概　　况

软件特点:

- 可仿真调试 C 和汇编语言
- 即使不连接仿真器，也可以软件模拟运行
- 仿真速度最高达 64MHz，国内之最
- 编辑，编译，调试在一个窗口内完成
- 调试后，在编辑窗内光标跟踪显示变量值和名称，类型，长度，存储，地址等属性，国内首创。
- 创建项目管理，方便项目编译仿真
- 超大文件编辑，文件长度只受硬盘容量的限制
- 可在窗口自行设置仿真器频率，无须硬件跳线
- 变量可用鼠标快速加变量观察窗口，无须按键输入名称
- 编译错误定位
- 可将用户板上的程序装入仿真区调试或存盘
- 同时调试观察源文件和反汇编窗口及其他窗口
- 可仿真调试各种格式的文件（源文件，二进制文件，十六进制文件）

第一章　　软件的安装

1.1 计算机系统的最低要求：

操作系统：Windows 95/98/NT;
CPU: 486-100 以上;
内存：>= 32M;
硬盘剩余空间：20M

1.2 软件安装

lope 仿真环境采用了 keil c51 编译软件 (5.0 或者以上版本), 建议用户购买正版的 keil c51,将正版的 c51 必要的文件拷贝到相应的子目录下，覆盖演示版的软件即可。

对应的子目录如下：

正版 keil c51	LOPE 安装的文件夹
BIN 等目录	XXXX\kc51\BIN子目录
INC 子目录	XXXX\kc51\INC子目录
LIB 子目录	XXXX\kc51\LIB子目录

其中 XXXX 是安装的路径。

关于 keil c51 正版软件的更多信息，可在如下网站得到：

http://www.keil.com

1.2.1 将软件光盘插入光驱，运行 setup.exe文件。
1.2.2 按提示继续安装，
1.2.3 重新启动计算机。
1.2.4 在“开始”栏可以找到安装好的软件 “lope”。

第二章 LOPE 的界面

lope在中文 Windows95/98/2000之下运行，最充分地发挥了 32 位操作系统的优势,完美的体现了中文多窗口软件的风貌。

lope的集成软件包括主窗口，所有的菜单命令，快捷按钮， 源文件编辑器，各种仿真窗口等。在这个主窗口下，可以完成从编辑，编译直到仿真调试的所有操作。

为了充分发挥多窗口调试的优点,建议用户将显示器分辩 率设置为大于 800x600，现在的显示器和显示卡一般都能支持（在 Windows中“我的电脑／控制面板／显示器／设置／ 桌面区域”中移动箭头到 800x600，按“确定”，再按“保持”）

2.1 主窗口

C语言和汇编语言的主窗口基本相同 主要是编译和调试略有不同。

2.1.1 运行 lope后，显示 C 语言的主窗口如图 2.1。

选择软件仿真和硬件仿真，按 “确定”即可。

图 2.1

2.1.2 工具栏，图 2.2

图 2.2

通过工具栏的快捷键可快速操作程序，实际上是快速执行经常使用的菜单命令。

工具栏上安排有十三个快捷键，快捷键都带有提示功能，当 鼠标在这些按键上停留超过0.8秒，按键的附近将弹出一个黄色背景的小窗口，提示这个按键的作用。

十三个快捷键的作用如下：


运行指示，单步或跟踪运行闪一次，全速运行连续时闪动 。


打开源文件(C 或 ASM)；


保存正在编辑的源程序文件。


编译+连接一次完成，完成后才可仿真调试；


编译+连接项目。


单步运行；

 跟踪运行；

 cpu 复位；

 全速运行；

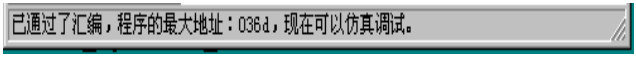
 暂时停止运行；

 运行到光标行；

 设置或清除断点；

2.1.3 状态条

状态条位于窗口的底部，用来显示程序运行时的状态或提示，例如显示编辑行号，提示编译出错信息等，示例如下：



2.1.4 主菜单

主菜单位于窗口的顶部，几乎所有的操作都可以用主菜单上 的命令实现。主菜单上的许多命令可以用热键快速操作，命令的右边指出了该键的名称。例如编辑菜单中的 “拷贝字符串 ”命令可以用“Ctrl+C”键来快速操纵。

主菜单可以用光标点击选中，也可以按 Alt+字母选中，例如上例编辑菜单可以用 “Alt+e”弹出。

主菜单各项命令的详细功能和操纵方法在后面逐一介绍。

2.2 窗口与排列

[窗口]：

为便于了解仿真情况，“窗口“项设有各种观察窗口，如图 2。23

图 2.23

2.2.1[调试工具栏]：作用是在主菜单内取消或显示工具栏。

2.2.2[项目窗口]：显示项目内容。

2.2.3[编译信息窗口]：显示编译后成功或出错信息。图 2.24

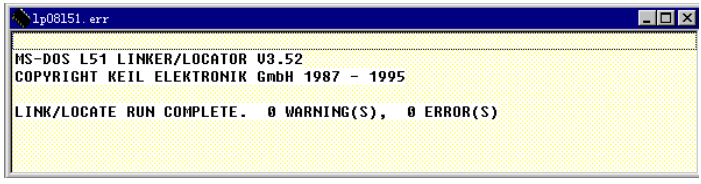


图 2.24

2.2.4[**反汇编窗口**]:**显示编译后反汇编状况**。反汇编窗口将程序仿真区的二进制代码翻译成汇编指令显示，反汇编窗口可以用 “窗口”命令打开或关闭，有时也会自动打开，当程序运行异常时，反汇编窗口会自动打开，显示当时的现场情况，让程序员分析异常的原因。

反汇编窗口通常是自动跟踪PC，一次反汇编一段，但不会清除前面已经显示的行。在源文件窗口单步，跟踪，断点等调试时，反汇编窗口也同步地跟踪显示对应的反汇编指令，除非关闭反汇编窗口。

如果是 C 语言源程序可能会对应好多行汇编指令，这时对应的是第一条反汇编指令。

如果是汇编语言源程序，则必定是一一对应。

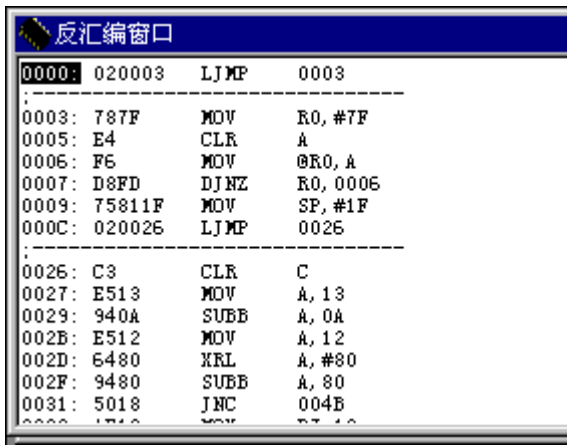
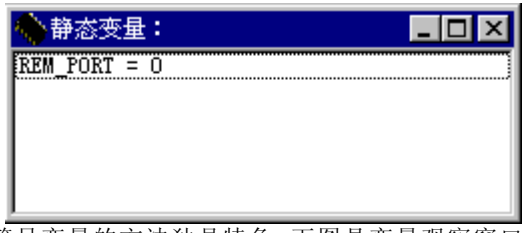


图 2.25

2.2.5[公共变量窗口]：显示公共变量。图 2.26

观察符号变量数值，是高级仿真器极重要的功能.LOPE观察



符号变量的方法独具特色，下图是变量观察窗口，其中“REM_PORT=0”显示了该变量当时的运行数值。

图 2.26

可用鼠标点击改变数值。

2.2.6[特殊功能寄存器窗口]：图 2.27

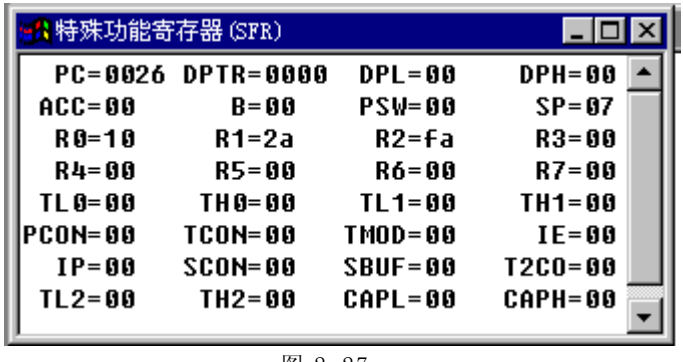


图 2.27

2.2.7[输入/输出口(I/O)]:图 2.28

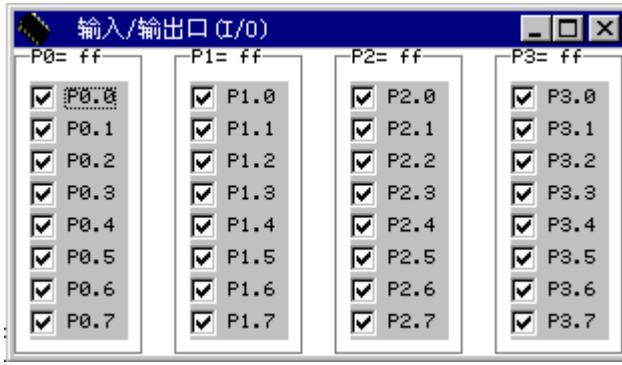


图 2.28

打勾为高电平，否则低电平。可以对准某一引脚，用鼠标点击改变逻辑电平，逻辑电平会立即反映到物理引脚上。

2.2.8[内部存储器(RAM)]:图 2.29

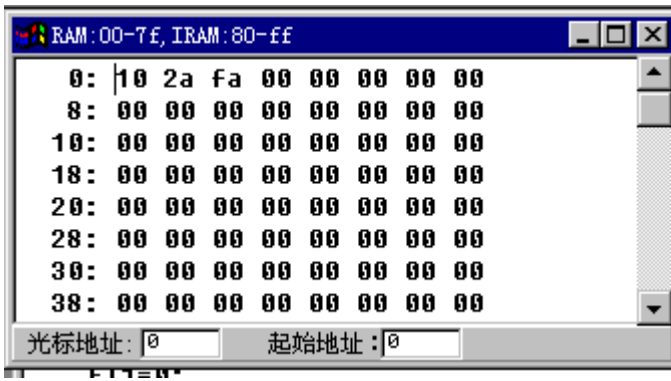


图 2.29

2.2.9[外部存储器(XRAM)]:图 2.10

图 2.10

2.2.10[程序存储器(CODE)]:图 2.11

图 2.11

2.2.11[堆栈存储器(SP)]:图 2.12

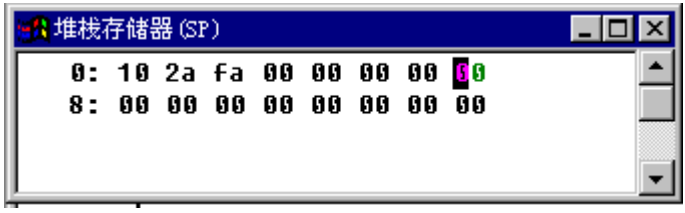


图 2.12

其中显示为绿色数据是当前堆栈数据。

2.2.12[位寻址存储器(BIT)]:图 2 .13

图 2.13

2.3 修改窗口中的数据

2.3.1 移动光标到某一数据直接按键输入。

2.3.2 对准数据双击鼠标，弹出对话框，以不同格式输入。

2.3.3 对于输入/输出口(I/O)，可以对准某一引脚，用鼠标点击改变逻辑电平，打勾为高电平，否则低电平。也可以按回车改变。逻辑电平会立即反映到物理引脚上。

2.3.4 数据窗口控制键

PageUp 上翻一页

Page Down 下翻一页

Home 翻到 0000 地址
End 翻到最大地址页

2.4 窗口的排列

窗口的排列可用菜单命令 “**排列**” 选择，可以分为三种不同的排列方式。

菜单如下：图 2.14

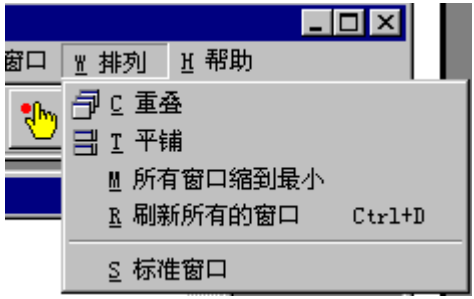


图 2.14

2.4.1[重叠]：重叠的效果如图 2.15.编辑窗口占主窗口一部分。

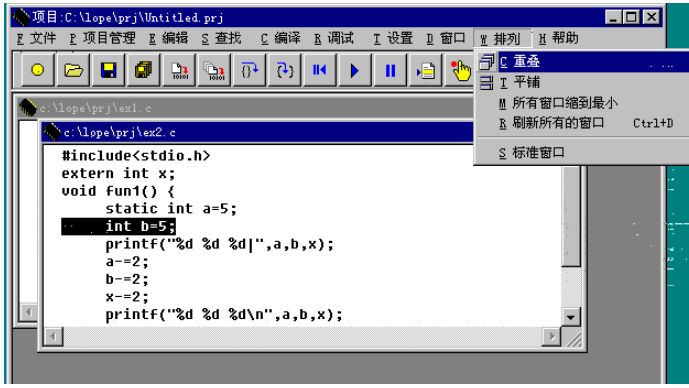


图 2.15

2.4.2[平铺]：所有窗口平铺在主窗口内。.

如：一个编辑窗口和一个反汇编窗口，点击 [平铺]将两个窗口平铺在一个窗体内。图 2.16

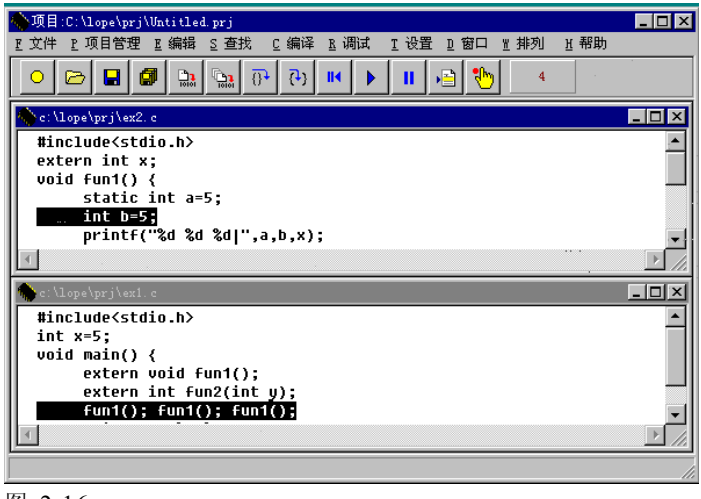


图 2.16

2.4.3[所有窗口缩到最小]：点击此项缩小所有窗口。图 2.17

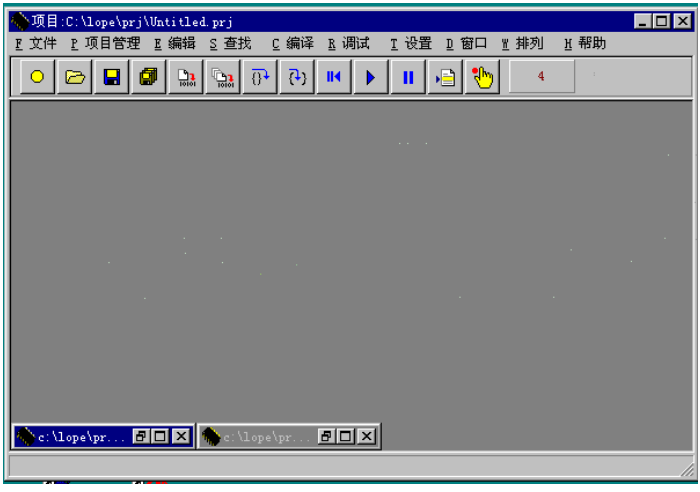


图 2.17

2.4.4[刷新所有窗口]：将所有窗口重新刷新。

2.4.5[标准窗口]：恢复显示标准窗口。

第三章　项目管理

3.1 项目管理的建立

为方便用户对软件的维护与管理，将所有构造一个程序的信息 都放入一个项目文件中，这是一个带 .prj 扩展名的文件。它包含：1 编辑器.连接器.制作及库选项。 2 组成该项目的所有文件的列表。

3.2 项目管理栏

项目管理栏有五个选项，如图 3.1,分述如下：

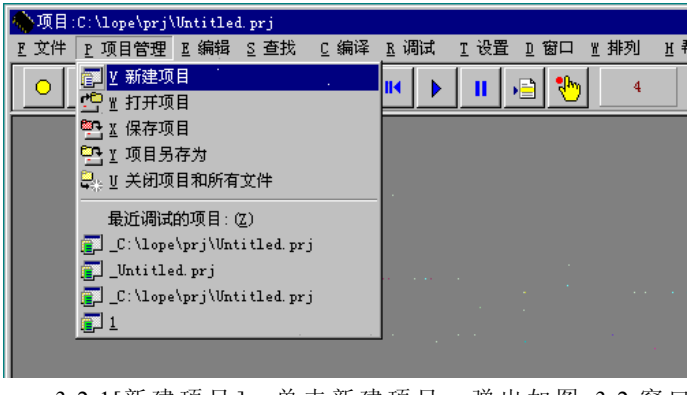


图 3.1

3.2.1[新建项目]：单击新建项目，弹出如图 3.2 窗口。

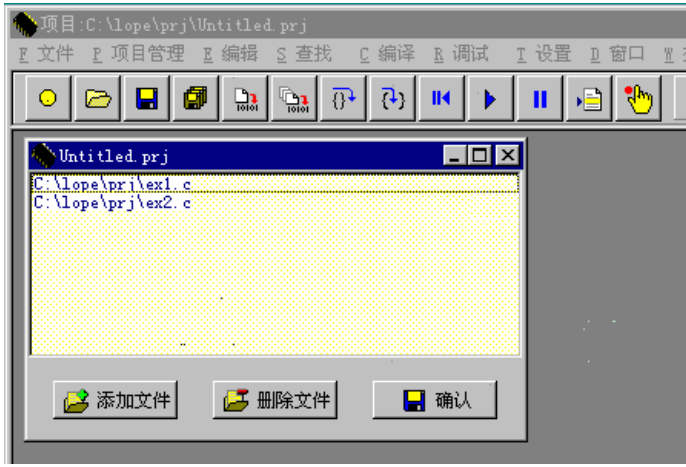


图 3.2

如在该项目中添加文件，单击“添加文件”栏，弹出如图 3.3 对话框所示。

图 3.3

将所需文件打开，该文件名出现在图 3.2 虚线框内，单击“确认”，该文件包含在新项目内。可供选择的文件如下：

1．*.c，c 语言源文件。

2．*.asm 汇编源文件。
3．*.obj 文件。
4．*.lib 文件。

以上列出的文件不必全部列入，例如，选择了 sub1.c,就不必再选 sub1.obj,另外，kc51 提供的标准库不要列入（*.lib）。如果用到自己制作的库，才需要将库文件列入。

如需删除新项目内的文件，点击“删除文件”，再点击图 3.2 虚框内文件名，单击“确认“，该文件即被删除。

3.2.2[打开项目]：单击“打开项目”项，弹出如图 3.4 所示窗口。

图 3.4

选中所需项目名，单击“打开”即可。

3.2.3[保存项目]：项目经修改编辑后，按此项即可保存在原项目内。也可在如图 3.2 中黄框内点击右键，弹出如图 3.5 所示选中“保存项目”即可。图 3.5 另几项分述如下：

“添加模块文件”：限添加 c 文件，asm 汇编文件，lib 库文件及.obj 目标文件。

“从项目中删除模块文件”：此项可将已选中的添加模块删除。

“编辑选中模块文件“：此项对选中的模块文件进行编辑。图 3.6

“编译连接项目中修改过的模块文件”：模块经修改后进行编译连接，如不能连接，显示错误行。以便修改。

“编译连接项目内的所有模块文件”：如编译连接通过显示“ c51 COMPILATION COMPLETE 0 WARNING(S), 0 ERROR(S),”

不通过则显示出错行。见图 3.7

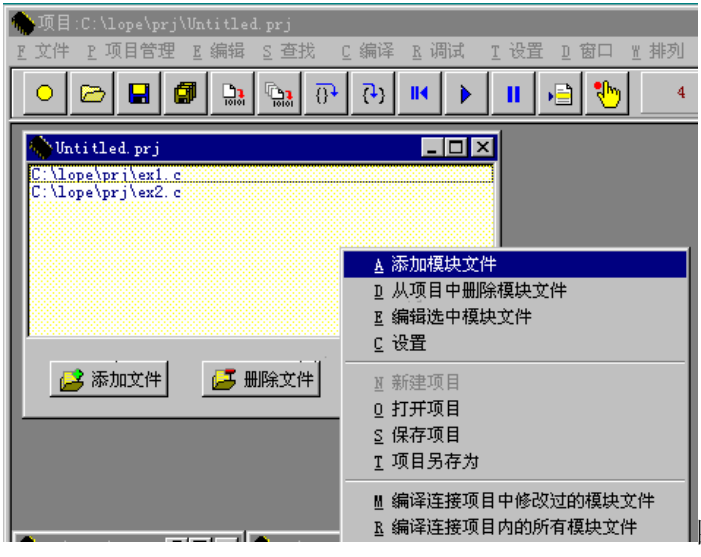
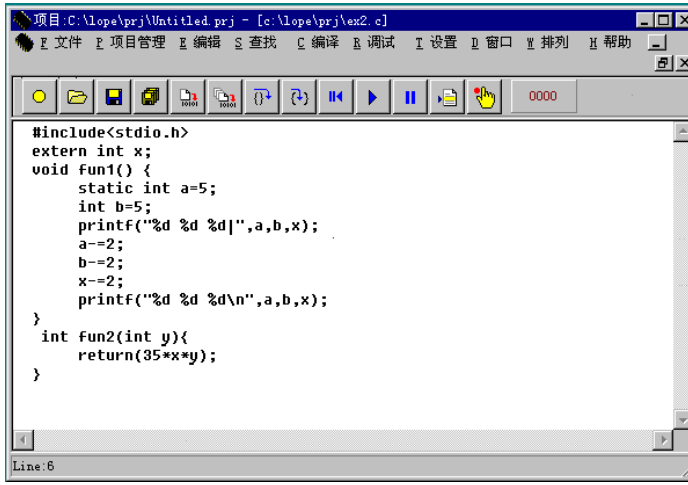
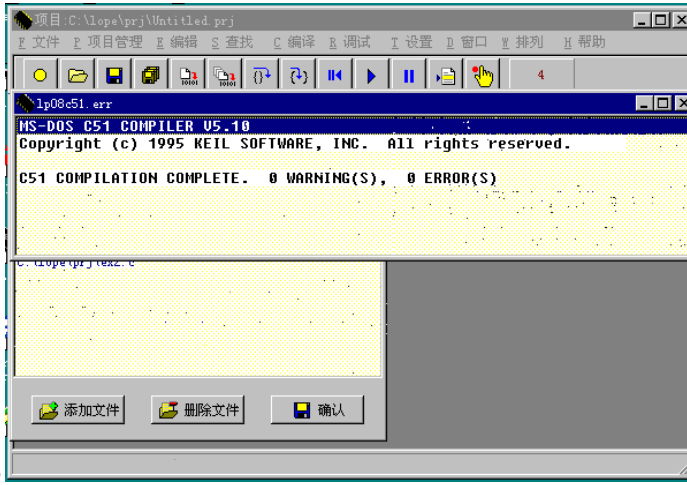


图 3.5

3.2.4[项目另存为]：当需将项目存入另外目录内用此项，也可点击右键选中此项。


项目:C:\lope\prj\Untitled.prj - [c:\lope\prj\ex2.c]
F 文件 P 项目管理 E 编辑 S 查找 C 编译 R 调试 T 设置 D 窗口 W 排列 H 帮助
0000
#include<stdio.h>
extern int x;
void fun1() {
static int a=5;
int b=5;
printf("%d %d %d|",a,b,x);
a-=2;
b-=2;
x-=2;
printf("%d %d %d\n",a,b,x);
}
int fun2(int y){
return(35*x*y);
}
Line:6

图


项目:C:\lope\prj\Untitled.prj
F 文件 P 项目管理 E 编辑 S 查找 C 编译 R 调试 T 设置 D 窗口 W 排列 H 帮助
4
lp08c51.err
MS-DOS C51 COMPILER V5.10
Copyright (c) 1995 KEIL SOFTWARE, INC. All rights reserved.
C51 COMPILATION COMPLETE. 0 WARNING(S), 0 ERROR(S)
添加文件
删除文件
确认

3.6 图

3.7

3.2.5[关闭项目和所有文件]：选中此项关闭项目窗口并退出所有文件。

3.2.6 举例说明：有一个多模块程序，两个文件，名为 ex1.c;ex2.c. 打开[文件]项选择[创建新文件]，输入 ex1.c. 如图 3.8

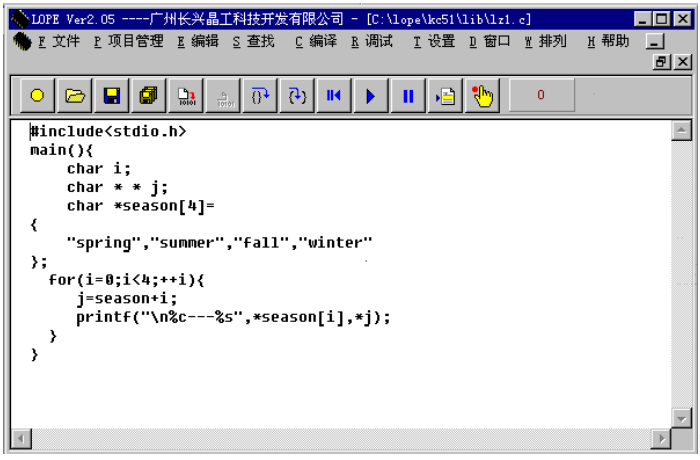


图 3.8

将文件保存，取名 ex1.c。

同样输入 ex2.c 文件。如图 3.9，并保存。

打开[项目管理]项[新建项目]，如图 3.10。将 ex1.c 与 ex2.c 分别添加窗口内并按[确定]。

打开[编译]项[编译连接项目]。自行对 ex1.c,ex2.c 编译连接，成功即可进行单步，运行等仿真调试。并形成项目文件.prj。失败显示出错信息。须进一步修改。

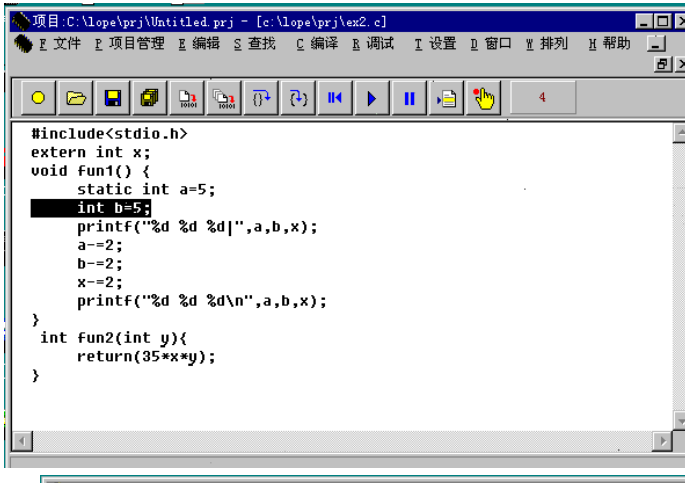

项目:C:\lope\prj\Untitled.prj - [c:\lope\prj\ex2.c]
F 文件 P 项目管理 E 编辑 S 查找 C 编译 R 调试 T 设置 D 窗口 W 排列 H 帮助
#include<stdio.h>
extern int x;
void fun1() {
static int a=5;
int b=5;
printf("%d %d %d|",a,b,x);
a-=2;
b-=2;
x-=2;
printf("%d %d %d\n",a,b,x);
}
int fun2(int y){
return(35*x*y);
}

图 3.9

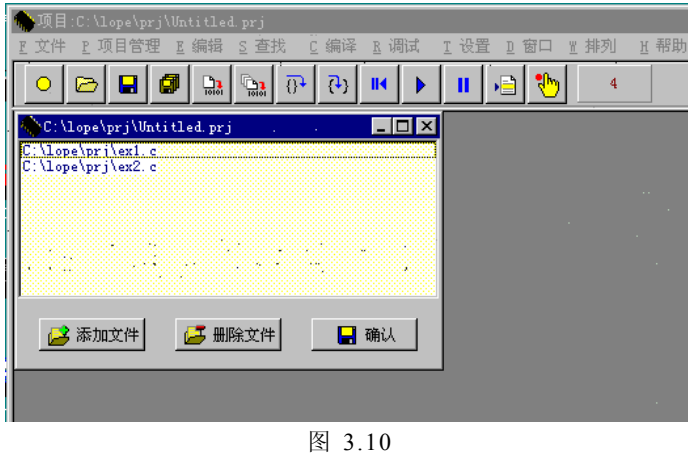

项目:C:\lope\prj\Untitled.prj
F 文件 P 项目管理 E 编辑 S 查找 C 编译 R 调试 T 设置 D 窗口 W 排列 H 帮助
C:\lope\prj\Untitled.prj
C:\lope\prj\ex1.c
C:\lope\prj\ex2.c
添加文件
删除文件
确认

图 3.10

第四章　编辑和编译(汇编)源程序

4.1 文件操作命令

lope可以仿真多种文件格式的程序，所有涉及文件的操作，都在主菜单的“文件”命令中完成。

请注意，为了确保编译正确进行，目录和文件不要使用长文件名。(长文件名是指：文件名超 8 个字符，扩展名超 3 个字符)

4.1.1 [创建新文件]:

第一次编写一个新的程序，须用此命令建立一个新文件。命 令如图 4.1

图 4.1

命令执行后，在主窗口右边弹出如下文件窗口。图 4.2

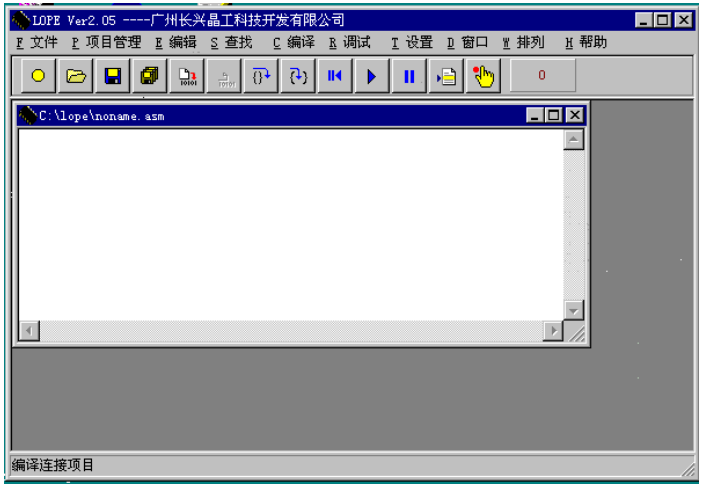


图 4.2

接着用户可在窗口中编写源程序。

4.1.1 修改已有的文件

用此命令修改原来已经存在的程序，命令如下：

4.1.2 择[打开文件]，弹出窗口如图 4.3

用户可进入适当的目录装载需要修改或仿真的程序文件。

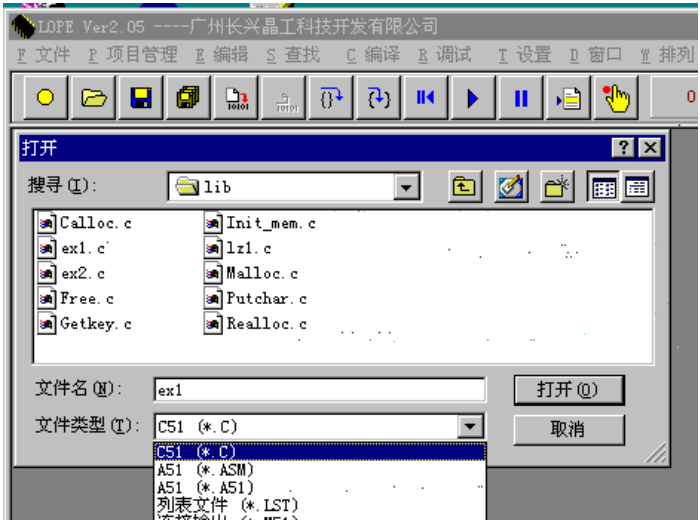


图 4.3

注意，选择文件类型，如 C 文件点击 C51（*.C）汇编选 A51(*ASM),然后点击文件名将文件打开。

为了确保编译正确进行，目录和文件不要使用长文件名。（长 文件名是指：文件名超 8 个字符，扩展超 3 个字符）

。

4.1.3［保存源文件］:

此命令用于保存当前正在编辑的源文件 ，如果该文件从未存盘过，将打开“另存为”对话框，让用户指定文件名和位置。此命令只保存文本文件，不保存其他仿真信息。

请注意，当执行编译或汇编命令时，程序会先保存文件，然后再编译或汇编，因此，如果是经常使用编译或汇编命令，则无需执行保存命令。

命令如图 4.4

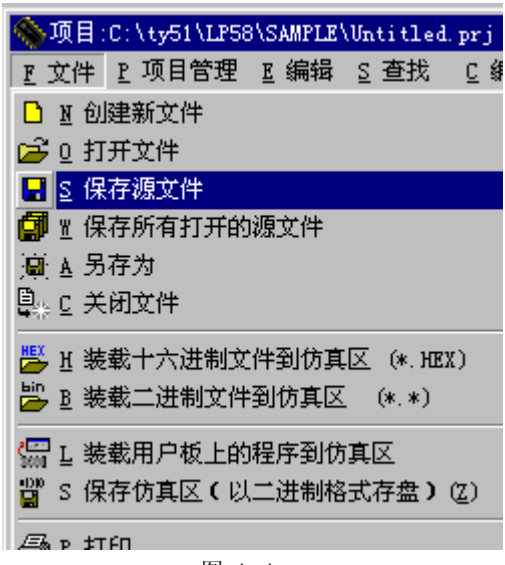


图 4.4

4.1.4 [另存为]:

此命令用于把当前编辑的文件更名保存或换位置保存。命令如图 4.5

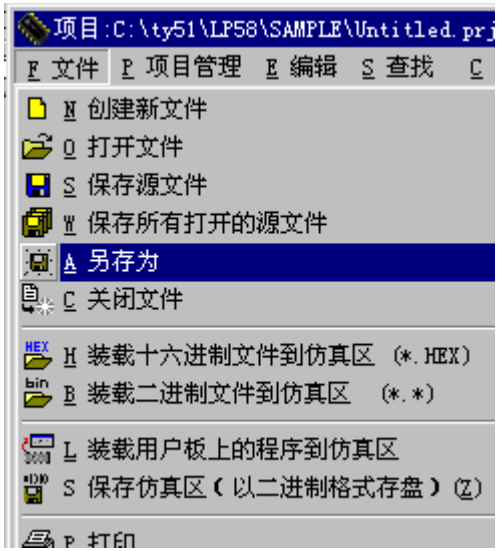


图 4.5

4.1.5[装载十六进制文件到仿真区](*.HEX):

Intel 十六进制文件是最通用的文件格式，各种汇编器和 C 语言编译器都可产生这种格式的文件，LOPE 系统允许用户单独装载 HEX 文件仿真。但是由于 HEX 文件不包含源文件和符号信息，只是以反汇编的形式仿真调试。HEX 文件装载后，并不是以文本格式显示在源文件窗口，而是被翻译成二进制格式，装载到仿真区，显示在反汇编窗口。

菜单命令如图 4.6

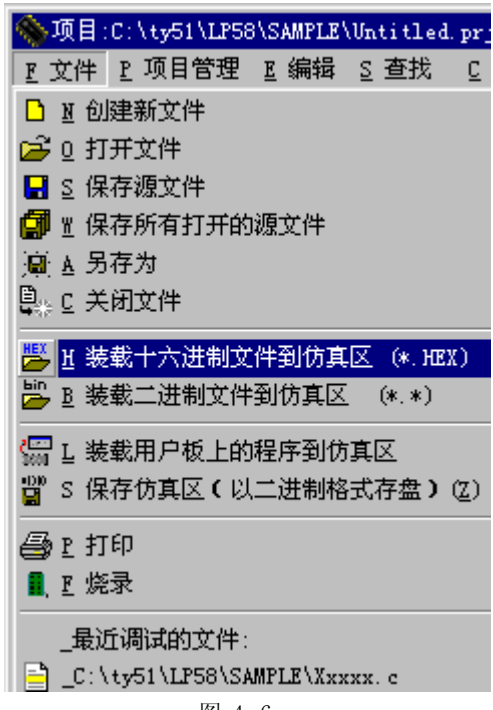


图 4.6

命令执行后的反汇编窗口如图 4.7

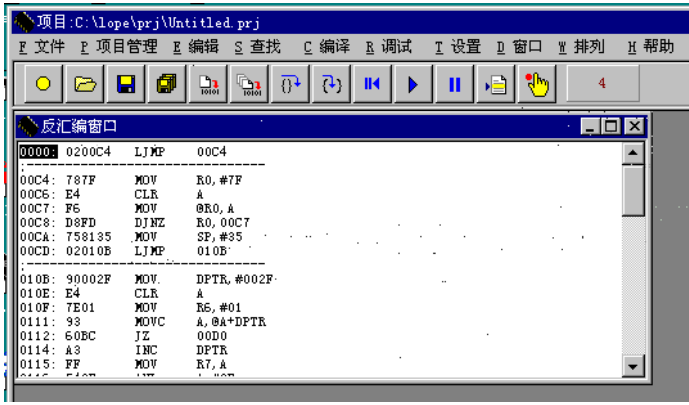


图 4.7

4.1.6［装载二进制文件到仿真区］:

二进制文件的仿真形式与十六进制文件相仿，只能以反汇编形式仿真调试，不同的是，不需要翻译，可直接装载到仿真区。

命令如图 4.8

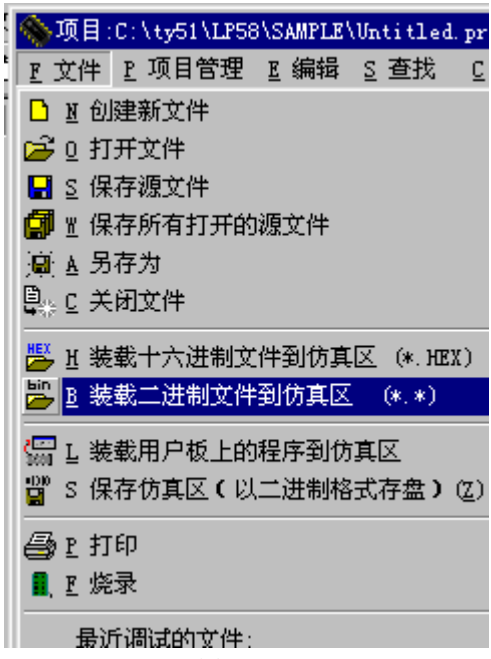


图 4.8

二进制文件不包含地址信息，数据是顺序存放的，文件被选定后，会弹出一个对话框，要求用户输入起始地址：

4.1.7［装载用户板上的程序到仿真区］：

有时候，用户可能需要调试用户板上EPROM中的程序，例如，EPROM是焊接死的，无法拆卸，这时可以先将EPROM的内容装载到系统仿真区，然后以反汇编形式运行和调试，LOPE有专门的菜单命令完成这个任务。装载时需要用户确定 EPROM的起始地址和终止地址。下面上图是菜单命令，下图是输入对话框，预设为8K字节，用户需以十六进制地址输入。图 4.10

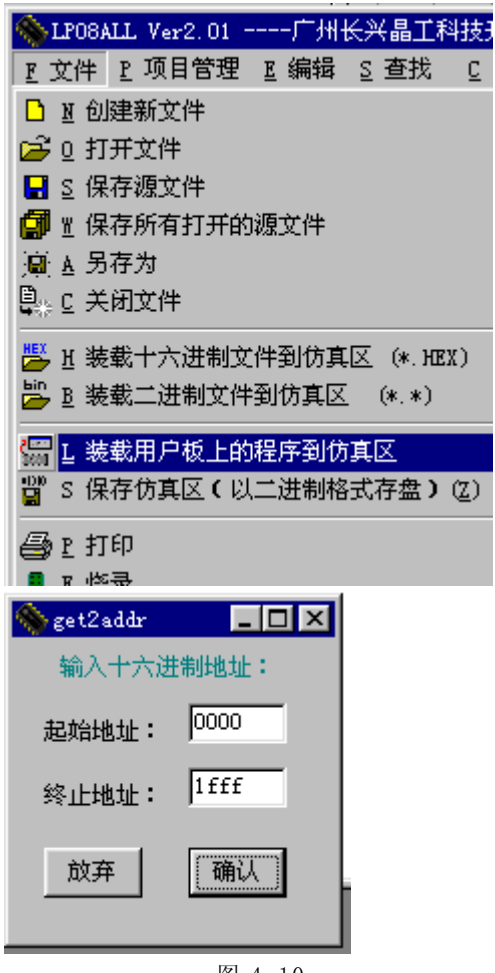

LP08ALL Ver2.01 ----广州长兴晶工科技
F 文件
P 项目管理
E 编辑
S 查找
C
N 创建新文件
O 打开文件
S 保存源文件
W 保存所有打开的源文件
A 另存为
C 关闭文件
H 装载十六进制文件到仿真区 (*.HEX)
B 装载二进制文件到仿真区 (*.*)
L 装载用户板上的程序到仿真区
S 保存仿真区（以二进制格式存盘）(Z)
P 打印
get2addr
输入十六进制地址：
起始地址：
0000
终止地址：
1fff
放弃
确认

图 4.10

4.1.8 [保存仿真区 (以二进制格式存盘)]:

如果用户以反汇编形式调试程序而且修改了程序，退出之前可以二进制格式存盘。建议用户不要在机器码级别上修改程序，而应在源文件中修改程序。

保存仿真区的操作与“装载用户板上的程序到仿真区”的操作相仿，下面图 4.11 是菜单命令，图 4.12 是输入对话框：

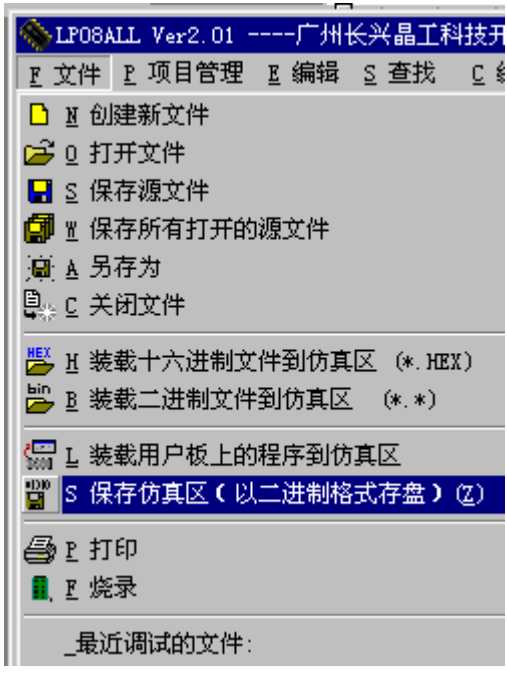


图 4.11

图 4.12

输入地址按确定后，弹出文件对话框，用户可以按 Windows的标准操作存盘。图 4.13

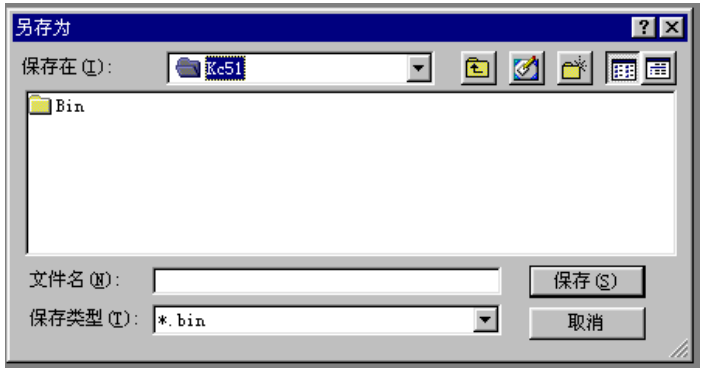


图 4.13

[打印]：此项可将窗口内容打印出来。

[烧录]：移动光标到此项，显示top烧录器型号，选中型号，即可将编译连接好的程序烧录到存储器内。

[最近调试的文件]：每此调试的文件，保存最后 4 次文件路经

于此窗内，方便下次调试。

4.2 编辑和查找的操作命令

一． 编辑：

4.2.1 用鼠标选择文本:打开文件后便可对程序编辑。

按住鼠标左键不要放开,水平或上下移动鼠标选择文本,松开鼠标左键后，文本既被选中的文本以反白显示。下图是一段选中的文本：图 4.14。请注意，有时运行了其它程序后，由于干扰，不能反白文本，须退出其它程序。

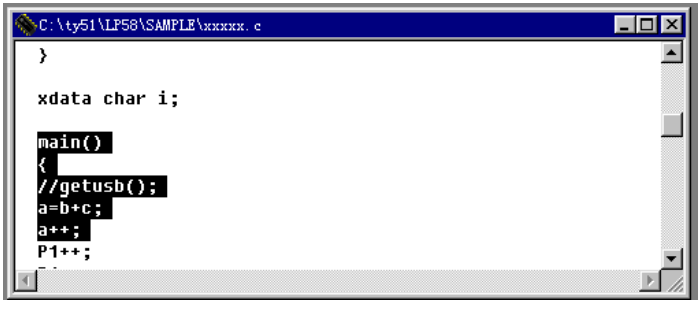


图 4.14

4.2.2 用键盘选择文本

按住 Shift键不动，按四个方向键选择文本，松开鼠标左键后，文本即被选中。选中的文本以反白显示。结果与鼠标选择相同。

4.2.3 [剪切]：

把代码编辑器中选择的文本剪切到剪贴板中。

4.2.4 [拷贝]：

把代码编辑器中选择的文本复制到剪贴板中。

4.2.5 [粘贴]：

把剪贴板中的内容复制到代码编辑器。除非又向剪贴板中剪切或复制了新的内容，剪贴板中的内容不变。也就是说，可以重复粘贴任意次。

4.2.6 删除

按 Delete 键，被选择的文本即被删除掉。如果没有选择的文本，则删除光标右边的一个字符。

二．查找：

4.2.7［查找］:

此命令用来在代码编辑器中查找指定的文本字符串，可在 “编辑窗口”中查找;“文件“中查找和是否替换。区分大小写。命令执行后，弹出标准查找对话框如图 4.15

图 4.15

在“查找目标”框内键入要查找的文本，也可以从剪贴板中 粘贴。

按“查找下一个”开始查找。可继续按 “查找下一个”再次查找。

4.2.8[在文件中查找]：先应打开项目文件并确认，再查找。结果如图 4.16.双击某行可查到对应的反白文本行。

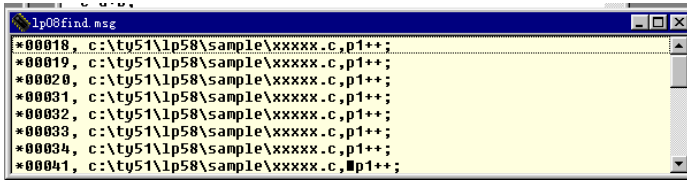


图 4.16

4.2.9［替换］:

此命令进行文本字符的替换操作。命令执行后，弹出替换对话框如下：图 4.17

图 4.17

在“查找”目标框内键入要查找的文本，也可以从剪贴板中粘 贴。

在“换为”目标框内键入要替换的字符

再按“替换一个”执行替换。

注意，再次替换下一个也必须再按 “替换一个”，执行替换。若替换窗内所有同名文本，执行 “全部替换”。

4.2.10[查找下一个]：按此键在编辑框内反白显示下一个要查找的字符。

4.3 编译（汇编）与连接

在编辑窗口编写的源程序文件必须经过编译（汇编），连接后才能在仿真器上运行或调试，LOPE将所有过程集成在一个窗口完成。LOPE采用 keil C51编译 c 源文件。源程序经编译后，如果有错误，弹出错误窗口，显示所有的错误。

一． 编译和连接的设置

该软件有一个标准的设置，适应大多数用户，无须设置。如确需设置，可按“设置“栏进行设置。图 4.18

4．18

4.3.1[设置 A51汇编]：设置 A51汇编有“列表文件“，“目标文件”两项。需要进行“单步”“跟踪”“断点”等仿真操作，按“标准设置“。”缺省设置“运行速度快，但不能调试。图 4.19

图 4.19

4.3.2[设置 C51 编译]：如果是 C 语言，则执行“C51 编译”栏，弹出对话框如图 4.20 所示。设置操作与前同，也有标准设置与缺省设置。可根据自己需要自行设置。

图 4.20

4．3．3[设置 L51 连接]：经过编译（汇编）后，keil c51 提供 bl51.exe 文件将编译（汇编）形成的 obj 文件连接，连接设置如图 4.21。

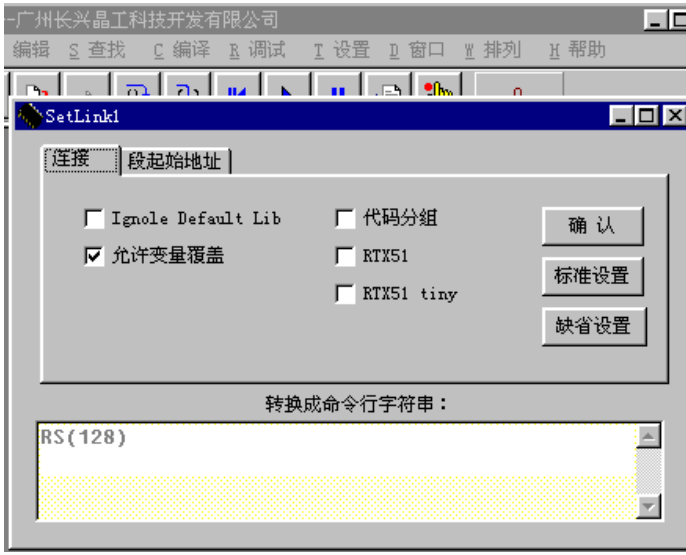


图 4.21

该设置也分“标准设置”与“缺省设置”，作用同“a51”设置。

4.3.4[设置 cpu 时针]：如图 4.22。可根据需要设置。

图 4.22

4.3.5[设置仿真代码长度]：如图 4.23,根据单片机和仿真器 等级自行设置。

图 4.23

4.3.6[设置 P87LP769]:该设置只是针对 P87LP769 系列设置，具体要求请查看该芯片手册。图 2.24

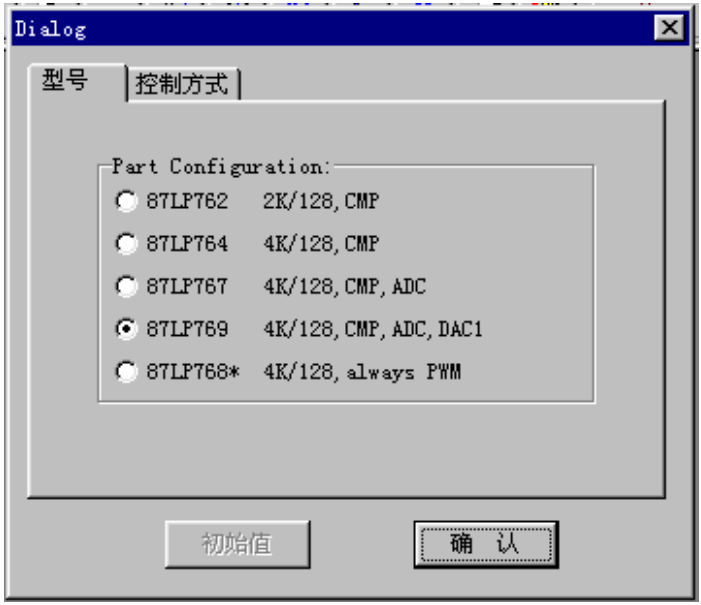


图 4.24

二． 编译 （C 语言）

编译命令用来对用户编辑输入的源程序进行语法检查 ，将 C 语言程序翻译成没有定位的汇编语言。

通常一个程序要通过反复编辑和编译才能最后通过，即使是经验丰富的专业程序员，也不可能全不出错误。源程序经编译或连接后，如果有错误，会自动将含有错误的语句显示，

经过编译通过后的程序，还不能运行调试，还需要连接才能最后形成可运行的文件。

C 语言允许连接多个目标文件 (.OBJ)，这些文件可以是分开的 C 语言通过编译后产生的 OBJ文件，也可以是汇编语言产生的 OBJ文件。当然，这些文件必须遵守连接格式的要求，否则会产生连接错误。

C 语言允许多个源文件分开编译，最后用连接命令连接为一个可运行文件。不过系统的编辑窗口允许超大文本编辑，用户可以用一个文件来编写一个大的程序。

编译菜单命令如图 2.25:

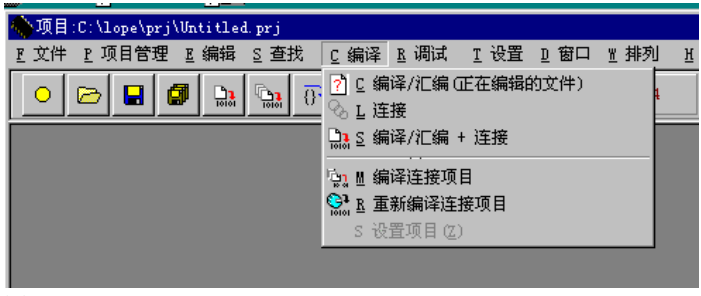


图 2.25

4．3．7[编译或汇编（正在编辑的文件）]：对正在编辑的文件进行编译或已编好的 c51,A51文件必须先选中一个文件入编辑 窗口，然后进行编译。如有错误，显示错误行，便于修改。双击错 误行，会在编辑框反白显示错误。图 2.26

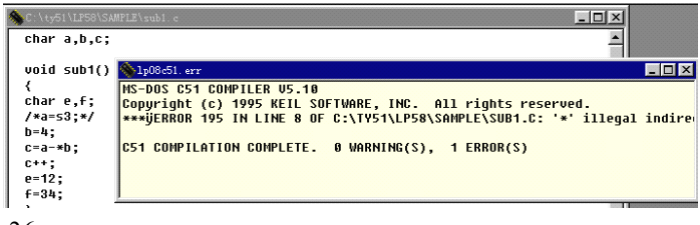


图 2.26

4.3.8[连接]：对编辑窗口的文件经多次修改无误，进行连接 。

将目标文件(.OBJ)连接成可执行的文件(.HEX)，同时产生符号调试必需的各种控制文件。“HEX”可以用烧录器烧录程序，大部分烧录器都支持这种格式。

C 语言允许连接多个目标文件（.OBJ),这些文件可以是分开的 C

语言通过编译后产生的 OBJ文件，也可以是汇编语言产生的 OBJ文件，多个目标文件的连接请参考“编译和连接的设置项目”中的“其他连接文件选择”。

汇编语言不需要连接。汇编完成后即可进行仿真调试。

4.3.9[编译（汇编）+连接]： 一次完成（C 语言）

编译和连接命令合为一个命令来执行，简化了操作。但如果用户不想马上运行，还是单独编译命令为好，这样速度快,且便于查错。

4.3.10[编译连接项目]：

由于在项目工程内有多个文件组合而成,故先将相关文件添加 在框内，确认后编译连接。如图 2.27.

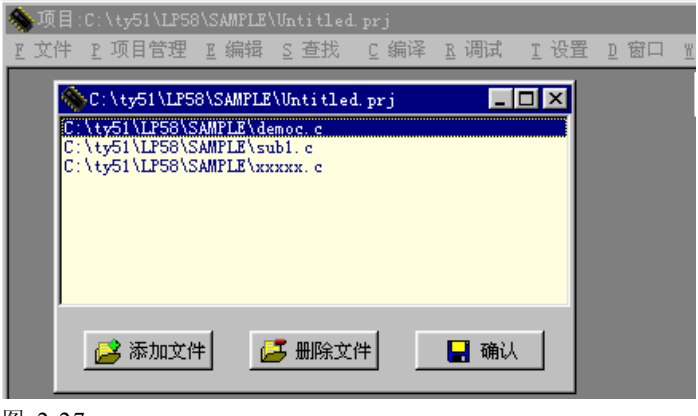


图 2.27

如果编译连接失败会显示错误行。但不能双击错误行指定错误点。可按提示逐行修改。直到满意为止。图 .2.8

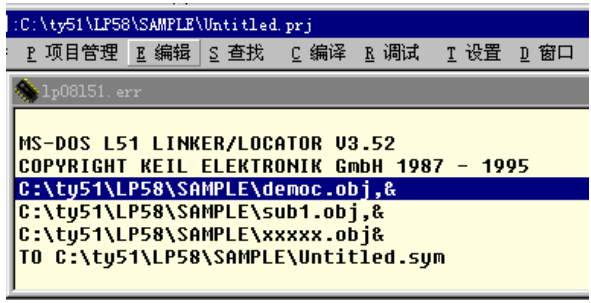


图 2.28

4.3.11[重新编译连接项目]：须说明的是，项目经编译连接后出错，修改原文件，就出现再编译时已编译过的 obj 文件不再编译，只编译修改过的文件，因此，速度较快。而重新编译连接项目即包括已编译过的文件，速度稍慢，用户自行选择。

第五章　调试运行程序

5.1 运行和调试程序

调试命令非常丰富，大部分命令集中在“调试”菜单项目中。有一部分最常用的命令也可以用 “弹出菜单”执行(在编辑窗口中，按鼠标右键可随时弹出)。下面用户可根据自己习惯选择一种使用 。

调试前请设置好仿真频率。执行 [设置/设置 cpu时针]项。

CPU振荡频率可以让用户选择，而且非常灵活，可以直接使用用户板上的晶体振荡器，又可以由仿真系统提供振荡频率。系统 提供共 14 种频率，最高 64mc，最低 362kc。所有这些频率的选择都由软件控制，不需要硬件跳线。 (见第四章图 4.2.2)

如果单选按钮选择了“使用用户时钟”,则列表框中的频率无 效，实际频率=用户板晶振。

如果单选按钮选择了 “使用仿真器时钟”，则实际频率=列框中选中的频率。

调试窗口如图 5.1。

图 5.1

5.1.1 单步运行程序

菜单命令:“调试/单步”，也可以按 F8。工具栏 

C 语言:单步运行一行程序，如果从一行包含有函数调用的语句开始，而且函数中也包括程序，则该函数被看作一行语句执行，在函数中不会停留。

汇编语言:单步运行一条指令，如该指令是一条 CALL 指令，则会运行多条指令，要运行完被调用的子程序的所有指令，直到执行 RET 指令返回，程序停在 CALL 指令+3 的地址。如果子程序陷入死循环或者不返回，则要用“暂停”命令停止。

单步命令执行完后，所有的窗口会更新显示。可以观察到运行

后的数据变化。

5.1.2 跟踪运行程序

菜单命令:“调试 / 跟踪 ”，也可以按 F7。工具栏 

C 语言：跟踪运行一行程序，与 “单步 ”不同的是，如果从一行包含有函数调用的语句开始，则程序转移到函数体中，执行函 数中的一行程序。

汇编语言：跟踪运行一条指令，如该指令是一条 CALL 指令，则跟踪执行子程序中的一条指令。跟踪命令永远不会陷入死循环， 不需要用 “暂停 ”命令停止。

跟踪命令执行完后，所有的窗口会更新显示。可以观察到运行后的数据变化。

5.1.3 全速运行程序

菜单命令:“调试 / 全速运行 ”，也可以按 F5。工具栏 

不管是 C 语言还是汇编语言，全速运行命令以实际的 CPU 时针频率，从当前的 PC 地址开始运行程序。程序运行后，主窗口的右上角的 “运行指示器 ”不断变化，表示程序正在运行。有三个命 令可以使程序停止，效果各不相同，用户可根据需要选择：

a 暂停命令：停在正在运行的 PC 处，光标移到 PC 的源程序行，适合于观察现场，例如查看陷入死循环的点。

B 复位命令：复位 CPU，PC=0000，光标移到文件首行，适合于重新调试程序。

C 放弃命令：复位 CPU，光标不变，源文件窗口不变，适合于 接着修改源文件。

5.1.4 运行到光标所在的行

菜单命令：“调试/运行到光标所在的行 ”，也可以按 F4。

此命令与设置断点然后运行至断点的功能相仿。不过 ，不会在程序中设置任何点。

命令执行后，程序运行到光标所在的行停下。

如果程序无法运行到光标行，运行指示器会不断变化，这时可 用“暂停”，“复位”或“放弃”命令停止运行。

5.1.5 PC 移到光标所在的行

菜单命令：“调试/PC=光标所在的”，也可以按 Ctrl+F4。

有时用户可能只需要调试某一个子程序，这时可以把 PC 指针直接移到子程序的首行，再进行调试，而不必从头开始。

5.1.6 设置断点或取消断点

菜单命令：“调试/设置（取消）断点”，工具栏 

可在源程序中设置断点，程序全速运行时，遇到断点行的地址 时，会立即停止运行，各窗口显示出当时的状态。仿真器设计了 “全空间无条件断点”，可在任意地址，任意行设置断点，断点的数目无限制。

5.1.7 终止运行

当运行指示器在不断变化时，表示程序在全速运行，要终止运行有三个命令可以使用，效果各不相同，用户可根据需要选择其一 。终止运行的三个命令如下：

a 暂停命令：

菜单命令：“调试/暂停运行”，工具栏 

停在正在运行的 PC 处，光标移到 PC 的源程序行，适合于观察现场，例如查看陷入死循环的点。

B 复位命令：

菜单命令：“调试/cpu 复位”，工具栏 

复位 CPU，PC=0000，光标移到文件首行，适合于重新调试程序 。

C 放弃命令：复位 CPU，光标不变，源文件窗口不变，适合于接着修改源文件。

5.1.8 连续单步或跟踪

菜单命令：“调试/连续单步运行”和“调试/连续跟踪运行”。

适合观察程序连续运行状况。

5.1.9 清除运行指示器

主菜单“调试”/“清除运行指示器”命令可将其清除为零。

鼠标箭头对准运行指示器，点击左键也可将其清除为零。

例如运行时的显示：54，清除后显示为：0000。

5.1.10 跟踪显示符号变量的数值

程序的调试阶段,经常要查看符号变量的数值，高级语言更加是如此。传统的方法是让用户在键盘上键入变量名，然后将变量名 加入到变量显示框，既烦琐又容易出错。 LOPE 系统增加了一种高级的方法，用户只要用鼠标左键点一下变量，在鼠标的附近会显示 一个黄色的小框，该变量的数值会立即显示出来，不需要转移视线 既可看清楚变量。既方便又快捷。下面显示的是变量 I 的十进制数值，右边显示的是 ASC 码。

鼠标箭头移开变量以后，黄色的小框会自动消失。

```
uchar i;
d<<=(16-n);
for(i=0;i<n;i++)
      {if(d&0x
      else SD24=0;
```

i = 2...' '

查看变量属性，将鼠标对准变量点击右键，弹出对话窗口如 图 5 .2。选择“查看变量类型”项，可看到变量的名称，类型，长 度，存储内外存储器，地址等属性。

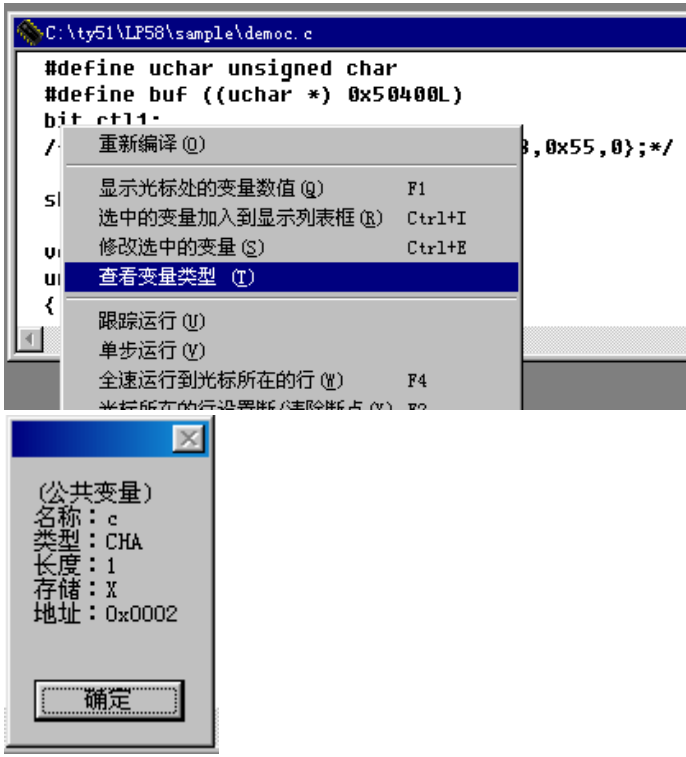


图 5.2

如果想进一步查看这个变量的二进制，十六进制等格式的显示，或者想修改这个变量，可双击鼠标弹出对话框。

5.1.11将变量加入到“变量列表框”

虽然轻点鼠标查看变量是最方便的操作，但是如果想长时间观察变量则不如把变量名加入到专门显示变量的“变量显示列表框”，这样，程序每运行一步，变量随即更新。操作如下：

(1) 光标指到变量，点击鼠标左键，观察变量；
(2) 按 Ctrl+I 加入该变量。
也可用弹出菜单命令加入操作如下：
(1) 光标指到变量，点击鼠标左键，观察变量 ；
(2) 点击鼠标右键，弹出菜单如图 5.3：

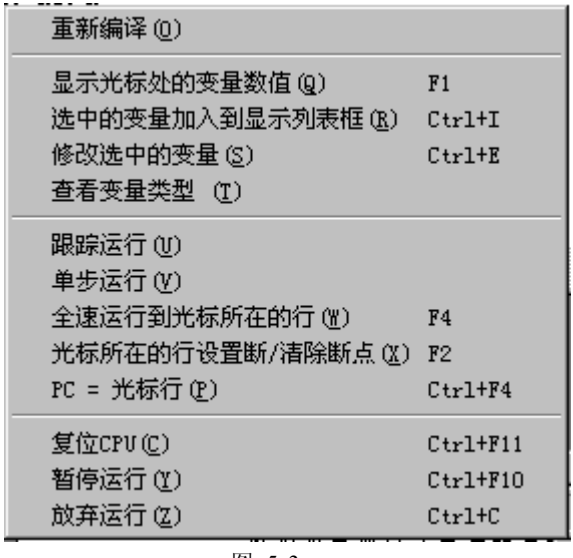


图 5.3

(3) 选择“选中的变量加入到变量列表框 ”命令加入。
变量加入后，显示如图 5.4：

```
}
SCON=0xc0;
```

图 5.4

5.2 修改符号变量或寄存器

修改符号变量或寄存器有两种操作方法，使用鼠标结合键盘操作。通常以使用鼠标最为方便，如果连续修改大量数据，则以键 盘操作比较快。

5.2.1 用鼠标进入“修改对话框”

移动鼠标，箭头对准变量或寄存器，双击鼠标左键，弹出修改对话框。

5.2.2 用键盘进入“修改对话框”

上下移动光标，对准变量或寄存器，按回车弹出修改对话框。

5.2.3 “修改对话框”的操作

“修改对话框”如图 5.5：

图 5.5

“修改对话框”将同一个数值用四种格式显示，分别为十进制 ，

十六进制，二进制和浮点数。可以任选其中之一，在相应的编辑 框中，键入修改的数字，按回车或 “确认”结束。

要注意的是，如果输入的数值与原来的相等，即使 “确认”，也不会引起实际的操作（CPU 不会对寄存器或 RAM“读/写”）。

5.3 其他操作

5.3.1 以反汇编形式调试程序

不管是否装入了仿真文件，都可以随时打开反汇编窗口。反汇编窗口将程序仿真区的二进制代码翻译成汇编指令显示。

反汇编窗口打开后，可以执行 “单步”，“跟踪”，“全速”等操作。这些命令与源文件窗口的命令相同，究竟在哪一个窗口执行，取决于当前激活的是哪一个窗口，用鼠标点击可以转换窗口。

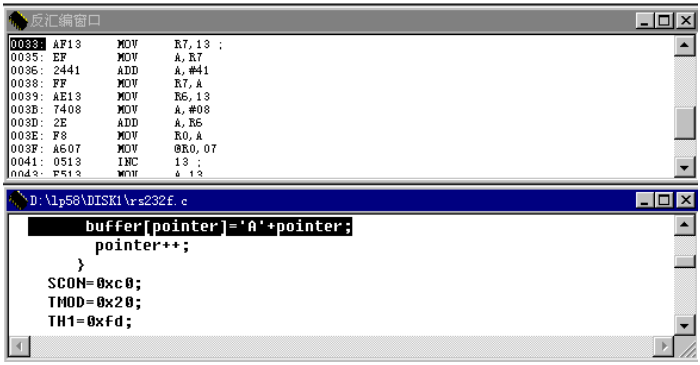


图 5.6

当程序运行异常时，反汇编窗口会自动打开，显示当时的现场情况，让程序员分析异常的原因。

反汇编窗口通常是自动跟踪 PC，一次反汇编一段，但不会清 除前面已经显示的行。在源文件窗口单步，跟踪，断点等调试时， 反汇编窗口也同步地跟踪显示对应的反汇编指令，除非关闭反汇编 窗口。

即使没有源文件，反汇编窗口也照样可以单独调试运行。

如果是 C 语言源程序可能会对应好多行汇编指令，这时对应的是第一条反汇编指令。

如果是汇编语言源程序，则必定是一一对应。

5.4 系统电源

系统总是使用自带的电源，不能由用户板提供电源，这样可以避免因用户板电源接反而造成系统板损坏。系统自身消耗约 200mA 电流，另外，系统可以向用户提供 5V／200mA 稳压电源，系统电源经二极管隔离后，通过仿真头第 40 脚向用户提供 5V／200mA 稳压电源。

5.5 烧录（固化）

程序经过调试后，最后将十六进制文件 (*.HEX)烧录到单片机或者 EPROM 中运行。

附录一 A51 汇编语言参考

5.1 宏汇编器 A51

5.1.1 符号与表达式

8051 单片机汇编语言程序由若干条 8051 指令行组成，8051 指令行的一般形式为：

[标号:] 8051 助记符[操作符 1][,操作符 2][,操作符 3][;注解]

其中,“标号”是可选项，它可用来表示程序的转移地址，同时可方便程序的调试。

“8051 助记符”是 8051 单片机的指令助记符。

“操作符 1~3”是可选项,它的使用依赖于不同的 8051 指令助记符。有些指令不需要操作符，有些指令则需要 1~3 个操作符。操作符可以是数字、符号或地址。

数字可以使用 10 进制、16 进制、8 进制或 2 进制数。10 进制数以字符“D”为后缀，16 进制数以字符“H”为后缀，8 进制数以字符“O”为后缀，2 进制数以字符“B”为后缀，省略后缀时则默认为 10 进制数。立即数的前面须冠以符号“#”。

A51 宏汇编器允许使用符号来表示数值、地址和寄存器名等，以增加程序的可读性。符号名最长为31 个字符，第一个字符为英文字母“A”～“Z”或“a”～“z”“、”“_” 或“?”，后续字符可为上述字符或数字“0”～“9”。标号也是一种符号。一些符号已 经预定义为 A 51 的保留字，用户不能对它们重新定义。这些符号及 其意义如下所示。

A,R0-R7,DPTR,PC,C,AB,AR0-AR7

符号″ $″ 是一个特殊的汇编符号，它表示当前段的当前地址计数器。CODE、DATAI-DATA、BIT 和 XDATA 这 5 个段都有不同的地址计数器。每执行一条指令，地址计数器的值也随之增加。如果当前段发生变化，地址计数器也将自动变到新段。例如下面的一条指令：

HALT：SJMP $

表示跳转到标号 HALT 处。

A51 中有三类运算符：算术运算符、逻辑运算符和关系运算符，如下所示。

算术运算符：　＋　－　*

逻辑运算符：　MOD,( ),NOT,HIGH,LOW,SHL,AND,OR,XOR

关系运算符：　>=,<=,<>,=,<,>

A51 宏汇编器的运算符具有如表 5.10 所示的优先级，一个表达式中存在多个不同优先级的运算符时将按它们的优先级顺序进行运算，如果一个表达中各个运算符都具有相同的优先级，则按从左到右的顺序进行运算。

宏汇编器中运算符的优先级：

一级优先　()

二级优先　NOT,HIGH,LOW

三级优先　+,-

四级优先　*,/,MOD

五级优先　+,-

六级优先　SHL,SHR

七级优先　AND,OR,COR

八级优先　>=,<=,=,<,>

5.1.2 汇编伪指令

1.SEGMENT 指令

SEGMENT 指令用来声明一个再定位段和一个可选的再定位类型,指令格式如下:

再定位段名 SEGMENT 段类型[再定位类型]

其中,“再定位段名”用于指明所声明的段。

“段类型”用于指定所声明的段将处的存储器地址空间。可用的类段型有：CODE、XDATA、DATA、IDATA 和 BIT。

“再定位类型”是可选项，用于定义将由 L51 决定的段位置。

2. EQU 指令

EQU 指令用于将一个数值或寄存器名赋给一个指定的符号

名，指令格式如下：

符号名 EQU 表达式

或　符号名 EQU 寄存器名

3.SET 指令

SET 指令的功能 EQU 指令类似，不同的是用 SET 指令定义过的符号可被重新定义指令格式如下：

符号名 SET 表达式 或　符号名 SET 寄存器名

4. BIT 指令

BIT 指令用于将一个位地址赋给指定的符号名。指令格式如下：

符号名　BIT　位地址

经 BIT 指令定义过的位符号名不能被改变。

5.DATA 指令

DATA 指令用于将一个内部RAM 的地址赋给指定的符号名。指令格式如下：

符号名 DATA 表达式

数值表达式的值应在 0~255 之间,表达式必须是一个简单再定位表达式。经 DATA 指令定义过的符号在程序中不能被重新定义。

6.XDATA 指令

XDATA 指令用于将一个外部 RAM 的地址赋给指定的符号名 。指令格式如下：

符号名 XDATA 表达式

表达式必须是绝对或简单再定位表达式。经 XDATA 指令定义过的符号在程序中不能被重新定义。

7.IDATA 指令

IDATA 指令用于将一个间接寻址的内部 RAM 地址赋给指定的符号名。指令格式如下：

符号名 IDATA　表达式

表达式必须是绝对或简单再定位表达式。经 IDATA 指令定义过的符号在程序中不　能被重新定义。

8.CODE 指令

CODE 指令用于将程序存储器ROM 地址赋给指定的符号名。

指令格式如下：

符号名 CODE 表达式

表达式必须是绝对或简单再定位表达式。经CODE 指令定义过的符号在程序中不能被重新定义。

A51 中的保留和初始化存储器空间汇编伪指令可用来在存储器空间内保留和初始化字、字节或位单元，保留空间始于当前地址的绝对段和当前偏移的再定位段。保留和初始化存储器空间指令共有 4 个，分述如下。

1. DS 指令

DS 指令以字节为单位在内部或外部数据区保留存储器空间。

指令格式如下：

[标号:] DS 数值表达式

数值表达式中不能包含前向参考量。 DS 指令使当前数据段的地址计数器增加表达

式结果之值，地址计数器与表达式结果之和不能超过当前地址空间的限制。括号内的标号是可选项，使用了标号时，标号值将是保留区的第一个字节地址。

2. DBIT 指令

DBIT 指令在内部数据区的 BIT 段内以位为单位保留存储器 空间。指令格式如下：

[标号:] DBIT 数值表达式

数值表达式中不能包含前向参考量。可再定位或外部符号。 DBIT 指令使 BIT 段的地址计数器增加表达式结果之值，地址增加以位 为单位。括号内的标号是可选项，使用了标号时，标号值将是保留区的第一个位地址。

例子：

```
RSEG    BIT_SEG1        ;选择 BIT 段
PRG     22H             ;定义起始地址
FLAG_ARE: DBIT 8        ;在 BIT 段保留 8 个位,FLAG_ARE 为位地址〞22H〞
FLAG1 :   DBIT 2        ;在 BIT 段保留 2 个位, FLAG1 为位地址〞22H〞
```

3. DB 指令

DB 指令以给定表达式的值的字节形式初始化代码空间。指令格式如下。

[标号:]DB 表达式表

表达式表中可包含符号、字符串、或表达式等项，各个项之 间用逗号隔开，字符串

应用引号括起来。括号内的标号是可选项，如果使用了标号，则标号的值将上表达式表中第一个字节的地址。DB 指令必须位于 CODE 段之内，否则将会发生错误。

4. DW 指令

DW 指令以给定表达式的值的双字节形式 (16位值)初始化代码空间。指令格式如下 : [标号:]DW 表达式表

表达式表中可包含符号、字符串或表达式等项，各个项之间用逗号隔开。字符串应用引号括起来，并且一个字符串最多只能包含两个字符。括号内的标号是可选项，如果使用了标号，则标号的值将是表达式表中第一个字节的地址。如果 DW 指令不在 CODE 段内将会发生错误。

A51 中控制程序连接的汇编伪指令可用来标明当前程序模块、当前模块中需要使用的外部符号名以及可被其它模块使用的公共符号名，从而允许连接定位器 L51 将几个不同的程序模块连接成一 个绝对模块。控制程序连接的汇编伪指令共有 3 个，分述如下：

1. PUBLIC 指令

PUBLIC 指令用于声明可被其它模块使用的公共符号名。指 令格式如下：

PUBLIC 符号 [,符号 [,...]]

2. EXTRN 指令

EXTRN 指令是与 PUBLIC 指令相配套的,如果要使用其它模 块中用 PUBLIC 指令声

明了的公共符号,则必须采用 EXTRN 指令将这些公共符号列出来。指令格式如下：

EXTRN 段_类型(符号_表),...

3. NAME 指令

NAME 指令用来标明当前程序模块。指令格式如下：

NAME 目标模块名

目标模块名最多可包含40 个字符，第一个字符不能是数字。需要注意的是 NAME 所标明的是模块名而不是文件名，每一个模 块只允许有一个模块名，如果源程序中没有给出模块名，则以不带扩展名的文件名作为模块名。

控制汇编状态的指令：

1. END 指令

END 指令用来控制汇编结束。在每个汇编语言程序的最后一行必须有一条 END 指令，并且 END 指令只能出现一次，如果程序 中使用了多个 END 指令，则 A51 只对第一个 END 指令前面的程序 进行汇编，以后的程序行将被忽略。

2.ORG 指令

ORG 指令用来改变汇编器的计数器，从而设定一个新的程序 起始地址。指令格式如下：

ORG 表达式

表达式必须是绝对或简单再定位表达式。表达式中不允许有 前向参考量，并且只能使用绝对或当前段符号。如果当前段是绝对段 ，表达式的结果值作为新的地址计数器的绝对值。如果当前段是再定位段，则表达式的结果值作为的地址偏移，绝对地址要由 L51 确定。用 ORG 指令可以改变地址计数器的值，但不会产生一个新的段，因此当前段中可能会存在地址间隙。在绝对段中使用 ORG 指令时，表达式的结果值不能小于绝对段的基地址。

3.USING 指令

USING 指令通知汇编器使用 8051 的哪一个工作寄存器组。指 令格式如下：

USINU表达式

A51 中还有两类段选择指令再定位段选择指令和绝对段选择指令。它们用来选择当前段是再定位段还是绝对段，使用不同的段选择指令将使程序定位在不同的地址空间之内。

再定位段选择指令为 RSEG，它用于选择一个已在前面定义 过的再定位段作为当前段，指令格式如下：

RSEG 段名

绝对段选择指令为 CSEG、DSEG、XSEG、ISEG 和 BSEG，分别选择绝对代码段、内部绝对数据段、外部绝对数据段、内部间接寻址绝对数据段和绝对位寻址数据段。指令格式如下：

CSEG [AT 绝对地址表达式]

DSEG [AT 绝对地址表达式]

XSEG [AT 绝对地址表达式]

ISEG [AT 绝对地址表达式]

BSEG [AT 绝对地址表达式]

附录二　C51 库函数

附录列出 Keil C51 的所有库函数供参考。(如果要了解其详细资 料，请参考有关的书籍)。Keil c51 软件包中提供了如下库函数文件：

C51S.LIB:不包含浮点运算的小型库函数

C51FPS。LIB：包含浮点运算的小型库函数

C51C。LIB：不包含浮点运算的紧凑型库函数

C51FPC。LIB：包含浮点运算的紧凑型库函数

C51L。LIB：不包含浮点运算的大型库函数

C51FPL。LIB：包含浮点运算的大型库函数

80751。LIB：应用于 Philips8X751 系列单片机的库函数

字符函数（CTYPE.H:）

extern bit isalpha (char);

检查参数字符是否为英文字母，是则返回 1，否则返回 0。

extern bit isalnum (char) ;

检查参数字符是否为英文字母或数字字符，是则返回 1，否则返回 0。

extern bit iscntrl (char);

检查参数值是否在 oxoo～oxlf 之间或等于 ox7f，如果为真则返回值为 1，否则返回值为 0。

extern bit isdigit (char);

检查参数的值是否为数字字符，是则返回 1，否则返回 0。

extern bit isgraph (char);

检查参数是否为可打印字符，可打印字符的值域为 ox21～ox7e。为真时返回值为 1，否则返回值为 0。

extern bit isprint (char);

除了与 isgraph相同之外，还接受空格符 (ox20)。

extern bit ispunct (char);

检查字符参数是否为标点、空格或格式字符。如果是空格或是 32 个标点和格式字符之一（假定使用 ASCII字符集中 128 个标准字符）则返回 1，否则返回 0。

extern bit islower (char);

检查参数字符的值是否为小写英文字母，是则返回 1，否则返回 0。

extern bit isupper (char);

检查参数字符的值是否为大写英文字母，是则返回 1，否则返回 0。

extern bit isspace (char);

检查参数字符是否为下列之一 空格、制表符、回车、换行、垂直制表符和送纸。如果为真则返回 1，否则返回 0。

extern bit isxdigit (char);

检查参数字符是否为 16 进制数字字符，如果为真则返回 1，否则返回 0。

extern char toint (char) ;

将 ASCII 字符的 0~9、a~F(大小写无关)转换为 16 进制数字。

extern char tolower (char) ;

将大写字符转换成小写形式，如果字符变量不在‘A’~‘Z’之间，则不作转换而直接返回该字符。

extern char toupper (char) ;

将小写字符转换为大写形式，如果字符变量不在‘a’~‘z’之间则不作转换而直接返回该字符。

define toascii (c) ((c)&0x7f)

该宏将任何整型数值缩小有效的ASCII 范围之间,它将变量和 0x7f 相与从而去掉第 7 位以上的所有数位。

define tolower (c) (c 一 ‘A’ + ‘a’)

该宏将字符 C 与常数 ox20 逐位相或。

#define toupper (c) ((c) 一 ‘a’ + ‘A’)

该宏将字符 C 与常数 oxdf 逐位相与。

一般 I/O 函数（STDIO.H）

extern cher_getkey ();

从 8051 的串口读入一个字符并返回入字符，然后等待字符输入，这个函数是改变整个输入端口机制时应作修改的唯一一个函数。

extern char_getkey();

从 8051 的串口读入一个字符并返回读入的字符,然后等待字符输入 ，这个函数是改变整个输入端口机制时应进行修改的唯一一个函数。

extern cher getchar ();

getchar 使用_getkey从串口读入字符，并将读入的字符马上传给putchar函数输出，其它与 _getkey 函数相同。

extern char * gets(char * s,int n);

该涵数通过 getchar串口读入一个长度为 n 的字符并存入由"s"指向的数组

extern cher ungetchar (char);

将输入字符回送输入缓冲区。

extern char putchar(char);

通过 8051串口输出字符，与函数 _getkey一样，这是改变整个输出机制所需修改的唯一一个函数。

extern int printf(const char *,…);

printf以一定的格式通过 8051的串行口输出数值和字符串，返回 值为实际输出的字符数

extern int sprintf(char* s,const char *,…);

与 print 的功能相似，但数据不是输出到串行口，而是通过一个指 针s，送入可寻址的内存缓冲区，并以 ASCLL码的形式存储。

extern int puts(const char * s);

利用 putcchar函数将字符串和换行符写入串行口，错误时返回 EOF，否则返回 0。

extern int scanf(const char * ,…);

利用 getchar函数从串行口读入数据，

extern int sscanf(char* s, const char *,…);

sscanf与 scanf 的输入方式相似但字符串的输入不是通过串行口而是通过另一个以空结束的指针。

extern void vprintf(const char * s, char * argptr);

vprintf()利用 putchar()函数输出格式化字符串和数据值。

extern void vsprintf(char * s,const char * fmtstr,char * argptr);

vprintf()将格式化字符串和数字值输出缓冲区内，返回值为实际写入到输出字符串的字符数。

字符串函数（STRING.H）

extern void * memchr (void * s1, char val, int len);
memchr 顺序搜索字符串 ‘s1’ 的头 ‘len’ 个字符以找出字符 ‘val’, 成功时返回 s1 中指向 val 的指针, 失败时返回 NULL。
extern char memcmp (void* s1, void * s2, int len);
再入属性 reentrant/intrinsic
memcmp逐个字符比较串 s1 和 s2 的前 len 个字符，成功(相等)时返回 0, 如果串 s1 大于或小于 s2，则相应地返回一个正数或一个负数 。
extern void * memcpy (void * dest, void* src, int len);
再入属性 reentrant

memcpy从 src 所指向的内存中拷贝len个字符到 dest 中，返回指向 dest 中最后一个字符的指针。如果 src 和 dest 发生交迭，则结果是不可预测的。
extern void * memccpy (void * dest, void * src, char val, int len);
memccpy 拷贝 src 中 len 个元素到 dest 中。如果实际拷贝了 len 个字符则返回 NULL。拷贝过程在拷贝完字符 val 后停止，此时返回指向 dest 中下一 个元素的指针。
extern void * memmove (void dest, void * src, int len);
再入属性 reentrant/intrinsic
memmove 的工作方式于 memcpy 相同, 但拷贝的区域可以交迭。
extern void * memset (void * s, char val, int len);
再入属性 reentrant/intrinsic
memset 用 val 来填充指针 s 中 len 个单元。
extern void * strcat (char * s1, char * s2);
strcat 将串 s2 拷贝到 s1 的尾部。strcat 假定 s1 所定义的地址区域足以接受两个串。返回指向 s1 串中第一个字符的指针。
extern char * strncat (char* s1, char* s2, int n);
strncat 拷贝串 s2 中 n 个字符到 s1 的尾部，如果 s2 比 n 短，则只拷贝 s2（包括串结束符）。
extern char strcmp (char* s1, car * s2);

再入属性 reentrant/intrinsic

strcmp比较串 s1和 s2，如果相等则返回0，如果s＜s2，则返回一个负数如果 s1＞s2，则返回一个正数。

extern char strncmp(char* s1, char * s2,int n);

strncmp比较串 s1 和 s2 中的前 n 个字符。返回值与 strcmp相同。

extern char * strcpy(char* sl, char * s2);

再入属性 reentrant/intrinsic

strcpy 将串 s2，包括结束符，拷贝到s1 中，返回指向s1 中第一个字符的指针。

extern char * strncpy (char* s1, char* s2,int n);

strncpy与 strcpy相似，但它只拷贝 n 个字符。如果 s2 的长度小于 n ，则 s1 串以‘0’补齐到长度 n。

extern int strlen (char * sl);

strlen返回串 s1 中的字符个数，包括结束符。

extern char * strchr (char * sl, char c);

extern int strpos (char * sl, char c);

再入属性 reentrant

strchr搜索 sl 串中第一个出现的字符‘c’，如果成功则返回指向该字符的指针，否则返回 NULL。

extern char * strrchr (char* sl, char c);

extern int strrpos (char* sl, char c);

strrchr搜索 sl 串中最后一个出现的字符 ‘c’，如果成功则返回指向该字符的指针，否则返回 NULL。

extern int strpsn (char* s1 ,char * set);

extern int strcspn (char* sl, char* set);

extern char * strpbrk (char* sl, char * set);

extern char * strrpbrk (char * sl, char * set);

srtpsn搜索 sl 串中第一个不包括在 set 串中的字符，返回值是 sl 中包括在 set 里的字符个数。如果 sl 中所有字符都包括在 set 里面，则返回 sl 的长度(不包括结束符)。如果 set 是空串则返回 0。

标准函数（STDLIB.H）

extern bouble atof (char* sl);

atof 将 sl 串转换成浮点数值并返回它。输入串中必须包含与浮点值规定相符的数。C51 编译器对数据类型 float 和 bouble 相同对待。

extern long atol (char* sl);

atol 将 sl 串转换成一个长整型数值并返回它，输入串中必须包含与长整型数格式相符的字符串。

extern int atoi (char* sl);

atoi 将 sl 转换成整型数值并返回它，输入串中必须包含与整型数格式相符串。

void* calloc(unsignedint n, unsigned int size);

calloc 返回为 n 个具有 size 大小对象所分配的内存的指针，如果返回 NULL，则表明无这么多内存空间可用。所分配的内存区域用 0 进行初始化。

void free (void xdata* p)

free 释放指针 p 所指向的存储器区域，如果 p 为 NULL，则该函数无效，p 必须是以前用 calloc、malloc 或 realloc 函数分配的存储器区域。

void init_mempool(void xdata* p,unsigned int size);

init_mempool 对可被函数 calloc、free、malloc 和 realloc 管理的存储器区域进行初始化，指针 p 表示存储区的首地址，size 表示存储区的大小。

void* malloc(unsignedint size);

malloc 返回为一个 size 大小对象所分配的内存指针。如果返回 NULL，则无足够的内存空间可用。内存区不作初始化。

void* realloc(void xdata* p,unsigned int size);

realloc 改变指针 p 所指对象的大小。

extern int rand();

Rand 返回一个 0 到 32767 之间的伪随机数，对 rand 的相继调用，将产生相同序列的随机数。

extern void srand(int n);

Srand 用来将随机数发生器初始化成一个已知（或期望）值。

数学函数（MATH.H）

```
  extern int abs (int val);
  extern char cabs (char val);
  extern floaf fabs (floaf val);
  extern long labs (long val);
```
再入属性 reentrant
ads计算并返回 val的绝对值。
```
  extern float exp(float x);
  extern float log(float x);
  extern float log10 (float x);
```
eap返回以e为底x的幂，log返回x的自然对数(e=2.718282),log10返回以10为底x的对数。
```
  extern float sqrt (float x);
```
sqrt返回 x 的正平方根。
```
  extern int rand ();
  extern void srand (int n) ;
```
rand返回一个 0 到 32767之间的伪随机数。
```
  extern float cos (float x);
  extern float sin (float x);
     extern float tan (float x);
```
cos返回 x 的余弦值，sin返回 x 的正弦值，tan返回 x 的正切值。
```
    extern float acos (float x);
    extern float aisn (float x);
    extern float atan (float x);
         extern float atan2 (float y, float x);
```
acos返回 x 的反余弦值,aisn返回 x 的反正弦值,atan返回 x 的反正切值。
```
  extern float cosh (float x);
  extern float sinh(float x);
  extern float tanh(float x);
```
cosh返回 x 的双曲余弦值，sinh返回 x 的双曲正弦值，tanh返回 x 的双曲正切值。
```
  extern void fpsave (struct FPBUF*p) ;
```

extern void fprestore (struct FPBUF *p) ;

fpsave保存浮点子程序的状态，fprestore恢复浮点子程序的原始状态。

extern float ceil (float x) ;

ceil返回一个不小于 x 的最小整数（作为浮点数）。

extern float floor (float x) ;

floor返回一个不大于 x 最大整数（作为浮点数）。

extern float modf (float x,float*ip)；

modf 将浮点数 x 分成整数和小数两部分。

extern float pow (float x,float y) ;

pow 计算 X^y 的值，如果变量的值不合要求，则返回 NaN，当 x= =0 且 y＜=0 或当 x＜0 且 y 不是整数时会发生错误。

6.1.6 绝对地址访问 ABSACC.H

define CBYTE ((unsigned char *)0x50000L)

define DBYTE ((unsigned char *)0x40000L)

define PBYTE ((unsigned char *)0x30000L)

define XBYTE ((unsigned char *)0x20000L)

上述宏定义用来对 8051 系列单片机的存储器空间进行绝对地址访问,可以作字节寻址。

define CWORD ((unsigned int *)0x50000L)

define DWORD ((unsigned int *)0x40000L)

define PWORD ((unsigned int *)0x30000L)

define XWORD ((unsigned int *)0x20000L)

这个宏与前面一个宏相似，只是它们指定的数据类型为unsigned int。通过灵活运用不同的数据类型，所有的 8051地址空间都可以进行访问。

内部函数（INTRINS.H)

unsigned char_crol_(unsignedchar val, unsigned char n);

unsigned int_irol_(unsignedint val, unsigned char n);

unsigned long_lrol_(unsignedlong val, unsigned char n);

crol,_irol_和_lrol_将变量 val循环左移 n 位,它们与 8051单片机的“RLA”指令相关。

unsigned char _cro_crol_, _irol和_lrol_将变量 val 循环右移 n 位。

void _nop_(void);

_nop_产生一个 8051 单片机的 NOP 指令。

dit_testbit_(ditx);

_testbit_产生一个 8051 单片机的 JBC 指令

r_(unsigned char val, unsigned char n);

unsigned int _irol_(unsignedint val, unsigned char n);

crol, _irol_和_lrol_将变量 val 循环右移 n 位。

void _nop_(void);

_nop_产生一个 8051 单片机的 NOP 指令。

dit_testbit_(ditx);

_testbit_产生一个 8051 单片机的 JBC 指令

变量参数表(STDARG.H)

typedef char * va_list

va_list被定义成指向参数表的指针。

#define va_start(ap,v) ap=(va_list)&v+sizeof(v)

宏 va_start初始化指向参数的指针。

#define va_arg(ap,t) (((t*)ap)++[0])

宏 va_arg从 ap 指向的参数表中返回类型为 t 的当前参数。

#define va_end(ap)

关闭参数表，结束对可变参数表的访问。

全程跳转(SETJMP.H)

extern int setjmp(jmp_buf env);

setjmp将程序执行的当前环境状态信息存入变量 env 之中。

extern void longjmp(jmp_buf env,int val);

longjmp恢复调用 setjmp 时存在 env 中的状态。

访问 SFR 和 SFR-bit 地址的 REGXXX。H

头文件 REGXXX。H 中定义了许多 8051 单片机中所有的特殊功能寄存器（SFR）名，从而可简化用户