KEIL C51 说明书之

µVision2 调试命令

翻译者：易小龙
网名：yialong
E-mail:yialong@163.com

原文件：
Keil C51 dbg51.chm

www.c51bbs.com
网站协助发布

目　录

µVision2 调试命令

µVision2 支持许多种命令，你可以在 **Output Window - Command** 窗口输入这些命令以用于调试程序。这些命令可分为以下几类，下表将对每个类别的共性功能进行描述。

类别	描述
断点命令	创建和删除断点。当程序运行到特殊的指令时，可以使用断点停止程序运行，或执行调试命令或用户程序。
通用命令	提供多种调试操作。
存储器命令	显示和改变存储器的内容。
程序命令	执行目标程序，分析程序运行性能。
断点命令	定义在监视窗显示的存储器区域，和变量。

本章将用类别形式列举调试命令，并按每种命令的字母顺序分别给出命令的详细情况，以供参考。

注意
输入命令只需使用命令名称中带下划线的大写字母。例如，**BreakSet** 命令时输入 **BS**。

在输入命令时，语法生成器会显示出可能的命令、选择和参数。在输入过程中，µVision2 会逐步减少列举的相似的命令，以你输入的字母相符。

如果输入 **B**，语法生成器会减少列举的命令	
如果命令清楚的话，将列出选项	
在输入的整个过程，语法生成器进行引领和帮助，以避免输入的错误。	

存储器命令

可以用以下存储器命令显示或改变存储器的内容。

命令	描述
ASM	在线编译代码。
DEFINE	定义在µ**Vision2** 函数中可用的类型变量。
DISPLAY	显示存储器中的内容。
ENTER	输入数值到一个指定的存储器区域。
EVALUATE	计算一个表达式并输入结果。
MAP	指出存储器区域使用参数
UNASSEMBLE	反汇编程序存储器
WATCHSET	增加一个监视变量到监视窗口。
WATCHKill	清除所有的监视窗口的监视变量。

程序命令

程序命令可以运行代码和一次步越一个程序指令。而且，µVision2 提供一个高级性能分析器，让你很容易地确定目标程序中的关键点。

命令	描述
COVERAGE	显示代码运行覆盖情况。
Esc	停止程序运行。
Go	开始程序运行。
Performance Analyzer	启动内建性能分析器。
PSTEP	步越指令，不步入程序或函数。
OSTEP	步出当前函数。
TSTEP	步越指令，会步入函数。

断点命令

通过µVision2 提供的断点功能，你可以在某些条件下停止目标程序的运行。断点可以设在读操作、写操作和执行操作上。

命令	描述
BREAKDISABLE	关闭一个或多个断点。
BREAKENABLE	开启一个或多个断点。
BREAKKILL	从断点列表中删除一个或多个断点。
BREAKLIST	列举当前的所有断点。
BREAKSET	加入一个断点表达式到断点列表中。

通用命令

以下命令并不属于任何命令类别。他们包含在µVision2 中，使得调试更容易、更方便。

命令	描述
ASSIGN	给串口窗分配输入输出源。
DEFINE	创建一个工具箱按钮。
DIR	产生一个变量名目录。
EXIT	退出调试模式。
INCLUDE	读出并执行一个命令文件中的命令。
KILL	删除µVision2 调试函数和工具箱按钮。
LOAD	载入目标模块和 HEX 文件。
LOG	对调试窗日记文件进行创建、查询状态和关闭操作。
MODE	设置 PC COM 口的波特率，以及停止位数目。
RESET	复位µVision2，复位存储器映射分配，复位预定义变量。
SAVE	保存一段存储器内容到一个 Intel HEX386 文件。
SCOPE	显示目标程序中的模块和函数的地址分配。
SET	设置预定义变量的字符串值。
SIGNAL	显示信号函数状态和删除已启动的信号函数。
SLOG	对串口窗日记文件进行创建、查询状态和关闭操作。

ASM

语法	描述
ASM	显示当前在线汇编代码的地址
ASM start address	设置在线汇编地址为 **start address**
ASM instruction	汇编指定的指令代码，并将结果码存入存储器中的当前在线汇编地址。当前在线编译地址增加指令的字节数。

ASM 命令显示和设置当前汇编地址，让你输入汇编指令。当指令输入时，结果码就存在程序存储器。你可以用在线汇编器来纠正错误，或对正调试的目标程序做一些临时的改变。

在线汇编器支持汇编指令助记符的使用。助记符随设备数据库的不同设备而异。

示例：

```
>ASM C:0x0000        /*  设置当前汇编地址为 C:0x0000    <8051 & 251) */
>ASM mov a,#12
>ASM mov r0,#0z20
>ASM movx @r0,a
>ASM inc r0
>ASM movx @r0,a
>ASM jmp c:0x8000

>ASM C:0020h         /*设置当前汇编地址为 C:0x0000 C:0x0020 <8051 & 251) */
>ASM CLR A
```

你可以通过在线汇编对话框输入汇编指令。右击“反汇编窗口”显示出菜单，选择“Inline Assembly…”项可打开该窗口。

ASSIGN

语法	描述
ASSIGN	显示串口输入输出的设置
ASSIGN *channel* < *inreg* > *outreg*	改变串口输入输出的设置

ASSIGN 命令显示和改变指定串口的输入输出。

以下表格显示了该命令支持的频道和缺省设置

渠道	缺省 Inreg	缺省 Outreg	描述
COM1	无	无	PC 机串口 1
COM2	无	无	PC 机串口 2
COM3	无	无	PC 机串口 3
COM4	无	无	PC 机串口 4
WIN	**SIN** 或 **S1IN**	**SOUT** 或 **S1OUT**	串行窗口

当前，µ**Vision2** 支持以下串口： **WIN, COM1, COM2, COM3,** 和 **COM4**。其中，**WIN** 是串口窗；**COM*x*** 频道代表 **PC** 机串口，可以用之与仿真单片机和目标系统通信。可以使用 **MODE** 命令设置 **PC** 机串口参数。请参见 **MODE** 命令。

输入和输出寄存器由你选择的 **CPU** 类型的设备数据库定义。可以用 **DIR** 命令列举所有有效的虚拟寄存器。

示例

```
>ASSIGN                        /* 显示串口设置                    */
  WIN: <S0IN >S0OUT            /* S0IN 提供串口输入               */
                               /* S0OUT 提供串口输出              */
>
>ASSIGN WIN <S1IN >S1OUT      /* 将 S1IN 和 S1OUT 寄存器分配给串口窗口*/
>ASSIGN                        /* 并显示这个分配                  */
  WIN: <S1IN >S1OUT
>ASSIGN COM1 < SIN > SOUT     /* 分配 SIN 和 SOUT 给串口 1       */
```

注

ASSIGN 命令不能用于µ**Vision2** 调试器与硬件设备通信时的目标模式。

BREAKDISABLE

语法	描述
BREAKDISABLE ***number , number…***	关闭，但不删除指定的断点。Numbers 是定义断点时指定的序号。
BREAKDISABLE *	关闭所有断点

BREAKDISABLE 命令关闭一个前面设定的断点。当运行到断点时，µVision2 通常会停止程序执行或执行一条指定的命令，关闭一个断点并不表示该断点被删除，而是使软件执行目标程序时忽略该断点。

示例

```
>BL                                    /*列举出所有断点        */

 0: (E C: 0xFF01EF) 'main', CNT=1, enabled
 1: (E C: 0xFF006A) 'timer0', CNT=10, enabled
     exec ("MyRegs()")

>BD 0                                  /* 关闭断点 0     */
>BD *                                  /* 关闭所有断点   */
>BL                                    /* 列举出所有断点*/

 0: (E C: 0xFF01EF) 'main', CNT=1, disabled
 1: (E C: 0xFF006A) 'timer0', CNT=10, disabled
     exec ("MyRegs()")
```

你也可以通过断点对话框来进行这项操作。可以在主菜单的 **Debug – Breakpoints…**.项打开该对话框。

BREAKENABLE

语法	描述
BREAKENABLE *number , number…*	开启指定断点
BREAKENABLE *	开启所有断点

BREAKENABLE 命令开启一个由 **BREAKDISABLE** 命令关闭的断点。

示例

```
>BE 0,1                    /* 开启 0 和 1 号断点    */
>BE *                      /* 开启所有断点          */
>BL                        /* 列举出所有断点        */

 0: (E C: 0xFF01EF) 'main', CNT=1, enabled
 1: (E C: 0xFF006A) 'timer0', CNT=10, enabled
     exec ("MyRegs()")
```

你也可以通过断点对话框来进行这项操作。可以在主菜单的 **Debug – Breakpoints…**打开该对话框。

BREAKKILL

语法	描述
BREAKKILL *number* , *number…*	删除指定的断点
BREAKKILL *	删除所有断点

BREAKKILL 命令删除由 **BREAKSET** 命令定义的断点。

示例：

```
>BK 0,1                    /* 删除 0 和 1 断点 */
>BK *                      /* 删除所有断点    */
```

你也可以通过断点对话框来进行这项操作。可以在主菜单的 **Debug – Breakpoints…**打开该对话框。

BREAKLIST

语法	描述
BREAKLIST	列举出所有断点

BREAKLIST 命令出所有断点。按以下格式显示。

```
number: (type) 'expression', CNT=count, enable_flag
exec ("command")
```

格式说明

number	断点索引号。
type	断点类型，可以是执行断点（e：紧跟着地址），条件断点（c），或存取断点（a：紧跟着“rd”表示读，“wr”表示写，“rw”表示读写，后跟地址）。
expression	断点定义的原始文字。
count	断点通过次数。如果 *count* 等于 2 时，µVision2 在第 2 次运行到该断点时停止程序运行或执行指定指令。
enable_flag	显示为 **enabled** 时表示是一个开启的断点，显示为 **disabled** 为一个关闭的断点。
command	当程序执行到断点时执行的命令。

示例

```
>BL                                                  /*列举出所有断点 */
 0: (E C: 0xFF01EF) 'main', CNT=1, enabled
 1: (E C: 0xFF006A) 'timer0', CNT=10, enabled
     exec ("MyRegs()")
 2: (C) 'sindex == 8', CNT=1, enabled
 3: (C) 'save_record[5].time.sec > 5', CNT=3, enabled
 4: (A RD 0x000037) 'READ interval.min == 3', CNT=1, enabled
 5: (A WR 0x000034) 'WRITE savefirst==5 && acc==0x12', CNT=1, enabled
```

你也可以通过断点对话框来进行这项操作。可以在主菜单的 **Debug – Breakpoints…**打开该对话框。

BREAKSET

语法	描述
BREAKSET *exp , cnt , "cmd"*	设置一个执行或条件断点。
BREAKSET READ *exp , cnt , "cmd"*	设置一个读断点。
BREAKSET WRITE *exp , cnt , "cmd"*	设置一个写断点。
BREAKSET READWRITE *exp , cnt , "cmd"*	设置一个读写断点。

BREAKSET 命令在指定的指令设置一个断点。

你可通过指定以下参数定义一个断点：

exp 指定地址，或由µ**Vision2** 在运行时计算的表达式。

cnt 断点通过次数。缺省为 **1**。

cmd 命令字符串。指定µ**Vision2** 在发生断点时执行的命令。当不指定命令时，在断点发生时，**Vision2** 停止程序运行。当指定了命令后，**Vision2** 会执行这条命令，但目标程序并不停止运行。命令可以指定一个 **Vision2** 用户或信号功能。你也可以将**_break_**变量设为 **1**，以停止程序运行。

µ**Vision2** 采用三种断点类型：执行断点、条件断点、以及存取断点。µ**Vision2** 规定使用断点时必须符合以下规定：

1. 当指定了存储器存取模式为 **READ**、 **WRITE** 或 **READWRITE** 时，这个断点就是一个存取断点。

2. 仅仅指定地址，断点为执行断点。

3. 当表达式不能精减为地址，断点为条件断点。

以下是对上述三种断点的说明。

执行断点

当运行到指定的代码地址时，执行断点停止程序运行或执行一条命令。执行断点不影响执行速度。

代码地址必须是一个指令的第一个字节的地址。如果将执行断点设置在一条指令的第二或第三个字节的地址上，断点将永不会发生。

只能对一个代码地址的设置一个执行断点，不允许多重定义。

条件断点

当达到指定的条件时，条件断点停止程序运行或执行一条命令。每执行一条指令都会重新计算一下条件表过式，因此，程序执行速度会受到影响，其程度取决于条件表达式的复杂度。条件断点的数量也影响程序的执行速度。

条件断点最具灵活性，因为有多个选项的条件可以通过表达式计算产生。

存取断点

存储器存取断点是指当一个指定地址被读、写或读写状态时，程序停止运行或执行一条特殊的命令。存储器存取断点对程序执行的速度影响不大，因为只有存取指定地址时，才进行指定的操作。

表达式必须仅由存储器地址和类型组成，以下是存取断点使用的一些规则：

1.一个存取断点的表达式必须只指定一种存储器类型。表达式不允许指定多个对象。

2.存取断点表达式仅允许使用**&, &&, <, <=, >, >=, ==, and !=**操作符。

存取断点示例：

```
BS WRITE time.sec                    /* 有效表达式 */
```

是有效的，而下面的断点：

```
BS WRITE time.sec + i0               /*无效表达式 */
```

是无效的，因为有两个参数值相加 **(timer.sec** 和 **i0)** ，不符合规则。

注意
用 **BREAKLIST** 命令来列举断点，存储器存取断点总是在 **READ, WRITE,** 或 **READWRITE** 之后。

示例

以下例子设置了一个在主程序地址的执行断点。

```
>BS main
```

以下例子设置了一个在 ***timer0*** 程序地址的执行断点。仅在该程序执行到第十次时，断点才发生，并执行命令 **MyRegs()**。在命令执行完后程序继续运行。

```
>BS timer0,10,"MyRegs()"
```

下面的例子设置了一个基于 ***sindex*** 变量的条件断点。当 ***sindex*** 等于 **8** 时，程序停止运行。

```
>BS sindex == 8
```

下面的例子设置了一个基于 ***save_record*** 数组变量的条件断点。当第三次执行到 ***save_record[5].time.sec*** 大于 **8** 时，程序停止运行。

```
>BS save_record[5].time.sec > 5, 3
```

下面的例子设置了一个基于 **interval.min** 变量 **READ** 模式的存取断点。当 **interval.min** 等于 **3** 时，程序停止运行。

```
>BS READ interval.min == 3
```

以下是设置了一个基于 ***savefirst*** 变量 **WRITE** 模式的存取断点。当 ***savefirst*** *被赋值，并* 等于 **5** 以及累加器 **acc** 等于 **0x12** 时，程序停止运行。

```
>BS WRITE savefirst == 5 && acc == 0x12
```

COVERAGE

语法	描述
COVERAGE	显示整个应用程序的运行覆盖情况。
COVERAGE *module*	显示选择的模块的运行覆盖情况。
COVERAGE *module**func*	显示模块中 *func* 函数的运行覆盖情况。

COVERAGE 命令显示µ**Vision2** 调试器的代码运行覆盖情况。与 **LOG** 命令搭配使用，可以将运行覆盖情况拷贝到日记文件。

也可使用代码覆盖窗口观察代码运行覆盖的情况。在主菜单的 **View - Code Coverage Window..**项可打开该窗口。

示例

下面的例子是用 **C51** 编译包中的 **MEASURE** 范例创建的。

```
\\Measure\MEASURE\{CvtB} - 32% of 52 instructions executed
\\Measure\MEASURE\{CvtB} - 0% of 127 instructions executed
\\Measure\MEASURE\{CvtB} - 100% of 1 instructions executed
\\Measure\MEASURE\{CvtB} - 100% of 6 instructions executed
\\Measure\MEASURE\{CvtB} - 0% of 595 instructions executed
\\Measure\MEASURE\{CvtB} - 0% of 5 instructions executed
\\Measure\MEASURE\{CvtB} - 100% of 1 instructions executed
\\Measure\MEASURE\SAVE_CURRENT_MEASUREMENTS - 0% of 46 instructions executed
\\Measure\MEASURE\TIMER0 - 72% of 94 instructions executed
\\Measure\MEASURE\READ_INDEX - 0% of 64 instructions executed
\\Measure\MEASURE\CLEAR_RECORDS - 100% of 24 instructions executed
\\Measure\MEASURE\MAIN - 31% of 215 instructions executed
\\Measure\MCOMMAND\{CvtB} - 0% of 74 instructions executed
\\Measure\MCOMMAND\MEASURE_DISPLAY - 100% of 44 instructions executed
\\Measure\MCOMMAND\SET_TIME - 0% of 58 instructions executed
\\Measure\MCOMMAND\SET_INTERVAL - 0% of 101 instructions executed
\\Measure\GETLINE\GETLINE - 78% of 51 instructions executed
\\Measure\TOUPPER\{CvtB} - 100% of 18 instructions executed
\\Measure\TOUPPER\{CvtB} - 51% of 91 instructions executed
\\Measure\TOUPPER\{CvtB} - 0% of 0 instructions executed
```

DEFINE

语法	描述
DEFINE *type identifier*	按指定类型定义一个名叫 ***identifier*** 变量。
DEFINE BUTTON "***label***", "***cmd***"	定义一个工具箱按钮。在主菜单 **View – Toolbox** 项可打开工具箱

DEFINE 命令可以创建一个带类型的变量，这个变量可以赋值。用这种方式创建的变量可以用来装载µVision2 函数的返回值，并可以指定µVision2 函数的输入值。

用 **DEFINE** 命令创建的变量并不占用仿真或目标 **CPU** 的存储器。它们只不过是指定类型的数值的符号名。用 **DEFINE** 命令创建的变量可以象其他公用变量一样使用。

变量类型一览表：

类型	描述
CHAR	带符号的字符型(**signed char**).
DOUBLE	双精度浮点型(**double**).
FLOAT	单精度浮点型 (**float**).
INT	带符号的整型 (**signed int**).
LONG	带符号的长整型 (**signed long**).

identifier 是变量名，它必须符合变量及标号的规则。

示例

```
>DEFINE CHAR TmpByte     /* 定义 TmpByte 为字符型变量*/
>DEFINE FLOAT TmpFloat   /* 定义 TmpFloat 为浮点型变量 */

>TmpFloat = 3.14159      /* 给 TmpFloat 赋值  */
>TmpFloat                /* 显示 TmpFloat 的值 */
3.14159
```

定义按钮

用这个命令增加一个按钮工具箱窗口。

label 是分配给按钮的名称。

cmd 是µ**Vision2** 命令或分配给按钮的命令。点击按钮就执行该命令。

示例：

```
>DEFINE BUTTON "clr dptr", "dptr=0"
>DEFINE BUTTON "show main()", "u main"
>DEFINE BUTTON "show r7", "printf (\"R7=%02XH\n\",R7)"
```

注意：
在上面最后一个定义按钮命令的例子中，**printf** 命令引用嵌套字符串 ， **printf** 命令中的格式字符串中的双引号必须使用**\"**符号来跳出，否则会引起语法错误。

当一个按钮被定义后，它会立即被加入到工具箱窗口中。每个按钮都被分配一个按钮号，并显示在工具箱窗口中。这个数字可以在删除工具窗中的某个按钮时使用。

DIR

语法	描述
DIR	显示当前模块的变量名称。
DIR *module*	显示指定模块的变量名称。
DIR *module* LINE **DIR *module**func* LINE**	显示指定模块或模块中某个函数的行号信息。
DIR *modulefunc***	显示模块中 ***func*** 函数的变量。
DIR BFUNC	显示所有预定义的µVision2 函数的名称。
DIR DEFSYM	显示用 **DEFINE** 命令创建的变量。
DIR FUNC	显示所有µVision2 函数的名称。
DIR LINE	显示当前模块的行数。
DIR MODULE	显示目标程序的所有模块名称。
DIR PUBLIC	显示所有全局变量的名称。
DIR SIGNAL	显示用户定义的信号函数的名称。
DIR UFUNC	显示用户定义函数的名称。
DIR VTREG	显示当前 CPU 支持的引脚寄存器的名称。

用 **DIR** 命令显示各种类型的符号变量。当不带参数的 **DIR** 命令显示当前模块名称。程序计数器指向地址的程序属于哪个模块，这个模块就是当前模块。**µVision2** 在装载目标程序时决定模块程序地址空间的分配。

µVision2 支持多种内部变量，这些变量的值可以使用 **DIR** 命令来显示。

语法	描述
module ***module**func***	显示指定模块或函数的变量名称。
BFUNC	显示所有预定义的µVision2 函数的名称。这些函数总是保持有效，不能删除和重定义。
DEFSYM	显示用 **DEFINE** 命令创建的变量。请参见 **DEFINE** 命令以获得更多信息。 使用这个选项，允许显示由'***DEFINE <type><name>***'命令创建的变量。
FUNC	显示所有当前定义的µVision2 函数的名称和原型。包括：预定义的函数、用户定义的函数和信号函数。 注意：µVision2 函数和你的目标程序中的函数是不一样的。
LINE ***module* LINE**	显示指定模块或函数的行号信息。

*module**func* LINE	
MODULE	在已载入目标程序条件下，显示目标程序的所有模块名称。
PUBLIC	显示所有全局变量的名称。这些变量在汇编语言中带有 **PUBLIC** 属性。非静态的 C 变量也归于全局变量一类。
SIGNAL	显示信号函数的名称。信号函数是用户函数，用于在背景运行。用信号函数产生 port 的输入。
UFUNC	显示所有用户定义函数的名称。用户函数即为用户定义的函数。
VTREG	显示当前 CPU 支持的引脚寄存器的名称。

示例：

以下例子使用 **C51** 编译包中的 **MEASURE** 例程创建。

DIR MODULE

```
>DIR MODULE      /* 显示所有模块名称 */
  MEASURE
  MCOMMAND
  GETLINE
  ?C_FPADD
  ?C_FPMUL
  ...
```

DIR *module*

```
>DIR \MEASURE    /* 显示模块'MEASURE'中各函数名 */
 MODULE: MEASURE
    C:0x000000 . . . . _ICE_DUMMY_ . . uint
     FUNCTION: {CvtB} RANGE: 0xFF03B7-0xFF07E5
      C:0x000000 . . . . _ICE_DUMMY_ . . uint
     FUNCTION: {CvtB} RANGE: 0xFF000B-0xFF000D
      C:0x000000 . . . . _ICE_DUMMY_ . . uint
     FUNCTION: SAVE_CURRENT_MEASUREMENTS RANGE: 0xFF000E-0xFF0069
     FUNCTION: TIMER0 RANGE: 0xFF006A-0xFF0135
      D:0x00000F . . . . i . . uchar
     FUNCTION: _READ_INDEX RANGE: 0xFF0136-0xFF01BF
      D:0x00003F . . . . buffer . . ptr to char
      D:0x000042 . . . . index . . int
      D:0x000007 . . . . args . . uchar
     FUNCTION: CLEAR_RECORDS RANGE: 0xFF01C0-0xFF01EE
      D:0x000006 . . . . idx . . uint
```

```
    FUNCTION: MAIN RANGE: 0xFF01EF-0xFF03B6
     I:0x000067 . . . . cmdbuf . . array[15] of char
     D:0x00003C . . . . i . . uchar
     D:0x00003D . . . . idx . . uint
```

DIR *module* LINE

```
>DIR \MEASURE LINE     /*显示 MESURE 模块的行号信息 */
 MODULE: MEASURE
C:0x000E . . . . #87
C:0x000E . . . . #88
C:0x003A . . . . #89
C:0x0049 . . . . #90
...
C:0x03B6 . . . . #291
C:0x03B6 . . . . #292
```

DIR PUBLIC

```
>DIR PUBLIC            /* 显示所有全局变量 */
  B:0x000640 . . . . T2I0 . . bit
  B:0x000641 . . . . T2I1 . . bit
  ...
  D:0x000023 . . . . current . . struct mrec
  C:0x0007CD . . . . ERROR . . array[16] of char
  X:0x004000 . . . . save_record . . array[744] of struct mrec
  C:0x00000E . . . . save_current_measurements . . void-function
  C:0x0001EF . . . . main . . void-function
  C:0x00047E . . . . menu . . array[847] of char
  D:0x000030 . . . . setinterval . . struct interval
  ...
  B:0x000601 . . . . IEX2 . . bit
  B:0x000600 . . . . IADC . . bit
```

DIR VTREG

```
>DIR VTREG           /*显示引脚寄存器及其值*/
    PORT0:  uchar, value = 0xFF
    PORT1:  uchar, value = 0xFF
    PORT2:  uchar, value = 0xFF
    PORT3:  uchar, value = 0xFF
    PORT4:  uchar, value = 0xFF
    PORT5:  uchar, value = 0xFF
    PORT6:  uchar, value = 0xFF
    PORT7:  uchar, value = 0x00
```

```
    PORT8:  uchar, value = 0x00
     AIN0:  float, value = 0
     AIN1:  float, value = 0
     AIN2:  float, value = 0
     AIN3:  float, value = 0
     AIN4:  float, value = 0
     AIN5:  float, value = 0
     AIN6:  float, value = 0
     AIN7:  float, value = 0
     AIN8:  float, value = 0
     AIN9:  float, value = 0
    AIN10:  float, value = 0
    AIN11:  float, value = 0
     SOIN:   uint, value = 0x0000
    SOOUT:   uint, value = 0x0000
     S1IN:   uint, value = 0x0000
    S1OUT:   uint, value = 0x0000
    VAGND:  float, value = 0
    VAREF:  float, value = 5
     XTAL:  ulong, value = 0xB71B00
   PE_SWD:  uchar, value = 0x00
    STIME:  uchar, value = 0x00
```

DIR

```
>$ = MAIN     /* 设置当前运行点为 main() */
>DIR          /* 显示当前模块的变量信息*/
 FUNCTION: MAIN RANGE: 0xFF01EF-0xFF03B6
  I:0x000067 . . . . cmdbuf . . array[15] of char
  D:0x00003C . . . . i . . uchar
  D:0x00003D . . . . idx . . uint
```

DIR DEFSYM

```
>DIR DEFSYM   /* 显示由 'DEFINE <type> <name>'命令定义的变量 */
            word00:    int, value = 0x0000
            byte00:   char, value = 0x00
           dword00:   long, value = 0x0
           float00:  float, value = 0
```

DIR FUNC

```
>DIR FUNC     /* 显示预定义µVision2 函数*/
predef'd:   void  MEMSET (ulong, ulong, uchar)
predef'd:   void  TWATCH (ulong)
```

```
predef'd:    int  RAND (uint)
predef'd:  float  GETFLOAT (char *)
predef'd:   long  GETLONG (char *)
predef'd:    int  GETINT (char *)
predef'd:   void  EXEC (char *)
predef'd:   void  PRINTF (char *, ...)
```

DISPLAY

语法	描述
DISPLAY ***startaddr, endaddr***	在存储器窗口（如果已打开的话）或命令窗显示从地址 ***startaddr*** 到 地址 ***endaddr*** 范围存储器的内容。

DISPLAY 命令在存储器窗口（如果已打开的话）或命令窗显示一定段地址的存储器内容。存储器区域显示为 **HEX** 或 **ASCII** 格式。也可在从主菜单中的 **View - Memory Window** 菜单项打开存储器窗口，显示其内容。

存储器内容显示每行由第一个字节的地址开始，最多显示 **16** 个字节 **HEX** 数，以及对应每个 **HEX** 数的 **ASCII** 字母。点**(".")**表示非打印字母。

如果地址参数 ***startaddr*** 和 ***endaddr*** 被省略，存储器内容显示从上一个 **DISPLAY** 命令显示的存储器地址结尾处开始显示，如果前面并没有使用过 **DISPLAY** 命令，将从程序存储器的 **0x0000** 地址开始显示。

如果命令带有地址参数 ***startaddr*** 和 ***endaddr***，存储器显示从 ***startaddr*** 到 **endadd** 的内容，或显示到下一个 **256** 字节。

对 **8051** 和 **251** 类型的单片机，地址必须加前缀以指定存储器类型。以下列出所有的存储器类型。例如。**X:0x0000** 指的是外部数据存储区 **XDATA** 的地址 **0x0000** 地址。

存储器类型	说明
B	位寻址 RAM 存储器(**BIT**).
C	代码存储器(**CODE**).
CO	常量存储器(251 **CONST**).
D	8051 内部直接寻址 RAM 存储器(**DATA**).
EB	扩展位寻址 RAM 存储器(251 **EBIT**).
ED	扩展内部直接寻址 RAM 存储器(251 **EDATA**).
HC	海量常量存储器(251 **HCONST**).
I	8051 内部间接寻址 RAM 存储器(**IDATA**).
X	外部数据存储器 (**XDATA**).

示例

```
>D main                    /* 显示从 main 函数开始的数据 */

>D X:0,0x100               /* 显示 256 字节的外部存储器数据 */
>                          /* 地址从 0 开始          */

>D menu                    /* 从 menu 开始显示       */

>D save_record, save_record + 0x2F
X:4000  62 62 62 62 62 62 62 62-62 62 62 62 62 62 62 62  bbbbbbbbbbbbbbbb
X:4010  62 62 62 62 62 62 62 62-62 62 62 62 62 62 62 62  bbbbbbbbbbbbbbbb
X:4020  62 62 62 62 62 62 00 00-00 00 00 00 00 00 00 00  bbbbbb..........
```

ENTER

语法	说明
ENTER ***type address = expr , expr …***	用指定数据类型的表达式(***expr***)改变在地址 ***address*** 的内容。

ENTER 命令可以改变从指定地址开始的存储器内容。该命令支持以下数据类型。

类型	说明
CHAR	有符号或无符号字符型。
DOUBLE	双精度浮点型。
FLOAT	浮点型。
INT	有符号或无符号整数。
LONG	有符号或无符号长整数

可以指定多重表达式，由逗号**(",")**隔开，表达式被转换成指定的数据类型，并保存在存储器中。

示例

```
>E CHAR x:0 = 1,2,"-µVision2-"                    /*输入字母 */
>D x:0
 X:0000  01 02 2D 64 53 63 6F 70 65 2D 00 00 00 00 00 00   ..-µVision2-......

>E FLOAT x:0x2000 = 3,4,15.2,0.33                 /* 输入浮点数    */
```

Esc

当你在命令窗键入 **Esc** 命令时， µ**Vision2** 停止运行目标程序。在停止目标程序后，寄存器窗口、变量监视窗口、反汇编窗口以及其它窗口会被更新以反应最新的 **CPU** 状态。

注
不可在 **exec** 预定义函数中使用 **Esc** 命令。要在一个调试函数停止程序运行，可以设系统变量**_BREAK_**为 **1**。

EVALUATE

语法	说明
EVALUATE ***expression***	使用 decimal, octal, HEX, 和 ASCII 格式显示表达式的结果。

EVALUATE 命令计算指定的表达式，并用 **decimal, octal, HEX** 和 **ASCII** 格式显示出结果。不带 **EVALUATE** 命令输入表达式仅以当前的数字格式显示，当前的数字格式要由 **RADIX** 系统变量选择。表达式可以包含几个子表达式，由逗号隔开。

示例

```
>eval -1
16777215T 77777777Q 0xFFFFFF '....'

>eval intcycle
0T 0Q 0x0 '....'

>intcycle = 0x12
>eval intcycle
18T 22Q 0x12 '....'

>eval 'a'+'b'+'c'
294T 446Q 0x126 '...&'

>eval main
16712175T 77600757Q 0xFF01EF '....'

>eval save_record[1].time
81931T 240013Q 0x1400B '..@.'

>eval save_record[1].time.sec
0T 0Q 0x0 '....'

>save_record[1].time.sec = 1
>eval save_record[1].time.sec
1T 1Q 0x1 '....'

>eval save_record[1].time.sec = 0
0T 0Q 0x0 '....'
```

EXIT

语法	说明
EXIT	停止µVision2 调试器模式。

EXIT 命令停止µ**Vision2** 调试器模式。

这个命令不允许作为 **exec** 函数的参数。

GO

语法	说明
GO ***startaddr, stopaddr***	从指定的 ***startaddr***（如果指定的话）开始运行程序，运行到 ***stopaddr***（如果指定的话）。

GO 命令使µ**Vision2** 你的目标程序。

键入命令后，程序将从 ***startaddr*** 开始运行。如果没有指定 ***startaddr***，则从当前程序计数器开始运行。

目标程序停止在 ***stopaddr*** 地址上。如果指定 ***stopaddr***，µ**Vision2** 会在这个地址设置一个临时断点，在程序停止时会删除该断点。如果没有指定 ***stopaddr***，目标程序只在运行到一个断点或在点击调试窗口的停止按钮才会停止运行。

目标程序停止运行后，寄存器、监视窗、调试窗口和其它窗口会更新以反映当前 **CPU** 状态。

注意
当使用条件断点时，µ**Vision2** 必须在每运行一条指令后判断断点条件。因此，即使用 **GO** 命令开始运行，µ**Vision2** 也是采用单步模式运行目标程序。

示例

```
>G,main          /*从当前程序计数器开始运行，到"main"停止 */
>G             /*从当前程序计数器开始运行，使用 Ctrl+C 或断点停止运行*/
```

INCLUDE

语法	说明
INCLUDE ***path filename***	打开文件 ***filename***，并读取文件或按µVision2 命令操作该文件。

INCLUDE 命令指定一个文件，并从文件中按行读出命令让µ**Vision2** 执行。利用 **INCLUDE** 命令，你可以在µ**Vision2** 中重复地进行某些操作。例如，你可能想创建一个 **INCLUDE** 文件，并使用该文件进行一系统操作：载入目标程序，并运行到 **main** 函数停止，创建一个工具箱按钮，创建几个用户函数。

INCLUDE 文件可以相互嵌套至**4**层。必须使用 **INCLUDE** 命令停止目标程序的运行。

示例

```
INCLUDE measure.ini
```

KILL

语法	描述
KILL BUTTON ***number***	删除一个工具箱按钮。
KILL FUNC *	删除所有µVision2 函数。
KILL FUNC ***function_name***	删除名为 ***function_name*** 的µVision2 函数。

KILL 命令可以删除前面定义的工具箱按钮和µ**Vision2** 函数。

KILL BUTTON 命令删除前面定义的工具箱按钮，当使用该命令时，必须指定需删除按钮的序号。序号显示在工具箱窗口每个按钮的前面。

KILL FUNC * 命令删除所有前面定义的µ**Vision2** 函数，但不会删除预定义µ**Vision2** 函数。

The KILL FUNC 命令删除指定名称的µ**Vision2** 函数或信号函数。

示例

```
>KILL FUNC ANALOG      /* 删除用户函数 "analog" */
>KILL FUNC myregs      /*删除用户函数"myregs" */
>KILL FUNC *           /*删除所有用户函数 */

>KILL BUTTON 3         /* 删除 3 号工具箱按钮*/

>KILL BUTTON 1         /* 删除 1 号工具箱按钮*/
```

LOAD

语法	说明
LOAD *path* ***filename*** **NOCODE**	载入一个绝对目标文件（absolute object file）或 Intel HEX 文件。 **NOCODE** 指引µVision2 仅载入符号信息，忽略代码记录。除先前已载入监控器（MON51, MON251, 或 MON166）的 CPU 驱动情况外，**NOCODE** 的功用只是相对的。

LOAD 命令可以让µ**Vision2** 调试器载入指定的一个文件。你可以在µ**Vision2** 调试开始运行时就载入当前项目的目标文件，只需 **Options for Target – Debug** 对话框中选中 **Load Application at Startup** 选项。

Absolute Object File

由 linker/locator 产生。当文件带调试信息进行转换时，文件包含有全部的符号调试信息、类型信息和行号。

Intel HEX

Intel HEX 文件是由目标到 Hex 格式转换器程序产生的。文件不含符号调试信息、没有类型信息和行号信息。仅支持汇编级的程序测试，而不支持源码级和符号调试。

注意

µ**Vision2** 分析指定文件的内容以确定文件类型。如果不能确定文件类型，就不能载入，并显示错误信息。

示例

```
LOAD C:\KEIL\C51\EXAMPLES\MEASURE\MEASURE
```

上面的命令行从 **c:\keil\c51\examples\measure** 目录载入 **measure** 文件。 µ**Vision2** 也会在这个目录中搜寻源文件。

```
LOAD MYPROG.HEX
```

这个载入命令行 **myprog.hex** 文件。

LOG

语法	说明
LOG > ***path filename***	创建一个名为 ***filename*** 的日记文件。命令窗口的输出将写入该文件。
LOG >> ***path filename***	打开一个已存在的名为 ***filename*** 的日记文件，并添加日记信息。如果该文件不存在，将创建文件。命令窗口的输出将添加到该文件中。
LOG	显示日记文件状态。
LOG OFF	关闭日记文件。

使用 **LOG** 命令可以创建、添加、检查状态和关闭一个日记文件。显示在命令窗口的输出会拷贝到日记文件。文件名可以指定驱动器号和路径。文件可以是一个字符串，如：**c:\usr\tmp\logfile**。

示例

```
LOG >C:\TMP\dslog                    /* 创建一个日记文件        */
LOG                                  /* 查询日记文件状态 */
 command log file: C:\TMP\dslog
LOG OFF                              /* 关闭日记文件 */
```

MAP

语法	说明
MAP	显示当前存储器映射。
MAP start, end READWRITEEXECVNM	按指定的存取方式映射指定的存储器段(***start-end***)。
MAP start, end CLEAR	清除一个存储器段映射。

µ**Vision2** 调试的目标程序会存取使用存储器。在大多数应用中，µ**Vision2** 使用目标程序的变量信息自动建立存储器映射。**MAP** 命令定义程序需要使用，但 µ**Vision2** 不会自动检测的存储器。

当运行目标程序时，µ**Vision2** 检查每一个存储器存取动作以确定是否超出存储器映射。在进行非法的存取操作时，µ**Vision2** 会报告一次存取错误。这将有助于定位和纠正程序中的存储器使用问题。

注意
如果程序使用存储器映射的 **I/O** 设备，或通过指针动态存取存储器，你可能需要改变存储器映射。

MAP 命令可以指定一个地址段，并指明存取方式，包括读**(READ)**、写**(WRITE)**和执行**(EXEC)**。存储器映射支持 **1** 个字节长度的段。

VNM 选项使得µ**Vision2** 认为指定的存储段为冯诺依曼式存储器结构。当使用 **VNM** 选项时，µ**Vision2** 会重叠外部数据存储器和代码存储器。对外部数据存储器的写操作也会改变代码存储器的内容。用 **VNM** 选项指定的地址范围可以不在代码区范围之内，可以小于 **64K**。 指定的地址范围必须是外部数据区域。

使用不带参数的 **MAP** 命令时，会显示目标程序当前的存储器映射。这条命令可以方便你检查存储器映射设置。

CLEAR 选项移除上一次使用 **MAP** 命令定义的地址段。

当使用µ**Vision2** 时，以下存储器映射已经定义。

CPU	地址段	存取类型
8051 Family	0x000000-0x00FFFF (**DATA**)	**READ WRITE**
	0x010000-0x01FFFF (**XDATA**)	**READ WRITE**
	0xFF0000-0xFFFFFF (**CODE**)	**EXEC READ**
251 Family	0x000000-0x00FFFF (**DATA**)	**READ WRITE**

	0x010000-0x01FFFF (**XDATA**)	**READ WRITE**
	0xFF0000-0xFFFFFF (**CODE**)	**EXEC READ**
166 Family	0x000000-0x00FFFF	**EXEC READ WRITE**

µVision2 最多支持 **16MB** 的存储器空间。可分为 **256** 个段，每个段为 **64K**。**µVision2** 对 **8051** 和 **251** 的存储器空间的缺省分段设置如下表：

参数值	存储空间
0x00	MCS® 51 **DATA** segment, 0x00:0x0000—0x00:0x00FF. MCS® 251 **EDATA** segment, 0x00:0x0000-0x00:0xFFFF.
0x01	MCS® 51 and MCS® 251 **XDATA** segment 0x01:0x0000-0x01:0xFFFF.
0x80-0x9F	MCS® 51 and MCS® 251 Code Bank 0 through Code Bank 31. 0x80 Code Bank 0, 0x81 Code Bank 1, etc.
0xFE	MCS® 51 and MCS® 251 **PDATA** segment 0xFE:0x0000-0xFE:0x00FF.
0xFF	MCS® 51 and MCS® 251 **CODE** segment 0xFF:0x0000-0xFF:0xFFFF.

尽管**µVision2** 最大可以支持 **16MB** 的目标程序存储器，只有要求映射的存储器段才会被映射。**µVision2** 要求存储器映射中的每个块都有两个拷贝。一个放置用来读写、执行的数据。另一个放置特殊的的属性数据，如存取类型，用于代码覆盖和性能分析功能的信息。由于这个原因，映射大容量的存储器可能会减速慢 **µVision2** 的运行速度，因为需要进行内存与硬盘的数据交换。

RESET MAP 命令清除所有映射段，恢复到如上所示的缺省设置。具体请参见 **RESET** 命令。

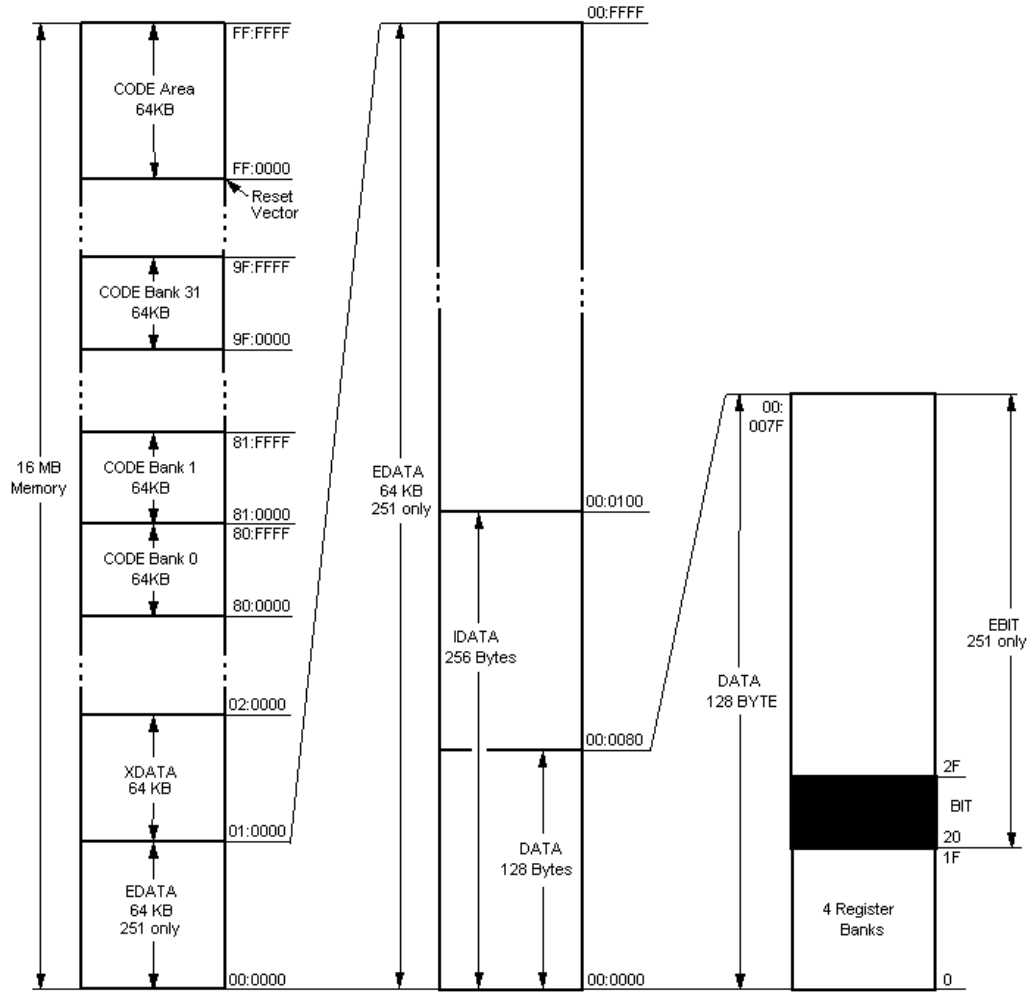
可以使用存储器映射窗查看和改动存储器映射。在主菜单的 **Debug - Memory Map….**项打开这个窗口。

注

对于 **8051** 系列 **CPU** 来说，映射小于 **256** 字节的外部数据区可能会产生问题。**8051** 通过使用 **MOVX @Ri** 指令，支持 **256** 字节 **XDATA** 的快速存取方法。当映射大于 **256** 字节的外部数据，**µVision2** 使用 **P2** 和 **P0** 实现 **16** 位的地址。对于少于 **256** 字节的映射，**µVision2** 仅使用 **P0**，因此，**XDATA** 必须从一个段头处开始（缺省为 **0x01:0x0000** 至 **0x01:0x00FF**）。

8051 和 251 单片机的存储器映射

下图显示出 **8051** 和 **80251** 单片机的存储器映射。



示例

```
MAP 0x10000,0x1FFFF read write          /* 开启 64K XDATA RAM   */
RESET MAP                               /* 复位映射                */
MAP 0xFF0000,0xFFFFFF exec read         /* 开启缺省代码段 */
MAP 0x018000,0x01FFFF read write VNM    /* 设为冯诺依曼存储器 */
```

MODE

语法	说明
MODE COM*x*, *baudrate*, *parity*, *databits*, *stopbits*	改变串口设置。

µVision2 可以让你改变 PC 机的串口设置。***baudrate*** 参数必须是一个有效的波特率值，如 1200, 2400, 9600, 或 19200。***parity*** 表示校验位，为 0 时表示无检验位，1 为奇校验，2 为偶校验。Databits 为 8 时，表示有 8 个数据位，为 7 表示为 7 个数据位。Stopbits 为 1 时，表示 1 个停止位；为 15，表示为 1½ 个停止位；而为 2 时表示为 2 个停止位。

MODE 命令也可以与 ASSIGN 命令联合使用，以设置仿真 CPU 的串口输入输出渠道。

示例

```
>MODE COM2, 19200, 0, 8, 1    /* 设置 COM2 的波特率为 19,200 bps, */
                              /*                无校验,*/
                              /*                1 个停止位 */
```

OSTEP

语法	说明
OSTEP	步出当前函数。如果没有调用函数，µVision2 会报告产生错误。

OSTEP 命令从当前的程序计数器开始运行，并停止在调用当前函数的指令的后一个指令上。**µVision2** 内部维持着一个当前调用嵌套函数列表，包括函数调用指令的程序地址。当在列表中没有调用的函数的情况下使用了 **OSTEP** 命令，**µVision2** 会报告错误。

如欲了解其它的跟踪命令，请参考 **PSTEP** 和 **TSTEP** 命令。

示例

```
>0     /* 步出当前函数 */
```

Performance Analyzer

语法	说明
PerformanceAnalyze	显示所有当前定义的性能分析段，以及他们占用的运行时间。
PerformanceAnalyze ***start , end***	定义一个新的段，加入到性能分析器中。
PerformanceAnalyze KILL *	删除所有当前定义的性能分析段。
PerformanceAnalyze KILL ***item , item…***	删除一个或多个当前定义的性能分析段。
PerformanceAnalyze RESET	复位性能分析器。

你可以使用µVision2 中的性能分析器来调整你的程序以达到最好的性能。首先指定你打算进行分析的部分，性能分析器在程序运行时会收集这些部分的运行情况以进行统计。你选择加以分析的部分是最快、最慢以及其平均执行时间都会被保持和记录。

性能分析器最多可分析 256 段代码。他会记录每段代码的运行次数和运行时间。

段指的是地址段。通常，它开始于一个函数的第一条指令，结束于函数的最后一条指令。但是，你也可以指定目标程序中的任何一部分指令为一个段。

程序运行时，性能分析器的结果显示在性能分析器窗口中。

以下是与性能分析器使用有关的命令。

PA

PA 命令不带任何参数时，显示每个定义的性能分析地址段、索引号（用来删除特定的段）、执行次数，以及最小、最大和总执行时间（用时钟周期表示）。

PA *start* , *end*

当 **PA** 命令带有目标程序中的函数名或一个地址段时，它创建一个新的地址段中入到性能分析器。如 ***start*** 是一个函数名，µVision2 自动获取该函数的开始、结束地址。如果 ***start*** 是一个地址，***end*** 就必须指定段的结束地址。你的目标程序在编译时必须启动全部的调试信息，以使性能分析的效果最佳。

注意
性能分析器地址段只能有唯一的入口和出口，不能包括 **RET** 指令，新的地址段不可以与老的地址段重叠。

PA KILL *

PA KILL * 命令从性能分析器中删除所有的地址段。

PA KILL *item*

PA KILL 命令后带一个索引号时，从性能分析器中删除与索引号对应的地址段。**µVision2** 会为每一个加入到性能分析器中的址段分配一个索引号。可以用 **PA** 显示所有的索引号。

PA RESET

PA RESET 命令删除所有已定义地址段的记录信息。你可以在目标程序运行了一段时间后，运行这个命令开始信息记录。

你也可以使用 **Setup Performance Analyzer** 对话框来定义程序地址段。在主菜单的 **Debug – Performance Analyzer…**项可以打开这个对话框。

示例

```
>PA main                          /* 为 main()定义一个段    */
>PA timer0                        /*为 timer0 ()定义一个段*/
>PA clear_records                 /* 定义多个段    */
>PA measure_display
>PA save_current_measurements
>PA read_index
>PA set_time
>PA set_interval
>
>PA                               /* 显示所有 PA 段  */
  0: main: (FF01EF-FF03B6)        /* FF01EF = C:0x01EF          */
  1: timer0: (FF006A-FF0135)
  2: clear_records: (FF01C0-FF01EE)
  3: measure_display: (FF07E7-FF084A)
  4: save_current_measurements: (FF000E-FF0069)
  5: read_index: (FF0136-FF01BF)
  6: set_time: (FF084B-FF08CA)
  7: set_interval: (FF08CB-FF09A5)

/* After execution of the user program ... */

>PA                               /*显示段及其状态*/
  0: main: (FF01EF-FF03B6)
     count=1, min=-1, max=0, total=167589
  1: timer0: (FF006A-FF0135)
     count=2828, min=33, max=254, total=226651
  2: clear_records: (FF01C0-FF01EE)
```

```
     count=1, min=27086, max=27086, total=27086
  3: measure_display: (FF07E7-FF084A)
     count=10, min=19495, max=19503, total=185027
  4: save_current_measurements: (FF000E-FF0069)
     count=491, min=205, max=209, total=100665
  5: read_index: (FF0136-FF01BF)
  6: set_time: (FF084B-FF08CA)
  7: set_interval: (FF08CB-FF09A5)
>
>PA KILL 7                      /* 删除 set_interval       */
>PA KILL 6                      /*删除 set_time           */
>PA KILL 5                      /*删除 read_index         */
>PA
  0: main: (FF01EF-FF03B6)
     count=1, min=-1, max=0, total=167589
  1: timer0: (FF006A-FF0135)
     count=2828, min=33, max=254, total=226651
  2: clear_records: (FF01C0-FF01EE)
     count=1, min=27086, max=27086, total=27086
  3: measure_display: (FF07E7-FF084A)
     count=10, min=19495, max=19503, total=185027
  4: save_current_measurements: (FF000E-FF0069)
     count=491, min=205, max=209, total=100665
>
>PA RESET                       /* 清除所有记录信息 */
>PA
  0: main: (FF01EF-FF03B6)
  1: timer0: (FF006A-FF0135)
  2: clear_records: (FF01C0-FF01EE)
  3: measure_display: (FF07E7-FF084A)
```

PSTEP

语法	说明
PSTEP ***expression***	执行或步越 ***expression*** 一个程序行或一条汇编指令。**PSTEP** 命令步越函数或子程序。

PSTEP 命令执行一条或多条源级指令或汇编指令，这取将于调试窗口的显示模式。**PSTEP** 命令并不步入函数中。如想了解如何步入调用的子函数中，请参见 **TSEP** 命令。

The PSTEP 命令步越源级指令还是汇编指令，取决于调试窗口的显示模式。下表列出调试窗口的显示模式。

显示模式	说明
Assembly（汇编）	**PSTEP** 步越汇编指令，但不步入子程序或函数。
Mixed（混合）	**PSTEP** 步越汇编指令，但不步入子程序或函数。
High-Level Language（高级语言）	**PSTEP** 步越 C 或 PL/M-51 程序中的一行程序，但不步入子程序或函数。

示例

```
>P 100      /* 执行 100 步，忽略调用*/
>P          /*执行 1 步，忽略调用*/
```

RESET

语法	说明
RESET	复位仿真或目标 CPU。
RESET MAP	复位映射（**MAP** ）分配。.
RESET ***variable***	复位 **SET** 变量。

RESET 命令有三种不同的用法。

当 **RESET** 命令不带附加参数时，µ**Vision2** 调试器复位仿真或目标 **CPU**。这等于一次处理器复位，复位后程序计数器设为 **0x0000**，所有特殊寄存器被复位至缺省值。目标程序仍然载入所有的调试信息。任何启动的信号函数都会被停止。

RESET MAP 命令复位映射分配，以及清除任何载入的目标程序的存储器和调试信息。可参见 **MAP** 命令获取更多信息。

Example

```
>RESET                    /* 复位 CPU          */
>RESET MAP                /* 复位寄存器映射分配*/
```

SCOPE

语法	说明
SCOPE *module**function*	显示指定模块或函数的地址范围。

SCOPE 命令显示：

- 目标程序的所有函数的地址范围（如果没有输入模块和函数名的话），
- 一个程序模块的所有函数的地址范围（仅指定模块名），
- 单个函数的地址范围（指定了模块和函数名）。

当载入一个带有调试信息的程序时，µ**Vision2** 调试器创建一个内部地址分配和变量对应信息表，并自动选出不合格的变量。

示例

在下面的例子中，缩进行反映了每个模块的源程序的布局。

```
>scope \measure\main                    /* 显示 main()的范围       */
MAIN RANGE: 0xFF01EF-0xFF03B6
>
>scope \measure                         /* 显示模块 'measure'的范围 */
MEASURE
  {CvtB} RANGE: 0xFF03B7-0xFF07E5       /* µVision2 哑范围段       */
  {CvtB} RANGE: 0xFF000B-0xFF000D
  SAVE_CURRENT_MEASUREMENTS RANGE: 0xFF000E-0xFF0069
  TIMER0 RANGE: 0xFF006A-0xFF0135
  _READ_INDEX RANGE: 0xFF0136-0xFF01BF
  CLEAR_RECORDS RANGE: 0xFF01C0-0xFF01EE
  MAIN RANGE: 0xFF01EF-0xFF03B6
>
>scope                                  /* 显示所有段的范围          */
MEASURE
  {CvtB} RANGE: 0xFF03B7-0xFF07E5
  {CvtB} RANGE: 0xFF000B-0xFF000D
  SAVE_CURRENT_MEASUREMENTS RANGE: 0xFF000E-0xFF0069
  TIMER0 RANGE: 0xFF006A-0xFF0135
  _READ_INDEX RANGE: 0xFF0136-0xFF01BF
  CLEAR_RECORDS RANGE: 0xFF01C0-0xFF01EE
  MAIN RANGE: 0xFF01EF-0xFF03B6
MCOMMAND
  {CvtB} RANGE: 0xFF09A6-0xFF0A23
```

```
  MEASURE_DISPLAY RANGE: 0xFF07E7-0xFF084A
  _SET_TIME RANGE: 0xFF084B-0xFF08CA
  _SET_INTERVAL RANGE: 0xFF08CB-0xFF09A5
GETLINE
  _GETLINE RANGE: 0xFF0A24-0xFF0A87
?C_FPADD
?C_FPMUL
?C_FPDIV
?C_FPCMP
...
```

注意

{CvtB}块是由µ**Vision2** 创建的，是因为调试信息不够而产生的。对不带调试信息的库模块和汇编语言模块，以及有一个有效名称但没有范围的块来说，µ**Vision2** 通常都会创建一个**{CvtB}**块。

SAVE

语法	说明
SAVE ***path filename addr1, addr2***	用 HEX386 格式保存地址范围 ***addr1*** 至 ***addr2*** 的内容到名为 ***filename*** 的文件中。

SAVE 命令保存一个存储器映象到文件中。这个文件保存为 **HEX386** 格式。保存的是地址范围 ***addr1*** 到 ***addr2*** 的存储器内容。保存在文件里的内容可以通过 **LOAD** 命令再载入µ**Vision2** 调试器。

注意
不能保存 **8051** 和 **251** 内部 **RAM** 的 **I:0x80** 到 **I:0xFF** 范围的内容。

SET

语法	说明
SET ***variable***	显示一个 **SET** ***variable*** 的分配情况。
SET ***variable*** = "***string***"	将 ***string*** 赋于指定指定的 **SET** ***variable***。

SET 命令设置与预定义变量相关连的子符串。也可以用 **SET** 命令显示当前与一个预定义变量相关连的字符串。**RESET** 命令清除与一个预定义变量相关连的字符串。请参见 **RESET** 命令获取更多信息。

可以使用 **SET** 命令和 **RESET** 命令对以下预定义变量进行操作。

变量名称	说明
SRC	源码级调试需要的源文件或列表文件的搜索路径。一次最多可为 **SRC** 变量指定 10 个路径。

示例

当载入一个目标模块时，特定的路径就加入到 **SRC** 的搜寻路径中。

```
>LOAD \OBJS\MEASURE          /* 载入"MEASURE"模块    */
>SET SRC                     /* 显示分配给 SRC 的所有路径 */
\objs
```

可以加入附加的路径信息，以搜寻源文件和列表文件。

```
>SET SRC =\SRC                /*附加路径指定 */
>SET SRC                     /*显示分配给 SRC 的所有路径*/
\objs
\src
```

SIGNAL

语法	说明
SIGNAL KILL ***function_name***	停止指定的已启动的信号函数。
SIGNAL STATE	显示已启动的信号函数。

SIGNAL 命令可以显示或停止已启动的信号函数。

示例

```
>SIGNAL KILL ANALOG0          /* 停止 analog0 信号函数*/
```

以上的例子演示了如何停止一个已启动的信号函数。

```
>SIGNAL STATE
 0  idle     Signal = ANALOG0 (line 10)
```

以上的例子显示了启动的信号函数。显示的依次是函数号，函数状态（空闲或运行），函数名称（本例为 **ANALOG0**），最后执行的程序行。

SLOG

语法	说明
SLOG > *path filename*	创建一个名为 ***filename*** 的日志文件，串口窗的输入输出写入该文件。
SLOG >> *path filename*	打开一个已存在的名为 ***filename*** 的日志文件以添加信息。如果文件不存在，则创建文件。串口窗的输入输出写入该文件。
SLOG	显示日志文件的状态。
SLOG OFF	关闭日志文件。

使用 **SLOG** 命令创建、添加、检查状态或关闭一个日志文件。显示在串口窗的输入输出会被拷贝到这个日志文件。指定的文件名中可以有驱动器号和路径。可以以字符串格式输入文件名，如： **c:\usr\tmp\logfile.**

示例

```
SLOG >C:\TMP\dslog                  /* 创建一个新日志文件*/
SLOG                                /* 查询日志文件状态 */
 serial log file: C:\TMP\dslog
SLOG OFF                            /* 关闭日志文件*/
```

TSTEP

语法	描述
TSTEP ***expression***	执行和单步运行 ***expression*** 个程序行或汇编指令。**TSTEP** 命令步入函数或子程序。

TSTEP 命令执行一条或多条源级指令或汇编指令（取决于调试窗口的显示模式）。**TSTEP** 命令步入调用的函数。请参见 **PSTEP** 命令了解 **PSTEP** 命令的信息。

TSTEP 命令步越源级代码还是汇编指令，取决下面表格所列的调试窗口的显示模式。

显示模式	描述
Assembly	**TSTEP** 步越汇编指令，步入子程序和函数。
Mixed	**TSTEP** 步越汇编指令，步入子程序和函数。
High-Level Language	**TSTEP** 步越 C 或 PL/M-51 高级语言的程序指令，步入子程序和函数。

示例

```
>T 100          /* 执行 100 步 */
>T              /* 执行一步    */
```

UNASSEMBLE

语法	描述
UNASSEMBLE ***address***	在调试窗口显示反汇编代码。如果指定 ***address*** 的话，从 ***address*** 开始反汇编，否则从继续上次 **UNASSEMBLE** 命令的显示。

UNASSEMBLE 命令对代码存储器进行反汇编，并在 **Disassembly** 窗口显示反汇编代码。在 **Disassembly** 窗口反汇编代码的显示有两种模式：高级语言混合汇编语言模式，或汇编语言模式。你可以在反汇编窗口右击鼠标显示菜单选择显示的模式。

注意

为进行源码级调试，µ**Vision2** 调试器需要一个包含行号信息的目标模块。

示例

```
U main          /* 从 main()开始反汇编 */
U               /* 从上次 U 命令停止的地址继续反汇编*/
U C:0x0         /* 从地址 C:0x0000 开始反汇编*/
```

WATCHKILL

语法	描述
WATCHKILL *windownr*	删除监视窗口的一页中的所有监视点。

WATCHKILL 命令删除所有的在监视窗口中的已定义的监视点表达式。

windownr 参数指定监视窗口的页号。

示例

```
>WK 1      /* 删除所有的在监视窗口 1 中的已定义的监视点表达式*/
>WK 2
```

WATCHSET

语法	说明
WATCHSET ***windownr, expression , base***	定义一个监视点表达式，并加入到监视窗口中的一页。 ***base*** 参数指定使用的数字格式（10 或 16 进制）。

WATCHSET 命令可以定义显示在监视窗口的监视点表达式。在每一个程序执行命令后，如 GO、OSTEP、PSTEP、和 TSTEP 命令，监视窗口都会更新显示。

windownr 参数指定监视窗口的页号。你可以指定数字格式为 **10** 或 **16**，表达式的值将以这种格式显示。

示例

```
>WS interval,0x0a LINE
>WS save_record[0].analog
>WS save_record[0]
>WS sindex
```

译者有感

工欲善其事，必先利其器。

在翻译过程中，对 KEILC 的强大功能又增加了许多新的认识。