INSTALLATION GUIDE

for

UO-LISP Version 1.5b

on

Z80 Based CPU, TRS-80 Model's I and III
with TRSDOS or TRSDOS like Operating Systems

Distributed by
Northwest Computer Algorithms
P. O. Box 90995
Long Beach, CA 90809
(213) 426-1893

QUICK CHECK INSTALLATION PROCEDURE

If you are unable to read any of the files on these disks, please return them to Northwest Computer Algorithms and we will gladly supply you with new ones. If you have any problems with the system, please let us know as soon as possible. Your comments about the system will be greatly appreciated.

Installing UOLISP (It is assumed that you are familiar with the operation of TRSDOS and the use of the COPY and BACKUP commands.) Please note that the programming environment is called UO-LISP while the actual name of the command file is LISP48/CMD.

1. Please make a backup copy of the diskettes as soon as possible. The **MASTER PASSWORD is UOLISP**. The disks are shipped in Model I format and if you are running a Model III you will want to convert the disk to Model III format. Please consult your manuals for the correct procedure to convert the formats.

2. We suggest that you start by creating a LISP system disk with the following files on it:

    LISP48/CMD, COMP, RLISP, DEMO/RED

This collection of routines will enable you to run the demos mentioned below. As you become more experienced with the system you will want to include more of the packages in your LISP environment. Also if you can before you run the demos listed below print out the DEMO/RED file to look at
during the demo of RLISP.

3. To verify that you have a working copy of the system, mount the disk you created in step 2 above in one of the drives and enter the following. Assume that the disk was mounted in drive A.

```
DOS READY

LISP48
UOLISP V1.5B <date created>

*
```

The time between entering LISP48 and when the asterisk (*) prompt appears should be less than 10 seconds. If some message other than the * prompt appears, something is wrong. You should write down any output that appears on your console and return this and the disk to Northwest Computer Algorithms or call immediately.

Enter the following program. You can exit to the TRSDOS command level by typing (QUIT) at the beginning of any input line.

```
(DE SUPERREVERSE (A)
  (COND ((ATOM A) A)
        (T (CONS (SUPERREVERSE (CDR A))
                 (SUPERREVERSE (CAR A)) ))))
```

Make sure you get the number of parentheses correct. UOLISP should respond after you type a <RETURN> on the last like almost immediately with:

```
SUPERREVERSE
```

and another prompt. If not, you are probably missing a parentheses somewhere. If you have gotten this far, then type the following:

```
(SUPERREVERSE '((A . B) . (C . D)))
```

UOLISP will respond very quickly with:

```
((D . C) B . A)
```

Now run the demo program written in RLISP in file DEMO/RED. If you did not create a print out of the file DEMO/RED use <shift-@> to control the scrolling of the demo.

Enter:

```
(FLOAD "COMP ")
```

UO-LISP will load the compiler and respond with:

```
NIL
```

Turn on the compilation flag by entering:

```
(SETQ !*COMP T)
```

UO-LISP will respond with:

```
T
```

Now load the RLISP language by entering:

```
(FLOAD "RLISP ")
```

UO-LISP will respond with:

```
NIL
```

To start RLISP enter:

```
(BEGIN)
```

UO-LISP will respond with:

```
RLISP. V1.5, <dated created>
```

You are now in RLISP and the syntax will be different than LISP. Now enter:

```
IN "DEMO/RED ";
```

RLISP will respond by reading and executing the DEMO/RED demo file. If you made it this far without typing errors the demo should run to completion. If you did not create a print out of the DEMO/RED file use <shift-@> to control the demo scrolling.

```
demo runs... ... ...
```

Now to go back to LISP enter:

```
LISP;
```

UO-LISP will respond with:

```
"Entering Lisp"
```

Now to exit LISP enter:

```
(QUIT)
```

Control is now returned to the TRSDOS command control program.

4. UOLISP is in an operational state. Please spend time going over the longer Installation Procedure that follows since it contains valuable information not contained in the manual. Also please spend time getting to know the placement of information in the Manual. There is a reasonable index. If you have any questions or comments about the manual please feel free to contact us at any time.

BUGS & FEATURES

When entering in a file name it is best to leave a space after the name inside the quotes, example "foo ", otherwise you will probably get disk error reports from TRSDOS. When entering characters A to Z do not use the shift key. If you do use the shift key some defined LISP functions may not be recognized by the reader. If you create a function where a shifted character is part of the name you may not remember having used the shift key and UO-LISP will generate an undefined function error. There is a miss spelling in the manual in the edit section. The function to restore a file created by the editor is listed as "RSTR" in the manual this should read "RESTRE".

Concerning the included **NEWSLETTER**, although most of the information concerns the CP/M based systems from time to time there will be TRSDOS news as well. Please be alert to the differences between the news items.

NON-TRSDOS OPERATING SYSTEMS

So far the NON-TRSDOS systems that are TRSDOS like in nature, available for the TRS-80 Model I & III **HAVE SHOWN TO BE COMPATIBLE WITH UO-LISP.** If you are running a NON-TRSDOS operating system and experience problems please call or write immediately.

USE OF HIGH MEMORY FOR DEVICE DRIVERS

Those of you running the UO-LISP Programming Environment with a version of TRSDOS or other operating systems like LDOS and DOS+3.5 which allow device drivers to be placed in high memory **PLEASE TAKE NOTE.**

This version of the UO-LISP Programming Environment was designed before high memory located device drivers were commonly utilized. The UO-LISP compiler currently expects to have access to all of available high memory. THUS, if not monitored the compiler might try to use the space occupied by some device driver. To prevent this from happening there is a function not documented in the manual which will indicate where the binary program space allocator is pointing. Use this function to monitor your binary space pointer and avoid letting the compiler allocate space used by a device driver. Also note that "FLOADing" a file uses binary program space too.

The function not documented is (BPS!$), it returns the next available byte of binary program space available for allocation. Allocation occurs low to high memory.

IMPORTANT NOTE
IMPORTANT NOTE
IMPORTANT NOTE
IMPORTANT NOTE

The tutorial guide included in this package was written for the CP/M based UO-LISP V2. You will find that it refers to a manual you did not receive. It will also refer to functions not included in your system.

The CP/M based version of the tutorial guide is now included with the TRSDOS based version of UO-LISP V1. **WHY INCLUDE IT?** We have found that a high percentage of those purchasing UO-LISP V1 have little or no LISP experience. You will find that about 90% of the tutorial guide is useful in learning LISP. We hope to retrofit the tutorial guide to TRSDOS some time in the future.

B U G R E P O R T

System_____

Version Number/Date_____

Name_____

Address_____

_____

_____

Zip_____

Phone and hours: (_____)-_____-_____/_____

Description Please provide a brief description of what you think the problem is and what its symptoms are. Please provide a complete program and data listing if possible so that we can duplicate the problem.

NCA USE ONLY: RECVD:___/___/___ REPL:___/___/___ RESP:___

P.O. Box 90995 Long Beach, California 90809 (213)-426-1893

I. Overview

The UOLISP system requires a minimum of 32k to operate and a single floppy disk. A two disk system is required to copy the software from the distribution disk to a disk with the operating system. UOLISP is configured for operation with the TRS-80 DOS 2.3 system on either the Model I or the Model III.

This manual and the users manual are not intended as introductions LISP programming. Users interested in learning LISP programming are advised to consult one of the books listed in the bibliography.

II. Installation of the System.

The floppy disk contains executable code for all the packages described in the UOLISP manual. The disk should be copied to another using the operating system BACKUP utility. The disk you have received contains none of the TRS-80 operating system except a directory. The files can be copied to a disk with the operating system COPY command. Alternatively the disk can be used as a data disk in a second drive.

III. Starting up LISP

After mounting the floppy in the default disk drive you execute the program LISP by just typing LISP32 or LISP48 for either the 32k or 48k versions. The system should respond with:

```
UOLISP. V1.x. date
*
```

within a few seconds. If not, the disk was not properly copied.

The V1.x is the version of the system you are running. All correspondence should mention this number. The prompt character is always an asterisk.

IV. TRS-80 File Names

In the users guide file names are given in a system independent form. In practice these file names must conform to the TRS-80 file name conventions [2].

File names within UOLISP are always strings in double quotation marks. Important: the last character of the file name should always be a blank.

Some conventions for file name extensions are:

/LSP - A LISP syntax source file.

/RED - An RLISP syntax source file.

no extension - Fast load files.

The following are file name equivalents for the files given in the users guide.

Guide	TRS-80	
"COMP"	"COMP "	Compiler.
"RLISP"	"RLISP "	RLISP parser.
"EDIT"	"EDIT "	Structure Editor.
"TRACE"	"TRACE "	Trace package.
"META"	"META "	Little META TWS.
"LAPP"	"LAPP "	LAP pretty printer.
"OPT"	"OPT "	Optimizer for compiler.
"VECTORS"	"VECTORS "	Vector package for RLISP.
"PRETTY"	"PRETTY "	Pretty printer.

V. Diskette Contents

The following files are present on the distribution diskette:

1. LISP32/CMD - A 32k version of UOLISP. Does not permit operation of all the packages. Does not have as many free cells or as large a symbol table. Not distributed with V1.5.

2. LISP48/CMD - The full sized 48k version.

3. COMP - A fast load version of the compiler.

4. RLISP - A fast load version of the RLISP parser.

5. EDIT - A fast load version of the structure editor.

6. TRACE - A fast load version of the trace package.

7. DEMO/RED - A demonstration program in RLISP.

8. LAPP - A pretty printer for the LAP assembler. Formats output and dumps code and addresses in hexadecimal.

9. OPT - A peephole optimizer for the compiler and assembler. Can be loaded with the compiler to produce smaller and faster code.

10. VECTORS - A vector package for use with RLISP. Implements arrays of any type.

11. PRETTY - A pretty printer. It is interfaced to RLISP and the EDITOR or may be loaded alone.

VI. Storage Requirements

The following are the approximate free spaces available in the 32k and 48k systems:

Contents	32k	48k
String Space (characters)	1500	3072
Free cells (pairs)	1325	2560
Symbol table (entries)	320	512
Code pointers (entries)	128	172
Stack space (entries)	4000	4096
Binary Program Space (bytes)	5158	13359

The following fast load files may be loaded at any address in binary program space. They take up the following amount of space:

RLISP	5143
Compiler	4737
Trace package	814
Editor	1798
LAP pretty printer	567
Optimizer	2695
Vector package	965
PRETTY printer	1244

Lisp Interpreter code size: approximately 7000.

These sizes do not include dotted-pairs and symbol table space required for their execution. The base LISP interpeter

including all data but less any free area requires about 13k bytes.

VII. Control Keys

All input and output from the TRS-80 keyboard is in capital letters. A few keys have special functions. The CLEAR key will interrupt any running program and cause an ERROR(0,NIL). During all output operations, any key which is typed (other than CLEAR) will cause temporary suspension of the program until another key is typed at which time execution will continue. Note that CLEAR is only effective during output operaions.

The standard editing controls of the TRS-80 for backspace and line clear are implemented. The BREAK key has no effect.

VIII. TRS-80 Graphics

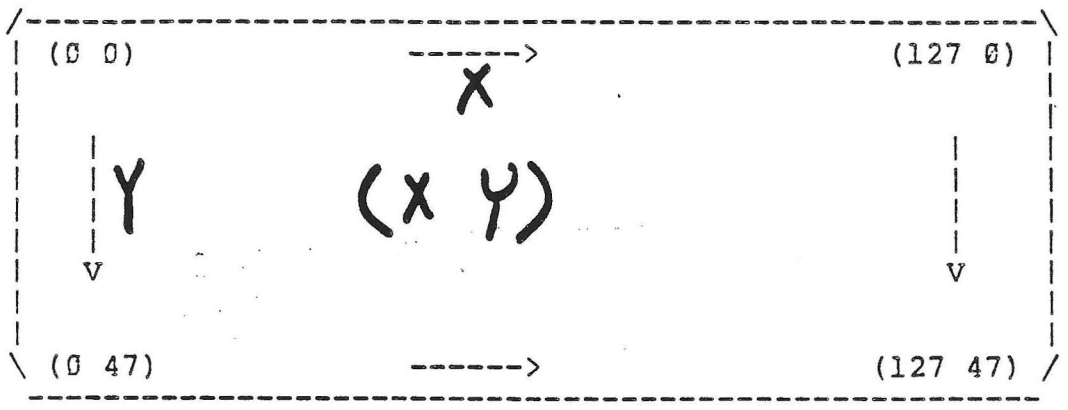
Three new functions have been added to permit low resolution graphics on the TRS-80 screen. Special thanks go to Bruce Douglass for these functions.

(RESETB X:integer Y:integer):NIL
Type: EVAL, SPREAD.
Turns off the pixel at location (X Y) on the TRS-80 screen. X should be in the range 0 to 127 inclusive and Y the range 0 - 47 inclusive. Other values may destroy the system.

(SETB X:integer Y:integer):NIL
Type: EVAL, SPREAD.
Turns on the pixel at location (X Y) on the TRS-80 screen. The X and Y values should be constrained as in RESETB.

(TESTB X:integer Y:integer):boolean
Type: EVAL, SPREAD.
If the pixel at location (X Y) is on, T is returned, otherwise NIL is returned. The X and Y values should be constrained as in RESETB.

The TRS-80 screen is referenced by the following X and Y coordinates:



IX. Program Descriptions.

A. Lisp Interpreter.

The LISP interpreter is the main program of the entire system. It functions much like a BASIC interpreter, reading function definitions, evaluating functions, handling errors and so on. The features provided by this program include:

1. A garbage collector - Most LISP data structures are constructed from dotted-pairs which are kept in a free list. When all these are used up by a program, the garbage collector locates all those not in use and returns them to the free list. Garbage collection in the 48k system generally takes about two seconds.

2. Disk I/O - UOLISP supports input from an auxiliary disk file whose name is supplied by the user. It also supports (even at the same time) output to a disk file whose name is supplied by the user. The current version of UOLISP does not support output to the line printer.

3. Functions - UOLISP supports both EXPR and FEXPR type functons. The EXPR type function is the usual form of function which has all its arguments evaluated before it is called. The FEXPR type function does not evaluate its arguments unless programmed to do so. In this manner, the user can experiment with control constructs other than those provided by the base system.

4. Hashed Symbol Table - The run time symbol table of UOLISP exists in 32 buckets each associated with a particular set of symbols determined by a hash function. This assures very fast input of LISP files from disk.

5. Arithmetic - UOLISP uses 13 bit signed arithmetic permitting numbers between -4096 and +4095.

6. Strings - Arbitrary strings of characters for error

messages and the like are implemented.

7. Function Set - Nearly all Standard LISP functions are implemented. The only exceptions are those dealing with floating point numbers, vectors, COMPRESS and EXPLODE, and some of the I/O functions. In all, 102 of Standard LISP's 123 functions are implemented. There are 20 additional functions not in Standard LISP also available to the user.

8. MAP functions - All 6 MAP functions are supported.

9. Fast Loader - A loader of relocatable "fast load" files is part of the resident system.

10. TRS-80 graphics - Three functions turn on, turn off, and test individual pixels on the 128 by 48 TRS-80 low resolution graphics screen.

B. Fast Load

It is expected that most functions will be compiled even during the debugging phase of a program. User implemented programs can be compiled as a block into a relocatable form called "fast load". Storing files in this form saves space and time. As many fast load files as will fit into available space can be loaded in any order. All support programs of the system are saved in this way, including the compiler, RLISP, the trace package and the editor.

This form of file loads considerably faster than reading the source and compiling it. Likewise, programs which are too big to fit into storage may be compiled into the fast load format and then loaded into storage where the compiler would normally be stored.

C. The LISP Compiler.

The LISP compiler converts LISP functions into code directly executable by the Z-80 microprocessor inside the TRS-80. The compiler can either generate a fast load file, or dump the instructions directly into main storage for execution. Compiled code uses about 1/2 the space of interpreted code and runs up to 5 or more times faster. It generally requires less space to run.

Compiled and interpreted code can be mixed. Some functions of a program can be interpreted and others compiled without consideration by the user. A special "fast link" permits function calls to be placed directly inline, a "slow link" scheme permits functions to be redefined, and traced even if they are compiled. The operation of the compiler is given in great detail in the users guide.

D. The LAP pretty printer.

This module formats the output of the LAP assembler for easy examination. Addresses and instructions are printed in their hexadecimal forms rather than decimal. The LAP instructions and labels are printed in a fixed format without parentheses.

E. The Optimizer.

This module takes the LAP code created by the compiler and performs 13 different optimizations on it. This include: removing redundant load register instructions, dead code, converting long to short jumps, using some special Z80 instructions, and removing extra stack frame allocations. The optimizer can also be enabled to open code some functions producing even faster code, sometimes as much as 30 % (with some increase in code size). A final class of optimizations removes some of the run time type checking and produces even faster code.

F. S-expression Pretty Printer.

This package formats LISP S-expressions by indenting them in a reasonable form. The expressions will not run over the screen boundaries as set by LINELENGTH. The pretty printer is interfaced to RLISP and the structure editor. It can be loaded directly into LISP and used by explicitly calling the PRETTYPRINT function.

G. The LISP Structure Editor.

A very simple LISP program permits the user to enter functions, execute them, and then save them in a disk file. The functions can be modified and listed. The disk file can be listed using regular system utilities, read by BASIC programs, and even edited by some of the more advanced editors (not EDTASM). The editor can be used in conjunction with PRETTYPRINT for more legible files.

H. The RLISP High Level Language.

The RLISP programming language was implemented by A. C. Hearn in 1973 to facilitate the implementation of a symbolic algebra system, REDUCE [3-4]. A reasonably complete subset of the syntax has been implemented for UOLISP. For example, in LISP a recursive factorial routine might be programmed like this:

```
(DE FACT (N)
  (COND ((LESSP N 2) 1)
        (T (TIMES N (FACT (DIFFERENCE N 1))))))
```

The same procedure written in RLISP looks more pleasing to the

experienced user of modern block structured programming languages:

```
EXPR PROCEDURE FACT N;
IF N < 2 THEN 1
  ELSE N * FACT(N - 1);
```

The RLISP parser is cleanly interfaced with the system and can be loaded from disk when required. The parser takes a little more than 4k bytes of storage. In the 48k system it may be loaded with the LISP compiler with about 5k bytes left over for compiled programs. The 32k system will not support both RLISP and the compiler together.

I. The Trace Package.

This set of functions permits users to watch the evaluation of both interpreted and compiled functions. The arguments of a function are printed out before the routine is entered, and the value of the function is displayed before it is exited. A BREAK function permits the user to stop a function and examine its local and global state.

J. The Vector Package.

This collection of routines implements the Standard LISP vector functions. Vectors (arrays) can be created at any time and can be of arbitrary size. The elements of a vector can be of any type and can even be of mixed types. Consequently a 2 dimensional array is implemented as a vector of vectors. Vectors can even be input to a program. For example:

```
@0, 1, @0, 1#, 3, "HELLO"#
```

creates a 5 element vector with the values 0, 1, @0, 1#, 3, and "HELLO" in its 5 locations.

X. Running the Demonstration Program.

The demonstration program can be run in either an interpreted mode, or can be compiled and run (it cannot easily be converted to a fast load file) on 48k machines. Enter the following:

```
LISP48 <enter>
(FLOAD "RLISP ") (BEGIN) <enter>
ON COMP; <enter>
IN "DEMO/RED ";  <enter>
```

The demonstration program will being to run. To freeze the output on the screen, type any character (other than CLEAR). To resume output, type any other key. Use of the compiler is

recommended because of of the length of the demonstration.

List of References

1. J. Marti, A. C. Hearn, M. L. Griss, C. Griss, "Standard LISP Report", SIGPLAN Notices, Vol. 14, No. 10, October 1979, pp. 48-68.

2. "TRSDOS & DISK BASIC Reference Manual", Radio Shack, Fort Worth, Texas, 1979.

3. A. C. Hearn, "REDUCE 2 Symbolic Mode Primer", Utah Computational Physics, Operating Note No. 5.1, October 1974.

4. A. C. Hearn, "REDUCE 2 User's Manual", Utah Computational Physics, UCP-19, March 1973.

Bibliography

1. Winston, P. W., Horn, B. K. P., "LISP", Addison-Wesley Publishing Company, Reading, Massachusetts, 1981.

2. Allen, J., "Anatomy of LISP", McGraw-Hill, New York, 1978.

3. McCarthy, J., Abrahams, P. W., Edwards, D. J., Hart, T. P., Levin, M. I., "LISP 1.5 Programmer's Manual", The MIT Press, Cambridge, Massachusetts, 1962.

4. Siklossy, L., "Let's Talk LISP", Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

5. Weissman, C., "LISP 1.5 Primer", Dickenson Publishing Company, Belmont, California, 1967.