第 10 章　汇编语言编程和调试工具

学习汇编语言的目的就是要用汇编语言程序来解决实际问题，下面以 Microsoft 的 MASM 6.15 为基础来介绍汇编语言程序的开发过程，以及汇编语言程序的调试工具。

10.1　汇编语言的开发环境

10.1.1　开发过程

在此先介绍在 DOS 环境下的汇编语言程序开发过程，再介绍 Windows 下 32 位汇编语言程序的汇编、连接方法。实验环境中，MASM 安装在 D:\MASM 目录下。

1. 编写源程序

可用计算机系统中各种能编辑文本文件的编辑器来编辑汇编源程序。常用的编辑器有：EDIT、记事本、写字板、Word 和 WPS 等。源文件的类型为：.ASM。

2. 汇编程序

当源程序编写好后，可用 MASM 命令来汇编该源程序。如果源程序没有语法错误，那么，将生成目标文件(类型为.OBJ)，为最终生成可执行文件作准备；如果源程序有错误，汇编程序将显示出错误位置和原因，也可用列表文件(类型为.LST)来查看出错位置和原因。

下面给出一些使用该命令的实例。(用户输入的命令用“下画线”来标识)

例 10.1　查看 MASM 命令的功能。

```
MASM /?
…
/Zi Generate symbolic information for CodeView
/Zd Generate line-number information
```

其中：选项/Zi 和/Zd 与符号跟踪有关，所以在调试程序中经常使用这两个选项。

例 10.2　用 MASM 命令汇编源程序。

```
MASM Exam.ASM
…
Invoking: ML.EXE /I. /Zm /c Exam.ASM
…
Assembling: Exam.ASM
```

如果 MASM 命令显示了类似如上的处理结果，那么表示源文件 Exam.ASM 已成功汇编，并已生成了其目标文件 Exam.OBJ。

例 10.3　用 MASM 命令汇编源程序。

```
MASM Exam.ASM
…
Invoking: ML.EXE /I. /Zm /c Exam.asm
```

```
…
Assembling: Exam.ASM
Exam.ASM(5): error A2070: invalid instruction operands
```

如果 MASM 命令显示了类似如上的处理结果，那么，表示源文件有错，没有生成其目标文件。在本例中，显示第 5 行有语法错：非法的指令操作数。这时，要检查源程序的第 5 行，看看输入指令时是否有误。

假如源程序有许多错误，很难记住全部出错位置，那么，可借助列表文件来排错。

例 10.4 在汇编源程序的同时，生成其列表文件。

```
MASM Exam.ASM, ,Exam
…
Assembling: Exam.ASM
Exam.ASM(5): error A2070: invalid instruction operands
```

列表文件 Exam.lst 是一个文本文件，可用编辑器直接阅读，并可看出其错误的位置和原因。下面是一个列表文件的实例。

```
EDIT Exam.lst
Microsoft (R) Macro Assembler Version 6.15.8803    09/28/08 12:35:48
Exam.ASM                                                Page 1 - 1
0000                            _TEXT   SEGMENT
                                        ASSUME CS:_TEXT
 0000  B4 01                    Start:  MOV     AH, 1
 0002  CD 21                            INT     21h
                                        MOV     DL, AX
Exam.ASM(5) : error A2070: invalid instruction operands
 0004  80 EA 20                         SUB     DL, 32
 0007  B4 02                            MOV     AH, 2
 0009  CD 21                            INT     21h
 000B  B8 4C00                          MOV     AX, 4C00h
 000E  CD 21                            INT     21h
 0010                                   _TEXT   ENDS
                                        END     Start

Microsoft (R) Macro Assembler Version 6.15.8803    09/28/08 12:35:48
Exam.ASM                                                Symbols 2 - 1
Segments and Groups:
            Name    Size    Length      Align       Combine Class
_TEXT.............. 16 Bit    0010        Para          Private
Symbols:
    N a m e Type    Value   Attr
Start.............  L Near  0000    _TEXT
        0 Warnings
        1 Errors
```

3. 连接程序

当源文件汇编成功后，即可用连接程序(LINK.EXE)生成可执行文件。

例 10.5 查看连接程序(LINK.EXE)的具体选项。

```
LINK /?
```

```
LINK <objs>,<exefile>,<mapfile>,<libs>,<deffile>
Valid options are:
 /?                     /ALIGNMENT
 /BATCH                 /CODEVIEW
 /CPARMAXALLOC          /DOSSEG
 /DSALLOCATE            /DYNAMIC
 /EXEPACK               /FARCALLTRANSLATION
 /HELP                  /HIGH
 /INFORMATION           /LINENUMBERS
 /MAP                   /NODEFAULTLIBRARYSEARCH
 /NOEXTDICTIONARY       /NOFARCALLTRANSLATION
 /NOGROUPASSOCIATION    /NOIGNORECASE
 /NOLOGO                /NONULLSDOSSEG
 /NOPACKCODE            /NOPACKFUNCTIONS
 /NOFREEMEM             /OLDOVERLAY
 /ONERROR               /OVERLAYINTERRUPT
 /PACKCODE              /PACKDATA
 /PACKFUNCTIONS         /PAUSE
 /PCODE                 /PMTYPE
 /QUICKLIBRARY          /SEGMENTS
 /STACK                 /TINY
 /WARNFIXUP
```

例 10.6　用连接程序生成执行文件。

方法 1：

```
link Exam.ASM
…
  Run File [Exam.exe]:
  List File [nul.map]:
  Libraries [.lib]:
  Definitions File [nul.def]:
  LINK : warning L4021: no stack segment
```

这种方法需要确认连接过程中的各种文件名，如果使用文件名的默认值，那么直接按“回车”键即可。在上面四个文件名中，最重要的两个文件名是：可执行文件名和库文件名。一般情况下，无须更换最终生成的执行文件名；如果在连接过程中需要其它的库文件，则在显示第三行提示时，输入所需要的库文件名。

最后一行显示的警告信息：没有堆栈段，这是因为源程序中没有定义堆栈段。一般情况下，该警告信息可以不必理会，因为操作系统在装入运行程序时会为其安排堆栈段。

方法 2：在文件名后面加上分号“;”，使用各类文件的默认名。

```
LINK Exam;
  …
  LINK : warning L4021: no stack segment
```

4. 运行程序

当要运行生成的可执行文件时，可直接输入其文件名即可。

```
Exam
```

5. 符号调试程序

当程序的运行结果不是预期结果时，就需要调试程序，找出错误的语句或逻辑关系。MASM 系统提供了可用于源程序一级的调试工具 CV(CodeView)。有关 CV 的使用参见“调试工具”中的介绍。

例 10.7 若 Exam.ASM 汇编通过，可用如下命令生成含有 CV 调试信息的执行文件。

```
MASM /Zi /Zd Exam;
LINK /Co Exam
CV Exam.exe
```

在 MASM 6.15 中，MASM 和 LINK 已整合在一起，即 ML.EXE，也就是说可以用 ML 来汇编源程序和连接目标程序。ML 命令的格式如下：

```
ML [ /options ] filelist [ /link linkoptions ]
```

ML 的选项较多，这里只列出几个常用的选项。

/c	仅汇编不连接;	/Zi	目标文件调试用的源程序行号；
/Zd	目标文件含有调试信息；	/omf	生成 OMF 格式的目标文件；
/coff	生成 COFF 格式文件；	/Bl<linker>	指定连接程序。

例 10.8 用 ML 汇编、连接源程序。

```
ML Exam.ASM
```

先汇编源程序，生成 Exam.OBJ，再连接(调用 LINK.EXE)，生成 Exam.EXE。

以上介绍的是汇编语言程序汇编、连接的一般方法。

ML.EXE 也可以用来汇编和连接 Win32 的源程序，此时需注意以下几个方面。

(1) 用/coff 选项，以便生成 COFF 格式的目标码，没有此选项，则默认为/omf。

(2) 用 Win32 的连接程序(MASM 6.15 软件包中的 Link.EXE 用来连接 16 位程序)，以及相应的连接设置。例如，使用 VC 中的 Link.EXE(改名为 Link32.EXE 以便区别)，以及连接选项/subsystem:console 或/subsystem:windows。

(3) 在 Win32 中不提供 DOS 系统功能调用及 BIOS 功能调用，只能使用 Win32 的 API，所以在连接时要给出包含 API 相应的函数库(如 Uusr32.LIB, Kernel32.EXE 等)。在实验环境中，将 Win32 的相应的函数库放在 Lib32 目录中。

例 10.9 用 ML 汇编、连接源程序 E32.ASM，生成 Win32 格式的运行程序。

```
ML /c /coff e32.ASM
LINK32 /subsystem:console /libpath:d:\masm\lib32 32.OBJ
```

或者用 ML 汇编并连接：

```
ML /coff /Bl Link32.EXE e32.ASM /link /libpath:d:\masm\lib32 /subsystem:console
```

先汇编生成 COFF 格式的 e32.OBJ，再调用 Link32.EXE 生成 e32.EXE。

10.1.2 VC 中汇编集成环境的设置

通过如下设置(以 VC 6.0 为例)，可用 VC 集成环境来开发 Win32 的汇编语言程序。

(1) 创建一个空 Project，根据需要选一个类型，如图 10.1 所示，选 Win32 Application 类

型，创建一个名为 Demo 的空 Project。

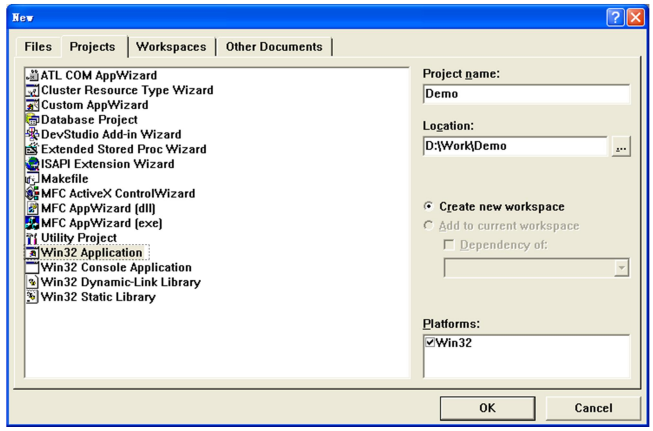

图 10.1　创建 Project 对话框

(2) 将源程序加入到 Source Files 中，如图 10.2 所示，加入了 e32.ASM。

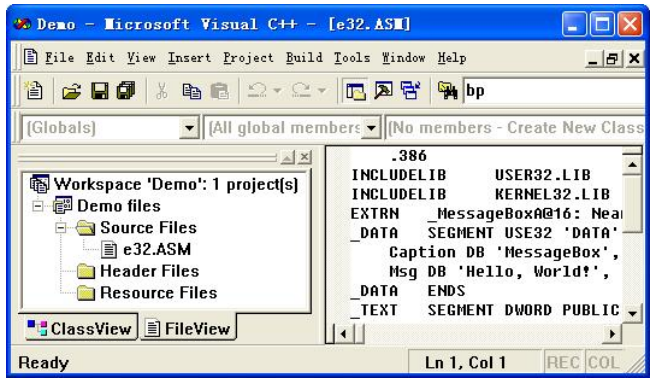

图 10.2　将 e32.ASM 加入到 Source Files

(3) e32.ASM 的 Debug 和 Release 设置。

Win32 Debug 的设置如下：(如图 10.3 所示)

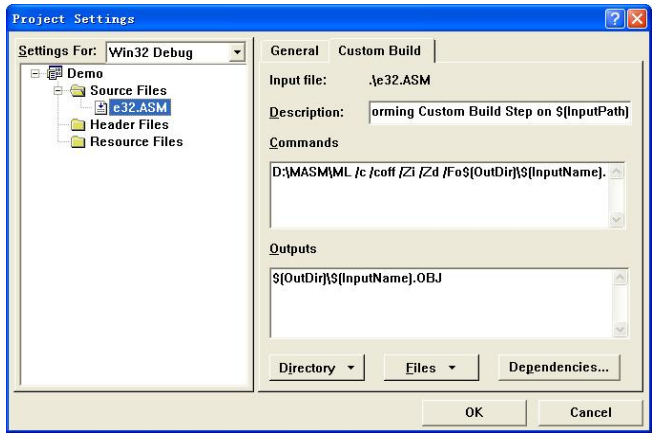

图 10.3　在 Win32 Debug 下 e32.ASM 的 Custom Build 设置

Commands： D:\MASM\ML /c /coff /Zi /Zd /Fo$(OutDir)\$(InputName).OBJ $(InputPath)
Outputs： $(OutDir)\$(InputName).OBJ
Win32 Release 的设置如下：(如图 10.4 所示)
Commands： D:\MASM\ML /c /coff /Fo$(OutDir)\$(InputName).OBJ $(InputPath)
Outputs： $(OutDir)\$(InputName).OBJ

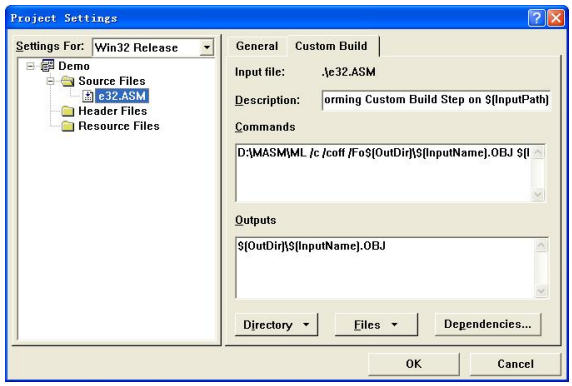

图 10.4 在 Win32 Release 下 e32.ASM 的 Custom Build 设置

e32.ASM 的 Custom Build 设置完毕后，就可以如同 C 程序一样来调试汇编源程序了。

10.2 调 试 工 具

汇编语言程序的调试工具较多，这里仅介绍 16 位环境下使用较多的两种调试工具。

10.2.1 DEBUG

DEBUG 是 DOS 下的调试工具，只能处理 16 位地址，以及显示 8086 的指令及寄存器。DEBUG 的命令格式如下：

```
DEBUG  文件名 [参数表]
```

其中：文件名指定被调试的文件；参数表是被调试文件运行时所需要的参数。

被调试的文件可以是系统中的任何文件，但通常它们的后缀为.EXE 或.COM。

运行 DEBUG 后显示的提示符是“-”，此时可输入各种命令，见表 10.1。

表 10.1 DEBUG 命令及其含义

命 令	格 式	功 能 说 明
A	[地址]	输入汇编指令，未指定地址则接着上次输入
C	源内存块范围 目标起始地址	两个内存块内容比较，显示内容不同的字节单元地址
D	[内存范围]	显示指定范围内的内存内容
E	地址 [字节值表]	从“地址”开始，以字节为单位显示并修改单元内容
F	内存范围 字节值表	用指定的字节值表来填充内存块
G	[=起始地址] [断点地址]	从起点(或当前地址)开始执行，直到断点或终点

续表

命　令	格　　式	功 能 说 明
H	数值 1 数值 2	显示两个十六进制数值之和、差
I	端口地址	从端口输入
L	[地址 [驱动器号 扇区 扇区数]]	从磁盘读
M	源块范围 目标起始地址	源内存块内容传送到目标内存块
N	文件标识符 [文件标识符…]	指定文件名，为读/写文件做准备
O	端口地址 字节值	向端口输出
P	[=地址] [指令数]	按执行过程，但不进入子程序调用或软中断
Q		退出 DEBUG
R	[寄存器名]	显示和修改寄存器内容（r:看所有寄存器 “r 寄存器”：看某个寄存器冒号后可直接改寄存器内容）
S	内存块范围 字节值表	在内存块内搜索指定的字节值表
T	[=地址] [指令数]	跟踪执行，从起点(或当前地点)执行若干条指令
U	[范围]	反汇编，显示机器码所对应的汇编指令
W	[地址 [驱动器号 扇区 扇区数]]	向磁盘写内容，BX:CX 为写入的字节数

修改标识位：
-R F
NU UP EI PL NZ NA PO NC -直接修改的内容

关于参数的几点说明：

① 进制：在 DEBUG 中输入或显示的数据都是十六进制形式。

② 分隔：命令和参数、参数和参数之间要用空格、逗号或制表符等分隔。

③ 地址：用“段:偏移”表示，如 1A00:0，或“段寄存器:偏移”表示，如 DS:0。

④ 范围：有两种表示方式。

- 起始地址 结束地址，例如，10:0 100，表示 10:0～100 共 101 字节的内存块；
- 起始地址 L 长度，例如，10:0 L100，表示 10:0～FF 共 100 字节的内存块。

⑤ 字节值表：由若干个字节值组成，也可以是用'或者"括起来的字符串。

⑥ 驱动器号：0——驱动器 A、1——驱动器 B、2——驱动器 C、3——驱动器 D 等。

例 10.10　启动 DEBUG，并装入 Test.exe 文件(假设该文件已存在)。

方法 1：

```
DEBUG Test.exe
-
```

方法 2：

```
DEBUG
-N test.exe
-L
```

例 10.11　用 A 命令输入汇编指令。

```
-A 100
0AE1:0100    MOV     CX, 100
0AE1:0103    ADD     AX, CX
0AE1:0105    LOOP 103
0AE1:0107 (直接按回车结束输入)
```

例 10.12　比较 DS:0～10 的内存块与从 100:20 开始的内存块内容。

-C DS:0 10 100:20　或　-C DS:0 L11 100:20

例 10.13　检查 1000:0 开始的 3 字节内容，并置 41, 42, 43。

-E 1000:0
1000:0000 CD.41　20.42　FF.43 (每个字节以空格结束，最后以回车键结束)
注：每个字节用空格键结束，再移至后一个字节；或'-'结束，再移至前一个字节。

例 10.14　显示以 2000:0～F 内存块内容，再用'abc'来填充它。

-D 2000:0 L10

2000:00008A 04 0E 02 FF 03 0E 02 - 76 09 0E 02 B1 98 00 C0 ……v……

-F 2000:0 F 'abc'　或　-F 2000:0 F 61 62 63

例 10.15　将 SS:0～40 内存块内容传送到 ES:10 开始的内存块中。

-M SS:0 40 ES:10

例 10.16　R 命令的使用示例，参见表 10.2。

-R　　显示所有 16 寄存器内容
AX=00CD　BX=0000　CX=0000　DX=0000　SP=FFEE　BP=0000　SI=0000　DI=0000
DS=0AE1　ES=0AE1　SS=0AE1　CS=0AE1　IP=0102①　NV UP EI NG NZ NA PO NC②
0AE1:0102　　0101　　ADD　　[BX+DI],AX　DS:0000=4241
注①：指示正准备执行的指令的存放地址；
注②：Flags 只显示 OF,DF,IF,SF,ZF,AF,PF,CF 的状态，所用的符号列示在表 10.2 中；

-R AX　　显示 AX 寄存器内容，并修改为 1234。(直接输入回车则不修改)
AX 00CD
:1234
-R F　　显示 FLAGS 内容，并将 ZF,CF 置 1。(直接输入回车则不修改)
NV UP EI NG NZ NA PO NC - CY ZR

表 10.2　DEBUG 中标志位的符号表示

标　志　位	OF	DF	IF	SF	ZF	AF	PF	CF
置 1	OV	DN	EI	NG	ZR	AC	PE	CY
清 0	NV	UP	DI	PL	NZ	NA	PO	NC

例 10.17　反汇编 CS:100～10A 的内容。

-U CS:100 10A　(显示的格式：左列地址，中间列是机器代码，右列是汇编指令)
0AE1:0100　B90001　MOV　CX,0100
0AE1:0103　01C8　ADD　AX,CX
0AE1:0105　E2F9　LOOP　0103
0AE1:0107　A30010　MOV　[1000],AX
0AE1:010A　CC　INT 3

例 10.18　执行指令的演示示例。

-G 0AE1:100　　从 0AE1:100 开始执行，直到程序终止
-G 0AE1:100 107　从 0AE1:100 开始执行，直到 107 处停止，或程序终止
-T 0AE1:100　　从 0AE1:100 开始单步执行 1 条指令
-T 0AE1:100 8　从 0AE1:100 开始单步执行 8 条指令
-T　　从当前地址开始单步执行 1 条指令
-P 0AE1:100 8　从 0AE1:100 开始单步执行 8 条指令，不进入子程序/中断内部

10.2.2　CodeView

CodeView 是一个多窗口的全屏幕调试工具，其功能比 DEBUG 强大得多。可调试多种语言的源程序，支持 16 位地址模式下的各种指令。允许用户运行程序或单步执行，可以设置断点，在程序运行期间查看并修改内存或寄存器内容。DEBUG 中的大部分命令均可在 CodeView 的 Command 窗口内执行。

1. CodeView 的启动和退出

DOS 环境中输入 CV Exam.EXE 即可进入 CodeView 环境，其界面布局如图 10.5 所示。退出 CodeView 可以选择 File 选单中的 Exit 选项，或在 Command 窗口中输入 Q 命令，此时，系统返回到 DOS 提示符。

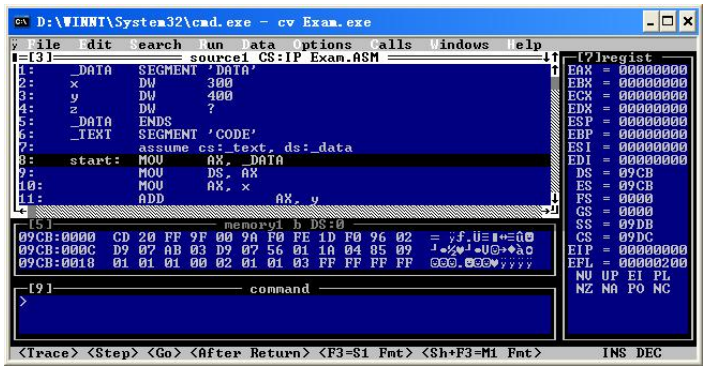

图 10.5　CodeView 界面布局

2. CodeView 工具的各窗格

图 10.5 显示了 4 个窗格，其中左上侧 source1 是主窗格，显示被调试的源程序；左侧中部的 memory1 窗格用于显示内存单元的内容；左下侧的 command 窗格中可以输入前面介绍的 Debug 调试命令；右侧的 register 窗格显示寄存器的内容。

除了这 4 个窗格，CodeView 中还有其他窗格，分别有各自不同的作用和功能。在 Windows 选单中可以看到 CodeView 中的窗格列表，0～9 共 10 个，对应的快捷方式为 Alt+数字。

0. Help　　提供 CodeView、汇编语言等有关帮助信息。
1. Local　　列出当前所有的局部变量。可通过 Option 选单改变当前范围。
2. Watch　　查看执行期间变量或表达式的值。可通过 Data 选单添加/删除。
3. Source1　　可显示源程序及对应的机器代码。可由 Option 选单设置。
4. Source2　　同 Source1。主要用于查看程序的不同部分。

5. Memory1　用于显示内存单元内容。可通过 Option 选单进行设置。
6. Memory2　同 Memory1。
7. Register　显示寄存器组的内容。可通过 Option 选单设置 16/32 寄存器。
8. 8087　显示 FPU 的浮点寄存器组的内容。
9. Command　可输入 DEBUG 命令。

3. 功能键

F1　获得帮助信息。
F2　显示/隐含寄存器组窗口。
F3　Source 窗口中代码的三种显示方式的切换。
F4　显示程序的输出屏幕。
F5　相当于 DEBUG 的 G 命令，执行到下一个逻辑断点，或到程序终止。
F6　依次进入当前屏幕所显示的窗口。
F7　与 F5 功能相同。
F8　相当于 DEBUG 的 T 命令，单步执行指令。
F9　设置/取消断点，用鼠标左键双击之也可。
F10 相当于 DEBUG 的 P 命令，单步执行指令，不进入的子程序内部。