Turbo Assembler
(汇编)参考手册

从海莱 译

北京联想计算机集团公司
一九九〇年八月

Turbo Assembler(汇编)参考手册

丛海莱　译

北京联想计算机集团公司

一九九0年八月

目　录

此为试读,需要完整PDF请访问: www.ertongbook.com

第一章　预定义符

Turbo Assembler 提供了许多预定义符(predefined symhol)供程序使用。这些符号在源文件中不同位置可以有不同的值。它们类似用 EQU 指令(directive)所定义的符号。当 Turbo Assembler 在文件中遇到这些符号时，就把该预定义符当前所具有的值代进去。

这些符号有的是正文(Text)或字符串常量，有些是数值常量，另外还有别名(alias)值。字符串值可以随意在想使用的地方使用。例如使用 DB 指令来定义一串数据字节：

```
NOW     DB     ??Time
```

数值预定义符值可以随意在任何地方使用：

```
IF     ??Version     GT     100h
```

别名值则使预定义符变成它所代表值的同义字，可让你随便在任何想使用一般符号名称处使用：

```
ASSUME     CS:     @Code
```

所有的预定义符都可使用于 MASM 和 Ideal 模式下。

如果在编译时使用/ML 命令列选择项，则所使用的前置定义符号必须遵照以下规则。

以@字母开头的预定义符适用规则：组成符号名称各部分单词的第一个字母应该大写；单词中的其余字母为小写。例如：

```
@CurSeg
@FileName
```

至于以两个问号(??)开头的符号，在使用/ML 选择项时所有字母都必须小写。

在一般情况下，使用以两个问号(??)开头的符号是可以大小写并用的。只有/ML 命令行选择项例外：所有字母都必须小写。

@Code

功能　CODE 段名称的别名

说明　当使用简单分段指令(如.MODEL 等等)时，此别名可让你在诸如 ASSUME 及段覆盖(segment override)等表达式中使用程序码段名。

实例

```
.DATA
mov ax, @Code
mov ds, ax
ASSUME     DS: @Code
```

@CodeSize

功能　显示程序码的内存模块(code memory model)的数值常量。

说明　对于使用近程(near)程序指针(code pointer)的小型(small)或紧缩型(compact)内存模块，@CodeSize 被设为 0；而对于所有其他使用远程(Far)程序指针的模块，则@CodeSize 被设为 1。此符号(根据内存模块的形式)可以用来控制函数指针(pointer to function)被编译情况。

实例

```
IF @CodeSize EQ 0
```


此为试读，需要完整PDF请访问：www.ertongbook.com

```
PROCPTR DW PROC1      ;指向进程程序的指针
ELSE
PROCPTR DD PROC1      ;指向远程程序的指针
ENDIF
```

@Cpu

功能　返回当前处理器信息

说明　@Cpu 意义返回值中各位分别表示不同的处理器性能：

位	意义
0	使用8086指令
1	使用80186指令
2	使用80286指令
3	使用80386指令
7	使用特权指令(privileged instruction)(80286及80386)
8	8087数值处理器指令
10	80287数值处理器指令
11	80387数值处理器指令

此处未定义的位保留作将来使用。在使用@Cpu 时将这些未定义的位屏蔽(Mask)成 0，以便程序仍与 Turho Assembler 的未来版本相容。

由于 8086 类处理器是朝上相容的，因此在你使用.286 这类指令来启动处理器类型时，性能较低的处理器类型(80286、80186)也会自动被启动。

注意　这个符号只提供在编译时(assembly-time)(经由.286 或相关指令)所选择的处理器的信息。至于程序在运行时(run-time)的工作处理器类型并未显示。

实例
```
IPUSH=@Cpu AND 2      ;186 以上的处理器可运行
IF IPUSH              ;立即入堆栈(immediate push)
PUSH 1234
ELSE
        mov ax, 1234
          push ax
ENDIF
```

@CurSeg

功能　当前段的别名

说明　@CurSeg 在编译期间随需要而变，来反映当前段名称。通常在使用完简单分段指令(如.MODEU 等)之后会用到它。你可以利用@CurSeg 来完成 ASSUME 说明段覆盖或任何需要用到当前段名称的说明。

实例
```
.CODE
```

```
ASSUME CS: @CurSeg
```

@Data

功能　近程数据组名称的别名

说明　当你用到简单分段指令(如.MODEL 等)时，这个别名可让你在象 ASSUME 或段覆盖等表示式中使用由所有近程数据段(.DATA、.CONST、.STACK)共用的组名称。

实例
```
.CODE
mov ax, @Data
mov ds, ax
ASSUME DS: @Data
```

@DataSize

功能　显示数据内存模块

说明　对于使用近程数据指针的小型(Small)及中型(medium)内存模块，@DataSize 被设为 0，若是使用远程数据指针的紧缩(compact)及大型(large)，模块则被设为 1，至于巨型(huge)内存模块则@DataSize 被设为 2。

可以使用此符号(根据内存模块的形式)来控制数据指针被编译的情形。

实例
```
IF @DataSize EQ 1
    Lea si, Dataptr
    mov al, [BYTE PTR  ES: SI]
ELSE
    les si, Dataptr
    mov al, [BYTE PTR ES:SI]
ENDIF
```

??Date

功能　当前日期的字符串

说明　??Data 定义一个表示今天日期的字符串。日期字符串的实际格式由 DOS 的国家内码来决定。

参看　??Time

实例
```
ASMDATE DB ??Date      ;8 个字节的字符串
```

@FarData

功能　表示初始化(initalized)远程数据段名称的别名

说明　当你使用简单分段指令(如.MODEL 等)时，此别名可让你在诸如 ASSUME 或段覆盖的表示式中使用初始化的远程数据段(.FARDATA)名称。

实例
```
mov ax, @FarData
mov ds, ax
ASSUME DS: @FarData
```

@FarData?

功能　未初始化的(uninitialized)远程数据段名的别名

说明　当你使用简单分段指令(如.MODEL 等)时，这符号可让你在如 ASSUME 或段覆盖的表示式中使用未初始化的远程数据段名称。

实例
```
mov ax, @FarData?
mov ds, ax
ASSUME DS: @FarData?
```

@FileName

功能　当前编译文件(assembly file)的文件名

参看　??Filename

??Filename

功能　当前编译文件(assembly file)的文件名字符串

说明　??Filename 定义一个含 8 个字符的字符串来表示正被编译的文件名称。若文件名称小于 8 个字符，则以空格填入。

实例
```
SRCNAME DB ??Filename      ; 8 个字节
```

?? Time

功能　当前时间的字符串

说明　??Time 定义一表示当前时间的字符，时间字符串的实际格式由 DOS 的国家内码来决定。

参看　??Date

实例
```
ASMTIME DB ??Time      ; 8 字节的字符串
```

??Version

功能　表示所使用的 Turbo Assembler 版本号

说明　高位字节表示主要(major)版本号码，而低位字节则表示次要(minor)版本号码。例如 V2.1 被表示成 0201H。

??Version 可让你在撰写源程序时能尽量利用 Turbo Assembler 某特定版本的特性。

此符号还可以让你的原文件知道它们是否正在 MASM 或是在 Turbo Assembler 下编译，因为??Version 在 MASM 下是没有定义的。

实例
```
IFDEF ??Version
   ;TURBO ASSEMBLER STUFF
ENDIF
```

@WordSize

功能　显示 16 或 32 位段的数字

说明　若当前段是16位段则@WordSize返回2，若是32位的段则返回4。

实例

```
.IF @WordSize EQ 4
    mov esp, 0100h
ELSE
    mov sp, 0100h
ENDIF
```

第二章　操作符

操作符(operator)可将操作数(operand)和指令(instruction 或 directive)连在一起构成许多复杂的说明。操作符作用在操作数(如程序符号名或常数值等)上。当 Turbo Assembler 在编译源文件时，它会计算各表达式，然后用所得结果取代该表达式。当你在计算一个会随源程序修改而改变的值时，也可以使用这样的运算式。

本章将详细介绍 Turbo Assembler 提供的各操作符(运算符)。

算术精确度

Turbo Assembler 会根据你是否曾以.386 或.386P 指令启动 80386 处理器而决定采用 16 或 32 位算术运算。当你以 80386 处理器或在 Ideal 模式中编译程序时，有些说明得出的结果会和在 16 位模式中运算所得的有所不同；例如：

```
DW (1000h* 1000h/)  /1000h
```

在 32 位模式中将产生一个 1000h 的字节，而在 16 位模式中则为一个值为 0 的字节。这是因为在 16 位模式中前头的方法已经造成溢位(overflow)，存回去的只是所得结果的低 16 位。

操作符优先级

Turbo Assembler 以下列规则来计算运算式：

- □ 具有较高优先级(Precedence)的操作符比具有较低优先级的先执行
- □ 操作符的优先级相同时，则从表达式的最左边开始运算。
- □ 若运算式含有一包含于括号(parentheses)中的子运算式(subexpression)时，则此子运算式被最先运行，因为括号中的运算式具有最高的优先权(priority)。

有些操作符的优先级在 Ideal 模式和 MASM 模式中不太一样。下面两个表分别列出两种模式下操作符优先权的情况。两表中的第一行表示的操作符都表示它们具有最高的优先权，而最末行上的操作符则表示其优先权最低。而同一行上的各操作符其优先权是一样的。

表 2.1　MASM 模式的操作符优先级

<>, (), [], LENGTH, MASK, SIZE, WIDTH
·〔结构或页选择(structure member selertor)〕
HIGH, LOW
+，－(正负号)
:〔段覆盖(segment orer ride)〕
OFFSET, PTR, SEG, THIS, TYPE
*, /, MOD, SHL, SHR
+，－(二元加减运算)
EQ, GE, GT, LE, LT, NE

NOT
AND
OR, XOR
LARGE, SHORT, SMALL, .TYPE

表 2.2 Ideal 模式的操作符优先级

< >, [], LENGTH, MASK, OFFSET, SEG, SIZE, WIDTH, HIGH, LOW
+，－(正负号)
*, /, MOD, SHL, SHR
+，－(二元加减运算)
EQ, GE, GT, LE, LT, NE
NOT
AND
OR, XOR
: (段覆盖)
· (结构成员选择)
HIGH(指针之前)，LARGE，LOW(指针之前)，PTR，SHORT，SMALL

下面就按字母顺序逐个介绍操作符。

()

功能　标记某运算式以便先进行计算
模式　MASM,Ideal
语法　(expression)
说明　使用括号可以改变操作符原来具有的优先级。括号内的任何运算都比它们之前或之后的运算先执行。
参看　+，－,*,/,MOD,SHL,SHR
实例　(3+4)*5;结果为 35
　　3+4*5;　结果为 23

*

功能　将两个整数运算表达式相乘
模式　MASM,Ideal
语法　expression 1 * exprssion2
说明　expression1 和 expreesion2 本身必须都能得出一整常数。
参看　+,-,/,MOD,SHL,SHR
实例　SCREENSIZE=25*80;屏幕的最大字节数

此为试读,需要完整PDF请访问: www.ertongbook.com