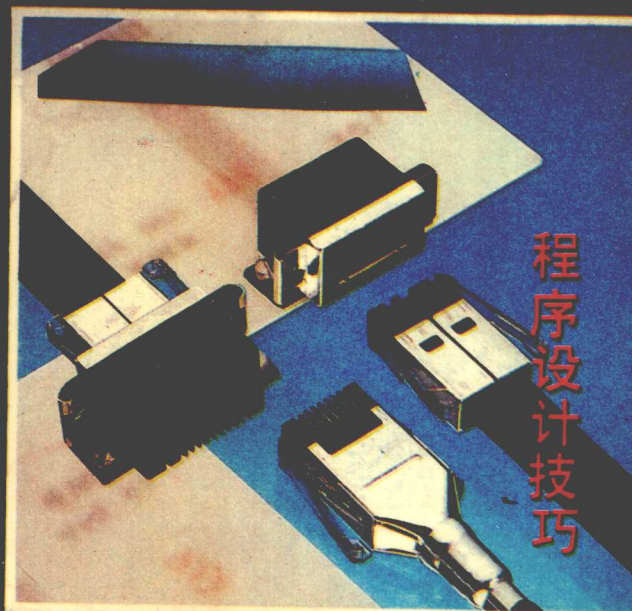


Turbo Assembler

汇编大全

(中册)



程序设计技巧



中国科学院希望高级电脑技术公司

一九九〇年九月



Turbo Assembler 汇编大全

程序设计技巧

(中册)

汪亚文 谢 康 译
文 韬 校

中国科学院希望高级电脑技术公司
一九九〇年九月

前言

Turbo Assembler 是 Borland 公司推出的可与 MicroSoft 公司的 **MASM** 相匹敌的汇编语言软件。从某种角度来看, **Turbo Assembler** 的总体性能要比 **MASM** 好。**Turbo Assembler** 的速度是 **MASM** 所远不能比拟的。**Turbo Assembler** 与 **MASM** 完全兼容, 带有可选的 **Ideal** 模式。

Turbo Assembler 支持 **Turbo Debugger** 源级调试器, 方便、高效的调试工具会帮助你减轻调试汇编原会有的负担。汇编的繁琐你将不再会感觉到, 代之以汇编的高速、对计算机的完全控制。

Turbo 系列高级语言的使用者若想优化代码、提高速度、使用低级功能, 请使用 **Turbo Assembler**, 因为 **Turbo Assembler** 使 **Turbo** 系列高级语言与汇编的接口简化。

<<**Turbo Assembler** 大全>>分成三个部分: <<**Turbo Assembler** 用户指南>>, <<**Turbo Assembler** 程序设计技巧>>, <<**Turbo Assembler** 使用手册>>。

本书介绍使用 **Turbo Assembler** 所需的基本指令, 基本的汇编程序设计方法。

由于编译器水平有限, 加上时间仓促, 错误难免, 欢迎指正。

文韬

1990.8.

目 录

前言

第六章 Turbo Assembler 与 Turbo C 的接口	1
§ 6.1 在 Turbo C 中使用嵌入式汇编	1
§ 6.1.1 嵌入式汇编如何工作	3
§ 6.1.1.1 Turbo C 如何知道使用嵌入式汇编模式	6
§ 6.1.1.2 激活 Turbo Assembler 处理嵌入式汇编	7
§ 6.1.1.3 Turbo C 在何处汇编嵌入式汇编码	7
§ 6.1.1.4 将-L 开关用于 80186/80286 指令	8
§ 6.1.2 嵌入式汇编语句的格式	9
§ 6.1.2.1 嵌入式汇编中的分号	9
§ 6.1.2.2 嵌入式汇编中的注解	9
§ 6.1.2.3 访问结构/联合的元素	10
§ 6.1.3 嵌入式汇编示例	12
§ 6.1.4 嵌入式汇编的限制	15
§ 6.1.4.1 内存和地址操作数限制	15
§ 6.1.4.2 嵌入式汇编中缺少隐含的自动变量大小	16
§ 6.1.4.3 必须保存寄存器	18
§ 6.1.4.3.1 保存调用函数和寄存器变量	18
§ 6.1.4.3.2 抑制内部寄存器变量	18
§ 6.1.5 嵌入式汇编码相对于纯 C 代码缺点	18
§ 6.1.5.1 降低了可移植性和可维护性	18
§ 6.1.5.2 降低了编译速度	18
§ 6.1.5.3 仅可由 TCC 使用	19
§ 6.1.5.4 损失了优化能力	19
§ 6.1.5.5 限制了对错误的反跟踪	19
§ 6.1.5.6 调试限制	19
§ 6.1.5.7 用 C 开发而用嵌入式汇编编译最终代码	20
§ 6.2 在 Turbo C 中调用 Turbo Assembler 函数	20
§ 6.2.1 Turbo C 与 Turbo Assembler 的接口机制	21
§ 6.2.1.1 内存模式和段	21
§ 6.2.1.1.1 简化的段伪指令与 Turbo C	21
§ 6.2.1.1.2 过时风格的段伪指令与 Turbo C	23
§ 6.2.1.1.3 段缺省: 何时需要装载段?	25
§ 6.2.1.2 公共量和外部量	28
§ 6.2.1.2.1 下划线	28
§ 6.2.1.2.2 大小写字母的意义	29

§ 6.2.1.2.3 标号类型·····	29
§ 6.2.1.2.4 远类型的外部量必须在任何段之外·····	30
§ 6.2.1.3 链接器命令行·····	32
§ 6.2.2 Turbo Assembler 与 Turbo C 的交互性·····	32
§ 6.2.2.1 参数传递·····	32
§ 6.2.2.2 保存寄存器·····	38
§ 6.2.2.3 返回值·····	39
§ 6.2.3 从 Turbo C 中调用 Turbo Assembler 函数·····	40
§ 6.2.4 Pascal 调用约定·····	43
§ 6.3 在 Turbo Assembler 中调用 Turbo C·····	44
§ 6.3.1 链入 C 的启动码·····	44
§ 6.3.2 确保已正确设置了段·····	45
§ 6.3.3 执行调用·····	45
§ 6.3.4 在 Turbo Assembler 调用 Turbo C 函数·····	46
第七章 Turbo Assembler 与 Turbo Pascal 的接口·····	49
§ 7.1 Turbo Pascal 内存映象·····	49
§ 7.1.1 程序段前缀·····	49
§ 7.1.2 代码段·····	49
§ 7.1.3 全局数据段·····	50
§ 7.1.4 堆栈·····	50
§ 7.1.5 堆·····	51
§ 7.2 Turbo Pascal 中寄存器的用法·····	51
§ 7.3 近调用还是远调用?·····	51
§ 7.4 与 Turbo Pascal 共享信息·····	51
§ 7.4.1 \$L 编译伪指令和外部子程序·····	51
§ 7.4.2 PUBLIC 伪指令: 使 Turbo Pascal 可利用 Turbo Assembler 的信息·····	52
§ 7.4.3 EXTRN 伪指令: 使 Turbo assembler 可利用 Turbo Assembler 的信息·····	53
§ 7.4.3.1 使用 EXTRN 对象的限制·····	55
§ 7.4.4 使用段定位·····	56
§ 7.4.5 无效代码的消除·····	56
§ 7.5 Turbo Pascal 参数传递约定·····	56
§ 7.5.1 值参·····	56
§ 7.5.1.1 标量类型·····	57
§ 7.5.1.2 实型·····	57
§ 7.5.1.3 单精度、双精度、扩展的和复合型: 8087 类型·····	57
§ 7.5.1.4 指针·····	57
§ 7.5.1.5 串·····	57
§ 7.5.1.6 记录和数组·····	57
§ 7.5.1.7 集合·····	57
§ 7.5.2 变量参数·····	58

§ 7.5.3 栈的维护	58
§ 7.5.4 存取参数	58
§ 7.5.4.1 使用 BP 寄存器寻址堆栈	58
§ 7.5.4.1.1 ARG 伪指令	59
§ 7.5.4.1.2 .MODEL 和 Turbo Pascal	60
§ 7.5.4.1.3 使用另一个基址或变址寄存器	60
§ 7.6 Turbo Pascal 中的函数结果	61
§ 7.6.1 标量函数结果	61
§ 7.6.2 实型函数结果	61
§ 7.6.3 8087 函数结果	61
§ 7.6.4 串函数结果	61
§ 7.6.5 指针函数结果	61
§ 7.7 为局部数据分配空间	61
§ 7.7.1 分配私有静态存储区	61
§ 7.7.2 分配动态存储区	62
§ 7.8 由 Turbo Pascal 调用汇编语言子程的例子	63
§ 7.8.1 通用十六进制转换子程序	63
§ 7.8.2 交换两个变量	66
§ 7.8.3 扫描 DOS 环境	69
第八章 Turbo Assembler 与 Turbo Basic 的接口	74
§ 8.1 传递参数	74
§ 8.1.1 不在当前数据段的变量	76
§ 8.1.2 什么类型的调用?	76
§ 8.2 弹出堆栈	76
§ 8.3 为 Turbo Basic 创建一个汇编程序	77
§ 8.4 调用一个在线汇编子程序	77
§ 8.5 在内存中安装一个 Turbo Basic 子程序	79
§ 8.5.1 隐藏串	80
§ 8.5.2 绝对调用(CALL ABSOLUTE)	81
§ 8.5.2.1 到一固定内存位置作 CALL ABSOLUTE	81
§ 8.5.2.2 到内存不定位置作 CALL ABSOLUTE	82
§ 8.5.2.3 CALL ABSOLUTE 的其它问题	83
§ 8.6 调用中断	83
§ 8.7 样本程序	84
第九章 Turbo Assembler 与 Turbo Prolog 的接口	87
§ 9.1 声明外部谓词	87
§ 9.2 调用约定和参数	87
§ 9.2.1 命名约定	88
§ 9.3 写汇编语言谓词	88
§ 9.3.1 实现 double 谓词	91

§ 9.4 用多重流模式实现谓词·····	93
§ 9.5 从汇编函数调用 Turbo Prolog 谓词·····	95
§ 9.5.1 表和函子·····	97
第十章 Turbo Assembler 高级程序设计·····	101
§ 10.1 段前缀·····	101
§ 10.1.1 一种可选形式·····	102
§ 10.1.2 在什么情况下段前缀并不起作用·····	103
§ 10.1.3 访问多个段·····	104
§ 10.2 局部标号·····	105
§ 10.3 自动确定转移大小·····	108
§ 10.4 超前引用代码和数据·····	113
§ 10.5 使用重复块和宏·····	116
§ 10.5.1 重复块·····	116
§ 10.5.1.1 重复块和可变参数·····	119
§ 10.5.2 宏·····	120
§ 10.5.2.1 嵌套宏·····	124
§ 10.5.2.2 宏与条件句·····	125
§ 10.5.2.3 用 EXITM 终止扩展·····	126
§ 10.5.2.4 在宏内定义标号·····	127
§ 10.6 良好的数据结构·····	128
§ 10.6.1 STRUC 伪指令·····	129
§ 10.6.1.1 使用 STRUC 的好处和坏处·····	132
§ 10.6.1.1.1 结构域名维一·····	133
§ 10.6.1.1.2 嵌套结构·····	133
§ 10.6.1.1.3 结构初始化·····	134
§ 10.6.2 RECORD 伪指令·····	136
§ 10.6.2.1 访问记录·····	137
§ 10.6.2.1.1 WIDTH 算子·····	139
§ 10.6.2.1.2 MASK 算子·····	139
§ 10.6.2.2 为什么要使用记录·····	140
§ 10.6.3 UNION 伪指令·····	142
§ 10.7 段伪指令·····	145
§ 10.7.1 SEGMENT 伪指令·····	145
§ 10.7.1.1 name 和 align 域·····	146
§ 10.7.1.2 combine 域·····	146
§ 10.7.1.3 use 和 class 域·····	147
§ 10.7.1.4 段长度、段类型、段名和段嵌套·····	147
§ 10.7.2 段排序·····	149
§ 10.7.3 GROUP 伪指令·····	150
§ 10.7.4 ASSUME 伪指令·····	152

§ 10.7.5 简化的段伪指令·····	156
§ 10.7.6 多段程序示例·····	160
第十一章 80386 及其它处理器 ·····	165
§ 11.1 用汇编语言代码切换处理器类型·····	165
§ 11.2 80186 和 80188·····	166
§ 11.2.1 启动 80186 汇编·····	166
§ 11.2.2 新增伪指令·····	166
§ 11.2.2.1 PUSHA 和 POPA·····	166
§ 11.2.2.2 ENTER 和 LEAVE·····	167
§ 11.2.2.3 BOUND·····	168
§ 11.2.2.4 INS 和 OUTS·····	169
§ 11.2.3 8086 指令的扩展形式·····	170
§ 11.2.3.1 压入立即数·····	170
§ 11.2.3.2 移位量和循环量可以是立即数·····	170
§ 11.2.3.3 乘以立即数·····	171
§ 11.3 80286·····	172
§ 11.3.1 启动 80286 汇编·····	172
§ 11.4 80386·····	173
§ 11.4.1 选择 80386 汇编模式·····	173
§ 11.4.2 新增段类型·····	173
§ 11.4.2.1 简化的段伪指令和 80386 段类型·····	177
§ 11.4.2.2 48 位数据类型 FWORD·····	178
§ 11.4.3 新增寄存器·····	180
§ 11.4.3.1 32 位通用寄存器·····	180
§ 11.4.3.2 32 位标志寄存器·····	182
§ 11.4.3.3 32 位指令指针·····	182
§ 11.4.3.4 新段寄存器·····	183
§ 11.4.4 新的寻址方式·····	185
§ 11.4.5 新增加的指令·····	189
§ 11.4.5.1 位测试·····	189
§ 11.4.5.2 位扫描·····	190
§ 11.4.5.3 带符号扩展或零扩展的数据移动·····	191
§ 11.4.5.4 转换成双字或四倍字数据·····	192
§ 11.4.5.5 多字移位·····	192
§ 11.4.5.6 条件设置字节·····	194
§ 11.4.5.7 装配 SS、FS 和 GS·····	194
§ 11.4.6 扩展指令·····	195
§ 11.4.6.1 特殊版本的 MOV 指令·····	196
§ 11.4.6.2 32 位版本的 8086 指令·····	197
§ 11.4.6.2.1 新版本的 LOOP 和 JCXZ·····	197

§ 11.4.6.2.2 新版本的串指令	198
§ 11.4.6.2.3 IRETD	199
§ 11.4.6.2.4 PUSHFD 和 POPFD	199
§ 11.4.6.2.5 PUSHAD 和 POPAD	199
§ 11.4.6.3 新版本的 IMUL	199
§ 11.4.7 混合使用 16 和 32 位指令和段	200
§ 11.4.8 80386 函数示例	202
§ 11.5 80287	207
§ 11.6 80387	207
第十二章 Turbo Assembler 中的 Ideal 模式	208
§ 12.1 什么是 Ideal 模式?	208
§ 12.2 为什么要使用 Ideal 模式?	208
§ 12.3 进入和退出 Ideal 模式	209
§ 12.4 MASM 模式与 Ideal 模式之间的区别	210
§ 12.4.1 Ideal 模式下的标记符	210
§ 12.4.1.1 符号标记符	210
§ 12.4.1.2 重复成员名	211
§ 12.4.1.3 浮点数标记符	211
§ 12.4.2 正文等价符和数字等价符(EQU 和 =伪指令)	211
§ 12.4.3 表达式和操作数	212
§ 12.4.3.1 在方括号[]算子	212
§ 12.4.3.2 操作数示例	212
§ 12.4.4 算符	214
§ 12.4.4.1 结构成员中的句号	214
§ 12.4.4.2 结构指针	214
§ 12.4.4.3 SYMTYPE 算子	214
§ 12.4.4.4 HIGH 算子和 LOW 算子	214
§ 12.4.4.5 可选的 PTR 算子	215
§ 12.4.4.6 SIZE 算子	215
§ 12.4.5 伪指令	216
§ 12.4.5.1 列表控制符	216
§ 12.4.5.2 以句点(.)开始的伪指令	217
§ 12.4.5.3 伪指令名与符号名的反序	217
§ 12.4.5.4 作为伪指令参量的引用串	218
§ 12.4.6 段和段组	218
§ 12.4.6.1 访问属于段组的段中的数据	219
§ 12.4.7 定义近代码标号或远代码标号	220
§ 12.4.8 外部符号、公用符号和全程符号	221
§ 12.4.9 其它方面的区别	222
§ 12.4.9.1 抑制定位	222

§ 12.4.9.2 BOUND 指令的操作数.....	222
§ 12.4.9.3 宏内的注释.....	222
§ 12.4.9.4 局部符号.....	223
§ 12.5 MASM 模式与 Ideal 模式下程序设计的对比.....	223
§ 12.5.1 MASM 模式下的程序示例.....	223
§ 12.5.2 Ideal 模式下的程序示例.....	224
§ 12.5.3 对 MASM 模式和 Ideal 模式的剖析.....	226

第六章 Turbo Assembler 与 Turbo C 的接口

许多程序设计者能够——而且的确——用汇编语言编写所有的程序；而另外一些程序设计者则更愿意用高级语言完成繁重的工作，只有在需要低级控制或高效率的代码时才使用汇编语言；还有一些程序设计者主要用汇编语言进行程序设计，只偶而用到高级语言库和高级语言结构。

作为一种实际需要，Turbo C 对 C 语言与汇编语言的混合使用提供了良好的支持，它对汇编代码与 C 代码的嵌合使用提供了不只一种、而是两种支持机制。Turbo C 的嵌入式汇编特征提供了一种简单而明快的方法，使用这种方法可以直接将汇编代码放入 C 语言函数中。对于宁愿全部使用汇编语言设计各独立模块的汇编程序设计者而言，可将 Turbo Assembler 模块分别汇编，然后与 Turbo C 代码相链接。

本章首先介绍在 Turbo C 中如何使用嵌入式汇编，然后详细讨论如何将分别汇编过的 Turbo Assembler 模块与 Turbo C 相链接，并剖析在 Turbo C 代码中调用 Turbo Assembler 函数的过程，最后介绍如何在 Turbo Assembler 代码中调用 Turbo C 函数。（注意：凡涉及到 Turbo C 时，均指 1.5 或更高版本的 Turbo C。）下面开始具体讨论这几方面的内容。

§ 6.1 在 Turbo C 中使用嵌入式汇编

如果用户试图想象出使用汇编语言改进 C 语言程序的理想方法，用户可能会想到，如果能将汇编语言指令插入 C 代码中的关键位置，那么汇编代码的高速性和低级控制特性一定能明显地改进程序的性能。同时，用户可能希望避免汇编语言与 C 语言接口传统的复杂性，而且希望不改变任何 C 代码就可以做到上述几点，这样就不必改变已可以正常运行的 C 代码。

Turbo C 的嵌入式汇编特征可以满足用户的各种愿望。嵌入式汇编仅仅是一种可将任何汇编代码放入 C 程序的任何位置，并可以全面访问 C 语言常量、变量甚至函数的能力。的确，嵌入式汇编可以极大地改进程序的性能，因为它与严格地用汇编语言编写的程序几乎具有同样强大的功能；例如，Turbo C 库中高性能的程序代码就是用嵌入式汇编完成的。使用嵌入式汇编，用户可以在 C 程序中按自己的意愿加入汇编语句，而不必考虑两者之间的接口。

考虑下面的 C 代码，它是嵌入式汇编的一个例子：

```
i=0;                /*set i to 0 (in C) */
asm dec WORD PTR i;  /*decrement i (in assembler) */
i++;                /* increment i (in C) */
```

第一行和最后一行是正常的 C 语句，但中间行呢？正如用户可能猜想的一样，以 asm 为

开始的行是嵌入式汇编代码行。如果用调试器查看由该 C 程序的源代码所编译出的可执行码,可以发现

```
mov WORD PTR [bp-02],0000
dec WORD PTR [bp-02]
inc WORD PTR [bp-02]
```

其中,嵌入式汇编码 DEC 指令在编译出的代码

```
i=0;
```

和

```
i++;
```

之间。

从根本上看,每当 Turbo C 编译程序碰到标识嵌入式汇编的关键字 `asm` 时,它就将相关的汇编行直接插到编译后的代码中,只有一点发生变化:对 C 变量的引用被转换成与之等价的适当的汇编语言形式,正如前一个例子中,对变量 `i` 的引用被转换成了 `WORD PTR [BP-2]` 一样。简言之,用户可以用 `asm` 关键字将任意汇编码插到 C 代码任意位置。(对嵌入式汇编码所完成的动作也有一些限制,“嵌入式汇编的限制”一节将讨论这些限制。)

将汇编代码直接插入到 Turbo C 产生的代码中,这似乎有些危险,而事实上,嵌入式汇编也的确有其冒险之处。尽管 Turbo C 仔细地编译其代码,以避免与嵌入式汇编码交互时所潜在的危险,但毫无疑问,功能性错误的嵌入式汇编码肯定会引起严重的错误。

另一方面,任何编写得很粗糙的汇编语言代码,无论是嵌入式汇编语句还是单独的汇编模块,其运行都具有潜在的盲目性和破坏性;这是汇编语言的高速度性和低级控制能力所付出的代价。除此之外,相对于纯汇编码中出现的错误而言,嵌入式汇编码中出现的错误并不常见,因为 Turbo C 注意到了许多程序设计的细节,例如进入和退出函数、传递参数、分配变量的地址等。总之,在 C 代码中加入嵌入式汇编码可以方便地改进程序的性能,尽管用户不得不花一定的代价去排除偶然的汇编语言错误,但花这种代价仍然是值得的。

嵌入式汇编中重要的几点说明:

1. 为了使用嵌入式汇编,用户必须激活 Turbo C 的命令行版本,即 `TCC.EXE`。而 Turbo C 的用户接口版本 `TC.EXE` 并不支持嵌入式汇编。
2. 用户所拥有的 Turbo Assembler 拷贝盘中带有的 `TLINK` 很可能与 Turbo C 拷贝中带有的 `TLINK` 不是同一个版本。既然为了支持 Turbo Assembler,在 `TINK` 中增加了一些重要的功能,同时,无疑还会进一步增加新的功能,所以重要的是,用户最好用自己所拥有的最新版本的 `TLINK` 链接包含有嵌入式汇编行的 Turbo C 模块。最安全的方法是,用户务必确保存放链接程序的盘上只有一个 `TLINK.EXE` 文

件；而此 TLINK.EXE 文件应该是用户拥有的 Borland 公司的其它产品随带的 TLINK.EXE 文件中版本号最新的那一个。

§ 6.1.1 嵌入式汇编如何工作

通常情况下，Turbo C 直接将每个源 C 代码文件编译成目标文件，然后激活 TLINK 将这些目标文件链接成可执行的程序。图 6.1 描述了这种编译——链接过程。要开始这种过程，用户可输入命令行

TCC filename

该命令行指示 Turbo C 先将 FILENAME.C 编译成 FILENAME.OBJ，然后激活 TLINK 将 FILENAME.OBJ 链接成 FILENAME.EXE。

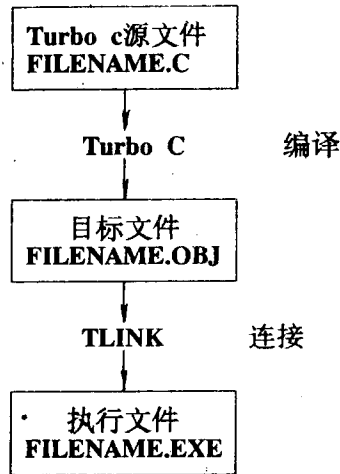


图 6.1 Turbo C 编译和连接循环

但使用嵌入式汇编时，Turbo C 会自动在编译——链接链中加入一个步骤。

Turbo C 在处理每个含有嵌入式汇编代码的模块时，首先将整个模块编译成汇编语言源文件，然后激活 Turbo Assembler 将产生的汇编码汇编成目标文件，最后激活 TLINK 对这些目标文件进行链接。图 6.2 描述了该过程，即描述了 Turbo C 如何根据含有嵌入式汇编代码的 C 源文件产生一个可执行文件。要开始此过程，用户可打入命令行

TCC -B filename

该命令行指示 Turbo C 先编译产生 FILENAME.ASM，再激活 Turbo Assembler 将 FILENAME.ASM 汇编成 FILENAME.OBJ，最后激活 TLINK 将 FILENAME.OBJ 链接成 FILENAME.EXE。

Turbo C 只是将嵌入式汇编码传递给汇编语言文件。这种机制的精华在于 Turbo C 无须了解怎样汇编嵌入式代码；相反，Turbo C 将 C 代码编译成与嵌入式汇编码同级的代码——汇编语言代码——并让 Turbo Assembler 完成汇编工作。

要了解 Turbo C 究竟怎样处理嵌入式汇编，用户可输入取名为 PLUSONE.C 的下列程序：

```
#include <stdio.h>
int main(vvoid)
{
```

```

int TestValue;
scanf("%d",&TestValue);          /* get the value to increment */
asm inc WORD PTR TestValue /* increment it (in assembler) */
printf("%d",TestValue);           /* print the incremented value */
}

```

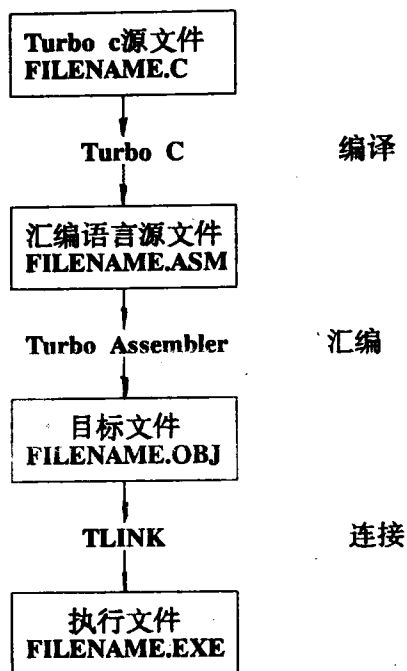


图 6.2 Turbo C 编译、汇编和连接

然后用命令行

TCC -S plusone

对其进行编译。可选项-S 指示 Turbo C 在将 C 文件在编译成汇编代码之后停下，所以文件 **PLUSONE.ASM** 现在处于用户盘上。用户可以发现 **PLUSONE.ASM** 形式如下：

```

ifndef ??version
?debug macro
    ENDM
    ENDIF
name Plusone
_TEXT SEGMENT BYTE PUBLIC 'CODE'
DGROUPGROUP _DATA, BSS
    ASSUME CS:_TEXT, DS:DGROUP, SS:DGROUP
_TEXT ENDS
_DATA SEGMENT WORD PUBLIC 'DATA'
_D@ LABEL BYTE
_D@W LABEL WORD

```

```

_DATA ENDS
_BSS SEGMENT WORD PUBLIC 'BSS'
_b@ LABEL BYTE
_b@w LABEL WORD
      ?debug C E90156E11009706C75736F6E652E63
      ?debug C E96009B9100F696E636C7564655C737464696F2E68
      ?debug C E90009B91010696E636C7564655C7374646172672E68
_BSS ENDS
_TEXT SEGMENT BYTE PUBLIC 'CODE'
;      ?DEBUG L 3
_main PROC NEAR
      push    bp
      mov     bp,sp
      dec     sp
      dec     sp
;      ?debug L 8
      lea     ax, WORD PTR [bp-2]
      push    ax
      mov     ax, OFFSET DGROUP:_s@
      push    ax
      call    NEAR PTR _scanf
      pop     cx
      pop     cx
;      ?debug L 9
      inc     WORD PTR [bp-2]
;      ?debug L 10
      push    WORD PTR [bp-2]
      mov     ax, OFFSET DGROUP:_s@+3
      push    ax
      call    NEAR PTR _printf
      pop     cx
      pop     cx
@1:
;      ?debug L 12
      mov     sp,bp
      pop     bp
      ret
_main ENDP
_TEXT ENDS
_DATA SEGMENT WORD PUBLIC 'DATA'

```

```

_s@ LABEL BYTE
    DB 37
    DB 100
    DB 0
    DB 37
    DB 100
    DB 0
_DATA ENDS
_TEXT SEGMENT BYTE PUBLIC 'CODE'
EXTRN _printf:NEAR
EXTRN _scanf:NEAR
_TEXT ENDS
    PUBLIC _main
    END

```

(Turbo C 支持嵌入式汇编，所以为用户做了大量的工作。看此代码之后，想必用户对 Turbo C 所做的全部工作一定十分欣赏吧！)在注释行

```

;debug L 8

```

之下，用户可以看到 Scanf 调用的汇编码，其后是

```

;debug L 9

```

```

inc WORK PTR [bp-2]

```

这是递增 TestValue 的嵌入式汇编指令。(注意，Turbo C 自动将 C 语言变量 TestValue 转化成该变量的等价汇编地址[BP-2]。)在嵌入式汇编指令之后是 printf 调用的汇编码。

重要的是要了解，Turbo C 将 Scanf 调用编译成汇编语言，将嵌入式汇编码直接插入到输出的汇编文件中，然后将 Printf 调用编译成汇编语言。得到的结果文件是一个有效的汇编源文件，这样就可以用 Turbo Assembler 进行汇编。

如果不用可选项-S，Turbo C 将直接激活 Turbo Assembler 汇编 PLUSONE.ASM，然后激活 TLINK 将活到的目标文件 PLUSONE.OBJ 链接成可执行文件 PLUSONE.EXE。这是 Turbo C 对嵌入式汇编进行处理的一般模式；使用-S 可选项只是为了解释这一过程，以便观察在处理嵌入式汇编时 Turbo C 所使用的中间汇编语言步骤。当编译出将要链接成可执行程序代码时，可选项-S 并不是特别有用，但它提供了一种方便的手段，通常可以利用它检查嵌入式汇编码中的指令以及由 Turbo C 产生的代码指令。如果用户不清楚嵌入式代码转换后的形式，可以使用-S 可选项检查.ASM 文件。

§ 6.1.1.1 Turbo C 如何知道使用嵌入式汇编模式

一般情况下，Turbo C 直接将 C 代码编译成目标代码。有多种方法可以告知 Turbo C 支持嵌入式汇编，先将源文件编译成汇编语言，然后激活 Turbo Assembler。

命令行参数-B 指示 Turbo C 将 C 代码编译成汇编代码，再激活 Turbo Assembler 汇编产生的代码，从而得到目标文件。

命令行参数-S 指示 Turbo C 将 C 代码编译成汇编码后停止运行。使用-S 参数时，由 Turbo C 产生的.ASM 文件可以分别汇编，并链接到其它 C 模块或汇编模块上。除了调试和检查之外，在使用了-B 参数后一般不再使用-S。

伪指令 #pragma

#pragma inline

与命令行可选项 -B 有同样的功能，它指示 Turbo C 将 C 代码编译成汇编代码，再激活 Turbo Assembler 汇编已得到的代码。当遇到 #pragma inline 时，Turbo C 在汇编输出模式下重新启动编译。因而，最好将伪指令 #pragma inline 尽可能放到 C 语言源代码的首部，因为以 #pragma inline 开头的任何 C 语言源代码均被编译两次，一次被正常地编译成目标码，一次被编译成汇编码。尽管这对其它任何过程都没有妨碍，但很费时间。

最后，当使用 -B、-S 和 #pragma inline 时，如果 Turbo C 接触到嵌入式汇编码，则编译器给出下列警告：

Warning test.c 6:Restarting compile using assembly in function main

然后以汇编输出模式重新编译，正如此时使用了伪指令 #pragma inline 一样。用户可以使用 -B 或 #pragma inline 避免这种警告，因为一碰到嵌入式汇编就开始重新编译相对而言要慢得多。

§ 6.1.1.2 激活 Turbo Assembler 处理嵌入式汇编

Turbo C 要激活 Turbo Assembler，首先必须找到 Turbo Assembler。但究竟怎样找到 Turbo Assembler 则依赖于 Turbo C 的不同版本而有所不同。

版本 1.5 以上的 Turbo C 希望在当前目录或 DOS 环境变量 PATH 所定义的目录之一中找到 TASM.EXE，即找到 Turbo Assembler。Turbo C 基本上可以在同一环境下激活 Turbo Assembler，用户也可以在命令行提示符下打入命令

TASM

运行 Turbo Assembler。所以，如果 Turbo Assembler 处于当前目录或命令搜索路径所表示的任一目录中，Turbo C 就可以自动寻找并运行 Turbo Assembler 处理嵌入式汇编。

在这一方面，1.0 和 1.5 版本的 Turbo C 稍有些不同，因为这两个版本的 Turbo C 出现于 Turbo Assembler 问世之前，它们通过激活 Microsoft Macro Assembler，即 MASM 来处理嵌入式汇编。所以，这两个版本的 Turbo C 在当前目录或命令搜索路径中寻找文件 MASM.EXE，而不是 TASM.EXE。因而，它们不能自动使用 Turbo Assembler。

注意：用户可阅读 Turbo Assembler 盘上的 README 文件，以了解如何修改这两个版本的 TCC，以便它能使用 TASM。

§ 6.1.1.3 Turbo C 在何处汇编嵌入式汇编码

嵌入式汇编码可以出现于 Turbo C 的代码段，也可以出现于 Turbo C 的数据段。出现在函数中的嵌入式汇编码被汇编到 Turbo C 的代码段，出现在函数之外的嵌入式汇编码被汇编到 Turbo C 的数据段。

例如，C 代码

```
/* Table of square Values */
asm SquareLookUpTable label word;
asm dw 0,1,4,9,16,25,36,49,64,81,100;
/* Function to loop up the square of a value between 0 and 10 */
int LoopUpSquare(int Value)
{
    asm mov bx,Value;                                /* get the value to square */
}
```