8051 演示套件

前言

该手册是 Keil 软件公司 8051 单片机软件开发工具的介绍。是为新用户和有兴趣的读者准备的使用指南。只需要阅读本书就可以正确地运行和使用该软件。这本用户指南包括以下章节的内容：

“第一章　介绍” 对本用户手册的概述。

“第二章　安装” 讲述怎样安装软件并设置工具的操作环境。

“第三章　8051 产品系列”讲述为 8051 单片机提供的不同产品。读完本章可以决定选用哪一种产品。

“第四章　8051 开发工具”讲述 8051 开发工具的主要特征，包括 C 编译器、汇编器、调试器和集成开发环境。

“第五章　使用 8051 工具”讲述通过演示程序指导用户如何使用这套工具。

“第六章　硬件产品”介绍辅助开发和调试的硬件工具。还有 80C517A 和 80C520 评估板以及 EPROM 仿真器。

“第七章　实时内核”讲述 RTX－51 实时操作系统。该章还提供多任务处理系统的概述。

“第八章　命令参考”简要讲述了 8051 开发工具的命令及控制符。

第一章　介绍

Keil 软件公司的 8051 单片机软件开发工具可用于众多的 8051 派生器件以实现嵌入式应用。开发工具清单如下：

- C51 优化 C 编译器
- A51 宏汇编器
- 8051 工具（连接器、目标文件转换器、库管理器）
- Windows 版 dScope 源程序级调试器/模拟器
- Windows 版μVision 集成开发环境

这些工具都集合在一个套件内（见第三章）。独立的工具在第四章详细介绍。除了以上的开发工具以外，还提供实时内核、评估板和调试硬件，这些内容详见第六章和第七章的讲述。这套工具是为专业软件开发者设计的，但任何水平的编程者都可使用。

手册主题

该手册讨论了许多主题，包括：

- 将软件安装到系统并将其调整到最佳性能（见第二章）。
- 为你的应用系统选择最佳工具套件（见第三章）。
- 使用 8051 开发工具（见第四章）。
- 运行演示程序（见第五章）。

如果想要马上开始使用，应当安装软件并运行演示程序。

评估和演示套件

Keil 提供两套软件供用户对开发工具进行评估。

C51 演示套件是开发工具的示范版本。演示套件中的工具并不产生实际的目标代码，而是产生列表文件，可通过列表文件查看编译器和其它工具产生的代码。

C51 评估套件是工具的评估版本。评估套件的工具可产生最大为 2K 的应用程序。可使用该套件评估开发工具的效用，并可用于产生小型的目标应用系统。

第二章 安装

这一章讲述如何设置一个操作环境以及如何将软件安装到硬盘上。在开始安装之前，必须进行以下步骤：

- 确定你的计算机系统符合最低配置要求。
- 将安装盘备份。

系统要求

为了保证编译器和工具的正常工作，系统必须满足软件和硬件的最低配置。

对 Windows 版工具，必须满足以下条件：

- 100％IBM 兼容 386 或以上 PC
- Windows3.1 版或以上
- 至少 4M 内存
- 硬盘至少有 6MB 磁盘空间

对 DOS 版工具，必须满足以下条件：

- 100％IBM 兼容 386 或以上 PC，带 640K 内存
- MS-DOS3.1 版或以上
- 硬盘至少有 6MB 磁盘空间

C 编译器及工具要求 CONFIG.SYS 文件中至少定义 20 个文件及 20 个缓冲区。此外还需要足够的环境空间供编译器和工具的环境变量使用（见“环境设置”章节）。

CONFIG.SYS 文件应当和下列格式相似：

BUFFERS＝20
FILES＝20
SHELL＝C:\COMMAND.COM/e: 1024/p

如果在 DOS 下收到信息“Out of environment space”，就需要通过增加示例中 1024 的值以增加环境空间数量。详情参考 DOS 用户指南。

备份磁盘

强烈建议用户将安装磁盘拷贝到一个备份盘上。然后用备份磁盘来安装软件。将原磁盘妥善保存，以防止备份磁盘丢失或损坏。

安装 DOS 版产品

要安装 DOS 版，将第一个安装盘插入驱动器 A：并在 DOS 提示符下键入以下命令：

A：INSTALL

然后按安装程序的提示继续安装。

安装 Windows 版产品

要安装 Windows 版产品：

- 将第一个安装盘插入驱动器 A：
- 在程序管理器的“文件”菜单中选择“运行”
- 在命令行提示符下键入 A：SETUP
- 选择“OK”按钮

然后按安装程序的提示继续安装。

目录结构

安装程序将开发工具复制到下列根目录中的子目录，所使用的目录由安装的工具套件决定。

目录	描述
\C51	8051 开发工具
\C51EVAL	8051 评估工具

在创建适当的目录后，安装程序将开发工具复制到下列子目录中。

子目录	描述
…\ASM	汇编器包含文件
…\BIN	可执行文件
…\DS51	dScope-51 DOS IOF 驱动器
…\EXAMPLES	示例应用程序
…\RTX51	RTX－51
…\RTX_TINY	RTX－51
…\INC	C 编译器文件
…\LIB	C 编译器库文件和启动代码
…\MON51	目标监控文件
…\TS51	tScope-51 DOS IOT 驱动器

环境设置

下表列出了环境变量、它们的默认路径以及简要的描述。

变量	路径	描述
PATH	\C51\BIN	定义 8051 开发工具的路径。
PATH	\C51EVAL\BIN	定义 8051 评估工具的路径。
TMP		定义产生暂存文件的路径。要得到最佳性能，定义的路径应当是 RAM 磁盘。如果环境变量已经定义，路径必须存在。否则，工具将报告严重出错。
C51INC	\C51\INC	定义标准 C51 编译器包含文件的路径。
C51LIB	\C51\LIB	定义标准 C51 编译器库文件的路径。

通常，环境设置由安装程序自动安装自动安装到 AUTOEXEC.BAT 文件中。如果想将这些设置放入一个单独的批处理文件中，必须写入如下的环境设置：

8051 开发工具	8051 演示工具
PATH=C:\C51\BIN;…	PATH＝C:\C51EVAL\BIN;…
SET C51INC=C:\C51\INC	SET C51INC=C:\C51EVAL\INC
SET C51LIB=C:\C51\LIB	SET C51LIB=C:\C51EVAL\LIB

改善系统性能

可以使用两种方法改善 C51 编译器和工具的性能。这些技术可以帮助改善大多数应用系统的性能：

- 为编译器和工具软件提供一个 RAM 磁盘用于暂存文件。
- 使用磁盘高速缓冲区（disk cache）保存最近访问过的磁盘文件。

使用 RAM 磁盘

如果计算机有足够的扩展或扩充存储器，可以考虑使用 RAM 磁盘。由于 RAM 磁盘的内容保存在 RAM 中，访问的速度将会非常快。如果使用 RAM 磁盘，可以将环境变量 TMP 设成 RAM 磁盘的名称。由于可以使用 RAM 磁盘存放暂存文件，这样可加速许多工具软件的执行。

有许多 RAM 磁盘的软件可用。RAMDRIVE.SYS 和 VDISK.SYS 是 DOS 最常装载的 RAM 磁盘程序名。参考 DOS 手册学习如何安装这些程序。

使用磁盘高速缓冲区

磁盘高速缓冲区利用巨大的存储区来暂存从磁盘读取的信息。当计算机访问磁盘时，先检查所要的信息是否已在高速缓冲区内。如果是，就从高速缓冲区内而不是从磁盘内读取信息。这显然要比从磁盘读取信息快许多。

通常，软件的开发包括编辑－编译－编辑－编译的循环。这种情况下磁盘高速缓冲区提高了编辑器、编译器和连接器的性能。编辑器、编译器、源文件和目标文件全都可以放入高速缓冲区，这样对

磁盘的访问就降到最少。

MS－DOS5.0 和 6.0 都有磁盘高速缓冲区的应用程序，叫做 SMARTDRV.SYS。参考 DOS 使用手册学习如何安装和使用该程序。

第三章 8051 产品系列

Keil 提供工业用的 8051 开发工具。为了帮助你熟悉我们是如何分配工具的，下面介绍工具套件的概念。工具套件是几个应用程序的集合，这些程序用来创建 8051 应用系统。使用汇编器汇编 8051 汇编程序，使用编译器将 C 源代码编译成目标文件，使用连接器创建一个绝对目标文件模块供仿真器使用。

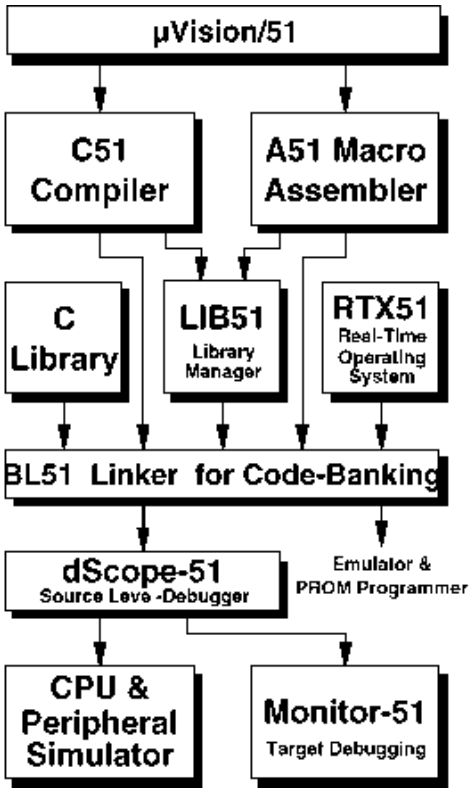
8051 开发工具套件

使用 Keil 的开发工具，其项目开发周期和任何软件开发项目都大致一样。

1. 创建 C 或汇编语言的源程序
2. 编译或汇编源文件
3. 纠正源文件中的错误
4. 从编译器和汇编器连接目标文件
5. 测试连接的应用程序

工具套件概述

上面所述的开发周期用方框图表示最合适。如图所示。



用µVision/51 IDE 创建源文件，然后通过 C51 编译器或 A51 汇编器。编译器或汇编器处理源文件并创建浮动目标文件。目标文件可通过 LIB51 库管理器创建库。库是一个专门格式的、有顺序的目标模块程序集。连接器可对其进行处理。目标文件和库文件通过连接器创建一个绝对目标模块。绝对目标文件或模块是没有浮动代码的目标文件。绝对目标文件中的所有代码都有固定的位置。

由连接器创建的绝对目标文件可用于编程 EPROM 或其它存储器件。绝对目标模块也可和 dScope-51 调试器/模拟器或电路内部仿真器一起使用。

dScope-51 调试器/模拟器对于快速可靠的高级语言程序的调试非常理想。调试程序包括一个高速模拟器和一个目标调试器。可对整个 8051 系统包括片内外围功能进行仿真。通过装载特殊的 I/O 驱动器，可对不同的 8051 派生器件的外围功能进行仿真。和 Monitor-51 相连后，调试程序甚至可以在目标硬件上达到源程序级的仿真。

RTX-51 实时操作系统是一个用于 8051 系列的多任务处理内核程序。RTX-51 实时内核简化了系统的设计、编程以及对时间有严格要求的复杂系统的调试。内核完全集成在 C51 编译器中并且非常易用。任务描述表和操作系统的一致性由 BL51 连接器/定位器自动控制。

工具套件介绍

上图列出了 Keil 8051 开发工具的全部内容。该图中列出的工具包括下面将要介绍的专业开发者套件。除了专业开发者套件外，Keil 还为 8051 开发者提供了其它许多工具。为了更好地说明每套工具的内容，我们按照功能排序。首先介绍功能最强的专业开发者套件。

PK51－C51 专业开发者套件

该套件包括了专业的 8051 开发者创建复杂应用系统所需要的一切工具。该套件的组件如下：

- C51 优化 C 编译器
- A51 宏汇编器
- BL51 代码连接器/定位器
- OC51 目标文件转换器
- OH51 目标－十六进制转换器
- LIB51 库文件管理器
- dScope-51 模拟器/调试器
- tScope-51 目标调试器
- Monitor-51 ROM 监视和终端程序
- 集成开发环境
- RTX-51 Tiny 实时操作系统

另外，专业开发者套件还包括为 Windows 用户提供的下列工具：

- Windows 版 dScope-51 模拟器/调试器
- Windows 版μVision/51 集成开发环境

专业开发者套件可配置用于所有 8051 派生器件。该套件中所有工具需运行在 100％IBM PC386 或以上兼容机的 DOS 环境下。

DK51－C51 开发者套件

DK51－C51 开发者套件是为那些需要在完全 DOS 环境下进行 8051 开发的用户设计的。该套件可使用户在 DOS 开发平台上创建复杂的嵌入式应用系统。该套件包括以下组件：

- C51 优化 C 编译器
- A51 宏汇编器
- BL51 代码连接器/定位器
- OC51 目标文件转换器
- OH51 目标－十六进制转换器
- LIB51 库文件管理器
- dScope-51 模拟器/调试器
- tScope-51 目标调试器
- Monitor-51 ROM 监视和终端程序
- 集成开发环境

开发者套件可配置用于所有 8051 派生器件。该套件中所有工具需运行在 100％IBM PC386 或以上兼容机的 DOS 环境下。

CA51－C51 编译器套件

CA51－C51 编译器套件是需要 C 编译器而不需要调试系统的开发者的最佳选择。该套件可使开发者为目标硬件创建 8051 应用系统。该编译器套件可配置用于所有的 8051 派生器件。该套件中的工具需运行在 100％IBM PC386 或以上兼容机的 DOS 环境下。

A51－A51 宏汇编器套件

A51 宏汇编器套件包括 8051 汇编器和所有创建 8051 应用系统所需的工具。该汇编器套件可配置用于所有的 8051 派生器件。该套件中的工具需运行在 100％IBM PC386 或以上兼容机的 DOS 环境下。

DS51-dScope-51 模拟器套件

DS51 模拟器套件包括与 A51 汇编器套件一起使用的调试器/模拟器和 CA51 编译器套件。由于模拟器可对程序指令进行单步操作，使用该套件可迅速找出 8051 应用系统出现问题的位置。还可以观察程序变量、SFR 和存储器。该套件包括以下组件：

- dScope-51 模拟器/调试器
- tScope-51 目标调试器

- Monitor-51 ROM 监视和终端程序

该模拟器套件可配置用于大多数 8051 派生器件。该套件中的工具需运行在 100％IBM PC386 或以上兼容机的 DOS 环境下。

FR51－RTX－51 Full 实时内核程序

FR51－RTX－51 Full 实时内核程序是一个用于 8051 单片机的实时操作系统。RTX-51 Full 全实时内核提供特征超集以及 BITBUS 和 CAN 通讯协议界面库。具体内容参看“第七章 实时内核程序”。

工具套件比较表

下表为每个开发工具套件的项目清单。通过该表用户可选择最合适的工具套件。

	PK51	DK51	A51
8051	√	√	√
汇编器	√	√	√
编译器	√	√	
模拟器	√	√	
IDE	√	√	√
RTX	√		
Windows	√		
DOS	√	√	√

第四章 8051 开发工具

这一章介绍 8051 单片机家族的特征和优点，以及 Keil 的开发工具。开发工具可帮助开发者迅速并成功的实现设计目标。

8051 单片机家族

8051 问世于二十世纪八十年代早期。由于 8051CPU 内核有着杰出的特性以及外围功能。在本世纪仍然可以得到良好的应用。现今不同的芯片供应商可提供超过 200 种 8051 派生器件。有超过半数的嵌入式项目使用 8051 系列单片机。作为嵌入式处理器，8051 是一枝独秀的。

典型的 8051 家族成员包含 8051CPU 内核、数据存储器、程序存储器和一些外围功能。灵活的存储器界面使用户可以通过标准的外设和存储器件扩展 8051 的性能。

8051 开发工具

Keil5 为 8051 提供下列开发工具：

- C51 优化 C 编译器
- A51 宏汇编器
- BL51 代码连接器/定位器
- OC51 目标文件转换器
- OH51 目标－十六进制转换器
- LIB51 库文件管理器
- Windows 版 dScope-51 模拟器/调试器
- Windows 版μVision/51

C51 优化 C 交叉编译器

C 语言是一种通用编程语言。它提供高效代码、结构化编程元素及丰富的运算符。C 不是一个大型的语言，不是为特定领域内的应用而设计的。C 的普遍性使它可以为各种不同的软件任务提供便利有效的编程方案。许多应用设计使用 C 比其它专门语言更有效。

Keil 的 C51 优化交叉编译器（MS－DOS 版）是完全符合 ANSI（美国国家标准协会）标准的 C 语言工具。C51 编译程序产生 8051 单片机使用的代码。但它不是一个适合 8051 目标硬件的通用 C 编译器。

对于大多数 8051 应用，使用像 C 这样的高级语言比使用汇编程序更具优点。例如：

- 不需要了解处理器的指令集，对 8051 的存储器结构也不必要了解。
- 寄存器分配和寻址方式由编译器进行管理。
- 指定操作的变量选择组合提高了程序的可读性。
- 可使用与人的思维更相近的关键字和操作函数。
- 与使用汇编语言编程相比，程序的开发和调试时间大大缩短。
- 库文件可提供许多标准的例程（例如格式化输出、数据转换和浮点运算）加入到应用程序当中。
- 通过 C 可实现模块化编程技术，从而可将已编制好的程序加入到新程序中。
- C 语言可移植性好且非常普及。C 编译器几乎适用于所有的目标系统。已完成的软件项目可以容易地转换到其它的处理器或环境。

C51 语言扩展

C51 编译器是符合 ANSI 标准的 C 编译器。C 语言的扩展支持 8051 单片机的应用，包括：

- 数据类型
- 存储器类型
- 存储器模型
- 指针
- 再入函数
- 中断函数
- 实时操作系统
- PL/M 和 A51 源文件接口

下面章节将简要介绍这些扩展功能。

数据类型

C51 编译器支持下表列出的数据类型。除了这些标量类型外，还可以将变量组合到结构、联合及阵列中。除了指明的类型，可通过指针访问这些数据类型。

数据类型	位	字节	值的范围
bit[1]	1		0~1
带符号 char	8	1	-128~+127
无符号 char	8	1	0~255
enum	16	2	-32768~+32767
short	16	2	-32768~+32767
short	16	2	0~65535
int	16	2	-32768~+32767
int	16	2	0~65535
long	32	4	-2147483648~+2147483647
long	32	4	0~4294967295
float	32	4	±1.175494E-38~±3.402823E+38
sbit[1]	1		0~1
sfr[1]	8	1	0~255
sfr16[1]	16	2	0~65535

1. bit，sbit，sfrs 和 sfr16 数据类型专门用于 8051 硬件和 C51 编译器。并不是 ANSI C 的一部分，不能通过指针进行访问。

bit，sbit，sfrs 和 sfr16 数据类型用于访问 8051 的特殊功能寄存器。例如，sfr P0 = 0x80 定义变量 P0 并将其分配特殊功能寄存器地址 0x80。在 8051 上是 P0 口的地址。

当结果表示不同的数据类型时，C51 编译器自动转换数据类型。例如，位变量在整数分配中就被转换成一个整数。除了数据类型的转换之外，带符号变量的符号扩展也是自动完成的。

存储器类型

C51 编译器支持 8051 及其派生器件结构并提供对 8051 所有存储区的访问。每个变量可以明确地

分配到指定的存储空间。对内部数据存储器的访问比对外部数据存储器的访问快许多。因此，应当将频繁使用的变量放在内部数据存储器，而把较少使用的变量放在外部数据存储器中。

变量的定义包括了存储器类型的指定。可以指定变量存放的位置。

存储器类型	描述
code	程序存储器（64K 字节）；通过操作码 MOVC @A+DPTR 进行访问
data	直接寻址内部数据存储器；对变量的最快访问（128 字节）
idata	间接寻址内部数据存储器；访问整个内部地址空间（256 字节）
bdata	位寻址内部数据存储器；允许位和字节混合寻址（16 字节）
xdata	外部数据存储器（64K 字节）；通过 MOVX @DPTR 访问
pdata	页外部数据存储器（256 字节）；通过 MOVX @Rn 访问

存储器模型

存储器模型决定用于函数自变量、自动变量和没有明确存储类型的变量的默认存储器类型。在命令行中使用 **SMALL**、**COMPACT** 和 **LARGE** 控制命令指定存储器类型。

SMALL 在该模型中，所有变量都默认位于 8051 内部数据存储器。这和使用 **data** 指定存储器类型的方式一样。此模型对于变量访问的效率很高，但所有的数据对象和堆栈必须适合内部 RAM。堆栈的大小很关键。因为使用的堆栈空间是由不同函数嵌套的深度决定的。通常，如果 BL51 连接器/定位器将变量都配置在内部数据存储器内，SMALL 模型是最佳选择。

COMPACT 使用 COMPACT 模型，所有变量都默认在外部数据存储器的一页内。这和使用 **pdata** 指定存储器类型一样。该存储器类型适用于变量不超过 256 个字节。此限制是由寻址方式所决定的。该存储器模型的效率低于 SMALL 模型。对变量访问的速度要慢一些，但比 LARGE 模型快。地址的高字节通常通过口 2 设置。编译器没有设置该口。

LARGE 在 LARGE 模型中，所有变量都默认位于外部数据存储器。这和使用 **xdata** 指定存储器类型一样。使用数据指针（**DPTR**）进行寻址。通过数据指针访问外部数据存储器的效率较低。特别是当变量为 2 个字节或更多字节时，该模型的数据访问比 SMALL 和 COMPACT 产生更多的代码。

指针

C51 编译器支持使用“*”号说明的指针。可以使用指针执行标准 C 中所有可执行的操作。但由于 8051 及其派生器件的独特结构，C51 支持两种不同类型的指针：存储器特殊指针和普通指针。

普通指针

普通指针的说明和标准 C 指针相同。例如：

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

普通指针总是使用三个字节进行保存。第一个字节用于存储器类型。第二个字节用于保存偏移量的高字节。第三个字节用于保存偏移量的低字节。普通指针可以访问 8051 存储空间任何位置的变量。因此许多库程序使用此类型的指针。使用这种普通隐式指针可访问数据而不用考虑数据在存储器中的位置。

存储器特殊指针

在指针的说明中，存储器特殊指针总是包含存储器类型的指定，并总是指向一个特定的存储器区域。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 */
```

由于存储器类型在编译时指定，因此，无类型指针需要存储器类型字节，而已定义类型指针则不

需要。已定义类型指针可用一个字节（**idata,data,bdata** 和 **pdata** 指针）或两个字节（**code** 和 **xdata** 指针）存储。

比较：存储器特殊&普通指针

用户可通过存储器特殊指针加速 8051C 程序。下面的例子为几个不同指针说明中代码&数据规模和执行时间之间的差异。

描述	idata 指针	Xdata 指针	Generic 指针
示例程序	char idata *ip; char val; val = *ip;	char xdata *xp; char val; val = *xp;	char *p; char val; val = *p;

所产生的 8051 程序代码	MOV R0,ip
	MOV val,@R0
		MOV DPL,xp+1
		MOV DPH,xp
		MOV A,@DPTR
		MOV val ,A
			MOV R1,p+2
			MOV R2,p+1
			MOV R3,p
			CALL CLDPTR

指针大小	1 字节数据	2 字节数据	3 字节数据
代码大小	4 字节代码	9 字节代码	11 字节代码 + Lib.
执行时间	4 个周期	7 个周期	13 个周期

再入函数

再入函数可以同时由几个程序共用。当执行再入函数时，其它程序可以中断执行并开始执行同一个再入函数。通常，C51 函数不能递归调用或用于导致重入的方式。受到该限制是因为函数自变量和局部变量都存放在固定的存储器位置。再入函数属性允许说明那些可以重入的函数，因此可以实现递归调用。例如：

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

再入函数可以递归调用，也可以同时被两个或更多程序调用。它经常用于实时应用或中断代码和非中断代码必须共用一个函数的情况。对于每个再入函数，根据存储器的模型在内部或外部存储器模拟再入堆栈区。

中断函数

当中断发生时，C51 编译器提供一个调用 C 函数的方法。这使用户可以用 C 创建中断服务程序。用户只需要关心中断数和选择的寄存器组。编译器自动产生中断向量和进入及退出代码。中断函数属性，当包含在一个说明中时，指定所关联的函数为中断函数。此外，用户可以指定用于中断的寄存器组。

```
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 编译器在 CPU 寄存器中最多可传递三个函数自变量。由于自变量不必从存储器中读写，因此显著地提高了系统的性能。参数的传递可通过 **REGPARMS** 和 **NOREGPARMS** 控制命令进行控制。

下表列出了用于不同自变量和数据类型的寄存器。

自变量数	char， 1 字节指针	int, 2 字节指针	long, float	普通指针
1	R7	R6&R7	R4-R7	R1-R3
2	R5	R4&R5		
3	R3	R2&R3		

如果没有寄存器可用于参数传递或包含的参数太多，使用固定存储器位置传递超出部分的参数。

函数返回值

CPU 寄存器总是用于函数返回值。下表列出了返回类型和所用的寄存器。

返回类型	寄存器	描述
bit	进位标志	
char,unsigned char,1-byte pointer	R7	
int,unsigned int,2-byte pointer	R6&R7	R6 为高字节，R7 为低字节
long,unsigned long	R4 – R7	R4 为高字节，R7 为低字节
float	R4 – R7	32 位 IEEE 格式
generic pointer	R1 – R3	存储器类型在 R3 中，R2 为高字节，R1 为低字节

寄存器优化

根据程序的前后关系，C51 编译器最多分配 7 个 CPU 寄存器用于寄存器变量。函数执行中的任何寄存器修改都由 C51 编译器在每个模块中标明。连接器/定位器产生一个总体的寄存器文件，其中包含了所有被外部函数改变的寄存器的信息。因此，C51 编译器知道在应用中被每个函数使用的寄存器，并优化每个 C 函数的寄存器分配。

实时操作系统支持

C51 编译器很好地集成了 RTX-51 多任务实时操作系统。在连接过程中产生并控制任务描述表。详细内容参考“第七章 实时内核”。

汇编接口

从 C 可以很容易地访问汇编程序，反之亦然。函数参数通过 CPU 寄存器进行传递，或使用 **NOREGPARMS** 命令时通过固定存储器位置进行传递。函数返回值总是在 CPU 寄存器中传递。用户可以使用 **SRC** 命令指导 C51 编译器产生一个准备用 A51 汇编器汇编的文件，而不是一个目标文件。例如下面的 C 源文件：

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

当使用 SRC 命令编译时，产生下列汇编输出文件。

```
?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
                ADD     A,#01H          ;load LSB of the int
                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
```

可以使用**#pragma asm** 和**#pragma endasm** 预处理程序命令将汇编指令插入到 C 源代码中。

与 PL/M-51 的接口

Intel 的 PL/M-51 是一种广泛使用的编程语言。它在许多方面和 C 相似。用户可以轻松地将 C 程序和 PL/M-51 程序连接起来。

代码优化

C51 编译器是一个主动优化编译器。意思是说编译器采取一定的步骤确定产生的代码和输出的目标文件是高效的代码。编译器分析所产生的代码并使之成为最高效的指令序列。这确保了 C 程序在最小程序空间内实现尽可能高效的运行。

C51 编译器提供 6 种不同级别的优化。高级优化包含低级优化。下面列出了 C51 编译器可执行的所有优化：

- 常量合并：一个表达式或地址计算式内的几个常量合并成一个常量。
- 跳转优化：跳转反演或扩展为最终目标地址，使程序效率得以提高。
- 无用代码消除：将不可能执行的代码（无用码）从程序中删除。
- 寄存器变量：自动变量和函数自变量尽可能放在寄存器中。没有为这些变量保留数据存储器空间。
- 参数通过寄存器传递：通过寄存器最多可传递 3 个函数自变量。
- 全局共用的子表达式消除：将在一个函数中多次出现的子表达式和地址计算式尽可能只计算一次。

8051 特殊优化

- 窥孔优化：当存储器空间或时间可作为结果保存时，用简化操作代替复杂操作。
- 访问优化：在操作中直接计算并包含常量和变量。
- 数据覆盖：数据和位段函数被认为是可覆盖的，并通过 BL51 连接器/定位器用其它数据和位段覆盖。
- Case/Switch 优化：Case 和 Switch 语句，根据它们的数据、序列和位置，可以使用跳转表或跳转串进行优化。

代码产生选项

- **OPTIMIZE（SIZE）**：子程序代替共用的 C 操作。在降低程序运行速度的前提下，减小了程序代码占用的空间。
- **OPTIMIZE（SPEED）**：共用的 C 操作内嵌扩展。增加程序代码的规模换取程序速度的提高。
- **NOAREGS**：C51 编译器不再使用绝对寄存器访问。程序代码独立于寄存器组
- **NOREGPARMS**：参数传递总是在局部数据段内而不是在专门寄存器内进行。使用＃pragma 创建的程序代码和 C51 编译器、PL/M-51 编译器和 ASM-51 汇编器的较早版本兼容。

全局寄存器优化

C51 编译器支持宽范围的寄存器优化。下面的例子为 C51 5.0 版和 C51 3.4 版的比较。由于使用应用寄存器优化，C51 编译器知道由外部函数使用的寄存器不由外部函数改变的寄存器用于寄存器变量。这样所产生的代码占用更少的数据和代码空间并且执行得更快。在下面的例子中，input 和 output 为外部函数，仅需占用几个寄存器。

带全局寄存器优化　　无全局寄存器优化

```
main () {
unsigned char i;
unsigned char a;
```

```
                    while (1) {
                    i = input ();                    /* get number of values */
?C0001:                                              ?C0001:
    LCALL    input                                       LCALL    input
;- assigned to R6-                                       MOV      DPTR,#i
    MOV      R6,AR7                                      MOV      A,R7
                                                         MOV      @DPTR,A
                    do {
                    a = input ();                /* get input value */
?C0005:                                              ?C0005:
    LCALL    input                                       LCALL    input
;- assigned to R7-                                       MOV      DPTR,#a
    MOV      R5,AR7                                      MOV      A,R7
                                                         MOVX     @DPTR,A
                    output (a);                      /* output value */
    LCALL    _output                                     LCALL    _output
                    } while (--i); /* decrement values */
    DJNZ     R6,?C0005                                   MOV      DPTR,#i
                                                         MOVX     A,@DPTR
                                                         DEC      A
                                                         MOVX     @DPTR,A
                                                         JNZ      ?C0005
                    }
    SJMP     ?C0001                                      SJMP     ?C0001
                    }
    RET                                                  RET
    代码规模：18 字节                                    代码规模：30 字节
```

调试

C51 编译器使用 Intel 目标格式（OMF51）并产生完全的符号信息。此外，编译器可以包含所有需要的信息。例如，变量名、函数名和行数等等。这样可以用 dsope-51 或 Intel 兼容的仿真进行详细完全的调试和分析。所有 Intel 兼容仿真器都可用作程序调试。此外，**OBJECTEXTEND** 控制命令将额外的变量类型信息嵌入目标文件。用户必须向仿真器供应商确认是否和 Intel OMF51 目标模块兼容以及是否能接受 Keil 的目标模块。

库文件

C51 编译器包含了 7 个不同的 ANSI 编译库文件。可根据不同的功能要求进行优化。

库文件	描述
C51S.LIB	Small model library without floating-point arithmetic
C51FPS.LIB	Small model floating-point arithmetic library
C51C.LIB	Compact model library without floating-point arithmetic
C51FPC.LIB	Compact model floating-point arithmetic library
C51L.LIB	Large model library without floating-point arithmetic
C51FPL.LIB	Large model floating-point arithmetic library
80C751.LIB	Library for use with the Philips 8xC751 and derivatives.

源代码提供给执行硬件相关 I/O 功能的库模块，并建立在**\C51\LIB** 目录下。用户可使用这些源文件帮助执行目标硬件中 I/O 器件的 I/O 功能。

内部库程序

编译器所带的库包括了许多作为内部函数的程序。非内部函数产生 **ACALL** 或 **LCALL** 指令执行库程序。内部库程序产生嵌入代码（更快且更有效率）执行库程序。

内部函数	描述
crol	字符循环左移
cror	字符循环右移
irol	整数循环左移
iror	整数循环右移
lrol	长整数循环左移
lror	长整数循环右移
nop	空操作（8051 NOP 指令）
testbit	测试并清零位（8051 JBC 指令）

列表文件举例

C51 编译器产生一个列表文件。其中包含源代码、命令信息、汇编列表和符号表。

```
C51 COMPILER V5.02,   SAMPLE          07/01/95    08:00:00    PAGE 1
DOS C51 COMPILER V5.02, COMPILATION OF MODULE SAMPLE
OBJECT MODULE PLACED IN SAMPLE.OBJ
COMPILER INVOKED BY: C:\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
```

The C51 compiler produces a listing file with page numbers as well as time and date of the compilation. Remarks about the compiler invocation and object file output are displayed in this listing.

The listing includes a line number for each statement and a nesting level for each block enclosed within curly braces (`{`and `}`).

Error messages and warning messages are included in the listing file.

The **CODE** compiler option includes an assembly code listing in the listing file. Source line numbers are

```
                                        ; 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)
```

embedded within the generated code.

A memory overview provides information about the 8051 memory areas that are used.

The total number of errors and warnings is stated at the end of the listing file.

A51 宏汇编器

A51 宏汇编器是用于 8051 单片机家族的宏汇编器。它将符号形式的汇编语言转换成可再定位的目标代码。

功能概述

A51 汇编器将一个汇编源文件转换成一个浮动目标模块。如果使用 DUBUG 控制，目标文件包含了供 dScope 或硬件仿真器使用的全部符号信息。除了目标文件之外，A51 汇编器还产生一个列表文件。其中可包含符号表和交叉参考信息。A51 汇编器和 Intel ASM-51 源模块完全兼容。

配置

A51 汇编器支持 8051 家族的所有成员。8051 的特殊功能寄存器是预先定义的，但是 NOMOD51 控制可以使用户通过处理器-特殊包含文件覆盖这些定义。A51 汇编器装载了 8051、8051Fx、8051GB、8052、80152、80451、80452、80515、80C517、80C515A、80C517A、8X552、8XC592、8XCL781、8XCL410 和 80C320 等微控制器的包含文件。用户也可以轻松创建其它 8051 家族成员的包含文件。

列表文件举例

下面的例子显示了由 A51 汇编器所产生的列表文件。该列表文件包含源代码、产生的机器码、命令信息和符号表。

```
A51 MACRO ASSEMBLER    Test Program    07/01/95 08:00:00    PAGE 1
DOS MACRO ASSEMBLER A51 V5.02
OBJECT MODULE PLACED IN SAMPLE.OBJ
ASSEMBLER INVOKED BY: C:\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        25   SJMP Repeat
                 26   ;
----             27   RSEG CONST
0000 54455354    28   TXT: DB 扒EST PROGRAM?00H
0004 2050524F
0008 4752414D
```

The A51 assembler produces a listing file with page numbers as well as the time and date of the assembly. Remarks about the assembler invocation and the object file output are displayed in this listing.

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

The listing file includes a line number for each source line.

If a source line generates code, the HEX values are displayed at the beginning of the line.

Error messages and warning messages are included in the listing file.

```
000C 00                                                          The position of each error
                29                                               is clearly marked.
                30
                31
----            32          RSEG      BITVAR     ; TXTBIT=0 read from CODE
0000            33  TXTBIT: DBIT 1               ; TXTBIT=1 read from XDATA
                34
                35          END
XREF SYMBOL TABLE LISTING                                        The XREF assembler
N A M E         T Y P E V A L U E     ATTRIBUTES / REFERENCES    option produces a cross
BITVAR . . . . . . B SEG 0001H REL=UNIT 9# 32                    reference list. The cross
CONST. . . . . . . C SEG 000DH REL=UNIT 8# 27                    reference report shows all
INITSERIAL . . . . . C ADDR ----- EXT 4# 17                      symbols and the line
PROG . . . . . . . . C SEG 0010H REL=UNIT 7# 15                  numbers in which they are
PUTSTRING. . . . . . C ADDR ----- EXT 4# 23                      used. The line number
PUT_CRLF . . . . . . C ADDR ----- EXT 4# 24                      where the symbol is
REPEAT . . . . . . . C ADDR 0005H R SEG=PROG 22# 25              defined is marked with a
RESET . . . . . . . C ADDR 0000H A 13#                           pound symbol (`#`).
SAMPLE . . . . . . . N NUMB ----- 2
START. . . . . . . . C ADDR 0000H R SEG=PROG 13 17#
TXT. . . . . . . . . C ADDR 0000H R SEG=CONST 22 28#             The register banks used,
TXTBIT . . . . . . . B ADDR 0000H.0 R SEG=BITVAR 5 5 21 33#      and the total number of
                                                                 warnings and errors is
REGISTER BANK(S) USED: 0                                         stated at the end of the
ASSEMBLY COMPLETE.   0 WARNING(S),   0 ERROR(S)                  listing file.
```

BL51 代码连接/定位器

BL51 代码连接/定位器将一个或多个目标模块组合成一个可执行的 8051 程序。连接器还解析外部和其它共用的引用，并将绝对地址分配给浮动的程序段。

BL51 代码连接/定位器处理由 Keil C51 编译器和 A51 汇编器以及 Intel PL/M-51 编译器和 ASM-51 汇编器所创建的目标模块。BL51 自动选择所需要的合适的库和连接。

数据地址管理

BL51 连接器通过重叠不相关的函数变量管理 8051 有限的内部存储器。对大多数 8051 应用系统来说，极大地降低了所需要的存储空间。

BL51 分析函数间的引用并实现存储器的重叠。可以使用 OVERLAY 命令人为地控制函数引用。使用 NOOVERLAY 可以完全禁止存储器重叠。当使用间接调用函数或调试时，使用这些命令禁止重叠是很有效的。

代码排序

BL51 支持创建大于 64K 的应用程序。由于 8051 不直接支持超过 64K 字节的代码地址空间，必须外部硬件交换代码区。该硬件必须由 8051 软件进行控制。该处理称作代码空间切换。

BL51 可让用户管理一个公共区域和 32 个存储区（每个区最多可达 64K 字节）总共高达 2M 字节的存储空间。支持外部硬件的软件包含一个短的汇编文件，用户可在专门的硬件平台上编辑。

BL51 使用户可在指定的区域放置特定的程序模块。谨慎地使用不同区域的分组函数可使用户创建出一个大而有效的应用程序。

公共区域

存储区切换程序中的公共区域是一个所有存储区随时可访问的区域。公共区域在物理上不可交换或移动。公共区域的代码复制在每个存储区或位于一个单独的 EPROM 区（如果公共区域没有交换）。

公共区域包含可随时访问的程序段和常量。还可包含频繁使用的代码。默认情况下，下列为自动位于公共区域的代码段：

复位和中断向量

- 代码常量
- C51 中断函数
- 存储区切换跳转表
- 一些 C51 实时库函数

其它存储区的执行功能

代码存储区由额外的软件控制地址线进行选择。这些地址线是由 8051I/O 口线或存储器映射锁存器进行模拟。BL51 在其它代码存储区中产生一个函数跳转表。当调用不同的存储区的函数时，程序对存储区进行切换，跳转到所需要的函数，并在调用完成后回到原来的存储区。

存储区切换处理大约需要 50 个 CPU 周期和额外的两个字节堆栈空间。通过同一存储区中互相关联的函数可提高系统的性能。多个存储区频繁调用的函数应当放置在公共区域。

列表文件举例

下面的例子为由 BL51 创建的映射文件：

```
BL51 BANKED LINKER/LOCATER     V3.52        07/01/95 08:00:00  PAGE 1
MS-DOS BL51 BANKED LINKER/LOCATER V3.52,      INVOKED BY:
C:\C51\BIN\BL51.EXE SAMPLE.OBJ
MEMORY MODEL: SMALL
```

The BL51 code banking inker/locator produces a map file with the time and date of the link/locate run.

```
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)
```

The invocation line and the selected memory module are listed.

```
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
```

The link-map contains a table of the memory usage of the physical 8051 memory area.

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

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

Error messages and warnings are listed at the end of the map file. These messages indicate possible problems during the link/locate run.

OC51 目标文件转换器

OC51 目标文件转换器为在目标模块中的每个代码区创建绝对目标模块。当用户创建一个存储区切换应用时，BL51 产生存储目标模块。符号调试信息复制到绝对目标文件中并可供 dScope 或仿真器使用。

用户可使用 OC51 目标文件转换器创建绝对目标模块，然后可使用 OH51 目标-HEX 转换器创建 Intel Hex 文件。

OH51 目标 HEX 转换器从绝对目标模块中创建 Intel Hex 文件。而绝对目标模块可由 BL51 或 OC51 创建。Intel Hex 文件是 ASCII 文件，包含了应用程序的 16 进制表达式。将它们装入器件编程器就可写入 EPROM。

LIB51 库管理器

LIB51 库管理器可使用户创建和保存库文件。一个库文件是一个或多个目标文件的格式化集合。库文件提供简便的方法可组合和引用大量的目标文件。BL51 可有效地使用库文件。库管理器可使用户创建一个库文件，将目标模块加入库文件，从库文件中移去目标模块和列出库文件的内容。库管理器可交互或从命令行进行控制。

dScope-51 Windows 版

dScope-51 是一个源级的调试器和模拟器。可调试/模拟由 Keil C51 编译器和 A51 汇编器以及 PL/M-51 编译器和 ASM-51 汇编器所创建的程序。dScope-51 是一个纯软件产品。它可使用户在没有目标硬件的情况下模拟 8051 的特性。用户可在硬件准备之前用 dScope-51 调试自己的嵌入式应用程序。dScope-51 可模拟许多 8051 的外围功能，包括内部串行口、外部 I/O 口和定时器。

如何使用 dScope-51 请参考“dScope 模拟/调试器概述”章节。

µVision/51 Windows 版

µVision/51 是一个集成的软件开发平台。包括全功能编辑器、项目管理器、程序生成工具和环境控制。当用户使用µVision/51 时，就不再需要学习任何一个工具的命令语句。µVision/51 提供以下特性可加速用户的嵌入式应用开发：

- 标准 Windows 用户界面
- 所有环境的对话框和开发工具设置
- 多文件编辑能力
- 用户可自定义密码序列的全功能编辑器
- 将外部程序加入下拉式菜单的应用管理器
- 创建和保存项目的项目管理器
- 从项目中建立目标程序的集成程序生成工具
- 在线帮助系统

如何使用µVision/51 参考“µVision IDE 概述”章节。

第五章 使用 8051 工具

为了使用户方便地评估和熟悉 8051 产品系列。Keil 提供一个评估磁盘，装有一个演示程序和工具的限制版。演示程序也包含在标准的产品套件中。

该章介绍了µVision 和 dScope，并演示如何使用它们进行编译、连接和运行所给的演示程序。该章包括以下内容：

- 打开µVision 和 dScope
- µVision 集成开发环境概述
- dScope 模拟/调试器概述
- 演示程序
- 建立和运行演示程序 HELLO
- 建立和运行演示程序 MEASURE
- 建立和运行演示程序 BADCODE

本章中的演示程序和描述使用 Windows 版工具作为图解。C51 演示套件和 C51 评估套件所分配的工具相同。如果想得到 DOS 版的评估套件请和销售商联系。

注：C51 评估套件包括 8051 工具的评估版本。评估工具在功能和应用程序的代码规模上受到限制。请参考“评估套件说明”。如果要创建大型的应用程序则需要购买开发工具套件。参考“第三章 8051 产品系列”。

打开µVision 和 dScope

µVision 和 dScope 都是标准的 Windows 版应用程序。通过双击由安装程序创建的程序组中对应的图标就可运行它们。

µVision IDE 概述

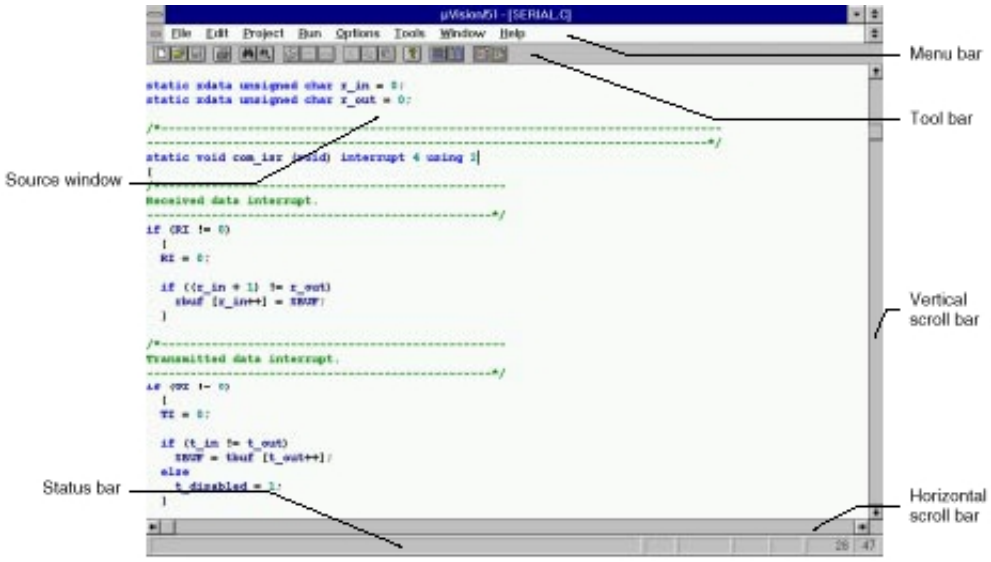
µVision 是一个集成软件开发平台，其中包含了编辑器、项目管理器和程序生成器。µVision 支持所有的 Keil 8051,251 和 166 工具。µVision 提供以下特性帮助用户加快嵌入式应用的开发过程：

- 用户可定义密码序列的全功能编辑器
- 将外部程序加入下拉式菜单的应用管理器
- 创建和保存项目的项目管理器
- 汇编、编译和连接应用程序的集成程序生成工具
- 所有环境的对话框和开发工具设置

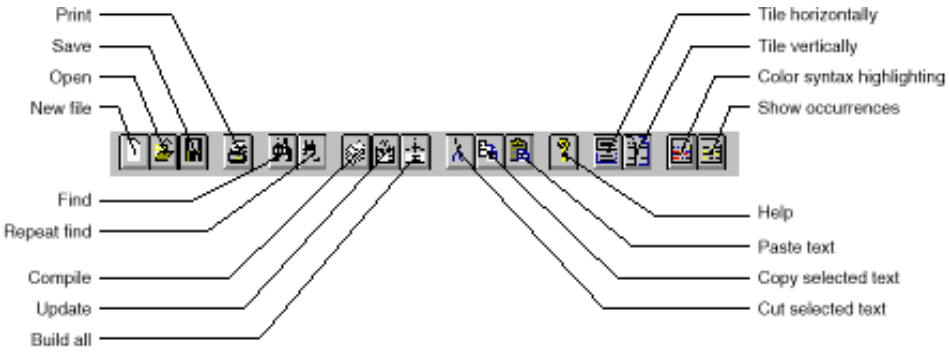
关于环境

在µVision 中，用户可通过键盘或鼠标选择开发工具的菜单命令、设置和选项。也可使用键盘输入程序文本。

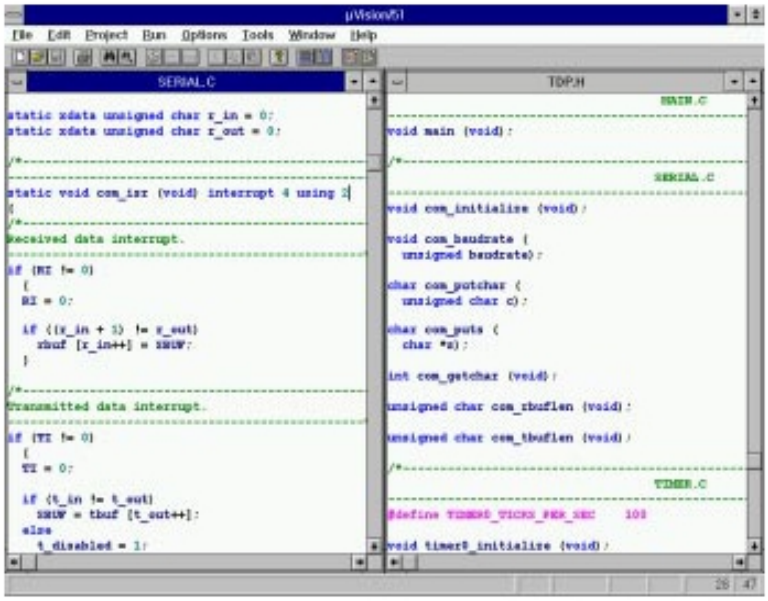
µVision 屏幕提供一个用于命令输入的菜单条，一个可迅速选择命令按钮的工具条和一个或多个源程序窗口、对话框及显示信息。



使用工具条上的按钮可快速执行µVision 的许多功能。



µVision 可同时打开和查看多个源文件。当在一个窗口写程序时可以参考另一个窗口的头文件信息。通过鼠标或键盘可移动或调整窗口大小。



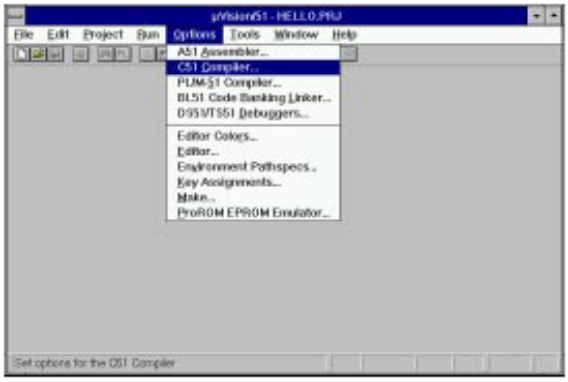
编辑器

µVision 自带的编辑器可定制成和许多流行的文本编辑器相似。对几乎所有编辑器，用户都可改变键配置。下面列出了一些可实现的编辑器功能：

Beginning of File	Destructive Backspace	Next Error
Beginning of Line	End of File	Open File
Beginning of Page	End of Line	Page Down
Cascade Windows	End of Page	Page Up
Close File	Exclusive Mark	Paste from Clipboard
Copy to Clipboard	Forward Quick Search	Previous Error
Cursor Down	Forward Replace	Previous Window
Cursor Left	Full Search	Print File
Cursor Right	Insert Template	Repeat Last Search
Cursor Up	Mark Block	Reverse Quick Search
Cut to Clipboard	Mark Columns	Undo
Delete	Mark Lines	Word Left
Delete Line	Move/Resize Window	Word Right
Delete to End of Line	New File	

菜单命令

可以通过菜单条上的下拉菜单和编辑器命令控制µVision 的操作。可使用鼠标或键盘选取菜单条上的命令。



菜单条提供文件操作、编辑器操作、项目保存、外部程序执行、开发工具选项设置、窗口选择及操作和在线帮助等功能。

开发工具选项

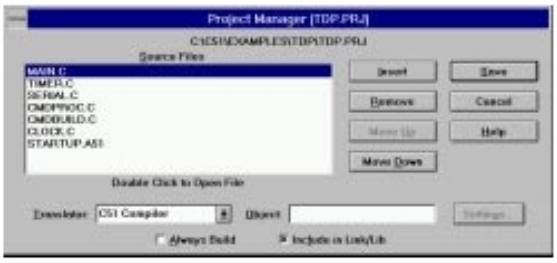
µVision 可设置 C51 编译器和 A51 汇编器等软件开发工具的选项。使用鼠标或键盘可选择相应的项目或更改选项设置。

项目管理器

大多数嵌入式程序都包含了数个源文件。这说明一个项目是包含了许多独立文件的集合。一些文件需要用 C51 编译器进行编译。一些文件需要汇编，还有一些需要自定义转换以创建一个目标程序。

为了适应复杂的项目管理，µVision 带有项目管理功能。项目管理器可使用户创建和保存一个项目。这样可使目标程序得到随时更新。项目管理器可处理文件—文件相关性、包含文件嵌套。和建立目标的实际时序操作相同。

使用项目管理器对话框定义组成项目的源文件，使用项目菜单中的命令编译源文件和产生目标，然后使用模拟和仿真器命令执行、测试和调试应用程序。

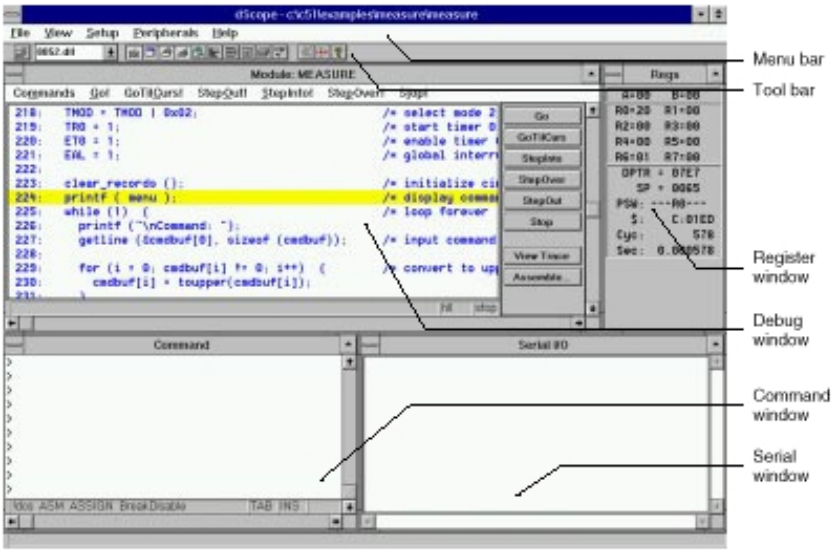


项目的所有特征都保存在项目文件中。项目文件包括：组成目标程序的源文件；编译器、汇编器和连接器命令行选项；调试和模拟器选项；制造工具选项

dScope 模拟/调试器概述

dScope 是一个用于 Keil 软件产品系列的源级调试/模拟器。用户可使用 dScope 调试应用程序。此外，dScope 还可调试 Intel PL/M-51 编译器和 ASM-51 汇编器生成的应用程序。

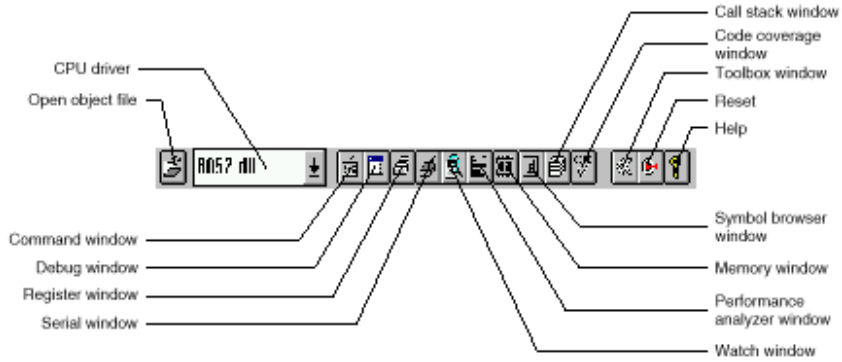
dScope 是一个纯粹的软件产品。可模拟大多数 8051 微控制器的特性而不需要目标硬件。用户可在硬件准备好之前使用 dScope 测试和调试嵌入式应用系统。dScope 可模拟许多 8051 的外围功能，包括内部串行口、外部 I/O 口和定时器。通过使用动态连接库（DLLs）可支持不同的 8051 派生器件。除了模拟 CPU 外，dScope 还具有 8051 监控程序 MON51 接口界面。



关于调试器

在 dScope 中，用户可使用键盘或鼠标选择菜单命令和调试选项，上面所示的 dScope 屏幕还提供

一个用于命令输入的菜单条、一个可快速选择命令按钮的工具条和几个显示寄存器、存储器内容、串行口和命令的窗口。用户可使用工具条上的按钮迅速显示或隐藏窗口。



CPU 模拟

dScope 模拟虚拟的 8051 微控制器。可通过使用 DLLs 支持不同的 CPU。在装入目标程序之前，必须从工具条上的 CPU 驱动器对话框中选择合适的 CPU 驱动器。也可从文件菜单中选择装入 CPU 驱动器命令。下面是 dScope 中包含的 CPU 驱动器：

CPU 驱动器 DLLs	所支持的器件
80320.DLL	Dallas Semiconductor 80C320, 80C520, and 80C530.
8051.DLL	8051, 8031, 80C51, and 80C51
80515.DLL	80C515 and 80C535
80515A.DLL	80C515A and 80C535A
80517.DLL	80C517 and 80C537
80517A.DLL	80C517A and 80C537A
8051FX.DLL	8051FA, 8051FB, and 8051FC
8052.DLL	8052, 8032, 80C52, and 80C32
80552.DLL	8xC552
80751.DLL	8xC750, 8xC751, and 8xC752
80410.DLL	8xCL410
80781.DLL	8xCL781

dScope 最多可模拟 16M 字节的存储器。这些存储器可读、写或代码执行访问。dScope 将捕获并报告非法的存储器访问。

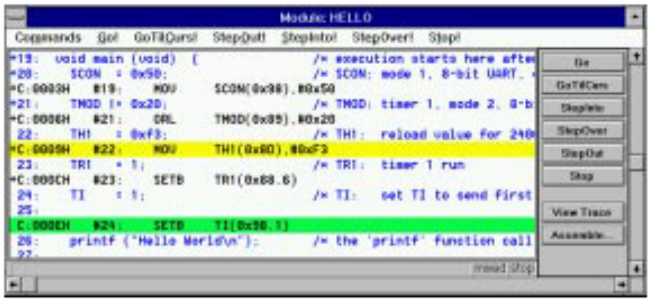
除了存储器映象之外，dScope 还提供集成外围功能的支持。CPU 的片内外围功能由 DLL 中的 CPU 驱动器支持。

用户可通过外围功能菜单选择和显示片内外围元件使用外围功能。还可使用对话框中的控制改变每个外围功能的特性。

调试窗口

在装入合适的 CPU 驱动器之后，用户准备装入目标程序。可使用工具条上的按钮打开目标文件或通过文件菜单使用装入目标文件命令。

当应用程序装入之后，dScope 调试窗口就显示出用户的 C、汇编或 PL/M-51 源程序文本内容。



在调试窗口中可使用命令菜单提供的三种显示格式：

View High Level.. 该格式显示和和源文件一样的源文件文本。

View Mixed. 该格式显示初始源文件文本加上编译器或汇编器产生的汇编代码。

View Assembly. 该格式只显示源文件所产生的汇编代码。

除了目标程序外，调试窗口还可显示最多 512 条先前执行命令的历史记录。要使能历史记录，选择调试窗口中命令菜单上的记录命令。

命令窗口

用键盘和鼠标选择选项来输入命令控制 dScope。在窗口中可输入几乎所有的 dScope 命令。在命令提示符‘>’后输入命令即可。

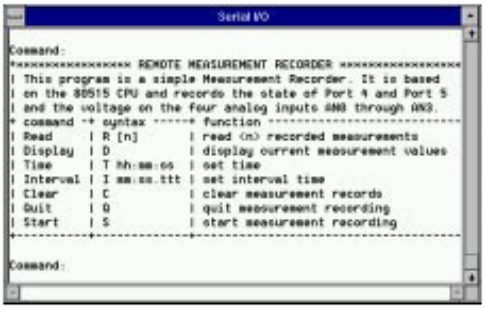
从命令窗口还可显示和改变变量、寄存器和存储器位置。还可输入汇编代码补充或测试部分程序。例如，可在命令提示符后输入下列文本命令。

文本	作用
DPTR	显示 DPTR 寄存器
R7 = 12	将值 12 分配给寄存器 R7
time.hour	显示时间结构的小时部分
time.hour++	增加时间结构的小时部分
index = 0	将值 0 分配给变址寄存器

使用命令窗口控制 dScope 是不受限制的。也可用鼠标从从菜单条中选择下拉式菜单和从工具条调用命令。

串行窗口

dScope 提供一个串行窗口用作串行输入和输出。从模拟 CPU 输出的串行数据显示在该窗口中。在该窗口中录入的字符输入到模拟 CPU。



这使用户不需要外部硬件就可模拟 CPU 的 UART 功能。

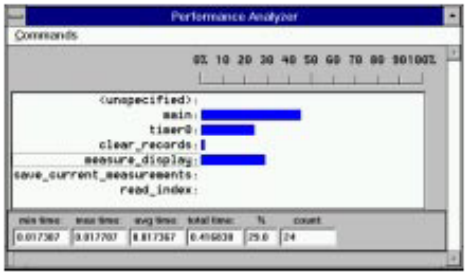
观察窗口

观察窗口显示变量和复杂的结构。当用户想要看到缓冲器上的结果或数据结构时，该功能是非常有用的。

不仅可观察程序的变量，还可使用在命令窗口中的命令提示符下输入的标准 C 表达式改变它们。

性能分析窗口

dScope 有一个内建的性能分析器可记录函数和程序块的统计时间。性能分析结果在窗口中显示。



性能分析器窗口显示每个函数的名称或每个模块的存储范围，还带有一个条状图显示在函数或模块上占用的时间比率。用户可选择一个函数观察在窗口底部的统计数据。下面所示为每个函数或程序块的统计数据：

- **min time** 函数或模块占用的最小时间
- **max time** 函数或模块占用的最大时间
- **avg time** 函数或模块占用的平均时间
- **total time** 函数或模块占用的总时间
- **count** 函数或模块进入的次数

其它特性

除了上述特征以外，dScope 还提供许多其它功能支持调试环境。

函数

dScope 一个强大的功能是可让用户定义和使用类 C 函数以作不同方面的用途。例如，可创建 dScope 函数操作片内外围功能，dScope 的扩展命令集以及产生输入到硬件口的数字和模拟信号。dScope 可实现三种类型的函数：

- 用户函数 扩展调试器的命令范围
- 信号函数 产生输入到 8051 外围功能的信号
- 内建函数 提供可用于用户或信号函数的便利的应用程序（例如 **printf** 和 **memset**）

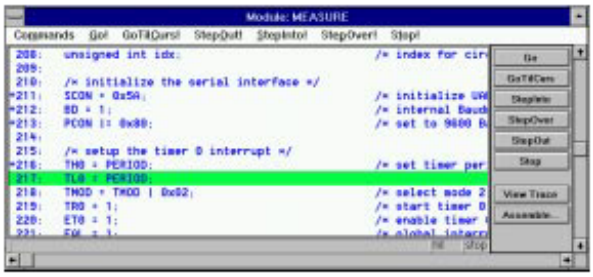
参考“信号函数”一节了解如何使用 dScope 中的函数。

断点

在高级语句、汇编器指令和条件表达式中可方便地设置断点。将鼠标指针移到所要设置断点的行或指令双击即可。当 dScope 到达一个断点时，可自动执行宽范围的操作探针以运行宏函数。

代码覆盖

dScope 提供将已执行的代码行进行标注的代码覆盖功能。在调试窗口中，已执行的代码行左边标注‘+’号。



当用户测试嵌入式应用程序以确定还没有验证过的代码段时，可使用该功能。代码覆盖对话框还提供有用的信息和代码覆盖统计。

演示程序

该节讲述了包含在评估套件和产品套件中的演示程序。用户可应用演示程序学习如何使用开发工具。此外，用户还可将演示程序复制用于自己的应用程序。

演示程序位于\C51\EXAMPLES\目录下。每个演示程序和项目文件以及批处理文件存放在一个单独的子目录下，这样可帮助用户迅速建立和评估每个演示程序。

下表为演示程序清单将其子目录：

目录	描述
\A51\	**A51** 是一个用于 A51 汇编器的演示程序。
\BADCODE\	**BADCODE** 是一个带有一些语法错误的演示程序。使用µVision 打开 BADCODE.PRJ 项目文件并进行编译。µVision 将指出 BADCODE.C 中的每个错误。详见“BADCODE：一个带语法错误的例子”。
\BL51_EX1\	**BL51_EX1** 是一个用 C 写成的存储区切换程序。该演示程序调用不同代码存储区的函数。使用 BL51_EX1.PRJ 项目文件建立该程序。
\BL51_EX2\	**BL51_EX2** 是一个保存不同代码区共用信息的 C 程序。使用 BL51_EX2.PRJ 项目文件建立该程序。
\BL51_EX3\	**BL51_EX3** 是一个代码区切换程序。该程序在不同的代码区只有一个带函数的模块。使用 BL51_EX3.PRJ 项目文件建立该程序。
\BL51_EX4\	**BL51_EX4** 是一个调用不同区内函数的代码区切换的 Intel PL/M-51 程序。该程序为 PL/M-51 格式，作用和 BL51_EX1 相同。使用 BL51_EX3.PRJ 项目文件建立该程序。需要使用 Intel PL/M-51 编译器。
\CSAMPLE\	**CSAMPLE** 是一个简单加减法计数器的演示程序。该程序是一个多模块项目，可使用 CSAMPLE.PRJ 项目文件建立该程序。
\DHRY\	**DHRY** 是一个 DHRYSTONE 基准程序，计算并在主 CPU 每秒显示一次 dhrystone 数。该程序主要提供给基准爱好者。使用 DHRY.PRJ 项目文件建立该程序。
\FIB\	**FIB** 程序产生 fibonacci 数并指导用户如何使用再入函数属性说明递归函数。使用 FIB.PRJ 项目文件建立该程序。
\HELLO\	**HELLO** 是一个嵌入式 8051 C 程序。使用 HELLO.PRJ 项目文件建立该程序。详见有关章节。
\LSIEVE\	**LSIEVE** 演示过滤 Eratosthenes prime 数产生器的大模型方案。该程序主要提供给基准爱好者。使用 LSIEVE.PRJ 项目文件建立该程序。
\MEASURE\	**MEASURE** 是一个采集模拟和数字数据的 C 程序。它模拟一个可能建立在气象站或过程控制应用中的数据采集系统。使用 MEASURE.PRJ 项目文件建立该程序。详见有关章节。
\RTX_EX1\	**RTX_EX1** 演示使用 RTX-51 进行一系列多任务处理。使用 RTX_EX1.PRJ 项目文件建立该程序。
\RTX_EX2\	**RTX_EX2** 演示使用信号的 RTX-51 应用。使用 RTX_EX2.PRJ 项目文件建立该程序。
\SAMPL517\	**SAMPL517** 演示程序提供一个利用 80C517 算术处理器的 RPN-1 型计算器。使用 SAMPL517 项目文件建立该程序。

目录	描述
\SSIEVE\	**SSIEVE** 演示过滤 Eratosthenes prime 数产生器的小模型方案。该程序主要提供给基准爱好者。使用 SSIEVE.PRJ 项目文件建立该程序。
\TDP\	**TDP** 程序演示如何将中断驱动串行 I/O 口和由中断驱动定时器驱动的报警时钟相连接。使用 TDP.PRJ 项目文件建立该程序。
\TRAFFIC\	TRAFFIC 演示如何使用 RTX-51 实时执行程序控制交通灯。使用 TRAFFIC.PRJ 项目文件建立该程序。
\WHETS\	**WHETS** 是一个 WHETSTONE 基准程序，计算并在主 CPU 每秒显示一次 dhrystone 数。使用 WHETSTONE.PRJ 项目文件建立该程序。

要开始使用其中某个演示程序，必须进入到每个例子所在的目录。然后可使用所提供的 DOS 批处理文件或μVision Windows 版项目文件建立和测试演示程序。

下面的章节讲述如何使用工具建立下列演示程序：

- HELLO：你的第一个 C51 程序
- MEASURE：一个远程测量系统
- BADCODE：一个带语法错误的例子

HELL0：你的第一个 8051 C 程序

演示程序 HELLO 位于\C51\EXAMPLES\HELL0\子目录下。HELLO 实现的功能仅仅是从串行口输出显示字符“Hello World”。整个程序包含在单个源文件 HELLO.C 中。如下所示：

```
/*-----------------------------------------------------------------------------------------------------------------
HELLO.C
Copyright 1995 KEIL Software, Inc.
-----------------------------------------------------------------------------------------------------------------*/
#pragma DEBUG OBJECTEXTEND CODE      /* pragma lines can contain                  */
                                     /* command line directives                   */
#include <reg51.h>                   /* special function register declarations    */
                                     /* for the intended 8051 derivative          */
#include <stdio.h>                   /* prototype declarations for I/O functions  */
/*******************/
/* main program */
/*******************/
void main (void)      {              /* execution starts here after stack init        */
SCON = 0x50;                         /* SCON: mode 1, 8-bit UART, enable rcvr         */
TMOD |= 0x20;                        /* TMOD: timer 1, mode 2, 8-bit reload           */
TH1  = 0xf3;                         /* TH1:   reload value for 2400 baud             */
TR1  = 1;                            /* TR1:   timer 1 run                            */
TI     = 1;                          /* TI:    set   TI to send first char of   UART  */
printf ("Hello World\n");            /* the `printf` function call                    */
while (1) {                          /* An embedded program does not stop and         */
       ; /* . . . */                 /* never returns. We`ve used an endless          */
   }                                 /* loop. You may wish to put in your own         */
}                                    /* code were we`ve printed the dots (. . .).     */
```

这个小的应用可帮助用户确定使用编译、连接及调试一个程序。这些操作可以通过批处理文件在

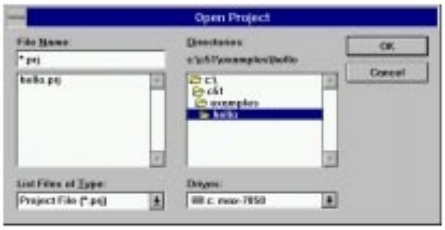
DOS 命令行下执行或通过µVision Windows 版使用给定的项目文件。

硬件要求

用于 HELLO 的硬件基于标准 8051 CPU。片内外围功能仅使用了串行口。由于 dScope 可以模拟该程序所需要的硬件，因此用户不需要目标 CPU。

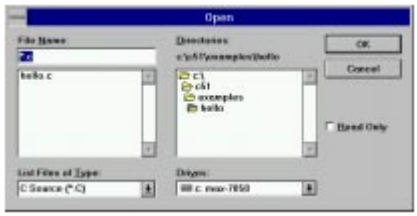
HELLO 项目文件

在µVision 中，应用都保存在项目文件中。项目文件包括了所有与项目有关的源文件名。并告知工具如何编译、汇编和连接以产生一个可执行的目标程序。一个名为 HELLO.PRJ 的文件已创建出来用于 HELLO。要装入该项目文件，选择项目菜单的打开命令并从\C51\EXAMPLES\HELLO 目录下选择 HELLO.PRJ 打开。

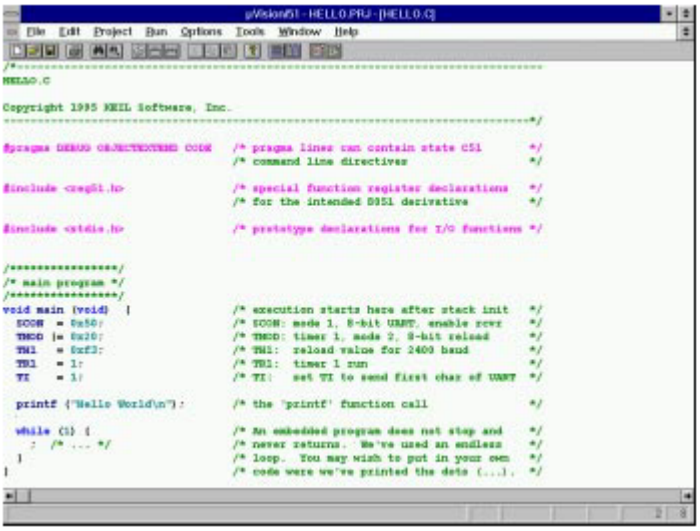


编辑 HELLO.C

要编辑 HELLO.C，选择文件菜单中的打开命令。µVision 出现一个打开文件对话框，从文件清单中选择 HELLO.C 并按下 OK 按钮。



µVision 在一个窗口装入并显示 HELLO.C 的内容。



编译和连接 HELLO

当用户准备编译和连接项目时，单击工具条上的 Build All 按钮，或者选择项目菜单中建立项目命令。µVision 开始编译和连接项目中的源文件并创建一个绝对目标模块。可将其装入 dScope 进行测试。在建立过程中，µVision 在窗口中显示状态。

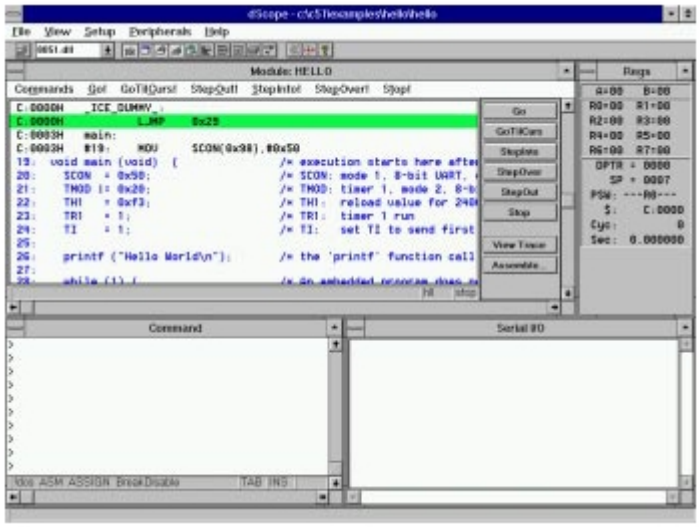
当建立完成时，µVision 显示建立完成信息。

用户可随时按 **ESC** 键终止建立。

使用 dScope 测试 HELLO

当 HELLO 程序编译和连接之后，用户可使用 dScope 调试/模拟器进行测试。在µVision 中，从 Run 菜单中选择 DS51 模拟器命令并当 dScope 命令参数对话框显示时按 **Enter**。

µVision 将初始化文件（HELLO.INI）传递给 dScope。该文件包含用于装入 CPU 驱动器 DLL 和 HELLO 演示程序的 dScope 命令。当 dScope 装入时，下面的屏幕显示：



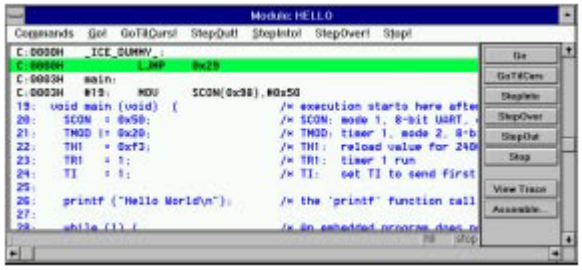
运行 HELLO

要运行 HELLO 程序，在调试窗口中单击 Go 按钮或在命令提示符下输入 g。HELLO 程序执行并在串行窗口显示字符“Hello World”。

在 HELLO 输出“Hello World”之后，程序就进入死循环。若要终止执行，单击调试窗口中的 Stop 按钮或键入 **Ctrl+C**。在终止程序后可键入 exit 退出 dScope 调试器。

单步运行 HELLO

使用调试窗口中的 Step 按钮可单步运行 HELLO 程序。



首先，确认复位 CPU 驱动器。终止程序模拟运行，然后在命令提示符下输入以下命令：

reset

g,main

reset 命令复位模拟 8051 CPU。**g,main** 命令开始执行程序并当它到达主 C 函数时停止。要单步执行程序，单击调试窗口中的 Step Over 按钮。每按一次执行一条语句。当前执行的指令总是处于高亮状态。它随着指令的执行而向下移动。如果一直按着 Step Over 按钮，程序就一直运行下去。

用户可随时退出 dScope，终止 HELLO 的执行并在命令提示符下输入 exit 即可退出 dScope。

MEASURE：一个远程测量系统

MEASURE 演示程序位于\C51\EXAMPLES\MEASURE\子目录下。MEASURE 运行一个收集模拟及数字信号的远程测量系统。例如气象站和过程控制应用中的数据采集系统。MEASURE 由三个源文件组成：GETLINE.C，MCOMMAND.C 和 MEASURE.C。

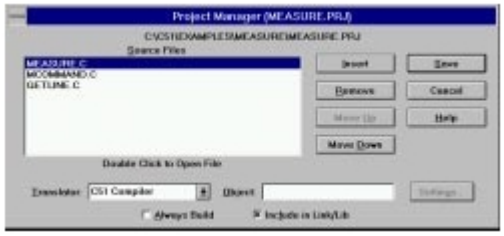
MEASURE 记录两个 8 位数字口和 4 个模拟输入口输入的数据。定时器控制采样率。采样间隔可配置为 1 微秒到 60 分钟。每次测量将当前时间和所有输入通道保存到 8K 字节的 RAM 缓冲区。

硬件要求

用于 MEASURE 的硬件基于 80516 CPU。该微控制器提供模拟和数字输入功能。口 4 和口 5 用作数字输入，AN0～AN3 用作模拟输入。由于 dScope 模拟了该程序所需要的所有硬件功能，因此不需要目标 CPU。

MEASURE 项目文件

MEASURE 的项目文件名为 MEASURE.PRJ。要装入该项目文件，选择项目菜单的打开命令并从\C51\EXAMPLES\HELLO 目录下选择 HELLO.PRJ 打开。从项目菜单中选择 Edit Project 命令，将显示项目管理器对话框。



项目管理器对话框显示 MEASURE 项目所包含的源文件。在该项目中有三个源文件。

MEASURE.C 该源文件包含了测量系统的主 C 函数和定时器 0 中断子程序。主函数初始化 80517 的所有外围功能并执行系统的处理命令。定时器 0 中断子程序管理实时时钟和系统的测量采样。如果输入通道较少，定时器 0 可保持和 8051 的兼容性。

MCOMMAND.C 该源文件处理显示、时间和间隔命令。这些功能从主函数调用。显示命令列出从 0.00V 到 5.00V 之间电压的浮点格式值。

GETLINE.C 该源文件包含了用于接收串行口字符的命令行编辑器。

要从项目管理器对话框中打开一个源文件，双击文件名即可。要关闭项目管理器对话框，按 **ESC** 键或 Cancel 按钮。

编译和连接 MEASURE

当用户准备编译和连接项目时，单击工具条上的 Build All 按钮，或者选择项目菜单中建立项目命令。µVision 开始编译和连接项目中的源文件并当建立完成时显示建立完成信息。

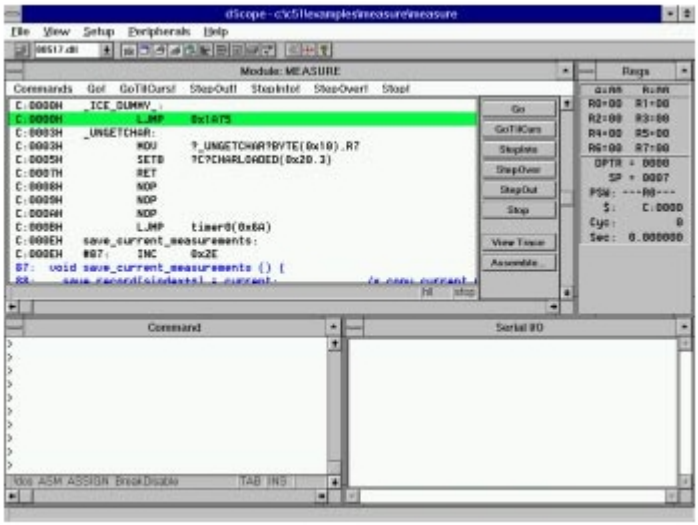
编译和连接完成后，就可开始测试 MEASURE 演示程序。

用 dScope 测试 MEASURE

MEASURE 演示程序设计成从片内串行口接受命令。如果有实际的目标硬件，可使用主计算机或哑终端与 80517 CPU 进行通讯。如果没有目标硬件，可使用 dScope 模拟硬件。也可使用串行窗口提供串行输入。

MEASURE 程序结束编译和连接之后，可使用 dScope 进行测试。在µVision 中，选择 DS51 模拟器命令，在出现对话框后按 **Enter** 键。

µVision 传递给 dScope 的初始化文件自动加载到 CPU 驱动器和 MEASURE 程序。dScope 显示下面的信息。



远程测量系统命令

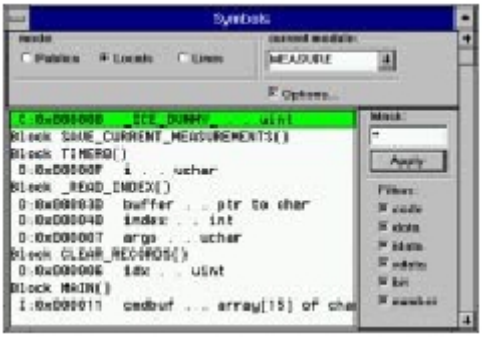
MEASURE 支持的串行命令如下表所示。这些命令由 ASCII 码文本字符组成。所有的命令必须以回车结束。

命令	串行文本	描述
Clear	C	清零缓冲区测量记录
Display	D	显示当前时间和输入值
Time	T *hh:mm:ss*	设置当前时间（24 小时格式）
Interval	I *mm:ss.ttt*	设置测量采样的间隔时间。间隔时间必须在 0:00.001（1ms）到 60:00.000（60 分钟）之间
Start	S	启动测量记录。在收到启动命令后，MEASURE 在给定的间隔时间采样所有输入数据
Read	R[count]	显示测量数据。可以使用 read 命令指定最近采样显示的数量。如果未指定数量，read 命令将发送所有记录的测量值。如果间隔时间大于 1 秒，可读出测量值。否则，记录必须停止。
Quit	Q	退出测量记录

观察调试符号

MEASURE 演示程序配置了所有调试信息，其中包括全局和局部符号、行号和高级类型信息。要看到这些信息，单击工具条上的 Symbol Browser 按钮打开符号浏览窗口。然后，选择 Locals radio 按钮，

选项检查框如下所示：



dScope 支持使用鼠标拖曳的方式访问符号。用鼠标将 **idx** 符号从符号浏览窗口拖曳到命令窗口，选择命令窗口并回车，dScope 显示 **idx** 的值。

通过选择存储空间过滤器，可过滤显示的符号。如果清除数据检查框，数据存储区内的所有符号都将从显示中移去。

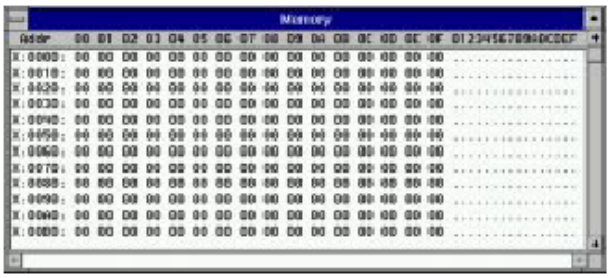
用户可指定一个搜索屏蔽限制符号的显示。如要限制开头字母为‘I’的字符显示，键入‘I*’并单击 Apply 按钮。

观察存储器内容

dScope 在存储器窗口显示 HEX 和 ASCII 码。单击工具条上的 Memory 按钮打开存储器窗口。在命令窗口输入想要查看的地址范围，例如：

D X: 0x0000, X: 0xFFFF

由于存储器窗口不能立即显示整个存储器范围。可以使用滚动条滚动显示存储器区域。滚动的范围由指定的地址范围决定。该例为 0x0000 到 0xFFFF。



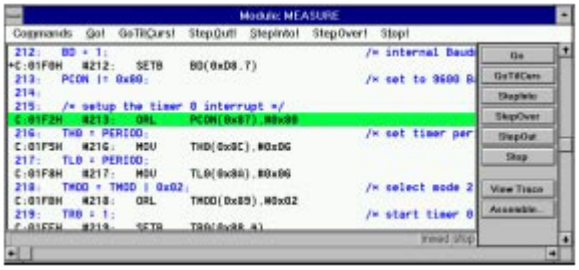
如要显示片内数据存储区，在命令窗口输入如下内容：

D I: 0x0000, I: 0xFF

当应用程序运行时，dScope 可动态地更新存储器窗口。要打开动态更新，从 Setup 菜单选择 Update Memory 窗口命令。当检查 Update Memory 窗口时，动态显示使能。

改变观察模式

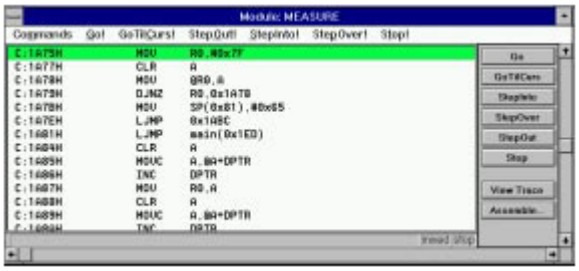
可以改变 dScope 调试窗口的模式。单击工具条上的 Debug 按钮打开调试窗口。然后，打开调试窗口的 Commands 菜单选择 View High level,Viw Mixed 或 View Assembly。例如，View Mixed 改变为源文件和汇编代码的混合显示。



程序执行

在开始模拟 MEASURE 程序之前，使用工具条上的 Debug, Register 和 Serial 按钮显示调试、寄存器和串行窗口。如果屏幕不够大，可关闭其它窗口。

从工具条上选择 Reset 按钮复位 dScope。在调试窗口中，从 Commands 菜单选择 View Mixed 命令，然后单击 StepInto 按钮。



StepInto 按钮可单步执行程序并进入函数调用。多次重复按下 StepInto 按钮将循环清零 CPU 片内数据区。

要跳过初始化代码并直接进入主函数，选择命令窗口并输入“G，main”。dScope 执行启动代码并停在主函数的第一条语句。

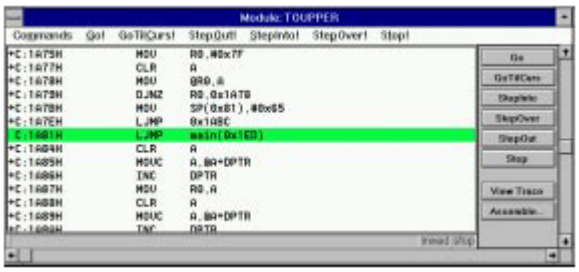
运行到当前光标行停止

当前光标行是指标记当前汇编或高级语句的行。可使用键盘或鼠标移动该行。dScope 允许将当前光标行作为一个临时断点。使用该特性可跳过程序中的代码。例如，用户可跳过初始化代码并停止在主函数调用的前一条指令。可使用下列两种方式之一来实现：

Variant 1：将光标行移到 **LJMP main** 指令。可使用键盘移动或用鼠标单击该行。单击调试窗口中的 GoTilCurs 按钮，dScope 从当前程序计数器指向的地址启动并停止在当前光标行。

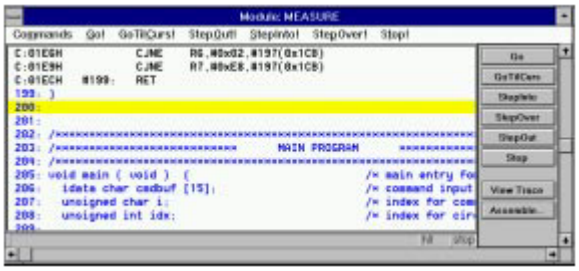
Variant 2：在 **LJMP main** 指令处双击鼠标右键可选择当前光标行。dScope 从当前 PC 启动并运行到当前光标行停止。

程序计数器现在就指向 **LJMP main** 指令。



单步执行高级语句

单击调试窗口中的 StepInto 按钮，dScope 跳转到 MEASURE 演示程序的主函数。从调试窗口 Command 菜单中选择 View High level 命令。



在高级模式中观察程序时，单步运行的方式有所改变。单步执行一条高级语句而不是一条汇编指令。单击 StepInto 按钮并当程序计数器行向下移动时进行观察。

注：

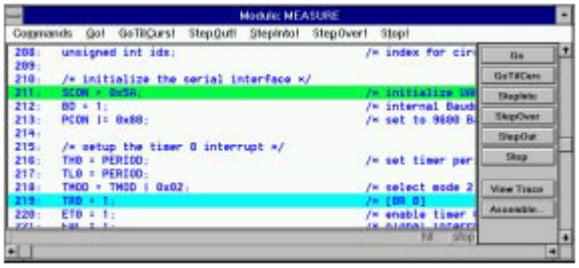
StepOver 按钮的操作与 StepInto 很相似，但不同点在于 StepOver 把一个函数调用看作一条单个的语句。

单步跳出函数

在偶然情况下，用户进入了不需要进入的函数，可使用 StepOut 按钮结束该函数的执行并返回到函数调用的下一条语句。

设置和移去断点

在调试窗口中双击所指向的程序行即可设置该行为一个断点。所选择的行呈高亮状态并且在行尾



显示[BR n]标号。如果在 TR0＝1 这条语句设置断点，调试窗口如下所示：

单击 Go 按钮，dScope 开始执行程序并在到达断点时停止。要去掉断点，双击设置断点的行。

调用堆栈

dScope 在程序执行时内部寻迹函数嵌套。通过打开 Call Stack 窗口可随时观察函数嵌套。按下工具条上的 Call Stack 按钮显示 Call Stack 窗口。

该对话框列出了当前所有嵌套的函数。每行包含嵌套级数、调用函数的数字地址和函数的符号名称。

口输入

dScope 提供两种不同的方式设置数字和模拟口输入。可使用主窗口的 Peripheral 菜单观察和改变输入口线的状态或在命令窗口输入 I/O 的值。下面所示为在命令窗口输入命令改变口的值。

PORT4=0x23	将数字输入口 3 设为 0x23
AIN1=3.3	将模拟输入口 AIN1 设为 3.3 伏

信号函数

dScope 允许用户建立信号函数作为数字或模拟输入信号。要装入信号函数，单击调试窗口的 Stop 按钮停止程序的执行，并在命令窗口输入以下命令：

INCLUDE analog.inc

该命令装入文件 ANALOG.INC。该文件定义一个信号函数用于调整模拟通道 0 的模拟输入值。函数如下所示：

```
SIGNAL void analog0 (float limit) {
    float volts;
    printf  ("ANALOG0 (%f)  ENTERED\n",  limit) ;
```

```
        while   (1)     {                                                   /* forever */
            volts = 0 ;
            while   (volts <= limit)    {
            ain0 = volts ;                                          /* analog input-0 */
            twatch (30000) ;                                /* 30000 Cycles Time-Break */
            volts += 0.5 ;                                          /* increase voltage */
        }
        volts = limit - 0.5 ;
        while (volts >= 0.5)    {
            ain0 = volts ;
            twatch   (30000) ;                              /* 30000 Cycles Time-Break */
            volts -= 0.5 ;                                          /* decrease voltage */
        }
    }
}
```

在装入模拟包含文件后，在命令窗口输入下列命令：

ANALOG0　(5.0)

G

该命令将模拟通道 0 的电压设置为 0～5V 并启动程序运行。

选择串行窗口并输入 **D Enter**，就可以观察到模拟输入的信号从 0～5V 间变化。

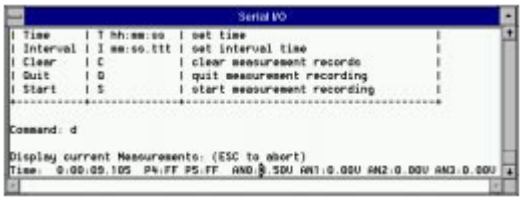
跟踪记录

在调试到达断点时，用户一般都想得到在断点处诸如寄存器值和其它一些细节的信息。dScope 为这个目的提供了跟踪记录功能。

要使能跟踪记录，选择 Commands 菜单中的 Record Trace 命令打开指令跟踪记录。当跟踪记录使能时，dScope 最多可记录 512 条汇编指令和寄存器内容。

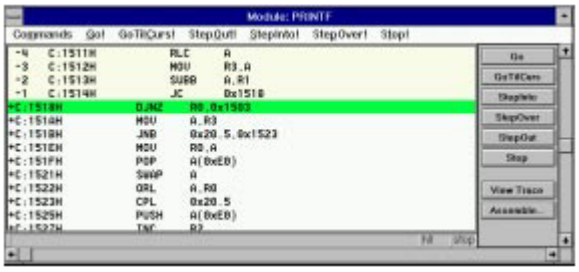
以 MEASURE 程序为例，开始运行 MEASURE 程序（单击调试窗口中的 Go 按钮）并选择串行窗口。MEASURE 显示一个菜单并在 Command 出现后等待输入，在串行窗口输入 **d**。

输入此命令后，MEASURE 开始显示测量值、记录时间、两个口值和最后的模拟输入值。



串行窗口显示连接到 80517 串行口上的哑终端的内容。

单击调试窗口中的 Stop 按钮可立即终止程序的执行。单击 View Trace 按钮观察跟踪缓冲器。



调试窗口的上半部分显示历史记录。调试窗口的下半部分显示程序计数器以后的指令。程序计数器行作为历史记录和还未执行指令的分隔符。

历史记录行以负数开始。跟踪缓冲器最新的记录为-1。最久的记录为-512。当缓冲器已满时，最老

的记录将被删除。

用户可使用键盘或鼠标滚动显示跟踪缓冲器。寄存器窗口显示跟踪缓冲器中所选指令的寄存器内容。

注：在观察跟踪缓冲器之前必须停止程序的执行。

观察点

观察点用于观察简单变量、结构和数组的内容。用户可使用 Watchpoints 对话框设置观察点。从 Setup 菜单中选择 Watchpoints 命令即可显示该对话框。

下面的步骤教您如何定义两个观察点：一个是无符号 int 变量 **sindex**，另一个是包含嵌套 **time** 的结构 **current**。

要加入观察点 **sindex**：在 Expr 输入行键入 sindex 并单击 Define watch 按钮。

要加入观察点 **current:** 在 Expr 输入行键入 current 并选择 Multiple radio 按钮在单独的行显示结构成员，再单击 Define watch 按钮。

观察窗口现在就包含了刚才定义的两个观察点。

sindex 的值在单独行内显示。

第二个观察点 **current** 产生较多输出。在单独行显示的结构成员缩进以反映嵌套的级数。最后几行显示保存在 **analog** 数组中的数据。

观察窗口在每次命令（StepInto,StepOut 或 Go）执行结束时更新。用户可选择 Setup 菜单中的 Update Watch Window 命令，将 dScope 配置为程序执行时周期性更新观察窗口。

断点

用户使用断点在一个给定的地址或指定的条件下将程序停止。执行断点是最简单的形式，一个函数地址或行数指向停止的地方。

当一个变量包含一定值时，用户可能希望程序停止执行。下面的例子讲述当 **current.time.sec** 结构成员设为 3 时，如何停止程序的执行。

从 Setup 菜单选择 Breakpoints 命令显示断点对话框。在表达式输入行输入 current.time.sec= =3。在 Count 输入行输入 1。选择写检查框（该选项指定当表达式写入时，仅作为断点测试）。

完成以后单击 Define 按钮设置断点。按下列步骤测试断点条件：

1．复位 dScope，

2．开始执行 MEASURE 演示程序（单击调试窗口中的 Go 按钮），

3．在 MEASURE 命令提示符下按回车。

在几秒钟后，dScope 停止运行。调试窗口中的程序计数器行标志产生断点所在的行。

使用性能分析器

使用 dScope 集成的性能分析器可对应用程序进行时序分析。指定一个供 dScope 使用的地址范围或一个函数。如要进行时序分析，在命令窗口输入以下命令：

PA main

PA timer0

PA clear_records

PA measure_display

PA save_current_measurements

PA read_index

```
RESET   PA                                          /* Initialize PA */
```

这些命令创建了用于时序统计的性能分析器地址范围。用户可使用Setup菜单中的Setup Performance Analyzer 命令创建或观察范围。

执行下列步骤对性能分析器进行观察：

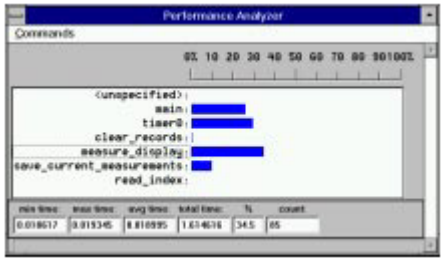
1．使用工具条上的按钮打开性能分析器窗口。显示以上定义的地址范围。*<unspecified>*行累加定义范围外所有的执行时间，

2．复位 dScope，

3．单击调试窗口中的 Go 按钮启动程序的执行，

4．选择程序窗口并输入 **S Enter D Enter**。

性能分析器窗口显示每个范围的条状图。



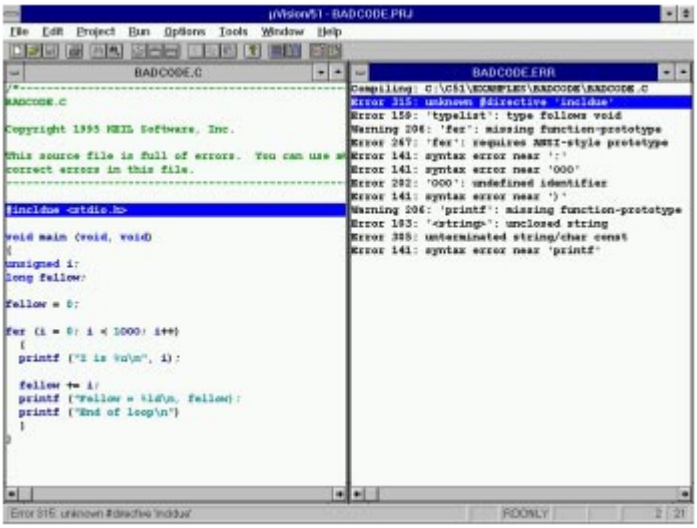
条状图动态更新并显示每个范围中执行代码所占时间的百分比。单击单个范围可看到时间的统计数据。

BADCODE：带语法错误的例子

名为**BADCODE.C**的文件位于\C51\EXAMPLES\BADCODE\子目录下。该文件演示如何使用µVision在源程序中找出错误并显示错误信息。

使用 File 菜单中的 Open 命令打开 **BADCODE.C** 文件从 Project 菜单选择 Compile File 命令编译该文件。在编译完成后，µVision 确定文件中有错误并显示一个错误信息窗口供用户阅读。

用户可使用错误窗口中的光标键滚动显示由编译器产生的错误。当光标在行之间移动时，源程序窗口更新显示对应错误产生的源程序行。



当错误窗口显示时，可能会覆盖一部分源程序窗口。可垂直或水平并排显示这两个窗口。

第六章 硬件产品

Keil 提供众多的硬件产品帮助用户进行 8051 的软件开发。当前，我们的硬件产品包括：

- ProROM EPROM 仿真器
- MCB517A 评估板
- MCB520 评估板

ProROM EPROM 仿真器

ProROM 是一个 EPROM 仿真器。通过它将 PC 的并行口和目标硬件的 ROM 插座相连。使用 ProROM EPROM 仿真器可以迅速开发并测试你的嵌入式目标程序。将 64K 字节程序下载到 ProROM 中只需数秒，在软件反复操作时不需要花费几分钟时间等待 EPROM 编程器和擦除器。

ProROM 带有一个装载器程序使二进制或十六进制文件下载更容易。此外，还可以使用带μVision 开发环境的 ProROM 使开发周期实现自动化。

ProROM EPROM 仿真器包括以下组件：

- 用户指南
- 软件和文件转换工具
- ProROM EPROM 仿真器
- 28 脚 DIP 接口电缆
- PC 并行口电缆

ProROM 为软件开发提供迅速方便的解决方案。

第七章 实时内核

该章讲述可用于 8051 微控制器的不同的实时操作系统。

RTX-51 实时操作系统

RTX-51 实时操作系统是一个多任务处理内核用于 8051 系列微控制器。它简化了复制的、对时间有严格要求的应用的软件设计。

以下为 RTX-51 两个不同的版本：

RTX-51 Full 执行循环和优先任务切换，最多有 4 个任务优先级。RTX-51 Full 和中断函数以并行方式工作。可使用一个邮箱系统在任务间传递信号和信息。从存储器库中可分配和释放存储器。还可强制一个任务等待另一个任务或中断产生中断、定时溢出、信号或信息。

RTX-51 Tiny 是 RTX-51 Full 的子集。RTX-51 Tiny 不需要外部数据存储器就可运行在单片 8051 系统上。RTX-51 Tiny 支持 RTX-51 Full 的许多特性，但以下除外：

1．由循环多任务处理和信号实现任务切换。
2．不支持优先任务切换。
3．不包含信息子程序。
4．无存储器库分配子程序。

该章余下部分将 RTX-51 Full 和 RTX-51 Tiny 统称为 RTX-51。不同之处在应用处说明。

简介

许多微控制器的应用要求同时进行多任务或工作的执行。对这样的应用，可使用实时操作系统（RTOS）对系统资源（CPU，存储器等等）进行灵活的安排。RTX-51 功能强大且简单易用，可用于所有的 8051 派生器件。

使用标准 C 结构编写并编译 RTX-51 程序，然后使用 C51 对其进行编译。为了指定任务 ID 和优先级，只有少数偏离标准 C。RTX-51 程序还需要包含实时可执行头文件、使用 BL51 代码库连接/定位器的连接以及合适的 RTX-51 库文件。

单任务程序

标准 C 程序从主函数开始执行。在嵌入式应用中，主函数通常被编码成死循环并被认为是一个连

续执行的单任务。例如：

```
int   counter;
void main   (void)   {
   counter = 0;
   while (1)   {                                        /* repeat forever */
      counter++;                                        /* increment counter */
   }
}
```

循环程序

一个更复杂的 C 程序可执行不使用 RTOS 的循环伪-多任务处理模式。在该模式中，任务或函数从一个死循环中循环调用。例如：

```
int   counter;
void main   (void)   {
   counter = 0 ;
   while (1)   {                                        /* repeat forever */
      check_serial_io () ;
      process_serial_cmds () ;                          /* process serial input */
      check_kbd_io ();
      process_kbd_cmds () ;                             /* process keyboard input */
      adjust_ctrlr_parms () ;                           /* adjust the controller */
      counter++ ;                                       /* increment counter */
   }
}
```

RTX-51 循环排程

RTX-51 执行循环多任务处理，允许几个循环或任务准并行执行。任务并不同时执行而是按时间分段执行。有效的 CPU 时间划分成时间段并由 RTX-51 将时间段分配给每个任务。每个任务允许执行预定数量的时间，然后，RTX-51 切换到另一个任务运行。时间段非常短，通常仅有几个毫秒。因此，任务看起来好象是同时执行的。

RTX-51 使用一个定时子程序，其中断驱动是由 8051 的硬件定时器提供的。产生周期性中断用作驱动 RTX-51 时钟。

RTX-51 不要求程序中有主函数。它自动从任务 0 开始执行。如果有主函数，必须使用 **os_create_task**（RTX-51 Tiny）和 **os_start_system**（RTX-51 Full）函数人为地启动 RTX-51。

下面的例子演示一个简单的 RTX-51 应用，仅使用循环任务排程。该程序中的两个任务都是简单的计数器循环。RTX-51 开始执行函数名为 job0 的任务 0。该函数加到另一个任务时叫做 job1。在 job0 执行一会儿后，RTX-51 切换到 job1，然后再切换到 job0。该处理过程无限重复下去。

```
#include <rtx51tny.h>
int counter0;
int counter1;
void job0   (void) _task_ 0   {
os_create (1);                                  /* mark task 1 as ready */
while (1)   {                                   /* loop forever */
counter0++;                                     /* update the counter */
   }
}
```

```
void job1 (void) _task_ 1   {
   while (1)   {                                   /* loop forever */
      counter1++;                                  /* update the counter */
   }
}
```

RTX-51 事件

在等待一个任务事件段结束时，可使用 **os_wait** 函数向 RTX-51 发出信号，使其可以开始执行另一个任务的执行。该函数暂停当前任务并等待产生指定的事件。在该时间里，可执行任何其它的任务。

使用 RTX-51 超时

使用 **os_wait** 函数等待的最简单的事件是 RTX-51 时钟报时信号中的超时周期。该类型的事件可用于需要产生延时的任务。这可用作代码中的切换查询。在这样的情况下，只需要每 50ms 检查一次切换。

下面的例子演示在允许其它任务执行时如何使用 **os_wait** 函数延迟执行。

```
#include   <rtx51tny.h>
int counter0;
int counter1;
void job0 (void) _task_ 0   {
   os_create (1);                                  /* mark task 1 as ready */
  while (1)   {                                    /* loop forever */
      counter0++ ;                                 /* update the counter */
      os_wait (K_TMO, 3) ;                         /* pause for 3 clock ticks */
   }
}
void job1 (void) _task_ 1   {
   while (1)   {                                   /* loop forever */
      counter1++ ;                                 /* update the counter */
      os_wait (K_TMO, 5) ;                         /* pause for 5 clock ticks */
   }
}
```

在上面的例子中，job0 象前面一样使能 job1。但在 counter0 加 1 之后，job0 调用 os_wait 函数暂停 3 个报时信号。此时，RTX-51 切换到下一个任务 job1。在 counter1 加 1 之后，job1 也调用 os_wait 函数暂停 5 个报时信号。现在，RTX-51 没有其它可执行的任务，因此在它能够执行 job0 之前，只有进入空闲循环等待 3 个报时信号过去。

该程序的结果就是每过 3 个报时信号 counter0 加 1，每过 5 个报时信号 counter1 加 1。

使用 RTX-51 信号

在等待另一个任务的信号（或二进制信号）时，可使用 os_wait 函数暂停一个任务。这可用于协调两个或多个任务。等待一个信号的过程如下：如果一个任务准备等待一个信号且信号标志为 0，任务暂停直到信号发送出去为止。如果当任务查询信号时信号标志已经为 1，标志清零且任务继续执行。如下面程序所示：

```
#include   <rtx51tny.h>
int   counter0;
int   counter1;
void job0   (void) _task_ 0   {
   os_create (1) ;                                 /* mark task 1 as ready */
   while (1)   {                                   /* loop forever */
```

```
      if (++counter0 == 0)                    /* update the counter */
        os_send_signal (1) ;                        /* signal task 1 */
    }
  }
  void job1 (void) _task_ 1   {
    while (1)   {                                   /* loop forever */
      os_wait (K_SIG, 0, 0) ;                  /* wait for a signal */
      counter1++ ;                            /* update the counter */
    }
  }
```

在上面的例子中，job1 在收到来自任何其它任务的信号之前一直等待。当它收到信号时，counter1 加 1 并继续等待下一个信号。job0 对 counter0 加 1 直到它溢出为 0。当溢出后，job0 发送一个信号到 job1 且 RTX-51 准备开始执行 job1。job1 直到 RTX-51 到达下一个定时信号时才开始执行。

优先级和占先

上述程序有一个缺点，即 job1 收到 job0 信号后并不立即开始执行。在某些情况下，由于时序的原因这是不可接受的。RTX-51 允许对任务分配优先级。当较高优先级的任务有效时，它中断或占先较低优先级的任务。这叫做优先级多任务处理或占先。

注：RTX-51 Tiny 不支持优先级和占先。

可修改上面 job1 的函数说明，使其的优先级高于 job0。默认情况下，所有任务的优先级都为 0。这是最低的优先级。优先级可从 0～3。下面的例子演示如何将 job1 的优先级定义为 1。

```
  void job1 (void) _task_ 1 _priority_ 1   {
    while (1)   {                                   /* loop forever */
      os_wait (K_SIG, 0, 0) ;                  /* wait for a signal */
      counter1++ ;                            /* update the counter */
    }
  }
```

现在，只要 job0 发送信号给 job1，job1 立即开始执行。

带 RTX-51 的编译和连接

RTX-51 完全集成在 C51 编程语言中。这使得产生 RTX-51 应用变得较容易控制。用户不需要写任何 8051 汇编程序或函数。仅需要使用 C51 编译 RTX-51 程序并使用 BL-51 进行连接。

例如，可使用下列带 RTX-51 Tiny 的命令行：

```
C51 EXAMPLE.C
BL51 EXAMPLE.OBJ   RTX51TINY
```

使用下列命令行实现带 RTX-51 的编译和连接：

```
C51 EXAMPLE.C
BL51 EXAMPLE.OBJ   RTX51
```

中断

RTX-51 中断函数以并行方式工作。中断函数可与 RTX-51 进行通讯并可将信号或信息发送到 RTX-51 任务。RTX-51 Full 将中断配置为一个任务。

信息传递

RTX-51 Full 支持下列函数在任务间交换信息：**isr_recv_message,isr_send_message,os_send_message** 和 **os_wait**。

信息是一个可被存储器模块看作是数字或指针的 16 位值。RTX-51 Full 支持使用存储器库系统的

变量信息。

CAN 通讯

通过 RTX-51/CAN 可以容易地实现 CAN。RTX-51/CAN 是一个集成在 RTX-51 Full 中的 CAN 任务。RTX-51 CAN 任务通过 CAN 网络实现信息的传递。其它 CAN 终端可配置成带 RTX-51 或不带 RTX-51。

BITBUS 通讯

RTX-51 Full 包含了主控器和被控器 BITBUS 任务，用于支持与 Intel 8044 的信息传递。

事件

RTX-51 支持用于 **os_wait** 函数的事件如下：

- **Timeout** 暂停一定数量时钟周期的运行中任务的执行。
- **Interval** 和 **Timeout** 相似，但是 **interval** 用于必须同步执行的任务。
- **Signals** 用于任务间的协调
- **Messages** 用于信息的交换*
- **Interrrupt** 用于等待 8051 硬件中断的任务*
- **Semaphores** 用于共享系统资源的管理*

* 表示这些事件只可在 RTX-51 Full 中实现。

RTX-51 函数

下表列出了 RTX-51 的一些函数以及简要描述和执行时间（RTX-51 Full）。

函数	描述	CPU 周期
isr_recv_message *	接收一个信息 (从中断调用).	71 (连同信息)
isr_send_message *	发送一个信息 (从中断调用).	53
isr_send_signal	将一个信号发送给任务 (从中断调用).	46
os_attach_interrupt *	将任务分配给中断源	119
os_clear_signal	删除一个已发送的信号	57
os_create_task	将一个任务加入执行队列	302
os_create_pool *	定义一个存储器库	644 (规格 20 * 10 字节)
os_delete_task	从执行队列移除一个任务	172
os_detach_interrupt *	移除中断分配	96
os_disable_isr *	禁能 8051 硬件中断	81
os_enable_isr *	使能 8051 硬件中断	80
os_free_block *	将一个块送回存储器库	160
os_get_block *	从存储器库取出一个块	148
os_send_message *	发送一个信息 (从任务调用).	443 with task switch
os_send_signal	将一个信号发送给任务(从任务调用).	408 with task switch 316 with fast task switch 71 without task switch
os_send_token *	设置一个信号标志 (从任务调用).	343 with fast task switch 94 without task switch
os_set_slice *	设置 RTX-51 系统时钟时间段	67
os_wait	等待一个事件	68 for pending signal 160 for pending message

***** 这些函数仅可在 RTX-51 Full 中实现。

RTX-51 中额外的调试和支持函数如下所示：

函数	描述
oi_reset_int_mask	禁能对 RTX-51 的外部中断源
oi_set_int_mask	使能对 RTX-51 的外部中断源
os_check_mailbox	关于指定邮箱状态的返回信息
os_check_mailboxes	关于系统中所有邮箱状态的返回信息
os_check_pool	关于存储器库中块的返回信息
os_check_semaphore	关于指定信号标志状态的返回信息
os_check_semaphores	关于系统中所有信号标志状态的返回信息
os_check_task	关于指定任务的返回信息
os_check_tasks	关于系统中所有任务的返回信息

CAN 函数

CAN 函数仅可在 RTX-51 Full 上实现。CAN 控制器支持 Philips 82C200,80C592 和 Intel 82526。

函数	描述
can_bind_obj	将一个目标连接到任务；当收到目标时启动任务
can_def_obj	定义通讯目标
can_get_status	得到 CAN 控制器状态
can_hw_init	初始化 CAN 控制器硬件
can_read	直接读一个目标的数据
can_receive	接收所有拆分目标
can_request	向指定的目标发送一个远程帧
can_send	在 CAN 总线上发送一个目标
can_start	启动 CAN 通讯
can_stop	停止 CAN 通讯
can_task_create	创建 CAN 通讯任务
can_unbind_obj	将任务和目标间的绑定拆分
can_wait	等待一个绑定目标的接收
can_write	将新数据写入一个目标但不发送

技术数据

描述	RTX-51 Full	RTX-51 Tiny
任务数	256;最大 19 个任务有效	16
RAM 要求	40..46 字节 DATA 20..200 字节 IDATA(用户堆栈) 最小..650 字节 XDATA	7 字节 DATA 3*<任务计数>IDATA
代码要求	6KB..8KB	900 字节
硬件要求	定时器 0 或定时器 1	定时器 0
系统时钟	1000..40000 个周期	1000..65535 个周期
中断等待时间	<50 个周期	<20 个周期
上下文切换时间	70..100 个周期(快速任务) 180..700 个周期(标准任务) 由堆栈装载决定	100..700 个周期 由堆栈装载决定
信箱区系统	8 个邮箱，每个邮箱有 8 个整数入口	不能实现
存储器库系统	最多 16 个存储器库	不能实现
信号标志	8*1 位	不能实现

第八章 命令参考

该章主要讲述用于 Keil 8051 开发工具的命令和控制符。这些命令和控制符以表格的形式列出。带下划线的字符表示特定控制符和命令的缩写。

A51 宏汇编器

调用： A51 sourcefile directives
A51 @commandfile

此处 sourcefile 为汇编器源文件名，
commandfile 为包含完整命令行的文件名，
directives 为下表中所列出的参数。

A51 控制符	含义
DATE(date)	将 **date** 串放入头文件(最多 9 个字符)
DEBUG	在目标文件中包含调试符号信息
ERRORPRINT [(filename)]	将错误信息输出到 **filename**.
INCLUDE(filename)	包含汇编中的 **filename** 内容
MACRO	使能标准宏处理 Enables standard macro processing
MPL	使能 Intel-类型宏处理
NOAMAKE	排除目标文件中的 **AutoMAKE** 信息
NOCOND	排除列表文件中的未汇编的条件汇编代码
NOGEN	禁止列表文件中的宏扩展
NOLINES	排除目标文件中的行数信息
NOLIST	排除列表文件中的汇编器源代码
NOMACRO	禁止标准宏处理
NOMOD51	禁止预定义的 8051-专用特殊功能寄存器
NOSYMBOLS	排除列表文件中的符号表
NOSYMLIST	排除列表文件中的符号定义
OBJECT[(filename)], **NOOBJECT**	使能或禁止目标文件输出。如果指定文件名，则目标文件以指定的 **filename** 为名保存
PAGELENGTH(n)	设置列表文件每一页的最大行数
PAGEWIDTH(n)	设置列表文件每一行的最大字符数
PRINT[(filename)], **NOPRINT**	使能或禁止列表文件输出. 如果指定文件名，则列表文件以指定的 **filename** 为名保存
REGISTERBANK(num, ...), **NOREGISTERBANK**	指示使用一个或多个寄存器区，或者指示没有使用寄存器区
RESET (symbol, ...)	将指定的符号赋值为 0000h
SET (symbol, ...)	将指定的符号赋值为 0FFFFh
TITLE(title)	在列表文件的页眉包含 **title**
XREF	在列表文件中包含符号交叉引用的列表

C51 编译器

调用： C51 sourcefile directives
C51 @commandfile

此处 sourcefile 为 C 源文件名，
commandfile 为包含完整命令行的文件名，
directives 为下表中所列出的控制参数。

C51 控制符	含义
CODE	在列表文件中包含一个汇编列表
COMPACT	选择 **COMPACT** 存储器模型
DEBUG	在目标文件中包含调试信息
DEFINE	在命令行定义处理器名
FLOATFUZZY	在浮点比较中指定位的数目
INTERVAL	指定中断向量的间隔
INTVECTOR(n), **NOINTVECTOR**	指定中断表的偏移量，使用 **n**，或排除目标文件中的中断向量
LARGE	选择 **LARGE** 存储器模型
LISTINCLUDE	在列表文件中包含 include 文件的内容
MAXARGS(n)	指定保留用作变量长度参数列表的字节数
MOD517	使能对 Siemens 80C517 及其派生器件的附加硬件的支持
MODDP2	使能对 Dallas 半导体 80C320/520/530 和 AMD 80C521 的附加硬件的支持
NOAMAKE	排除目标文件中的 **AutoMAKE** 信息
NOAREGS	禁止使用 **ARn** 指令进行绝对寄存器寻址
NOCOND	排除列表文件中的跳转条件代码
NOEXTEND	禁止 8051/251 扩展，仅处理 ANSI C 结构
NOINTPROMOTE	禁止 ANSI 整数提升规则
NOREGPARMS	禁止寄存器中的参数传递
OBJECT[(filename)], **NOOBJECT**	使能或禁止目标文件输出。如果指定文件名，则目标文件以 **filename** 为名保存
OBJECTEXTEND *	在目标文件中包含附加的变量类型信息
OPTIMIZE	指定执行编译器时的优化级别
ORDER	将存储器中变量定位成和在源文件中说明的顺序相同
PAGELENGTH(n)	设置列表文件每一页的最大行数
PAGEWIDTH(n)	设置列表文件每一行的最大字符数
PREPRINT [(filename)]	使用所有宏扩展产生一个预处理器列表文件。如果指定文件名，则预处理文件以 **filename** 为名保存
PRINT[(filename)], **NOPRINT**	使能或禁止列表文件输出。如果指定文件名，则列表文件以 **filename** 为名保存
REGFILE(filename)	指定产生文件名以包含寄存器使用信息
REGISTERBANK	选择寄存器区以使用源文件中的函数
ROM({SMALL|COMPACT|LARGE})	控制 **AJMP** 和 **ACALL** 指令的产生
SMALL	选择 **SMALL** 存储器模型
SRC	创建一个汇编源文件代替一个目标文件
SYMBOLS	在列表文件中包含所使用符号的列表
WARNINGLEVEL(n)	控制所产生警告的类型和严重性

L51/BL51 连接器/定位器

调用： BL51 inputlist│To outputfile││directives│

L51 inputlist│To outputfile││directives│

BL51 @commandfile

L51 @commandfile

此处 inputlist 为由逗号隔开的目标文件和库文件的列表。

outputfile 为连接器创建的绝对目标模块名

commandfile 为包含完整命令行的文件名，

directives 为下表中所列出的控制参数。

BL51 控制符	含义
BANKAREA *	指定代码区所处的地址范围
BANKx *	指定代码区 0 到 31 的起始地址、段和绝对目标模块
BIT	定位和排序 **BIT** 段
CODE	定位和排序 **CODE** 段
COMMON *	指定放置在公共区的起始地址、段和绝对目标模块。该命令实质上和 **CODE** 指令相同
DATA	定位和排序 **DATA** 段
IDATA	定位和排序 **IDATA** 段
IXREF	在列表文件中包含交叉引用报告
NAME	指定用于目标文件的模块名
NOAMAKE	排除目标文件中的 AutoMAKE 信息
NODEBUGLINES	排除目标文件中的行号信息
NODEBUGPUBLICS	排除目标文件中的公共符号信息
NODEBUGSYMBOLS	排除目标文件中的局部符号信息
NODEFAULTLIBRARY	排除运行时间库中的模块
NOLINES	排除列表文件中的行号
NOMAP	排除列表文件中的存储器映射信息
NOOVERLAY	防止局部 **BIT** 和 **DATA** 段的覆盖或重叠
NOPUBLICS	排除列表文件中的公共符号信息
NOSYMBOLS	排除列表文件中的局部符号信息
OVERLAY	指示连接器覆盖局部 数据&位 段并使用户可改变段之间的引用
PAGELENGTH(n)	设置列表文件每一页的最大行数
PAGEWIDTH(n)	设置列表文件每一行的最大字符数
PDATA	指定 **PDATA** 段的起始地址
PRECEDE	定位和排序内部数据存储区中先于其它段的段
PRINT	指定列表文件的名称
RAMSIZE	指定片内数据存储区的规模
REGFILE(filename)	指定所产生的文件名以包含寄存器使用信息
RTX51 *	包含 RTX-51 full 实时内核的支持
RTX51TINY *	包含 RTX-51 tiny 实时内核的支持
STACK	定位和排序 **STACK** 段
XDATA	定位和排序 **XDATA** 段

***** 表示这些命令仅可用于 BL51 代码分区连接器/定位器。

OC51 目标文件转换器

调用： OC51 banked_file

此处 banked_file 为分区目标文件名

OH51 目标-Hex 转换器

调用： OH51 absfile HEXFILE(hexfile)

此处 absfile 为绝对目标文件名

hexfile 为将要创建的 Intel HEX 文件名

LIB51 库管理器

调用： LIB51 | command |

此处 command 为在下表所列出的其中一个控制命令。如果未给出命令，LIB51 进入交互式命令模式。

LIB51 控制符	含义
ADD	将一个目标模块加入库文件
CREATE	创建一个新的库文件
DELETE	从库文件中移去一个目标模块
EXIT	退出库管理器交互模式
HELP	显示库管理器的帮助信息
LIST	显示保存在库文件中的模块和公共符号信息