C51 系列微控制器的开发工具

# uVision2 入门教程

使用指南０２．２００１

# Keil Software 声明：

本文档所述信息不属于我公司的承诺范围，其内容的变化也不会另行通知。本文档所述软件的出售必须经过授权或签订特别协议。本文档所述软件的使用必须遵循协议约定，在协议约定以外的任何媒体上复制本软件将触犯法律。购买者可以备份为目的而做一份拷贝。在未经书面许可之前，本手册的任何一部分都不允许为了购买者个人使用以外的目的而以任何形式和任何手段(电子的，机械的)进行复制或传播。



Keil C51（TM）和 uVision（TM）是 Keil Elektronik GmbH 的商标。
Microsoft（R）和 Windows（TM）是 Microsoft Corporation 的商标或注册商标。
PC（R）是 International Business Machines Corporation 的注册商标。

---

*注意*
*本手册假定你已经熟悉微软操作系统和 8051 系列产品的硬件和指令集。*

---

我们尽全力去做来保证这本手册的正确，从而保证我们个人，公司和在此提及的商标的形象。

# 前言

这本手册是 Keil Software 公司关于 8051 系列 MCU 的开发工具的介绍。它向新用户和有兴趣的读者介绍本公司的产品。这本使用指南包含下列各章：

第 1 章 简介：概述并描述了 Keil Software 为 8051 系列 MCU 提供的不同产品。

第 2 章 安装：描述了该如何安装软件以及如何设置工具的操作环境。

第 3 章 开发工具：描述了集成有调试器，C 编译器，汇编器的 uVision2 IDE 的主要特性和用途。

第 4 章 建立应用：描述该如何建立项目，编辑源文件，编译并报告语法错误， 产生运行代码。

第 5 章 测试程序：描述了如何使用 Vision2 debugger 模拟并测试你的整个应用。

第 6 章 调试功能：讨论了扩展 uVision2 debugger 功能的各种函数。

第 7 章 示例程序：提供几个示例程序以说明该如何使用 Keil 8051 开发工具。

第 8 章 实时操作系统：讨论了 RTX-51 Tiny 版和 RTX-51 Full 版，并提供一个示例程序。

第 9 章 使用片上外围设备：描述了如何使用 C51 编译器访问片上外围设备，本章也包括几个应用注意事项。

第 10 章 CPU 和程序启动代码：描述了如何为你的应用设置 8051CPU。

第 11 章 使用 Monitor-51：讨论该如何初始化 Monitor 并把它安装到你的目标板上。

第 12 章 命令参考：简单地介绍了 Keil 8051 开发工具的命令和控制。

# 本文档中使用如下约定:

| 举例 | 描述 |
| --- | --- |
| **README.TXT** | 黑粗体用来表示执行文件，数据文件，源文件，环境变量和你在命令提示行键入的命令。<br>这些文字往往表示你必须按照字面的字符键入，如：**CLS DIR BL51.EXE** |
| `Courier` | 这种形式的字体用来表示在屏幕或打印机上出现的信息. |
| *Variables* | 斜体字表示必须由你提供的信息，如：在语法字符串中的"*projectfile*"表示你必须提供实际的项目名称。少数情况下斜体字也用来表示强调。 |
| Elements that Repeat. . . | 省略号（. . .）表示一个你可以替换的内容。 |
| Omitted code<br>:<br>: | 垂直的省略号用来在源程序列表中表示一段被忽略的程序。如：<br>Void main (void) {<br>:<br>:<br>while (1); |
| [*Optional Items*] | 方括号表示命令行或输入域中的可选项。如：<br>C51 TEST.C PRINT [(filename)] |
| { *opt1* \| *opt2* } | 包括在大括号中的被'\|'分开的文字表示一组选项，必须从中选一。 |
| **Keys** | 以 sans serif 字体出现的字符表示键盘上实际的键,如:<br>"Press **Enter** to continue." 中的 **Enter** 表示键盘上的回车键. |
| Point | 移动鼠标，直到光标直到期望的条目上。 |
| Click | 单击鼠标. |
| Drag | 鼠标拖动操作. |
| Double-Click | 双击鼠标. |

# 目录

# 第 1 章　简介

感谢您允许 Keil Software 为您提供 8051 系列单片机的软件开发工具，利用本工具您可以开发所有 8051 系列单片机的嵌入式应用。

---

***注意***

*尽管我们在本手册中称它为 8051 开发工具，其实它支持所有的由 8051 家族派生而来的类型。*

---

Keil Software 的 8051 开发工具提供以下程序，你可以用它们来编译你的 C 源码，汇编你的汇编源程序，连接和重定位你的目标文件和库文件，创建 HEX 文件，调试你的目标程序。从 21 页开始的第三章“开发工具”一章中将对每一个程序进行详细描述。

- Windows 应用程序 uVision2 是一个集成开发环境，它把项目管理，源代码编辑，程序调试等集成到一个功能强大的环境中。
- C51 美国标准优化 C 交叉编译器从你的 C 源代码产生可重定位的目标文件。
- A51 宏汇编器从你的 8051 汇编源代码产生可重定位的目标文件。
- BL51 连接/重定位器组合你的由 C51 和 A51 产生的可重定位的目标文件，生成绝对目标文件。
- LIB51 库管理器组合你的目标文件，生成可以被连接器使用的库文件。
- OH51 目标文件到 HEX 格式的转换器从绝对目标文件创建 Intel HEX 格式的文件。
- RTX-51 实时操作系统简化了复杂和对时间要求敏感的软件项目。

在 16 页的产品一览中将对由这些工具组成的“开发套件”进行描述。它们是为专业开发人员而设计的，但所有层次的编程人员都可以用它们来获得 8051 微控制器的绝大部分应用。

## 手册主题

本手册讨论的主题有：

- 怎样为你的应用选择最好的工具包，参照 16 页的“产品一览”。
- 怎样在你的系统上安装本软件，参照 16 页“安装”。
- 本开发工具的特征， 2 1 页。
- 怎样用 uVision2 IDE 创建一个完整的应用，57 页。
- 怎样调试程序，怎样用 uVision2 调试器模拟你的目标硬件，93 页。
- 在 C51 编译器中该如何访问片上外围设备和 8051 派生系列产品的特殊功能，114 页。
- 怎样运行示例程序，149 页。

---

*注意*
*为了立即开始，请参照第二章安装软件，然后参照第七章运行示例程序。*

---

## 本文档的最后改动

本软件和手册最后一刻的变化和修改在 **RELEASE.TXT** 中，位于**\KEIL\UV2** 和 **\KEIL\C51\HLP** 文件夹中。

花点时间读一下这些文件，看看这些变化和修改是否对安装产生影响。

## 测试版工具包和产品工具包

Keil Software 把软件分成两种类型，测试版和正式版。

测试版：包括 8051 工具的测试版本和本用户手册，你可以用它们产生目标代码小于 2K 字节的应用。
此套件主要是让你测试我们产品的效力，并产生小的应用。

正式版：（在 16 页讨论）包括没有限制的 8051 工具和全套手册(含本手册)，正式版套件包含 1 年的免费技术支持和产品升级。升级通过 **www.keil.com** 提供。

## 用户类型

本手册针对三种用户：测试用户，新用户，有经验的用户。

测试用户是那些还没有购买本软件但已经要求使用测试开发包以进一步了解本工具和本工具的性能的用户。测试开发包包括有 2K 字节目标代码限制的工具和几个为 8051MCU 系列产品而创建的应用。即使你是一个测试用户，你最好也花点时间阅读本手册。它解释了怎样安装本软件，为你提供本开发工具的初步信息，并介绍了示例程序。

新用户是那些第一次购买本开发工具的用户。你所购买的软件为你提供最新的开发工具，技术手册和示例程序。如果你对 8051 或本工具比较生疏，花点时间学习本手册中描述的示例程序。它们为新用户和没有经验的用户快速起步提供了一个指南和帮助。

有经验的用户是指那些以前已经用过 Keil 8051 开发工具，现在升级到最新版本的用户。升级软件产品包含最新的开发工具和示例程序。

## 请求援助

Keil Software 的全体员工专注于为您提供最好的开发工具和文档资料。如果你对本手册有建议的话请跟我们联系。如果你认为你发现了一个软件上问题，请在联系技术支持中心前，按下面的步骤做：

1．阅读与你试图完成的工作或任务相关的章节。

2．确定你所用的是最新的版本，到 **www.keil.com** 核对升级内容以确定你使用的是最新版本。

3．分析所发现的问题，确定它是汇编器的问题，还是编译器，连接器，库管理器或其他的开发工具的问题。

4．进一步通过减少你的代码到几行使问题更明确。

如果你在经过上述步骤后问题仍然存在，请你向我们技术支持中心报告。请包含你的产品序列号和版本号。我们倾向于你通过 E-mail 的方式发送。如果你通过 FAX 联系，请确定包含我们可以与你联系上的你的名字和电话号码(电话和传真)。

请尽可能详细地描述你所遇到的问题。你描述的越详细，我们就能越快地找到解决办法。如果你能用仅仅一页的代码描述你遇到的问题，请把它 E-mail 给我们。如果可能，请确定你的问题能够在开发工具上重复出现。请避免发送整个应用代码或很长的代码给我们，以免延误我们对你的答复。

---

*注意*

*你总是可以从* ***www.keil.com/support*** *获得技术支持,产品升级,应用笔记和示例程序*

---

# 软件开发流程

当你使用 Keil Software 工具时，你的项目开发流程和其它软件开发项目的流程极其相似。

1、创建一个项目，从器件库中选择目标器件，配置工具设置。

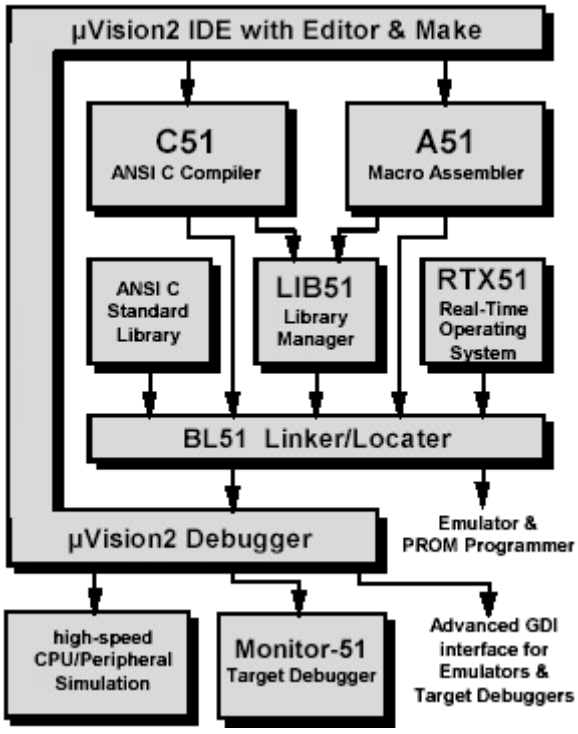
2、用 C 语言或汇编语言创建源程序。

3、用项目管理器生成你的应用。

4、修改源程序中的错误。

5、测试，连接应用。

一个完整的 8051 工具集的框图可以最好地表述此开发流程。每一个组件在下面详细描述。




## uVision2 IDE

uVision2 集成开发环境集成了一个项目管理器，一个功能丰富、有错误提示的编辑器，以及设置选项，生成工具，在线帮助。利用 uVision2 创建你的源代码并把它们组织到一个能确定你的目标应用的项目中去。uVision2 自动编译，汇编，连接你的嵌入式应用，并为你的开发提供一个单一的焦点。

## C51 编译器和 A51 汇编器

源代码由 uVision2 IDE 创建，并被 C51 编译或 A51 汇编。编译器和汇编器从源代码生成可重定位的目标文件。

Keil C51 编译器完全遵照 ANSI C 语言标准，支持 C 语言的所有标准特性。另外，直接支持 8051 结构的几个特性被添加到里面。

Keil A51 宏汇编器支持 8051 及其派生系列的全部指令集。

## LIB51 库管理器

LIB51 库管理器允许你从由编译器或汇编器生成的目标文件创建目标库。库是一种被特别地组织过并在以后可以被连接重用的对象模块。当连接器处理一个库时，仅仅那些被使用的目标模块才被真正使用。

## BL51 连接器/定位器

BL51 连接器/定位器利用从库中提取的目标模块和由编译器或汇编器生成的目标模块创建一个绝对地址的目标模块。一个绝对地址目标模块或文件包含不可重定位的代码和数据。所有的代码和数据被安置在固定的存储器单元中。此绝对地址目标文件可以用来：

- 写入 EPROM 或其它存储器件。
- 由 uVision2 调试器使用来模拟和调试。
- 由仿真器用来测试程序。

## uVision2 调试器

uVision2 源代码级调试器是一个理想地快速，可靠的程序调试器。此调试器包含一个高速模拟器，能够让你模拟整个 8051 系统，包括片上外围器件和外部硬件。当你从器件库中选择器件时，这个器件的特性将自动配置。

uVision2 调试器为你在实际目标板上测试你的程序提供了几种方法：

- 安装 MON51 目标监控器到你的目标系统并且通过 Monitor-51 接口下载你的程序。
- 利用高级的 GDI（AGDI）接口，把 uVision2 调试器绑定到你的目标系统。

## Monitor-51

uVision2 调试器支持用 Monitor-51 进行目标板调试。此监控程序驻留在你的目标板的存储器里，它利用串口和 uVision2 调试器进行通信。利用 Monitor-51，uVision2 调试器 可以对你的目标硬件实行源代码级的调试。

## RTX51 实时操作系统

RTX51 实时操作系统是一个针对 8051 系列的多任务核。RTX51 实时内核从本质上简化了对实时事件反应速度要求高的复杂应用系统的设计，编程和调试。RTX51 实时内核是完全集成到 C51 编译器中的，从而方便使用。任务描述表和操作系统的连接由 BL51 连接器/定位器自动控制。

# 产品一览

Keil Software 提供第一流的 8051 系列开发工具，我们把我们的开发工具捆绑到不同的开发包或工具套件。17 页的对照表说明了整个 Keil Software 8051 开发工具。每一个套件及其内容描述如下：

## PK51 专业开发套件

PK51 专业开发套件包括了所有专业开发人员创建和调试复杂 8051 嵌入式应用系统所要用到的一切工具。PK51 专业开发套件可以针对所有的 8051 及其派生系列进行配置使用。

## DK51 开发套件

DK51 开发套件是 PK51 专业开发套件的精简版本。它不包括小型 RTX51 实时操作系统。此套件可以针对所有的 8051 及其派生系列进行配置使用。

## CA51 编译套件

CA51 编译套件是那些需要 C 编译器而不需要调试系统的开发人员的最好选择。CA51 开发包仅仅包含 uVision2 IDE。uVision2 调试器不包括在内。此套件可以针对所有的 8051 及其派生系列进行配置使用。

## A51 汇编套件

A51 汇编套件包括一个汇编器和你创建嵌入式应用所需要的所有功能。此套件可以针对所有的 8051 及其派生系列进行配置使用。

## RTX51 实时操作系统（FR51）

RTX51 实时操作系统是一个 8051 系列 MCU 的实时内核。RTX51 FULL 提供 RTX51 TINY 的所有功能和一些扩展功能，并且包括 CAN 通信协议接口。

## 开发套件和工具的对照表：

利用此表选择你所需要的开发套件.

| Components | PK51 | DK51† | CA51 | A51 | FR51 |
|---|---|---|---|---|---|
| µVision2 Project Management & Editor | ✓ | ✓ | ✓ | ✓ | |
| A51 Assembler | ✓ | ✓ | ✓ | ✓ | |
| C51 Compiler | ✓ | ✓ | ✓ | | |
| BL51 Linker/Locator | ✓ | ✓ | ✓ | ✓ | |
| LIB51 Library Manager | ✓ | ✓ | ✓ | ✓ | |
| µVision2 Debugger/Simulator | ✓ | ✓ | | | |
| RTX51 Tiny | ✓ | | | | |
| RTX51 Full | | | | | ✓ |

# 第 2 章　安装

本章解释如何设置操作环境以及如何在你的硬盘上安装本软件。在开始安装程序之前请：

- 确认你的计算机系统符合最小的需求。
- 制作一份安装盘的副本。

## 系统需求

- 为了取得比较好的运行效果，最低的硬件和软件配置必须满足：
- 具有奔腾，奔腾 II 或兼容的处理器的个人计算机。
- 操作系统为 WIN95，WIN98，WINNT4.0，或更高。
- RAM 大于 16MB
- 20MB 的硬盘空余空间。

## 安装详细说明

所有的 Keil 产品都带有一个安装程序，安装方便。8051 开发工具的安装步骤如下：

- 插入 Keil 开发工具光盘。
- 从 CD 浏览界面选择安装软件。
- 跟随提示进行安装操作。

---

*注意*
*当你插入 CD 时，你的计算机可能会自动浏览 CD。如果没有，运行* ***\KEIL\SETUP\SETUP.EXE*** *安装软件。*

---

## 文件夹组织结构

安装程序复制开发工具到基本目录的各个子目录中。默认的基本目录是 C:\KEIL。下表列出的文件夹结构是包括所有 8051 开发工具的全部安装信息。你的安装信息由你购买的开发套件决定。

| 文件夹 | 描述 |
| --- | --- |
| C:\KEIL\C51\ASM | 汇编 SFR 定义文件和模板源程序文件。 |
| C:\KEIL\C51\BIN | 8051 工具的执行文件。 |
| C:\KEIL\C51\EXAMPLES | 示例应用。 |
| C:\KEIL\C51\RTX51 | 完全实时操作系统文件。 |
| C:\KEIL\C51\RTX_TINY | 小型实时操作系统文件。 |
| C:\KEIL\C51\INC | C 编译器包含文件。 |
| C:\KEIL\C51\LIB | C 编译器库文件，启动代码和常规 I/O 资源。 |
| C:\KEIL\C51\MONITOR | 目标监控文件和用户硬件的监控配置。 |
| C:\KEIL\UV2 | 普通 uVision2 文件。 |

在本使用指南中，我们假定用户采用默认的文件夹结构。如果你安装你的软件到一个不同的文件夹，你必须调整路径名以适应你的安装。

# 第 3 章　开发工具

Keil 8051 开发工具提供数个十分有用的特性，可以帮助你快速地成功开发嵌入式应用。这些工具使用简单并保证你达到你的设计目的。

## uVision2 集成开发环境

uVision2 IDE 是一个基于 Window 的开发平台，包含一个高效的编辑器，一个项目管理器和一个 MAKE 工具。

uVision2 支持所有的 KEIL 8051 工具，包括 C 编译器，宏汇编器，连接/定位器，目标代码到 HEX 的转换器。uVision2 通过以下特性加速你的嵌入式系统的开发过程：

- 全功能的源代码编辑器。
- 器件库用来配置开发工具设置。
- 项目管理器用来创建和维护你的项目。
- 集成的 MAKE 工具可以汇编，编译和连接你的嵌入式应用。
- 所有开发工具的设置都是对话框形式的。
- 真正的源代码级的对 CPU 和外围器件的调试器。
- 高级 GDI（AGDI）接口用来在目标硬件上进行软件调试，以及和 Monitor-51 进行通信。
- 与开发工具手册和器件数据手册和用户指南有直接的链接。

---

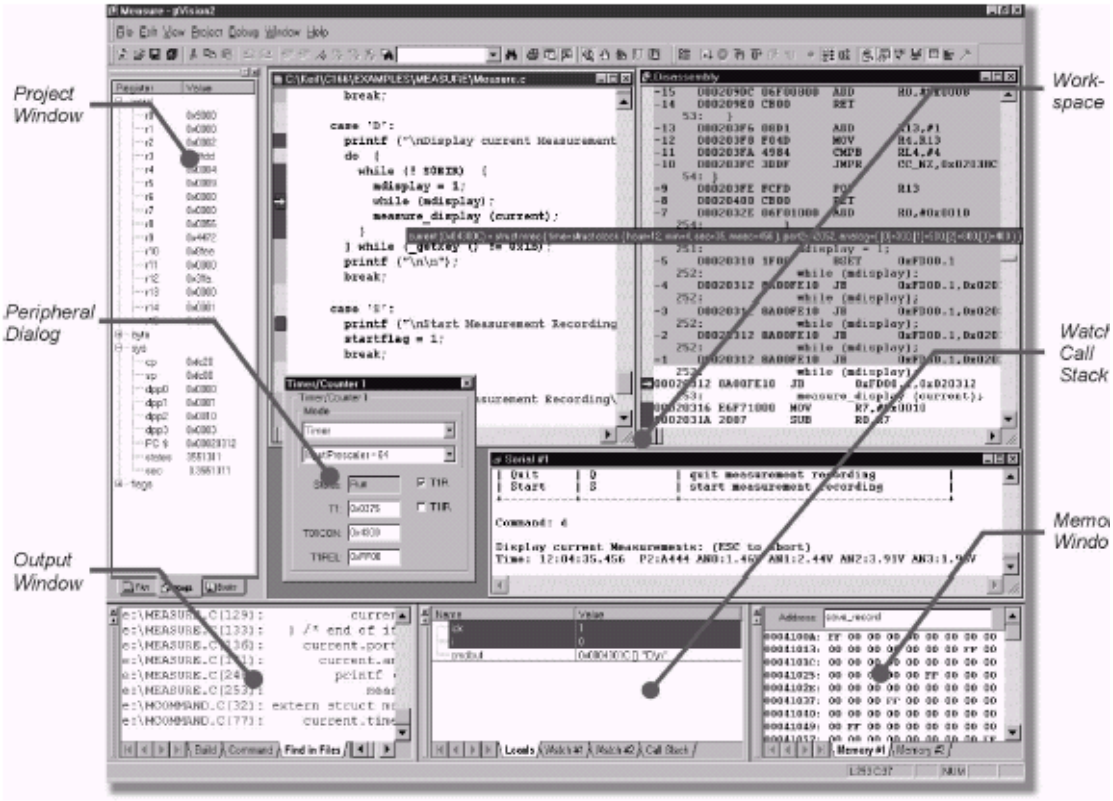
*注意*

*uVision2 调试器的特性只有* ***PK51*** *和* ***DK51*** *套件具备。*

---

## 关于开发环境

uVision2 界面提供一个菜单，一个工具条以便你快速选择命令按钮，另外还有源代码的显示窗口，对话框和信息显示。uVision2 允许同时打开浏览多个源文件。



## 菜单条，工具条和快捷键

菜单条提供各种操作菜单，如：编辑操作，项目维护，开发工具选项设置，调试程序，窗口选择和处理，在线帮助。工具条按钮允许你快速地执行 uVision2 命令。键盘快捷键（你自己可以配置）允许你执行 uVision2 命令。下面的表格列出了 uVision2 菜单项命令，工具条图标，默认的快捷键以及他们的描述：

### 文件菜单和命令（File）

| 菜单 | 工具条 | 快捷键 | 描述 |
|---|---|---|---|
| New |  | Ctrl+N | 创建新文件 |
| Open |  | Ctrl+O | 打开已经存在的文件 |
| Close | | | 关闭当前文件 |
| Save |  | Ctrl+S | 保存当前文件 |
| Save all |  | | 保存所有文件 |
| Save as… | | | 另外取名保存 |
| Device Database | | | 维护器件库 |
| Print Setup… | | | 设置打印机 |
| Print |  | Ctrl+P | 打印当前文件 |
| Print Preview | | | 打印预览 |
| 1-9 | | | 打开最近用过的文件 |
| Exit | | | 退出 uVision2，提示是否保存文件。 |

**编辑菜单和编辑器命令(Edit)**

| 菜单 | 工具条 | 快捷键 | 描述 |
|---|---|---|---|
| Home | | | 移动光标到本行的开始 |
| End | | | 移动光标到本行的末尾 |
| Ctrl+Home | | | 移动光标到文件的开始 |
| Ctrl+End | | | 移动光标到文件的结束 |
| Ctrl+<- | | | 移动光标到词的左边 |
| Ctrl+-> | | | 移动光标到词的右边 |
| Ctrl+A | | | 选择当前文件的所有文本内容 |
| Undo |  | Ctrl+Z | 取消上次操作 |
| Redo |  | Ctrl+Shift+Z | 重复上次操作 |
| Cut |  | Ctrl+X | 剪切所选文本 |
| | | Ctrl+Y | 剪切当前行的所有文本 |
| Copy |  | Ctrl+C | 复制所选文本 |
| Paste |  | Ctrl+V | 粘贴 |
| Indent Selected Text |  | | 将所选文本右移一个制表键的距离 |
| Unindent Selected Text |  | | 将所选文本左移一个制表键的距离 |
| Toggle Bookmark |  | Ctrl+F2 | 设置/取消当前行的标签 |
| Goto Next Bookmark |  | F2 | 移动光标到下一个标签处 |
| Goto Previous Bookmark |  | Shift+F2 | 移动光标到上一个标签处 |
| Clear All Bookmarks |  | | 清除当前文件的所有标签 |
| Find |  | | 在当前文件中查找文本 |
| | | F3 | 向前重复查找 |
| | | Shift+F3 | 向后重复查找 |
| | | Ctrl+F3 | 查找光标处的单词 |
| | | Ctrl+] | 寻找匹配的大括号，圆括号，方括号（用此命令将光标放到大括号，圆括号或方括号的前面） |
| Replace | | Ctrl+H | 替换特定的字符 |
| Find in Files… |  | | 在多个文件中查找 |

*/****************************译者注-开始******************************************/*

***Ctrl+]****命令在我的 uVision2 2.20a 中好象没有作用。另外，我的 uVision2 的* ***Edit*** *菜单中还有一个* ***Goto Matching brace*** *命令在最后，功能是选择匹配的一对大括号，圆括号或方括号中的内容。但是，在操作之前，你必须把光标置于其中一个括号的旁边，前或后都可以，但是要注意，必须紧靠！*

*/****************************译者注-结束******************************************/*

**选择文本命令**

在 uVision2 中，你可以通过按住 **Shift** 键和相应的光标操作键来选择文本。如，**Ctrl+->** 是移动光标到下一个词，那么，**Ctrl+Shift+->** 就是选择当前光标位置到下一个词的开始位置间的文本。

当然，你也可以用鼠标来选择文本，操作如下：

| 要选择... | 鼠标操作 |
|---|---|
| 任意数量的文本 | 在你要选择的文本上拖动鼠标 |
| 一个词 | 双击此词 |
| 一行文本 | 移动鼠标到此行的最左边，直到鼠标变成右指向的箭头，然后单击。 |
| 多行文本 | 移动鼠标到此行的最左边，直到鼠标变成右指向的箭头，然后相应拖动。 |
| 一个距形框中的文本 | 按住 **Alt** 键，然后相应拖动鼠标。 |

**视图菜单（View）**

| 菜单 | 工具条 | 快捷键 | 描述 |
|---|---|---|---|
| Status Bar | | | 显示/隐藏状态条 |
| File Toolbar | | | 显示/隐藏文件菜单条 |
| Build Toolbar | | | 显示/隐藏编译菜单条 |
| Debug Toolbar | | | 显示/隐藏调试菜单条 |
| Project Window |  | | 显示/隐藏项目窗口 |
| Output Window |  | | 显示/隐藏输出窗口 |
| Source Browser |  | | 打开资源浏览器 |
| Disassembly Window |  | | 显示/隐藏反汇编窗口 |
| Watch & Call Stack Window |  | | 显示/隐藏观察和堆栈窗口 |
| Memory Window |  | | 显示/隐藏存储器窗口 |
| Code Coverage Window |  | | 显示/隐藏代码报告窗口 |
| Performance Analyzer Window |  | | 显示/隐藏性能分析窗口 |
| Symbol Window | | | 显示/隐藏字符变量窗口 |
| Serial Window #1 |  | | 显示/隐藏串口 1 的观察窗口 |
| Serial Window #2 | | | 显示/隐藏串口 2 的观察窗口 |
| Toolbox |  | | 显示/隐藏自定义工具条 |
| Periodic Window Update | | | 程序运行时刷新调试窗口 |
| Workbook Mode | | | 显示/隐藏窗口框架模式 |
| Options… | | | 设置颜色，字体，快捷键和编辑器的选项 |

**项目菜单和项目命令（Project）**

| 菜单 | 工具条 | 快捷键 | 描述 |
|---|---|---|---|
| New Project … | | | 创建新项目 |
| Import μVision1 Project … | | | 转化 uVision1 的项目 |
| Open Project … | | | 打开一个已经存在的项目 |
| Close Project… | | | 关闭当前的项目 |
| Target Environment | | | 定义工具、包含文件和库的路径 |
| Targets, Groups,Files | | | 维护一个项目的对象、文件组和文件 |
| Select Device for Target | | | 选择对象的 CPU |
| Remove… | | | 从项目中移走一个组或文件. |
| Options… | | Alt+F7 | 设置对象、组或文件的工具选项 |
| File Extensions |   | | 选择不同文件类型的扩展名 |
| Build Target |  | F7 | 编译修改过的文件并生成应用 |
| Rebuild Target |  | | 重新编译所有的文件并生成应用 |
| Translate… |  | Ctrl+F7 | 编译当前文件 |
| Stop Build |  | | 停止生成应用的过程 |
| 1-9 | | | 打开最近打开过的项目 |

**调试菜单和调试命令（Debug）**

| 菜单 | 工具条 | 快捷键 | 描述 |
|---|---|---|---|
| Start/Stop Debugging |  | Ctrl+F5 | 开始/停止调试模式 |
| Go |  | F5 | 运行程序，直到遇到一个中断 |
| Step |  | F11 | 单步执行程序，遇到子程序则进入 |
| Step over |  | F10 | 单步执行程序，跳过子程序 |
| Step out of Current function |  | Ctrl+F11 | 执行到当前函数的结束 |
| Stop Running |  | ESC | 停止程序运行 |
| Breakpoints… | | | 打开断点对话框 |
| Insert/Remove Breakpoint |  | | 设置/取消当前行的断点 |
| Enable/Disable Breakpoint |  | | 使能/禁止当前行的断点 |
| Disable All Breakpoints |  | | 禁止所有的断点 |
| Kill All Breakpoints |  | | 取消所有的断点 |
| Show Next Statement |  | | 显示下一条指令 |
| Enable/Disable Trace Recording |  | | 使能/禁止程序运行轨迹的标识 |
| View Trace Records |  | | 显示程序运行过的指令 |
| Memory Map… | | | 打开存储器空间配置对话框 |
| Performance Analyzer… | | | 打开设置性能分析的窗口 |
| Inline Assembly… | | | 对某一个行重新汇编，可以修改汇编代码 |
| Function Editor… | | | 编辑调试函数和调试配置文件 |

**外围器件菜单（Peripherals）**

| 菜单 | 工具条 | 快捷键 | 描述 |
|---|---|---|---|
| Reset CPU |  | | 复位 CPU |
| Interrupt,<br>I/O-Ports,<br>Serial,<br>Timer,<br>A/D Converter,<br>D/A Converter,<br>I2C Controller,<br>CAN Controller,<br>Watchdog | | | 打开片上外围器件的设置对话框，对话框的种类及内容依赖于你选择的 CPU |

**工具菜单（Tool）**

利用工具菜单，你可以配置，运行 Gimpel PC-Lint，Siemens Easy-Case 和用户程序。通过 Customize Tools Menu…菜单，你可以添加你想要添加的程序。更详细的信息，请参考 72 页的“Using the Tools Menu”。

| 菜单 | 工具条 | 快捷键 | 描述 |
|---|---|---|---|
| Setup PC-Lint… | | | 配置 Gimpel Software 的 PC-Lint 程序 |
| Lint | | | 用 PC-Lint 处理当前编辑的文件 |
| Lint all C Source Files | | | 用 PC-Lint 处理你项目中所有的 C 源代码文件 |
| Setup Easy-Case… | | | 配置 Siemens 的 Easy-Case 程序 |
| Start/Stop Easy-Case | | | 运行/停止 Siemens 的 Easy-Case 程序 |
| Show File (Line) | | | 用 Easy-Case 处理当前编辑的文件 |
| Customize Tools Menu… | | | 添加用户程序到工具菜单中 |

**软件版本控制系统菜单（SVCS）**

用此菜单来配置和添加软件版本控制系统的命令。更详细的信息参见 76 页的“Using the SVCS Menu”。

| 菜单 | 工具条 | 快捷键 | 描述 |
|---|---|---|---|
| Configure | | | 配置软件版本控制系统的命令 |
| Version Control… | | | |

**视窗菜单（Window）**

| 菜单 | 工具条 | 快捷键 | 描述 |
|---|---|---|---|
| Cascade | | | 以互相重叠的形式排列文件窗口 |
| Tile Horizontally | | | 以不互相重叠的形式水平排列文件窗口 |
| Tile Vertically | | | 以不互相重叠的形式垂直排列文件窗口 |
| Arrange Icons | | | 排列主框架底部的图标 |
| Split | | | 把当前的文件窗口分割为几个 |
| 1-9 | | | 激活指定的窗口对象 |

**帮助菜单（Help）**

| 菜单 | 工具条 | 快捷键 | 描述 |
|---|---|---|---|
| Help topics | | | 打开在线帮助 |
| About μ Vision | | | 显示版本信息和许可证信息 |

uVision2 有两种操作模式

- 创建模式：让你编译应用中所有的文件，以产生执行程序。此模式的特性在 57 页“Creating Applications”中描述。
- 调试模式：提供一个非常强劲的调试器，你可以用它来调试你的程序。此模式的特性在 93 页“Testing Programs”中描述。

在两种模式下，你都可以用源文件编辑器来编辑你的源代码。

## C51 优化的 C 语言交叉编译器

Keil C51 交叉编译器是一个基于 ANSI C 标准的针对 8051 系列 MCU 的 C 编译器，生成的可执行代码快速、紧凑，在运行效率和速度上可以和汇编程序得到的代码相媲美。

- 和汇编语言相比，用 C 语言这样的高级语言有很多优势，比如：
- 对处理器的指令集不必了解，8051 CPU 的基本结构可以了解，但不是必须的。
- 寄存器的分配以及各种变量和数据的寻址都由编译器完成。
- 程序拥有了正式的结构（由 C 语言带来的），并且能被分成多个单独的子函数。这使整个应用系统的结构变得清晰，同时让源代码变得可重复使用。
- 选择特定的操作符来操作变量的能力提高了源代码的可读性。
- 可以运用和人的思维很接近的词汇和算法表达式。
- 编写程序和调试程序的时间得到很大程度的缩短。
- C 运行连接库包含一些标准的子程序，如：格式化输出，数字转换，浮点运算。
- 由于程序的模块结构技术，使得现有的程序段可以很容易的包含到新的程序中去。
- ANSI 标准的 C 语言是一种丰常方便的，获得广泛应用的，在绝大部分系统中都能够很容易得到的语言。

因此，如果需要，现有的程序可以很快地移植到其他的处理器上，节省投资。

## C51 语言的扩展

虽然 C51 是一个兼容 ANSI 的编译器，但为了支持 8051 系列 MCU 还是加入了一些扩展的内容。C51 编译器的扩展内容包括：

- 数据类型
- 存储器类型
- 指针
- 重入函数
- 中断服务程序
- 实时操作系统
- 和 PL/M 及 A51 源程序的接口

以下各节简单地描述了上述的扩展特性。

## 数据类型

本 C51 编译器支持下表列出的各种规格的数据类型。.除了这些数据类型以外，变量可以组合成结构，联合及数组。除非特别说明，这些变量都可以用指针存取。

| Data Type | Bits | Bytes | Value Range |
|---|---|---|---|
| **bit †** | 1 | | 0 to 1 |
| **signed char** | 8 | 1 | -128 to +127 |
| **unsigned char** | 8 | 1 | 0 to 255 |
| **enum** | 16 | 2 | -32768 to +32767 |
| **signed short** | 16 | 2 | -32768 to +32767 |
| **unsigned short** | 16 | 2 | 0 to 65535 |
| **signed int** | 16 | 2 | -32768 to +32767 |
| **unsigned int** | 16 | 2 | 0 to 65535 |
| **signed long** | 32 | 4 | -2147483648 to 2147483647 |
| **unsigned long** | 32 | 4 | 0 to 4294967295 |
| **float** | 32 | 4 | ±1.175494E-38 to ±3.402823E+38 |
| **sbit †** | 1 | | 0 to 1 |
| **sfr †** | 8 | 1 | 0 to 255 |
| **sfr16 †** | 16 | 2 | 0 to 65535 |

注：* bit, sbit， sfr,和 sfr16 为 8051 硬件和 C51 及 C251 编译器所特有，它们不是 ANSI C 的一部分，也不能用指针对它们进行存取。

这些 **sbit， sfr** 和 **sfr16** 类型的数据使你能够操作 8051MCU 所提供的特殊功能寄存器。例如，下面的表达式：

```
sfr P0 = 0x80; /* Define 8051 P0 SFR */
```

声明了一个变量 **P0**，并且把它和位于 0x80（8051 的端口 0）处的特殊功能寄存器联系在一起。

当结果的数据类型和源数据类型不同时，C51 编译器在数据类型间自动进行转换。例如，一个 bit 变量赋值给一个 interger 变量时将会被转换为 integer。当然，你可以用类型表示进行强制转换。数据转换时要注意，有符号变量的转换，其符号是自动扩展的。

## 存储器类型

本 C51 编译器支持 8051 及其派生类型的结构，能够访问 8051 的所有存储器空间。具有下表列出的存储器类型的变量都可以被分配到某个特定的存储器空间。

| 存储器类型 | 描述 |
|---|---|
| code | 程序空间（64 Kbytes）；通过 MOVC @A+DPTR 访问。 |
| data | 直接访问的内部数据存储器；访问速度最快（128 bytes）。 |
| idata | 间接访问的内部数据存储器；可以访问所有的内部存储器空间（256 bytes）。 |
| bdata | 可位寻址的内部数据存储器；可以字节方式也可以位方式访问（16 bytes）。 |
| xdata | 外部数据存储器（64 Kbytes）；通过 MOVX @DPTR 访问。 |
| pdata | 分页的外部数据存储器（256 bytes）；通过 MOVX @Rn 访问。 |

访问内部数据存储器将比访问外部数据存储器快的多。由于这个原因，你应该把频繁使用的变量放置在内部数据存储器中，把很少使用的变量放在外部数据存储器中。这通过使用 SMALL 模式将很容易就做到。通过定义变量时包括存储器类型，你可以定义此变量存储在你想要的存储器中。

在变量的声明中，你可以包括存储器类型和 **signed** 或 **unsigned** 属性。

```
char data var1;
char code text[] = "ENTER PARAMETER";
unsigned long xdata array[100];
float idata x,y,z;
unsigned int pdata dimension;
unsigned char xdata vector[10][4][4];
char bdata flags;
```

如果在变量的定义中，没有包括存储器类型，将自动选用默认或暗示的存储器类型。暗示的存储器类型适用于所有的全局变量和静态变量，还有不能分配在寄存器中的函数参数和局部变量。默认的存储器类型由编译器的参数 SMALL，COMPACT 及 LARGE 决定。这些参数定义了编译时使用的存储模式。

## 存储模式

存储模式决定了默认的存储器类型，此存储器类型将应用于函数参数，局部变量和定义时未包含存储器类型的变量。你可以在命令行用 SMALL，COMPACT 和 LARGE 参数定义存储模式。定义变量时，使用存储器类型显式定义将屏蔽默认存储器类型。

### 小(SMALL)模式

所有变量都默认在 8051 的内部数据存储器中。这和用 data 显式定义变量起到相同的作用。在此模式下，变量访问是非常快速的。然而，所有数据对象，包括堆栈都必须放在内部 RAM 中。堆栈空间面临溢出，因为堆栈所占用多少空间依赖于各个子程序的调用嵌套深度。在典型应用中，如果具有代码分段功能的 BL51 连接/定位器被配置成覆盖内部数据存储器中的变量时，此 SMALL 模式是最好的选择。

### 紧凑（COMPACT）模式

此模式中，所有变量都默认在 8051 的外部数据存储器的一页中。地址的高字节往往通过 Port 2 输出。其值必须由你在启动代码中设置，编译器不会为你设置。这和用 pdata 显式定义变量起到相同的作用。此模式最多只能提供 256 字节的变量。这种限制来自于间接寻址所使用的 R0,R1（MOVX @R0/R1）。这种模式不如 SMALL 模式高效，所以变量的访问不够快。不过它比 LARGE 模式要快。

### 大（LARGE）模式

在大模式下，所有的变量都默认在外部存储器中（xdata）。这和用 xdata 显式定义变量起到相同的作用。数据指针（DPTR）用来寻址。通过 DPTR 进行存储器的访问的效率很低，特别是在对一个大于一个字节的变量进行操作时尤为明显。此数据访问类型比 SMALL 和 COMPACT 模式需要更多的代码。

---

*注意*

*你或许应该一直使用小模式，它产生最快，最紧凑，效率最高的代码。*

*你最好显式定义你的变量的存储器类型。只有当你的应用不能在* ***SMALL*** *模式下操作时，你才需要往上增加你的存储模式。*

---

## 指针

C51 编译器支持用星号（*）进行指针声明。你可以用指针完成在标准 C 语言中有的所有操作。

另外，由于 8051 及其派生系列所具有的独特结构，C51 编译器支持两种不同类型的指针：存储器指针和通用指针。

### 通用指针

通用或未定型的指针的声明和标准 C 语言中一样。如：

```
char *s; /* string ptr */
int *numptr; /* int ptr */
long *state; /* long ptr */
```

通用指针总是需要三个字节来存储。第一个字节是用来表示存储器类型，第二个字节是指针的高字节，第三字节是指针的低字节。

通用指针可以用来访问所有类型的变量，而不管变量存储在哪个存储空间中。因而许多库函数都使用通用指针。通过使用通用指针，一个函数可以访问数据，而不用考虑它存储在什么存储器中。

通用指针很方便，但是也很慢。在所指向目标的存储空间不明确的情况下，它们用的最多。

### 存储器指针

存储器指针或类型确定的指针在定义时包括一个存储器类型说明，并且总是指向此说明的特定存储器空间。例如：

```
char data *str; /* ptr to string in data */
int xdata *numtab; /* ptr to int(s) in xdata */
long code *powtab; /* ptr to long(s) in code */
```

正是由于存储器类型在编译时已经确定，通用指针中用来表示存储器类型的字节就不再需要了。

指向 idata，data，bdata 和 pdata 的存储器指针用一个字节保存，指向 code 和 xdata 的存储器指针用两个字节保存。使用存储器指针比通用指针效率要高，速度要快。当然，存储器指针的使用不是很方便。在所指向目标的存储空间明确并不会变化的情况下，它们用的最多。

### 存储器指针和通用指针的比较

使用存储器指针可以显著的提高 8051 C 程序的运行速度。

下面的示例程序说明了使用不同的指针在代码长度，占用数据空间和运行时间上的不同。

| Description | Idata Pointer | Xdata Pointer | Generic Pointer |
|---|---|---|---|
| C 源程序 | idata *ip;<br>char val;<br>val = *ip; | char xdata *xp;<br>har val;<br>val = *xp; | char *p;<br>char val;<br>val = *xp; |
| 编译后的代码 | MOV R0,ip<br>MOV val,@R0 | MOV DPL,xp +1<br>MOV DPH,xp<br>MOV A,@DPTR<br>MOV val,A | MOV R1,p + 2<br>MOV R2,p + 1<br>MOV R3,p<br>CALL CLDPTR |
| 指针大小 | 1 byte | 2 byte | 3 byte |
| 代码长度 | 4 bytes | 9 bytes | 11 bytes + library call |
| 执行时间 | 4 cycles | 7 cycles | 13 cycles |

## 重入函数

多个进程可以同时使用一个重入函数。当一个重入函数被调用运行时，另外的一个进程可能中断此运行过程，然后再次调用此重入函数。通常情况下，C51 函数不能被递归调用，也不能应用导致递归调用的结构。有此限制是由于函数参数和局部变量是存储在固定的地址单元中。重入函数特性允许你声明一个重入函数。即可以被递归调用的函数。如：

```
int calc (char i, int b) reentrant
{
int x;
x = table [i];
return (x * b);
}
```

重入函数可以被递归调用，也可以同时被两个或更多的进程调用。重入函数在实时应用中及中断服务程序代码和非中断程序代码必须共用一个函数的场合中经常用到。

对每一个重入函数来说，根据存储模式，重入堆栈被安置在内部或外部单元中。

---

*注意*

*在一个基本函数的基础上添加* ***reentrant*** *说明，从而使它具有重入特性。*

*需要注意的是,你可以选择哪些必须的函数为重入函数，而不需将全部程序声明为重入函数。*

*把全部程序声明为重入函数将增加目标代码的长度并减慢运行速度。*

---

## 中断服务程序

C51 编译器允许你用 C 语言创建中断服务程序。你仅仅需要关心中断号和寄存器组的选择。编译器自动产生中断向量和程序的入栈及出栈代码。在函数声明时包括 **interrupt**，将把所声明的函数定义为一个中断服务程序。另外，你可以用 **using** 定义此中断服务程序所使用的寄存器组。

```
unsigned int interruptcnt;
unsigned char second;
void timer0 (void) interrupt 1 using 2
{
    if (++interruptcnt == 4000)
    {   /* count to 4000 */
        second++; /* second counter */
        interruptcnt = 0; /* clear int counter */
    }
}
```

## 参数传递

C51 编译器能在 CPU 寄存器中传递最多三个参数，由于不用从存储器中读出和写入参数，从而显著提高了系统性能。参数传递由 REGPARMS 和 NOREGPARMS 编译参数所控制。下表列出了不同的参数和数据类型所占用的寄存器：

| 参数数目 | char, 1-byte pointer | int, 2-byte pointer | long, float | generic pointer |
|---|---|---|---|---|
| 1 | R7 | R6 & R7 | R4 －R7 | R1 — R3 |
| 2 | R5 | R4 & R5 | | |
| 3 | R3 | R2 & R3 | | |

如果没有 CPU 寄存器供参数传递所用，或太多的参数需要传递时，地址固定的存储器将用来存储这些额外的参数。

## 函数返回值

函数返回值总是通过 CPU 寄存器进行。下表列出了返回各种数据时所用的 CPU 寄存器：

| 返回数据类型 | 寄存器 | 描述 |
|---|---|---|
| bit | Carry Flag | |
| char,unsigned char,1-byte pointer | R7 | |
| int, unsigned int, 2-byte pointer | R6 & R7 | MSB in R6, LSB in R7 |
| long, unsigned long | R4 — R7 | MSB in R4, LSB in R7 |
| float | R4 — R7 | 32-Bit IEEE format |
| generic pointer | R1 — R3 | Memory type in R3, MSB R2, LSB R1 |

## 寄存器优化

根据程序前后的联系，C51 编译器分配最多 7 个寄存器来存储寄存器变量。C51 编译器能分析每个程序模块中对寄存器的修改。连接程序产生一个全局的、项目级的寄存器文件，此文件包含被外部程序改变的所有寄存器的信息。因而，C51 编译器知道整个应用中每个函数所使用的寄存器，并能为每个 C 函数优化分配 CPU 寄存器。

## 对实时操作系统的支持

C51 编译器很好地集成了 RTX-51 Full 和 RTX-51 Tiny 多任务实时操作系统。任务描述表在连接过程中控制和产生。关于 RTX 实时操作系统的详细信息请参考 169 页开始的"RTX-51 Real-Time Operating System"一章。

## 和汇编语言的接口

你可以很容易在 C 程序中调用汇编程序，反之依然。函数参数通过 CPU 寄存器传递，或使用 NOREGPARMS 参数指示编译器通过固定的存储器传递。从函数返回的值总是通过 CPU 寄存器传递。除了直接产生目标代码外，你还可以用 SRC 编译参数指示编译器产生汇编源代码文件（供 A51汇编器使用）。例如下面的 C 语言源代码：

```
unsigned int asmfunc1 (unsigned int arg)
{
    return (1 + arg);
}
```

用 SRC 指示 C51 编译器编译时产生以下汇编文件：

```
?PR?_asmfunc1?ASM1  SEGMENT CODE
PUBLIC                  asmfunc1
        RSEG    ?PR?_asmfunc1?ASM1
        USING 0
 asmfunc1:
;---- Variable 'arg?00' assigned to Register 'R6/R7' ----
        MOV A,R7        ; load LSB of the int
        ADD A,#01H              ; add 1
        MOV R7,A        ; put it back into R7
        CLR A
        ADDC A,R6       ; add carry & R6
        MOV R6,A
?C0001:
        RET             ; return result in R6/R7
```

你可以用**#pragma asm** 和 **#pragma endasm** 预处理指示器来在你的 C 语言程序中插入汇编指令。

## 和 PL/M-51 的接口

Intel 的 PL/M-51 是一种流行的编程语言，在很多方面和 C 语言类似。你很容易就可以将 C 程序和 PL/M-51 程序联接起来。

在你用 alien 声明 PL/M-51 函数后，你就可以从 C 语言中调用它们。所有在 PL/M-51 模块中定义的全局变量都可以在 C 语言程序中使用。例如：

```
extern alien char plm_func (int, char);
```

PL/M-51 编译器和 Keil Software 工具都产生 OMF51 格式的目标文件。连接程序使用 OMF51 文件来处理外部字符变量，而不管它们在什么地方声明和使用。

## 代码优化

C51 是一个杰出的优化编译器，它通过很多步骤以确保产生的代码是最有效率的（最小和/或最快）。编译器通过分析初步的代码 产生最终的最有效率的代码序列，以此来保证你的 C 语言程序占用最少空间的同时运行的快而有效。

C51 编译器提供 9 个优化级别。每个高一级的优化级别都包括比它低的所有优化级别的优化内容。以下列出的是目前 C51 编译器提供的所有优化级别的内容：

- 常量折叠：在表达式及寻址过程中出现的常量被综合为一个单个的常量。
- 跳转优化：采用反转跳转或直接指向最终目的的跳转，从而提升了程序的效率。
- 哑码消除：永远不可能执行到的代码将自动从程序中剔除。
- 寄存器变量：只要可能，局部变量和函数参数被放在 CPU 寄存器中，不需要为这些变量再分配存储器空间。

- 通过寄存器传递参数：最多三个参数通过寄存器传递。
- 消除全局公用的子表达式：只要可能，程序中多次出现的相同的子表达式或地址计算表达式将只计算一次。
- 合并相同代码：利用跳转指令，相同的代码块被合并。
- 重复使用入口代码：需要多次使用的共同代码被移到子程序的前面以缩减代码长度。
- 公共块子程序：需要重复使用的多条指令被提取组成子程序。指令被重新安排以最大化一个共用子程序的长度。

/**********译者注-开始**********/
*对于操作硬件有时序要求的应用,慎用第 9 级优化.*
*因为它将可能调整你的指令顺序.*
/**********译者注-结束**********/

## 对 8051 的特殊优化

- 窥孔优化：当能够缩小代码空间或执行时间时，复杂的操作被简单的操作代替。
- 访问优化：常量和变量被计算后直接包含在操作中。
- 扩展访问优化：用 DPTR 做存储器指针来增加代码的密度。
- 数据覆盖：一个函数的数据和位变量空间是可覆盖的，BL51 连接器将采用覆盖技术来分配变量空间。
- Case/Switch 优化：根据使用的数字，序列和位置，用跳转表或一连串的跳转指令来优化 switch 及 case 结构。

## 代码生成选项

- 空间优化：公共 C 操作被子程序代替。以程序执行速度的降低来换取程序代码空间的缩减。
- 时间优化：公共 C 操作被嵌入到程序中。以程序代码空间的增加来换取程序执行速度的提高。
- 不用绝对寄存器：不用绝对寄存器地址访问。程序代码依赖于寄存器的分段。
- 不用寄存器传递参数：用局部数据段来传递参数，而不用寄存器。这是为了兼容早期版本的 C51 编译器，PL/M-51 编译器和 ASM-51 汇编器。

## 调试

C51 编译器使用 Intel 目标格式（OMF51）来产生目标文件和全部的字符变量信息。而且，编译器还包含所有必要的信息，如：变量名，函数名，行数以及 uVision2 调试器或任何兼容 Intel 格式的仿真器用来逐条、彻底地调试和分析程序所需要的信息。

另外，使用 OBJECTEXTEND 参数可以指示编译器产生附加的变量类型信息到目标文件中。这样，利用相应的仿真器就可以显示变量和结构的数据信息。你可以向你的仿真器供应商询问是否支持 Intel OMF51 格式和 Keil 软件生成的目标模块。

## 库函数

C51 编译器包含有 ANSI 标准的 7 个不同的的编译库，从而满足不同功能的需要。

| 库文件 | 描述 |
|---|---|
| C51S.LIB | 小模式库，不支持浮点运算 |
| C51FPS.LIB | 小模式库，支持浮点运算 |
| C51C.LIB ` | 紧凑模式库，不支持浮点运算 |
| C51FPC.LIB | 紧凑模式库，支持浮点运算 |
| C51L.LIB | 大模式库，不支持浮点运算 |
| C51FPL.LIB | 大模式库，支持浮点运算 |
| 80C751.LIB | Philips 8xC751 及其派生系列使用的库 |

和硬件相联系的输入/输出操作的库函数模块的源代码文件位于\KEIL\C51\LIB 文件夹中。你可以利用这些文件来修改你的库以适应你目标板上的任何器件的输入/输出操作。

## 内连的库函数

本编译器的库中包含一定数量的函数是内连函数。内连函数不产生 ACALL 或 LCALL 指令来执行库函数。以下的内连函数产生内连的代码，因而它比一个调用函数要快而有效。

| 内连函数 | 描述 |
|---|---|
| _crol_ | 字节左移 |
| _cror_ | 字节右移 |
| _irol_ | 整数左移 |
| _iror_ | 整数右移 |
| _lrol_ | 长整数左移 |
| _lror_ | 长整数右移 |
| _nop_ | 空操作 |
| _testbit_ | 判断并清除（8051 JBC 指令） |

## 编译器的调用

通常情况下，当你创建你的项目时，C51 编译器由 uVision2 IDE 调用。当然，你也可以在 DOS 方式，在命令行键入 C51 来运行。你的 C 源程序文件名必须和编译控制参数一起在命令行输入。如：

```
>C51 MODULE.C COMPACT PRINT (E:M.LST) DEBUG SYMBOLS
C51 COMPILER V6.00
C51 COMPILATION COMPLETE. 0 WARNING(S), 0 ERROR(S)
```

编译器控制参数可以在命令行输入，也可以在文件中的开头用#pragma 定义。要想知道全部的编译器控制参数，请参考 213 页 的"C51/C251 Compiler"

## 示例程序

下面的程序显示了 C51 编译器的一些功能特性。本 C51 编译器根据编译控制参数产生相应的 OMF51 格式的目标文件。

编译器报告所有必要的信息，如：变量名，函数名，行数以及 uVision2 调试器或其它仿真器用来详细调试和分析程序所需要的信息。

编译后，C51 编译器产生一个列表文件。文件中包含：源代码，指示信息，汇编清单和字符表。

下页中展示了一个 C51 编译器生成的示例列表文件。

```
C51 COMPILER V6.00,  SAMPLE            01/01/2001  08:00:00  PAGE 1

DOS C51 COMPILER V6.00, COMPILATION OF MODULE SAMPLE
OBJECT MODULE PLACED IN SAMPLE.OBJ
COMPILER INVOKED BY: C:\KEIL\C51\BIN\C51.EXE SAMPLE.C CODE

stmt level source
   1       #include <reg51.h>   /* SFR definitions for 8051 */
   2       #include <stdio.h>   /* standard i/o definitions */
   3       #include <ctype.h>   /* defs for char conversion */
   4
   5       #define EOT  0x1A    /* Control+Z signals EOT */
   6
   7       void main (void)  {
   8   1      unsigned char c;
   9   1
  10   1      /* setup serial port hdw (2400 Baud @12 MHz) */
  11   1      SCON = 0x52;      /* SCON */
  12   1      TMOD = 0x20;      /* TMOD */
  13   1      TCON = 0x69;      /* TCON */
  14   1      TH1  = 0xF3;      /* TH1  */
  15   1
  16   1      while ((c = getchar ()) != EOF)  {
  17   2         putchar (toupper (c));
  18   2         }
  19   1      P0 = 0;   /* clear Output Port to signal ready */
  20   1      }

ASSEMBLY LISTING OF GENERATED OBJECT CODE

             ; FUNCTION main (BEGIN)
                                           ; SOURCE LINE # 7
                                           ; SOURCE LINE # 11
0000 759852        MOV     SCON,#052H
                                           ; SOURCE LINE # 12
0003 758920        MOV     TMOD,#020H
                                           ; SOURCE LINE # 13
0006 758869        MOV     TCON,#069H
                                           ; SOURCE LINE # 14
0009 758DF3        MOV     TH1,#0F3H
000C           ?C0001:
                                           ; SOURCE LINE # 16
000C 120000  E     LCALL   getchar
000F 8F00    R     MOV     c,R7
0011 EF            MOV     A,R7
0012 F4            CPL     A
0013 6008          JZ      ?C0002
                                           ; SOURCE LINE # 17
0015 120000  E     LCALL   _toupper
0018 120000  E     LCALL   _putchar
                                           ; SOURCE LINE # 18
001B 80EF          SJMP    ?C0001
001D           ?C0002:
                                           ; SOURCE LINE # 19
001D E4            CLR     A
001E F580          MOV     P0,A
                                           ; SOURCE LINE # 20
0020 22            RET
             ; FUNCTION main (END)


MODULE INFORMATION:   STATIC OVERLAYABLE
   CODE SIZE        =     33    ----
   CONSTANT SIZE    =   ----    ----
   XDATA SIZE       =   ----    ----
   PDATA SIZE       =   ----    ----
   DATA SIZE        =   ----       1
   IDATA SIZE       =   ----    ----
   BIT SIZE         =   ----    ----
END OF MODULE INFORMATION.


C51 COMPILATION COMPLETE.  0 WARNING(S),  0 ERROR(S)
```

*The C51 Compiler produces a listing file with line numbers and the time and date of compilation.*

*Information about compiler invocation and the object file generated is printed.*

*The listing contains a line number before each source line and the instruction nesting ( ) level.*

*If errors or possible sources of errors exist an error or warning message is displayed.*

*Enable under µVision2 Options for Target – Listing - Assembly Code the C51 CODE directive. This gives you an assembly listing file with embedded source line numbers.*

*A memory overview provides information about the occupied 8051 memory areas.*

*The number of errors and warnings in the program are included at the end of the listing.*

- C51 编译器产生行号，编译时的时间和日期。
- 编译器的运行和产生的目标文件的信息被记录在案。
- 列表文件在每个源代码前面包含行号和{}的嵌套层数。
- 如果错误或可能错误的代码存在，一个错误或告警信息将显示出来。
- 选择在 µ Vision2-Options for Target – Listing 中的 Assembly Code 代码指示选项，将在列表文件的汇编代码处加入原代码所在的行号。
- 存储器一览表提供了 8051 存储器占用信息
- 程序中的错误和告警总数包括在文件的结尾处.

# A51 宏汇编器

本 A51 是一个 8051MCU 系列的宏汇编器。它把汇编语言翻译成机器代码。本 A51 汇编器允许你定义你程序中的每一个指令，在需要极快的运行速度，很小的代码空间，精确的硬件控制时使用。本汇编器的宏特性让公共代码只需要开发一次，从而节约了开发和维护的时间。

## 源码级调试

本 A51 汇编器在生成的目标文件中包含全部的行号，字符及其类型的信息。这让你的调试器能够精确显示程序变量。行号是为了 uVision2 调试器或第三方仿真器源代码级调试你汇编过的程序时使用的。

## 功能一览

本 A51 汇编器翻译汇编源程序为可重定位的目标代码。它产生一个列表文件，其中可以包含或不包含字符表及交叉参考信息。

本 A51 汇编器支持两种宏处理：

### 标准的宏处理

这是一个比较容易使用的宏处理，它允许你在你的 8051 汇编代码中定义和使用宏。它标准的宏语法和其它许多汇编器中使用的相同。

### 宏处理语言(MPL)

是一个和 Intel ASM51 宏处理兼容的字符串替换工具。MPL 有几个预先定义好的宏处理功能来执行一些有用的操作，如：字符串处理或数字处理。

本 A51 汇编器宏处理的另一个有用的特性是根据命令行参数或汇编符号进行条件汇编。代码段的条件汇编能帮助你实现最紧凑的代码。

它也让你可以从一个汇编源代码文件产生不同的应用。

## 列表文件

下面是汇编器产生的列表文件的例子：

```
A51 MACRO ASSEMBLER  Test Program        07/01/99 08:00:00 PAGE     1

DOS MACRO ASSEMBLER A51 V6.00
OBJECT MODULE PLACED IN SAMPLE.OBJ
ASSEMBLER INVOKED BY: C:\KEIL\C51\BIN\A51.EXE SAMPLE.A51 XREF

LOC  OBJ       LINE  SOURCE
                  1  $TITLE ('Test Program')
                  2  NAME    SAMPLE
                  3
                  4  EXTRN CODE (PUT_CRLF, PUTSTRING, InitSerial)
                  5  PUBLIC  TXTBIT
                  6
                  7  PROG    SEGMENT         CODE
                  8  CONST   SEGMENT         CODE
                  9  BITVAR  SEGMENT         BIT
                 10
----              11          CSEG  AT     0
                 12
0000 020000   F  13  Reset:  JMP   Start
                 14
----             15          RSEG  PROG
                 16   ; *****
0000 120000   F  17  Start:  CALL  InitSerial ;Init Serial Interface
                 18
                 19  ; This is the main program. It is an endless
                 20  ; loop which displays a text on the console.
0003 C200     F  21          CLR   TXTBIT       ; read from CODE
0005 900000   F  22  Repeat: MOV   DPTR,#TXT
0008 120000   F  23          CALL  PUTSTRING
000B 120000   F  24          CALL  PUT_CRLF
000E 80F5     F  25          SJMP  Repeat
                 26  ;
----             27          RSEG  CONST
0000 54455354    28  TXT:    DB    'TEST PROGRAM',00H
0004 2050524F
0008 4752414D
000C 00
                 29
                 30
                 31
----             32          RSEG  BITVAR  ; TXTBIT=0 read from CODE
0000             33  TXTBIT: DBIT  1       ; TXTBIT=1 read from IDATA
                 34
                 35          END

IREP SYMBOL TABLE LISTING
---- ------ ----- -------

N A M E             T Y P E  V A L U E   ATTRIBUTES / REFERENCES

BITVAR . . . . . .  B SEG    0001H       REL=UNIT   9# 32
CONST. . . . . .  C SEG    000DH       REL=UNIT   8# 27
INITSERIAL . . . .  C ADDR   -----       EXT   4# 17
PROG . . . . . .  C SEG    0010H       REL=UNIT   7# 15
PUTSTRING. . . . .  C ADDR   -----       EXT   4# 23
PUT_CRLF . . . . .  C ADDR   -----       EXT   4# 24
REPEAT . . . . . .  C ADDR   0005H   R   SEG=PROG   22# 25
RESET. . . . . .  C ADDR   0000H   A      13#
SAMPLE . . . . . .  N NUMB   -----         2
START. . . . . .  C ADDR   0000H   R   SEG=PROG   13 17#
TXT. . . . . .  C ADDR   0000H   R   SEG=CONST   22 28#
TXTBIT . . . . . .  B ADDR   0000H.0 R   SEG=BITVAR   5 5 21 33#

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE.  0 WARNING(S), 0 ERROR(S)
```

*A51 produces a listing file with line numbers as well as the time and date of the translation. Information about assembler invocation and the object file generated is printed.*

*Typical programs start with EXTERN, PUBLIC, and SEGMENT directives.*

*The listing contains a source line number and the object code generated by each source line.*

*Error messages and warning messages are included in the listing file. The position of each error is clearly marked.*

*Enable under µVision2 Options for Target – Listing – Cross Reference to get a detailed listing of all symbols used in the assembler source file.*

*The register banks used, and the total number of warnings and errors are at the end of the listing file.*

- A51 汇编器产生一个列表文件，包括行号，汇编时的时间和日期。关于汇编器运行和目标文件产生的信息被记录下来。
- 通常情况下，程序从 EXTERN，PUBLIC， and SEGMENT 指示器开始。列表文件包含了每个源代码的行号及每行产生的代码。
- 列表文件包含了错误和告警信息，错误和告警的位置被明显的标识出来。
- 选择在 µ Vision2-Options for Target － Listing 中的 Cross Reference 选项，将在列表文件列出源程序中所用到的所有字符。
- 存储器组的占用信息和程序中的错误和告警总数包括在文件的结尾处.

# BL51 具有代码分段功能的连接/重定位器

本 BL51 是具有代码分段功能的连接/重定位器，它组合一个或多个目标模块成一个 8051 的执行程序。此连接器处理外部和全局数据，并将可重定位的段分配到固定的地址上。

本 BL51 连接器处理由 Keil C51 编译器，A51 汇编器和 Intel PL/M-51 编译器，ASM-51 汇编器产生的目标模块。连接器自动选择适当的运行库并连接那些用到的模块。

你也可以在命令行上输入相应的目标模块的名字的组合来运行本连接器。BL51 连接器默认的控制参数是经过仔细选择的，因而不需要定义附加的控制参数就可以适应大多数应用。当然，你可以很容易为你的应用作特定的设置。

## 数据地址管理

本连接器通过覆盖那些不会同时使用的函数变量的技术来管理 8051 有限的内部存储器资源，这极大地降低了大多数应用对存储器的需求。本 BL51 连接器分析函数间的引用以决定存储的覆盖策略。你可以用 OVERLAY 指示器来人为控制函数间的引用，这些引用被连接器用来确定哪些存储器单元是独占的。NOOVERLAY 指示器让 BL51 不进行覆盖连接，这在使用间接调用的函数时或为了调试而禁止覆盖时比较有用.

## 代码分段

本 BL51 连接器支持创建程序空间大于 64KB 的应用。既然 8051 不能直接操作大于 64KB 的代码地址空间，就必须由外部硬件来交换代码段。完成此功能的硬件必须要在 8051 中运行的程序的控制中。这就是大家所知的段(块)切换。

本 BL51 连接器让你管理一个公共的区域和 32 个最大 64KB 空间的块，从而达到总共 2MB 的分段程序空间。支持外部硬件块切换的软件包括的一个汇编程序可以由你来编辑，以适应你应用中的特定硬件平台。

本 BL51 连接器让你定义哪个段装载哪个特定的程序模块。通过仔细考虑，把各个函数分配到不同的段中来创建一个非常大而有效的应用。

### 公共段

段切换程序中的公共段是一块在任何时候，在所有的段中都可以访问的存储器。此公共段在物理上就不能切换出局或变换地址空间。

在公共段中的代码可以复制到每个段中（如果切换整个程序空间）或驻留在一个独立的地址空间或器件中（公共段不用切换）。

公共段包含那些必须在所有时候都要用到的程序段和常量。它还可以包括经常使用的代码。默认情况下，以下的代码内容将自动分配在公共段中：

- 复位和中断向量
- 代码常量
- C51 中断服务进程
- 分段开关跳转表
- 一些 C51 实时运行库函数

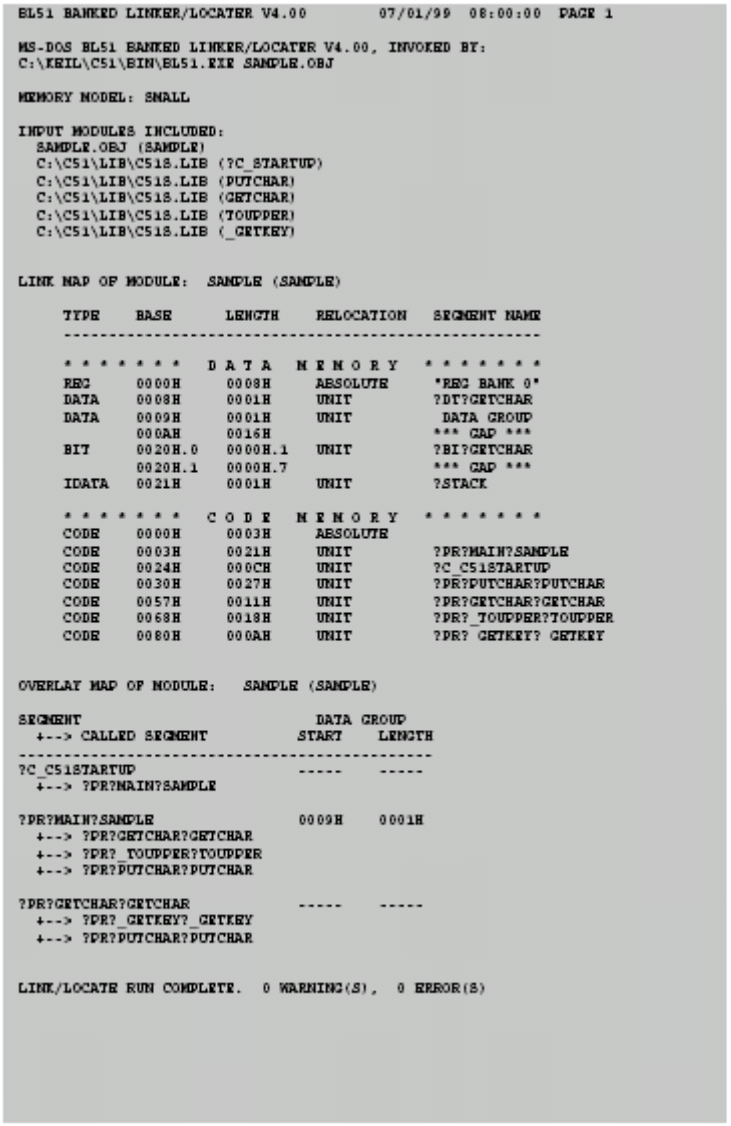
### 执行其它段中的程序

分段代码空间是通过附加的由软件控制的地址线控制的，这些地址线可以是由 8051 的 I/O 口或位于存储器空间的锁存器来模拟。本 BL51 连接器为位于其他段中的函数生成一个跳转表，当你用 C 语言调用一个位于不同的段中的函数时，它要先切换段，再跳到目标程序运行，完成后再回到调用的那个段中去，并继续往下执行。这种段切换处理需要附加的 50 个 CPU 指令周期和占用 2Bytes 堆栈空间。

如果你把相关的函数分配在相同的段中将显著的提高系统的性能。那些需要从多个段中经常调用的函数应该位于公共段中。

## 映象文件

下面是 BL51 产生的一个例子文件:



```
BL51 BANKED LINKER/LOCATER V4.00        07/01/99  08:00:00  PAGE 1

MS-DOS BL51 BANKED LINKER/LOCATER V4.00, INVOKED BY:
C:\KEIL\C51\BIN\BL51.EXE SAMPLE.OBJ

MEMORY MODEL: SMALL

INPUT MODULES INCLUDED:
  SAMPLE.OBJ (SAMPLE)
  C:\C51\LIB\C51S.LIB (?C_STARTUP)
  C:\C51\LIB\C51S.LIB (PUTCHAR)
  C:\C51\LIB\C51S.LIB (GETCHAR)
  C:\C51\LIB\C51S.LIB (TOUPPER)
  C:\C51\LIB\C51S.LIB (_GETKEY)


LINK MAP OF MODULE:  SAMPLE (SAMPLE)

     TYPE    BASE      LENGTH    RELOCATION   SEGMENT NAME
     -----------------------------------------------------

     * * * * * * * *   D A T A   M E M O R Y   * * * * * * * *
     REG     0000H     0008H     ABSOLUTE     "REG BANK 0"
     DATA    0008H     0001H     UNIT         ?DT?GETCHAR
     DATA    0009H     0001H     UNIT          DATA GROUP
             000AH     0016H                   *** GAP ***
     BIT     0020H.0   0000H.1   UNIT         ?BI?GETCHAR
             0020H.1   0000H.7                 *** GAP ***
     IDATA   0021H     0001H     UNIT         ?STACK

     * * * * * * * *   C O D E   M E M O R Y   * * * * * * * *
     CODE    0000H     0003H     ABSOLUTE
     CODE    0003H     0021H     UNIT         ?PR?MAIN?SAMPLE
     CODE    0024H     000CH     UNIT         ?C_C51STARTUP
     CODE    0030H     0027H     UNIT         ?PR?PUTCHAR?PUTCHAR
     CODE    0057H     0011H     UNIT         ?PR?GETCHAR?GETCHAR
     CODE    0068H     0018H     UNIT         ?PR?_TOUPPER?TOUPPER
     CODE    0080H     000AH     UNIT         ?PR?_GETKEY?_GETKEY


OVERLAY MAP OF MODULE:   SAMPLE (SAMPLE)

SEGMENT                          DATA GROUP
  +--> CALLED SEGMENT          START    LENGTH
-----------------------------------------------
?C_C51STARTUP                  -----    -----
  +--> ?PR?MAIN?SAMPLE

?PR?MAIN?SAMPLE                0009H    0001H
  +--> ?PR?GETCHAR?GETCHAR
  +--> ?PR?_TOUPPER?TOUPPER
  +--> ?PR?PUTCHAR?PUTCHAR

?PR?GETCHAR?GETCHAR            -----    -----
  +--> ?PR?_GETKEY?_GETKEY
  +--> ?PR?PUTCHAR?PUTCHAR


LINK/LOCATER RUN COMPLETE.  0 WARNING(S),  0 ERROR(S)
```

*BL51 produces a MAP file (extension .M51) with date and time of the link/locate run.*

*BL51 displays the invocation line and the memory model.*

*Each input module and the library modules included in the application are listed.*

*The memory map contains the usage of the physical 8051 memory.*

*The overlay-map displays the structure of the program and the location of the bit and data segments of each function.*

*Warning messages and error messages are listed at the end of the MAP file. These may point to possible problems encountered during the link/locate run.*

- BL51 产生一个包含连接时的时间和日期的映象文件(*.M51)。
- BL51 显示调用的命令和存储模式。
- 应用中包含的每个模块和库模块被列出来。
- 存储器映象文件包含 8051 实际存储器的使用信息。
- 覆盖映象图显示了程序结构和每个函数的数据和位段。
- 错误和告警总数包括在文件的结尾处，这些映象图指出在连接定位时可能面临的问题。

## LIB51 库管理器

本库管理器让你建立和维护库文件。一个库文件是格式化的目标模块（由编译器或汇编器产生）的集合。库文件提供了一个方便的方法来组合和使用大量的连接程序可能用到的目标模块。利用 uVision2 项目管理器的 Options for Target – Output – Create Library 选项可以建造一个库。你也可以从命令行运行 LIB51 程序。命令行参数参照 218 页"LIB51 / L251 Library Manager Commands。

使用库有一系列优点。安全，高速和减少磁盘空间仅仅是使用库的一小部分原因。另外，库提供了一个好的分发大量函数而不用分发大量函数源代码的手段。例如，ANSI C 的库是作为一套库文件提供的。

uVision2 项目 C:\KEIL\C51\RTX_TINY\RTX_TINY.UV2 允许你修改和创建 RTX51 小型实时操作系统库。你很容易创建你自己的库，用来包括象串行 I/O，CAN 和闪存操作这样一些你自己一次又一次要用到的流程。一旦这些流程调试无误后，你就可以把它们转换成库。由于库只包含目标模块，不用在每个项目中重新编译这些模块，所以生成应用的时间将缩短。

连接定位程序连接最终应用是用到的库文件。库中的模块仅仅在需要的时候才被提取加到程序中，没有被你的应用调用的库函数不会出现在你的最终结果中。连接器把从库中提取出来的模块和其他目标模块做同样的处理。

## OC51 分段目标文件转换器

此 OC51 转换器为一个分段目标模块中的每一个代码段创建绝对的目标模块。分段目标模块是你生成一个分段代码切换应用时由 BL51 创建的。字符变量的调试信息被拷贝到转换后的绝对目标模块中，以便给 uVision2 调试器和其他仿真器使用。

你可以从命令行用 OC51 去为你分段目标模块中的每一个代码段创建绝对目标模块。

然后，你还可以用 OH51（目标代码到 HEX 文件的转换器）为每一个绝对目标模块产生相应的 Intel HEX 格式的文件。

## OH51 目标代码到 HEX 文件的转换器

此转换器为绝对目标模块创建 Intel HEX 格式的文件。绝对目标模块可以由 BL51 或 OC51 产生。 Intel HEX 文件是 ASCII 文件，它用十六进制的数表示你的应用系统的目标模块。它们可以很容易的下载到编程器，以便写入 EPROMS 器件。

# 第 4 章　创建应用

我们提供了一个测试版本，测试版本中包含示例程序和我们工具的受限使用版本。从而使你很容易的评估，熟悉我们的系列产品。测试版本中的示例程序同样包含在正式版本中。

注意：

Keil C51 测试版本在功能和你能够创建的应用的代码长度上都有限制。关于这点的详细信息请参考“版本发行说明”。要创建大的应用系统，你必须购买其中一个开发包。关于开发包（套件）的详细描述，参考 P16 的“产品一览”。

本章描述了 uVision2 的创建模式并向你演示如何使用它来创建一个简单的程序(译者注：原文为示例程序)。以及生成和维护项目的一些选项：包括文件输出选项，C51 编译器的关于代码优化的配置，uVision2 项目管理器的特性等。

## 创建项目

uVision2 包括一个项目管理器，它可以使你的 8051 应用系统设计变得简单。要创建一个应用，你需要按下列步骤进行操作：

- 启动 uVision2，新建一个项目文件并从器件库中选择一个器件。
- 新建一个源文件并把它加入到项目中。
- 增加并配置你选择的器件的启动代码。
- 针对目标硬件设置工具选项。
- 编译项目并生成可以编程 PROM 的 HEX 文件。

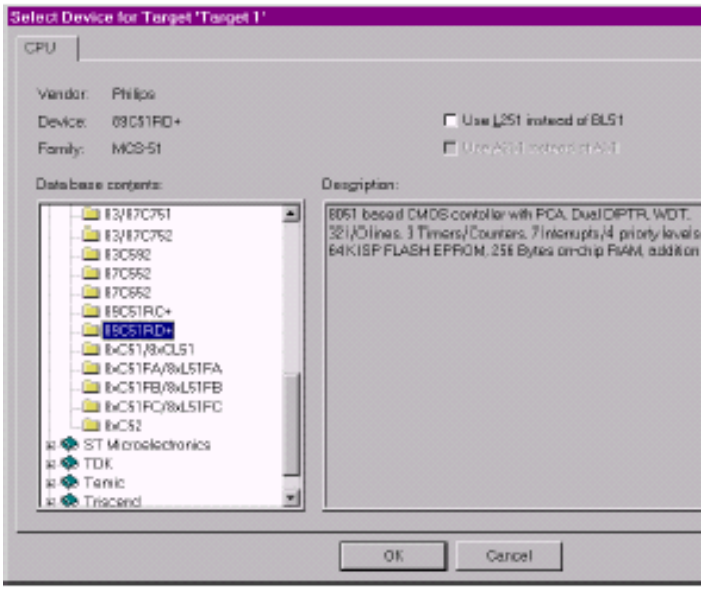
这里将一步一步的进行描述，从而指引你如何去创建一个简单的 uVision2 项目。

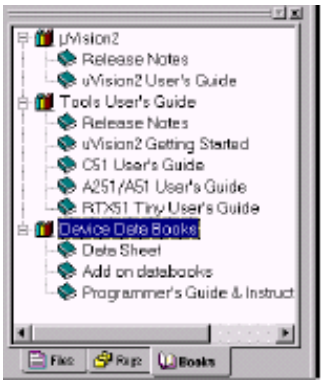
## 启动 uVision2 并创建一个项目

uVision2 是一个标准 Windows 应用程序，直接点击程序图标就可以启动它。要新建一个项目文件，从 uVision2 的 Project 菜单中选择 New Project，这将打开一个标准的 Windows 对话框，此对话框要求你输入项目文件名。

我们建议你为每个项目建一个单独的文件夹。你可以在弹出的对话框中点击新建文件夹的图标来得到一个空的文件夹，然后选择子文件夹并键入项目的名称，如Project1。uVision2 将创建一个文件名为Project1.UV2的新项目文件。新的项目文件包含了一个以默认的文件名命名的目标和文件组。你可以在项**Project Window – Files**看到这些名字。

现在从菜单 Project – Select Device for Target 为你的项目选择一个 CPU。弹出的对话框中显示的是器件数据库。你只要选择所需要的 MCU 就可以了。在例子程序中我们选择 Philips 80C51RD + CPU。该选择就为 80C51RD+器件设置了工具选项，这种方式简化了工具的配置。



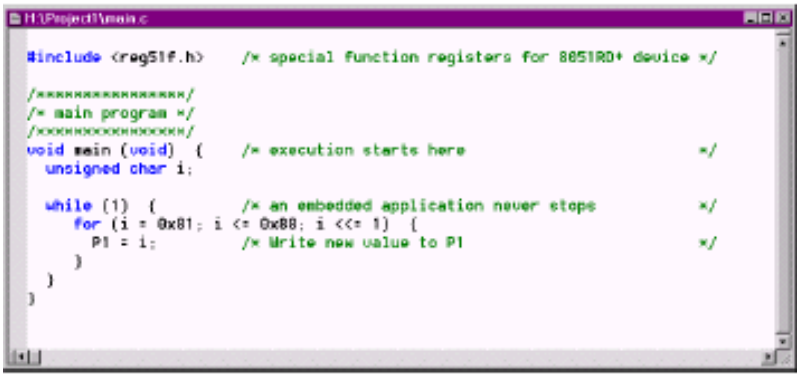

***注意:***

*对于一些器件，uVision2 环境需要你手工输入一些附加的参数。仔细阅读此选择器件对话框中 Description 下面的信息，它可能提供了你所选择器件的一些附加的说明。*

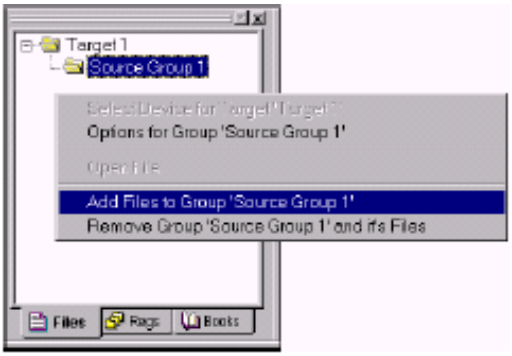
一旦你从器件库中选择了一个 CPU，你就可以在项目窗口的 Books 页打开此 CPU 的用户手册。这些用户手册是 Keil 开发工具光盘中的一部分。

## 新建一个源文件

你可以用菜单选项 File-New 来新建一个源文件。这将打开一个空的编辑窗口让你输入你的源代码。当你把此文件另存为*.C 的文件后，uVision2 将高亮显示 C 语言语法字符。我们把我们的例子程序保存为 MAIN.C。



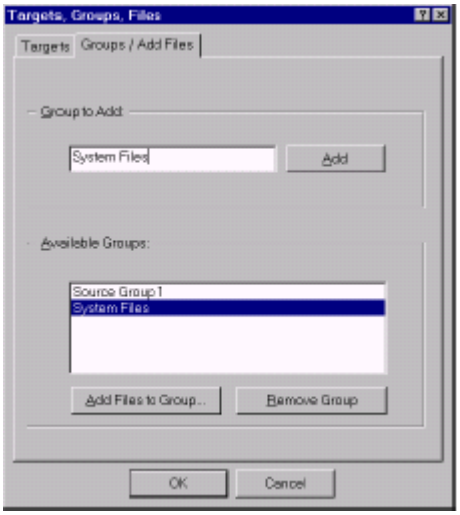

一旦你创建了源文件，你就可以把它加入到你的项目中。uVision2 提供了几种手段让你把源文件加入到项目中。例如，你可以右击 Project 窗口 - Files 页中的文件组来弹出快捷菜单，菜单中的 Add Files 选项打开一个标准的文件对话框，从对话框中选择你刚刚生成的文件 MAIN.C。




## 增加和配置启动代码.

文件 STARTUP.A51 是大多数不同的 8051CPU 准备的启动代码。启动代码清除数据存储器并初始化硬件和重入函数堆栈指针。另外，一些 8051 派生产品要求初始化 CPU 来迎合你设计中的相应的硬件。例如,Philips 8051RD+提供的片上 xdata RAM 应该在启动代码中启用。假如你需要修改启动文件来迎合你的目标硬件，你应该把文件 STARTUP.A51 复制一份到你的项目文件夹中。

为你选择的 CPU 的配置文件创建一个文件组是一个良好的习惯。用菜单 **Project – Targets, Groups, Files…** 打开对话框来添加一个名为 System Files 的文件组到你的目标中。也在此对话框中，你用 **Add Files to Group…**按钮把文件 STARTUP.A51 添加到你的项目中。

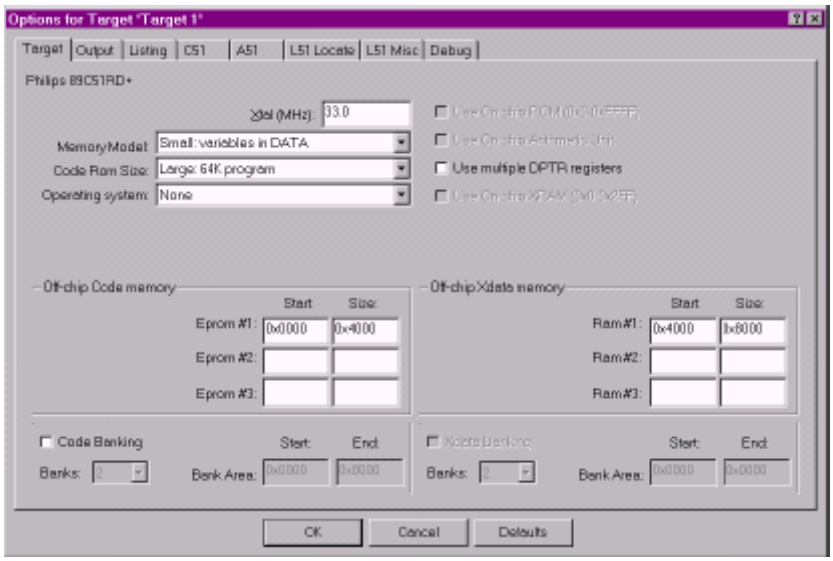



项目窗口的文件页列出了你项目的所有条目.


现在,你的 uVision2 的 **Project Window － Files** 应该显示上图中的文件结构。在项目窗口中双击文件名 STARTUP.A51，你就可以把它在编辑器中打开。然后按照 P197 页的第十章(CPU and C Startup Code)的描述来配置启动代码。 如果你使用你所选择器件的片上 RAM，在启动代码中的设置必须匹配 **Options － Target** 对话框中的设置。下面来讨论 **Options － Target** 对话框。

## 为目标设置工具选项

uVision2 允许你为你的目标硬件设置选项。Options for Target 对话框可以通过工具条图标打开。在目标的各个页中，你可以定义和你的目标硬件及你所选器件的片上元件相关的所有参数。下图显示了我们例子的设置：



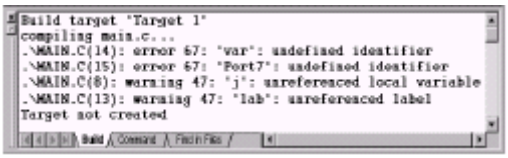

下表描述了目标对话框的一些选项：

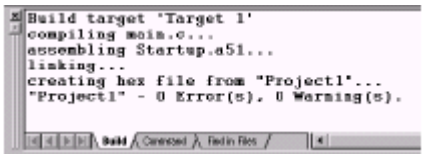
| 对话框条目 | 描述 |
|---|---|
| Xtal | 定义 CPU 时钟，对于大多数应用中和实际的 XTAL 频率相同。 |
| Memory Model | 定义编译器的存储模式。对于一个新的应用，默认的是 SMALL 模式。参照 P78 存储模式和存储器类型的讨论。 |
| Allocate On-chip<br>Use multiple DPTR registers | 定义在启动代码中使能的片上元器件的使用。如果你使用片上 xdata RAM，那你应该在文件 STARTUP.A51 中使能 XRAM 的访问。 |
| Off-chip … Memory | 在此定义你目标硬件上所有的外部存储器区域。 |
| Code Banking<br>Xdata Banking | 为代码和数据的分段（Banking）定义参数。详细信息参照 P67 代码分段（Code Banking）。 |

*注意：*
*有些选项只有在你使用 LX51 连接器时才有用。LX51 连接器只在 PK51 中提供.*

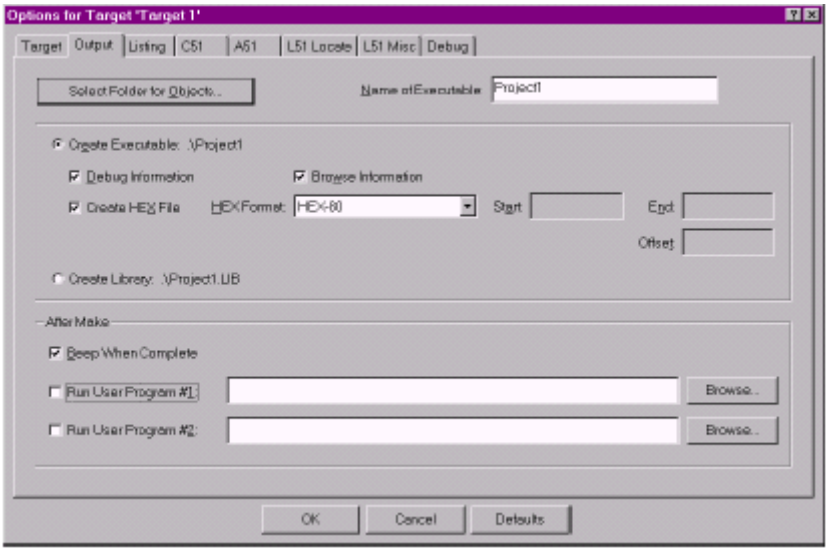
## Build 项目并生成 HEX 文件

通常情况下，在 Options － Target 对话框中的设置已经足够使你开始一个新的应用。通过单击工具条上的 Build 目标的图标，你可以编译所有的源文件并生成应用。当你的应用中有语法错误时，uVision2 将在 Output Window － Build 页显示这些错误和告警信息。双击一个信息将打开此信息对应的文件并定位到语法错误处。




一旦你成功的生成了你的应用，你就可以开始调试了。uVision2 调试器功能的讲述请参照 P93 页“Chapter 5”的 Testing Programs"。在你调试完你的应用后，需要创建一个 HEX 文件来烧片子或软件模拟。当 **Options for Target － Output** 中的输出 HEX 文件使能时，uVision2 每进行一次 Build 都生成 HEX 文件。如果定义了 **Options for Target － Output** 中的 **Run User Program #1** 选项时，在生成操作完成后，将自动运行此处定义的操作。如编程 PROM 器件。



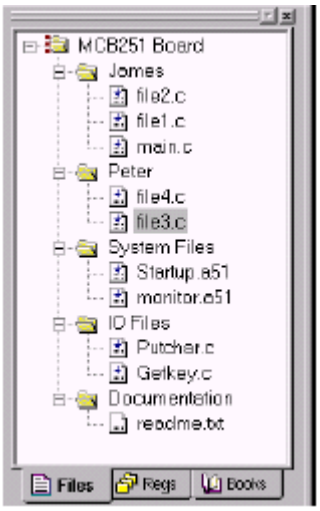
现在你能够修改已经存在的源文件或添加一个新的源文件到项目中。Build 目标工具条按钮只编译修改过的或新加进来的文件，然后生成执行文件。uVision2 维护一个文件包含清单，从而知道某个源文件用到的所有的包含文件。而且工具配置的选项也被保存在此清单中，所以 uVision2 能够只编译那些需要重新编译的文件。利用 Rebuild All Target(原文为 Rebuild Target)命令，所有的文件都被重新编译，而不论是否被修改过。

## 项目目标和文件组

通过使用不同的项目目标，uVision2 允许你为一个应用创建几个不同的程序。你也许需要一个目标用来测试，另一个目标是你应用系统的发行版本。在同一个项目文件中，允许每个目标进行独立的工具设置。

**文件组**是用来让你把项目中相关的文件放在一组。这在要求把文件按功能块组织和确定你的软件开发团体中的工程师们的职责时特别有用。我们已经在本例子中使用了文件组去分离与 CPU 相联系的文件和其它文件。利用这个技术，可以轻松的管理拥有几百个文件的复杂项目。

**Project － Targets, Groups, Files…** 对话框允许你创建项目目标和文件组。我们已经用此对话框增加了系统配置文件。下图中显示了一个例子项目的结构。




项目窗口向你展示所有的组和相联系的文件。文件按照显示的顺序编译和连接。你可以通过拖放来移动文件的位置。你也可以单击目标或组的名字去修改它。右击将打开弹出菜单，弹出菜单可以让你：

1)设置工具选项
2)删除条目
3)添加文件到组
4)打开文件

在 Build 工具条上，你可以快速改变当前的目标。

## 浏览项目窗口中的文件和文件组的属性

在项目窗口的文件页中用不同的图标来标识不同的文件和文件组属性。下面是这些图标及其对应的属性的描述：


此图标是在文件图标上加一箭头而成，用来表示被编译、连接到项目中去的文件。


文件图标，用来表示不被连接到项目中去的文件，典型的如文档文件。另外，在文件的属性窗口中取消**"Include in Target Build"**复选项,将使该文件剔除出项目。剔除出项目的文件也用此图标。参照 87 页的**“文件和文件组的详细选项 － 属性对话框”**


图标上有一把钥匙，用来表示只读文件。典型的如被软件版本控制系统(SVCS)控制的文件，因为 SVCS 把他所控制的文件的本地拷贝设置为只读属性。参照 76 页的"使用 SVCS 菜单"


此图标左边有三个点，用来表示有特殊选项的文件(图 4)或文件组(图 5)。参照 87 页的**“文件和文件组的详细选项 － 属性对话框”**

---

***注意:***

*不同的图标让你能够快速浏览到一个项目不同的目标中的工具设置。*

*你所看到的图标总是反映当前所选目标的属性。例如，你在一个目标中对一个文件或文件组设置了特殊选项，那只有在你选择了此目标时，那些你设置了特殊选项的文件的图标上才会有三个点。*

---

*/**********译者注-开始**********/*

*第二个图标为文件图标，所有属性文件的图标都以此为基础，再加上箭头，钥匙，三点中的零到三个，来表示文件对于所在目标的属性。更进一步说，只读是属于文件自己的，即一个文件具有只读属性，那它在任何目标中都具有。但，是否包含在项目中，是否设置了特殊选项,是文件对于目标的属性，即在一个目标中的一个文件的图标上有箭头和(或)三点,在另一个目标中并不一定如此。*

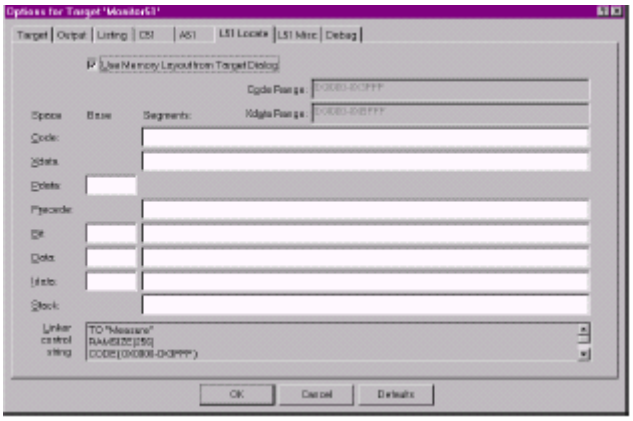
*/**********译者注-结束**********/*

## 浏览配置对话框

此选项对话框让你设置所有的工具选项。通过 **Project Window － Files** 中的弹出式菜单，你可以为文件组或者一个单独的文件设置不同的选项；在此处的描述中，你仅仅能够获得对话框的相应的标签页。利用对话框中的帮助按钮，你可以获得大多数对话框条目的帮助。下表描述了目标对话框的选项：

| 对话框的标签页 | 描述 |
|---|---|
| Target | 定义你应用的硬件。详细信息参见第 62 页。 |
| Output | 定义 Keil 工具的输出文件并让你定义生成处理后执行的用户程序。更多信息参见 P82。 |
| Listing | 定义 Keil 工具输出的所有列表文件。 |
| C51 | 设置 C51 编译器的特别的工具选项，如：代码优化或变量分配。更多信息参见 P80。 |
| A51, AX51 | 设置汇编器的特别的工具选项，如宏处理。 |
| L51 Locate<br>LX51 Locate | 定义不同类型的存储器和存储器的不同段的位置。典型情况下，你可以选择**"Memory Layout from Target Dialog"**来获得自动设置，如下图所示。更多信息参见 P86。 |
| L51 Misc<br>LX51 Misc | 其它的与连接器相关的设置，如：告警或存储器指示。 |
| Debug | μVision2 Debugger 的设置。更多信息参见 P101。 |
| Properties | 文件和文件组的文件信息和特别选项，参照 P87 的**"File and Group Specific Options － Properties Dialog"** |

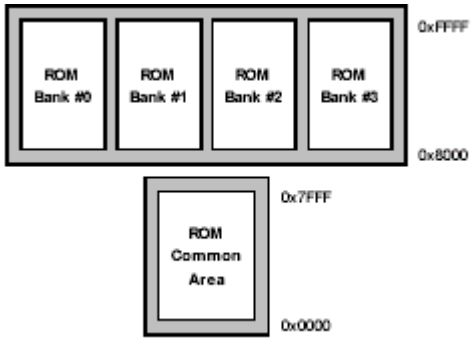
在 L51 标签页中，一旦你使能 **Use Memory Layout from Target Dialog** 选项，uVision2 使用从你所选择器件和目标标签页中所得到的存储器信息。你还可以添加附加的段到这些设置中。



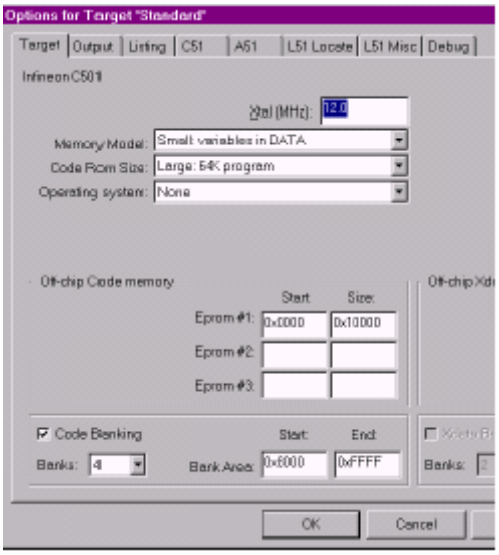
## 代码分块（Code Banking）

一个标准的 8051 器件能寻址 64KB 的代码空间。为了突破 64KB 程序空间的限制，Keil 8051 工具支持代码分块。这个技术让你管理一个公共的区域和 32 个每个最大达 64KB 的块，从而达到总共 2MB 的代码切换空间。

例如：你的硬件设计可以包括一个位于 0x0000-0x7fff 空间的 32KB 的 ROM（如你所知的公共区域或公共 ROM），和四块位于 0x8000-0xffff 空间的 32KB 的 ROM（如你所知的代码块 ROM）。通常情况下，代码块通过端口线选择。右图显示了此应用的存储器结构。



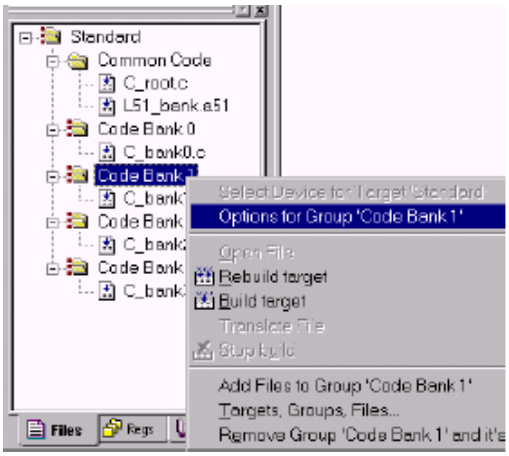

代码分段在 **Options for Target － Target** 标签页中使能和配置。在此对话框中，输入你硬件支持的代码分块的个数，和分块的区域。上例中的存储器分块的配置如右图所示。

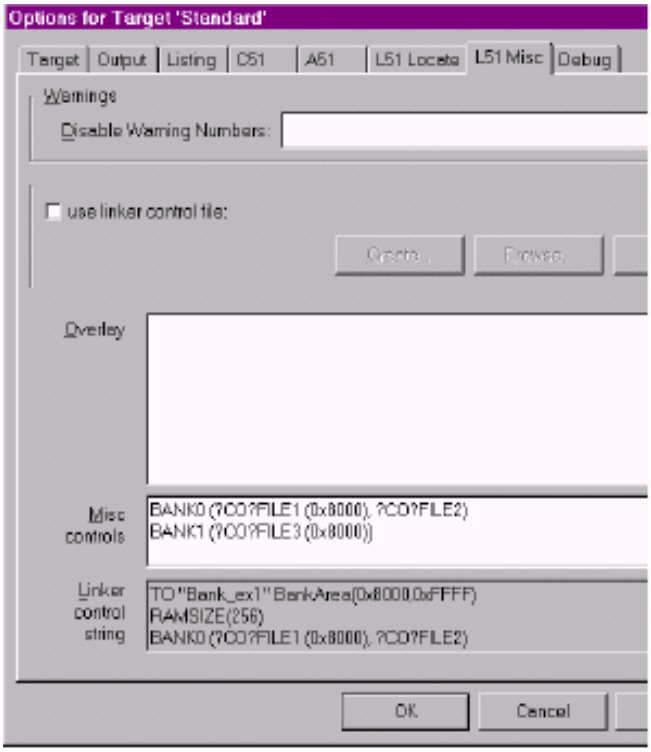



为了配置你的分块的硬件，你需要将文件 **C51\LIB\L51_BANK.A51** 加入到你的项目中。复制此文件到你的项目文件夹，把它和你项目中的其它源文件放在一起，并添加它到你项目的一个文件组。文件 **C51\LIB\L51_BANK.A51** 必须修改以便迎合你的目标硬件。

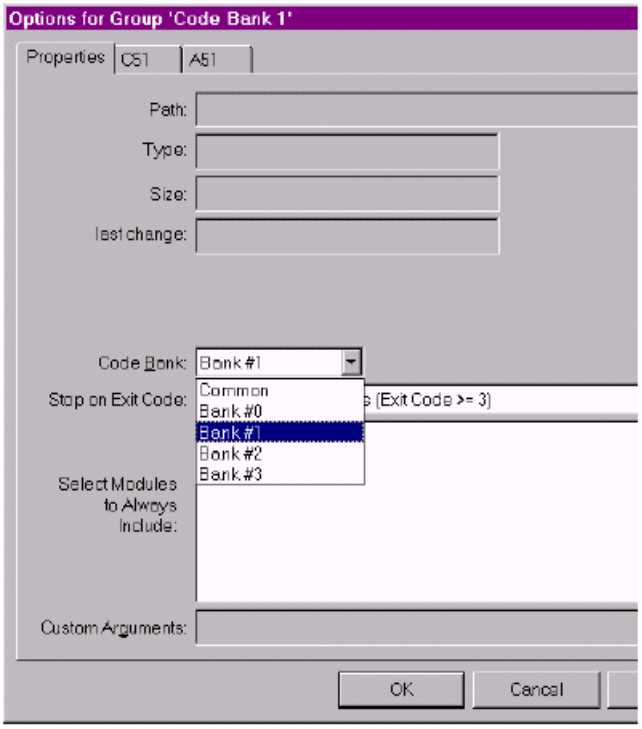
每个文件或文件组通过其 **Options － Properties** 对话框可以被定义到一个特定的代码块中。

通过右击项目窗口中的文件或文件组来打开此文件或文件组的 **Options – Properties** 对话框。此对话框允许你选择哪个代码块或公共区域。




公共区域可以在所有的代码块中访问，此公共区域常常包括那些必须一直需要访问的进程和数据常量，如：中断进程，中断和复位向量，字符串常量和块切换进程。所以，连接器只把一个模块中的程序段定位到相应的块区域中（数据常量放在公共区域中）。如果你能够确定某些常量段的信息只被某个特定的代码块访问，你可以在 **Options for Target – L51 Misc** 中用 **BANKx** 来指示连接器把这些常量段定位到此特定的代码块中。




以上的步骤完成了你的代码分块应用的配置。uVision2 调试器完全支持代码分块，从而让你能够调试程序。

如果你在 **Options for Target – Output** 对话框中使能**"Create HEX File"** 选项，uVision2 将为每个代码块生成一个从地址 0 开始的 64KB 的物理映像。你需要用你的 PROM 编程器把这些 HEX 文件写到你的 EPROM 中相应的存储空间中去。

# uVision2 功能

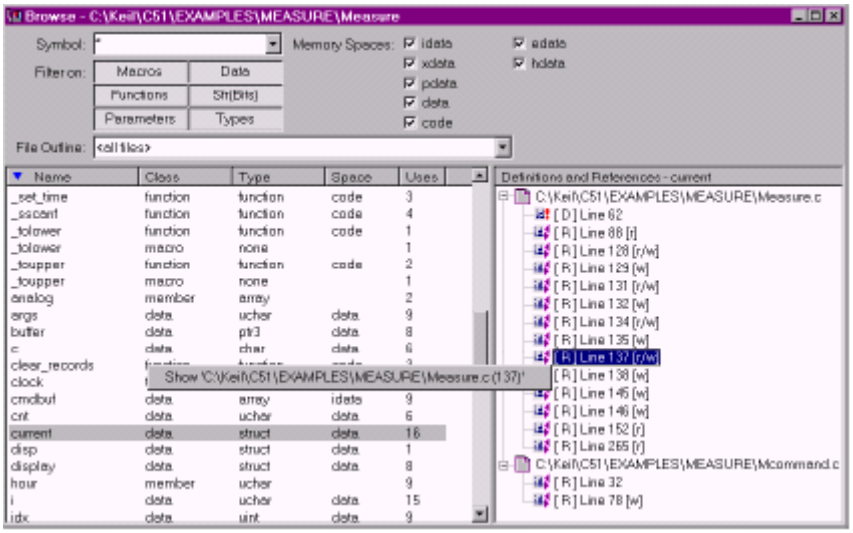
uVision2 包含许多强大的功能，为你的整个软件项目中提供帮助。这些功能将在下面几节描述：

## 多个文件中查找

菜单 **Edit** － **Find in Files** 打开一个对话框，实现在所有特定的文件中查找一个字符。查找结果显示在输出窗口的 **Find in Files** 页。双击输出窗口的 **Find in Files** 页中的某个输出，编辑器将定位到匹配此字符串的文本行。

## 资源浏览器

此资源浏览器显示你程序中的所用的符号的信息。如果 **Options for Target** － **Output** 中 **Browser Information** 选项被选中，编译器编译时将把浏览信息包含到项目文件中。用菜单 **View** － **Source Browser** 去打开资源浏览器窗口。



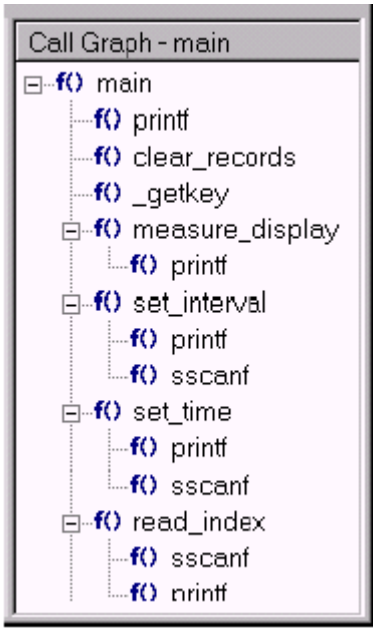

此浏览窗口列出了符号名，类，类型，存储器空间和使用的次数。单击列名，将按此列排序显示信息。你可以用下表描述的选项来过滤浏览的信息：

| 资源浏览器选项 | 描述 |
|---|---|
| Symbol | 定义一个屏蔽格式，用来显示与其相匹配的符号。<br>此屏蔽格式可以包含实际的字符和以下通配字符：<br># 匹配数字 0-9<br>$ 匹配任意字符<br>* 匹配任意字符串（包括 NULL） |
| Filter on | 选择需要显示的符号的类型 |
| File Outline | 选择一个文件，显示此文件中的符号资源 |
| Memory Spaces | 定义需要显示的符号的存储类型 |

下表中列出了一些符号代码的例子

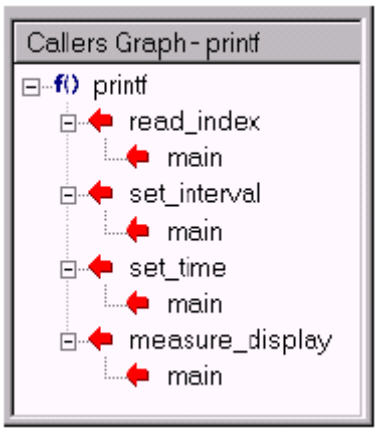
| 代码 | 代表的符号名 |
|---|---|
| * | 可代表任何符号。在符号浏览器中是默认的代码。 |
| *#* | 可代表有一个数字的符号，数字在符号的任何位置。 |
| _a$#* | 代表的符号为下划线开头，接字母 a,任一字母，任一数字，最后可不加字符也可加任何字符。 |
| _*ABC | 下划线加任意字符或不加，然后接 ABC |

右击资源浏览器 Definitions and references 框中显示的定义或引用，弹出的菜单可以让你在编辑器打开相应的内容。对于函数，你还可以看到调用和被调用的关系图。在此 Definitions and references 框中还为你提供了一些附加的信息：




| 符号 | 描述 |
|---|---|
| [D] | 定义处 |
| [R] | 引用处 |
| [r] | 读操作处 |
| [w] | 写操作处 |
| [r/w] | 读/写操作处 |
| [&] | 地址引用处 |

你可以使用编辑窗口中的浏览信息。选中你要找的条目并用鼠标右键打开局部菜单。或者使用下列键盘快捷键。




| 快捷键 | 描述 |
|---|---|
| F12 | 跳转到定义处；将光标置到符号定义处。 |
| Shift+F12 | 跳转到引用处；将光标置到符号引用处。 |
| Ctrl+Num+ | 跳转到下一处的引用或定义处。 |
| Ctrl+Num - | 跳转到前一处的引用或定义处 |

## 开发工具参数的键序列值表示

一个键序列可以用来从 uVision2 开发环境向外部用户程序传递参数。键序列可以应用在工具菜单，SVCS 菜单以及用户程序（在 **Options for Target-Output** 对话框中定义）的参数传递中。一个键序列是键码和文件码的组合。下面列出有效的键码和文件码：

| 键码 | 定义由文件码所确定的文件的路径 |
|---|---|
| % | 包含扩展名的文件名，但是不包含路径（如：PROJECT1.UV2） |
| # | 包含绝对路径的文件名（如：C:\MYPROJECT\PROJECT1.UV2） |
| % | 包含扩展名的文件名,但是不包含路径（如:PROJECT1.UV2） |
| @ | 不包含路径和扩展名的文件名（如：PROJECT1） |
| $ | 文件夹名（如：C:\MYPROJECT） |
| ~ | 当前光标所在位置的行号（仅仅在文件码为 F 时有效） |
| ^ | 当前光标所在位置的列号（仅仅在文件码为 F 时有效） |

注：键码~和^只在文件码为 F 时有效。要在用户程序的命令行使用$, #, %, @, ~ 或 ^，用$$, ##, %%, @@, ~~ 或 ^^格式。例如，@@在用户程序的命令行上提供一个单独的@字符。

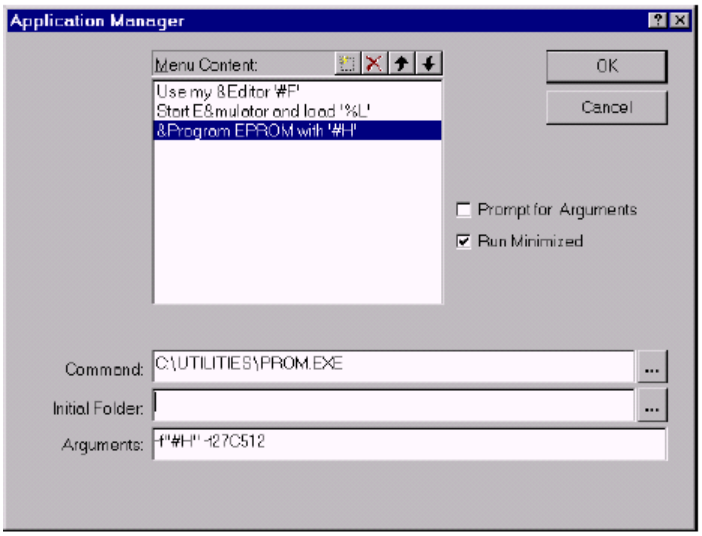
| 文件码 | 定义插入在用户程序命令行上的文件名或参数 |
|---|---|
| F | 在 **Project Window - Files** 页中选定的文件（如：MEASURE.C）。如果选择的目标名，则返回项目文件；如果选择的文件组名，则返回当前活动编辑器中的文件。 |
| P | 当前项目名（如：PROJECT1.UV2） |
| L | 连接器输出文件，通常为调试而生成的可执行文件（如：PROJECT1） |
| H | HEX 应用文件（如：PROJECT1.H86） |
| | uVision2 可执行文件（如：C:\KEIL\UV2\UV2.EXE） |

以下是应用在 SVCS 系统中的文件码，更详细的信息参照 P76 的**“使用 SVCS 菜单”**

| 文件码 | 定义插入在用户程序命令行上的文件名或参数 |
|---|---|
| Q | 为 SVCS 系统保持注释的文件名 |
| R | 为 SVCS 系统保持修正码的字符串 |
| C | 为 SVCS 系统保持校验码的字符串 |
| U | 用户名（在 **SVCS-Configure Version Control -User Name** 中定义） |
| V | 文件名（在 **SVCS-Configure Version Control -Database** 中定义） |

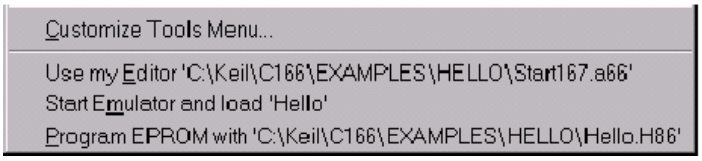
Q, R, C, U 和 V 只能和键码%组合使用。

## 使用工具菜单




通过工具菜单，你可以运行外部程序。要添加自定义程序到工具菜单，可以通过菜单 **Tools – Customize Tools Menu...**打开对话框，在此对话框中配置外部用户应用的参数。右边的对话框示例了一个工具设置。

上图所示的对话框的设置将扩展工具菜单，如右所示。




此对话框的选项说明如下：

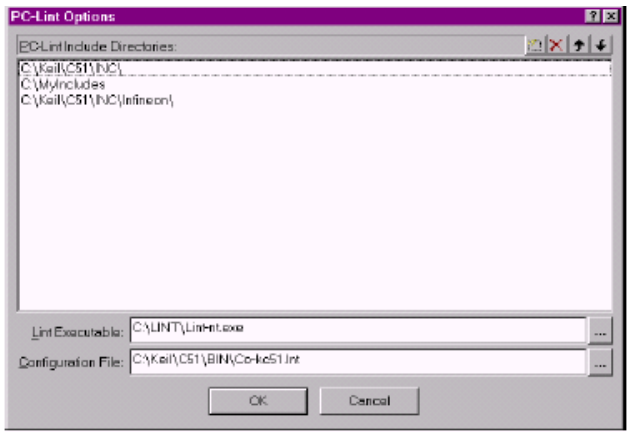
| 对话框条目 | 描述 |
|---|---|
| Menu Content | 工具菜单上显示的文本。其中的每行都可以包括键码和文件码。用与号(&)来定义一个快捷键。你可以定义当前选择的菜单行的在下面列出的各个选项。 |
| Prompt for Arguments | 如果选中，那么在你点击此菜单时，一个对话框将弹出，提示你输入用户程序的命令行参数。 |
| Run Minimized | 如果选中，将使此应用程序运行时的窗口最小化。 |
| Command | 输入你点击此菜单时运行的程序的路径。 |
| Initial Folder | 应用程序的当前工作目录。如果为空，uVision2 用当前项目所在的目录。 |
| Arguments | 传递给应用程序的命令行参数。 |

基于应用程序的命令行输出被复制到一个临时文件中，当此应用执行完成后，此临时文件的内容将在 **Output Window -Build** 页列出。

### 运行 PC-Lint

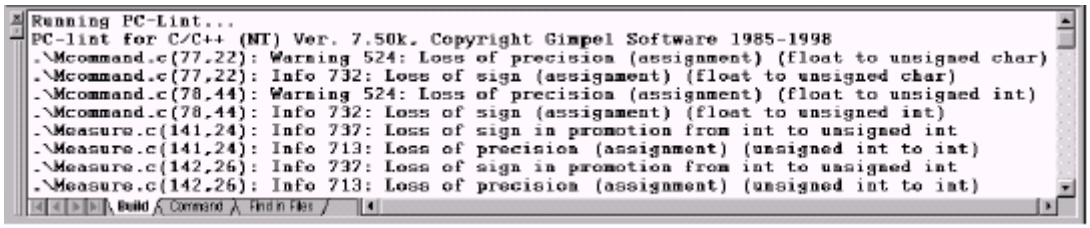
Gimpel Software 的 PC-Lint 核对你应用中的所有模块 C 源代码的语法和语义。PC-Lint 标识出可能的错误和矛盾的地方，并停在模糊的、错误的或无效的 C 代码处。PC-Lint 可以显著地减少你花费在目标应用上的调试精力。

在你的 PC 机上安装 **PC-Lint**，然后在 **Tools – Setup PC Lint** 对话框中输入参数。本例显示了一个典型的 PC-Lint 的配置。




为了在 **Output Window-Build** 页得到正确的输出，你应该使用位于 **KEIL\C51\BIN** 文件夹中的配置文件。

在安装 **PC-Lint** 后，你就可以检查(lint)你的源代码了。菜单 **Tools – Lint** 运行 **PC-Lint** 来检查当前编辑器中的文件。菜单 **Tools – Lint All C Source Files** 运行 **PC-Lint** 来检查你项目中所有的 C 源文件。**PC-Lint** 的输出信息被重定向到 **Output Window-Build** 页。双击 **PC-Lint** 的输出信息将使编辑器定位到相应的位置。




为了在 **Output Window-Build** 页获得正确的结果，PC-Lint 需要在配置文件中包含以下的选项：

```
-hsb_3                                          // 3 lines output, column below
-format="*** LINT: %(%f(%l) %)%t %n: %m"        // Change message output format
-width(0,10)                                    // Don't break lines
```

配置文件 **C:\KEIL\C51\BIN\CO-KC51.LNT** 已经包含这些行。强烈推荐使用此配置文件，因为它还包含 Keil 51 编译器需要其它的 **PC-Lint** 选项。

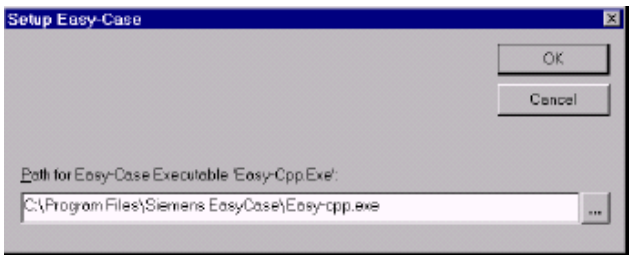
**Siemens Easy-Case**

µVision2 为 Siemens Easy-Case 提供了一个直接的接口。Easy-Case 是一个图形和程序文档编辑器。你可以用 Easy-Case 编辑源程序代码。另外，一些µVision2 调试器中的命令在 **Easy-Case** 环境中也有效。

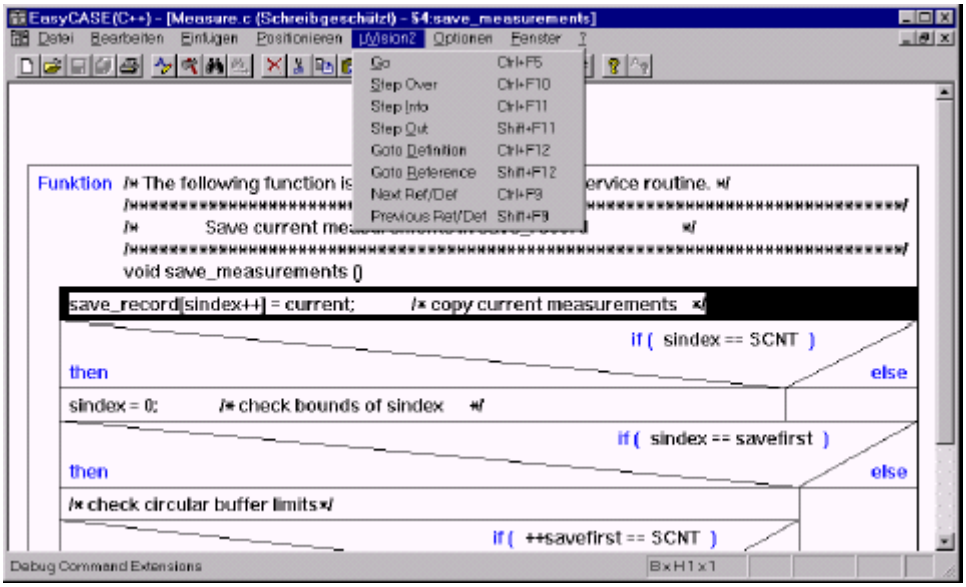
**安装 Easy-Case:** 为了在 Easy-Case 中使用µVision2 调试器命令，文件 **C:\KEIL\UV2\UV2EASY-CPP.INI** 中的配置设置应该增加到 WINDOWS 系统目录中的文件 **EASY-CPP.INI** 中。这可以通过任何文本编辑器或 DOS 的 copy 命令来实现。copy 命令如下：

```
C:\>CD C:\WINNT
C:\WINNT>COPY EASY-CPP.INI+C:\KEIL\UV2\UV2EASY-CPP.INI EASY-CPP.INI
```

在µVision2 的 **Tools – Setup Easy-Case** 对话框中输入 **EASYCPP.EXE 的路径**。到此就完成了 Siemens Easy-Case 的配置。



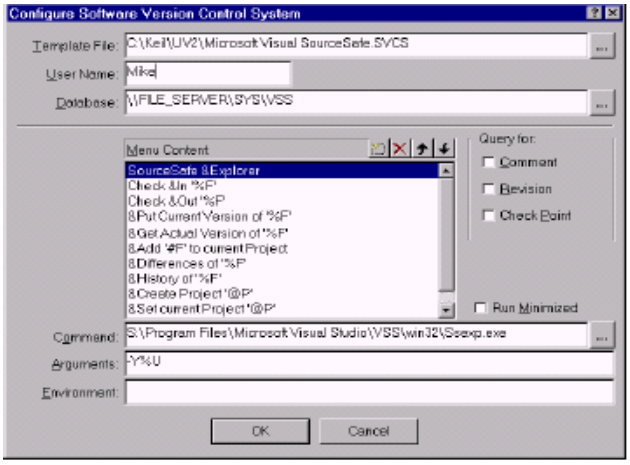

**利用 Easy-Case 浏览源代码:** 你可以用 **Tools – Start/Stop Easy-Case** 来启动 Easy-Case。菜单项目 **Tools – Show …** 在当前位置打开µVison2 中活动编辑器中的文件。Easy-Case 的菜单 **µVision2** 提供几个调试命令以允许程序在µVision2 调试器中执行。




## 使用 SVCS 菜单

uVision2 为软件版本控制系统（SVCS）提供了一个可配置的接口。为以下几个 SVCS 提供预先配置的模板文件：Intersolv PVCS,Microsoft SourceSafe, and MKS Source Integrity。

通过 SVCS 菜单，调用你的版本控制系统的命令行工具。SVCS 菜单的配置存储在一个模板文件中。



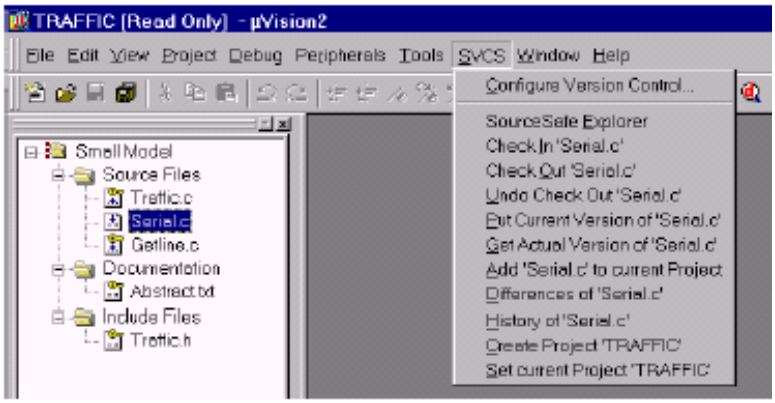

此菜单的配置通过 **SVCS - Customize SVCS Menu** 对话框进行。此对话框的选项解释如下：

| 对话框条目 | 描述 |
|---|---|
| Template File | SVCS 菜单配置文件的名字。推荐一个项目开发组的所有成员使用相同的模板文件。所以模板文件应该复制到文件服务器中。 |
| User Name | 用户名，用来登录到 SVCS 系统。在参数行它是通过%U 文件码传递的。 |
| Database | SVCS 系统使用的数据库的文件名或路径。通过%V 文件码传递。 |
| Menu Content | 显示在 SVCS 菜单中的文本行。可以包括键码和文件码。用字符'&'定义快捷键。对于当前被选中的菜单行，你可以定义以下列出的选项。 |
| Query for … Comment Revision | 允许你在执行此 SVCS 命令前询问一些附加的信息。注释将被复制到一个临时文件中，此文件通过文件码%Q 作为一个参数传递给 SVCS 命令。版本和校验信息通过文件码%R 和%C 作为字符串传递。 |

| CheckPoint | |
|---|---|
| Run Minimized | 如果你想以最小化的窗口来执行应用，使能此选项。 |
| Command | 当你点击此 SVCS 菜单项时将调用的程序文件。 |
| Arguments | 传递给此 SVCS 程序文件的命令行参数。 |
| Environment | 在执行此 SVCS 程序前需要设置的环境变量。 |

命令行 SVCS 应用程序的输出被复制到一个临时文件中。当 SVCS 命令完成后，此临时文件的内容被列出在 **Output Window － Build** 页。

如右图所示，是一个 SVCS 菜单。**在 Project Window － Files** 页一个选中的文件是一个 SVCS 参数。目标名使用***.UV2** 格式的项目名。如你所见的文件的本地拷贝是一个只读文件，所以它的图标上有一个钥匙样符号。




uVision 项目被保存为两个单独的文件。项目设置保存在***.UV2** 文件中：此文件将被 SVCS 查找，并且可以用来指导编译生成应用代码。uVision2 的本地配置被保存在***.OPT** 文件中，包含视窗位置和调试设置。

下表列出了典型的 SVCS 菜单项。根据你配置的不同，你的菜单项也许和此不同或还有其它增加的项。包含文件可以作为文档资料添加到项目中，以便 SVCS 可以迅速访问到它们。

| SVCS 菜单项 | 描述 |
|---|---|
| Explorer | 打开交互式的 SVCS 浏览器。 |
| Check In | 把文件保存到 SVCS 的数据库中，并把本地文件设置为只读属性。 |
| Check Out | 从 SVCS 中取得最新版本的文件，并把本地文件设置为可修改属性。 |
| Undo Check Out | 取消 Check Out 操作 |
| Put Current Version | 把文件保存到 SVCS 的数据库中，但对本地文件仍然可以修改。 |
| Get Actual Version | 从 SVCS 中取得一个只读文件的最新版本。 |
| Add file to Project | 添加文件到 SVCS 项目。 |
| Differences, History | 显示关于每个特定文件的 SVCS 信息。 |
| Create Project | 创建一个和本地 uVision2 项目文件名相同的 SVCS 项目。 |

---

***注意：***

*你也许要用一个文本编辑器修改预先配置好的***.SVCS** 文件来适应程序路径和工具参数。*

*在你选择一个新的项目后，Microsoft SourceSafe 要求运行**设置当前项目**命令。删除**SSUSER** 环境变量，使用工作站的登录名。*

*MKS Source Integrity 的默认配置是在服务器上预先创建一个项目数据库，工作站这边是一个本地的发送邮箱形式的工作空间。*

*Intersolv PVCS 在创建和维护项目上都没有预先配置。*

---

# 编写优化代码

许多配置参数影响你的 8051 应用的代码质量。然而，对于绝大部分应用，使用默认的工具设置都能产生良好的代码，你应该知道这些改善代码空间和执行速度的参数。这一节描述代码的优化技术。

## 存储模式和存储类型

对代码空间和执行速度影响最大的是存储模式。存储模式影响变量的存取。详细信息参照 35 页的“存储模式”。存储模式的选择在 **Options for Target － Target dialog** 页。

## 全局寄存器优化

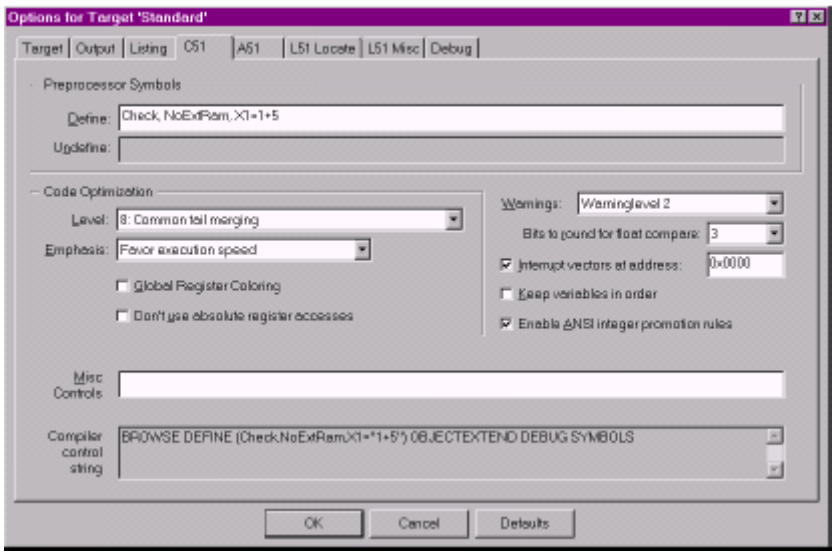
Keil 8051 工具支持大范围的寄存器优化，此选项的使能对应 **Options for Target – C51** 对话框中的 **Global Register Optimization** 选项。利用大范围寄存器优化，C51 编译器知道哪些寄存器被外部程序修改。那些没有被外部程序修改的寄存器将被用来存储寄存器变量。这样，C 编译器产生的代码将占用较少的空间，并且执行速度更快。为了改善寄存器的分配，uVision2 在 Build 时对 C 语言源程序自动进行多次编译。

下例中，input 和 output 是外部程序，只需要很少的寄存器。

| With Global Register Optimization | Without Global Register Optimization |
|---|---|
| *main () {*<br>*unsigned char i;*<br>*unsigned char a;*<br>*while (1) {*<br>*i = input ();* | */* get number of values */* |
| ?C0001:<br>LCALL input<br>;- 'i' assigned to 'R6' -<br>MOV R6,AR7 | ?C0001:<br>LCALL input<br>MOV DPTR,#1<br>MOV A,R7<br>MOV @DPTR,A |
| *do {*<br>*a = input ();* | */* get input value */* |
| ?C0005:<br>LCALL input<br>;- 'a' assigned to 'R7' -<br>MOV R5,AR7 | ?C0005:<br>LCALL input<br>MOV DPTR,#a<br>MOV A,R7<br>MOVX @DPTR,A |
| *output (a);* | */* output value */* |
| LCALL _output | LCALL _output |
| *} while (--i);* | */* decrement values */* |
| DJNZ R6,?C0005 | MOV DPTR,#1<br>MOVX A,@DPTR<br>DEC A<br>MOVX @DPTR,A<br>JNZ ?C0005 |
| *}* | |
| SJMP ?C0001 | SJMP ?C0001 |
| *}* | |
| RET | RET |
| **Code Size: 18 Bytes** | **Code Size: 30 Bytes** |

## 其它 C51 编译器指示参数

还有好几个其它的 C51 编译参数能够改善代码质量。这些参数的选择由 **Options – C51** 对话框提供。在一个应用中，你可以用不同的编译设置来编译 C 程序模块。通过列表文件，你可以比较由不同的编译设置编译生成的代码的质量。




下表描述了 C51 对话框页的选项：

| 对话框条目 | 描述 |
|---|---|
| Define | 定义预处理符号 |
| Undefine | 仅仅在组或文件的 Options 中有效，它允许你删除高一级的 Options 中定义的预处理符号。 |
| Code Optimization Level | 定义 C51 优化级别。通常情况下，你不用改变默认的值。利用其最高级别的优化：“9：公共块提取为子程序”，编译器检测多次使用的指令块，并把它们提取到一个子程序中。在分析代码时，编译器也试图用简单的指令重新安排指令序列。既然编译器插入子程序和 CALL 指令，优化后的代码的运行速度也许会变慢。此种优化往往能够显著提高代码密度。 |
| Code Optimization Emphasis | 你可以根据代码空间或执行速度为目的来选择优化。利用“Favor Code Size”，C51 编译器用库函数调用代替运行速度快的内联代码。 |
| Global Register Optimization | 使能**”Global Register Optimization”**，参见 79 页的详细描述。 |

| 对话框条目 | 描述 |
| --- | --- |
| Don't use absolute register accesses | 禁止使用寄存器 R0-R7 的绝对地址访问。由于 C51 不能使用 ARx 符号，所以代码将有一点点增加，比如：用 PUSH 和 POP 指令时，需要插入替代代码。然而，代码也将不依赖于寄存器段。 |
| Warnings | 选择 C51 输出告警信息的级别。级别 0 禁止所有告警。 |
| Bits to round for float compare | 决定浮点数比较的位数。 |
| Interrupt vectors at address | 通知 C51 编译器为中断函数产生中断向量并定义中断向量表的基本地址。 |
| Keep Variables in Order | 通知 C51 编译器根据变量在 C 语言源程序中的定义来顺序分配变量地址。此选项不影响代码质量。 |
| Enable ANSI interger promotion rules | 当为进行比较而将比整型表达式短的表达式类型转换为整型表达式时进行符号位扩展的表达式。此选项通常将增加代码长度，但是需要兼容 ANSI 标准时必须这样做。 |
| Misc Controls | 允许你输入特别的 C51 编译指导参数。当你使用一个非常新的 8051 器件而需要特别的编译指导参数时，使用此选项。 |
| Compiler Control String | 显示 C51 编译器运行时的编译指导参数的名称。让你能够立即确认你的源文件的各个编译选项。 |

## 数据类型

8051 CPU 是一个 8 位微控制器。用 8 位字节（如：char 和 unsigned char）的操作比用整数或长整数类型的操作要更有效。

# 技巧和窍门

接下来的一节讨论一些关于项目管理的一些高级技巧。你不会经常用到以下的特性，但是读了这一节，你将对 uVision2 的性能有更深的了解。

## 导入 uVision1 的项目到 uVision2

你可以用下面的步骤来把在 uVision1 中创建的项目导入到 uVision2 中：

1、创建一个新的 uVision2 项目文件并按照 58 页的描述从器件数据库中选择一个 CPU。注意很重要的一点是新的 uVision2 项目必须创建在你已经存在的 uVision1 项目文件夹中。

2、单击菜单 **Project － Import uVision1 Project**，在弹出的对话框中选择上述文件夹中一个已经存在的 uVision1 项目。只有新项目的文件列表是空的时，此菜单才是可用的。

3、导入命令同时也把旧连接器的设置导入到连接对话框中。但是，我们推荐你用 uVision2 的 **Options for Target-Target** 对话框选项来定义你目标硬件的存储器结构。一旦你这样做了，你就可以使能 **Options for Target － L51 Locate** 对话框中的 **Use Memory Layout from Target Dialog** 选项，并在此对话框中删除用户类型和用户区域的设置。

4、仔细核对是否所有的设置都正确的复制到了新的 uVision2 项目文件中。

5、现在你可以在新 uVision2 项目中创建文件组，如 64 页“Project Targets and File Groups”叙述的那样。
然后，你就可以把文件拖放到此新文件组中。

---

***注意：***

*由于 uVision2 在许多方面和以前的版本不一样，所以不可能 100%的把 uVision1 项目转换为 uVision2 项目。*

*在你导入 uVision1 项目后，仔细核对工具设置是否正确转换。一些 uVision1 项目设置，如：用户编译器和库模块列表，不能转换到 uVision2 项目中。还有 dScope 调试器设置也不能复制到 uVision2 项目文件中。*

---

*/*********************************译者注-开始*********************************/*

*步骤2 中的描述与我的uVision2 菜单的实际情况不一致：我的uVision2 中的菜单点击后弹出一个提示对话框，内容与本页的描述部分内容相似，问你是否继续？继续后，提示选择新项目和旧项目。而且菜单一直有效。*

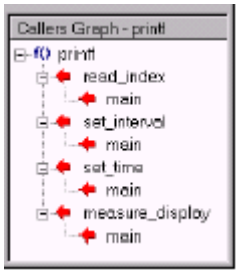
*步骤3 中：我的版本中的L51 Locate 对话框中没有用户类型和用户区域的设置条目，因而无从删除。也许，原文相关部分应翻译为：……并且此对话框中已经删除了用户类型和用户区域的设置。*

*/*********************************译者注-结束*********************************/*

## Build 后运行外部程序

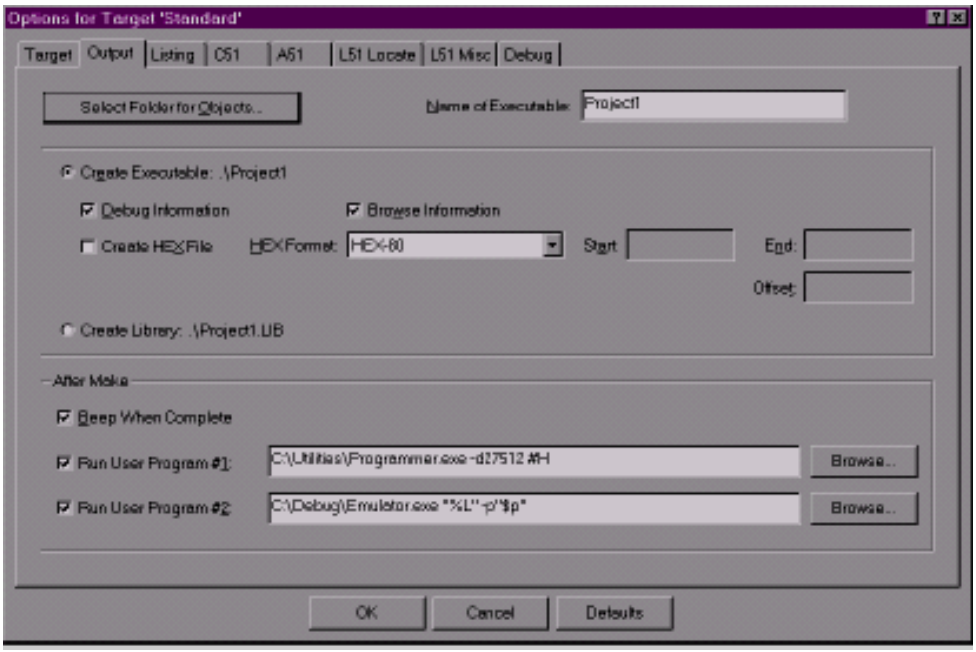
**Options for Target － Output** 对话框允许你输入最多两个用户程序在 Build 成功后开始执行。使用键序列码你可以从 uVision2 项目管理中传递参数到这些用户程序中。

你可以在编辑器窗口中使用浏览器的信息。选择你想要寻找的条目，右击打开弹出式菜单，或使用下面的快捷键：




| 快捷键 | 描述 |
|---|---|
| F12 | 跳转到定义处；将光标置到符号定义处。 |
| Shift+F12 | 跳转到引用处；将光标置到符号引用处。 |
| Ctrl+Num+ | 跳转到下一处的引用或定义处。 |
| Ctrl+Num － | 跳转到前一处的引用或定义处 |

工具参数的键序列参照 71 页的描述。



上例中，**User Program #1**的调用参数是带绝对路径的Hex格式的输出文件（如**C:\MYPROJECT\PROJECT1.HEX**）。**User Program # 2** 的调用参数仅仅是连接器的输出文件**PROJECT1**的名字（不带后缀）和参数**"-p"**表示的指向项目的路径**C:\MYPROJECT**。如果你的文件夹名字中包含如~,#等特别字符，你应该用引号把键序列括起来。

## 为列表文件和目标文件指定单独的文件夹

你可以指定工具的输出文件到不同的文件夹：

- **Options for Target － Output** 对话框让你选择目标文件的文件夹。当你为项目的各个目标的目标文件指定一个独立的文件夹时，uVision2 仍然把在此之前的生成过程产生的目标文件视为有效。即使你修改了你的项目目标，Build Target 命令也只是重新编译那些修改过的文件。
- **Options for Target － Listing** 对话框利用 **Select Folder for List Files** 按钮为列表文件提供同样的功能。

## 使用 uVision2 器件库中没有列出的微控制器

uVision2 器件库包含所有的标准的 8051 产品。然而，也有一些用户定做的器件和未来的器件目前没有包含进来。如果你需要使用这些没有列出的 CPU，你有两种选择：

- 选择 Generic 中列出的一个器件。8051（all Variants）器件允许你配置所有的工具参数，所以支持所有类型的 CPU。在 **Options for Target － Target** 对话框中定义片上存储器和扩展的存储器。
- 输入一个新的 CPU 到 uVision2 器件库中。点击菜单 **File － Device Database**，打开对话框，选择一个和你想添加的器件最相似的 CPU，然后修改参数。在选项框中的 **CPU** 设置定义了基本的工具设置。这些参数如下表所述：

| 参数 | 详细定义 |
|---|---|
| IRAM (range) | 片上 IRAM 的地址空间 |
| XRAM (range) | 片上 XRAM 的地址空间 |
| IROM (range) | 片上 ROM（flash）的地址范围。起始地址必须是 0。 |
| CLOCK (val) | 当你选用此器件时的默认时钟频率。 |
| MODA2 | Atmel 各种器件的双数据指针。 |
| MODDP2 | Dallas 各种器件的双数据指针。 |
| MODDPX | 使能扩展的 24 位的数据指针寄存器（如：ADuC812）。 |

| | |
|---|---|
| MODP2 | Philips and Temic 各种器件的双数据指针。 |
| MOD517DP | Infineon C500 系列器件的多用途数据指针。 |
| MOD517AU | Infineon C500 系列器件的算术逻辑单元。 |
| MOD_CONT | 使能支持 Dallas 390 Contigious 模式。 |
| MX | 使能支持 Philips 80C51MX。 |

其它的选项变量定义 CPU 数据手册和 uVision2 调试 DLL 文件。当增加一个新的器件到数据库中时，保持这些变量不变。

## 创建一个库文件

在 **Options for Target － Output** 对话框中选择创建库文件。uVision2 将调用库管理器而不是连接/定位器。此时，由于库中的代码将不需要连接和定位，所以，在 L51 Locate 和 L51 Misc 选项页的输入将被忽略。同时，在目标页中的 CPU 和储存器的设置也无关紧要。如果你计划将你的代码应用到不同的 8051 器件中，你可以选择器件库中 Generic 下的一个 CPU。

## 复制工具设置到一个新的目标中

当你在 **Project － Targets, Groups, Files…**对话框中增加一个新目标时，使能选项 **Copy all Settings from Current Target**。从一个已经存在的目标复制工具设置到当前的目标的步骤如下：

1、用 **Remove Target** 命令（在 Targets,Groups,Files…对话框中）删除当前的目标。
2、选择你想要复制工具设置的目标为当前目标。
3、使能 **Copy all Settings from Current Target** 后，再把刚刚删除的目标加进来。

## 放置代码到绝对存储器空间

有时，需要把一些代码段放在特定的存储器地址。下例中，名叫 **alarm_control** 的结构体将要放在 0xC000 地址处。这个结构体在文件 ALMCTRL.C 中定义并且这个模块只包含此结构体的声明。

```
struct alarm_st {
    unsigned int alarm_number;
    unsigned char enable flag;
    unsigned int time_delay;
    unsigned char status;
};
struct alarm_st xdata alarm_control;
:
:
```

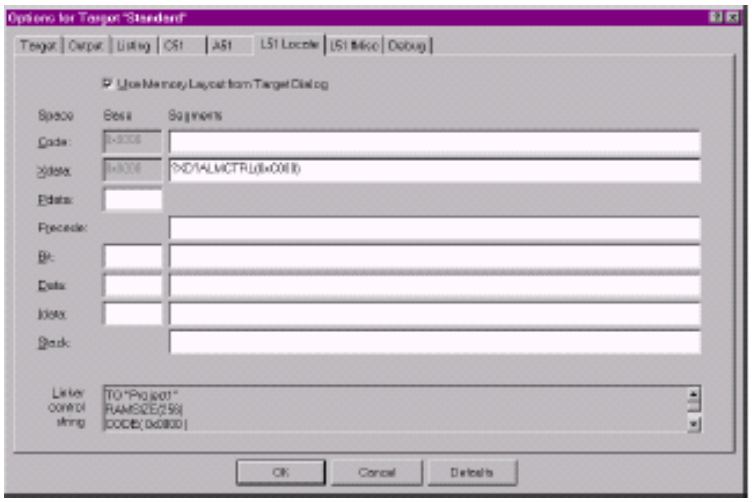
C51 编译器为 **ALMCTRL.C** 产生目标文件，包含一个存储在 **xdata** 存储空间中的变量段。变量 **alarm_control** 是放置在**?XD?ALMCTRL** 段中。在 **Options for Target – L51 Locate** 页中，uVision2 允许你定义任何段的起始地址。下例中，连接/定位器将把名为**?XD?ALMCTRL** 的段放在 xdata 储存空间的从 0xC000 起始的地方。

---

***注意：***

*C51 提供关键词_at_和一些能够存取绝对地址的宏。更详细的信息参见第六章的 "C51 User's Guide"*

---

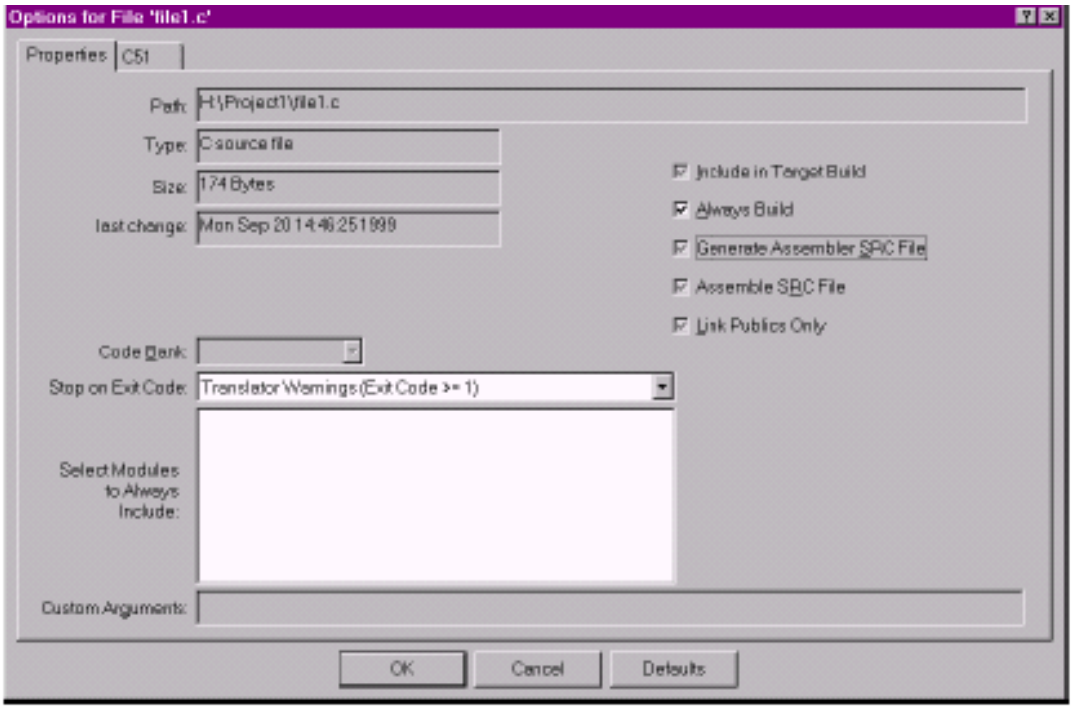


## 文件和文件组的特定选项 - 属性对话框

uVision2 允许你通过 **Project Window － Files** 页的弹出式菜单设置文件和文件组的特定选项，操作如下：选择一个文件或文件组，右击，选择 **Options for …**菜单。然后你就可以浏览或设置刚才所选择的文件或文件组的特定的选项。此对话框中有三态选择。如果一个选项为灰色的（默认值）表明包含上一级文件组或目标的同样的设置。下表描述了属性对话框中的各个选项：

| 对话框条目 | 描述 |
|---|---|
| Path, Type, Size Last Change | 显示所选文件的信息。 |
| Include in Target Build | 禁止此选项将把此文件或文件组排除在目标之外。如果此选项没有选上，<br>uVision2 将不编译和连接此文件或文件组。当你为几个不同的硬件系统使用此相同的项目文件时，这对那些配置文件很有用。 |
| Always Build | 使能此选项后，在每一次生成过程中都将从新编译此源程序模块，而不管是否修改过。这对包含**__DATE__** 和 **__TIME__** 宏的文件很有用，这些宏用来记录应用程序的版本信息。 |
| Generate Assembler SRC File | 通知 C51 编译器为 C 源程序产生汇编语言代码源程序。通常当 C 语言源程序包含**#pragma asm / endasm** 项时，选用此项。 |
| Assemble SRC File | 此选项和 **Generate Assembler SRC File** 选项一起使用，将 C51 产生的汇编语言源程序汇编为目标文件，连接到应用中。 |

| 对话框条目 | 描述 |
|---|---|
| Link Publics Only | 此选项只对 Lx51 有用，它通知连接器只包含此模块中的公共符号。通常在你想使用其它不同的应用入口地址或其中的变量地址时使用此选项。在项目包含一个绝对目标文件时的绝大多数情况下适应。 |
| Stop on Exit Code | 定义一个基于编译信息的生成退出代码。默认时，uVision2 在一次生成过程中编译所有文件，而忽略错误或告警信息。 |
| Select Modules to Always Include | 允许你一直包含库中某些特定的模块。更多信息请参见 89 页的“一直包含特定的库模块”。 |
| Custom Arguments | 当你的项目包含需要其它编译器编译的文件时此行才需要。更详细的内容参见 90 页的“使用用户编译器” |




本例中，我们为文件 FILE1.C 定义的属性为：编译时遇到告警即停止编译过程；无论此文件是否修改，在每一次生成过程中都要编译此文件。

## 编译带 asm / endasm 项的 C 语言模块

如果你在你的 C 语言源程序中包含了汇编语句，C51 编译器要求你先产生一个汇编语言源文件，再编译此汇编源文件。在此情况下，使能属性对话框中的 **Generate Assembler SRC File** 和 **Assembler SRC File** 选项。

---

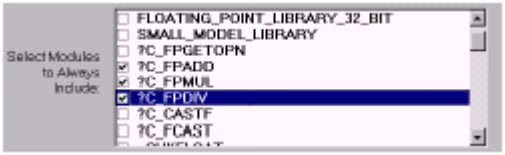
***注意：***

*请核对你是否能够用内联的内部函数代替汇编代码。由于这样做后，你的 C 语言源程序将不容易移植到其它平台上，所以通常情况下避免汇编代码片段比较好。C51 编译器为你提供了几个内联的函数，从而允许你访问所有的特殊外围器件。通常情况下，C 语言源代码不需要插入汇编指令。*

---

## 一直包含特定的库模块

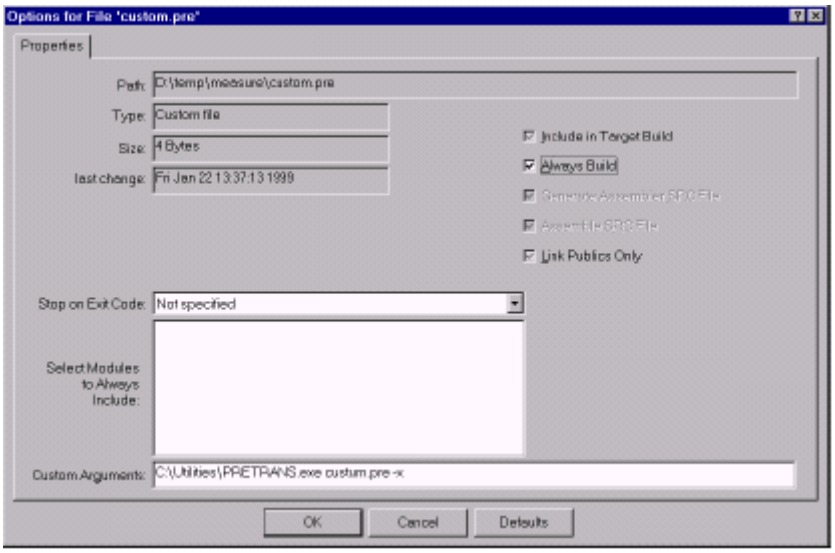
属性对话框允许你定义那些需要一直包含在项目中的库模块。这在你生成一个应用的基本部分时也许用到，此基本部分包含未来加载的程序用到的一些基本子程序。这儿先增加包含需要的目标模块的库文件，在 **Project Window － Files** 页右击此库文件，打开弹出式菜单，点击 **Options for...**，在属性窗口选择那些需要一直包含的模块。




仅仅选择那些个你想在任何情况下都包含在你的目标应用中的模块。

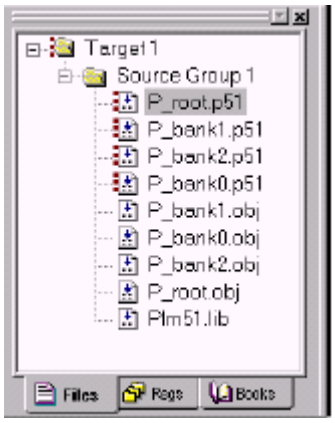
## 使用用户编译器

如果你增加一个带未知文件后缀的文件到你的项目中，uVision2 要求你定义此文件的文件类型。你可以选择用户文件并用一个用户编译器来处理此文件。用户编译器和其命令行参数在 **Options – Properties** 对话框中的 **Custom Arguments** 行一起定义。通常情况下，用户编译器将从用户文件产生一个源文件。你需要增加此源文件到你的项目中并用 A51 或 C51 将此源文件编译成能够连接到你的应用中的目标文件。




在此例中，我们为文件 **CUSTOM.PRE** 定义了用带 **-X** 参数的 **C:\UTILITIES\PRETRANS.EXE** 程序来编译。注意，我们使用了 Always Build 选项来确保每次 Build 时都编译此文件。

## 使用 Intel PL/M-51




如果你还有 Intel PL/M-51 源文件需要包含到你的 uVision2 项目中，你可以以用户文件的形式增加这些源文件到 uVision2 项目树中。另外，你也必须插入由 PL/M51 生成的*.OBJ 文件和 Intel PLM51.LIB。

针对 PL/M-51 编译器的选项在 **Options** － **Properties** 对话框中设置。这个对话框用右击此源文件来打开。


针对 PL/M-51 编译器的选项在 **Options** － **Properties** 对话框的 **Custom Arguments** 行中输入。

## 文件扩展名

**Project － File Extensions** 对话框允许你为一个项目设置默认的文件扩展名。你可以输入几个扩展名，彼此用分号隔开。文件扩展名是与项目相关的。

## 不同的编译器和汇编器设置

通过 **Project Window － Files** 页的弹出式菜单，你可以为一个文件组或一个文件设置不同选项。在对话框页中有三态选择；如果一个选项为灰色，那么高一层的设置将被继承。你可以利用这种方法为整个文件组定义一定的工具设置，并能够为此组中单独的一个源文件该变设置。

## 版本和序列号信息

当你打开菜单 **Help － About** 时，将列出所有工具的详细的信息。在你向我们报告遇到的问题时，请提供这些信息。