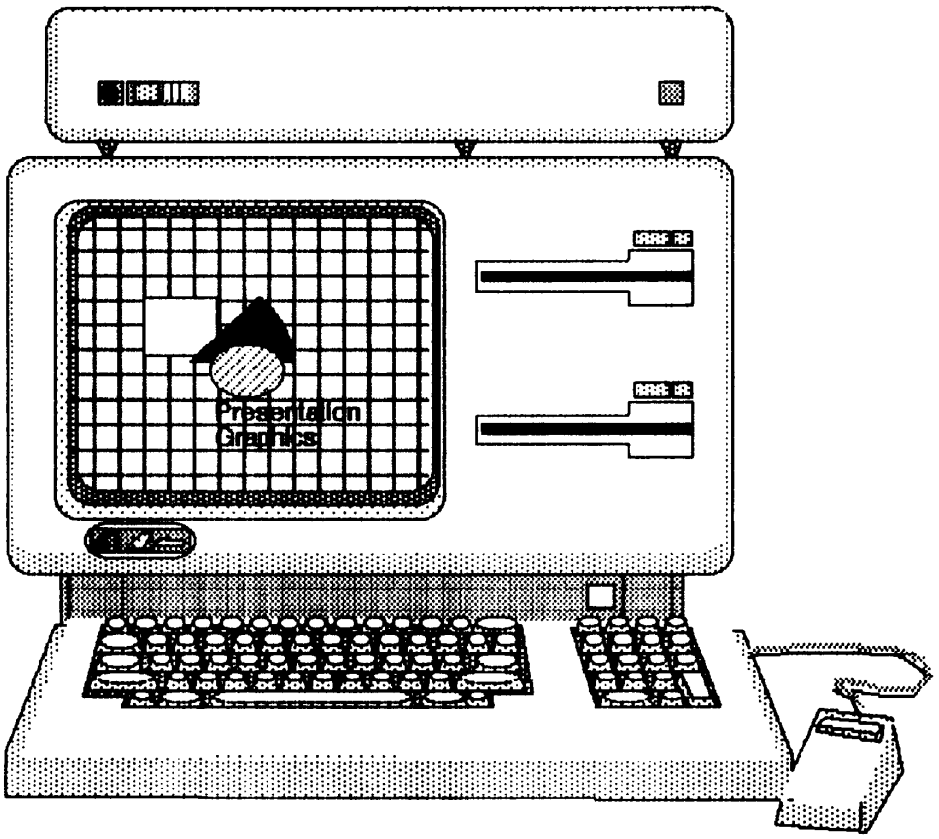
Apple Lisa Computer
Technical Information

Apple Lisa Computer: Pascal Review (DTC 1988)

Lisa Computer: 1983 - 1985

Apple Lisa Personal Computer
1983 to 1985

Lisa Pascal Review by DTC



David T. Craig - 736 Edgewater, Wichita, Kansas 67230 - (316) 733-0914

"LisaPascalReviewDTCOct88 0.PICT" 246 KB 1999-02-01 dpi: 360h x 363v pix: 2539h x 3222v

A Review of Apple's Lisa Pascal

by
David Craig
736 Edgewater, Wichita KS 67230

(October 9, 1988)

INTRODUCTION

In January 1983 Apple Computer Inc. announced their new Lisa computer. Many computer users saw the Lisa as a revolutionary machine, especially for a microcomputer in 1983. The Lisa based its user interface upon the work that Xerox PARC had done for its SmallTalk language and the Xerox Star computer. The Lisa user interface used windows, pull-down menus, icons, and the mouse. Most Lisa owners were very familiar with the Lisa Office System, the program which manages all the graphical user interface issues, but few knew of the WorkShop development environment and Lisa Pascal which made all this software possible.

The Lisa WorkShop is a command line shell similar in appearance to Apple's older Apple // and /// UCSD-like Pascal shells. The WorkShop provides access to various development tools of which the Pascal Compiler is the center piece. Originally begun in early 1980 as a port from Apple // Pascal, Lisa Pascal evolved into a very powerful compiler. This compiler generated most of the application and system code which made the Lisa work. To make the Lisa software development cycle easier Apple decided to write most of this software in Pascal. This coding effort resulted in nearly one million lines of Pascal code for the Lisa Desktop Libraries, a set of about 100 units which establish the protocols to follow across all the different Lisa applications. Even the Lisa Operating System is written in about 90,000 lines of Pascal with about 10,000 lines in 68000 assembly code for the low-level I/O and time critical portions.

UNIQUE FEATURES

Lisa Pascal is a superset of UCSD Pascal and initially resembled Apple /// Pascal 2.0. Up to the introduction of the Lisa Apple's Pascal compilers had all generated P-code, but the Lisa Pascal Compiler, in conjunction with the Lisa Code Generator, produces native 68000 object code. When the Macintosh was introduced in January 1984 Lisa Pascal became much more powerful since the Lisa was the only initial development environment for the Macintosh. Overall, Lisa Pascal implements the following unique features (occasional 68000 code is used to illustrate the actual implementation):

- **Direct compilation of Apple // and /// source code**
  Since the Lisa Pascal Compiler began life as an Apple // compiler it can compile source code from either the // or ///. This source code must not contain any // or /// device specific portions since the Lisa architecture differs greatly from either machine. Once a // or /// program is compiled the Lisa screen resembles the standard 24 line by 80 column screen.

- **Long Integer type (LONGINT)**
  This type, implemented as a signed four byte integer, allows manipulation of integers in the range −2,147,483,648 to 2,147,483,647. To convert a regular two byte INTEGER to a LONGINT the new ORD4 conversion operator is used. The HIWRD operator extracts the high ordered word (bits 16-31) from a LONGINT, and LOWRD extracts the low

ordered word (bits 0-15). These last two operators are implemented by the Compiler as follows (I is an INTEGER and L is a LONGINT):

```
I := LOWRD(L);  MOVE.W  D0,D5   { D0 = L, D5 = I }
I := HIWRD(L);  SWAP    D0
                MOVE.W  D0,D5
```

- **Detailed compilation and code generation statistics**
  The Lisa Pascal Compiler generates I-code (intermediate code) which is actually only standard UCSD Pascal P-code. The Code Generator takes the compiler's I-code file and produces 68000 object code. For a listing of the generated 68000 code the $ASM compiler directive is used. This produces a listing with the Pascal source statements followed by their object and assembly code. The Compiler and Code Generator display a fair amount of compilation statistics. A sample follows:

```
Lisa Pascal Compiler V3.76 (05-Apr-85)          14:00:51 07-Oct-88
(c)1981 SVS, Inc.  (c)1983, 1984 Apple Computer, Inc.

Input file - [.TEXT] LisaPascalTester
List file - [.TEXT]
Output file - [LisaPascalTester] [.OBJ]

[242091 words]    TEST_INL
[242073 words]    TEST_LEA
[242039 words]    TEST_BIT
[242000 words]    TEST_TYP
[241897 words]       ZERO_REC
[241903 words]    TEST_LAR
[242067 words] LP_TEST

Elapsed time: 1.814 seconds.
Compilation complete - no errors found.  113 lines.

Lisa Pascal MC68000 Code Generator V3.65 (20-Mar-85) 14:00:58 07-Oct-88
(c)1981 SVS, Inc.  (c)1983, 1984 Apple Computer, Inc.

Input file - $I+
Input file - [.I] LisaPascalTester
Output file - [LisaPascalTester] [.OBJ] LisaPascalTester

TEST_INL - TEST_INL    Code size =     32
TEST_LEA - TEST_LEA    Code size =    178
TEST_BIT - TEST_BIT    Code size =    116
TEST_TYP - TEST_TYP    Code size =    136
ZERO_REC - $2000000    Code size =     68
TEST_LAR - TEST_LAR    Code size =     46
LP_TEST  - LP_TEST     Code size =    676

Elapsed time: 2.550 seconds.
Total code size = 1252
```

- **Bit manipulation**
  The bit routines manipulate the 32 bits within LONGINT values. These functions perform bit testing, setting, clearing, rotating, and shifting. The names, operation,

sample Pascal code, and generated assembly code follows (b is BOOLEAN, L and L2 are LONGINT):

```
BCLR     Clear a bit       BCLR(L,0);            BCLR   #$0000,D0
BSET     Set a bit         BSET(L,0);            BSET   #$0000,D0
BTST     Test if bit = 1   b := BTST(L,1);       BTST   #$0001,D0
BITROTR  Rotate right      L2 := BITROTR(L,1);   ROR.L  #$1,D0
BITROTL  Rotate left       L2 := BITROTL(L,1);   ROL.L  #$1,D0
BITSR    Shift right       L2 := BITSR(L,1);     LSR.L  #$1,D0
BITSL    Shift left        L2 := BITSL(L,1);     LSL.L  #$1,D0
BITNOT   Logical NOT       L2 := BITNOT(L);      NOT.L  D0
BITXOR   Logical XOR       L2 := BITXOR(L,1);    MOVEQ  #$01,D1 EOR.L D1,D0
BITOR    Logical OR        L2 := BITOR(L,1);     OR.L   #$00000001,D0
BITAND   Logical AND       L2 := BITAND(L,1);    AND.L  #$00000001,D0
```

- **Loop exit control**

The LEAVE keyword causes an exit of the currently executing loop (FOR-NEXT, WHILE-DO, REPEAT-UNTIL). A sample follows which prints the integers from 0 to 6:

```
PROCEDURE test_LEAVE;
VAR i : INTEGER;
BEGIN
  FOR i := 0 TO 9 DO BEGIN
    WRITELN('i = ',i:1);
    IF i = 6 THEN LEAVE;
  END;
END;
```

- **Floating point calculations based upon SANE**

All floating point values are manipulated by the Standard Apple Numeric Environment (SANE), a very fast and accurate floating point engine which is based upon IEEE-standard numerics. Four new number types are available:

```
SINGLE      4 bytes (same as REAL)          7 place accuracy
DOUBLE      8 bytes                        14 place accuracy
EXTENDED   10 bytes                        19 place accuracy
COMP       10 bytes (treated as an integer)
```

Several SANE constants are available:

```
PI               Pi (3.141592...)
MINNORMEXTENDED  Minimum EXTENDED value
MINNORMDOUBLE    Minimum DOUBLE value
MINNORMREAL      Minimum REAL/SINGLE value
MAXCOMP          Maximum COMP value
INF              Infinity (E.g., r := 1/0 --> r = INF)
```

- **Object-oriented programming**

The Lisa Pascal Compiler supports another Pascal-like object-oriented language called Clascal. This language was originally developed as the foundation language for Apple's ToolKit libraries. Several keywords are present to define an object and its methods:

SUBCLASS METHODS CLASS THISCLASS

- **Relaxed order of label, type, constant, and variable declarations**
  The order of LABEL, TYPE, CONSTANT, and VAR declarations is unimportant as long as no forward-like relationships exist.

- **Date and time constants**
  These predefined constant strings contain the compilation date and time as follows:

```
COMPDATE   compilation date (E.g.,: "07-Oct-88")
COMPTIME   compilation time (E.g.,: "14:01:35")
```

- **Type coercion**
  Types with the same physical size allow direct assignment between their variables as seen below:

```
TYPE t_R = REAL;  t_L = LONGINT;
VAR  R : t_R ;    L : t_L ;

L := t_L (R); { t_R and t_L are 4 bytes each }
```

- **Large function results**
  Functions can return variables whose size is greater than four bytes. An example follows:

```
TYPE t_rect = RECORD { 8 bytes in size }
                left, right, top, bottom : INTEGER
              END;

FUNCTION zero_rect (the_rect : t_rect) : t_rect;
  BEGIN
    WITH the_rect DO
      BEGIN
        left := 0; right := 0; top := 0; bottom := 0;
      END;
    zero_rect := the_rect;
  END;
```

- **Procedural/Functional parameters**
  Procedure or function parameters can themselves be procedures or function calls. For example, the following routine from the Lisa's MathLib Unit implements a generic sorting routine where the routines Sorted and Swap are defined by the user:

```
PROCEDURE Math_Sort(first, last: INTEGER;
                    FUNCTION  Sorted(i, j: INTEGER): BOOLEAN;
                    PROCEDURE Swap  (i, j: INTEGER);
                    VAR error : BOOLEAN);
```

- **Object code inclusion**
  Assembler object can can be placed directly within the Pascal source code by using the INLINE keyword. The format is

```
PROCEDURE/FUNCTION name (parms); INLINE obj_code_words;
```

  This stores the obj_code_words values as is into a program. Generally this feature is never used in Lisa programs, but is used by Macintosh software developers as a quick

method of accessing the Macintosh ToolBox through the Trap Dispatcher. The following assembly listing describes exactly the use of INLINE and the generated object code:

```
3       3 -- A   PROCEDURE test_INLINE;
4       4 --
5       5 -- B     PROCEDURE xyz(k: INTEGER); INLINE $ABCD,$1234,$5678;
6       6 --
7       7 0- A     BEGIN
8       8 --         xyz(255);
        000008 3F3C 00FF            MOVE.W  #$00FF,-(A7)
        00000C ABCD                 TRAP    $ABCD       ; Trap  oall
        00000E 1234                 .WORD   $1234
        000010 5678                 .WORD   $5678
        000014 4E75                 RTS
9       9 -0 A     END;
```

- **Access to QuickDraw**
  The Lisa and Macintosh user interfaces are made possible by the QuickDraw unit. This unit implements several very powerful and fast bitmap graphic operations. For example, QuickDraw can display 4,000 characters per second, 800 lines per second, and 160 large solid rectangles per second.

CONCLUSION

In 1985 Apple Computer stopped producing the Lisa and ended all support for it. The Lisa WorkShop and its Pascal Compiler were still supported since many Macintosh developers were using it to produce Macintosh software. Since the Lisa development environment proved over the years to be quite powerful it was ported over to the Macintosh where it is currently known as the Macintosh Programmer's WorkShop (MPW). MPW Pascal is a direct descendant of Lisa Pascal 3.1.

Overall, I have found the Lisa WorkShop and the Lisa Pascal Compiler to be an excellent implementation of the UCSD Pascal development environment which I will continue to use until my Lisa breaks.

<<< That's All, Folks ... >>>