Table of Contents

Introduction

This section is an overview of the 2500 A.D. Z30 Cross Assembler. The intent of this manual is to describe the operation of the Assembler. It is assumed that the user is familiar with the Z80 operation and instruction set.

The 2500 A.D. Z80 Assembler enables the user to write programs which can then be assembled into relocatable object code and linked to the desired execution address using the 2500 A.D. Linker.

The Assembler will process any size file, as long as enough memory is available. All the buffers used by the Assembler are requested and expanded as needed, with the exception of the Source Code Input Buffer, the Object Code Output Buffer and the Listing Buffer.

The Conditional Assembly section enables the user to direct the Assembler to process different sections of the source file depending on the outcome of assembly time operations. Conditionals may be nested to 248 levels, and the Assembler aids the programmer in detecting conditional nesting errors by not only checking for unbalanced conditional levels, but also by displaying the current active conditional level in the object code field of the listing.

The Assembly Time Calculation section will perform calculations with up to 16 pending operands, using 80 bit arithmetic. The algebraic hierarchy may be changed through the use of parenthesis.

The Listing Control section provides for listing all or just sections of the program, with convenient Assembler error detection overrides, along with Assembly Run Time Commands that may be used to dynamically change the listing mode. Also, in this section is a description of the LINKLIST directive, which allows the linker to relocate listings.

The 2500 A.D. Linker allows files to either be linked together or just used for external reference resolution. As with the Assembler, all buffers used by the Linker are requested as needed. The Linker is capable of outputting several different formats. The format may be changed by using an Assembler Directive or selecting the desired output from the Linker option field. Programs may specify up to 256 user defined section names, and the Linker is capable of processing up to 256 identical section names. See the Linker Description section of this manual for a complete description.

Operating Instructions

Prompt Mode

To run the Assembler type : *x80*

The Assembler responds with :

Listing Destination ? (N, T, P, D, E, L, <CR =N>):
with the abbreviations as follows:

N	=None
T	= Terminal
P	= Printer (Single User Systems Only)
D	=Disk
E	= Error Only
L	= List On/Off

After this the Assembler prompts the operator for the source code filename _ as shown below.

Input Filename:

When entering your source filename you may specify an extension or the assembler will look for an extension of asm. Once you have specified your input filename the assembler will prompt you for the output filename.

Output Filename:

If the user responds to the output filename prompt with just a carriage return, the output file will receive the same filename as the input file, with an extension of obj. If the response is a filename with no extension, the output file will be under that filename with an extension of obj.

It the listing is to be under List On/Off Assembler directive control the additional prompt shown below is output:

LIST ON/OFF Listing Destination (T, P, D, <CR> = T) :

The abbreviations are the same as shown above.

The List On/Off control allows the user to list only selected parts of the source file. For more information see the Listing Control section of this manual.

If Error Only is chosen the Assembler will prompt the user for the destination as follows:

Error Only Listing Destination (T, P, D, <CR, = T) :

If the listing is being sent to the printer (available on single user systems only) or the disk, the Assembler will prompt for a Cross Reference Listing.

Note for VMS users:.

Assuming the assembler is located in a directory named $diskt[x80], the following command must be entered for the examples shown above to work:

x80 == "$diski :[x80]x80.exe"

Command Line Mode

The Assembler may also be invoked using a command line. In this case, the input filename is specified first, then the output filename, and then a list of options. Both the ouput filename and the listing destination are optional. The general form of the command, with optional fields shown in brackets, is as follows:

x80 [-q] input_filename [output_filename] [-t, -p, -d, -px, -dx]

The -q option stands for Quiet mode. If this option is selected, the only screen messages output from the Assembler will be error messages and the line on which they occur. This option must be placed before the input filename. Not all options can be used at the same time.

Below are some examples of legal command lines.

Input Filename Only

x80 inout_filename

This command causes the Assembler to process the source file input_filename. If no extension is specified, it is assumed to be .asm. Since no options are specified, they will default to Error Only listing with the terminal as the destination. The output filename will be the same as the input filename but with an extension of .obj.

Input Filename and Output Filename

x80 input_filename output _filename

This command is identical to the previous one except that the Assembler will name the object file output_filename'.

Listing to Terminal

x80 input_filename output_filename -t

This command will assemble the input file input_filename and send the listing to the terminal. The optional output filename specification causes the assembler to generate an object file named output_filename.

Listing to Printer

x80 input_filename -p

This command will assemble the input file input_filename and send the listing to the printer. The output filename will be input_filename with and extension of obi

Listing to Printer with Cross Reference

x80 input_filename output_filename -px

This command will assemble the input file input_filename and send the listing and cross reference table to the printer. The optional output filename specification causes the assembler to generate an object file named output_filename. The printer option is only available on single user systems.

Listing to Disk

x80 input_filename output_filename -d

This command will assemble the input file input_filename and send the listing to the disk. The disk listing file will have the same name as the output file, but will have an extension of 1st. The optional output filename specification causes the assembler to generate an object file named output_filename.

Listing to Another Drive or Directory

MSDOS

```
x80 input_filename output_filename -d,a:
x80 input_filename output_filename -d, \new\
```

UNIX

```
x80 input_filename output_filename -d, /new/
```

VMS

```
x80 input_filename output_filename -d,$diskl :[a]
x80 input_filename output_filename -d, [new]
```

This enables the user to send the listing to a different drive or directory other than the current one.

Error Only Listing to the Terminal

```
x80 input_filename output_filename -et
```

This command will assemble the input file input_filename and send error messages to the terminal. The optional output filename specification causes the assembler to generate an object file named output_filename.

Listing to Disk with Cross Reference

```
x80 input_filename output_filename -dx
```

This command will assemble the input file input_filename and send the listing and cross reference table to the disk. The disk listing file will have the same name as the output file, but will have an extension of 1st. The optional output filename specification causes the assembler to generate an object file named output_filename.

Error Only Listing to the Printer

x80 input_filename output_filename -ep

This command will assemble the input file input_filename and send error messages to the printer. The optional output filename specification causes the assembler to generate an object file named output_filename. The printer option is available only on single user systems.

Error Only Listing to the Disk

x80 input_filename output_filename -ed

This command will assemble the input file input_filename and send error messages to the disk. The disk listing file will have the same name as the output file, but will have an extension of 1st. The optional output filename specification causes the assembler to generate an object file named output_filename.

List On/Off to Terminal

x80 input_filename -lt

This command will assemble the input file input_filename and send LIST ON/OFF blocks to the terminal.

List On/Off to Printer

x80 input_filename -lp

This command will assemble the input file input_filename and send the LIST ON/OFF blocks to the printer. This option is only available on single user systems.

List On/Off to Disk

x80 inputillename -ld

This command will assemble the input file **input_filename** and send the **LIST ON/OFF** blocks to the disk. The disk listing file will have the same name as the output file, but with an extension of **1st.**

Note for **VMS users:**

Assuming the -·-·t3sembler is located in a directory named $diskt[x801, the following command must be entered for the examples shown above to work:

x80 == "Sdiski :[x80]x80.exe"

System Defaults

The following default filename extensions will be used by the 2500 A.D. programs if no

extension is specified by the user.

Assembler

asm	-	Input to the Assembler
obj	-	Output from the Assembler
pak		Packed output from the Assembler
1st		Listing file

Linker

obj	Input to the Linker
lib	Library file
tsk	Executable Object Code
hex	Intel Hex and Extended Intel Hex
tek	Tektronix Hex
s19	Motorola S19
s28	Motorola 528
s37	Motorola 537

Librarian

obj	Input to the Librarian
pak	Packed input to the Librarian
lib	Output from the Librarian

Note that because of the additional information included in the Assembler output file, the Linker must always be run, even if the program is assembled at the desired run address and there are no external references. This is so that all the additional information can be removed and a file with the desired output format can be generated.

Assembler Error Processing

When an assembly error is encountered, the action taken by the Assembler depends on the listing mode it is currently operating under.

If the No List option was specified, the statement causing the error and the error message will be output to the terminal, the display will be turned on and the Assembler will halt just as if the user had typed As. The reason for this is to give the user a chance to see exactly where the error is. This will occur on pass 1 as well as pass 2. Note that some errors are not detectable on pass 1, such as undefined symbols. After the error has been displayed, the output can be turned off using AN.

If the listing is being sent to the printer or the disk, then errors encountered on pass 1 are sent to the terminal but not the printer or disk, and the Assembler does not halt. On pass 2, the error is output to the printer or disk as well as the terminal and the assembly continues.

If the listing is being sent to the printer or disk under assembler djrective control, any errors encountered during pass 1 are output to the terminal but not the printer or disk, and the assembly continues. Errors detected during pass 2 are output to the printer or disk and the terminal, even if the error is not inside a block that was specified to be listed.

Assembler Run Time Commands

The following commands are active during the assembly process. These commands are active during pass 1 as well as pass 2, and override the listing mode specified when the Assembler was first activated.

Unix Assembler Run Time Commands

Msdos Assembler Run Time Commands

VMS Assembler Run Time Commands

Ctrl S - Ctrl 0 - Del C - Del T - Del D - Del M - Del N -
Stop terminal output
Start terminal output
Terminate the assembly
Display the output at the terminal
Send the output to the disk
Multiple output (Terminal & Disk)
No output
Ctrl S Ctri CI Esc C Esc T Esc P Esc D Esc M Esc N
Stop terminal output
Start terminal output
Terminate the assembly
Display the output at the terminal Display the output at the printer Send the output to the disk Multiple output (Terminal & Disk) No output
Ctr
Ctr
Ctr C, C Ctr C, T Ctr C, D Ctr C, M Ctr C, N
Stop terminal output
Start terminal output
Terminate the assembly
Display the output at the terminal
Send the output to the disk
Multir'e output (Terminal & Disk)
No output

Assembly Language Syntax **Number Base Designations**

Assembly Language Syntax

This section describes the syntax used by the 2500 A.D. Cross Assembler.

Number Base Designations

Number bases are specified by the following:

Binary	_	B
Octal		0 or 0
Decimal	-	D or no base designation
Hex	-	H or a preceding % sign
Ascii	-	Single or double quotes - "X" or 'X'

The two character sequences between single or double quotes shown below are predefined. However, the **TVVOCHAR ON** directive must be used to enable these.

"CR" or	'CR'	-	Carriage return
"LF" or	'LF'	-	Line feed
"SP" or	'SP'	-	Space
"Hr or	'HT'	-	Horizontal tab
"NL" or	'NL'	-	Null

Program Comments

Comment lines must start with a semi-colon or asterisk in column 1, unless the COMMENT directive is used. Comments after an instruction do not need a semi-colon if at least 1 space or tab precedes the start of the comment if the assembler is running in Spaces Off mode. If the assembler is running in Spaces On mode, all comments after an instruction must be preceded by a semi-colon. See the **SPACES** directive for more information and for the default mode.

Program Counter

The special character dollar sign ($) may be used in an expression to specify the program counter. The value assigned to the dollar sign is the program counter value at the start of the instruction.

Labels

Non-Local labels may be any number of characters long, but only 32 characters are

significant. Labels may start in any column if the name is terminated by a colon. If no colon is used, the label must start in column 1. All labels must start with an alpha character. Upper and lower case characters are considered to be different.

Local Labels

A Local Label is a label which can be used like any "non local" label. The difference is that the definition of a Local Label is only valid between "non local" labels. The adjective "local" refers to the area between labels which retain their definition through the entire program. When a program passes from one local area to the next, local label names can be reused. This feature is useful for labels referenced only within a "local area", as defined above, and original label names are not necessary.

The assembler identifies a local label by the ($) prefix or suffix. This identifier can be changed with the LLCHAR directive. Please see the section entitled 'Directive Definition Control' for more information on this directive. Following are some examples of the use of Local Labels.

In this example, there are three "non-local" labels, LABEL1, LABEL2, and LABEL3. Local Labels, $1 and $2, or 1$ and 2$, have different definitions when referenced in different local ar973. Note that $1 is not considered to be the same as 1$. Any character may be used in a Local Label. Local Labels may be up to 32 characters long. Operators such as '+' should never be used in Local Labels. Local labels will not be terminated if the directives VAR, DEFL, SECTION, ENDS and $

are used. •

High Byte

To load the high byte of a 16 bit value the unary greater than sign,>, should be used. This allows bits 8 through 15 to be used as a byte value which is relocatable.

Low Byte

To load the low byte of a 16 bit value the unary less than sign,<, should be used. This allows bits 0 through 7 to be used as a byte value which is relocatable.

Upper / Lower Case

Upper and lower case labels are recognized as different labels. The labels used for section names and macro names are also different if the label is in lower case rather than upper case.

```
      LABEL?:
            $1:      NOP
            $2:      JMP $1 JMP $2
LABEL1:
1$:     NOP
2$: JMP 1$
                                 JMP 2$

   LABEL2:                              LABEL2:
   $1:     NOP     1$ NOP
   $2:     JMP $1  2$ JMP 1$
           JMP $2                               JMP 2$

   LABELS:                              LABELS:
   $1:     NOP     1$ NOP
   $2:     JMP $1  2$ JMP 1$
           JMP $2                               JMP 2$
```


Addressing Modes

Immediate

The data is contained in the instruction.

Examples:

```
LD      HL,1234H        ; Ld HL with the HEX number 1234.
LD      HL,DATA         ; Ld HL with the value associated with the
                        Label 'DATA'.
```

Register

The data is contained in a CPU register.

Examples:

```
LD      A,B             ; Ld the contents in register B into register A
SUB     D               ; Subtract the contents of register D from
                        register A.
```

Register Indirect

The operand address is pointed to by a register.

Examples:

```
LD      A,(HL)          ; Ld A with the contents of the location
                        pointed to by HL.
LD      (HL),B          ; Store the contents of B in the memory
                        address pointed to by HL
```


Direct Addressing

The address of the operand is contained in the instruction.

Examples:

```
LD      HL,(1234H)      ; Ld HL with the contents of memory location
                        1234 HEX
LD      (ADDRESS),HL    ; Store HL in the memory location 'ADDRESS'
LD      (ABCDH),HL      ; Store HL in the memory location ABCD
                        hexadecirn9:
```

Indexed

The operand address is the sum of the 8 bit offset in the instruction and the contents of either IX or IY.

Examples:

```
LD      A,(IX+4)        ; Ld A with contents of the memory location
                        pointed to by adding 4 to the contents of register
                        IX
LD      (IX+DATA8),B    ; Store B in the memory location obtained
                        by adding the value associated with 'DATA8'
                        to the contents of register IX
```

Relative

The operand address is relative to the current instruction. If the address is given using a numerical value, the calculation is from the start of the next instruction.

Examples:

```
DJNZ    LOOP            ; The Assembler calculates the address
                        by subtracting the address of the label 'LOOP'
                        from the address of the next instruction
JR      4               ; The destination is 4 BYTES past the start
                        of the next instruction
```


Assembler Directives

This section describes the Assembler Directives. Directives may be preceded by a decimal point if desired to help differentiate them from program instructions.

Storage Control

ORG
ORIGIN

Sets the program assembly address. If this directive is not executed, the assembly address defaults to 0000.

END VALUE

This directive defines the end of a program or an included file. The expression following an END statement is optional and if it exists, specifies the program starting address. This address is encoded in the output file if a program starting address record type exists in the output format definition.

LABEL: ASCII STRING

Stores STRING in memory up to but not including either a carriage return or a broken bar character ("|", Hex 70). A label is optional. Following are some examples of ASCII.

```
ASCII   Hello   ; Stores the Ascii representation of
                Hello in consecutive memory locations.
                Incidentally, this comment would be
ASCII   Hello   stored also.
                ; Now the comment wouldn't be stored.
                The next example shows termination with
ASCII   Hello   just a carriage return.
```


LABEL: **DB** VALUE
FCB
DEFB
BYTE
STRING

The Assembler will store the value of the expression in consecutive memory locations. The **BYTE** expression may be any mixture of operand types with each one separated by a comma. Ascii character strings must be bracketed by apostrophes. If the string contains an apostrophe, this can be specified with two apostrophes in a row. If no expression is given, one byte is reserved and zeroed. A label is optional. Following are some examples of the use of the **BYTE** directive.

```
.BYTE       ; Reserves 1 zeroed byte.
.BYTE   10  ; Reserves 1 byte = 10 decimal.
.BYTE   1,2,3  ; Reserves 3 bytes,= to 1,2 & 3 in that order.
        .BYTESYMBOL-10 ; Searches the symbol table
        for SYMBOL, subtracts 10 decimal from it's
        value, and stores the result.
```

.BYTE'Hello' ; Stores the Ascii equivalent of the string Hello in consecutive memory locations.

.BYTE'Hello', ODH ; Same as above example, with the addition of a carriage return at the end. Spaces are ignored before operands, but the comma is required.

.BYTE '2500 A.D."s' ; Embedded apostrophe.

LABEL: DW VALUE
FDB
DEFW
LWORD

This directive will store the va:ue of the expression in a 16 bit storage location. Mutple words may be initialized by separating each expression with a comma. If no expression is given, 1 word is reserved and zeroed. A label is optional.

LABEL: LONG VALUE
LONGW
LWORD

This directive will store the value of the expression in a 32 bit storage location. Multiple long words may be initialized by separating each expression with a comma. If no expression is given, 1 long word is reserved and zeroed. A label is optional.

LABEL: FCC STRING

Stores STRING in memory until a character is reached that matches the first character. The first character and the second matching character are not stored. A label is optional. Typical usage is as follows:

```
FCC     /This is a test string/
        DC      "String"
```

This directive sets the high bit on the last character of a string.

```
        DS      SIZE,VALUE
        RMB
        DEFS
```

This directive will reserve the number of bytes specified by SIZE. No value is stored in the reserved area. This directive differs from the BLKB directive in that if the storage locations are at the end of a program section, the output from the Linker is executable, and the Linker is not required to stack another module on top of this section, the reserved bytes are not included in the output file.


```
LABEL:      FLOAT       VALUE
```

Converts the value specified into single precision floating point format. The value is not rounded but is truncated if the mantissa is larger than 24 bits. The directive does not allow scientific notation.

```
FLOAT   178.125
FLOAT   100,.125,-178.125
```

LABEL: DOUBLE VALUE

Converts the value specified into double precision floating point format. The value is not rounded but is truncated if the mantissa is larger than 52 bits. The directive does not allow scientific notation.

```
DOUBLE 178.125
DOUBLE 100,.125,-178.125
```

LABEL: BLKB SIZE, VALUE

Reserves the number of bytes specified by SIZE. If the value field is present, that value is stored in each byte. Otherwise, the reserved bytes are zeroed. A label is optional.


```
BLKB    20      ;Reserves 20 zeroed bytes
BLKB    20,0    ;Reserves 20 zeroed bytes
```

```
BLKB    20,FFH      ;Reserves 20 bytes and stores FF Hex in each one
```

LABEL: BLKW SIZE, VALUE

Reserves the number of 16 bit words specified by SIZE. If the value field is present, that value is stored in each word. Otherwise, the reserved words are zeroed. A label is optional.

```
BLKW    20          ;Reserves 20 zeroed words
BLKW    20,0        ;Reserves 20 zeroed words
BLKW    20,FFFFH    ;Reserves 20 words and stores FFFF Hex in each one
```

LABEL: BLKL SIZE, **VALUE**

Reserves the number of 32 bit long words specified by SIZE. If the value field is present, that value is stored in each long word. Otherwise, the reserved long words are zeroed. A label is optional.

```
BLKL    20          ;Reserves 20 zeroed long words
BLKL    20,0        ;Reserves 20 zeroed !long words
BLKL    20,FFFFH    ;Reserves 20 long words and stores FFFF Hex in each
                    one
```


Definition Control

LABEL: EQU VALUE
EQUAL

Equates LABEL to VALUE. VALUE may be another symbol or any legal arithmetic expression.

LABEL: VAR VALUE
DEFL

Equates LABEL to VALUE, but may be changed as often as desired throughout the program. A label defined as a variable should not be redefined by an EQUAL directive.

LLCHAR CHARACTER

The default character for designating a Local Label is the ($). This directive changes the character which identifies a particular symbol as a Local Label. Symbols that designate number bases should be avoided, unless they are used on the trailing end of the label.

LABEL: MACRO ARGS

Specifies the start of a Macro Definition.

ENDM
MACEND

Specifies the end of a Macro Definition.

MACEXIT

This directive causes the immediate exit from a macro. The difference between MACEXIT and MACEND is that during the macro definition process, MACEXIT does not terminate the macro, and if MACEXIT is in the path of a false conditional assembly block, it is not executed. All conditional assembly values are restored to the same state as when the macro was invoked.

MACDELIM CHARACTER

This directive is used to pass an argument containing a comma into a macro. The default mode is for commas to always be argument separators. The allowed characters are '{', '(' and '[. All characters between matching delimiter pairs will be passed through

as one argument. Please refer to the Macro Examples section of this manual for some examples of the use of this directive.

XDEF LABEL
GLOBAL
PUBLIC

Specifies the label as a global label that may be referenced by other programs. Multiple labels may be specified as long as each one is separated by a comma. Below are some examples of the correct use of GLOBAL.

```
GLOBAL SYM1      ; Declares the label SYM1 to be accessible to other pro-
                   grams. The Linker will resolve external references.

                 GLOBAL SYM1,SYM2 ; Multiple declarations on
                 the same line are legal separated by a comma. The
                 spaces are ignored.
```


GLOBALS ON

This directive causes the Assembler to treat all labels after GLOBALS ON as global labels which may be referenced by other programs. This directive will not affect Local Labels. Below is an example of the use of GLOBALS.

```
GLOBALS     ON
SYM1        NOP ;Declares the labels SYM1 and SYM2,
                SYM2    NOP ;accessible to other programs. This
                directive is not ;reset by the module and endmod
                directives. The ;default is GLOBAL OFF.
```

GLOBALS OFF

This directive returns the Assembler to the default mode which requires Global symbols to be specified with GLOBAL directives.

XREF LABEL
EXTERN
EXTERNAL

Specifies the label as being defined in another program. Multiple labels may be specified as long as each one is separated by a comma.

LABEL: **ASK** PROMPT

Outputs 'PROMPT to the terminal and waits for a 1 character response, from which 30 hex is subtracted. The purpose of this is usually to introduce a 0/1 flag into the program. 'LABEL' is set equal to the result. A carriage return terminates 'PROMPT'. On pass 2, the line is output along with the response.

The following is an example of 'ASK':

```
DISK SIZE: ASK ASSEMBLE FOR 8" (=1) OR 51/4" (=0) DRIVES ? :
```


Assembly Mode

LABEL: SECTION

This directive allows user defined section names to be generated. The Assembler has 2 predefined sections, **CODE and DATA.** The total number of section names allowed per file is 256. Each name may be up to 32 characters long. Lower and upper case are considered to be different. After the section has been defined, the program may switch back and forth simply by using the name as a mnemonic. The default section is **CODE.** Sections may be nested. As with all directives, a section name may be preceded by a decimal point. See the Linker Operating Instructions section of this manual for information on how the Linker handles section names. Below are some examples of defining section names and switching between different sections.

```
                        NOP     ;This instruction goes into the CODE
                                section ;by default

                .DATA           ;Switch to the predefined DATA section
                .BYTE           ;This byte goes into the DATA section

SECTION1:       .SECTION        ;Define a new section. The definition makes
                                ;this section active automatically

                        NOP     ;This instruction goes into the
                                SECTION1 ;section
```

```
.CODE           ;Switch back to the section named CODE
NOP             ;This instruction goes into the CODE section
.SECTION1       ;Switch to the user defined section SECTION1
                NOP ;This instruction goes into the
SECTION1 ;section
.BYTE           ;Any section may contain code or data or both.
```

ENDS

This directive is used in conjunction with the **SECTION** directive. **ENDS** enables the termination of nested sections in a file.

ABSOLUTE

This directive enables the assembler to use page 0 addresses when possible. This directive is supported for compatibility with our series 3.0 assemblers. For a more detailed discussion, refer to the **"Absolute versus Relative"** section of this manual. Executable instructions should always be assembled in Relative mode.

RELATIVE

This directive enables the assembler to return from Absolute mode to Relative mode. Executable instructions should always be assembled in Relative mode. This is the default mode.

RADIX VALUE

2 **or** B =Binary

8	or	0	or	0	=Octal
10	or	D			= Decimal
16	or	H			= Hexadecimal

No expression = return to default mode which is base 10, and assume all others will be designated with B, Q, D or H after the constant. Note that when base 16 is specified there is no way to define a decimal or binary number, since both D and B are legal hexadecimal numbers.

INCLUDE filename

Directs the Assembler to include the named file in the assembly. Filenames may include pathnames. Filename extensions must be completely specified. Includes may not be nested.

SPACES ON

This directive enables spaces in between operands. When spaces are enabled, comments must begin with a semi-colon. The default mode is spaces off.

SPACES OFF

This directive disables spaces in between operands. When spaces are disabled, comments do not need to start with a semi-colon. This is the default mode.

TWOCHAR ON

This directive enables the ascii two character abbreviations shown below. The default mode is **TWOCHAR OFF.**

"CR" or 'CR' - Carriage return
"LF" or 'LP - Line feed
"SP" or 'SP' - Space
"HT" or 'HT' - Horizontal tab
"NL" or 'NL' - Null

TWOCHAR OFF

This directive disables the ascii two character abbreviations shown in the previous directive. This is the default mode.

MODULE

This directive is meant to be used in conjunction with the **ENDMOD** directive and the Library Manager. Normally, libraries are composed of many small routines. When the Linker cannot find a Global Symbol in any of the files that are involved in the link, it can search the libraries for the symbols it cannot find. This means that each routine must be in a separate file and each file must be assembled separately.

Instead of having separate files, each routine can be bracketed with the **MODULE** and **ENDMOD** directive, which allows all the routines to be in one file. This essentially causes the Assembler to treat each module as a totally separate assembly, so references to External symbols must be declared External, and symbols used by other modules must be declared Global. All modules must be terminated with an **ENDMOD.** Modules may not be nested. Modules may have include files within them, but they may not be inside an include file. There is no limit on the number of modules that may be in a file. The Assembler will produce an output file with an extension of **pak.** This file can only be processed by the Librarian, but is simple to manipulate with the Librarian commands **ADD ALL** and **REPLACE ALL.** Please see the section entitled **Librarian Commands** for information on these commands. Following is an example of the use of **MODULE** and **ENDMOD.**

If the above file was named **test.asm,** it would be assembled as usual but the output filename would be **test.pak.** Note during the assembly how the Assembler **restarts** at the beginning of each module.

```
 .MODULE      JUMP_TABLE
 .GLOBAL      JUMP_TABLE

 .EXTERN      ROUTINE1
 .EXTERN      ROUTINE2
;Define library name
;Make table available to other
;modules in file
;Define externals

 .WORD        ROUTINE1
 .WORD        ROUTINE2
 .ENDMOD
 .MODULE      ROUTINE1
 .GLOBAL      ROUTINE1

 NOP
 .ENDMOD
 .MODULE      ROUTINE2
 .GLOBAL      ROUTINE2

 NOP
 .ENDMOD
 .END
```

```
;Store Routine Addresses

;Define end of module ;Define library name ;Make routine available

;Define end of module ;Define library name ;Make routine available

;Define end of module ;Define end of file
JUMP_TABLE:


ROUTINE1:


ROUTINE2:
```


SPACES ON

This directive enables spaces in between operands. When spaces are enabled, comments must begin with a semi-colon. The default mode is spaces off.

SPACES OFF

This directive disables spaces in between operands. When spaces are disabled, comments do not need to start with a semi-colon. This is the default mode.

TWOCHAR ON

This directive enables the ascii two character abbreviations shown below. The default mode is TWOCHAR OFF.

"CR" or 'SR' -	Carriage return
"LF" or 'LF' -	Line feed
"SP" or 'SP' -	Space
"HT" or 'HT' -	Horizontal tab
"NL" or 'NL' -	Null

TWOCHAR OFF

This directive disables the ascii two character abbreviations shown in the previous directive. This is the default mode.

MODULE

This directive is meant to be used in conjunction with the ENDMOD directive and the Library Manager. Normally, libraries are composed of many small routines. When the Linker cannot find a Global Symbol in any of the files that are involved in the link, it can search the libraries for the symbols it cannot find. This means that each routine must be in a separate file and each file must be assembled separately.

Instead of having separate files, each routine can be bracketed with the **MODULE** and **ENDMOD** directive, which allows all the routines to be in one file. This essentially causes the Assembler to treat each module as a totally separate assembly, so references to External symbols must be declared External, and symbols used by other modules must be declared Global. All modules must be terminated with an **ENDMOD.** Modules may not be nested. Modules may have include files within them, but they may not be inside an include file. There is no limit on the number of modules that may be in a file. The Assembler will produce an output file with an extension of **pak.** This file can only be processed by the Librarian, but is simple to manipulate with the Librarian commands **ADD ALL** and **REPLACE ALL.** Please see the section entitled **Librarian Commands** for information on these commands. Following is an example of the use of **MODULE** and **END MOD.**

If the above file was named **test.asm,** it would be assembled as usual but the output filename would be **test.pak.** Note during the assembly how the Assembler **restarts** at the beginning of each module.

```
.MODULE     JUMP TABLE
.GLOBAL     JUMP_TABLE

.EXTERN     ROUTINE1
```

```
.EXTERN        ROUTINE2
;Define library name
;Make table available to other
;modules in file
;Define externals

JUMP_TABLE: .WORD ROUTINE1
                    .WORD          ROUTINE2
                    .ENDMOD
                    .MODULE        ROUTINE1
                    .GLOBAL        ROUTINE1

ROUTINE1:           NOP
                    .ENDMOD
                    .MODULE       ROUT1NE2
                    .GLOBAL        ROUTINE2

ROUTINE2:           NOP
                    .ENDMOD
                    .END
;Store Routine Addresses

;Define end of module ;Define library name ;Make routine available


;Define end of module ;Define library name ;Make routine available


;Define end of module ;Define end of file
```


ENDMOD

This directive is used in conjunction with the **MODULE** directive and terminates each module in a file. Please refer the **MODULE** directive for examples of the use of **ENDMOD.**

COMMENT CHARACTER

This directive allows the user to write blocks of comments at time. A comment block is executed as follows:

```
COMMENT        X
```

Where **X** can be any character. The Assembler will treat everything from the first **X** to the second **X** as a comment block. Since the terminating character is not scanned for until the next line the comment field must be two lines long.

BIT7 ON

This directive will causes the Assembler to set the high bit of each character in an Ascii String. This applies to the **ASCII** directive and the **BYTE** directive only, and it only applies to the BYTE directive when the characters are enclosed in single or double quotes. In otherwords, data values will not be affetected. The Assembler defaults to **BIT7 OFF.**

BIT7 OFF

This directive returns the Assembler to it's default mode, which is to leave bit 7 cleared on Ascii characters.

Conditional Assembly

IFZ VALUE
IFE

The Assembler will assemble the statements following the directive up to an ELSE or ENDIF directive if the VALUE is equal to zero. Conditional statements may be nested up to 248 levels. VALUE can be an arithmetic expression, another symbol or a string.

IF VALUE
IFN
IFNZ
COND

Assemble the statements following the directive up to an ELSE or ENDIF directive if the value of VALUE is not equal to zero. Conditional statements may be nested up to 248 levels.

1FTRUE VALUE
IFN FALSE

This direct've is actually the same as IFNZ, but is more logical when using assembly time comparisons. If the specified condition is true, then the following statements are assembled up to an ELSE or ENDIF directive. If the condition is not true, the statements up to an ELSE or ENDIF directive are not assembled.

IFNTRUE VALUE
IFFALSE

This directive is the same as IFZ, and is the complement to IFTRUE. If the specified condition is false, then the following statements are assembled up to an ELSE or ENDIF directive. If the condition is true, then the statements up to an ELSE or ENDIF directive are not assembled.

IFDEF LABEL

This directive will activate a symbol table search, and if **LABEL** is found, then the statements following this one up to an ELSE or ENDIF directive will be assembled. If LABEL is not found, then the statements following this statement up to an ELSE or ENDIF directive will not be assembled.

IFNDEF LABEL

This directive is the complement of IFDEF. The symbol table is searched and if LABEL is not found, the statements following this one up to an ELSE or ENDIF directive are

assembled. If LABEL is found, then the statements following this one up to an ELSE or ENDIF directive are not assembled.

IFSAME STRING1,STRING2
IFNDIFF

This directive compares STRING1 to STRING2, and conditionally assembles the statements following this statement depending on the result of the comparison. If the two strings are identical then the statements up to an ELSE or ENDIF directive are assembled. If the strings are not identical, then the statements up to an ELSE or ENDIF directive are not assembled. The strings may be one of two different types, namely with spaces or without spaces. However, both strings being compared must be of the same type. If the strings contain spaces, then the beginning and end of each string must be denoted with an apostrophe, with embedded apostrophes denoted by the use of two apostrophes. If the strings do not contain spaces, then the apostrophes are not required. This mode is very useful when comparing macro parameter arguments. In both cases, the strings must be separated with a comma. Following are some examples of the use of **IFSAME.**


```
IFSAME          'test string','test string'
IFSAME          '2500 A.D."s','2500 A.D."s'
IFSAME          X,Y
```

In the first example above, the strings contain spaces and therefore must be bracketed by apostrophes. The second example shows embedded apostrophes, which are represented by using two apostrophes. In the third example, a macro might be testing for a certain register, and since the strings do not contain spaces, they do not need to be enclosed in apostrophes.

IFNSAME STRING1,STRING2 **IFDIFF**

This directive is the complement to IFSAME. If the two strings are not identical, the statements after this statement are assembled up to an ELSE or ENDIF directive. If the two strings are identical the statements up to an ELSE or ENDIF directive are not assembled. The syntax rules governing the form of the strings are the same as for IFSAME. See IFSAME for examples of the use of this directive.

IFEXT LABEL

This directive will cause the Assembler to search the symbol table for the label, and assemble the statements fouowing this statement up to an ELSE or ENDIF if the label has been declared external. An error message is generated if the label is not found.

IFNEXT LABEL

This directive will cause the Assembler to search the symbol table for the label, and assemble the statements following this statement up to an ELSE or ENDIF if the label has not been declared external. An error message is generated if the label is not found.

IFABS LABEL
IFNREL

This directive will cause the Assembler to search the symbol table for the label, and assemble the statements following this statement up to an **ELSE** or ENDIF if the label is absolute (i.e. not relocatable). External labels are considered to be relocatable. An error message is generated if the label is not found.

IFREL LABEL
IFNABS

This directive will cause the Assembler to search the symbol table for the label, and assemble the statements following this statement up to an ELSE or ENDIF if the label is relocatable. External labels are considered to be relocatable. An error message is output if the label is not found.

IFMA EXP

This directive is intended to be used inside a macro, and will scan the macro call line for the existence of the argument number specified by the value of EXP. If the argument exists, the statements following this one up to an ELSE or ENDIF will be assembled. If the argument does not exist, the statements fol lowing this one up to an ELSE or ENDIF will not be assembled. No arguments can be detected by having **EXP =** 0. In this case, if no arguments are present in the macro call line, the following statements are assembled, and if arguments are present in the macro call line, the following statements are not assembled. See the Macro section of this manual for examples of the use of this directive.

IFNMA EXP

This directive is the complement to **IFMA,** and checks the macro call line to see if the argument number given by the value of EXP exists. If the argument is not present, the statements following this one up to an ELSE or ENDIF are assembled. If the argument is present, the statements following this up to an ELSE or ENDIF are not assembled. The existence of any arguments at all can be detected by

having EXP = 0. In this case, if there is at least one argument in the macro call line, the following statements will be assembled. If there are no arguments in the macro call line, the following statements will not be assembled. See the Macro section of this manual for examples of the use of this directive.

ELSE

Start of statements to be assembled if any of the above IF type of directives are false.

ENDC
ENDIF

Specifies the end of a conditional assembly block. When the Assembler detects unmatched IF - ENDIF pairs, an error message is output. Since recursive macros will almost always be controlled by IF type directives, the IFCLEAR directive may be needed. The difference between the two is that ENDIF is always executed, while IFCLEAR is not executed when it is inside a false conditional assembly block.

IFCLEAR

This directive performs exactly the same function as ENDIF, except that it is not executed when it is inside a false conditional assembly block. This directive can be usd in a recursive macro to maintain balanced IF - ENDIF pairs, allowing the macro to eventually terminate, yet still taking advantage of the IF - ENDIF checking performed by the Assembler. This directive can be used to perform the same function when a macro contains a MACEXIT directive for early macro exits, since these would almost alwayt PLe controlled by an IF directive of some sort. See the Macro section of this manual for examples of the use of this directive.

EXIT "MESSAGE"

This directive is meant to be used inside of a conditional and will terminate the assembly if it is executed. MESSAGE is output by the assembler as an error message. If the surrounding condition is true, then the EXIT directive is executed, the user

defined error message is output, and the assembly is terminated. If the surrounding conditional is false, then the assembly coutinues without interruption. The maximum length of the user defined error message is 79 characters. An example of EXIT is as follows:

```
IFTRUE TABLE_SIZE .UGT. MAX_TABLE_SIZE
EXIT
END IF
```

NOTE: If the assembly is terminated, it will occur on the first•pass and no listing file will be created.

Assembly Listing Control

LIST ON
LIST

Turns listing on if **LIST ON/OFF** was specified as the listing destination when the Assembler was first entered. This directive must always be used before **LIST OFF.** In other words, at the start of the program, **LIST OFF** is assumed.

LIST OFF
NOLIST
NLIST

Turns listing off if **LIST ON/OFF** was -specified and **LIST ON** was executed. This is the default mode and therefore should only be used following a **LIST ON** directive.

MACLIST ON
IVILIST

Turns listing of MACRO expansions on. This is the default mode.

MACL1ST OFF
MNLIST

Turns listing of MACRO expansions off. The default is on.

CONDLIST ON

Turns on listing of false conditional assembly blocks. This is the default mode.

CONDL1ST OFF

Turns off listing of false conditional assembly blocks. The default is on.

ASCLIST ON

Turns on the listing of ascii strings that require more than 1 line of object code on the assembler listing.

ASCLIST OFF

Turns off the listing of ascii strings that require more than 1 line of object code on the assembler listing. Only the first line of the object code will be listed.

PW EXP

Sets the printer page width. The default page width is 132 columns.

PL EXP

Sets the printer page length. The default page length is 61 lines. The Assembler issues a form feed when this limit is reached or exceeded. If an error is encountered, the Assembler will output the form feed after the error message.

TOP EXP

This directive controls the number of lines from the top of the page to the page number. The default is zero.

PASS1 ON

Turns on the listing of pass 1. This can be used to help find errors due to the Assembler taking a different path on Pass 1 as compared to Pass 2. This condition will usually generate a 'Symbol value changed between passes' error. This directive can also be useful for finding nested conditional assembly errors.

PASS1 OFF

Turns off listing of pass 1 assuming **PASS1 ON** was executed.

PAG
PAGE
EJECT

Outputs a form feed to the listing device.

NAM STRING
TTL
TITLE
HEADING

Causes **STRING** to be printed at the top of every page. If **STRING** is not specified the **TITLE** directive will be turned off. The title may be changed as often as desired and may be turned off at any time. The maximum title length is 80 characters. Also, the first two tabs between the **TITLE** directive and the start of the string, if they exist, will be ignored. All spaces and tabs after this will be included in the title.

STTL STRING
SUBTITLE
SUBHL

Causes **STRING** to be printed at the top of every page. If **TITLE** was executed, the subtitle will appear below it. If **TITLE** was not executed or was turned off, the subtitle will still be output. If **STRING** is not specified, the directive will be turned off. The subtitle may be changed as often as desired and may be turned off at any time. The maximum subtitle length is 80 characters. As with the **TITLE** directive, the first two tabs between the **SUBTITLE** directive and the start of **STRING,** if they exist, will be ignored and any spaces and tabs that appear after that will be included in the subtitle.

Linker Control

FILLCHAR VALUE

The linker will fill in gaps which are created by the use of sections or origins with the value specified. This directive is only applicable to the executable output from the Linker. All other output formats will begin a new recored if an origin gap is detected.

RECSIZE VALUE

The record length may be changed for Intel Hex and Motorola S record outputs with this directive. By specifying a value standard 32 data bytes for Intel and 131 data bytes for Motorola will be replaced with VALUE.

SYMBOLS

This enables the symbols to be sent to an output file for the linker. This directive must be used to enable the Linker to output the Microtek symbol table format.

OPTIONS OPTION LIST

This directive is used to select the options for the Linker. For a list of the options see the Linker Options section of this manual. The default output filetype is Intel Hex. The output from the Linker may still be changed by using the Linker options field.

LINKLIST

This directive will cause the linker to relocate the assembler listings so that the execution address, the addresses in the object code field and the values in the cross reference table are the actual vlaues at run-time. This directive works with the listing to disk option only.

COMREC "String"

This directive allows the user to insert a comment record in the Motorola outputs. The format of COMREC is as follows.

COMREC "STRING"

Assembly Time Calculations

The following list gives the allowed assembly time calculations. Also shown is their priority level. Priority level 7 operations are the first to be performed. Parenthesis may be used to force the calculations to proceed in a different order. Calculations are performed using 80 bit integer arithmetic with the exception of exponentiation which only uses an 8 bit exponent. The maximum number of pending operations is 16.

OPERATION	PRIORITY	DESCRIPTION
Unary +	7	Optionally specifies a positive operand.
Unary -	7	Negates the following expression.
\ or .NOT.	7	Complements the following expression.
Unary >	7	Keeps the high order byte of the following address. This must be used to obtain relocatable byte address values.
Unary<	7	Keeps the low order byte of the following address. This must be used to obtain relocatable byte address values.

**	6	Unsigned exponentiation
*	5	Unsigned multiplication
/	5	Unsigned division
.MOD.	5	Remainder
.SHR.	5	Shift the preceding expression right (with 0 fill) the number of times specified in the following expression.
.SHL.	5	Shift the preceding expression left (with 0 fill) the number of times specified in the following expression.
+	4	Addition
-	4	Subtraction
& or .AND.	3	Logical AND
^ or .OR.	2	Logical OR
.X0R.	2	Logical exclusive OR

Assembly Time Comparisons

The following list gives the assembly time comparisons which will return all if the comparison is true and all O's if the comparison is false:

=	or	.EQ.	-	Equal
>	or	.GT.	-	Greater than
<	or	.LT.	-	Less than
		.UGT.	-	Unsigned greater than
		.ULT.	-	Unsigned less than

Absolute Versus Relative

The absolute directive enables the assembler to use page 0 addresses when possible and should be used when a symbol is required to have an absolute value. This directive is supported for compatibility with our series 3.0 assemblers. If the Absolute directive is used, the Relative directive must be used to return the assembler to relocatable mode. The assembler should always be returned to Relative mode before any executable instructions are assembled. The Absolute & Relative attributes do not change when the section is changed.

Another valid use of this directive is in laying out assembly language structures. This can be done in a user defined section as in the following example:

```
STRUCTURE_SECTION:      .SECTION
                        .ABSOLUTE
                        .ORIGIN          <initial offset>
NAME:                   .DS              <expression>
COMPANY:                .DS              <expression>
ADDRESS:                .DS              <expression>
CITY:                   .DS              <expression>
STATE:                  .DS              <expression>
ZIP_CODE:               .DS              <expression>
STRUCTURE_SIZE:         .DS                    0
```

where the Origin statement may be omitted if the "initial offset" for the structure is zero and "expression" is equal to the size of the corresponding member of the structure. Note that in this example, storage space for three different structures of this type could be reserved by the following code:

```
STRA:                   .DS              STRUCTURE_SIZE
STRB:                   .DS              STRUCTURE_SIZE
STBC:                   .DS              STRUCUTRE_SIZE
```

Forming structures in this way has the advantage of automatically computing offsets and structure sizes while allowing the programmer to add or delete elements of the structure without re-computing the offset for each individual member. If this section is linked as a "reference only section" by preceding the "load offset" given at link time with a hyphen, then the bytes reserved by the DS directives will not be included in the output file. For more information regarding "reference only" refer to the linker section of this manual.

Note that the correct procedure for generating executable code at absolute addresses is:

(1) Assemble in relative mode.

(2) Supply the linker with a "load offset" of zero for these instructions at link time.

If executable instructions are assembled in Absolute mode, relative references will be calculated with absolute values. The result of this is that displacements will be an absolute number, just as if the symbol was defined with the EQUAL directive. Consider the following example:

```
        .CODE
        .ABSOLUTE
.ORG                20H
LABEL: NOP
        JMP         LABEL
        .END
```

where LABEL has a value of 20H. This **JMP** instruction will use a +20H as its displacement value. This is equivalent to the instruction:

```
        JMP         20H
```

where the next instruction executed will always be +20H bytes away from this JMP instruction.

Macros

Definition

A macro is a sequence of source lines that will be substituted for a single source line. A macro must be defined before it is used. The Assembler will store the macro definition and, upon encountering the macro name, will substitute the previously defined source lines. Arguments may be included in the macro definition. Arguments may be substituted into any field except the comment field.

For macro definitions, dummy arguments may not contain spaces. However, for actual macro calls, arguments may be any type; direct, indirect, character string or register. Spaces are not allowed in arguments unless it is an Ascii string, in which case the string must be bracketed in apostrophes. If the string contains an apostrophe, this can be specified with two apostrophes in a row. Arguments will be passed through to any nested macros if the dummy argument names are identical. Macro nesting is limited only by the amount of memory space available.

To define a macro the .MACRO directive is used. A macro must have the .MACEND or .ENDM directive following the macro definition. The name of the macro is in the label field.

Argument Separators

In the macro call line arguments must be separated by commas, however leading spaces and tabs are ignored. If no argument is present, a single comma will serve as a place holder.

The * as an argument will not be used as the program counter but as the multiplication sign. In a macro body, the following argument separators are allowed:

```
  + -* ** & A =OM
.NOT. .AND. .OR. .X0R. .EQ. .GT. .LT .GT.
.ULT..SHR..SHL.
```

Labels In Macros

Labels are allowed in macro definitions. Labels may be defined in two ways: explicit or implicit. Explicit labels in the macro definition will not be altered by the Assembler. Implicit labels are followed by a #. The Assembler will substitute a 3 digit macro expansion number for the #. In this case, the label and the macro expansion number must not exceed 32 characters. An argument may be used to specify a label.

String Concatenation

The broken bar character (I = hex 7C) is used as the string concatenation operator. Concatenation may only be performed inside of a macro.

Value Concatenation

Concatenation of a string and the value of an expression may be achieved by using the broken bar character (I = hex 7c) followed by a left angled bracket, the expression, and a right angled bracket. No spaces are allowed between the broken bar and the left angled bracket. Following is an example of this operation:

```
CONCAT          .MACRO ARG
VALUE:          .VAR        VALUE+1
ARGI<VALUE*2>   .EQU        31
                .ENDM
VALUE           .VAR        0
                CONCAT      LABEL
```

The invocation, CONCAT LABEL, will produce:

```
LABEL2          .EQU        31
```

It is important to initialize VALUE before the macro is invoked. Otherwise, the label being generated will have a different value on pass 1 and pass 2.

Mnemonic Definitions

The Assembler tables are searched in the following order:

1st	-	Mnemonic Table
2nd	-	Macro Definition Table
3rd	-	Assembler Directive Table
4th	-	Section Name Table

To redefine a mnemonic the MACFIRST directive may be used. This will switch the order of the search to Macro Definition Table first and Mnemonic Table second.

Macro Examples

A macro could be written to do string comparisons. This macro demonstrates the use of this feature.

```
CMP_STRING:             .MACRO      ARG1
!FNMA                               1
CMP_STRING NEEDS AN ARGUMENT
MACEXIT
END IF
IFSAME                              "JANUARY",ARG1
MONTH                   BYTE        1
MACEXIT
END IF
IFSAME                              "FEBRUARY",ARG1
MONTH                   BYTE        2
MACEXIT
END IF
IFSAME                              "MARCH",ARG1
MONTH                   BYTE        3
MACEXIT
END IF
IFSAME                              "APRIL",ARG1
MONTH                   BYTE        4
MACEXIT
END IF
```

```
IFSAME                                  "MAY",ARG1
MONTH               BYTE                5
MACEXIT
END IF
IFSAME                                  "JUNE",ARG1
MONTH               BYTE                6
MACEXIT
END IF
ARGUMENT ERROR IN MACRO STRING
ENDM
CMP_STRING "APRIL"
END
```


The following example demonstrates the use of argument substitution in the operand field of a macro.

```
EMPLOYEE _INFO:     .MACRO          ARGI ,ARG2,ARG3
NAME:               .DB             AFIG1
DEPARTMENT:         ASCII           ARG2
DATE_HIRED:         .LONG           ARG3
                    .ENDM
                    EMPLOYEE INFO   'JOHN DOE',PERSONNEL,101085
                    .END
```

This example could be changed to pass the argument into the label field. This enables the structure to be altered.

```
EMPLOYEE_INFO:      .MACRO      APG1,ARG2,ARG3
ARG1:               .DS     30H
ARG2:               .DS     10H
ARG3:               .LONG
                  .ENDM
                  EMPLOYEE INFO NAME,DEPARTMENT,DATE_HIRED
                  .END
```

The macro section also allows substitution into the mnemonic field. Also, a label can be generated within the macro with the # sign.

```
INSTRUCTION:        MACRO       ARG,VAL
                    ARC
LAB#:               DS          VAL
                    .MACEND
                    INSTRUCTION NOP,7
                    .END
```

To redefine a mnemonic the MACFIRST ON directive must precede the macro.

```
                MACF1RST ON
NO P:             .MACRO      ARG
                DB            ARG
                .ENDM
                NOP           FFH
                END
```

Another macro directive, MACDELIM, can be used to pass commas into a macro. The following examples show the syntax for this directive.

```
                MACDELIM
DELIM_EX:         MACRO       ARG1 ARG2
                BYTE          FFH,ARG1,ARG2 ENDM
                DELIM_EX      {,A4H},(,1 2111

                MACDELIM
DELIM_EX:         MACRO       ARG1
                BYTE          FFH ARG1 ENDM
                DELIM_EX      [,A4H]

                MACDELIM
DELIM_EX:         MACRO       ARG1
                BYTE          FFH ARG1 ENDM
                DELIM_EX      (,A4H)
```


Recursion

Below is an example of a recursive macro that reserves the number of data bytes defined by dummy argument ARG1 and fills them with the value specified by ARG2,ARG3,ARG4,ARG5,ARG6. This also demonstrates the use of MACEXIT and IFCLEAR. Count is decremented each time the loop is executed successfully. The macro is called again with the statement RESERVE the arguments following.

```
RESERVE: .MACRO     ARG1,ARG2,ARG3,ARG4,ARG5,ARG6
COUNT: .VAR         ARG1
        .1FZ        COUNT
        .1FCLEAR
        .MACEXIT
        .END1F
```

```
COUNT: .VAR             COUNT-1
       .BYTE            ARG2,ARG3,ARG4,ARG5,ARG6
       RESERVE          COUNT,ARG2,ARG3,ARG4,ARG5,ARG6
       .MACEND
```

This macro would be called with a statement such as the following:

```
RESERVE 10,AH,BH,CH,DH,EH          ; Fill 50 bytes with the sequence ABCDE.
```

It is perfectly legal for a recursive macro, such as the one in the above example, to call another recursive macro and so forth out to whatever level is desired. Also, note the use of the 'FCLEAR directive, which maintains the conditional IF - ENDIF pair balance. This can be used but is not required because the MACEXIT directive will return all conditionals to their original state.

Assembler Error Messages

Error - CAN'T CREATE OUTPUT FILE - DISK MAY BE FULL
Meaning -The disk may actually be full or the operating system is not allowing enough files to be open at onetime. See System Requirments to correct this error.

Error -CAN'T OPEN INPUT FILE
Meaning - The operating system is not allowing enough files to be open at one time. See System Requirements to correct this error.

Error - CAN'T FIND FILENAME.OBJ
Meaning - The .OBJ filename does not exist or the operating system is not allowing enough files to be open at onetime. See System Requirements to correct this error.

Error - SYNTAX ERROR

Meaning - Usually a missing comma or parenthesis.

Error - CAN'T RESOLVE OPERAND
Meaning - Can't tell what the programmer intended.

Error -ILLEGAL ADDRESSING MODE
Meaning - Can't address the operand using this form.

Error - ILLEGAL ARGUMENT
Meaning - Operand can't be used here.

Error - MULTIPLY DEFINED SYMBOL
Meaning - Symbol defined previously (not including '.VAR')

Error - ILLEGAL MNEMONIC
Meaning - Mnemonic doesn't exist and wasn't defined as a Macro.

Error - # TOO LARGE
Meaning - The destination is too small for the operand.

Error - ILLEGAL ASCII DESIGNATOR
Meaning - Bad punctuation on Ascii character.

Error - HEX # AND SYMBOL ARE IDENTICAL
Meaning - A label exists that is exactly identical to a hex number that is being used as an operand. Even the hex number indicator must be in the same place for this error to be generated.

Error -UNDEFINED SYMBOL
Meaning - Symbol wasn't defined during pass 1.

Error - RELATIVE JUMP TOO LARGE
Meaning - Destination address in a different page.

Error - EXTRA CHARACTERS AT END OF OPERAND Meaning -
Usually a syntax or format error.
Note - This error is the last check on any instruction before the Assembler proceeds to the next tine and indicates that there are extra characters after a legal operand terminator.

Error - LABEL VALUE CHANGED BETWEEN PASSES
Meaning - Symbol value decode during pass 1 not = pass 2.
Note - This error is usually caused by the Assembler taking different paths on Pass 1 as compared to Pass 2 due to conditional directive arguments changing value. The directive PASS1 ON/OFF can be useful in finding these types of errors.

Error - ATTEMPTED DIVISION BY ZERO
Meaning - Divisor operand evaluated to O.

Error -ILLEGAL EXTERNAL REFERENCE
Meaning -External reference can't be used here.

Error -NESTED CONDITIONAL ASSEMBLY UNBALANCE DETECTED
Meaning -Any type instruction without a matching '.ENDIF'

Error - ILLEGAL REGISTER
Meaning - The spectied register is not legal for the instruction

Error - CANT RECOGNIZE NUMBER BASE
Meaning - The number base specified is not one the assembler accepts.

Error - NOT ENOUGH PARAMETERS
Meaning - The were more arguments than parameters in a macro.

Error - ILLEGAL LABEL 1ST CHARACTER
Meaning - Labels must start with an alpha character.

Error - MAXIMUM EXTERNAL SYMBOL COUNT EXCEEDED
Meaning - There were too many externals in a module.
Note - There is a maximum of approximately 500 externals per module.

Error - MUST BE IN SAME SECTION
Meaning - The instructions operand is in a different section.

Error - NON-EXISTENT INCLUDE FILE
Meaning - The include file could not be found.

Error - ILLEGAL NESTED INCLUDE
Meaning - One included file contains an .INCLUDE directive. This error may also indicate that an included file did not have an END statement.

Error - NESTED SECTION UNBALANCE
Meaning - A nested section definition without an ENDS

Error - MISSING DELIMETER ON MACRO CALL LINE
Meaning - Unmatched delimeters when a macro was invoked.

Error - MULTIPLE EXTERNAL IN THE SAME OPERAND
Meaning - More than one external exists in the same operand.

Error - A LABEL IS ILLEGAL ON THIS INSTRUCTION.
Meaning -This is used to flag labels that would not obtain a relocation value. Such as ENDM or MACEND. Thus, the label is not allowed for the instruction.

Error - MACRO STACK OVERFLOW
Meaning - Macros are nested too deeply.
Note -This error can be caused by too many recursive macro calls. The stack has room for approximately 700 nested or recursive macro calls.The number of calls is affected by the number of arguments the macro uses.

Error - MISSING LABEL
Meaning - A label is required for this instruction.

Error - OPERAND MUST BE DEFINED AS AN 8 BIT RELOCATABLE VALUE. Meaning - This occurs when a 16 bit address is used in an 8 bit instruction. The < or > sign must be used to make the value relocatable.

Error - MISSING RIGHT ANGEL BRACKET
Meaning - Right angle bracket is mandatory.

Error - MACRO NAME MUST APPEAR ON SAME LINE AS MACRO DEFINMON

Error - ILLEGAL LOCAL LABELS
Meaning - Labels can't be defined as local. For example .VAR.

Error - MISSING MODULE DIRECTIVE

Error - MISSING ENDMOD DIRECTIVE

Error - 'Module' CAN'T BE IN 'Include' FILE

Error - 'Endmod' CAN'T BE IN 'Include' FILE