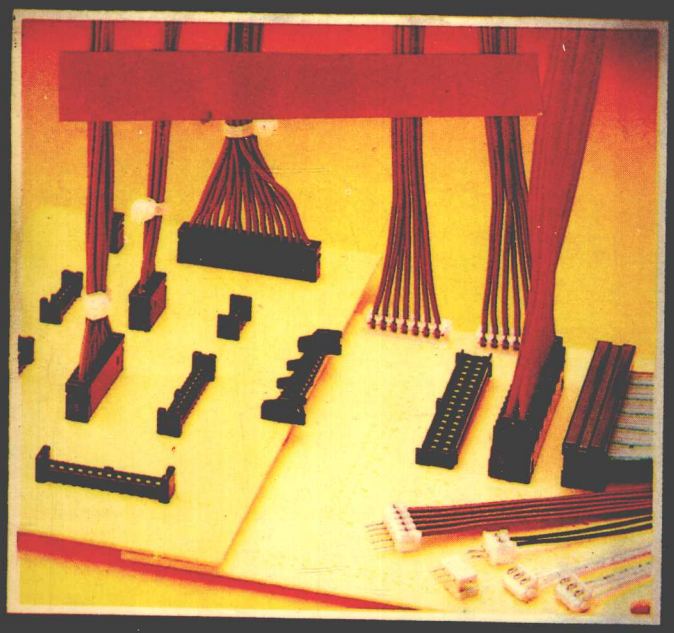
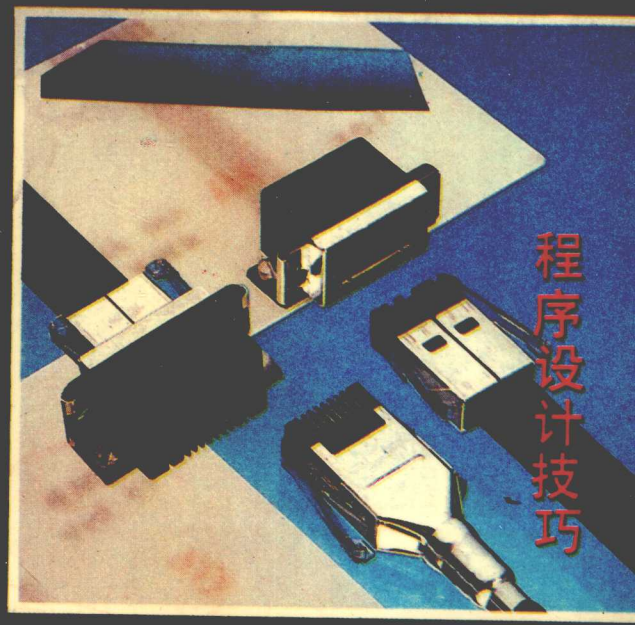
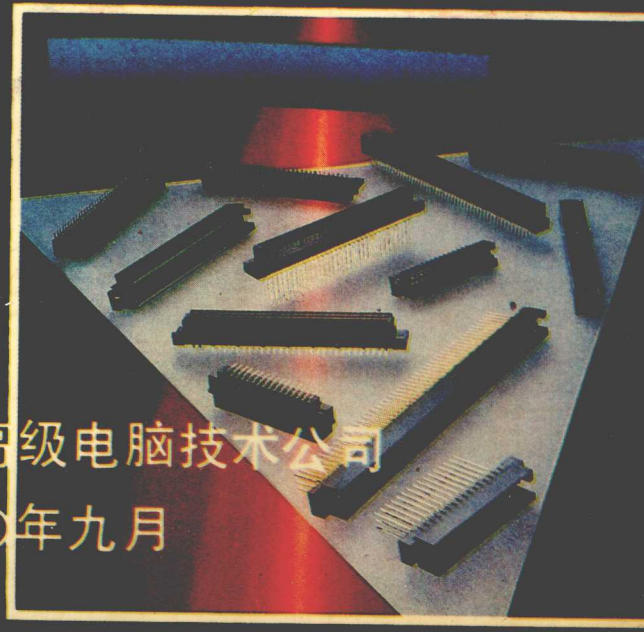
Turbo Assembler
汇 编 大 全

（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。

由于编译者水平有限，加上时间伦促，错误难免，欢迎指正。

文韬

1990.8.

目 录

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这几点，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而的汇编语言错误，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 Trubo 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 TLINK 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。

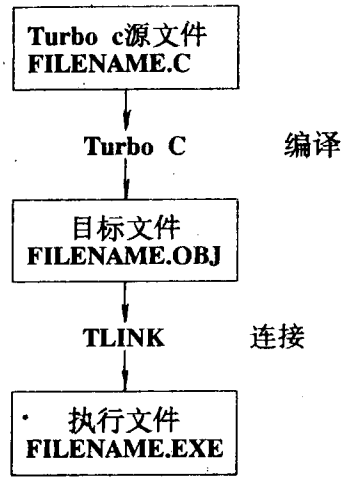


图 6.1 Turbo C 编译和连接循环

但使用嵌入式汇编时，Trubo 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(vvid)
{
```

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

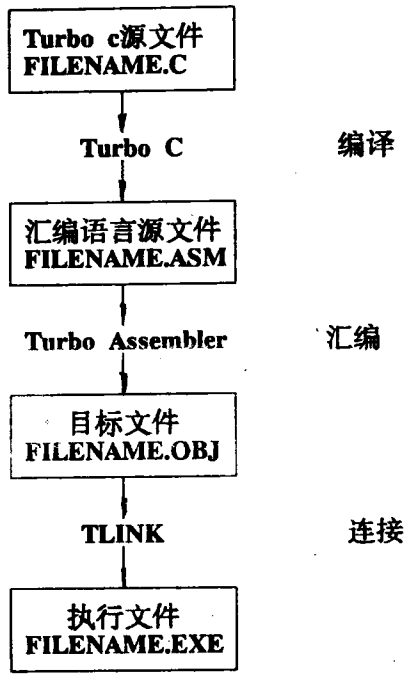


图 6.2 Turbo C 编译、汇编和连接

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 PLUSDNE.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 E90009B9100F696E636C7564655C737464696F2E68
        ?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的代码时，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 Maro 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        */
```