5.00.00 THE PL/9 COMPILER

The compiler resides in memory with the editor and may be considered to be an integral part of the editor when used in the interactive mode. The following commands are available whenever the (#) prompt is present:

Invoke the trace/debugger. PL/9 will compile the program without generating a listing, putting the code into memory, then will jump to the trace/debugger (described in its own section). No options are allowed with this command. The trace/debugger is covered in greater detail in the following section.

A

Compile the edit file without any listing, printout or object file. Generally used to perform a quick syntax/typographical error check.

A[:<options>]

Compile the file resident in memory. (A is used mainly for ergonomic compatibility with the MACE assembler.) Options are as follows:

A:C

Display the code generated for each source statement. The code generated for "INCLUDE" files (q.v.) will not be displayed. This option only makes sense when used with the 'T', 'P' or 'L' options.

A:M

Write object code directly into memory. PL/9 will not allow itself, its edit file or any of its tables to be over- written, and complains with the message "CAN'T WRITE TO $MMMM", where MMMM is the address of the attempted write. See the diagram of memory usage, in section three, for information on what areas of memory are used by PL/9 and its tracer.

A:N

Compile with a cross reference listing only. Defaults to terminal but may be directed to the printer (A:N,P) or to an output file (A:N,O).

A:P

Compile with a printer listing. The page number and the date will be printed at the top of each page. If there is a comment (i.e. /* ... */) on the first line of the program then this will be printed at the top of each listing page as a title, otherwise the PL/9 startup banner will be printed. The FLEX TTYSET 'WD' parameter is obeyed. Any lines longer than the defined 'WD' value will be truncated on the listing.

A:T

Compile with a listing on the terminal; no titles or page numbers will be printed. The FLEX TTYSET 'WD' parameter is obeyed. Any lines longer than the defined 'WD' value will be truncated on the listing.

A:L[=<FILENAME>]

Write the compile listing to disc into the named file. Use the Q command to see what default drive and extension will be used; if you don't like them use SETPL9 to change them. As for the (A:P) command titles and page numbers will be printed at the top of each page. This option is provided primarily to produce a text file for print 'spooling' or merging into a word processor text file. The file produced can also be 'P,LIST'ed.

A:O[=<FILENAME>]

Write object code to disc, overwriting any existing file of that name. Use the Q command to see what default drive and extension will be used; if you don't like them then use SETPL9 to change them.

A:<N1>-<N2>

If one of the T, P or L options is in force, the compiler can be requested to generate output for only the specified range of source line numbers. No symbol table will be output in this case. If the -<N2> is omitted then only one line will be generated.

A:$XXXX

When using either the 'M' or 'O' options it is frequently useful to be able to offset the program (for example when the object code is to be put into an EPROM and there is no RAM on the development system at the required address). The offset $XXXX is added to the program counter value (as printed on the object listing).

A:R

This option directs the compiler to use the MC8809 hardware vectors at $FFF2 through $FFFF in lieu of the RAM vectors you defined when using SETPL9. This option only has an effect if you use the following procedure names in your program: RESET, NMI, FIRQ, IRQ, SWI, SWI2 or SWI3. This facility enables you to test your program within the development system using its RAM vectors and then generate a file for use in the target hardware environment by invoking this option. This option only makes sense when used with 'A:O'.

Compile options may be strung together, as in the following examples:

A:P,100-200

Compile to the printer, generating a listing only for lines 100-200.

A:C,T,281

Compile to the terminal, displaying the generated object code for line 281 only.

A:M,$4000

Compile to memory, loading the program at a location offset by $4000 from any origin specified.

A:O,L

Generate a binary file and a listing file, both files having the names given by the Q command.

A:O=1.MYFILE.BIN,L=0.MYFILE.OUT

As above but override the default drive numbers, file names and file extensions.

A:O,R

Generate a binary file, using the default drive number, file name and file extension, but substitute the MC6809 hardware vectors ($FFF2 - $FFFF) for the RAM vectors defined by SETPL9.

NOTE: Any one, or all of the listing options (T, P or L) may be in force at any given time. i.e. it is possible to specify 'A:T,P,L,C' and generate a listing on the terminal, a listing on the printer and a disk file all with the code generated by PL/9 displayed if this is what you require.

CALLING THE COMPILER FROM FLEX

The PL/9 compiler may also be called from FLEX with multiple options specified, as the following examples illustrate:

```
+++PL9,1.FILENAME.EXT<CR>
```

Compile the file specified reporting any errors.

```
+++PL9,FILENAME,T<CR>
```

Compile the file specified using the default drive number and file extension that PL/9 was configured for using the SETPL9 command to the system console obeying TTYSET 'WD'.

```
+++PL9,FILENAME,C,P<CR>
```

Compile the file specified to the system printer obeying TTYSET 'WD' and displaying the generated object code

```
+++PL9,FILENAME,O,R<CR>
```

Compile the file specified to an output file (binary) substituting the MC6809 hardware vectors for the RAM vectors defined by SETPL9. The output file will have the same name as 'FILENAME' but will use the default drive and file extension specified by SETPL9.

```
+++PL9,FILENAME,O=0.OBJECT.BIN,R<CR>
```

As above but override the default drive and extension.

```
+++PL9,FILENAME,C,L,O<CR>
```

Compile the file but direct the output listing, with generated object code displayed, to a disk file using the default drive number and file extension defined by SETPL9. Also produce an object file on disk in the same manner.

```
+++PL9,FILENAME,C,L,O,M<CR>
```

Compile the file as above but also compile the file into memory.

NOTE: Any one, or all of the listing options (T, P or L) may be in force at any given time. i.e. it is possible to specify 'A:T,P,L,C' and generate a listing on the terminal, a listing on the printer and a disk file all with the code generated by PL/9 displayed if this is what you require.

ERROR HANDLING

When PL/9 detects an error it stops compiling and prints a message from the list below, followed by the source line in which the error was found, then by a caret (up arrow) under the position it had reached in that source line. This is usually within one or two character positions of the point at which the error was detected, although some errors may not be detected until the following line (for example when a semicolon is omitted). Once the error has been reported the compiler waits for the user to type a response. A carriage return will cause the editor to be re-entered at the line in which the error was detected, while any other input will cause compilation to resume after the first semicolon following the error. (This means that further errors may be reported as a result of a keyword such as BEGIN being skipped, leaving a mismatched END later on in the program.)

Most of the error messages are self-explanatory, but each is outlined below:

")" EXPECTED

The compiler was expecting a right parenthesis at this point.

"," EXPECTED

The compiler was expecting a comma at this point.

";" EXPECTED

The compiler was expecting a statement delimiter at this point.

"=" EXPECTED

The compiler was expecting an assignment operator at this point.

"BYTE", "INTEGER" OR "REAL" EXPECTED

The syntax so far has led the compiler to expect a BYTE, INTEGER or REAL identifier; for example if a colon is typed instead of a semicolon at the end of a GLOBAL statement.

ILLEGAL DECLARATION

You will get this message if you attempt to use the GLOBAL or DPAGE directives

after the compiler has generated any code.

NO ACTIVE PROCEDURE

A statement has been encountered within a procedure that may only reside outside a procedure.

NOT ALLOWED OUTSIDE A PROCEDURE

A statement has been encountered outside a procedure that may only reside within a procedure.

NUMBER OR CONSTANT EXPECTED

The symbol encountered should have been a number or a constant. For example, it is not permissible to assign a variable to a constant.

ONLY ONE GLOBAL DECLARATION!

There may be only one GLOBAL statement in a program.

READ-ONLY VARIABLE

An attempt has been made to assign a value to a data variable, i.e. one declared by the BYTE, INTEGER or REAL data declaration.

SYMBOL ALREADY EXISTS

The variable or procedure name indicated has already been declared, earlier on in the program.

"THEN" EXPECTED

An IF statement has been encountered that either has a missing THEN or some other error causing the compiler to miss the THEN.

THIS PROCEDURE DOESN'T RETURN A VALUE

An attempt has been made to use a procedure as a function subroutine without a return value having been specified.

TOO MANY BREAKS

Only ten BREAKs may be pending at any one time.

UNDEFINED SYMBOL

The variable or procedure name indicated has not been declared.

WARNING: SYMBOL ALREADY USED FOR ANOTHER VARIABLE TYPE

Usually caused by trying to give a CONSTANT the same name you have given a procedure.

WRONG PARAMETER TYPE

In a procedure call, the parameters supplied do not match those declared with the PROCEDURE statement. This message is also used to report errors when using pointers incorrectly.

WRONG VARIABLE TYPE

You will get this error message if you put a procedure name on the left side of an assignment expression, if JUMP or CALL attempt to use anything other than an integer or if you attempt to use a variable that has not been declared as a pointer as a pointer.

OTHER ERRORS

It is worth noting that there are some errors that may not be detected until considerably after they occur. Errors such as a missing END or a missing semicolon are unlikely to be detected immediately. A missing " at the end of a string can cause the whole of the rest of the program to be treated as a very long string! In cases such as these there is no substitute for human ingenuity - you shouldn't expect PL/9 to do ALL your thinking for you! If a visual inspection fails to locate the error, try removing instructions successively back up the program until the problem is located.

It is also worth noting that the compiler will not object to you giving a CONSTANT one of the names reserved for keywords. For example it will readily accept

```
CONSTANT RETURN=$0D;
```

knowing full well that 'RETURN' is a reserved word. Generally speaking the expressions that will use a constant will use it in such a way that the compiler will not confuse it with a keyword. Therefore this abuse of keyword names is not considered to be an error to the compiler. If you decide to use keywords as constants, IN SPITE OF OUR WARNINGS NOT TO, you should be sure of what you are doing first!