6.00.00 THE PL/9 TRACE/DEBUG FACILITY

The PL/9 symbolic trace/debugger allows the user to single-step, breakpoint or run his program and to examine variables as the program proceeds. The principle of operation is that in trace mode PL/9 inserts calls to the tracer and information relating to the program into the object code at every line of source enabling the debugger to retain control over the running program. The penalty paid for this facility is that the program runs somewhat slower, to an extent dependant partly upon the program itself but mainly upon the number of trace/debug facilities used. The trace instructions are not inserted into the code generated for INCLUDEd procedures; these will run at full speed since it is assumed that they will have been previously debugged.

The tracer for PL/9 is a separate file, called PL9_TD.CMD and PL/9 expects to find it on the system drive (usually drive 0). If it is not present then you will not be able to trace programs. It loads into the FLEX command area between $C100 and $C6FF, so please ensure that the program you are tracing does not make any reference to this area. It is quite permissible for the program to have an ORIGIN within this area as ORIGIN statements are ignored by the tracer. Refer to the PL/9 memory map in section 3.01.00 for further details.

W A R N I N G !

The ENDPROC on the last (main) procedure should not be present when using the tracer. The tracer 'senses' the JUMP to FLEX that is put at the end of the main procedure as meaning that the program has ended. If the ENDPROC is used this jump will be replaced by an RTS instruction which will cause the tracer to crash when it reaches the end of the program. Alternatively you may leave the ENDPROC in and place the statement 'JUMP $0003;' just before the ENDPROC while you are tracing a program. This will cause the tracer to re-enter PL/9 via its warm start address when the program ends.

C A U T I O N !

Any procedure that 'polls' the system console keyboard via a routine similar to 'GETKEY' (see the IOSUBS library descriptions) will not work properly when using the tracer. This is due to the fact that the tracer is also 'polling' the keyboard looking for a CONTROL C.

N O T E

The way the tracer works will often cause it NOT to print the last line in a program and/or not print the variables associated with the last line in the procedure when single stepping the program. This 'foible' of the tracer can be avoided by simply placing the statement 'ACCB=ACCB;' below the last source line in the program. This statement may be left in your program if desired as the compiler will not generate any code for it.

USING THE TRACER

To invoke the tracer, load the file as usual and check (using the A command) that there are no errors, then type T. PL/9 will compile the file, putting the object code at the top of available memory, ignoring any ORIGIN statements. It will then indicate how much space is available between the top of the symbol table and the start of the program (this is the area in which global and local variables are stored). The tracer will then be loaded from the system drive, then a startup banner will be printed. The first executable line (the last procedure declaration in the program) will be displayed and the tracer will stop and and wait for instructions. Note that there are no options associated with the T command.

```
    #T

    XXXXX BYTES AVAILABLE FOR STACK

0182 PROCEDURE MAIN;            /*  THIS IS THE MAIN PROGRAM  */
    &
```

The debug command mode is indicated by the editor prompt changing from # to &.

M O D E C O N T R O L C O M M A N D S

<ESCAPE>

Typing <ESCAPE> at the start of the line will return control to the editor.

E

EDITOR. This command also causes a return to the editor.

X

EXIT to the disk operating system.

M

MONITOR. Exit to the ROM system monitor.

S O U R C E F I L E R E L A T E D C O M M A N D S

<CR>

Display the current line i.e. the source line about to be executed. The entire line is printed, complete with any comments.

<NUMBER>P<TARGET>

Print part of the source file. This command is the same as the editor P command and has no effect on the execution of the program; it is included as an aid to the programmer deciding where to put breakpoints etc.

TRACER CONTROL COMMANDS

G

GO. Run the program, continuing until:

(1) a breakpoint is encountered.
(2) a control C is typed.
(3) the program ends.

S

SINGLE-STEP the program. Execution will stop at each source line, the line will be displayed and the tracer will wait for a key to be pressed before continuing. A space will cause the line to be executed, while anything else will cause a return to tracer command level.

R<NUMBER>

RUN <NUMBER> lines of the program. Execution can be terminated as for G above, or will stop after the specified number of lines has been executed.

T<NUMBER>

TRACE <NUMBER> lines of the program. Each source line will be printed before it is executed; otherwise as for R (which does not display the source lines). If <NUMBER> is zero, the result will be as if G had been typed, except that lines with breakpoints will not cause a break but will instead cause a printout of the source line, with execution then continuing automatically.

W<NUMBER>

WAIT at each line for a time dependant on the value of <NUMBER>. This allows a program to be slowed down when in TRACE so that its effects can more clearly be seen.

N<N1>-<N2>[,<N3>-<N4>....]

NO TRACE. The tracer will not stop at any line in any of the ranges specified. This command is very useful in that it can be made to skip delay loops, initialization sequences and procedures that have already been debugged.

Q

QUIT. Restart the program without re-compiling it.

B R E A K P O I N T S

B

Clear all breakpoints.

B<N1>,<N2>...

Set breakpoints at the specified line(s). Existing breakpoints are kept active; the only way to remove them is to clear all breakpoints using B by itself.

V A R I A B L E S

<VARIABLE LIST>

Print the values of specified program variables. Simple variables and vector elements (with numeric indices, not other variables) can be specified, and may be separated by either a semicolon (print on the same line) or a comma (start a new line). Examples:

Print all three values on one line:

```
&?CHAR;COUNT;POS
```

Print the first three values on one line and the fourth on the next line:

```
&?BUFFER;BUFFER(1);BUFFER(2),COUNT
```

Print each value on a separate line:

```
&?CHAR1,CHAR2,CHAR3
```

If a variable is specified that either does not exist or is not known to the current procedure then no value is printed.

D<VARIABLE LIST>

Print variable values whenever a source line is displayed. The values are specified as for (?) above and are printed before the source line. To prevent variable printing, use the command D with no list. The variable list is "remembered" from one compilation to the next.