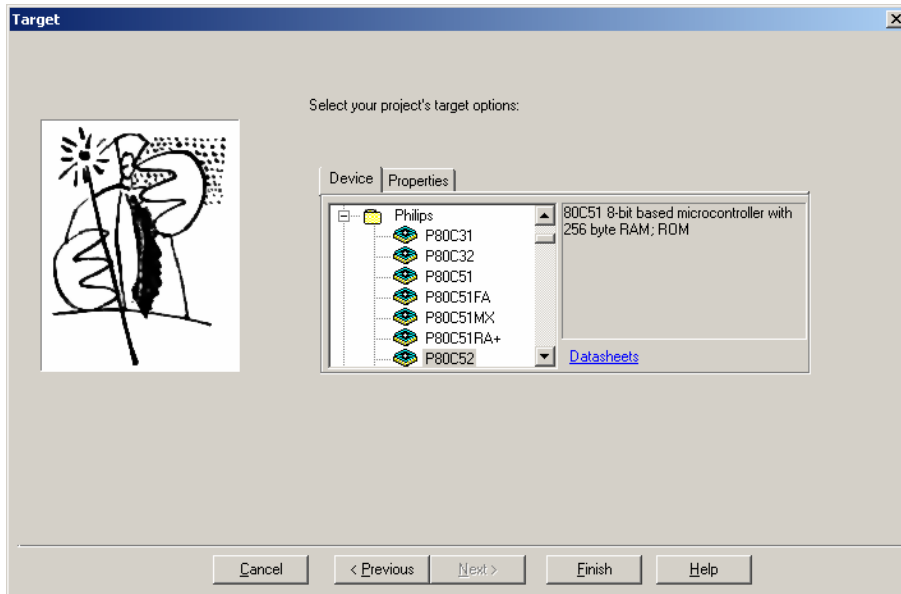


Figure 16: Selecting a Target Device

Click <Finish> to return to the RIDE main window.  
 Open Project → Add node Source/Application and

select the sources files to be included in the project.

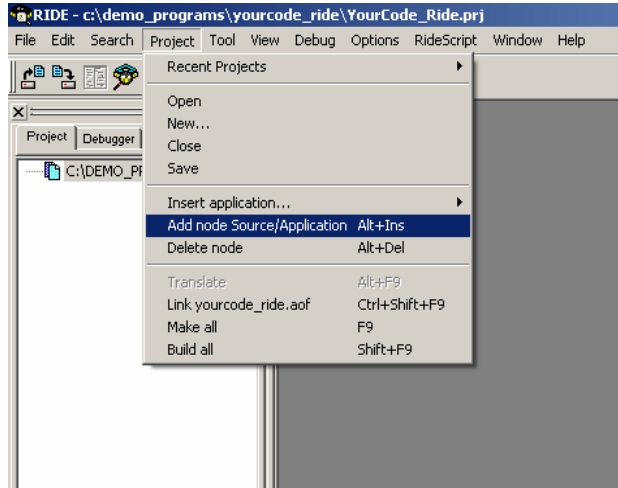


Figure 17: Adding Source File(s) to a Project

**5.2 Compiling without Creating a Project File**

To compile a file without creating a project, begin by creating or opening the source file via File → New or

File → Open. Once the source file is created, select the Option → TarGet (see below) to open the “Core Selection” dialog box. Select Philips → P80C52 as the target device (see Figure 19).

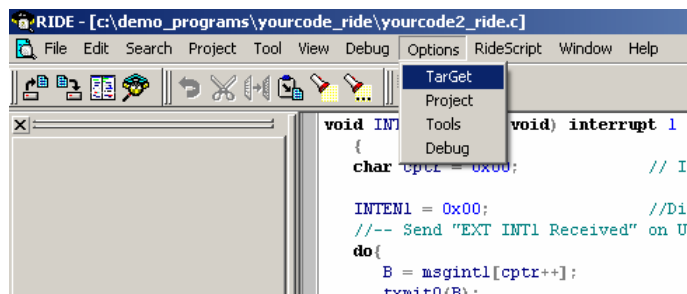


Figure 18: Preparing to Select a Target Device

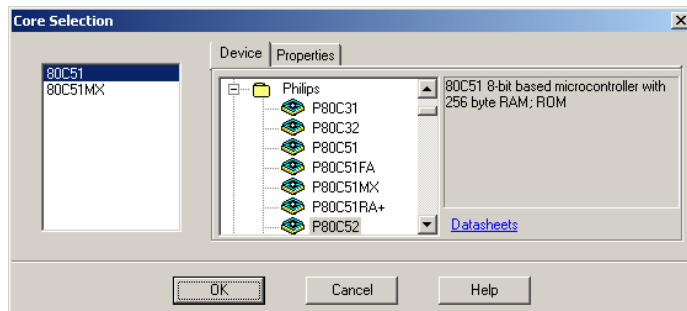


Figure 19: Selecting the Target Device

**5.3 Configuring RIDE Compilation and Output File Options**

Next, select Options → Project to open the “Options” dialog box.

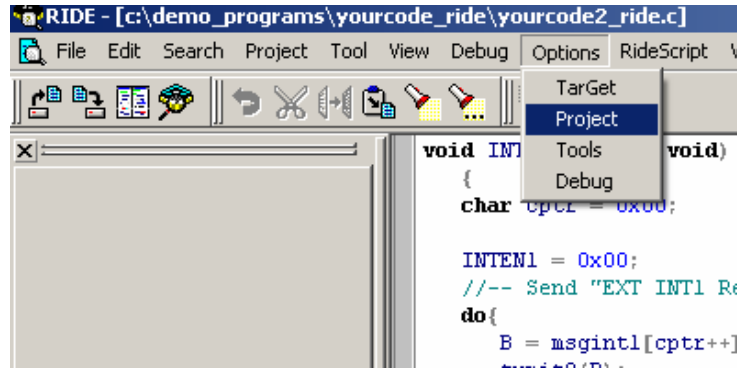


Figure 20: Configuring Compiling Options

Click on “+” next to RC51 to display the options associated with the C compiler and select “Listing”.

Select the options on the right side of the dialog box, as shown in Figure 21.

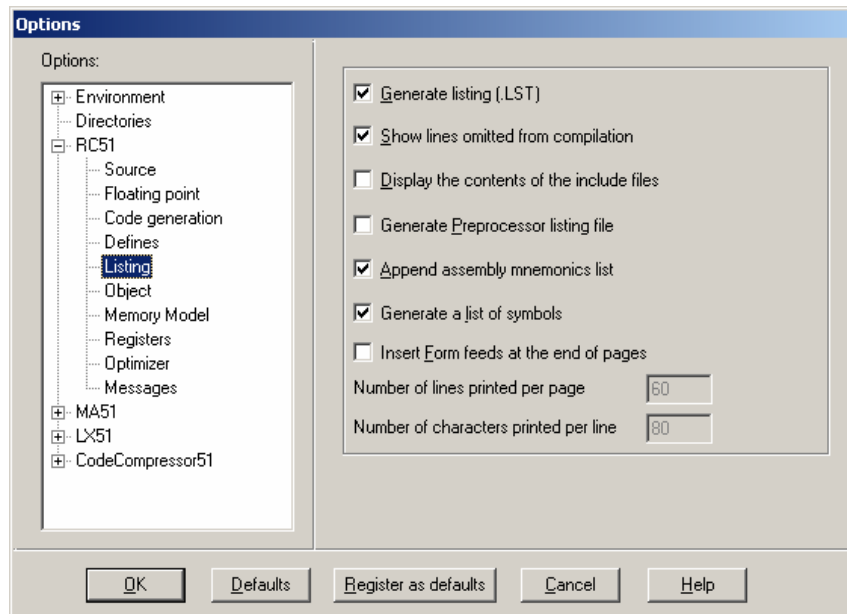


Figure 21: Configuring Output Listing

The RIDE compiler can use the second data pointer available on the VRS51L2xxx/3xxx for better code

optimization. To activate this feature, select “Philips Component with Dual DPTR” in the “Advanced features” section (Figure 22).

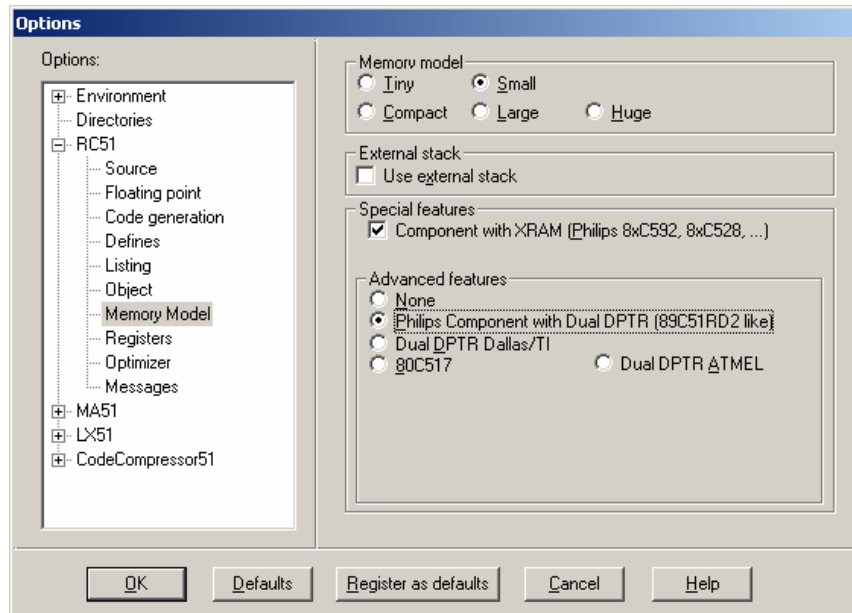


Figure 22: Configuring the Compiling Memory Model

To display the options associated with the RIDE Macro Assembler, click “+” next to MA51 and select

“Listing”. Select the options, as shown in Figure 23.

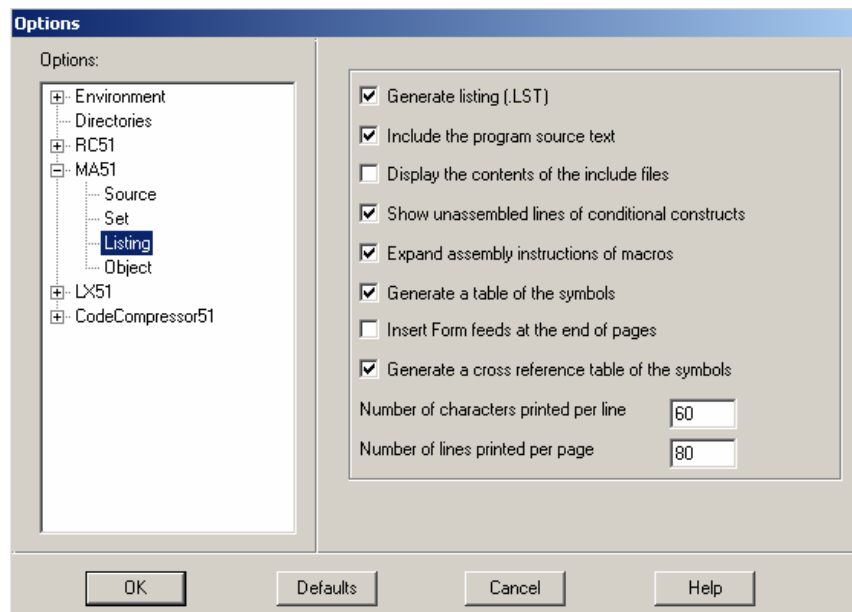


Figure 23: Configuring the Assembler Output Listing Parameters

Configure the target device’s memory via the Linker section. The default parameters will function in most situations.

Specifying the Xdata parameters allows the compiler to generate a warning message when the program points to an undefined area of xdata.

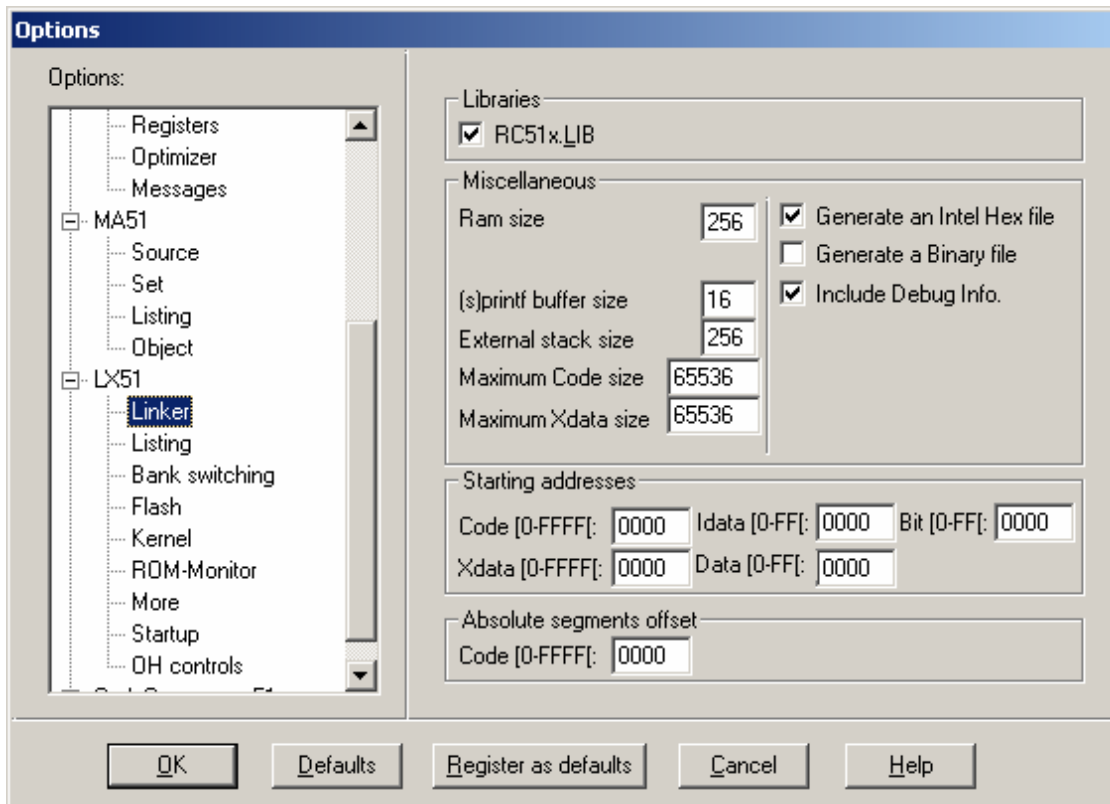


Figure 24: Configuring the Linker Options

Click <OK> to return to the RIDE main window. Begin editing and compiling your code. When compilation of the source code has been completed without error, upload it to the Versa 8051 device and run the debugger using Versa Ware JTAG.

## 6 Configuring the SDCC Compiler

The SDCC compiler is a command-line-based C compiler. It can be used with external text editors such

as PSPad, Crimson, or Syn. SDCC generates compact code with no code size limitations.

Documents on how to configure various text editors with SDCC are available on the Ramtron web site at [www.ramtron.com/doc/Products/Microcontroller/Support\\_Tools.asp](http://www.ramtron.com/doc/Products/Microcontroller/Support_Tools.asp)

## 7 Compiler Configuration for Use with Versa Ware Debuggers

The following section provides details on how to configure compilers for use with the Versa Ware JTAG Debugger.

### 7.1 Versa Ware JTAG

#### 7.1.1 All compilers (SDCC, RIDE, Keil $\mu$ Vision3)

- Output must be in the Intel-Hex80 format.
- Debug info must be present. The file must not have an extension and must be the same name as the Hex file.
- All files must be the same directory.

#### 7.1.2 Keil $\mu$ Vision3

- Output must be in the Intel-Hex80 format (off by default in Keil). This must be activated each time a project is created.
- Assembly code in the C compiler .LST must be checked and/or `#pragma SRC` must be specified (off by default). This must be activated each time a project is created.
- An .M51 file must be generated and have all options checked (cross-reference is optional and on by default).
- The project must be compiled/assembled/linked in the same location it will be debugged.
- When mixing assembler and C code or using more than one assembler file in a project, we recommend using `CSEG AT xxxxxH` instead of `ORG xxxxxH` to define the beginning of a non-default code segment. This defines absolute module segments and links properly. `ORG` statements are ignored and generate overlap warnings in Keil  $\mu$ Vision3.

#### 7.1.3 RIDE

- The project must be compiled/assembled/linked in the location it will be debugged in.

#### 7.1.4 SDCC

For SDCC projects with multiple source files:

- All .C files must be compiled with the `--debug` option.
- All assembler **MUST** be written as inline assembler.
- Compile with `--debug` for extensive variable extraction and faster loading.