2.00.00 PRODUCT OVERVIEW

PL/9 is a complete co-resident Editor/Compiler/Trace Debugger for the 6809.

PL/9 features an editor, identical to the one in 'MACE', that is very quick to learn and easy to use. It loads and saves files, finds and changes strings, appends comments, inserts and deletes lines, prints selected lines on the terminal or printer, passes commands to 'FLEX' and calls the co-resident single pass Compiler or Debugger.

PL/9 is a TRUE COMPILER that produces PURE 6809 machine code. PL/9 does not require a run time interpreter, with its associated loss of speed and license costs (as do most BASICs and PASCALs) ...NOR... does PL/9 impose any license fee or restrictions in regard to its MATH module (as does TSC's Native Code Pascal). The code PL/9 produces belongs to you, and you alone...a valuable consideration if you are writing programs to sell or integrate into systems.

PL/9's Trace Debugger allows you to single step or breakpoint a PL/9 program a source line at a time examining variables as you go.

PL/9 is a structured language loosely based upon the control structures found in BASIC, PASCAL, PL/M and 'C' but omitting the exotic data types and type checking. PL/9 has been specifically developed for dedicated control applications in a microcomputer environment. The language is designed to be a step up from assembly language. It retains most of the flexibility and speed of the latter but makes programs, particularly those with structured control arguments, shorter and more readable. PL/9 is largely self documenting owing to its ability to support variable names of up to 127 characters in length.

Functions not supported directly by the PL/9 compiler, such as disk drivers or I/O routines can easily be 'INCLUDEd' in PL/9 programs thus allowing the user to generate new KEY WORDS to suit his own particular requirements; a number of such functions (library modules) are included with the PL/9 compiler.

PL/9 makes extensive use of the STACK for temporary variable storage thus making all PL/9 modules <u>position independent and ROMable</u>. Variables may also be located at fixed positions in memory to facilitate interface with hardware, programs written in other languages such as PASCAL or BASIC, and programs written by several programmers, each with his own allocated variable storage area for parameter passing/scratch.

PL/9 recognizes three distinct data sizes: BYTE (8-bit), INTEGER (16-bit), and REAL (32-bit...8-bit exponent and 24-bit mantissa) floating point accurate to six or seven decimal digits. BYTE and INTEGER values may be signed (twos complement) or unsigned (ones complement). The unsigned BYTE and INTEGER variables have been provided to facilitate the bitwise operations and comparisons that are common in digital I/O. The data types available are:

1. BYTE (signed) in the range of -128 to +127.

2. BYTE (unsigned) in the range of 0 to +255.

3. INTEGER (signed) in the range of -32768 to +32767.

4. INTEGER (unsigned) in the range of 0 to +65,535.

5. REAL (floating point) in the range of +/-1 E-38 to +/-1 E+38.

Data assignment operators are: 'GLOBAL' (are allocated permanent space on the stack and known to all procedures, 'AT' (a fixed absolute address), and 'CONSTANT' (allows you to equate commonly used hex or decimal values to improve readability). An implicit 'LOCAL' assignment operator allows temporary (scratch-pad) variables to be defined for and known by one or more procedures (these variables are not allocated permanent space on the stack).

'ORIGIN' and 'STACK' statements allow the programmer to specify where in memory his object code (on a procedure by procedure basis) and global variables are to be located.

'DPAGE' statements allow the programmer to pre-set the MC6809 direct page register in order to improve the code efficiency of accessing AT variables.

Mathematical expressions:	(+), (-), (*), (/), negation (-), modulus (\).
Bitwise operators:	(AND), (OR), (EOR), (XOR), (NOT), (SHIFT), (SWAP).
Logical operators:	(.AND), (.OR), (.EOR), (.XOR).
Relational operators:	(=), (<>), (>=), (<=), (>), (<).
Functions:	(SQR), (INT), (INTEGER), (BYTE), (FLOAT).

Address pointers are supported in the forms: '.VARIABLE', '.VARIABLE(COUNT)', '.VARIABLE(COUNT*2)', etc. ('&' may be used in lieu of '.' if desired)

Control statements are: 'IF...THEN...ELSE', 'IF...CASE1...CASE2.(etc)...ELSE', 'BEGIN...END', 'WHILE', 'REPEAT...UNTIL', 'REPEAT...FOREVER', 'CALL', & 'JUMP'.

Control statement terminators supported are: 'RETURN', 'RETURN <condition>', 'BREAK' and 'GOTO'

The 'ASMPROC' and 'GEN' statements (or special files produced by 'MACE') may be used to insert machine code inside of PL/9 procedures to obtain special functions, such as indirect addressing e.g. (JSR [D3E5]) would be coded as: 'GEN $AD,$9F,$D3,$E5;'

Some of the more powerful aspects of PL/9 include direct access to accumulators 'A', 'B', 'D', 'X', 'CCR' (condition code register) and 'STACK'. These pseudo variables may be contained as part of an evaluation or assignment construction and greatly improve and simplify communication with external assembly language procedures.

PL/9 also possesses the ability to intercept the system's RAM (or the MC6809 hardware vectors at $FFF2 - $FFFF) vectors for: SWI, SWI1, SWI2, NMI, FIRQ, and IRQ INTERRUPTS. The MC6809 RESET vector at $FFFE/F can also be intercepted. Thus allowing PL/9 procedures to start the entire system from power-up, service interrupts, etc., WITHOUT RECOURSE TO ASSEMBLY LANGUAGE PROGRAMS.

PL/9 LIBRARY MODULES

Since PL/9 has been designed to produce code for TARGET hardware we have made no assumptions about the I/O environment or memory map of the system the code is to run in. By default this means that we cannot include I/O facilities in the compiler itself. For example if we built a PRINT statement into the compiler we would also have to build in an OUTCHAR routine. The OUTCHAR routine would HAVE to be configured for a specific hardware environment which would make the program 100% dependant on being run in a particular hardware configuration.

This is clearly undesirable if you are writing programs for a variety of hardware environments. To cater for this programming requirement PL/9 is supplied with a set of I/O libraries configured for the FLEX environment. When these library modules are INCLUDEd in the users programs the routines then APPEAR TO BE AN INTEGRAL PART OF THE LANGUAGE. The beauty of this arrangement is that you can have a library module, say IOSUBS.LIB, configured for the FLEX environment and another library module, say TARGETIO.LIB, with procedures of identical names and functions, configured to match the target hardware environment. Thus the users program may be tested within the development system by simply substituting IOSUBS.LIB whenever the program is compiled. Once the program is debugged you revert to TARGETIO.LIB and compile the program for the target hardware environment with the knowledge that 95% of the program is debugged. This 'dual library' approach can also be extended to cater for differences in memory maps between the development system environment and the target hardware environment.

The libraries supplied with PL/9 are as follows:

IOSUBS.LIB	This library module has a series of PL/9 procedures to simplify communication via the system console. Functions provided by this module include: (MONITOR), (GETCHAR), (GETCHAR_NOECHO), (GETKEY), (CONVERT_LC), (GET_UC), (GET_UC_NOECHO), (PUTCHAR), (PRINTINT), (INPUT text into a buffer), (CRLF), (PRINT string) and (SPACE).
TERMSUBS.LIB	This library module has a series of PL/9 procedures to handle 'intelligent' terminals. As supplied the procedures are configured for a SOROC IQ-120/LEAR SIEGLER ADM3 or ADM5 but they can be quite easily modified for other terminals. (NULLS), (ERASE_EOL), (ERASE_EOP), (CURSOR_COL/ROW), (HOME), (HOME_N_CLEAR), (ATTR_ON) and (ATTR_OFF);
HEXIO.LIB	This library module has a series of PL/9 procedures to simplify console I/O with hexadecimal numbers: (GET_HEX_NIBBLE), (GET_HEX_BYTE), (GET_HEX_ADDRESS), (PUT_HEX_NIBBLE), (PUT_HEX_BYTE) and (PUT_HEX_ADDRESS).
BITIO.LIB	This module provides two routines for simulating bit oriented I/O via the system console: (BITSIN) and (BITSOUT). It also includes three bit oriented input routines: (BITIN), (BITZ8IN) and (BITZ16IN), and three bit oriented output routines: (BITOUT), (BITZ8OUT), and (BITZ16OUT). These last six routines greatly simplify bit oriented I/O as they completely remove the requirement to perform bitwise AND/OR operations.
HARDIO.LIB	This module contains four routines that BASIC programmers should recognize: (PEEK), (DPEEK), (POKE) and (DPOKE). These functions greatly simplify working with absolute memory addresses being referenced by a pointer.

STRSUBS.LIB	This library module has a series of PL/9 procedures to simplify string handling. Functions provided by this module include: (STRLEN), (STRCOPY), (STRCAT), (STRCMP) and (STRPOS).
BASTRING.LIB	This library contains the familiar string manipulators found in BASIC: (LEFT), (MID) and (RIGHT).
FLEX.LIB	This library module has a series of 'ASMPROC' (pseudo assembly language procedures) to simplify the interface with the 'FLEX' disk operating system. Functions provided by this module include: (FLEX), (GET_FILENAME), (SET_EXTENSION), (REPORT_ERROR), (OPEN_FOR_READ), (READ), (OPEN_FOR_WRITE), (WRITE), (SET_BINARY), (CLOSE_FILE), (RENAME_FILE), (DELETE_FILE), (READ_SECTOR) and (WRITE_SECTOR).
SCIPACK.LIB	This library module contains various scientific functions to an accuracy of at least 5 decimal digits. Functions provided include: (LN natural log), (LOG base 10 log), (ALOG), (XtoY), (SIN), (COS), (TAN) and (ATN).
REALCON.LIB	This library module comprises two floating point conversion routines: 'BINARY' (converts an ASCII string into a REAL number) and 'ASCII' (converts a REAL number into an ASCII string).
REALIO.LIB	This library module contains two routines that facilitate console I/O with REAL numbers: (FINPUT) and (FPRINT).
NUMCON.LIB	This library module contains four routines that facilitate console I/O with INTEGER numbers: (BINTODEC), (PRDEC), (PRNUM) and (GETNUM).
SORT.LIB	This routine contains the 'primitive' required to implement sorting programs. It is similar to the sort function provided in most'C' compiler libraries: (SHELLSORT).

FLOATING POINT MATHS MODULE

In large multi-module systems or where several programmers may be involved in the development of a large software package it is often desirable to write, compile, and test program modules (particularly low-level modules) and then insert them in PROM, or at least leave them alone, for the remainder of the software development.

Under normal circumstances PL/9 will generate a MATHS MODULE for every PL/9 procedure file (or groups of procedure files compiled by 'spooling'). Where a large number of PL/9 procedure files are involved 'spooling' can prove to be very time consuming.

A unique capability of PL/9 is the ability to install the MATHS module in a fixed location in memory to be shared by all PL/9 procedures. The relocatable MATHS module produced by PL/9 is just over 1K in size. All entries into the module are made via a jump table to ensure that future additions to the module will be compatible with the current version.

The routines used in the MATHS module supports the 'REAL' numbers in the range of +/-1 E37 to +/- 1 E-38 with 6 or 7 significant (decimal) digits. A summary of the speeds (2 MHz 6809) are as follows:

OPERATION	REALS	INTEGERS
ADD	590 uSec	10 uSec
SUBTRACT	610 uSec	10 uSec
MULTIPLY	450 uSec	80 uSec
DIVIDE	1.31 mSec	450 uSec
SQ ROOT	2.91 mSec	---
SINE	8.41 mSec	---
COSINE	7.91 mSec	---
NATURAL LOG	7.41 mSec	---
X TO Y POWER	23.41 mSec	---

The REALCON.LIB conversion routines allows input, from external sources, in a wide range of formats, e.g. 1234, .052, .531, 3.6E12 -8.4316E-4, etc.

If the answer is >= 1,000,000 or < 0.01 the response will be in scientific notation. If the answer falls between these limits the response will be in decimal with all trailing zeros truncated (100.0000 will be 100).

2.00.01 BENCHMARK PERFORMANCE FIGURES

The figures for products other than PL/9 have been taken from published figures either in 'BYTE' magazine or '68 Micro Journal'. We assume no liability for their accuracy nor do we assume any liability for any omissions. To the best of our knowledge all other products were tested in a 2 MHz system.

Even though PL/9 places well in the benchmark tests these test results should not be used as the sole criteria to compare our product with the rest of the field. For example several of the other products offer number processing capabilities far in excess of PL/9's capabilities.

All tests of PL/9 have been conducted in the following hardware environment:

A Windrush 6809 Phoenix system with the 6809 processor running at 2.0 MHz, a Windrush 5/8, SD/DD, SS/DS disk controller, 5" DS, DD, 80 track, 3 Ms step YE-DATA (Qume) disk drives, I/O stretch (MRDY) enabled, 56K of STATIC RAM with a memory mapped video display emulating a Soroc IQ 120 with an effective baud rate in excess of 38K.

LANGUAGE	SIEVE OF ERATOSTHENES	PRIME NUMBER GENERATOR
ASSEMBLER	5.10	not available
IMS PASCAL (NATIVE)	8.78	not available
OS9 PASCAL (NATIVE)	not available	54
PL/9	9.4	56
INTROL 'C'	11.0	not available
TSC PASCAL (UNIFLEX)	34.0	not available
TSC PASCAL (FLEX)	54.0	59
IMS PASCAL (P-CODE)	105.00	not available
OMEGASOFT PASCAL (FLEX)	not available	72
BASIC-09 (INTERPRETER)	238.00	not available
DUGGERS 'C'	not available	74
OS9 PASCAL (P-CODE)	not available	112
DYNASOFT PASCAL (P-CODE)	309.00	not available
LUCIDATA PASCAL (P-CODE)	735.00	194
TSC XBASIC (UNIFLEX)	810.00	not available
TSC XBASIC (FLEX)	840.00	not available