LISP

Z/Z

This guide will not teach you LISP. It will show you how to use the specLISP interpreter for the Sinclair ZX Spectrum. If you are a LISP beginner, then we suggest you consult one of the books in our book list.

The interpeter is supplied with a demonstration program which requires 48K RAM to run. There are two copies of the interpreter and demo on side one of the cassette.

Differences between v1.1 and v1.2

Internally the major difference is that the storage management system has re-organised , resulting in a more efficient and bug free garbage collector. In v1.1 the FLP pointer (see section 6.2) pointed to the free area of memory. After garbage collection all the cells below FLP were compacted. In v1.2 the FLP pointer points to a list of free cells. When the interpreter is first run all the cells in memory are linked together (up to the bottom of the stack). The garbage collector does not compact cells but instead places unused cells back onto the freelist. However it is still necessary to compact cells prior to saving a program.The main result of the re-organisation is that the garbage collector runs much faster as it has no cells to compact.

A new function , STACK , has been introduced which sets the stack size and clears the memory. During initialisation the interpreter auotmatically performs a stack initialisation of 4k bytes.To change the stack size enter (stack n) , where n evaluates to a number. If you have a very recursive function the stack may overflow with disasterous results. If this happens re-load your program and increase the stack size.

The LPRINT function no longer exists. A new function PRT has been introduced which toggles output between the screen and printer.The following example illustrates the use of PRT by defining the LPRINT function.

```
(de lprint (s) (progn() (prt) (print s) (prt) ))
```

The namelist may be displayed by the new function OBLIST. The following shows an extract of the namelist after entering (oblist).

```
exit (subr 26445 pname (30821))
```

Exit is the name of a function , 26445 is the address of the machine code subroutine for the function , and 30821 is the internal representation of the characters "exit" i.e. its pname property.

A new function TRACE toggles the trace facility on and off. On entry to a function the parameters and their values are displayed , and on exit from the function the result is displayed.

e.g. * (de sq(n) (times n n))

```
sq
* (trace)      — switch trace on
t
* (sq 3)       — call sq.
n = 3          — parameter of sq traced

9              — result of sq traced
9              -- top level function result
* (trace)      — switch trace off
t
```

It is now possible to pause during evaluation by contiously pressing the space key , at the far right of the keyboard.

The 'cold start' entry point is now at address 24500 , and the 'warm start' is now at address 29069.

Programs produced using v1.1 will not run if loaded using v1.2

CONTENTS

version 1.2
PROGRAMMER'S MANUAL

SECTION

2.0 GETTING STARTED

(a) Rewind the cassette

(b) Type LOAD "" on the computer
Note that LOAD is a single key entry

(c) Press PLAY on the cassette player

(d) The screen will go black and the following message appear:

specLISP 1.2
(c) 1982 K.P. Stone Dec
Made in England

*

The '*' is the specLISP prompt and is displayed whenever i expected. If no prompt is displayed, then the function cal entered is being evaluated. When the evaluation is finishe result will be displayed followed by the prompt.

At any time, except during garbage collection, it is possib exit to the top level of the interpreter by pressing the 'C SHIFT' and 'SPACE' keys simultaneously. This is useful to a a runaway function.

Alternatively, it is possible to perform a return to BASIC pressing the 'CAPS SHIFT' and 1 keys simultaneously. After this, the user can use BASIC or can 'warm start' the inter by entering RUN USR 27806 and the state of the interpreter w be restored as if an exit had occurred. If the user does not overwrite the interpreter whilst in BASIC, then the interpre can be 'cold started' by entering RUN USR 24000.

During input mode only, pressing 'CAPS SHIFT' and 0 deletes previously entered character from the input buffer.

Pressing 'CAPS SHIFT' and 5 cancels the current input line b emptying the input buffer.

To summarise:

from within BASIC, 'cold start' by: RUN USR 24000.
from within BASIC, 'warm start' by: RUN USR 27806
delete previous character: 'CAPS SHIFT' 0
delete current line: 'CAPS SHIFT' 5
exit to BASIC: 'CAPS SHIFT' 1
exit to LISP: 'CAPS SHIFT' SPACE

specLISP SYNTAX

Identifiers may be any length, although for efficient use of mem
they should be multiples of 4 characters, i.e. 8 characters rath
than 5, 4 characters rather than 5. All characters are significa
Uppercase and lowercase characters are distinct, i.e. the identi
"TEST" is not the same as "test". specLISP predefined identifiers
are only recognised in lower case. Identifiers are terminated
by spaces, '(',')' or 'ENTER'. When a line is input longer than
254 characters, the interpreter will insert 'ENTER' as the 255th
character.

```
<s-expr>
<list>          = <list>|<dotted pair>|<atom>
<dotted pair>   = ( {<s-expr>}* )
<atom>          = (<s-expr> . <s-expr>)
<identifier>    = <identifier>|<number>
<char>          = <char> {<char>|<digit>}*
<digit>         = a..z|A..Z|:|;|<|>|=|?|@|?|/|↑|_| f
<number>        = 0..9
                = {+|-}<digit> {<digit>}*
```

4.0 SUMMARY OF specLISP FUNCTIONS

All of the specLISP functions and predicates are descri
three ways:

(a) the function syntax
(b) the function evaluation rules
(c) an example

Unless otherwise stated, all arguments of a function are
before execution of the function.

Predefined specLISP names are underlined for clarity.

In the following examples, k is the list (a b c d)
and square is the function defined as (times n n)

4.1 (apply x args)

x must be a lambda expression. Args is a list of Ar
to be applied to the lambda expression

e.g. (apply (quote square)(quote (4)))

=16

4.2 (and x1 x2 x3 ... xn)

Returns the value of xn if x1 to xn are not nil, othe
returns nil.

e.g. (and (car k)(cdr k))

= (b c d)

e.g. a) (cond ((null k) nil)
((eq (car k) (quote b)) (quote b))
(t (cdr k)))

If k = (a b c d) then the result is (b c d)

If k = (b c d) then the result is b.

e.g. b) (cond ((eq a b) (setq a 1) (setq b 2))
)

If a = b then set a to 1, and set b to 2.

4.9 (cons x1 x2)

A new cell is created with the car of the cell conta
x1, and the cdr of the cell containing x2.

e.g a) (cons k (quote e)
= ((a b c d) . e)

e.g. b) (cons 1 2)
= (1 . 2)

4.10 (cset name x)

x is placed as an apval property on the property list o
name. Name must evaluate to an identifier.

e.g. (cset (car k) k) = (a b c d)

i.e. (a b c d) is the apval property of a.

4.11 (csetq name x)

Equivalent to (cset (quote name) x)

4.12 (de function name (parameters) body)

The parameters and body are placed as a lambda expressio
under the expr property of function name. If an expr
property already exists, it will be replaced by the new
property.

Note: No evaluation occurs. The parameter list may be
empty.

e.g. (de square (n) (times n n)

4.13 (eq s1 s2)

If s1 and s2 are the same pointer

i.e. the same cell, then t is returned, otherwise n

e.g. (eq (quote (a b c d))k) = nil
(eq k k) = t

4.14 (equal s1 s2)

As eq, but also returns t if s1 and s2 are the same number or the same list structure.

e.g. (equal (quote (a b c d))k) = t

4.15 (eval s)

The value of s is itself evaluated.

e.g. (eval (quote (plus 2 3))) = 5

4.16 (exit)

Forces a return to the top level of the interpreter, t for handling errors. If exit is used within a functior when the exit is evaluated the current alist is lost a the alist existing at the time of the previous top lev function call is restored.

e.g. (exit) = (EXIT)

4.17 (funct function-name)

Creates a functional argument using the lambda express identified by function-name, and the current alist poi Function-name is not evaluated.

e.g. (funct square)

n.b. (cdr (funct function-name)) will display the alis

4.18 (gc)

The garbage collector is called, and returns nil.

e.g. (gc) = nil

n.b. The garbage collector can be extremely slow.

4.19 (get name indicator)

The property value under indicator on the property list of name is returned. If the indicator is expr then the function definition of name is returned.

e.g. (get (quote square)(quote expr)) = (lambda (n)(times

4.20 (greaterp n1 n2)

Returns t if number n1 is greater than number n2.

e.g. (greaterp 4 2) = t

4.21 ((lambda (parameters) body) arguments)

Usually used indirectly by using de. Can be used as an 'anonymous' function.

e.g. ((lambda (x)(times x x))4) = 16

Can be used in a put function to create a function under a property other than expr, and then evaluated by using apply.

e.g. (putprop (quote test)(quote lambda (n)(times n n)))
(quote example))
(apply (get (quote test)(quote example))(quote (4))

4.22 (lessp n1 n2)

Returns t if number n1 is less than number n2.

e.g. (lessp 4 2) = nil

4.23 (list s1 s2 s3...sn)

s1 ... sn are consed into a list.

e.g. (list (quote a)(quote b)) = (a b)

4.24 (load name)

Loads from cassette a previously saved memory image saved as name. The memory is restored to that at the time of the save. The contents of the current memory are lost. Name need not be present, in which case the first file found on the cassette will be loaded. Name is automatically truncated to ten characters. Analogous to the BASIC load command.

4.34 (progn (variables) s1 s2 ... sn)

The variables (if any) are bound to nil without bein evaluated. s1 to sn are sequentially evaluated and t value of sn returned. On exit from the progn functio the variables are removed from the alist.

e.g. (progn (z x)
(setq x k) = (a b c d)
(setq z x)

)

As a point of interest, note that the above setq's c be concatenated as (setq z (setq x k)) with the same

4.35 (putprop name value indicator)

The value is placed on the property list of name as indicator property.

e.g. (putprop (quote square)(quote (lambda (n)(times (quote expr))

Note that this example is equivalent to (de square (n

4.36 (quote x)

Returns x unevaluated. It is not necessary to quote nil and t.

e.g. (quote (a b c d)) = (a b c d)

4.37 (read)

A prompt is displayed and an s-expression is input. A pointer to the input list is returned.

e.g. (read)

*example

= example

4.38 (remprop atom indicator)

The property name called indicator is removed from th property list of atom. If the property was found, t i returned, else nil.

e.g. (remprop (quote a)(quote apval)) = t

4.39 (reverse x)

The list x is reversed.

e.g. (reverse k) = (d c b a)

4.40 (rplaca x1 x2)

The car of x1 is replaced by x2.

e.g. (rplaca (cons 1 2) 3) = (3.2)

Note: To remove all the properties of atom square do:
(rplaca (quote square) nil)

The error message "PNAME PROPERTY NOT FOUND" will be displayed after this command because the interpreter tries to print "square" but cannot find it.

4.41 (rplacd x1 x2)

The cdr of x1 is replaced by x2.

e.g. (rplacd k (quote z)) = (a.z)

4.42 (save name)

The memory image is saved as a file on cassette as name. Name is automatically truncated to ten characters. Analogous to the BASIC save command. It is advisable to do a garbage collection before saving a program.

4.43 (set name value)

The value is bound to name.

e.g. (set (car k)(cdr k)) = (b c d)

The value of a is (b c d) if k = (a b c d)

4.44 (setq name value)

Equivalent to (set (quote name) value)

4.45 (verify name)

Verifies the LISP program, name. Analogous to the BASIC verify command.

4.46 (while test s1 s2 ... sn)

While test is t, s1 to sn are sequentially evaluated. The result is always nil.

```
e.g.  (progn (i)
      (setq i 1)
      (while (lessp i 100)        = 100
      (print i)
      (setq i (add1 i))
      )
      i
      )
```

4.47 (zerop number)

Returns t if number is zero, otherwise nil.

e.g. (zerop 1) = nil

Note: if during load, save or verify, the program is interrupted by pressing space, then the interpreter will exit BASIC. To return to specISP 'warm start' the interpreter by entering RUN USR 27806.

Arithmetic Functions

Numbers must be in the region -32768 to +32767

4.48 (plus n1 n2)

e.g. (plus 2 3) = 5

Returns the sum of n1 and n2.

4.49 (diff n1 n2)

e.g. (diff 2 3) = 1

Returns the difference of n1 and n2.

4.50 (times n1 n2)

e.g. (times 2 7) = 14

Returns the product n1 and n2.

4.51 (div n1 n2)

e.g. (div 6 4) = 1

Returns the quotient after dividing n1 by n2.

4.52 (rem n1 n2)

e.g. (rem 6 4) = 2

Returns the remainder after dividing n1 by n2.

4.53 (add1 n)

e.g. (add1 3) = 4

Equivalent to (plus n 1)

4.54 (sub1 n)

e.g. (sub1 3) = 2

Equivalent to (diff n 1)

5.0 EVALUATION AND BINDING

5.1 Evaluation of S-expressions

The interpreter uses the deep binding technique
atoms are stored with their values in a list cal
alist (association list). For each atom the eval
(MEVAL) will search the alist. This is slow, but
binding does allow efficient implementation of fu
arguments (funarg).

If an atom is to have a constant value, the atom s
set by the CSETQ function. This function places th
as an APVAL property on the property list of the a
always looks for an APVAL property before searchin
alist and is, therefore, more efficient.

MEVAL evaluates s-expressions (s-expr) in the follo
can take place.

(1) If the s-expr is a number, then no further ev

(2) If the s-expr is a symbolic atom, the property
the atom is searched for an APVAL property. If
is no APVAL property, the alist is searched. If
atom is not on the alist, it is an unbound vari
and an error will occur. If (1) and (2) are not
fied, then the s-expr must be a list.

(3) If the car of the list is an atom, then it shou
a function.

(a) If the function has a SUBR property, then t
of the s-expr is evaluated by jumping to the
SUBR property value.

(b) If the function has an EXPR property, then th
of the s-expr is applied to the lambda expres
by MAPPLY.

(c) If the function has no EXPR or SUBR property,
the alist is searched for the atom. If the ato
is not found, then an error will occur. If the
value of the atom on the alist is not a funarg,
then it is not a function and an error will occ
Otherwise MAPPLY is called.

(4) (a) If the car of the list is a lambda expression,
the cdr of the s-expr is applied to the lambda
expression by calling MAPPLY.

e.g. (cond ((numberp x)(quote plus))
(t (quote cons))) x 3)

If x is a number, then the result is (plus x 3
otherwise it is (cons x 3)

5.2 Variable Binding

A bound atom is a cell, the car of which is the atom and the cdr of which is the atom's value. At the top level of the interpreter, any atom can be given a value.

As previously explained, when MEVAL encounters an atom, it first looks for an APVAL property. If there is no such property, the alist is searched for the first occurrence of the atom. An atom will only be on the alist if either:

(a) it was bound by a set/setq at the top level of the interpreter (i.e. before any function calls)

(b) it was declared as a progn variable

(c) it was bound within a higher level function

Note that MEVAL knows when it is at the top level because TLP is NIL (see sec 5.3). When a user defined function is called, the parameters are automatically bound to the valu in the call. If there are more formal parameters than actu parameters, then the extra formal parameters are bound to

e.g. (F 1 2) when F = (lambda (x y z) ...)
will bind x to 1, y to 2 and z to nil.

Within a called function, only atoms on the alist in the environment of the function can be accessed.

```
e.g.  *(de F(x)
      *(progn())
      *(setq y x)
      *(setq x(times x x))
      *))
      F
      *(setq x 5)
      5
      *(setq y 10)
      10
      *(F 4)
      16
      *x
      5
      *y
      4
```

With F, x is bound to 16, on exit from F the only x on the alist is the original value. The value of y is altered because y was set at the top level and not used as a parameter in F.

It is possible to directly access the alist by using
Remember that funcbn returns a funarg, the cdr of w
is the current environment (alist).

e.g. enter the following:

```
*(de F(n)(times n n))
F
*(setq x 1)
1
*(setq y 2)
2
*(cdr(funcbn F))
((y.2)(x.1))      <---the current alist
*(rplacd(cadr(funcbn F))3)   <---Changes value of y
(y.3)
*(cdr(funcbn F))
((y.3)(x.1))              <---Modified alist
*
```

specLISP does not permit a function to recieve an unsp
number of arguments. An example of a function of this
is AND. However, as with most LISP functions, it can b
emulated. To emulate the AND function, we can pass arg
to the pseudo AND as a list. Notice that the function
its arguments unevaluated.

e.g.
```
*(de p-and(args)
*(cond((null args)nil)
*((null(cdr args))(eval(car args)))
*(t(p-and(cdr args)))))
p-and
*(setq a 1)
1
*(setq b (quote(a b)))
(a b)
*(p-and(quote(a b)))
(a b)
*(p-and(quote(a b nil)))
nil
```

5.3 Function application

MAPPLY is passed three arguments:

(a) a pointer to the environment which will indicate part of the alist to be used when evaluating the function.

(b) a pointed to a lambda expression

(c) a pointer to a list of arguments

MAPPLY evaluates each argument and binds it to the corresponding formal parameter of the lambda expression. The bound values are placed in a temporary list pointed at by TLP, while the remainder of the parameter list is being evaluated.

When the parameters have been evaluated, they are remove from the temporary list and placed on the alist using environment pointer as the new alist pointer. The old alist pointer is saved in the temporary list and the function evaluated.

After evaluating the old value of the alist, the pointer is restored from the temporary list, and the result of the function is returned.

An example of using funarg

```
*(de test ()
*(progn(x)
*(setq x 100)
*(g(func square))))
test
*(de g (f x)
*(progn()
*(setq x 10)
*(f)))
g
*(de square()(times x x))
square
*(test)
10000
*
```

Executing test will return the result 10000 which has the value of x in the environment when the funarg was created.--

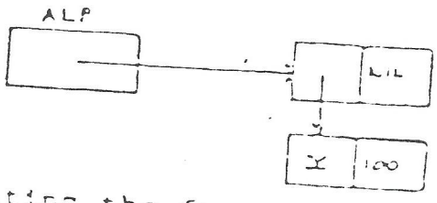
5.4 User Defined Functions

Every function has an environment; the environment is
the alist which is in existence at the time of the func
call.

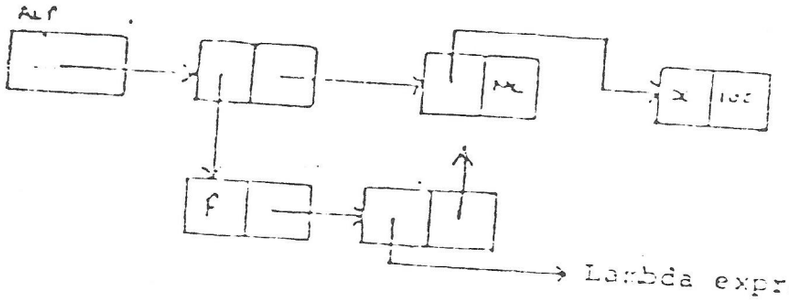
When a function is called, the subroutine MAPPLY is cal
first. If the function has an EXPR property, then the c
environemtn is passed to MAPPLY. If the function is a f
then the environment indicated by the funarg is passed t
MAPPLY.

A funarg is a functional argument and is created when a
function name is passed as a parameter. The funarg compr
the lambda expression of the argument and a pointer to t
current environment. The funarg is bound to the function
name and placed on the alist in the usual manner.

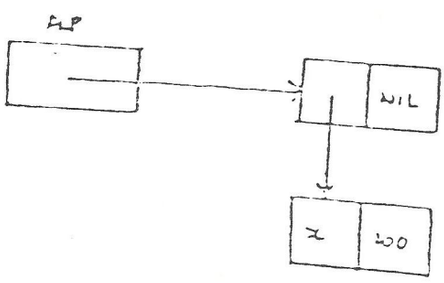
e.g. assume the only atom on the alist is x with a value
of 100



After evaluating the function call (g(funcaf)) the alist
will contain a funarg for f.

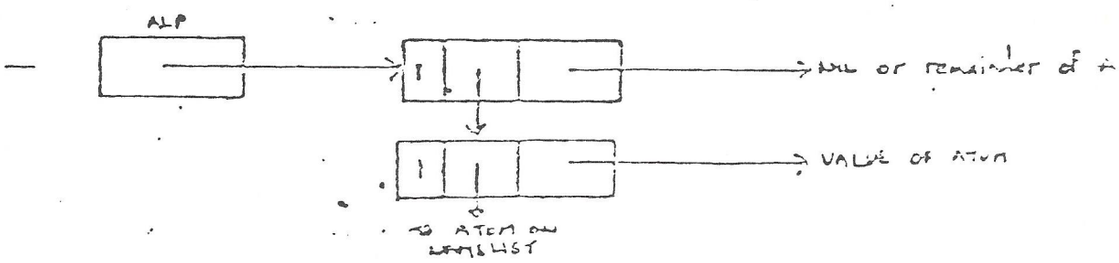


When the function f is called
from within g, the environment
will be that at the time of
the funarg creation.



Structure of namelist after reading the atom "aname"

ALP: Points to the alist, each element of which is a dotted pair of the form (atomname.value)



ELP: Points to the execution list, i.e. the current input command list.

TLP: Points to the temporary list, which is used when evaluating lambda expressions and as a 'stack'.

Another pointer, FLP points to the next free location for new cells in free space. Free space is the region of memory which has not yet been used or which has been reclaimed by garbage collection. FLP is updated every time a new cell is required. If FLP approaches the top of the stack, then the garbage collector is called.

6.3 The Garbage Collector

The memory space of the interpreter is occupied at one end by specLISP cells, and at the other end by the machine stack. When the memory is nearly exhausted, the garbage collector will automatically be called. The garbage collector collects all the cells which are no longer used. When properties are removed, atoms deleted from the alist etc., the cells still remain in the memory space.

The garbage collector marks each accessible cell by searching the alist, the namelist, the temporary list and the list which holds the current input line.

After marking the cells, the garbage collector sweeps the memory by compacting the marked cells and rearranging pointers to the cells. There may be pointers to the cells on the stack and, therefore, the stack is searched and pointers to cells rearranged if necessary.

USING MACHINE CODE

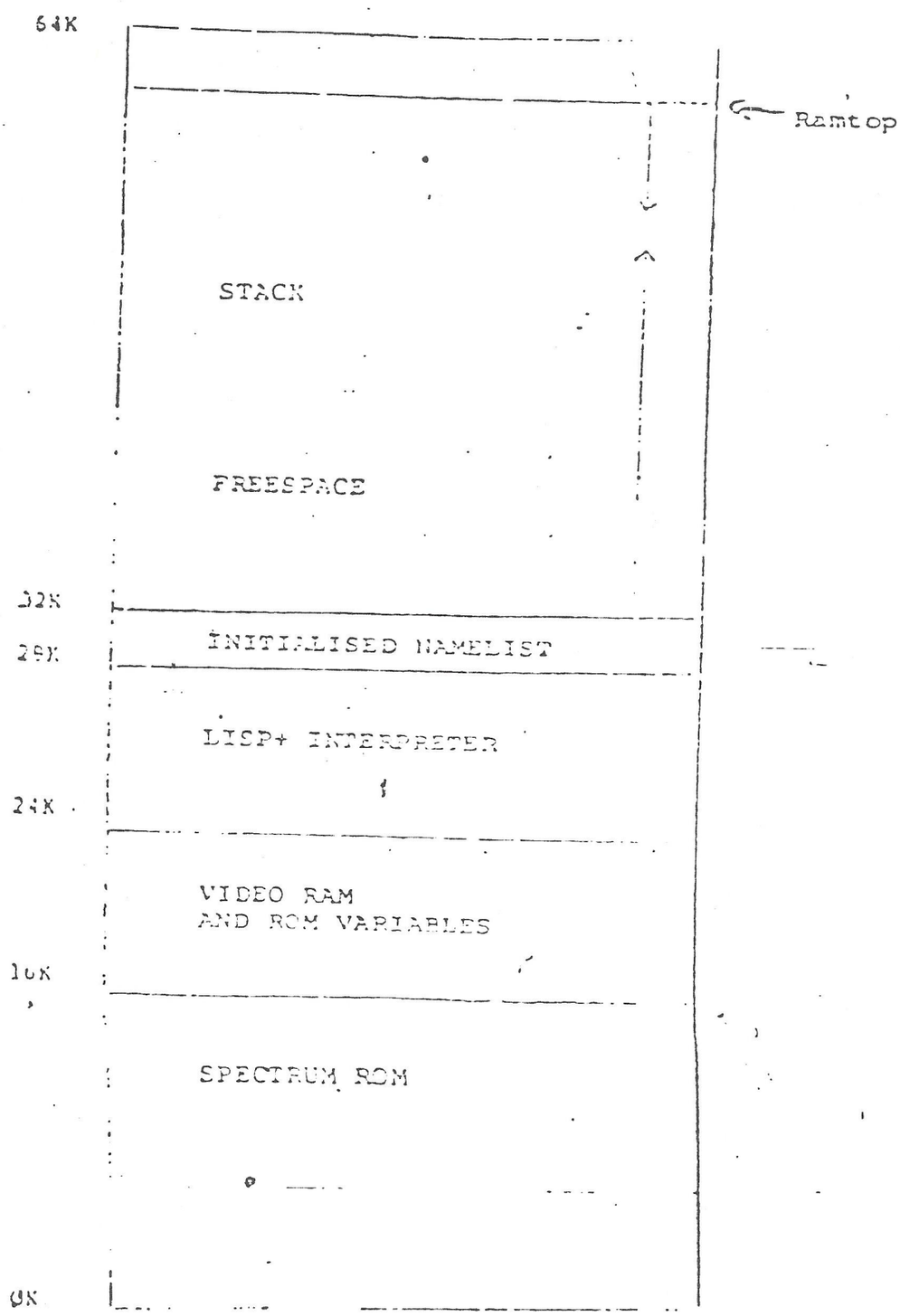
A machine code subroutine is identified in specLISP by an atom with the property name of SUBR. The value of the property is a 16 bit address. As specLISP does not recognises integers greater than +32767, a subroutine address greater than this must be entered as a negative number. The number to enter is n-65536 where n is the address of the subroutine.

specLISP assumes all free memory below RAMTOP is for cell storage, therefore, machine code subroutines must be located above RAMTOP. If your machine code occupies less than 169 bytes, then it can be entered in the area reserved for user defined graphics, which is not used by specLISP. If you require more memory, then RAMTOP must be altered in BASIC by CLEAR n where n is the top of the new specLISP freespace. Do this before you run the interpreter. Use the BASIC load/save/verify commands to save your machine code before using it.

For example, assume we have a machine code subroutine called scroll at location 65500.
The value of the SUBR property for scroll will be 65500-65536= -36
To inform specLISP of scroll, enter (put prop(quote scroll)-36(quote subr
The subroutine can be executed by *(scroll) i.e. as if scroll was a normal function.

If any parameters are in the function call, then on entry to the subroutine the HL registers will point to a list of the parameters If there are no parameters, HL will contain NIL (Ø). On exit from the subroutine, the interpreter expects to find a value in HL which will point to a list to be output. If HL contains Ø, then 'nil' will be output. Note: the IX register must not be altered, as it contains a pointer to the error return address on the stack. If HL does not contain a valid pointer or Ø, then the interpreter may 'crash'.

SPECTRUM MEMORY MAP WITH SpecLISP



8.1 The Database Functions

```
(de demo()
(progn(l1 l2)
(print (quote ok:))
(while t
  (setq l1 (read))
  (cond((atom l1)(print(quote(incorrect entry))))
    (t
    (setq l2 (list(car l1)(cadr l1)))
    (cond((equal(cddr l1)(quote(;)))(print(insert l
      ((equal(cddr l1)(quote(?)))(print(fetch l2)))
      ((equal(cddr l1)(quote(!)))(print(remove l2))
      (t(print(quote(incorrect entry))))
    ))
  )
)))


(de append(x y)
(cond((null x)y)
  (t(cons(car x)(append(cdr x)y)))
))



(de insert(fact)
(progn (l)
(setq l (get(car fact)(quote data)))
(cond((member(cadr fact)l)nil)
  (t(putprop (car fact)(append(cdr fact)l)(quote data
    fact)
)))



(de match (p d)
(cond ((and(null p)(null d))t)
  ((or (null p)(null d))nil)
  ((or (equal.(car p)(car d))(equal (car p)(quote _)))
    (match (cdr p)(cdr d)))))


(de fetch(patt)
(progn (item result)
(setq item (get(car patt)(quote data)))
(setq patt (cadr patt))
(while item
  (cond ((match patt (car item))
    (setq result (cons(car item)result))))
    (setq item (cdr item)))
result
))
```

```
(de delete (e l)
(cond ((equal e(car l))(cdr l))
  (t(cons (car l)(delete e(cdr l))))))

(de remove(fact)
(progn (l)
(setq l (get (car fact)(quote data)))
(cond ((member (cadr fact) l)
  (putprop (car fact)(delete(cadr fact)l)(quote data))
  fact
))))
```

3.0 MESSAGES

All specLISP messages are in uppercase characters.

1. Information Messages

1.1 WAITING FOR GARBAGE COLLECTOR

- The interpreter is performing a garbage colle
The garbage collector cannot be interrupted.

1.2 EXIT

- The interpreter has responded to (exit) or to
'CAPS SHIFT' SPACE, and has returned to the t

1.3 CANCEL

- The input buffer has been reset after entering
'CAPS SHIFT' 5.

2. Error Messages

2.1 x IS AN UNBOUND VARIABLE

- was not bound (using setq, set, csetq or cset)
at the top level of the interpreter and is not
a function parameter, and is not a local progn
variable.

2.2 x IS NOT A FUNCTION

- x has not been defined as a function name or is not
a funarg

2.3 x IS NOT A NUMBER

- The interpreter has attempted to perform an arith-
metic function using x, and x is not numeric.

2.4 CAN'T TAKE CAR OF x

- x is not a list

2.5 CAN'T TAKE CDR OF x

- x is not a list

2.6 x ARE INCORRECT ARGUMENTS

- x has been supplied to a specLISP function and has
either too few arguments or too many.

2.7 PNAME PROPERTY NOT FOUND

- An atom on the namelist has no pname property and, therefore, cannot be printed

2.8 ILLEGAL NUMBER FORMAT

- Illegal character entered, e.g. not in specLISP syntax

2.9 x IS NOT SYMBOLIC

- x is not a name

2.10 x IS NOT A FILE NAME

- x must be a name of a file to save

2.11 'C' EXPECTED

- a list can only begin with a 'C'

10.0 RECOMMENDED READING

(1) LISP - Winston and Horn
An excellent introduction to LISP

(2) ARTIFICIAL INTELLIGENCE - Winston

Part 1 is introduction to artificial intelligence techniques

Part 2 is introduction to LISP

(3) BYTE magazine August 1979

A special LISP issue

(4) ANATOMY OF LISP - Allen

A thorough book on LISP interpreters and compilers.