A P P E N D I X

Generating and Reading Assembly Listings

A listing file shows precisely how the assembler translates your source file into machine code. The listing documents the assembler's assumptions, memory allocations, and optimizations.

MASM creates an assembly listing of your source file whenever you do one of the following:

- Select the appropriate option in PWB.
- Use one of the related source code directives.
- Specify the /Fl option on the MASM command line.

The assembly listing contains both the statements in the source file and the binary code (if any) generated for each statement. The listing also shows the names and values of all labels, variables, and symbols in your file.

The assembler creates tables for macros, structures, unions, records, segments, groups, and other symbols, and places the tables at the end of the assembly listing. Only the types of symbols encountered in the program are included. For example, if your program has no macros, the symbol table does not have a macros section.

Generating Listing Files

To generate a listing file from within PWB, follow these steps:

1. From the Options menu, choose MASM Options.
2. In the MASM Options dialog box, choose Set Debug or Release Options.

The dialog box for Set Debug or Release Options lists the choices summarized in Table C.1. This table also shows the equivalent source code directives and command-line options.

Table C.1 Options for Generating or Modifying Listing Files

To generate this information:

To generate this information:	**In PWB[1], select:**	**In source code, enter:**	**From command line, enter:**
Default listing—includes all assembled lines	Generate Listing File	**.LIST** (default)	/Fl
Turn off all source listings (overrides all listing directives)	Generate Listing File (turn off)	**.NOLIST** (synonym = **.SFCOND**)	—
List all source lines, including false conditionals and generated code	Include All Source Lines	**.LISTALL**	/Fl /Sa
Show instruction timings	List Instruction Timings	—	/Fl /Sc
Show assembler-generated code	List Generated Instructions	—	/Fl /Sg
Include false conditionals[2]	List False Conditionals	**.LISTIF** (synonym = **.LFCOND**)	/Fl /Sx
Suppress listing of any subsequent conditional blocks whose condition is false	List False Conditionals (turn off)	**.NOLISTIF** (synonym = **.SFCOND**)	—
Toggle between **.LISTIF** and **.NOLISTIF**	—	**.TFCOND**	—
Suppress symbol table generation	Generate Symbol Table (turn off the default)	—	/Fl /Sn
List all processed macro statements	—	**.LISTMACROALL** (synonym = **.LALL**)	—
List only instructions, data, and segment directives in macros	—	**.LISTMACRO** (default) (synonym = **.XALL**)	—
Turn off all listing during macro expansion	—	**.NOLISTMACRO** (synonym = **.SALL**)	—
Specify title for each page (use only once per file)	—	**TITLE** *name*	/St *name*
Specify subtitle for page	—	**SUBTITLE** *name*	/Ss *name*
Designate page length and line width, increment section number, or generate page breaks	—	**PAGE** ⟦*length*,*width*⟧⟦+⟧	/Sp *length* /Sl *width*
Generate first-pass listing	—	—	/Ep

[1] Select MASM Options from the Options menu, then choose Set Dialog Options from the MASM Options dialog box.

2 See “Conditional Directives” in Chapter 1

Precedence of Command-Line Options and Listing Directives

Since command-line options and source code directives can specify opposite behavior for the same listing file option, the assembler interprets the commands according to the following precedence levels. Selecting PWB options is equivalent to specifying /Fl /S*x* on the command line:

- /Sa overrides any source code directives that suppress listing.
- Source code directives override all command-line options except /Sa.
- **.NOLIST** overrides other listing directives such as **.NOLISTIF** and **.LISTMACROALL**.
- The /Sx, /Ss, /Sp, and /Sl options set initial values for their respective features. Directives in the source file can override these command-line options.

Reading the Listing File

The first half of the listing shows macros from the include file DOS.MAC, structure declarations, and data. After the **.DATA** directive, the columns on the left show offsets and initialized byte values within the data segment.

Instructions begin after the **.CODE** directive. The three columns on the left show offsets, instruction timings, and binary code generated by the assembler. The columns on the right list the source statements exactly as they appear in the source file or as expanded by a macro. Various symbols and abbreviations in the middle column provide information about the code, as explained in the following section. The subsequent section, “Symbols and Abbreviations,” explains the meanings of listing symbols.

Generated Code

The assembler lists the code generated from the statements of a source file. With the /Sc command-line switch, which generates instruction timings, each line has this syntax:

offset ⟦*timing*⟧ ⟦*code*⟧

The *offset* is the offset from the beginning of the current code segment. The *timing* shows the number of cycles the processor needs to execute the instruction. The value of *timing* reflects the CPU type; for example, specifying the **.386** directive produces instruction timings for the 80386 processor. If the statement generates code or data, *code* shows the numeric value in hexadecimal notation if the value is known at assembly time. If the value is calculated at run time, the assembler indicates what action is necessary to compute the value.

When assembling under the default **.8086** directive, *timing* includes an effective address value if the instruction accesses memory. The 80186/486 processors do not use effective address values. For more information on effective address timing, see the "Processor" section in the *Reference* book.

Error Messages

If any errors occur during assembly, each error message and error number appears directly below the statement where the error occurred. An example of an error line and message is:

```
mov     ax, [dx][di]
listtst.asm(77): error A2031: must be index or base register
```

Symbols and Abbreviations

The assembler uses the symbols and abbreviations shown in Table C.2 to indicate addresses that need to be resolved by the linker or values that were generated in a special way. The example in this section illustrates many of these symbols.

The example listing was produced using "List Generated Instructions" and "List Instruction Timings" in PWB. These options correspond to the ML command-line switches /Fl /Sg /Sc.

Table C.2 Symbols and Abbreviations in Listings

Character	Meaning
C	Line from include file
=	**EQU** or equal-sign (=) directive
nn[xx]	**DUP** expression: *nn* copies of the value *xx*
----	Segment/group address (linker must resolve)
R	Relocatable address (linker must resolve)
*	Assembler-generated code
E	External address (linker must resolve)
n	Macro-expansion nesting level (+ if more than 9)
|	Operator size override
&	Address size override
nn:	Segment override in statement
nn/	**REP** or **LOCK** prefix instruction

Table C.3 explains the five symbols that may follow timing values in your listing. The *Reference* book will help you determine correct timings for those values marked with a symbol.

Table C.3 Symbols in Timing Column

Symbol	Meaning
m	Add cycles depending on next executed instruction.
n	Add cycles depending on number of iterations or size of data.
p	Different timing value in protected mode.
+	Add cycles depending on operands or combination of the preceding.
,	Separates two values for "jump taken" and "jump not taken."

```
Microsoft (R) Macro Assembler Version 6.10   09/20/00   12:00:00
listtst.asm                                               Page 1 - 1


                                        .MODEL  small, c
                                        .386
                                        .DOSSEG
                                        .STACK  256
                                        INCLUDE dos.mac
                              C StrDef  MACRO   name1, text
                              C name1   BYTE    &text
                              C         BYTE    13d, 10d, '$'
                              C l&name1 EQU     LENGTHOF name1
                              C         ENDM
                              C
                              C Display MACRO   string
                              C         mov     ah, 09h
                              C         mov     dx, OFFSET string
                              C         int     21h
                              C         ENDM
 = 0020                         num     EQU     20h
                                COLOR   RECORD  b:1, r:3=1, i:1=1, f:3=7
 = 35                           value   TEXTEQU %3 + num
 = 32                           tnum    TEXTEQU %num
 = 04                           strpos  TEXTEQU @InStr( , <person>,
<son> )

                                PutStr  PROTO   pMsg:PTR BYTE

 0004                           DATE    STRUCT
 0000  01                       month   BYTE    1
 0001  01                       day     BYTE    1
 0002  0000                     year    WORD    ?
                                DATE    ENDS
 0002                              U1      UNION
 0000  0028                        fsize   WORD    40
                                   bsize   BYTE    60
                                   U1      ENDS

 0000                                      .DATA

 0000 00000000                     ddData  DWORD   ?
 0004 1F                           text    COLOR   <>
 0005 01 14 07C9                   today   DATE    <01, 20, 1993>
 0009 00                           flag    BYTE    0
 000A       001E [                 buffer  WORD    30 DUP (0)
             0000
            ]

                                           StrDef   ending, "Finished."
 0046 46 69 6E 69 73 68      1  ending   BYTE    "Finished."
            65 64 2E
 004F  0D 0A 24              1          BYTE    13d, 10d, '$'
 = 0009                      1  lending EQU     LENGTHOF ending
 0052 54 68 69 73 20 69                 Msg      BYTE    "This is a
string","0"
            73 20 61 20
            73 74 72 69
            6E 67 30

                                   float   TYPEDEF         REAL4
                                   FPBYTE  TYPEDEF FAR     PTR BYTE
 0063 ---- 0052 R                  FPMSG   FPBYTE          Msg
                                   PBYTE   TYPEDEF         PTR BYTE
                                   NPWORD  TYPEDEF NEAR    PTR WORD
                                   PVOID   TYPEDEF         PTR
                                   PPBYTE  TYPEDEF         PTR PBYTE

 0000                                      .CODE
                                           .STARTUP
 0000                         *@Startup:
 0000   2   B8 ---- R         *        mov    ax, DGROUP
 0003   2p  8E D8             *        mov    ds, ax
 0005   2   8C D3             *        mov    bx, ss
 0007   2   2B D8             *        sub    bx, ax
 0009   3   C1 E3 04          *        shl    bx, 004h
 000C   2p  8E D0             *        mov    ss, ax
 000E   2   03 E3             *        add    sp, bx

                                   EXTERNDEF       work:NEAR
 0010   7m  E8 0000 E                      call    work
```

```
                                        INVOKE  PutStr, ADDR msg
 0013   2   68 0052 R      *        push   OFFSET Msg
 0016   7m  E8 0029        *        call   PutStr
 0019   2   83 C4 02       *        add    sp, 00002h

 001C   2   B8 ---- R                   mov     ax, @data
 001F   2p  8E C0                       mov     es, ax
 0021   2   B0 63                       mov     al, 'c'
 0023   4   26: 8B 0E                   mov     cx, es:num
            0020
 0028   2   BF 0052                     mov     di, 82
 002B   7n  F2/ AE                      repne   scasb
 002D   4   66| A1 0000 R               mov     eax, ddData
 0031   6   67& FE 03                   inc     BYTE PTR [ebx]

                                EXTERNDEF       morework:NEAR
 0034   7m  E8 0000 E                   call    morework

                                        Display ending
 0037   2   B4 09            1          mov     ah, 09h
 0039   2   BA 0046 R        1          mov     dx, OFFSET ending
 003C  37   CD 21            1          int     21h

                                       .EXIT
 003E   2   B4 4C          *        mov    ah, 04Ch
 0040  37   CD 21          *        int    021h



 0042                           PutStr  PROC    pMsg:PTR BYTE

 0042   2   55             *        push   bp
 0043   4   8B EC          *        mov    bp, sp
 0045   2   B4 02                       mov     ah, 02H
 0047   4   8B 7E 04                    mov     di, pMsg
 004A   4   8A 15                       mov     dl, [di]
                                        mov     ax, [dx][di]
listtst.asm(77): error A2031: must be index or base register

                                        .WHILE  (dl)
 004C   7m  EB 10          *        jmp    @C0001
 0059                      *@C0002:
 0059  37   CD 21                       int     21h
 005B   2   47                          inc     di
 005C   4   8A 15                       mov     dl, [di]
                                        .ENDW
 005E                      *@C0001:
 005E   2   0A D2          *        or  dl, dl
 0060 7m,3  75 F7          *        jne    @C0002
                                        ret
```

```
0062   4   5D             *        pop    bp
0063  10m  C3             *        ret    00000h
0064                           PutStr  ENDP

                               END
```

Reading Tables in a Listing File

The tables at the end of a listing file list the macros, structures, unions, records, segments, groups, and symbols that appear in a source file. These tables are not printed in the previous sample listing, but are summarized as follows.

Macro Table

Lists all macros in the main file or the include files. Differentiates between macro functions and macro procedures.

Structures and Unions Table

Provides the size in bytes of the structure or union and the offset of each field. The type of each field is also given.

Record Table

"Width" gives the number of bits of the entire record. "Shift" provides the offset in bits from the low-order bit of the record to the low-order bit of the field. "Width" for fields gives the number of bits in the field. "Mask" gives the maximum value of the field, expressed in hexadecimal notation. "Initial" gives the initial value supplied for the field.

Type Table

The "Size" column in this table gives the size of the **TYPEDEF** type in bytes, and the "Attr" column gives the base type for the **TYPEDEF** definition.

Segment and Group Table

"Size" specifies whether the segment is 16 bit or 32 bit. "Length" gives the size of the segment in bytes. "Align" gives the segment alignment (**WORD**, **PARA**, and so on). "Combine" gives the combine type (**PUBLIC**, **STACK**, and so on). "Class" gives the segment's class (**CODE**, **DATA**, **STACK**, or **CONST**).

Procedures, Parameters, and Locals

Gives the types and offsets from BP of all parameters and locals defined in each procedure, as well as the size and memory location of each procedure.

Symbol Table

All symbols (except names for macros, structures, unions, records, and segments) are listed in a symbol table at the end of the listing. The "Name" column lists the names in alphabetical order. The "Type" column lists each symbol's type.

The length of a multiple-element variable, such as an array or string, is the length of a single element, not the length of the entire variable.

If the symbol represents an absolute value defined with an **EQU** or equal sign (=) directive, the "Value" column shows the symbol's value. The value may be another symbol, a string, or a constant numeric value (in hexadecimal), depending on the type. If the symbol represents a variable or label, the "Value" column shows the symbol's hexadecimal offset from the beginning of the segment in which it is defined.

The "Attr" column shows the attributes of the symbol. The attributes include the name of the segment (if any) in which the symbol is defined, the scope of the symbol, and the code length. A symbol's scope is given only if the symbol is defined using the **EXTERN** and **PUBLIC** directives. The scope can be external, global, or communal. The "Attr" column is blank if the symbol has no attribute.