# SLR180

HD64180 Relocating Macro Assembler

## USER'S GUIDE



**SLR Systems**
1622 N. Main Street
Butler, PA 16001
(412) 282-0864

## COPYRIGHT NOTICE



## TRADEMARKS

## INTRODUCTION

**SLR180** is a powerful relocating macro assembler for Z80 compatible CP/M systems. It takes assembly language source statements from a disk file, converts them into their binary equivalent, and stores the output in either a core-image, Intel hex format, or relocatable object file. The mnemonics recognized are those of Zilog/Hitachi. The optional listing output may be sent to a disk file, the console and/or the printer, in any combination. Output files may also be generated containing cross-reference information on each symbol used.

## FEATURES

1) One pass operation (optional second pass)

2) Powerful nested macros,conditionals, and include files

3) Relocatable format allows extended math on externals and relocatables

4) Up to 15 different data, program, and common areas

5) Zilog/Hitachi mnemonics

6) Throughput of over 6000 Lines/Minute (8"SS/SD, 2Mhz)

7) Optional alphabetized symbol table

8) Optional alphabetized cross-reference

9) Directly generates a .COM, .HEX, or .REL file

10) Labels Significant to 16 characters

11) Supports time and date in listing

12) User Configurable

13) Supports ZCPR3 and CP/M+ error reporting

# TABLE OF CONTENTS

## RUNNING THE ASSEMBLER

To run the assembler, type

A>**SLR180** [COMMAND][,COMMAND]

where the brackets are not really typed, and what is enclosed in them is optional. If no COMMAND's are given on the initial command line, **SLR180** will prompt for a command line with a percent (%) sign. **SLR180** allows as many commands to be given as will fit on a 128 character line. Commands are separated by a comma. Note that input from the prompt is via the Read Console Buffer system call, so that commands may be passed through a SUBMIT file. However, there is a better way...

COMMANDs are defined as follows:

<Filename>[.<Drive>][/<Option>]

where **<Filename>** is the source file with the (default) extension 180.

. is a separator used only when <Drive> is present.

**<Drive>** is used to modify the default drive selection. The default condition selects the CP/M default drive for source file, enables binary output to the default drive, and enables the disk listing driver to the default drive. <Drive> is from one to three letters (@-P, where @ is the default drive) defined as follows:

first letter: drive on which <Filename>.180 is located

second letter: selects drive to receive output file - to disable binary file generation, use a Z in this location. If this letter is missing, output will go to the default drive.

third letter: selects the drive or device on which to place the listing output. If selected, the cross-reference and symbol table also go to this device. To send the output to a disk file, use a valid drive letter in this location. To output just to the console, use an X; Y selects output to the list device. If no listing-type output is desired, use a Z in this spot. If this letter is missing, output will go to the default drive.

/ is optional, but must be used when <Option> is used.

<Option> is one or more of the following letters, in any order:

A : absolute mode. Output file is COM.

C : enables output to the console.

D : don't do LOWER to UPPER case conversion.

E : execute LOWER to UPPER case conversion.

F : selects full listing (2 passes through source).

H : generate HEX file as output.

I : special indirect command file.

K : kill all console I/O and detach.

L : list output partial (1 pass through source).

M : generate Microsoft REL file as output.

N : new OPTION byte.

P : enables output to the list device.

Q : quit, abort.

R : generate standard REL file as output.

S : generate alphabetized symbol table.

T : input time and date string.

U : declares undefined symbols as external.

X : generates cross-reference.

Y : disable cross-reference.

6 : selects M-Rel output, 6 significant.

7 : selects M-Rel output, 7 significant.

For example, assuming standard default conditions:

```
A>SLR180 EXAMPLE/F,DUMP.ABZ/SXRP
```

tells **SLR180** to assemble A:EXAMPLE.180, create A:EXAMPLE.COM as the binary output, and create A:EXAMPLE.LST as the second pass full listing output. Next assemble A:DUMP.180, creating an S-REL file on drive B, and generating a symbol table and cross-reference on the printer.

**Efficiency Hints**

1. Use one-pass mode wherever possible. The only time two pass mode is necessary is generating a cross-reference, or a listing with forward references resolved.

2. If generating a one-module program, generate the COM or HEX output directly from the assembler.

3. For multiple module work, use an indirect file. For most assemblies, a major percentage of the assembly time is merely loading in the assembler. SLR180 will assemble all the files in an indirect file without reloading the assembler.

4. If you 180FIGure in a 2nd pass listing or cross-reference, you will ALWAYS get 2-pass mode unless you use the proper switches to disable it (/N).

Let's talk in more detail about the command line options.

## COMMAND LINE OPTIONS

Command line options are used to modify the default operating characteristics of **SLR180**. This is the default mode as supplied by SLR Systems (the default mode may be modified by the user by running the 180FIG utility):

> Generate a core-image .COM file on one pass through the source, converting lower-case items to upper-case, generate no listing output, no symbol table output, and no cross-reference output.

So, if you wanted to assemble the file DUMP.180 (included on your distribution diskette) to create the file DUMP.COM, both on drive B:, with no other output, you would type

**A>SLR180 DUMP.BB**

Note that the Z was not needed in the listing drive spot since no listing-type output was requested. If the files were on drive A,

**A>SLR180 DUMP**

would produce the desired result.

There are many other available options though, and this section describes them in a little more detail.

**A**

> The A option selects absolute mode operation and a default output file type of COM. This is the standard default mode, so it will not need to be used unless the default mode has been modified.

**C**

> The C option enables the console output driver. This has no effect in itself, but if any listing, symbol table or cross-reference outputs are selected, these items will be output to the console device (Error messages are always enabled to the console, unless /K is used).

D

The D option disables LOWER to UPPER case conversion. This is used when it is desired to have upper and lower case characters treated differently, such as when assembling output from a C compiler. Note that when this is in effect, reserved words (mnemonics, registers, and pseudo-ops) may be in any case or mixture. .LIST may be redefined as a macro, and .LISt will still be recognized as a pseudo-op.

E

The E option enables LOWER to UPPER case conversion. This is the standard default mode. Note that the conversion only affects characters not inside quoted strings, and that lower case characters are still lower case in a list file.

F

The F option selects FULL listing mode. This option forces **SLR180** into 2-pass mode, generating a listing output on the second pass to any and all enabled devices. Note that by default the disk driver is enabled.

H

The H option selects a binary output type compatible with the Intel HEX format.

I

The I option is a special case option used for indirect command file input. If this option is used, it can be the only option specified. The /I option causes **SLR180** to read the file <FILENAME>.SUB on the default drive (or the selected source drive) for its command lines. This is similar to using SUBMIT and XSUB except that 1) it is much faster, 2) it can be used from within a SUBMIT file, 3) it can be used without XSUB and SUBMIT leaving more RAM available for the assembly, and 4) it can be used while A: is not the default drive.

K

The K option kills all console I/O, including error messages. Also, under multitasking systems, a 'detach console' is issued slipping **SLR180** into the background.

**L**

The L option selects one-pass listing mode. This listing is similar to the F-mode full listing except that forward references list as undefined in the generated code columns. Note that by default the output will go to the default disk drive.

**M**

The M option forces relocatable mode. The binary output file has the default extension REL, and the format generated is compatible with the Microsoft relocatable format.

**N**

The N option is used to start over with new options. For instance, if you have M-REL format with symbol table and XREF selected from 180FIG, N will delete those options and put you back to the standard default mode.

**P**

The P option enables the PRINTER or CP/M list device output driver. Any listing, symbol table, or cross-reference output selected will be sent to the CP/M list device. Error messages will also be sent to the printer, even if no other listing output has been selected.

**Q**

The Q option is used to abort interactive command line acceptance. It is identical in function to a control-C, but it may also be used at other than the beginning of the line, from SUBMIT files, and from INDIRECT command files, all places where ^C is illegal.

**R**

The R option is used to select relocatable operation. The output file type is REL, and the format generated is SLR Format.

S

The S option selects the generation of an alphabetized symbol table. The output goes to all enabled devices. The output contains symbol names, types, and values. The number of symbols per line depends on the selected page width.

T

The T option is used to specify a time and date string for use in generating listings. Any following characters (up to 16) up to the next comma or carriage return are used as the time and date string. This option must be used as the last slash option. The input string is used automatically for subsequent assemblies unless overridden by another T option, or **SLR180** is reloaded.

U

The U option (ignored in absolute mode) instructs **SLR180** to automatically declare any UNDEFINED labels as externals to be resolved by the linker.

X

The X option selects cross-reference generation. This option forces two-pass mode, and the output will go to all enabled devices.

Y

The Y option deletes the XREF option if it was already selected (like from 180FIG).

6

The 6 option performs an implied M option, and selects 6 significant character generation for globals and externals. This overides the default selected by 180FIG.

7

The 7 option does the same as 6 except that **SLR180** will generate 7 significant characters for globals and externals.

**ASSEMBLER RUNTIME CONTROL**

There are several operations available at the console while **SLR180** is in operation. For instance, at any time, a ^C (Control-C) may be typed, causing **SLR180** to abort the assembly of the current file. This allows you to abort the assembly of a large file that had the wrong command line options, without rebooting.

Assembly operation may be temporarily halted with a ^S, and resumed with a ^Q.

The Console driver may be enabled and disabled (toggled) with a ^Z, and the list device driver may be toggled with a ^P. Note that these may be used whether or not their drivers were enabled in the command line. These are very useful for watching just some of the assembly where errors are occurring, or printing out just part of the assembly without inserting LIST and NLIST pseudo-ops in the source (if you are quick!).

A '?' causes **SLR180** to display the current filename and line number in the file so you can see exactly how things are progressing.

Any other character is merely echoed on the console and ignored.

# SOURCE LINE FORMAT

Source input to the assembler comes from a disk. Each line of input must be less than 128 characters, and must be terminated by a CR-LF sequence. Input lines follow the following syntax:

[Line #] [Label[:]] [Operation] [Parameters] [Comment]

where anything inside brackets is optional.

Each line is processed one at a time. First, if lower to upper case conversion is enabled (as in the standard default case), any lower case characters not inside quotation marks are converted to upper case, so that upper and lower case characters are interchangeable for labels, opcodes, etc.

## Line #

The line number is an optional field which contains a valid number. This item is optional and is ignored by the assembler.

## Label

Labels or symbols are made up of characters from the following set:

% $ ? . @ 0-9 A-Z a-z _

The first character of a label must not be from 0-9.

Symbols may be any length, however only the first 16 characters are significant. This rule holds true throughout unless M-REL operation is selected, in which case sixteen characters are still significant internally, but only six or seven (see 180FIG) are passed to the linker for entry points and external references. S-REL format allows full 16 significant characters on all references.

The colon following a label is optional in most instances. Two exceptions are 1) when the label is a reserved word, and 2) when the label is defined as both a macro and a label. In both these cases, the colon is required to distinguish the label from the reserved word or macro reference.

The following are valid symbols used as labels:

```
Label          ??ERROR        @spec
Never_Again    $12345         .ABCDEFGHIJKL
```

This example shows the colon used only where it is required:

```
0100 CPIR      MACRO     XX        ;Changes CPIR to a JP
0200           JP        XX
0300           ENDM
0400
0500 Start
0600           CPIR      CPIR      ;Means Jump to Label CPIR
0700 CPIR:                         ;Colon skips scanning the
0800           END                 ;Macro and reserved word
0900                               ;tables
```

Labels may start in any column.

**Operation**

This item can be one of the standard Zilog/Hitachi mnemonics for a 64180 instruction, one of the standard **SLR180** pseudo-ops, or a user-defined macro instruction. The macro table is searched first, allowing redefinition of any opcodes or pseudo-ops.

The standard Zilog mnemonics are defined well in other publications, so we will not discuss them in too much detail here, but several things should be pointed out.

For ease of use, **SLR180** will except long and short forms for the 8-bit operations. For instance, in Zilog mnemonics, to add 5 to the accumulator, you say

```
ADD  A,5
```

which makes sense. To add with carry the same thing, you say

```
ADC  A,5
```

which still makes sense. Subtract with carry is the same way, but Subtract without carry is different. It is used like this:

```
SUB  5
```

which is not very consistent. At times it can be hard to remember which form is needed.

**SLR180** alleviates this problem of inconsistency by accepting both forms for all 8-bit operations. For example,

```
ADD   A,5       ;produces what you would expect
ADD   5         ;produces the same thing
ADC   A,5       ;Yep
ADC   5         ;Same thing
SBC   A,5       ;Again
SBC   5         ;Same thing
CP    0f0h      ;generates a 0FEH, 0F0H
CP    A,0f0h    ;so does this
```

are all acceptable statements to **SLR180**. Use whichever you wish. However, for portability, you may want to adhere to the strict syntax.

Another comment should be made about the operation field. If **SLR180** is configured not to require colons on labels not starting in column 1, there is an area of ambiguity if you type an operation incorrectly. For instance, if you wanted the instruction OTIR, but incorrectly entered it:

```
OUTIR           ;OUTPUT ENTIRE BLOCK
```

**SLR180** will not generate an error, will not generate the proper code for OTIR, but will happily define a label OUTIR at that location. The only way **SLR180** will catch an error like that is if you do it twice, which will generate a 'previously defined' symbol error. For this reason it is recommended that you select colon requirements.

### Parameters

These may be the operands required by the given opcode or pseudo-op, or may be parameters to a macro instruction. Expressions and macros will be discussed later.

### Comments

Comments must start with a semicolon ';'. Everything on the line after the semicolon is ignored by the assembler.

## EXPRESSIONS

Operands often consist of expressions. Expressions are merely combinations of operators and operands. For example,

```
1          3+5          -Start          HIGH 0FE23h
```

are all valid expressions. Expressions can often get much more complex than that. If there is more than one operator in an expression, the expression is evaluated in the order of the operator 'precedence'. This table lists all the available operators, their functions, and their precedences.

| Operator | Function | Precedence |
|---|---|---|
| - | Unary Minus | 1 |
| + | Unary Plus | 1 |
| NOT or ~ | Bitwise NOT | 1 |
| HIGH | Take HIGH Byte | 1 |
| LOW | Take LOW Byte | 1 |
| NUL | Special NUL | 1 |
| TYPE | Returns Type | 1 |
| * | Unsigned Multiply | 2 |
| / | Unsigned Divide | 2 |
| MOD | Unsigned Modulo | 2 |
| SHR or >> | Shift Right | 2 |
| SHL or << | Shift Left | 2 |
| + | Add | 3 |
| - | Subtract | 3 |
| EQ or = | Equal | 4 |
| NE or <> | Not Equal | 4 |
| LT or < | Less Than | 4 |
| LE or <= | Less Than or Equal | 4 |
| GT or > | Greater Than | 4 |
| GE or >= | Greater Than or Equal | 4 |
| AND or & | Bitwise AND | 5 |
| XOR | Bitwise XOR | 6 |
| OR or \| | Bitwise OR | 6 |

Some operators have equivalent forms, i.e.,

```
XM & 3    is identical to    XM AND 3
```

in all respects. Note that the spaces around the '&' are not necessary, but are recommended to lessen the chances of confusion in macro processing. Embedded spaces are allowed between any items in an expression.

Default operator precedence may of course be overridden by use of parentheses. Be careful not to use parentheses at both ends of an expression unless you mean 'contents of'. For instance,

```
LD      HL,(1+3)*(4+7)      ;is treated as  LD  HL,(44)
                            ;not  LD  HL,44 as you would expect.
```

Note also that the characters (, [, and { are equivalent, as are ), ], and }, and may be used interchangeably. However, ( and ) are recommended for transportability and future compatibility.

A special case of an expression is the null expression, valid only in index register instructions. For example

```
LD      D,(IX)      ;is identical to   LD  D,(IX+0)
```

and is perfectly legal.

**TYPE**

The TYPE operator needs some explanation. It is a unary operator that returns the type of its operand expression. The type is defined as follows:

If the expression contains a forward reference, or is complicated enough to be sent to the linker to be resolved at link time, TYPE returns a ZERO. If the expression contains an external symbol plus or minus a constant or relative item, TYPE returns an 80H. Otherwise, the expression is defined locally, and TYPE returns a 20H plus one of the following:

| | |
|---|---|
| 0 | if absolute |
| 1 | if CSEG relative |
| 2 | if DSEG relative |
| 3 | if relative to a common block |

This is used mainly in handling macro parameters.

**NUMBERS**

Numbers are by default radix 10. The default radix may be overridden by using one of the following forms:

| | |
|---|---|
| nB | Binary |
| nO | Octal |
| nQ | Octal |
| nD | Decimal |
| nH | Hexadecimal |
| X'n' | Hexadecimal |

where n is any number of digits valid for the given radix. Note that the nB and nD forms will not be recognized if the default radix has been changed to larger than 11 or 13 respectively since the trailing character would be a valid digit in the default radix. Also, the first digit in the nH-type must be from 0-9.

## SPECIAL SYMBOLS

A $ is recognized as a special symbol representing the load counter at the beginning of the current statement or substatement. For instance, to loop forever:

```
        JR   $
```

is the same as

```
HERE:   JR   HERE
```

since $ is the load counter at the beginning of the instruction.

## STRINGS

Strings are sequences of characters delimited by either single or double quote pairs. For example,

```
'This is a string'
"This is also a string"
'This is not a legal string"
```

Strings may contain quotation marks, and are defined in several possible ways. A single quote may be contained in a string delimited by double quotes, and vise versa, by just using it:

```
'This is an "easy" example'
"It isn't nice to fool mother nature"
```

Of course, you can use a single quote in a single quote string by just using it twice, and the same holds for double quote strings.

```
"This is an ""easy"" example"
'It isn''t nice to fool mother nature'
```

## RELOCATABILITY

Many applications require the generation of relocatable code. This section describes the legal operations available on relocatable items.

What is a relocatable item? It is any item that is not defined at assembly time, but will be defined at link time. These items may be program relative, data relative, common relative, external, or any mathematical combination of the above items with other relocatable items or absolute items.

A special relocatable type is a relative type. A relative type is a program, data, or common relative item. Certain combinations of relatives generate absolute results, mainly a relative item minus another relative of the same type. For instance, a program relative item minus a program relative item yields an absolute result. Any other combination of relatives yields a "relocatable" item.

Absolutes may be added to or subtracted from relatives, yielding another relative.

Certain pseudo-ops require absolute values, such as "if" statements. Expressions that yield relative or relocatable results are illegal there and also in DEFS, etc.

Certain pseudo-ops can accept relative parameters. These are EQU, DEFL, .PHASE, and ORG.

Most other places where expressions are involved, any relocatable expression is allowed.

Suppose you wanted to load HL with a number equal to the distance from a data segment location to a program segment item. With most relocatable assemblers, this cannot be done directly. It must be calculated at run time like this:

```
LD      HL,DATAITEM
LD      DE,PROGITEM
OR      A
SBC     HL,DE
```

**SLR180** allows you to do directly what you originally wanted:

```
LD      HL,DATAITEM-PROGITEM
```

Obviously that can't be resolved at assembly time, so the expression is passed in reverse polish form to **SLRNK** for resolution at link time.

Certain combinations are not allowed in standard Microsoft format output; **SLRNK** Appendix B lists the operations that are legal and those **SLR Systems** has added. All combinations are allowed for use between **SLR180** and **SLRNK**, but certain operators are "extensions".

For example, you can generate a byte as a result of one or more relocatables like this:

```
LD      E,DATAITEM*EXTERNAL1+EXTERNAL2
```

**SLRNK** will give an error message if the result is out of range for a byte quantity.

It might even be nice to have a library routine that turns on a green light by setting the correct bit of a memory location, with the bit # depending on the application. You could then say

```
SET     GREEN-1,(HL)    ;TURN ON GREEN LIGHT
```

where GREEN is an external. **SLR Format** and modified M-REL can both handle that case.

# PSEUDO-OPERATIONS

**SLR180** makes many pseudo-ops available to the assembly language programmer. They will be described in this section.

The pseudo-ops available are divided into 6 categories for this discussion. The categories are

1. Program Counter Maintenance
2. Data Definition and Generation
3. Conditional Assembly
4. Macro Facility
5. Listing Control
6. Miscellaneous Pseudo-ops

Here comes the first category!

# 1. PROGRAM COUNTER MAINTENANCE

This category includes pseudo-ops that control the program counter and the loading address of the generated code.

**ABS**
**ASEG**

Absolute Segment. In a relative-mode assembly, this opcode causes **SLR180** to generate code for execution at a particular address, in this case, the address after the last absolute byte generated. If this is the first ASEG, then the address is 100H. This is very similar to absolute MODE except that code generated still has access to other relocatable items, such as externals. The generated code still needs to be linked before execution. This pseudo-op takes no parameters. This pseudo-op is ignored in absolute mode.

**ORG** <EXP>

Origin Program Location. This pseudo-op is used to specify a loading address <EXP> for the code that follows. The logical program counter is also set to <EXP>. The expression must yield an absolute or relative result defined on pass 1. ORG forces the mode of the expression. For example, if the expression yields a data-relative result, ORG forces an implied DSEG first. If the expression is absolute mode, ORG forces an implied ABS pseudo-op first:

```
ORG     100H
```

will cause code to be generated starting at address 100H. This is the default condition in absolute mode.

If no parameter is given, ORG returns the PC back to the value just preceding the last ORG.

**CSEG**
**PROG**
**REL**

Select Code or Program Loading Counter. This pseudo-op selects the code segment loading address and logical program counter. This is the default address space in relative mode. The address selected is that of the last code segment address used, which starts at 0. This pseudo-op is used to return to the code segment after having switched to another address space. For example,

```
0000' 21 0000"              LD   HL,MESG    ;Load Pointer to Mes
0003' CD 0000#              CALL OUTPUT     ;Output Mes
                            DATA            ;Put Mes in Data Spc
0000" 48 69 20   MESG       DB   'Hi There';Generate Mesg
                            PROG            ;Return to CSEG
0006' C9                    RET             ;End of Subroutine.
```

The data string actually will be located elsewhere at link time, and the RET instruction will be just after the CALL OUTPUT. The Program segment is usually used for program or machine code. This pseudo-op takes no parameters, and is illegal in absolute mode.

**DATA**
**DSEG**

Select Data Segment Loading Counter. This pseudo-op is normally used for generating data to be loaded at another area in the target machine. It is also useful for defining address space in RAM away from the code area in PROM. The loading counter and logical PC selected is the one from the last used DATA area. For example, if this code followed the code above, look what would be generated.

```
                            DSEG            ;define addresses in
0008"            LBL1       DS   1          ;RAM storage
0009"            LBL2       DS   10         ;more RAM storage
```

The actual load location for this address space is selected at link time with the /D or /A option. The DSEG address space is usually used for data storage. This pseudo-op takes no parameters, and is illegal in absolute mode.

**COM** /COMNAME/
**COMMON** /COMNAME/

Common Block. This pseudo-op selects the named common block for the new code type. The block name, like other symbols, is significant to the first 16 characters internally, to 16 externally in SLR Format, 7 in M-REL format.

Unlike PROG and DATA address spaces, common areas are not different in each module. For example, the first byte in the PROG area for module 1 will be at a different physical address than the first byte in the PROG area for module 2. With Common blocks, the address spaces are the same to each module, which allows access to the same data space from different modules.

The actual physical address of the common blocks is selected at link time with the /C or /A option.

This pseudo-op is illegal in absolute mode.

**.PHASE** <EXP>

Select Logical PC. This pseudo-op selects a new logical program counter, without changing the load counter. It is useful for generating code at the current location for later transfer and execution at location <EXP>. This pseudo-op is valid in any mode, and <EXP> can evaluate to an absolute or relative value.

Once inside a .PHASE block, all PC maintenance pseudo-ops are illegal until the next .DEPHASE pseudo-op is encountered, which brings us to another point.

**.DEPHASE**

End of Logical Phase Block. This pseudo-op terminates the current .PHASE block. For example,

```
                          ORG   100H
 0100 21 010E             LD    HL,RETT     ;Return Instr Physical
 0103 11 2000             LD    DE,DESTIN   ;Return Instr Destination
 0106 01 0001             LD    BC,1        ;Return Instr Length
 0109 ED B0               LDIR              ;Move it
 010B 21 2000             JP    DESTIN      ;Jump to It
                  RETT    .PHASE 2000H      ;Actual Execution addr
 2000 C9          DESTIN  RET               ;Generated at 010E with
                          .DEPHASE          ;Logical address of 2000H
```

Phase blocks may not be nested.

# 2. DATA DEFINITION AND GENERATION

This sections contains the pseudo-ops that are involved in defining and generating all types of data.

**DB** [EXP1][,EXP2]
**DEFB** [EXP1][,EXP2]
**DEFM** [EXP1][,EXP2]

Define Byte. This pseudo-op allows the generation of bytes of data. Data can be a single expression or string, or several separated by commas. In relocatable mode, the expressions may be absolute, relative, or relocatable. Note that a null item will generate no output, nor is it a syntax error. This is to accomodate missing parameters in macro expansions. For example,

```
0100 00 01 02 03  NUMS   DEFB      0,1,2,,,3
```

and

```
0104 40 ?? 02 48         DB        40H,LOW EXTERN1,'B'&1FH,'Hi'
```

are legal statements.

**DW** [EXP1][,EXP2]
**DEFW** [EXP1][,EXP2]

Define Word. This pseudo-op is used to generate 16-bit quantities, low byte first. Data can be a single expression or string, or several separated by commas. In relocatable mode, the expressions may be absolute, relative, or relocatable. Note that a null item will generate no output, nor is it a syntax error. This is to accomodate missing parameters in macro expansions. For example, the following are legal statements:

```
0100  0100  ????  START    DEFW      START,EXTERNAL SHR 4
0104  69 48 54 20          DW        'Hi There'
```

**DS** <EXP1>[,<EXP2>]
**DEFS** <EXP1>[,<EXP2>]

Define Space. This pseudo-op is used to allocate address space for variable storage or whatever. The address counter is incremented by <EXP1>, which must evaluate to an absolute value on the first pass. No code or data is generated to fill the address space skipped unless the optional <EXP2> is specified. <EXT2> is used to fill the declared space with a byte value. Note that <EXT2> does not need to be defined at assembly time. Note also that during COM file generation, if <EXP2> is missing, the space is filled with zeros. For example,

```
BUFA    DEFS    200H        ;Room for 512 byte buffer
```

declares a buffer BUFA with a size of 512 bytes. In HEX or REL mode, the initial contents of this buffer are not defined. On the other hand,

```
BUFA:   DS      200H,EMPTY ;Room for 512 byte buffer
```

declares the same buffer, but its contents are initialized to EMPTY, which possibly won't be defined until link time. The colon after the label is optional unless the label is a reserved word, or is also used as a macro name.

<LABEL> **EQU** <EXP>

Equate Directive. This pseudo-op declares a value <EXP> to be assigned to a label. <EXP> must evaluate to an absolute or relative item by the second pass. The label being defined must not have been previously defined. For example, the following statements

```
FALSE   EQU     0           ;false is 0
TRUE:   EQU     NOT FALSE ;true is ffff
```

assign values to symbols TRUE and FALSE. The colon after the label is optional in all cases.

<LABEL> **DEFL** <EXP>
<LABEL> **ASET** <EXP>

Define Label. This pseudo-op is identical to EQU except that it can be used to redefine a label that has been previously defined. For instance, it could be used to update a counter:

```
COUNT   DEFL    COUNT+1     ;increment count
```

```
EXT         LABEL1[,LABEL2]
EXTRN       LABEL1[,LABEL2]
EXTERNAL    LABEL1[,LABEL2]
BYTE EXT        LABEL1[,LABEL2]
BYTE EXTRN      LABEL1[,LABEL2]
BYTE EXTERNAL   LABEL1[,LABEL2]
```

External Declaration. This pseudo-op tells the assembler that the symbols in this list are not defined in this module, but are defined in another module, to be resolved at link time. Valid only in relocatable mode, this pseudo-op is necessary only if you don't use the /U option in your command line.

Symbols may also be declared external by following them by two pound-signs (##) when they are referenced. For example,

```
        LD      BC,EXTERNAL##
```

declares EXTERNAL to be an external.

```
ENT         <LABELIST>
ENTRY       <LABELIST>
GLOBAL      <LABELIST>
PUBLIC      <LABELIST>
```

Entry Point Declaration. This pseudo-op tells the assembler to make this list of symbols and their values available to the linker so that they may be referenced as EXTernals from other modules. This is not allowed in absolute mode.

```
        PUBLIC    LABEL1,LABEL2,LABEL3
```

Entry points may also be defined by following the symbol by two colons (::) when the symbol is defined.

```
LABEL1::        LD   A,3
LABEL2::        EQU   $
```

declares LABEL1 and LABEL2 as globals.

Symbols declared public must be defined on the first pass when generating SLR Format output.

**.ACCEPT** [STRING,]<SYMBOL>

The .ACCEPT pseudo-op is used to request a value for a symbol from the console. <SYMBOL> is a valid symbol name not elsewhere defined in the file. **SLR180** will print the optional STRING on the console, or if no STRING is given then the <SYMBOL> name, followed by a '?', and wait for the value. The value must be a valid expression, defined on the first pass (you can use symbols already defined). For example,

```
.accept   DEBUG_FLAG
```

will cause **SLR180** to print

```
DEBUG_FLAG ?
```

on the console. At this point you type a valid expression. If TRUE has already been defined, you could type

```
DEBUG_FLAG ? NOT TRUE
```

to set DEBUG_FLAG to false.

Since console buffer read is used, you can supply your input through a submit file.

Only one <SYMBOL> can be used per .ACCEPT statement.

**DEFC** EXP1[,EXP2]
**DC** EXP1[,EXP2]

The DEFC pseudo-op is used to generate strings that terminate with the high bit (bit 7) set. The <EXP>'s may be strings or valid byte data:

```
DEFC      'This terminates at the LF',0dh,0ah
```

In this case the 0ah is converted to an 8ah.

**DEFZ** EXP1[,EXP2]

The DEFZ pseudo-op is also used in generating data strings, but this one terminates with an extra zero byte.

```
DEFZ      'hi'
```

generates 68H,69H,00H.

# 3. CONDITIONAL ASSEMBLY

This section describes the facility whereby sections of code may be selected for assembly based on certain conditions.

One very useful way to utilize conditional assembly is in program debugging. For instance, you can insert code to print out intermediate values of variables, or messages to say where the program is, etc. By putting all that code in conditional assembly blocks, you would not have to delete the code after debugging, or re-insert it when something else crops up. You could just define a symbol DEBUG at the top of the module that defines whether or not the debug code is included in the assembled code. The following pseudo-ops allow conditional assembly.

**NOTE:** For any of the IF-type pseudo-ops that take an expression as a parameter, 180FIG can be used to select whether to look at just bit 0 (for DRI compatibility) or the whole 16-bit value.

**IF** <EXP>
**COND** <EXP>
**IFT** <EXP>

Conditional Assembly. This pseudo-op allows conditional assembly of a section of code. For instance, if you have code that you want assembled only during program debugging, you could use the statement

```
IF      DEBUG
```

to tell **SLR180** whether the following code should be assembled. If the expression <EXP> evaluates to an absolute zero, then the subsequent code up to the next active ELSE or ENDIF is not assembled. A non-zero value yields a true IF condition, causing the subsequent lines to be assembled. Since IF blocks may be nested, active means on the same nest level.

The expression must yield a defined absolute result on the first pass.

Conditional blocks may be nested as deeply as you need (theoretically to 65535 levels...).

**IFE** <EXP>
**IFF** <EXP>

This pseudo-op has the opposite effect of the IF pseudo-op. If the given expression evaluates to 0, a true IF condition is generated.

**ENDIF**
**ENDC**

Endif directive. This pseudo-op declares the end of the last conditional block declared.

**ELSE**

Else directive. This pseudo-op toggles the current IF condition. If the current IF condition is false, now it is true, and vise versa. For example,

```
PRNTOUT  ;call printer output routine
         IF        PARPNT          ;if parallel printer
         CALL      PAROUT          ;call parallel output
         ELSE                      ;otherwise
         CALL      SEROUT          ;call serial output
         ENDIF                     ;terminate if block
```

if PARPNT is true, the first call will be assembled, otherwise the second one will be used.

**IF0**

This pseudo-op is a special form of the IF pseudo-op. It takes no parameters, but assembles the following lines of code only in one-pass mode assemblies.

**IF1**

This is similar to the IF0 pseudo-op, except that it returns a true conditional only on pass 1 of 2-pass mode.

**IF2**

This is similar to the IF1 pseudo-op, except that it returns a true conditional only on pass 2 of 2-pass mode.

**IFDEF** <SYMBOL>

The IFDEF pseudo-op takes a valid symbol as an operand, and assembles the following lines of code only if the symbol has been defined in the module as absolute, relative or external.

**IFNDEF** <SYMBOL>

The IFNDEF pseudo-op performs the opposite function as IFDEF. The conditional block is assembled only if the given symbol is undefined.

**IFIDN** <PARAM1>,<PARAM2>

The IFIDN pseudo-op assembles the conditional block only if the character strings (not necessarily valid STRINGs) passed as PARAM1 and PARAM2 are identical. This is useful in testing macro parameters. Note that the angle bracket delimiters are required.

**IFDIF** <PARAM1>,<PARAM2>

The IFDIF pseudo-op performs the opposite function of IFIDN, the parameters must be different to assemble the conditional block. Note that the angle bracket delimiters are required.

**IFB** <PARAM>

IF BLANK. If the parameter passed is blank, a true IF is generated. This can be used to test for the existence of macro parameters. The 180FIG program can be used to select whether or not a space is treated as a blank parameter (ignore leading spaces). Note that the angle bracket delimiters are required.

**IFNB** <PARAM>

Opposite of IFB. IF NOT BLANK. Note that the angle bracket delimiters are required.

# 4. MACRO FACILITY

Macro facilities provide the ability to reproduce sequences of instructions repetitively and simply. **SLR180** supports the Intel standard macro facility, which includes several different types of macros.

The simplest form of macros are the inline macro types, REPT, IRP, and IRPC. These all store source lines for immediate repetition after which the lines are discarded.

[LABEL] **REPT** <EXP>

Repeat Macro. The repeat macro reads a group of statements up to the next matching ENDM or MEND pseudo-op. That is the BODY of the macro. The group of statements are then scanned and assembled <EXP> times. <EXP> must be defined and absolute on the first pass.

The body of the REPT macro may contain other proper macro definitions including other REPT macros. The body can also contain nested conditional pseudo-ops. Any active conditionals are automatically terminated at the end of each repetition of the macro, so that IF and ELSE directives are not required to have their matching ENDIF statements if they begin within the macro body.

In the following example we use the REPT macro to build a table of bytes from 20 to 0.

```
                0014            1 ??VAR   DEFL    20        ;set initial value
                                2         REPT    21        ;repeat 21 times
                                3         DEFB    ??VAR     ;generate the byte
                                4 ??VAR   DEFL    ??VAR-1   ;decrement the var
                                5         ENDM              ;end of macro body
0100  14              A         1         DEFB    ??VAR     ;generate the byte
0101  13              A         1         DEFB    ??VAR     ;generate the byte
0102  12              A         1         DEFB    ??VAR     ;generate the byte
0103  11              A         1         DEFB    ??VAR     ;generate the byte
0104  10              A         1         DEFB    ??VAR     ;generate the byte
0105  0F              A         1         DEFB    ??VAR     ;generate the byte
0106  0E              A         1         DEFB    ??VAR     ;generate the byte
0107  0D              A         1         DEFB    ??VAR     ;generate the byte
0108  0C              A         1         DEFB    ??VAR     ;generate the byte
0109  0B              A         1         DEFB    ??VAR     ;generate the byte
010A  0A              A         1         DEFB    ??VAR     ;generate the byte
010B  09              A         1         DEFB    ??VAR     ;generate the byte
010C  08              A         1         DEFB    ??VAR     ;generate the byte
010D  07              A         1         DEFB    ??VAR     ;generate the byte
010E  06              A         1         DEFB    ??VAR     ;generate the byte
010F  05              A         1         DEFB    ??VAR     ;generate the byte
0110  04              A         1         DEFB    ??VAR     ;generate the byte
0111  03              A         1         DEFB    ??VAR     ;generate the byte
0112  02              A         1         DEFB    ??VAR     ;generate the byte
0113  01              A         1         DEFB    ??VAR     ;generate the byte
0114  00              A         1         DEFB    ??VAR     ;generate the byte
```

[LABEL] **IRPC** <DUMMY>,<CHARLIST>

Indefinite Repeat Character. The IRPC macro reads the body of the macro, just like the REPT macro. The body of the macro is then scanned and assembled once for each character in the given CHARLIST. However, any occurrence of the item DUMMY in the body is replaced by the current character in CHARLIST. Well, almost any occurrence. See the section on dummy parameter evaluation.

As with the REPT macro, this can contain nested macros and conditionals.

In the next example, we use an IRPC macro to generate a string of characters, each with the high bit set to 1. Note that the DUMMY parameter XX is replaced by a single character for each iteration of the macro. Note also that an ampersand must be used in front of or after the DUMMY in order for it to be replaced within a quoted string.

Note also that comments started with two semi-colons are not stored with the macro, and therefore 1.) don't take up memory space, and 2.) don't appear in the expansion.

```
                          1     IRPC   XX,MESSAGE1 ;set hi bit of each
                          2     DB     '&XX'+80H    ;;this comment gone
                          3     ENDM
0100  CD          A       1     DB     'M'+80H
0101  C5          A       1     DB     'E'+80H
0102  D3          A       1     DB     'S'+80H
0103  D3          A       1     DB     'S'+80H
0104  C1          A       1     DB     'A'+80H
0105  C7          A       1     DB     'G'+80H
0106  C5          A       1     DB     'E'+80H
0107  B1          A       1     DB     '1'+80H
```

[LABEL] **IRP** <DUMMY>,<<PARAMLIST>>

Indefinite Repeat. The IRP is very similar to the IRPC, except that the parameters are multi-character items. Note that the brackets around the PARAMLIST are required.

In the following example, we generate four error messages with labels. Note that the label is constructed by 'concatenating' the letter M to the parameter. The ampersand (&) is used to do that.

```
                                1           mtlist
                                2           IRP     XX,<MACRO,.PHASE,IF,.DEPHASE>
                                3 XX&M:     DEFB    'missing &XX',0DH,0Ah
                                4           ENDM
0100  6D 69 73 73A              1 MACROM:   DEFB    'missing MACRO',0DH,0Ah
0104  69 6E 67 20A              1
0108  4D 41 43 52A              1
010C  4F 0D 0A   A              1
010F  6D 69 73 73A              1 .PHASEM:  DEFB    'missing .PHASE',0DH,0Ah
0113  69 6E 67 20A              1
0117  2E 50 48 41A              1
011B  53 45 0D 0AA              1
011F  6D 69 73 73A              1 IFM:      DEFB    'missing IF',0DH,0Ah
0123  69 6E 67 20A              1
0127  49 46 0D 0AA              1
012B  6D 69 73 73A              1 .DEPHASEM: DEFB 'missing .DEPHASE',0DH,0Ah
012F  69 6E 67 20A              1
0133  2E 44 45 50A              1
0137  48 41 53 45A              1
013B  0D 0A      A              1
                                5
```

<LABEL> **MACRO** [<DUMMYLIST>]

Macro Definition. This is the most sophisticated form of macro item. This statement declares the beginning of a MACRO definition. The <LABEL> must be present; it is the name by which the macro will be referred to later. It follows the same rules as labels. A colon after the label is optional.

The <DUMMYLIST> is optional. If present, it declares <dummy> items which will be replaced by other items when the macro is called. Multiple <dummy> items are separated by commas. <Dummy> items follow the same rules as labels.

All source lines following the macro header are stored in memory under the name <LABEL>, up to the next matching ENDM or MEND. Since macros may contain macro definitions (nested macro definitions), matching means the ENDM at the same nest level.

To call the macro, just use the macro name as you would any other opcode or pseudo-op, passing with it any desired parameters.

Suppose you are writing a program for CP/M systems in which you do a lot of displaying messages on the console. It might be nice to design a macro that would let you easily do that. You could then call the PRINT macro to print a string on the screen. For example,

```
                          1          .LALL
                          2 PRINT    MACRO    XX
                          3          LD       DE,STRING
                          4          LD       C,9
                          5          CALL     5
                          6          JP       STEND
                          7 STRING   DEFB     XX,0DH,0AH,'$'
                          8 STEND
                          9          ENDM
                         10          PRINT    'This is a test'
0100  11 010B     A      1          LD       DE,STRING
0103  0E 09       A      2          LD       C,9
0105  CD 0005     A      3          CALL     5
0108  C3 011C     A      4          JP       STEND
010B  54 68 69 73A      5 STRING   DEFB     'This is a test',0DH,0AH,'$'
011C               A      6 STEND
                         11
```

The line PRINT 'This is a test' generated the next 6 lines of macro output, to set up the data and system call to output the string on the console. But there is a problem. If the PRINT macro had been called again, the labels STRING and STEND would have been multiply defined, which is illegal (you won't go to jail, your program just won't assemble).

One way around that would be to pass another parameter to concat to the labels, to make them unique. **SLR180** provides another way, which will be discussed next.

**LOCAL** <LABELIST>

Local Labels. This pseudo-op declares a label or set of labels to be local to the current macro expansion, i.e., each time the macro is expanded or repeated, generate a unique label for each of these. The labels in <LABELIST> must be legal labels. The LOCAL statement should occur before the labels are referenced in the macro.

**SLR180** implements LOCAL labels by generating a unique label for each LOCAL consisting of two question marks followed by 4 hexadecimal digits. For example, the first LOCAL label encountered in an assembly would be replaced by the label ??0001. Therefore, you should avoid the use of labels that have that form in order to lessen the chances of confusion.

LOCAL label replacement is done at expansion time, not at macro definition time. This implies that parameters passed to a macro that match a local symbol name will also be replaced by a local label.

Note that LOCAL labels are valid in all macro types.

```
                                1          .LALL
                                2 PRINT    MACRO   XX
                                3          LOCAL   STRING,STEND
                                4          LD      DE,STRING
                                5          LD      C,9
                                6          CALL    5
                                7          JP      STEND
                                8 STRING   DEFB    XX,0DH,0AH,'$'
                                9 STEND    ENDM
                               10          PRINT   'This is a test'
                        A       1          LOCAL   STRING,STEND
0100  11 010B             A     2          LD      DE,??0001
0103  0E 09               A     3          LD      C,9
0105  CD 0005             A     4          CALL    5
0108  C3 011C             A     5          JP      ??0002
010B  54 68 69 73A         A     6 ??0001   DEFB    'This is a test',0DH,0AH,'$'
011C                       A     7 ??0002
                               11
                               12          PRINT   'This is a STRING test'
                        A       1          LOCAL   STRING,STEND
011C  11 0127              A     2          LD      DE,??0003
011F  0E 09                A     3          LD      C,9
0121  CD 0005              A     4          CALL    5
0124  C3 013F              A     5          JP      ??0004
0127  54 68 69 73A         A     6 ??0003   DEFB    'This is a ??0003 test',0DH,0AH,'$'
013F                       A     7 ??0004
                               13
                               14          END
```

**EXITM**

Exit macro expansion. This pseudo-op allows an easy way to terminate a macro expansion. It is usually used in conjunction with a conditional test.

EXITM is also allowed in an include file or maclib file, to halt the processing of those files.

Note that EXITM exits only a macro expansion, not a definition. Use ENDM or MEND to end the macro definition.

**INCLUDE** <FILENAME.EXT>
**$INCLUDE** <FILENAME.EXT>

Include another source file. This pseudo-op takes as an argument a valid CP/M file name. That disk file is then opened and inserted at the point. Note that your listing line numbers start over at 1, and that a letter precedes that number. That is the nesting level. Any errors generated while processing this included file will give the Included file name and line number in that file. Note also that the included file can also contain an INCLUDE directive. At the end of the included file, assembly continues at the previous file, next line. This is a very powerful feature for including common macro definitions, and even assembling large projects without resorting to relocatable code.

**MACLIB** <FILENAME.EXT>

Include Macro Definition File. This pseudo-op is identical to an INCLUDE, except that it is ignored on the second pass. This pseudo-op is used to read macro definition files, or any source files that produce no output code. This is to save time on 2-pass assemblies since the whole MACLIB file will be skipped on the second pass.

## Dummy Parameter Evaluation

While reading the body of a macro, **SLR180** looks for the occurrence of any of the optional dummy parameters in the text. There are several rules that are followed.

1. Dummy parameters are not replaced in a comment field.

2. A dummy parameter delimited by spaces, tabs, commas, etc., and not inside a quoted string, is automatically flagged for replacement at expansion time.

3. Any dummy parameter preceded or followed by an ampersand (&), whether or not it is contained in a quoted string, is flagged for replacement, and the leading and/or trailing ampersand is removed. This is useful for dummy replacement in strings, and also for concatenating a parameter to something else.

4. Any dummy parameter preceded by an up-arrow (^) is ignored. The up-arrow is used to cause a character to be taken literally, i.e., an up-arrow can be represented as two up-arrows in succession.

## Parameter Evaluation Rules

There are a number of special options available to the programmer when it comes to parameter passing. Here we will describe the rules that **SLR180** follows when evaluating parameters.

First, all leading blanks are skipped. In other words, the parameter starts at the first non-blank character. That first non-blank character determines what type of parameter is being evaluated.

If the first character is a quotation mark (single or double), then the parameter is processed as a STRING, following the rules for strings. The leading and trailing quotation marks are passed as part of the parameter.

If the first character is a '<' (less-than sign, refered to in this discussion as a left bracket), then the '<' is removed, and the text is processed up to the next matching '>'. By matching we mean that the brackets are nestable, so that if there are three left brackets, the parameter is processed up to the third right bracket exclusive. Strings encountered in the bracketed parameter are treated as such, and follow the rules for strings. Brackets contained in a string are not counted as part of the nesting level.

If the first character is a percent sign (%), then the expression evaluator is called to process the following valid expression. The expression must resolve to a defined absolute value. This value (16-bit unsigned) is then converted to a number in the current radix, which is then used as the parameter.

Otherwise, the item is treated as a regular old parameter.

Note that a special character in macros is the up-arrow. It causes characters to be taken literally. For instance, to pass an up-arrow (^) as a parameter, use two in succession. To pass a semicolon as a parameter, use ^;, which is equivalent to <;>. Try using the following parameter:

<This is a ^> and this is an ^^>

and see what you get.

One condition that occurs very often is a macro call with a missing or null parameter. This can be discovered at expansion time by using the special operator NUL. NUL returns a true value (-1) if what comes after it is nothing but tabs, spaces, comment field, or an end of line. Anything else will return a false value to NUL.

Look at the following example.

```
                          1
        0055              2 XX        EQU     55H
                          3
                          4 MAC1      MACRO   ?XX,?YY
                          5           DEFB    ?YY       ;SECOND
                          6           IF      NUL ?XX
                          7           EXITM
                          8           ELSE
                          9           IRPC    XX,?XX
                         10           LOCAL   LABEL
                         11 LABEL     DW      LABEL
                         12           DEFB    ^^XX
                         13           DEFB    '&XX'&1FH
                         14           ENDM
                         15           ENDM
                         16
                         17           MAC1    CONTROL,'DATA'
0100  44 41 54 41A        1           DEFB    'DATA'    ;SECOND
0104  0104          B     2 ??0001    DW      ??0001
0106  55            B     3           DEFB    XX
0107  03            B     4           DEFB    'C'&1FH
0108  0108          B     2 ??0002    DW      ??0002
010A  55            B     3           DEFB    XX
010B  0F            B     4           DEFB    'O'&1FH
010C  010C          B     2 ??0003    DW      ??0003
010E  55            B     3           DEFB    XX
010F  0E            B     4           DEFB    'N'&1FH
0110  0110          B     2 ??0004    DW      ??0004
0112  55            B     3           DEFB    XX
0113  14            B     4           DEFB    'T'&1FH
0114  0114          B     2 ??0005    DW      ??0005
0116  55            B     3           DEFB    XX
0117  12            B     4           DEFB    'R'&1FH
0118  0118          B     2 ??0006    DW      ??0006
011A  55            B     3           DEFB    XX
011B  0F            B     4           DEFB    'O'&1FH
011C  011C          B     2 ??0007    DW      ??0007
011E  55            B     3           DEFB    XX
011F  0C            B     4           DEFB    'L'&1FH
                         18 ;
                         19           MAC1    ,XX
0120  55            A     1           DEFB    XX        ;SECOND
                         20
                         21           END
```

# 5. Listing Controls

**TITLE** <ANYTHING>

Listing Title. This defines a new title to be put at the top of subsequent pages of listing. Takes effect BEFORE this line is printed, so this can be used in the first line of your source file to put a title on the first page. The title may be up to 80 characters, and may be truncated by **SLR180** to fit on the page.

Note that any tabs are compressed to single spaces. 180FIG can enable or disable the listing of the source line containing the TITLE directive.

**<ANYTHING>** can be in one of two forms. If the first non-blank character is a left paren, then the token is assumed to be in the following form:

(STRING)

where STRING is a valid delimited string, like 'Hi there y''all'. The delimiting quotes do not become part of the stored string.

If the first non-blank character is not a left paren, then the stored string begins with the first non-blank up to the first semicolon or CR. For example

```
     TITLE     This is my 'Title'
```

and

```
     TITLE     ('This is my ''Title''')
```

are equivalent.

**SUBTTL** <ANYTHING>
**$TITLE** <ANYTHING>

Listing Subtitle. This defines the subtitle to be used on subsequent listing pages. The subtitle can be up to 60 characters long. See TITLE for a definition of <ANYTHING>.

The default subtitle is the filename of the file being assembled. The subtitle is reset upon entry and exit to/from any INCLUDE file to the name of the current source file.

**PAGE** [PLENGTH][,PWIDTH]
**EJECT** [PLENGTH][,PWIDTH]
**$EJECT** [PLENGTH][,PWIDTH]
***EJECT** [PLENGTH][,PWIDTH]

Top of Page. If no parameters are given, this pseudo-op causes a new page to be started in the listing. Otherwise, one or two parameters are expected, the first being the new page length in lines, and the second being the new page width in columns. The standard default page length and width are set with 180FIG. Note that spaces are allowed between the * and EJECT.

```
PAGE    80,132      ;page = 80 lines by 132 columns
```

**LIST**
**.LIST**

Listing On. This is the default mode. This has effect only if a listing is being generated. This affects overall listing.

**NLIST**
**.XLIST**

No List. This pseudo-op stops all listing generation. This can be countered only by a subsequent LIST pseudo-op.

**MTLIST**

Multi-line List. This pseudo-op enables listing of extra lines generated when the given source line generates more than 4 bytes of output. The default case is to just list the first 4 bytes with the source line. For instance, the line

```
DEFB    'I Like Peanut Butter'
```

generates 20 bytes of code. The default case will just list the first four bytes, while MTLIST will cause 5 lines to be printed.

**NMTLIST**

No Multi-line List. This pseudo-op suppresses the listing of extra lines of output caused by more than 4 bytes being generated from one source line. This is the standard default case.

**CLIST**
**.LFCOND**

Conditional List. This pseudo-op enables listing of lines contained in FALSE conditional assembly blocks. Standard default is to list only the lines that are in TRUE conditional blocks.

**NCLIST**
**.SFCOND**

No Conditional Listing. This pseudo-op causes the suppression of listing output during FALSE conditional block processing. This is the standard default case.

**.TFCOND**

Toggle Conditional List. This pseudo-op causes the listing of conditional blocks to be turned OFF if it was ON, and ON if it was OFF.

**.LALL**

List All. This pseudo-op controls macro expansion listing only. It causes **SLR180** to list every line of macro expansion (does not override NLIST or .XLIST).

**.SALL**

Suppress All. This pseudo-op controls macro expansion listing only. It causes the suppression of all macro expansion listing.

**.XALL**

Partial List. This pseudo-op controls macro expansion listing only. .XALL causes **SLR180** to list only the lines that generate code output.

# 6. Miscellaneous Pseudo-ops

The ones that don't fit in anywhere else.

**END** [STARTADR]

End directive. This signifies end of input file to the assembler. The optional argument is used, in HEX and relocatable mode, to signify a starting address for the program. Note: this directive is ignored if encountered during a macro definition or within a FALSE conditional block.

```
END     START               ;start at start...
```

**NAME** <ANYTHING>

Module name. Valid only in relocatable code, this assigns a name to the generated relocatable module. If no name is defined, **SLR180** uses the source file name as the module name. See TITLE for a description of <ANYTHING>. Maximum size 16 chars. Syntax:

```
NAME      MYFILE
NAME      ('MYFILE')
```

**.REQUEST** <LIBLIST>

Request Library Search. This pseudo-op causes **SLRNK** to search the given filename.REL for any undefined externals. The modules are loaded as needed. The given filename.REL is assumed to reside on the default disk at link time. Note that the filename may be eight characters long in M-REL format, but only six characters if you included a drive specifier. SLR Format allows eight in either case (actually 16, but CP/M filenames are limited to eight). Requests may pass more than one filename, separated by commas. They will be searched in the order in which they are defined. For example:

```
REQUEST     C:MYLIB,FORLIB
```

**.Z80**

This pseudo-op has no effect.

**.CREF**

Cross-reference ON. This pseudo-op turns on the cross-reference builder for the subsequent source lines. This is the default condition. This pseudo-op is useful only when cross-reference generation has been selected by a command line option or through 180FIG.

**.XCREF**

Cross-reference OFF. This pseudo-op turns off the cross-reference builder for the subsequent lines of source up to the next .CREF. Useful only when cross-reference generation is in use, ignored otherwise.

**.COMMENT** <delim>

Comment text. This pseudo-op is used to turn text into a comment. The next non-blank character defines the <delim>iter. The rest of the source file up to line containing the next <delim> character is considered a comment field. This is useful for commenting out a chunk of code without putting a semicolon at the beginning of each line.

**.PRINTX**

Print text. This pseudo-op takes the rest of the source line and places it on the console. It is useful in displaying messages, values of parameters (by passing the parameter using a leading % to a macro containing .printx), etc.

**.RADIX** <EXP>

Set input and output radix. This pseudo-op is used to change the default input and output radix from base 10 to something else in the range of 2-16. The default radix is set to 10 before the <EXP> is processed. This is usually used to enter a group of data in a base such as 16 without requiring the trailing H. It also affects the base used in processing macro parameters preceded by a %.

# APPENDIX A - **ERROR MESSAGE SUMMARY**

There are many error conditions that **SLR180** can detect. This section describes the error messages that can be generated.

Error messages are of the form:

FILENAME - ERROR MESSAGE Line # ?????
<This Line Is The Source Line That Contains The Error Condition>

where FILENAME is the name of the current file or macro being processed, and ????? is the line number of that file or macro on which the error was discovered.

Note: errors discovered while processing the intermediate code give code address instead of line number for the error.

**Bad Dummy Param**

Dummy parameters must follow the syntax for valid labels.

**Bad File Name**

Something is syntactically incorrect with the given file name.

**Bad Opcode**

Something was found in the opcode field that is not in the opcode list, pseudo-op list, or macro table.

**Byte Out of Range**

This can occur in several places, with meaning depending on the instruction involved. For relative jumps and indexed addressing, the value generated must lie within the range of -128 to +127. For BIT, RES, and SET instructions, the bit number must be between 0 and 7. For RST instructions, the operand must be 0, 8, 10H, 18H, etc. Finally, for the IM instruction, the operand must be 0, 1, or 2. Something other than these was calculated.

**Comma Expected**

Guess what? You're right, a comma was expected. Possibly something extra ahead of your comma, or else your comma is missing. Or something.

**Expression Error**

This error message will only be generated if the expression evaluator gets very confused. Should not happen...

**Extra Operand**

Look at that line closely, **SLR180** found more items on it than belonged. Possibly you just forgot a semicolon before your comment, but it may be more critical than that.

**File Not Found**

**SLR180** tried to open the given file unsuccessfully.

**ID, (, or Unary Op Expected**

The expression evaluator was looking for an ID (valid number or label), left paren, or Unary Operator (such as HIGH or LOW). Something else was there.

**Illegal in Absolute File**

A pseudo-op was encountered that is valid only in relocatable assembly.

**Illegal - 0 used**

The expression could not be evaluated down to the type required by the pseudo-op on this line. A zero was used instead.

**Intermediate Corrupt**

An item was found while processing the intermediate code that should not be there. Try to determine what led up to that and report the bug.

**Line Too Long**

Source lines are limited to 127 characters in length.

**Macro Redef**

Macro Table is confused about a macro redefinition. Possibly you redefined the macro differently on different passes through MACLIB, IF1, IF2 or some other conditional situation. Macro redefinition is allowed, but obviously must be the same on each pass.

**Missing .DEPHASE**

Something was encountered that is illegal inside a PHASE block. Possibly you forgot a .DEPHASE pseudo-op.

**Missing ENDIF**

Oops! End of file encountered while in a conditional. Where was your ENDIF ?

**Missing ENDM**

Oops! End of file encountered while building a macro. Probably a missing ENDM or MEND.

**Missing IF**

An ELSE or ENDIF without IF.

**Missing MACRO**

Something was encountered that is not legal outside a macro, such as an ENDM, MEND, or LOCAL.

**Missing .PHASE**

A .DEPHASE pseudo-op was found without a preceding .PHASE pseudo-op. Check your phase blocking.

**No Backing Up in COM File**

Only in absolute mode, COM type files. The address given in the ORG statement is smaller than the last location generated. You cannot go backwards while generating a COM file.

**Out Of Memory**

No room left in memory. Symbol table, macro table, include tables, cross-reference table, etc, must all fit into memory. Ask about **SLR180+.**

**Previously Defined**

This LABEL has been defined already elsewhere in the file and is being redefined with something other than a DEFL. Change the name of one of your labels.

**String Syntax**

Strings can contain only printable characters. Either your string contains a control character (such as a TAB), or you forgot the trailing quotation mark.

**Symbol Expected**

Something was found that is illegal as an operation or label. Probably something was mis-typed on the line.

**Syntax Error**

This message appears when the assembler is too confused to know what went wrong.

**Too Many Commons**

An attempt was made to define more than 12 common blocks. That is the limit. Ask about **SLR180+.**

**Unexpected EOF**

The end of the source file was found at an inopportune time.

**')' Expected**

Somewhat self-explanatory.

**'/' Expected**

Somewhat self-explanatory. Common block name must be delimited by slashes, and contain only valid label-type characters.

**), Bin-op, or End of Exp Expected**

Well, the expression evaluator was looking for one of the above, a right paren, a binary operator (an operator that takes two arguments, such as * or /), or an end-of-expression delimiter. Something else was there.

# APPENDIX B - **HITACHI HD64180 INSTRUCTION SET**

The following definitions are used throughout this description of the HITACHI HD64180 Instruction Set.

| | |
|---|---|
| cc | C, NC, Z, NZ, P, M, PE, or PO (V & NV allowed in **SLR180** in place of PE and PO) |
| dd | C, NC, Z, or NZ |
| nn | Any 16-bit number or expression |
| (nn) | Represents the contents of address nn |
| n | Any 8-bit number or expression |
| r | Register A, B, C, D, E, H, or L |
| d | A one-byte expression in the range -128 to 127 |
| b | Expression in the range 0 through 7 |
| ss | Register pairs BC, DE, HL, SP |
| pp | Register pairs BC, DE, IX, SP |
| qq | Register pairs BC, DE, HL, AF |
| rr | Register pairs BC, DE, IY, SP |
| ir | (HL), (IX+d), or (IY+d) |
| m | r, (HL), (IX+d), or (IY+d) |
| s | r, n, (HL), (IX+d), or (IY+d) |

| | |
|---|---|
| ADC HL,ss | HL = HL + Carry + ss |
| ADC A,s | Acc = Acc + Carry + s |
| ADD A,n | Acc = Acc + n |
| ADD A,r | Acc = Acc + r |
| ADD A,(ir) | Acc = Acc + (ir) |
| ADD HL,ss | HL = HL + ss |
| ADD IX,pp | IX = IX + pp |
| ADD IY,rr | IY = IY + rr |
| AND s | Acc = Acc AND s |
| BIT b,m | Z = BIT b of m |
| CALL cc,nn | IF cc is true, call subroutine at nn |
| CALL nn | Call subroutine at nn |
| CCF | Complement Carry |
| CP s | Set Flags as result of Acc - s |
| CPD | CP (HL), DEC HL, DEC BC |
| CPDR | CP (HL), DEC HL, DEC BC, Rpt till BC=0 or Acc=(HL) |
| CPI | CP (HL), INC HL, DEC BC |
| CPIR | CP (HL), INC HL, DEC BC, Rpt till BC=0 or Acc=(HL) |
| CPL | Complement Acc - Acc = -(Acc + 1) |
| DAA | Decimal Adjust Acc |
| DEC m | m = m - 1 |
| DEC IX | IX = IX - 1 |
| DEC IY | IY = IY - 1 |
| DEC ss | ss = ss - 1 |
| DI | Disable interrupts |
| DJNZ e | B = B - 1, if B<>0, Jump to e |
| EI | Enable interrupts |
| EX (SP),HL | Exchange HL with (SP) top of stack |
| EX (SP),IX | Exchange IX with (SP) top of stack |

```
EX (SP),IY      Exchange IY with (SP)  top of stack
EX AF,AF'       Exchange AF with AF'
EX DE,HL        Exchange DE with HL
EXX             Exchange HL, DE, & BC with HL', DE', & BC'
HALT            Halt CPU till interrupt or reset
IM 0            Select Interrupt Mode 0
IM 1            Select Interrupt Mode 1
IM 2            Select Interrupt Mode 2
IN A,(n)        Acc = input from port n
IN r,(C)        r = input from port (C)
IN0 r,(n)       r = input from port n
INC m           m = m + 1
INC IX          IX = IX + 1
INC IY          IY = IY + 1
INC ss          ss = ss + 1
IND             (HL) = input from port (C), DEC HL, DEC B
INDR            (HL) = input from port (C), DEC HL, DEC B, rpt till B=0
INI             (HL) = input from port (C), INC HL, DEC B
INIR            (HL) = input from port (C), INC HL, DEC B, rpt till B=0
JP (HL)         PC = HL  (Jump to location (HL))
JP (IX)         PC = IX
JP (IY)         PC = IY
JP cc,nn        If cc is true, PC = nn
JP nn           PC = nn
JR dd,nn        If dd is true, jump relative to nn
JR nn           Jump relative to nn
LD A,(BC)       Acc = contents of location (BC)
LD A,(DE)       Acc = (DE)
LD A,I          Acc = I  (Interrupt Vector Reg)
LD A,(nn)       Acc = location nn
LD A,R          Acc = R  (Refresh Reg)
LD (BC),A       location (BC) = Acc
LD (DE),A       location (DE) = Acc
LD (ir),n       Store n in location (ir)
LD (ir),r       Store r in location (ir)
LD ss,nn        ss = nn
LD ss,(nn)      ss = contents of nn
LD I,A          I = Acc
LD IX,nn        IX = nn
LD IX,(nn)      IX = contents of nn
LD IY,nn        IY = nn
LD IY,(nn)      IY = contents of nn
LD (nn),A       Store a in location (nn)
LD (nn),ss      Store ss in location (nn)
LD (nn),HL      Store HL in location (nn)
LD (nn),IX      Store IX in location (nn)
LD (nn),IY      Store IY in location (nn)
LD R,A          Store A in R
LD r,(ir)       r = location (ir)
LD r,n          r = n
LD r,r          r = r
LD SP,HL        SP = HL
LD SP,IX        SP = IX
LD SP,IY        SP = IY
LDD             Location (DE) = Location (HL), DEC HL, DE, & BC
```

| | |
|---|---|
| LDDR | (DE) = (HL), DEC HL, DE, & BC, REPEAT TILL BC=0 |
| LDI | Location (DE) = Location (HL), INC HL & DE, DEC BC |
| LDIR | (DE) = (HL), INC HL & DE, DEC BC, Repeat till BC=0 |
| MLT ss | ss = LOW ss * HIGH ss |
| NEG | Acc = -Acc |
| NOP | Nothing |
| OR s | Acc = Acc OR s |
| OTDM | Port (C) = (HL), DEC HL, DEC C, DEC B |
| OTDMR | Port (C) = (HL), DEC HL, DEC C, DEC B, Repeat till B=0 |
| OTDR | Port (C) = (HL), DEC HL & B, Repeat till B = 0 |
| OTIM | Port (C) = (HL), INC HL, INC C, DEC B |
| OTIMR | Port (C) = (HL), INC HL, INC C, DEC B, Repeat till B=0 |
| OTIR | Port (C) = (HL), INC HL, DEC B, Repeat till B=0 |
| OUT (C), r | Port (C) = r |
| OUT (n),A | Port n = A |
| OUT0 (n),r | Port n = r |
| OUTD | Port (C) = (HL), DEC HL & B |
| OUTI | Port (C) = (HL), INC HL, DEC B |
| POP IX | IX = (SP), SP = SP + 2 |
| POP IY | IY = (SP), SP = SP + 2 |
| POP qq | qq = (SP), SP = SP + 2 |
| PUSH IX | SP = SP - 2, (SP) = IX |
| PUSH IY | SP = SP - 2, (SP) = IY |
| PUSH qq | SP = SP - 2, (SP) = qq |
| RES b,m | Reset bit b of m |
| RET | PC = (SP), SP = SP + 2 (Return from sub.) |
| RET cc | If cc is true, PC = (SP), SP = SP + 2 |
| RETI | Return from interrupt |
| RETN | Return from non-maskable interrupt |
| RL m | Rotate m left through carry |
| RLA | Rotate Acc left through carry |
| RLC m | Rotate ma left circular |
| RLCA | Rotate Acc left circular |
| RLD | Rotate (HL) & Lower nibble of Acc left 4 bits |
| RR m | Rotate m right through carry |
| RRA | Rotate Acc right through carry |
| RRC m | Rotate m right circular |
| RRCA | Rotate Acc right circular |
| RRD | Rotate (HL) & Lower nibble of Acc right 4 bits |
| RST p | SP = SP - 2, (SP) = PC, PC = p (Single byte call) |
| SBC A,s | Acc = Acc - Carry - s |
| SBC HL,ss | HL = HL - Carry - ss |
| SCF | Set Carry Flag |
| SET b,m | Set bit b of m |
| SLA m | Shift m left arithmetic |
| SLP | Sleep |
| SRA m | Shift m right arithmetic |
| SRL m | Shift m right logical |
| SUB s | Acc = Acc - s |
| TST r | FLAGS = Acc AND r |
| TST (HL) | FLAGS = Acc AND (HL) |
| TST n | FLAGS = Acc AND n |
| TSTIO n | FLAGS = Port (C) AND n |
| XOR s | Acc = Acc XOR s |

# APPENDIX C - **INTEL HEX FILE FORMAT**

The HEX file optionally generated by **SLR180** is the standard INTEL format HEX file. The file consists of one or more records of ASCII text defined below.

Record Syntax

```
:NNAAAATTDDDDDDDDDDDDDDDDDDXX
```

The first character is a colon (:)
The next two characters (NN) define the record length (00 - FF)

If the length is non-zero:

AAAA Defines the load address of the record. It is forward reading, i.e., High byte first.

The next two characters are the record type. TT is always zero for data records, 01 for ending record.

The following NN*2 characters define the actual binary to be loaded starting at AAAA.

XX is a checksum byte which, when added to the sum of all the previous bytes in this recrod, gives a total of zero (mod 256).

If NN = 0, then the value at AAAA is the optional starting or execution address. A record length zero also signifies end-of-file.

# APPENDIX D - 180FIG UTILITY

The 180FIGuration utility 180FIG.COM provided with SLR180 allows easy customization of SLR180 to a particular set of requirements. This section briefly describes its use.

To run 180FIG, type

```
A>180FIG SLR180.COM
```

where SLR180.COM is the file name you want to customize.

180FIG will respond with a series of questions requesting flag settings or item values. In parentheses is the current value of that item. You can modify the current value by typing a new one, or the item can be left the same by just entering a CR. The process can be aborted by typing a ^C. After the last question is answered, 180FIG will write out the new settings.

Now we will explain each question that 180FIG asks.

**Page Width (80) -**

The current default page width is 80 columns. Any listing output wider than that will be truncated. This should be set to the width of the printer you use most often, in columns (255 max). This parameter can be changed at assembly time with the PAGE pseudo-op.

**Page Length (60) -**

The current default page length is 60 lines. A form feed will be generated after every 60 lines of listing. This should be modified if your page size is different. This parameter can be modified at assembly time with the PAGE pseudo-op.

**Special Bits**

This allows you to modify the default assembly type, which is: absolute, COM-type, Lower to upper is enabled, no symbol table, no cross-reference, no automatic Externals. 180FIG tells you what the different bits in the byte stand for. You can toggle any bit by just typing in a number from 0 to 7. To leave that item the way it is, just type CR.

**List more than 4 bytes of object code (N) -**

The default setting is to list up to 4 bytes of object code for each source line. Any more than that (such as in a DB string) are not printed. A **Y** in this flag would cause extra bytes to be printed on successive listing lines, four per line. This flag has the same effect as the MTLIST & NMTLIST pseudo-ops.

**List lines encountered during false conditionals (N) -**

The default setting is to suppress the listing of lines encountered during a false conditional block. A Y in this flag will cause false conditional lines to be printed. This flag can be modified at assembly time by the CLIST and NCLIST pseudo-ops.

**Form Feed at start of listing (N) -**

The default setting causes no top-of-form to be issued before the first line is printed. This assumes that your printer is already at the top of a new page when the listing begins. If you want a leading Form Feed, just type a **Y** here.

**Macro Listing Option - 1=.LALL, 2=.XALL, 4=.SALL (2) -**

This controls the macro listing default setting. Read the descriptions of .LALL, .XALL, and .SALL to see if you want to change the default setting. You can change it by typing a 1, 2, or 4.

**Generate 6 Significant in M-Rel instead of 7 (N) -**

The Microsoft REL format supports 7 significant characters for globals and externals. However, most software that utilizes M-Rel format truncates to 6 characters (Fortran compatibility ?). You have your choice of 6 or 7 characters of significance.

**Print 16-bit values in logical direction (Y) -**

This controls the byte order (in the listing) for 16-bit values when printed. If you prefer to see the bytes in the order generated, use an **N**. The default case prints the high byte first.

**Suppress Lines Containing PAGE, TITLE, etc (N) -**

This controls the listing of lines that contain the pseudo-ops PAGE, TITLE, and .PRINTX. A **Y** here will cause those lines to be suppressed from any listing.

**Disable Interrupts (N) -**

**SLR180** was written to take full advantage of the Z80/64180 architecture and instruction set for reasons that are self-explanatory when you see it run. Certain systems that are 'Z80' CP/M machines do not provide a true Z80 environment. If your system is interrupt driven and destroys any Z80 registers on interrupts, a **Y** here will cause **SLR180** to disable interrupts whenever any special Z80 registers are in use. Interrupts will always be enabled during any system calls.

**Force Form Feed before Summary (N) -**

This allows you to control the summary printing. By default, the summary will be printed at the end of the listing. You can force the summary to be placed at the top of a new page by putting a **Y** here.

**Form Feed at End of Listing (Y) -**

This causes SLR180 to issue a form feed at the end of all listing output. This usually makes it easier for you to remove the listing from your printer, and also sets the top-of-form for the next listing without wasting any paper. The Form Feed can be eliminated by typing an **N** here.

**Time and Date in Listing (N) -**

If you are running on a system that provides time & date, **SLR180** provides the ability to have the time and date printed on the title line of your listing. To enable this, put a **Y** here.

**Special TIME & DATE Items**

The next three questions may be ignored if you 1) answered **N** to the above question, or 2) you are running under an operating system that provides MP/M compatible time and date (CP/M Plus, TurboDos, etc.).

If your system supports time and date, but not the system call 105 to return time and date, you can still have **SLR180** utilize your time and date information. The information must either be in the same form as that supplied by CP/M Plus:

```
DW    DATE      ;Number of days since 1 Jan 1978
                ;with 0001 being 1 Jan 1978
DB    HOURS     ;Hour of the day in BCD
DB    MINUTES   ;Minutes of the hour in BCD
```

or it must be in ASCII, 16 chars max, terminated by a -1.

If you can get your time and date formatted like that somewhere in your system, you can just tell 180FIG that you are supplying the address of the data. If you need to have some code executed to retrieve the time and date, **SLR180** will optionally call a subroutine for you, you just supply the address of the routine. The routine must emulate the System Call 105, that being, **SLR180** will supply an address in DE where you should store one of the above structures. All Z80 registers are available for your use, just be careful not to get too deep on the stack (you have 30 bytes to play with). Then lastly you need to tell **SLR180** which structure type you are returning.

**Take advantage of multi-sector I/O (Y) -**

System call 12 tells **SLR180** whether multi-sector I/O is available or not, at least usually. Certain 'CP/M compatible' operating systems return a code saying they are CP/M Plus or MP/M compatible, but they don't support multi-sector I/O. If that is your case, you will need to answer **N** here.

**Use Statement #'s instead of Line # in file (N) -**

This allows you to choose between two different line-numbering schemes. Statement #'s are numbers assigned as the listing is generated. This causes numbers to to be generated in numerical order even when macro's are expanded, include files processed, etc. Line #'s are the actual line number in the source file being processed. These reset to 1 on include files, macro expansions, etc, are fully nestable, and refer to the line # in the item being processed. These are the numbers used in error messages. Statement numbers can be more meaningful for cross-references using include files and macros.

**Print Line/Stmt # first on listing line (N) -**

This allows you to select the order of certain items in the listing line. A **Y** causes the line number to be listed first, whereas an **N** causes the hexadecimal values to be listed first. Suit yourself!

**Number of errors on which to abort (100) -**

This option provides a limit on the number of errors **SLR180** will process before aborting the current assembly. A value of 0 will cause this option to be ignored.

**Number of lines per console page (0=no paging) (24) -**

This provides paging for all console output, pausing for operator intervention after N lines. A value of zero causes this option to be ignored.

**Number of bytes (1-60) per line of HEX output (32) -**

This option allows you to specify the number of bytes per line when generating a .HEX file. Certain emulators, PROM programmers, etc are very particular about the format for 'Intel-Standard' HEX format.

**Close & ReOpen file in 2-pass mode (N) -**

This option is used for CP/M 'lookalikes' that don't like the way **SLR180** optimizes disk I/O. Molecular Computer users may need to set this flag when assembling files larger than one extent.

**Require : if label not in column one (Y) -**

This option allows full free form source lines. Be careful if you select N, because mistyped opcodes will be treated as labels, and no error generated.

**Conditionals test only bit 0 (DRI compatibility) (N) -**

This option, when set to N, causes 0 to be treated as false, and anything else as true. If this option is set to Y, then if bit 0 is off, that is false, bit 0 on is true, and all other bits are ignored.

**Suppress IF parameter errors in 1 pass mode (N) -**

This option will disable the reporting of IF operand errors in one-pass mode. IF operands containing forward references can cause different sections of code to be assembled on different passes, which will go undetected in one pass mode.

**Ignore leading space & tab chars in IF <> types (Y) -**

This option controls the significance of spaces and tabs in IF string parameter processing. IFB < > will return FALSE if this option is set to N (M80 compatible, space is not blank), but will return TRUE if set to Y.

**Fill unused space with 0 (N) or FF (Y) in COM (N) -**

This option is used to define the fill byte used in filling uninitialized space when generating a COM file.

**Generate Empty External Chains (M-REL) (N) -**

Normally symbols that are declared external but never referenced are not entered into the REL file. This option forces generation of all externals. Some compilers expect the assembler to strip unneeded externals, while others select library options by simply declaring labels external.

**ASEG Default to 0H instead of 100H (N) -**

The first ASEG encountered defaults to 100H for standard CP/M program generation. This can be changed to 0 for M80 compatability.

**ORG <ABSOLUTE> Yields Offset in Current Space (N) -**

Normally an implied ASEG is performed when ORG is passed an absolute parameter. This option allows that absolute number to be treated as an offset in the current address space.

The next series of prompts allow you to change the extensions looked for or generated on several different files. To change the extension, just type a new one, otherwise type a CR.

**Extension for source File (180) -**

This is the source file being assembled. The default case looks for an extension of 180. You can change it to ASM, MAC, SRC, or whatever you like.

**Extension for relocatable File (REL) -**

This is the extension generated when you are creating a binary output file in relocatable mode.

**Extension for absolute binary File (COM) -**

This is the extension generated when you are creating a binary output file in absolute core-image mode.

**Extension for Intel-Hex format file (HEX) -**

This is the extension generated when you are creating a binary output file in absolute Intel-hex (/H) mode.

**Extension for listing file (LST) -**

This is the extension used when generating a listing file to disk.

**Extension for temporary file ($$1) -**

This is the extension used for the temporary file generated if the intermediate code overflows in one-pass mode.

**Extension for /I file (SUB) -**

This is the extension looked for on the specified indirect command file.

**Default Ext for MACLIB file (MAC) -**

This is the default extension used for filenames given in a MACLIB or INCLUDE statement.

**Leader String to send to printer (1B,51,) -**

This allows you to specify a setup string of up to 8 bytes to be sent to the printer before a listing is started. You can use it to select compressed print, one-pass print, or just blank it out with an FF. Note that this string is sent only if you use the /P option, not a drive type of 'Y'.

**Trailer String to send to printer (1B,4E,) -**

This allows you to undo whatever you did above. Again, this is only done with a /P.

At this point, 180FIG writes the changes the the specified file. That's it!

# APPENDIX E - **ASCII TABLE**

## ASCII CHARACTER SET

The ASCII character set consists of single codes in the range of 0 to 7FH inclusive. This is always nice to have around when you need it.

| BIT 6 / I 5 / T 4 | C / 0 | 0 0 | 0 1 | 0 1 | 0 1 | 1 0 | 1 0 | 1 1 | 1 1 |
|---|---|---|---|---|---|---|---|---|---|
| 3210 | ROW L | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0000 | 0 | NUL | DLE | SP | 0 | @ | P | ` | p |
| 0001 | 1 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 0010 | 2 | STX | DC2 | " | 2 | B | R | b | r |
| 0011 | 3 | ETX | DC3 | # | 3 | C | S | c | s |
| 0100 | 4 | EOT | DC4 | $ | 4 | D | T | d | t |
| 0101 | 5 | ENQ | NAK | % | 5 | E | U | e | u |
| 0110 | 6 | ACK | SYN | & | 6 | F | V | f | v |
| 0111 | 7 | BEL | ETB | ' | 7 | G | W | g | w |
| 1000 | 8 | BS | CAN | ( | 8 | H | X | h | x |
| 1001 | 9 | HT | EM | ) | 9 | I | Y | i | y |
| 1010 | 10 | LF | SUB | * | : | J | Z | j | z |
| 1011 | 11 | VT | ESC | + | ; | K | [ | k | { |
| 1100 | 12 | FF | FS | , | < | L | \ | l | \| |
| 1101 | 13 | CR | GS | - | = | M | ] | m | } |
| 1110 | 14 | SO | RS | . | > | N | ^ | n | ~ |
| 1111 | 15 | SI | US | / | ? | O | _ | o | DEL |

# APPENDIX F - **LIMITED WARRANTY**

**SLR Systems** disclaims any warranty as to this product.

> SELLER MAKES NO WARRANTY EXPRESS OR IMPLIED, AND ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE WHICH EXCEEDS THE FOREGOING WARRANTY IS HEREBY DISCLAIMED BY SELLER AND EXCLUDED FROM ANY AGREEMENT.

Buyer exclusively waives its rights to any consequential damages, loss or expense arising in connection with the use of or the inability to use its goods for any purpose whatsoever.

No warranty shall be applicable to any damages arising out of any act of the Buyer, his employees, agents, patrons or other persons.

The remedies set forth herein are exclusive and the liability of Seller to any contract or sale or anything done in connection therewith, whether in contract, in tort, under any warranty, or otherwise, shall not, except as expressly provided herein, exceed the price of the equipment or part on which said liability is based.

No employee or representative of Seller is authorized to change this warranty in any way or grant any other guarantee or warranty.