

Programming in System RPL

Eduardo Kalinowski

First Edition

Preface

The programming features of the HP48 graphical calculator are very powerful. They allow you to do virtually anything. However, the documented programming functions, that are directly accessible to the user (the user language), is not everything the calculator can do.

There is another language: the System language. The User language is a subset of the System one, with just some commands and just a fraction of its power. However, the System language is not well documented. The existing documents on that subject are turned to someone who already knows it; they are just listings of the commands with some brief descriptions. Once you already know the language, even the brief descriptions can be left out, and those documents are really a very good source of information. But how does one *learn* System RPL?

The purpose of this book is exactly that: to be a way for someone who has already learned User RPL (if you haven't yet, learn it before, then come back to this), and wants to learn the *real* power of the calculator.

It is divided in three parts: the first (Basic RPL) teaches the basic aspects of the language, the ones the user must know in order to create some simple programs, which take and return values from and to the stack. The second part (Advanced RPL) deals about some more advanced concepts, such as other ways of getting input (directly from the keyboard, using input forms, etc.). Finally, the third and last part (Reference) is exactly what the name says. In the first two parts, only the most important commands about a subject are listed. In the reference, you will find many other commands. In the very end, there are two appendices: the first presents some tools for programming in System RPL and their basic features. The second explains how to create libraries.

I would greatly appreciate suggestions and corrections for further enhancements of this book.

July 12, 1998

Eduardo de Mattos Kalinowski

Acknowledgements

This work could not have been accomplished without the help of many people. Firstly, I would like to thank Mika Heiskanen for his entry point list by subject. That document was my primary source of information. I would also like to thank him for developing JAZZ, whose disassembler and other functions helped me a lot.

Many thanks to everybody who patiently answered several questions during the development of this (listed in alphabetical order):

Al Arduengo	Joe Horn	Christian Meland
Sune Bredahl	Werner Huysegoms	John H Meyers
Len Fellman	Dan Kirkland	Melissa Reid
Raymond Hellstern	Hakim Mazouz	

If I forgot someone, please forgive me, and be sure I am grateful also.

And lastly, I would like to thank the Hewlett-Packard team for bringing us such wonderful tool the HP48 is.

Disclaimer

This document is © by Eduardo M Kalinowski.

It is distributed in the hope it will be useful for those who want to learn System RPL or want a reference about it, but it is provided “as is”, without warranty of any kind, either expressed or implied. In no event, I shall be liable to you for damages, including any general, special or incidental or consequential damages caused directly or indirectly by the use of the information provided here. I do not assure that any information here is right. Use it at your own risk.

This document may be distributed only if in its whole, unmodified form; and if you do not charge anything for it (except nominal copying fees). Otherwise, I will want my share!

The latest version of this document and its errata can be found at <http://hp48g.home.ml.org>.

Table of Contents

Preface	i
Acknowledgements	ii
Disclaimer	iii
Table of Contents	iv
 PART I BASIC RPL	 1
Chapter 1 Introduction	3
1.1 Your first System RPL program	4
 Chapter 2 Object Types	 7
2.1 Binary Numbers	7
2.2 Real numbers	8
2.3 Complex numbers	8
2.4 Characters	9
2.5 Strings	9
2.6 Hexadecimal strings	9
2.7 Identifiers	10
2.8 Tagged objects	10
2.9 Lists	10
2.10 Arrays	11
2.11 Units	12
2.12 Symbolics	13
 Chapter 3 Stack operations	 14
 Chapter 4 Checking arguments	 15
4.1 Number of arguments	15
4.2 Argument type	16
4.3 Examples	17
 Chapter 5 Local variables	 19
5.1 Named local variables	19
5.2 Unnamed local variables	20
5.3 Suggestions	20
 Chapter 6 Runstream control	 22
 Chapter 7 Conditionals	 25
7.1 Tests	25
7.2 Object type tests	25
7.3 IF...THEN...ELSE	26
7.4 CASE	27
 Chapter 8 Loops	 28
8.1 Indefinite loops	28

8.2 Definite loops	29
Chapter 9 System and memory operations	30
9.1 Variables and directories	30
9.2 System commands	32
PART II ADVANCED RPL	35
Chapter 10 Error handling	37
10.1 Trapping errors	37
10.1.1 The protection word	37
10.2 Generating errors	38
Chapter 11 Keyboard control	39
11.1 Key locations	39
11.2 Waiting for a key	39
Chapter 12 Using InputLine	40
12.1 An example	41
Chapter 13 Creating input forms	42
13.1 Label definitions	42
13.2 Field definitions	43
13.3 Label and field counts	45
13.4 The message handler	45
13.5 The title string	45
13.6 Results of the input form	45
13.7 An example	45
Chapter 14 The parameterized outer loop	48
14.1 Parameterized outer loop words	49
14.2 The display	50
14.3 Error handling	50
14.4 Hard key assignments	50
14.5 Menu key assignments	51
14.6 Preventing suspended environments	52
14.7 The exit condition	52
14.8 An example	52
Chapter 15 The display	55
15.1 Display organization	55
15.2 Preparing the display	56
15.3 Controlling display refresh	57
15.4 Clearing the display	57
15.5 Annunciator control	58
15.6 Display coordinates	58
15.6.1 Display subgrobs	58
15.6.2 Scrolling the display	58
15.7 Displaying text	59
15.7.1 Medium font	59
15.7.2 Large font	60
15.7.3 Displaying warnings	60
15.8 The menu line	61
15.9 Using graphic objects	61
15.9.1 Grob tools	62
15.9.2 Grob dimensions	64

PART III REFERENCE	65
Chapter 16 Notes	67
Chapter 17 Binary integers	69
17.1 Built-in binary integers	69
17.2 Conversion words	72
17.3 Arithmetic functions	73
1.4 Tests	75
Chapter 18 Real numbers	76
18.1 Built-in real numbers	76
1.2 Conversion words	78
1.3 Real functions	78
1.4 Tests	80
Chapter 19 Complex numbers	81
19.1 Built-in complex numbers	81
19.2 Conversion words	81
19.3 Complex Functions	81
19.4 Tests	82
19.5 Reals and complex numbers functions	83
Chapter 20 Character strings	84
20.1 Built-in characters	84
20.2 Built-in character strings	85
20.3 Conversion words	86
20.4 Management words	87
20.5 Tests	91
Chapter 21 Hexadecimal strings	92
21.1 Conversion words	92
21.2 General functions	92
21.3 Arithmetic functions	93
21.4 Tests	94
Chapter 22 Tagged objects	95
Chapter 23 Arrays	96
23.1 General operations	96
23.2 Calculations	98
1.3 Statistics	100
Chapter 24 Unit objects	102
24.1 Creating units	102
24.2 General operations	102
24.3 Arithmetic functions	103
24.4 Tests	103
Chapter 25 Composites	104
25.1 General operations	104
25.2 Building	106
25.3 Exploding	107
25.4 Lists	107
25.5 Secondaries	108

Chapter 26 Meta objects	109
26.1 Stack functions	109
26.2 Combining functions	109
26.3 Meta and object operations	110
26.4 Other operations	111
Chapter 27 Symbolics	114
27.1 General operations	114
27.2 Mathematical operations	117
27.2.1 Collection	120
27.2.2 Expansion	121
27.2.3 Integration	121
27.2.4 Other functions	123
27.3 Meta symbolics functions	124
27.3.1 Adding operators	124
27.3.2 Changing operators	124
27.3.3 Splitting algebraic metas	125
27.3.4 Miscellaneous	126
27.3.5 Rules menu operations	126
Chapter 28 Library and backup objects	129
28.1 Port operations	129
28.2 ROM pointers	129
28.3 Libraries	130
28.4 Backup objects	132
Chapter 29 Stack operations	133
Chapter 30 Checking for arguments	137
30.1 Type checking	138
Chapter 31 Temporary environments	140
31.1 Built-in ids and lams	140
31.2 Conversion words	140
31.3 Temporary environments words	141
Chapter 32 Error handling	145
32.1 General words	145
32.2 Error generating words	147
Chapter 33 Flags and tests	148
33.1 Flags	148
33.2 General tests	149
33.3 Object type tests	150
Chapter 34 Runstream control	151
34.1 Quoting objects	153
34.2 Skipping objects	155
Chapter 35 Conditionals	156
35.1 Flag tests	157
35.2 Binary number tests	159
35.3 Real and complex numbers tests	161
35.4 Meta object tests	162
35.5 General object tests	163
35.6 Miscellaneous	164

Chapter 36 Loops	166
36.1 Indefinite loops	166
36.2 Definite loops	167
Chapter 37 Memory operations	169
37.1 Temporary memory	169
37.2 Recalling, storing and purging	170
37.3 Directories	172
37.4 The hidden directory	174
Chapter 38 Time and alarms	175
Chapter 39 System functions	179
39.1 User and system flags	179
39.2 General functions	180
Chapter 40 Keyboard control	183
40.1 User keys	183
40.2 Waiting for keys	184
Chapter 41 The display	186
41.1 Display organization	186
41.2 Preparing the display	186
41.3 Controlling display refresh	187
41.4 Clearing the display	189
41.5 Annunciator and modes control	190
41.6 Window coordinates	191
41.6.1 Scrolling the display	191
41.7 Displaying text	192
41.7.1 Medium font	192
41.7.2 Large font	193
41.7.3 Displaying warnings	193
Chapter 42 Graphics	194
42.1 Built-in grobs	194
42.2 Dimensions	194
42.3 Grob handling	195
42.4 Creating menu label grobs	197
Chapter 43 User functions	199
Chapter 44 Commands by name	204
Chapter 45 Commands by address	218
APPENDICES	233
Appendix A Tools for programming in System RPL	235
A.1 HP Tools	235
A.2 JAZZ	236
Appendix B Creating libraries	237
B.1 Converting a directory into a library	237
B.2 Creating libraries directly	238

Part I

Basic RPL

Chapter 1

Introduction

If you know how to create programs in User RPL (if you don't, learn before you continue reading this book), then you only know part of what the HP48 calculator can do. The System RPL programming language gives you power to do many things which you could not even imagine, and *fast*. For example, in System RPL you can handle all 29 object types available. User RPL only gives access to some of them. Or you can do math with 15-digit accuracy, use arrays with non-numeric elements, and much more.

But before we start talking of System RPL, let us go back to User RPL to explain how it really works. I know you are anxious to start with the big thing right now, but the following information is important to a good understanding of System RPL.

The HP48 programs (User and System) are not stored internally using the names of the commands. Only the addresses of the objects are stored. Each of these addresses takes 2.5 bytes. When a program is run, the only thing that is actually done is a "gosub" to that address.

Some times, the address is another program with more jumps to another program with more jumps, and so on... A return stack keeps track of all the jumps, and this return stack does not have a fixed size, so you can have as many jumps as necessary, and you will always return to where you were before. Of course, the jumps must end somewhere, either in a program written in machine language or in an object that just puts itself in the stack (numbers, strings, etc).

But if the programs are just addresses, how can they be edited? The answer is that the HP has a table of the user commands' names and their corresponding addresses. So, when you put an User RPL program in the stack, the HP searches the table to get the name of the commands corresponding to the addresses stored in memory, and then displays the program in a readable form. You can then edit it, and after the editing is done, the table is searched for the addresses of the commands named, and only they are stored in memory. This is why it takes a long time to edit a long User RPL program.

That is OK, as long as all the commands have names. Guess what? There are over two thousand commands without names. This is the distinction between User and System RPL. User RPL, the language described in the manual (the « » language), can only access the named commands. (Actually, it can access the unnamed commands via the command `SYSEVAL`, as long as you know the address of the command. But this is not efficient, except for an occasional use). System RPL can access all commands.

Because of that, System RPL programs cannot be edited directly. Special tools are needed for that. On Appendix A you will find information about the available tools for writing System RPL programs.

Programming in System RPL is worth all the work you have. It is much more powerful and faster, because it does no error checking. In System RPL, the programmer must be sure that no error occurs, otherwise a crash might happen. For example, if a command requires two arguments in the stack and they are not there, or are not of the type the function requires, a warmstart or even a memory loss could happen. Fortunately, there are commands for checking if there are enough arguments, for their types, and for some other possible error conditions. The difference is that you probably just need to check if all arguments are present once, when the program starts. You do not need to repeat the check later. In User RPL, all commands have error checking, so tests are done unnecessarily, slowing the program.

One more question: if the commands do not have names, how can you program in System RPL? All commands have address, so you can call the address directly, using a `PTR <address>` command, and whatever is at that address will be executed. But there is an easier way.

The commands *have* names. The names simply are not stored in the HP48. But the HP48 design team has given them names, and they are stored somewhere (in the tools for creating System RPL programs). You write a program using those names, and then the System RPL compiler searches the names in the tables, and converts them to addresses. This is called compiling or assembling. Some tools can also do the opposite: convert the addresses into command names. This is called decompiling or disassembling.

Some of the commands are classified as “supported”. They are guaranteed to stay at the same memory location in all ROM versions of the calculator, i.e., their address is not going to change, so programmers can use them safely. But there are commands that are classified as “unsupported”. Programmers must use call them using their address (`PTR xxx`), because they are not listed in the command table of the compiler. But the address could be different in each ROM version, so the program may not work correctly in other ROM versions, and could even crash the machine. However, since version “R” has been released, in 1993, no more have been released, so you can use them because it is very unlikely that a new version is going to be released now.

1.1 Your first System RPL program

Let us create a very simple System RPL program, and explain it in detail. The program will calculate the area of a sphere, given the radius in the stack. See Appendix A for information on how to compile it. If you downloaded the examples file, you will find it with the name `FIRST`.

```
::
  CK1NOLASTWD      ( check if there is an argument )
  CK&DISPATCH1    ( check if it is a real number )
  real ::          ( if it is )
    %2 %^          ( square the radius )
    %PI            ( put PI in the stack )
    %*             ( and multiply )
  ;
;
```

Before we start analyzing it, on note: in System RPL, the case is relevant, so `pi` is different from `PI`, which is different from `pI`. Be careful when typing. Also, everything between `()`'s is considered a comment. (The HP Tools requires a whitespace after the leading open parenthesis). Lines that have a `*` in the first column are also considered comments.

The first line contains the start of secondary (program) marker, `::` (`DOCOL` is its name). The end marker is `;` (`SEMI`).

Following, there is the command `CK1NOLASTWD`. This command checks if there is one argument in the stack, and if there isn't generates a "Too Few Arguments" error. The next command, `CK&DISPATCH0`, checks the argument type and can do different things for different argument types. Our program only supports one argument type: real numbers. If any other argument type is entered, a "Bad Argument Type" error will be produced. Argument checking is described in detail on Chapter 4.

Following, there is the code to do if the argument is a real number. Note that the code is between `::` and `;`. This is because only one object is expected after the argument type. A secondary (sub-program) is only one object (with other objects inside it), so if we want to evaluate more than one object, they must be included in a secondary.

The rest of the program is very simple. The number two is put in the stack, and the radius (entered by the user) is raised to that power. There is a command which squares the real number in level 1, but it is unsupported, so I decided to make the program longer, but safer and more readable. You can replace `%2 %^` for `PTR 1B47B`, the address of that command. This would save 2.5 bytes.

To end, π is put in the stack, and the squared radius is multiplied by it. The stack now contains the area.

This program is 25 bytes long (using the unsupported command), opposed to the 20 of the User RPL program `« SQ π * ->NUM »`. However, the User RPL version took 0.0156 seconds to calculate (with radius 1). The System RPL took only 0.0019 seconds.

System RPL compilers (see Appendix A) support the following structure: `DEFINE <name> <text>` (everything must be in one line only). Whenever `<name>` is found in the code, it will be replaced by `<text>`. For example, the program

```
::
  DEFINE 3DUP DUPDUP DUP
  CK1
  3DUP
;
```

is equivalent to

```
::
  CK1
  DUPDUP
  DUP
;
```

This is not part of the language, it is just a “help” compilers give you. Using this may save a lot of typing, and can also make your code more readable, like in the example on Chapter 5.

Chapter 2

Object Types

As we have seen before, the basics of programming in User RPL and in System RPL are the same: you put objects in the stack, call a function that takes those objects as arguments, and the results are put back in the stack.

There are several types of objects that can be put in the stack. These go from the simplest ones, like numbers or strings, to some more complicated, like symbolics (algebraic expressions) and some even more complicated, like ROM pointers (indicators to the location of a library command).

Now we will see how to create and deal with the various objects supported.

2.1 Binary Numbers

Binary numbers are the objects you will see more often. They are not the user-level binary integers; those are actually hexadecimal strings. These system-level binary integers (or bints, for short) are objects which are not directly accessible to the user. If you happen to have one in the stack, they show like `<10h>`. Try this: enter the following number in the stack (triple check if it is right): `#408Fh`. Now, type `SYSEVAL` and press ENTER. You should get `<10h>` in the stack, or perhaps `<16d>`, if you are in decimal mode. Internally, they are always in hexadecimal mode.

Bints are the objects you will see more often because most commands that require a numeric argument need that argument to be in the form of a binary integer, as opposed to the real numbers needed by user functions. So, they should be easy to create. And, indeed, they are. You can put one in stack just by entering it on your program (in decimal form). But that is not recommended. First, because you can also put a real number in stack by just entering it in the same way (we will see later how to differ one from another). So, it is a good idea to use the following structure: `# <hex>`. This way, you can be sure you will get a binary number, and your code is clearer. Unfortunately, you must use the hexadecimal representation.

The second reason is that there are several “built-in” binary numbers. You can put one of these in the stack by just calling their address. Since almost all of them are supported, to get `#6h` in the stack, you just use the word `SIX`. The main advantage is that if you enter `# 6`, it takes five bytes. The word `SIX`, as all other commands, take only 2.5 bytes. Some words put two or even three bints in the stack, so the savings are even greater. The list of built-in bints is on Chapter 17.

The four basic operations with bints are `#+`, `#-`, `#*` and `#/`. There are also many others, which are listed on Chapter 17.

Here is an example of program that just put three real numbers in the stack, using the three methods:

```
::
13      ( <13d> or <Dh> )
# D      ( the same, using preferred method )
THIRTEEN ( in this case, this method is shorter )
;
```

2.2 Real numbers

Real numbers can be created in two ways. The first is by just entering them, without any prefix. But this method can also be used to create bints. So how does the compiler know when you want a real number and when you want a bint? If the number includes a radix and/or an exponent, then it is a real number; otherwise, it is a bint.

But again, the preferred method is to use the structure `% <dec>`. This way, you will surely get a real number, and the code is more readable.

As for bints, there are also many built-in real numbers. They are listed in Chapter 18.

The basic operations using real numbers are `%+`, `%-`, `%*`, `%/` and `%^`. There are many other, which are listed in Chapter 18.

There is also another kind of real number, which is not accessible to the user and to User RPL programs. They are the Extended (or Long) Real Numbers. They work like normal real numbers, with two differences: they have a 15-digit precision opposed to the 12-digit of the normal real numbers, and their exponents are in the range from -50000 to 50000.

Extended real numbers are created using `%% <dec>`. If you happen to get one in the stack, the only thing that you will see is `Long Real`. The basic operations are the same, except that they are prefixed with `%%` instead of `%`. Let me make one thing clear, if it is not already: in User RPL, `+` adds any kind of object, for example real numbers, user binary integers (hexadecimal strings as we will see later), adds elements to lists, etc. In System RPL, the word `%+` only works for two real numbers. To add two binary integers, you must use `#+`. To add extended reals, the word is `%%+`. If you call a function with the wrong arguments, there is a possibility that your system will crash.

To convert from a real number to an extended real number, the command is `%>%%`. Similarly, the opposite function is `%%>%`. To convert from a bint to a (normal) real number, the function is `UNCOERCE`, and the opposite function is `COERCE`.

2.3 Complex numbers

Complex numbers are created with the following structure: `C% <real> <imag>`. The real and imaginary parts are real numbers, in decimal form. If you have the real and imaginary parts in the stack, the word

`%>C%` will create a complex number from them. The command `C%>%` takes a complex number and returns the real and imaginary parts.

There are also the Extended Complex Numbers, which are not accessible to the user. They are complex number whose real and imaginary parts are extended reals. They are created using `C%% <real> <imag>`, where the real and imaginary parts are extended reals. They show in the stack as `Long Complex`.

In Chapter 19, there is a list of all the commands related to complex numbers, including mathematical operations.

2.4 Characters

The characters are another data type not available to the user. They are a string with only one character. You create them with `CHR <char>` or using one of the many built-in characters (listed on Chapter 20), but in the stack they show as `Character`. To convert a character to a bint, use `CHR>#`. The bint returned is the code for the character. The opposite function is `#>CHR`.

2.5 Strings

Strings are created with `$ "<string>"`, or just `"<string>"`. There are some built-in strings, listed in Chapter 20. Or you can convert a character into a string, with the command `CHR>$`.

Two useful and simple functions which deal with strings are `LEN$` and `&$`. The first returns the length (in bytes) of a string as a bint, and the second concatenates two strings. To get a substring, i.e., part of a string, use the function `SUB$`. It expects three arguments: the original string, the start position (a bint) and the end position (also a bint). Everything between the start and end characters (inclusive) will be returned. And another function is `POS$`, which searches a string (in level three) for a character or string (in level two), starting from a specified position (a bint, in level one). The position of the first occurrence of the search string in the string is returned (as a bint) to level one. If it could not be found, `#0` is returned. There are also many other functions, see Chapter 20 for a list.

2.6 Hexadecimal strings

Hexadecimal strings are the “base” numbers the user can access. They are created using the structure `HXS <len> <hexbody>`. `len` is the length of the string, in hexadecimal form, and `hexbody` is the actual contents of it. The tricky part about it is that because of the HP internal architecture, you must enter the contents in reverse order. To get, for example, the hex string `#12AD7h`, you must enter `HXS 5 7DA21`. To get `#12345678h` use `HXS 8 87654321`.

To convert an hex string to and from a bint, use the commands `HXS>#` and `#>HXS`. To convert an HXS to and from a real number, use `#>%` (or `HXS>%`) and `%>#`.

See Chapter 21 for more commands related to hex strings.

2.7 Identifiers

Identifiers are names of objects in memory. To the user, they appear in the stack between `' '`. In System RPL, they are created with `ID <name>`. Another difference is that always that when you use the above structure, you do not get the identifier in the stack. It is always evaluated. So, if variable `anumber` contains 123.45 and you put somewhere in your program `ID anumber`, the stack will contain 123.45. To put an id to the stack, use `' ID <name>`. As you will see on Chapter 6, the command `'` puts the object after it in the stack.

You can convert a string to an id using `$>ID`, and the opposite transformation is archived with `ID>$`.

There is also another kind of identifiers: the temporary identifiers, or lams. These are used when creating local variables, and you will learn about them later in Chapter 5. They are created with `LAM <name>`, and work like normal ids.

2.8 Tagged objects

To create a tagged object, use the structure `TAG <tag> <object>`. Tag is a string, and object can be anything. To create `x: (2.3;3.5)`, for example, you would use `TAG x C% 2.3 3.5`. An object can have multiple tags, but there is not much use for that.

The word `>TAG` creates a tagged object, given the object (in level two) and a string representing the tag (in level one). `%>TAG` works the same way, but tags an object with a real number. `ID>TAG` tags an object with an identifier. To remove all tags from an object, call `STRIPTAGS`.

A few more commands related to tagged objects are listed on Chapter 22.

2.9 Lists

Lists are very easy to create: start the list with `{`, and end it with `}`. Inside, put as many objects as you wish, of any kind. One difference from User RPL is that if you put an id in a list, you will get its contents instead. So if you want the id itself, add `'` before. As in User RPL, you can have lists inside lists.

Lists are one kind of *composite object*. As the name says, they are composed of other objects. Other kinds are the secondaries (programs) and symbolics (algebraics). The commands described below for list also work for the other kinds of secondaries.

To concatenate two composites, put them in the stack and use `&COMP`. To add just one object to the head (beginning) or tail (end) of a composite, first put the composite in the stack, then the object, and call `>HCOMP` or `>TCOMP` respectively. To get the length of the composite (the number of objects, as a bint), just put in level one and use the word `LENCOMP`. To get one object of a composite, put the composite in level two, its number in level one (as a bint, naturally), and run `NTHELCOMP`. If the number were out of range, you would get a `FALSE`, otherwise the object and `TRUE`. `NTHCOMPDROP` is the above entry, followed by `DROP`. And to get part of a composite, use the function `SUBCOMP`. This function takes in level three the composite, in level two the start position (guess what? a bint) and in level one the end position (from now on, unless otherwise noted, all numeric arguments are bints). You will get a composite (of the same type, obviously) with the elements between the start and end positions, inclusive. This function checks if the numbers are not out of range (if they are, a null composite is returned. The same happens if the end position is greater than the start position).

Other commands can be found on Chapter 25.

2.10 Arrays

In user RPL, arrays can be only of real or complex numbers. In System RPL, you can have arrays of anything, even arrays of arrays.

If you use HP Tools or GNU Tools (see Appendix A), you can create an array using this structure: `ARRY m n [objs]`. This will create an $m \times n$ array. The objects are specified in order, from left to right and from top to bottom. All objects must be of the same type, and they must be actual objects, not pointers to objects. That means you cannot use built-in objects, you must use `% 2` if you want to have the number two on the array, and not `%2`.

Here is an example of a 3×3 array of real numbers. Note that there is only one pair of delimiters.

```
::
  ARRY 3 3 [ % 11 % 12 % 13
             % 21 % 22 % 23
             % 31 % 32 % 33 ]
;
```

If you use JAZZ (see Appendix A), you cannot create arrays this way. This structure is not supported. A tip to create an array in JAZZ is to create the array in the stack if possible (i.e., it is an array of real or complex numbers) or using the HP Tools. Then, use the command `DIS` in JAZZ to decompile the array into a form recognized by JAZZ. Insert that code in your program.

You can also create an array of (normal, not extended) real or complex numbers by putting them in order in the stack, and entering a list representing the dimensions of the array (real numbers, not bints) in level one. Then run `XEQ>ARRAY`. This function does error checks to ensure there are enough arguments and if they are of the supported types.

The function `ARSIZE` returns the number of elements in an array. You can get the dimensions of the array with `DIMLIMITS`, which returns a list of bints representing the array dimensions. To get one element of an array, put the element number in level two, the array in level one, and run `GETATELN`. You will get the element and `TRUE` if it was found or only `FALSE` if the element does not exist. More array functions are listed on Chapter 23.

There is also another kind of array: the linked arrays. Linked arrays are like normal arrays, except that they have a table with pointers to all the objects in the array. This makes access to array elements faster, because when you need to access one object in the linked array, the only thing necessary is to read the pointer to that object in the table, and go directly there. With normal arrays, a sequential search is necessary.

2.11 Units

Units are another kind of composite objects. They are not really difficult to create, just laborious.

Units start with `UNIT` and end with `;`. Inside, there are commands to define the unit. The best way to understand how a unit is created is by disassembling them. The unit object `3_kg*m^2/(A*s^3)` was created using

```
::
UNIT
%3      ( 1:3 )
CHR k   ( 2:3 1:k )
$ "g"   ( 3:3 2:k 1:g )
umP     ( 2:3 1:kg )
$ "m"   ( 3:3 2:kg 1:m )
%2      ( 4:3 3:kg 2:m 1:2 )
um^     ( 3:3 2:kg 1:m^2 )
um*     ( 2:3 1:kg*m^2 )
$ "A"   ( 3:3 2:kg*m^2 1:A )
$ "s"   ( 4:3 3:kg*m^2 2:A 1:s )
%3      ( 5:3 4:kg*m^2 3:A 2:s 1:3 )
um^     ( 4:3 3:kg*m^2 2:A 1:s^3 )
um*     ( 3:3 2:kg*m^2 1:A*s^3 )
um/     ( 2:3 1:kg*m^2/[A*s^3] )
umEND   ( 1:3_kg*m^2/[A*s^3] )
;
```

As you saw, creating units is done using the words `um^`, `um*`, `um/` and `umP`. The meaning of the first three ones is easy to guess. The last is used to create prefix operators (kilo, mega, mili, etc.). First enter the prefix as a character or string, and then the unit name (all operations take unit names as characters or strings). Run `umP` and the prefixed unit is created. Then call the other functions as needed. To end a unit, use `umEND`, which joins the number (entered first) to the unit part. This code could be made shorter if built-in characters and strings (listed on Chapter 20) were used.

Several operations can be done with units. The complete list is on Chapter 24. The most important are UM+, UM-, UM*, UM/ and UFACT, whose meanings are obvious; UMCNV, which works like user word CONVERT; UMSI, equivalent to UBASE and U>nbr, which returns the numeric part of a unit.

2.12 Symbolics

Symbolics, algebraic expressions, are another type of composite objects. They are created in a similar manner to units. They are delimited by SYMBOL and ;. Let us see an example of disassembly of the algebraic expression

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} :$$

```
::
SYMBOL
  ID x  ( 1:x )
  ID b  ( 2:x 1:b )
  xNEG  ( 2:x 1:-b )
  ID b  ( 3:x 2:-b 1:b )
  %2    ( 4:x 3:-b 2:b 1:2 )
  x^    ( 3:x 2:-b 1:b^2 )
  %4    ( 4:x 3:-b 2:b^2 1:4 )
  ID a  ( 5:x 4:-b 3:b^2 2:4 1:a )
  x*    ( 4:x 3:-b 2:b^2 1:4*a )
  ID c  ( 5:x 4:-b 3:b^2 2:4*a 1:c )
  x*    ( 4:x 3:-b 2:b^2 1:4*a*c )
  x-    ( 3:x 2:-b 1:b^2-4*a*c )
  xv    ( 3:x 2:-b 1:v[b^2-4*a*c] )
  x+    ( 2:x 1:-b+v[b^2-4*a*c] )
  %2    ( 3:x 2:-b+v[b^2-4*a*c] 1:2 )
  ID a  ( 4:x 3:-b+v[b^2-4*a*c] 2:2 1:a )
  x*    ( 3:x 2:-b+v[b^2-4*a*c] 1:2*a )
  x/    ( 2:x 1:-b+v[b^2-4*a*c]/[2*a] )
  x=    ( 1:x=-b+v[b^2-4*a*c]/[2*a] )
;
```

As you have seen, creating symbolics is very similar (and as laborious) to creating units. The variables are ids, and the functions are preceded by a lowercase x.

There are many functions that deal with symbolics (and most of them are not supported). The list is on Chapter 27.

Chapter 3

Stack operations

In System RPL, using the stack is almost the same as in User RPL. The basic operations are the same, except for little changes in the name: DUP, 2DUP (equivalent to User APL's DUP2), NDUP (DUPN), DROP, 2DROP (DROP2), NDROP (DROPN), OVER, PICK, SWAP, ROLL, UNROLL (ROLLD), ROT and DEPTH.

All commands that require or return a numeric argument (ROLL, UNROLL, PICK and DEPTH) use bints and not real numbers.

There is also new functions: UNROT, which is a ROT in the other way, i.e., 3 ROLL; and reverse, which takes n objects and a bint representing this count, and reverses their order. For example the program

```
:: %1 %2 %3 %4 FOUR reversym ;
```

when run will reverse the order of the reals, leaving 4, 3, 2 and 1 in the stack.

There are also many commands that do two or even three operations in sequence. The complete list can be found on Chapter 29. Here is a list of the most used ones:

DUP	DROP	SWAP	OVER	ROT	UNROT
DUPDUP	DROPDUP	SWAPDUP	OVERDUP	ROTDUP	UNROTDUP
—	2DROP	SWAPDROP	—	ROTDROP	UNROTDROP
—	DROPSWAP	—	OVERSWAP	ROTSWAP	UNROTSWAP
—	DROPOVER	SWAPOVER	—	ROTOVER	UNROTOVER
DUPROT	DROPROT	SWAPROT	—	—	—
DUPUNROT	—	—	OVERUNROT	—	—
—	DROPSWAPDROP	—	—	—	UNROTSWAPDRO
—	—	SWAPDROPDUP	—	—	—
—	—	SWAPDROPSWAP	—	ROTDROPSWAP	—
—	—	—	—	ROT2DROP	UNROT2DROP
—	—	SWAP2DUP	—	ROT2DUP	—
DUP3PICK	—	SWAP3PICK	—	—	—
—	—	SWAP4PICK	—	—	—
—	—	—	OVER5PICK	—	—
—	—	SWAP4ROLL	—	—	—
DUP4UNROLL	—	—	—	—	—
—	—	—	—	ROTROT2DROP	—

Chapter 4

Checking arguments

In System RPL, it is very important to check if all arguments required by a command are present, and if they are of a valid type. In User RPL, you don't have to worry about this: it is done automatically. In System RPL, very few commands do that, so this is left for the programmer. This may seem at first a disadvantage, but it is in fact an advantage: you just need to check the arguments once, in the beginning of the program. This generates a fast code, differently from User RPL the arguments are checked in every command.

There are two kinds of checks: for number of arguments, and for argument type.

4.1 Number of arguments

To check for a specific number of arguments, use one of the following commands. They check if there are enough arguments in the stack, and produce a "Too Few Arguments" error if not.

Command	When to use
CK0, CK0NOLASTWD	No arguments required
CK1, CK1NOLASTWD	One argument required
CK2, CK2NOLASTWD	Two arguments required
CK3, CK3NOLASTWD	Three arguments required
CK4, CK4NOLASTWD	Four arguments required
CK5, CK5NOLASTWD	Five arguments required

Each word `CK<n>` "marks" the stack below the `<n>`th argument, and saves a copy of the arguments, if argument recovery is enabled. In case an error happens, the stack is cleared by the marked level, and if argument recovery is enabled, the saved arguments are restored.

The `CK<n>` words save the name of the command in which they are executed, and if an error happens, that name is displayed. These words must be the first object in a program. Also, they should only be used in libraries, because if they are not part of a library, and if there is an error, it will be shown as something like `"XLIB 1364 36 Error:"`. To avoid this, use `CK<n>NOLASTWD`, which does not save the name of the command, thus is excellent for programs that are not part of a library. These words may be used in the middle of the program. Normally this should be done only after getting input from the user.

If your program uses a stack-defined number of arguments, use the words `CKN` or `CKNNOLASTWD`. These words first check for a real number in level

one, and then for the specified number of objects in the stack. The stacked is marked at level two, but only the real number is saved in LAST ARG.

4.2 Argument type

The words CK&DISPATCH1 and CK&DISPATCH0 are used to do different actions based on different types of arguments. They are used like this:

```
...
CK&DISPATCH1
  #type1  action1
  #type2  action2
  #type3  action3
  ...
  #type<n> action<n>
;
```

The type/action pairs are terminated by a SEMI (;).

This is how CK&DISPATCH0 works: it checks if the stack matches the definitions in #type1. If it does, action1 is executed, after which program execution resumes after SEMI. (Each action must be a single object, so if you want to do more than one action, they must be included in a secondary, i.e., between :: and ;). If the type definition does not match the stack, then type2 is checked, and so on. If no match was found, a “Bad Argument Type” error is generated.

The difference between CK&DISPATCH0 and CK&DISPATCH1 is that the latter, after completing the first pass, strips all the tags from the stack objects, and does a second pass. Only after the second pass without a match the “Bad Argument Type” error is generated.

Each type definition is a bint like this: #nnnnnn. Each n is an hexadecimal number representing the object in one position of the stack, according to the table below. The first n represents the object in level five, the second in level four, and so on. This way, #00201 represents a complex number in level three, any object in level two and a real number in level one; #00096 represents a symbolic class in level two and an id in level one. There are also two-digit object type numbers, ending in F. Each time you use one of these, the number of arguments that can be checked is reduced. For example, #13F4F represents a real number in level three, an extended real in level 2 and an extended complex in level one.

Value	Argument	User type
0	Any object	
1	Real number	0
2	Complex number	1
3	Character string	2
4	Array	3, 4
5	List	5
6	Global name	6

Value	Argument	User type
7	Local name	7
8	Secondary	8
9	Symbolic	9
A	Symbolic class	6, 7, 9
B	Hexadecimal string	10
C	Graphics object	11
D	Tagged object	12
E	Unit object	13
0F	ROM Pointer	14
1F	Binary integer	20
2F	Directory	15
3F	Extended real	21
4F	Extended complex	22
5F	Linked array	23
6F	Character	24
7F	Code object	25
8F	Library	16
9F	Backup	17
AF	Library data	26
BF	Access pointer (GX only)	27
CF	External object 2	28
DF	External object 3	29
EF	External object 4	30

There are also the words CK<n>&Dispatch, where <n> is a number from one to five. These words combine CK<n> with CK&DISPATCH1.

4.3 Examples

By disassembling and studying built-in words, you can learn a lot. Not only about argument checking, but also about many other things. For example, here is the disassembly of the user command STO:

```
:: CK2&Dispatch
* 2:any 1:tagged
  THIRTEEN XEQXSTO
* 2:any 1:id
  SIX      :: STRIPTAGS12 ?STO_HERE ;
* 2:any 1:lam
  SEVEN    :: STRIPTAGS12 STO ;
* 2:any 1:symb
  NINE     :: STRIPTAGS12 SYMSTO ;
* 2:grob 1:program [PICT]
  # 000C8  PICTSTO
* 2:backup 1:real number
  # 009F1  LBLSTO
* 2:library 1:real number
  # 008F1  LBSTO
;
```

The command `STO` starts by checking if there are two arguments present, stores the arguments and the command `STO` for error handling, and then dispatches to one of the actions listed. If the level one object is a tagged object, `STO` executes the word `XEQXSTO`. For a global name, `:: STRIPTAGS12 ?STO_HERE ;` is executed. And so forth, until the last one. If none of the types can be matched, then an “Bad Argument Type” error will be generated.

The `TYPE` command provides an example of dispatching at a point other than the start of a command. Its argument count and argument type checking parts are separated so that the latter part can be called by other system words that do not want to mark the stack. Here is its disassembly:

```
::
  CK1
  :: CK&DISPATCH0
    real      %0
    cmp       %1
    str       %2
    array     XEQTYPEARRY
    list      %5
    id        %6
    lam       %7
    seco      TYPESEC ( 8, 18 or 19 )
    symb      %9
    hxs       %10
    grob      % 11
    TAGGED    % 12
    unitob    % 13
    rompointer % 14
    THIRTYONE % 20 ( # )
    rrp       % 15
    # 3F      % 21 ( %% )
    # 4F      % 22 ( C%% )
    # 5F      % 23 ( LNKARRAY )
    # 6F      % 24 ( CHR )
    # 7F      % 25 ( CODE )
    library   % 16
    backup    % 17
    # AF      % 26 ( Library Data )
    any       % 27 ( external )
  ;
  SWAPDROP
;
```

In this case, `CK&DISPATCH1` could have been used, because tagged objects are explicitly listed on the table. Since the last item on the list is `any`, so type 27 is returned for any other object not listed.

The object names are built-in bints. See Chapter 17 for a list of built-in bints.

Chapter 5

Local variables

System RPL local variables (also known as temporary or lambda variables) work in the same way as in User RPL. You assign values to them, and these values can be recalled or changed at any time. But there is one difference: in System RPL you can create unnamed local variables, which saves memory. But before learning how to create and use unnamed local variables, let us learn how to use normal, named ones.

5.1 Named local variables

Creating named local variables is very similar to creating temporary variables in User RPL. You have to create a list of local identifier (lams), and run the command `BIND`. To recall the contents of one of them, just enter its local identifier. To store a new value, put that value and the lam in the stack, and run `STO`. To remove the local variables from memory, use `ABND` (“abandon”). The code is not checked for matching `BIND/ABND`, so you may include them in different programs if you wish. But this also means you must be sure to have an `ABND` for each `BIND`.

Here is a little program that creates two local variables, recalls their contents and assigns new values for them (it is called `LAM1`):

```
::
  %2 %3
  {
    ' LAM firstVar
    ' LAM secVar
  }
  BIND      ( firstVar contains 2, and secVar 3 )
  LAM firstVar ( recall contents from firstVar )
  LAM secVar   ( recall contents from secVar )
  DUP
  ' LAM firstVar
  STO          ( store new contents in firstVar )
  %+           ( results 5 )
  ' LAM secVar
  STO          ( store sum in secVar )
  ABND         ( delete variables from memory )
;
```

5.2 Unnamed local variables

If there is *no chance* that a new temporary environment will be created after the one you are about to create, you can use unnamed local variables (actually, they have a name: null name). The above program could be rewritten using null named temporary variables this way (now called `LAM2`):

```
::
  %2 %3
  { NULLLAM NULLLAM }
  BIND
  1GETLAM
  2GETLAM
  DUP
  1PUTLAM
  %+
  2PUTLAM
  ABND
;
```

The numbering is in the same order as the stack. There are functions to recall and store directly for up to 22 variables (`1GETLAM` to `22GETLAM`). To access variables with numbers higher than 23, you `GETLAM`, which takes a bint representing the variable number and returns its contents; and `PUTLAM`, which takes an object and the variable number, and stores that object in the specified variable.

5.3 Suggestions

The RPL Manual (file `RPLMAN.DOC` of the HP Tools) suggests that you use `DEFINE` (see Chapter 1) to make the use of unnamed local variables more readable. For example, you could use

```
::
  DEFINE I%YR      1GETLAM
  DEFINE !I%YR     1PUTLAM
  DEFINE NPayments 2GETLAM
  DEFINE !NPayments 2PUTLAM
  ...
  { NULLLAM NULLLAM }
  BIND
  ...
  I%YR      ( Recalls contents from first var )
  ...
  NPayments ( Recalls contents from second var )
  ...
  !I%YR     ( Stores something in first var )
  ...
  ABND
;
```

When you are binding a great number of local variables, instead of entering the following code (which takes 67.5 bytes)

```
...
{ NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM
  NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM
  NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM
  NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM }
BIND
...
```

use this, which takes only 12.5 bytes, saving 55 bytes:

```
... NULLLAM TWENTYFOUR NDUPN {}N BIND ...
```

You can even replace {}N BIND for DOBIND, which instead of a list of names, takes all names in the stack plus the count to bind the variables. This saves 2.5 bytes more.

Or you can also use TWENTYFOUR ' NULLLAM CACHE. However, if you use this, an extra variable is created to hold the count, so you must add one to the variable positions of the previous examples.

If you use JAZZ or GNU Tools (see Appendix A), you can use the following structure to create local variables:

```
::
%2 %3
{{ A B }} ( no spaces between the brackets )
A          ( recalls contents of A )
B          ( recalls contents of B )
DUP
!A          ( stores new contents in A )
%+
!B          ( stores sum in B )
ABND       ( destroy variables )
;
```

The code above binds 2 into variable A and 3 into variable B. To recall the contents of any variable, just enter its name. To store a new value, use !<name>, <name>! or =<name>.

The structure above is just a shortcut. The code is actually compiled as:

```
::
%2 %3
' NULLLAM TWO NDUPN BIND
1GETLAM 2GETLAM DUP 1PUTLAM
%+ 2PUTLAM ABND
;
```

Chapter 6

Runstream control

In System RPL, the normal program flow is simple: all commands are executed in the order they are found in the program. However, there are some commands that change this flow. There are conditional and loops structures, which you will see in the following chapters, and there are words whose sole purpose is to interfere with the normal program flow. The complete list is on Chapter 34. Below, the most important are listed.

Word	Stack and action
'	<p>(→ ob)</p> <p>This command assumes the next object after it is not <code>SEMI</code>. Then, if the next object in the runstream is an object, then that object is put into the data stack, and the interpreter pointer is moved to the next object. If the next object is a pointer, its pointee is put into the data stack, and the interpreter pointer is also moved to the next object. Basically, it means that as long as the object following ' is not <code>SEMI</code>, it is put into the data stack, instead of being executed. Execution resumes at the object following that one. For example, the program <code>:: %3 %4 ' SWAP EVAL ;</code> is equivalent to <code>:: %3 %4 SWAP ;</code>.</p>
'R	<p>(→ ob)</p> <p>If the object pointed to by the top pointer on the return stack (i.e., the first element in the second body of the runstream) is an object, then that object is pushed into the data stack, and the pointer is advanced to the next object in the same composite. If the pointer points to an object pointer whose pointee is not <code>SEMI</code>, the pointee is pushed into the data stack, and the return stack pointer is also advanced. If the pointee is <code>SEMI</code>, then if the first element in the second body in the runstream is a pointer to <code>SEMI</code>, a null secondary is pushed into the data stack and the return stack pointer is not advanced. Rewriting this in a way a normal human can understand, this command does the following: the object after the next <code>SEMI</code> is put in the stack. Then, the execution continues after 'R. If the object following <code>SEMI</code> was another <code>SEMI</code>, then a null composite is put in the stack. For example, assuming <code>n1</code> contains 1, <code>n2</code> contains 2, <code>n3</code> contains 3 and <code>n4</code> contains 4, the following code will put in the stack, in order 'n3', 1, 2 and 4:</p> <pre>:: :: 'R ID n1 ID n2 ; ID N3 ID N4 ;</pre>

Word	Stack and action
ticR	<pre>(→ ob TRUE) (→ FALSE)</pre> <p>This works similarly to 'R. The difference is that an object was found, it is returned along with TRUE. If it could not be found (i.e., the object after SEMI was another SEMI), FALSE is returned. The above code, if ticR were used instead, would return 'n3', TRUE, 1, 2 and 4. The following code</p> <pre>:: :: ticR ID n1 ID n2 ; ;</pre> <p>would return FALSE, 1 and 2.</p>
>R	<pre>(:: →)</pre> <p>This command takes a composite as argument. The body of the composite is put into the runstream, just below the top one (i.e., a pointer to the body of the composite is pushed into the return stack). All that above can be simplified as: the objects following >R are executed, and in sequence the composite taken as argument. For example, the code below would put 3, 4, 1, 2 in the stack, in this order:</p> <pre>:: ' :: ID n1 ID n2 ; >R ID n3 ID n4 ;</pre>
R>	<pre>(→ ::)</pre> <p>Creates a program object from the composite body pointed to by the top return stack pointer, pushes it into the data stack, and pops the return stack. That means that a composite is created from the elements following the next SEMI (until the other SEMI) and put in the stack. Execution continues at the object following R>. For example, the code</p> <pre>:: :: R> ID n1 ID n2 ; ID n3 ID n4 ;</pre> <p>puts into the stack :: ID n3 ID n4 ;, 1 and 2. If it were changed to</p> <pre>:: :: R> EVAL ID n1 ID n2 ; ID n3 ID n4 ;</pre> <p>then the stack would contain 3, 4, 1, 2.</p>
R@	<pre>(→ ::)</pre> <p>The same as R>, except that the return stack is not popped. That means that the code above, rewritten using R@, would produce 3, 4, 1, 2, 3, 4.</p>
IDUP	<pre>(→)</pre> <p>Duplicates the top body in the runstream, i.e., the next objects until a SEMI are executed twice.</p>

Word	Stack and action
RDUP	<p>(→)</p> <p>Duplicates top return stack level, i.e., the rest of the composite above the actual is executed twice. For example, the code below would return 1, 2, 3, 4, 3, 4 to the stack:</p> <pre>:: :: ID n1 RDUP ID n2 ; ID n3 ID n4 ;</pre>
RDROP	<p>(→)</p> <p>Pops the return stack, i.e., the rest of the composite above the actual is skipped. For example, the code</p> <pre>:: :: ID n1 RDROP ID n2 ; ID n3 ID n4 ;</pre> <p>would put in the stack only 1 and 2.</p>
?SEMI	<p>(flag →)</p> <p>Exits the actual composite if the flag is <code>TRUE</code>.</p>
COLA	<p>(→)</p> <p>Executes the next object in the composite and skip all the rest. For example, the code below would put 1 in the stack.</p> <pre>:: COLA ID n1 ID n2 ID n3 ;</pre>
SKIP	<p>(→)</p> <p>Skips the next object, and executes the rest. The code below would put 2 and 3 in the stack.</p> <pre>:: SKIP ID n1 ID n2 ID n3 ;</pre>
?SKIP	<p>(flag →)</p> <p>If the flag is <code>TRUE</code>, skips the next object.</p>

Chapter 7

Conditionals

In System RPL, conditionals are a bit different from User RPL. The first difference is that in User RPL, a false is a zero; any other value is a true. In System RPL, a false is `FALSE`, and a true is `TRUE` (amazing!). These words just put the corresponding objects in the stack. All commands that do a test return either one. Words like `IF...THEN` take one of these.

If you need, you can convert a `TRUE` or `FALSE` to a (real) 0 or 1 with `COERCEFLAG`. There is not a dedicated function to do the opposite transformation, like `UNCOERCEFLAG`, but `%0<>` does that.

There are many commands that put `TRUE`, `FALSE`, or some combination of them in the stack. See the list on Chapter 33.

The Boolean operators are present too: `NOT`, `AND`, `OR` and `XOR`. There are combinations: `ORNOT` and `NOTAND`. Finally, `ROTAND` does a `ROT` and then `AND`.

7.1 Tests

The test words are commands which take one or more arguments and return either `TRUE` or `FALSE`, after doing some kind of comparison to them. The tests for each kind of object type are listed on the chapter of the reference dedicated to the object type. General tests and tests for object type can be found on Chapter 33.

The most important of them are `EQ` and `EQUAL`. Both take two objects and return a flag. The first checks if the objects are the same, i.e., occupy the same address in memory. The second checks if the prolog and contents are the same.

If you put a string in level one, and press `ENTER`, `EQ` and `EQUAL` will return `TRUE`. However, if you enter a string, and then enter again the same string, only `EQUAL` will return `TRUE`. This happens because the contents of the strings are the same, but they are different objects in memory, occupying each a different address in memory. They just happen to have the same contents.

7.2 Object type tests

Use one of the following words to check if the object in level one is of a specific type. The words that start with `DUP` or `D` make a copy of the object first.

Object	No copy	With copy
Real number	TYPEREAL?	DUPTYPEREAL?, DTYPEREAL?
Complex number	TYPECMP?	DUPTYPECMP?
String	TYPECSTR?	DUPTYPECSTR?, DTYPECSTR?
Array	TYPEARRAY?	DUPTYPEARRY?, DTYPEARRY?
Real array	TYPERARRY?	Not available
Complex array	TYPECARRY?	Not available
List	TYPELIST?	DUPTYPELIST?, DTYPELIST?
Global identifier	TYPEIDNT?	DUPTYPEIDNT?
Local identifier	TYPELAM?	DUPTYPELAM?
Symbolic	YPESYMB?	DUPTYPESYMB?
Hex string	TYPEHSTR?	DUPTYPEHSTR?
Grob	TYPEGROB?	DUPTYPEGROB?
Tagged	TYPETAG?	DUPTYPETAG?
Unit	TYPEEXT?	DUPTYPEEXT?
ROM Pointer	TYPEROMP?	DUPTYPEROMP?
Binary integer	TYPEBINT?	DUPTYPEBINT?
Directory	TYPEERRP?	DUPTYPEERRP?
Character	TYPECHAR?	DUPTYPECHAR?
Program	TYPECOL?	DUPTYPECOL?, DTYPECOL?

7.3 IF...THEN...ELSE

The conditionals of the type IF...THEN...ELSE can be created using the words `RPIT` and `RPITE`.

Word	Stack and action
<code>RPIT</code>	(flag ob1 → ?) If the flag is <code>TRUE</code> , ob1 is <code>EVALUATED</code> . Otherwise, it is <code>DROPPED</code> .
<code>RPITE</code>	(flag ob1 ob2 → ?) If the flag is <code>TRUE</code> , ob1 is <code>EVALUATED</code> and ob2 is <code>DROPPED</code> . If the flag is <code>FALSE</code> , ob2 is <code>EVALUATED</code> and ob1 is <code>DROPPED</code> .

However, there are also available prefix versions of those operations, which are more commonly used.

Word	Stack and action
<code>IT</code>	(flag →) If the flag is <code>TRUE</code> , the next object is executed, otherwise it is skipped.
<code>ITE</code>	(flag →) If the flag is <code>TRUE</code> , the next object is executed, and the second is skipped. If it is <code>FALSE</code> , the next object is skipped and the second is executed.

For example, the program

```
... ' ID MyProg %0= RPIT ...
```

will execute `MyProg` if there is a real number 0 in the stack. Its equivalent using postfix notation is

```
... %0= IT ID MyProg ...
```

The program below will output “Equal” if the two objects are equal, otherwise it outputs “Not equal”:

```
... $ "Equal" $ "Not equal" EQUAL RPITE ...
```

Its equivalent using postfix notation is

```
... EQUAL ITE $ "Equal" $ "Not equal" ...
```

7.4 CASE

The CASE words (there are combinations of `case` with other tests and commands) are a combination of `IT`, `SKIP` and `COLA`. The basic word, `case` takes a flag in level one. If the flag is `TRUE`, the next object is executed, and the rest of the current stream is dropped, like `COLA`. If the flag is `FALSE`, then the next object is skipped and execution continues after it, like `SKIP`. For example, the following code outputs a string representing the bint in level one.

```
::
  DUP #0= case $ "Zero"
  DUP ONE #= case $ "One"
  DUP TWO #= case $ "Two"
  ...
;
```

There are many words that are a combination of `case` with tests or other actions. The complete list is on Chapter 35. One of them is `OVER#=case`. Its action is as the name says. First, `OVER` is executed. Then, `#=`. Finally, `case`. This way, the code above could be rewritten this way:

```
::
  ZERO OVER#=case $ "Zero"
  ONE OVER#=case $ "One"
  TWO OVER#=case $ "Two"
  ...
;
```

Chapter 8

Loops

As in User RPL, there are two types of loops in System RPL: the indefinite loops and the definite loops. Indefinite loops are loops in which you do not know beforehand how many times it will be executed: it will repeat until a specific condition is met. They are created in a very similar manner to User RPL indefinite loops. Definite loops, on the other hand, are executed a number of times specified before its start. They are not created exactly like in User RPL, but their use is simple and more powerful. For example, you can change the number of times to run the loop while running it.

8.1 Indefinite loops

In System RPL, indefinite loops can be of three types (the third is useless). The first is the WHILE loop. It is created like this:

```
BEGIN
  <test clause>
WHILE
  <loop object>
REPEAT
```

This kind of loop executes `<test clause>`, and if the test is `TRUE`, `<loop object>` is executed, and the loop starts again. If the test returned `FALSE`, then execution resumes past `REPEAT`. If the first test returned `FALSE`, this loop would never be executed.

This loop requires `<loop object>` to be a single object. The HP Tools automatically inserts all the objects between `WHILE` and `REPEAT` in a composite. However, if there is only one object, this is not desired, so JAZZ does not do that.

The second type of indefinite loop is the UNTIL loop. It is created like this:

```
BEGIN
  <loop clause>
UNTIL
```

This loop is always executed at least once. The word `UNTIL` expects a flag. If it is `TRUE`, the `<loop clause>` is executed again. If it is `FALSE`, execution continues past `UNTIL`.

There is also a third type of indefinite loop:

```
BEGIN
    <loop object>
AGAIN
```

This loop has no test. To exit it, an error condition must happen, or a direct manipulation of the return stack.

8.2 Definite loops

Definite loops are created with `DO`. `DO` takes two bints from the stack, representing the stop and start values. The start value is stored as the current index, which can be recalled with `INDEX@`. The stop value can be recalled with `ISTOP@`. You can store a new value to one of them with `INDEXSTO` and `ISTOPSTO`, respectively.

`DO`'s counterparts are `LOOP` and `+LOOP`. `LOOP` increments by one the index value, and checks if the new value is greater than or equal to the stop value, exiting the loop if it is. Otherwise, the loop is executed again. `+LOOP` works similarly, incrementing the index by the bint in level one.

The standard form of a `DO` loop is

```
stop start DO <loop clause> LOOP
```

which executes `<loop clause>` for each index value from `start` to `stop-1`.

There are several words provided to be used with `DO` loops, like `ONE_DO`, whose meaning is obvious. The list in on Chapter 36.

Here is an example of a simple loop which outputs the bints `#1h`, `#2h`, `#3h` and `#4h` to the stack:

```
::
    FIVE ONE
    DO
        INDEX@
    LOOP
;
```

It could be changed to:

```
::
    FIVE ONE_DO
        INDEX@
    LOOP
;
```

Chapter 9

System and memory operations

The words listed below deal with variables, directories and system functions, like changing the angle mode. Only the most important ones are listed below. For a complete list, turn to Chapter 37 and Chapter 39.

9.1 Variables and directories

The basic equivalents to the user functions `STO` and `RCL` are the words `CREATE`, `STO` and `@`:

Word	Stack and action
<code>CREATE</code>	<pre>(ob id →)</pre> <p>Creates a variable with the name <code>id</code> and contents <code>ob</code>. An error occurs if the <code>ob</code> is or contains the current directory ("Directory Recursion"). Assumes that <code>ob</code> is not a primitive code object.</p>
<code>STO</code>	<pre>(ob id →)</pre> <pre>(ob lam →)</pre> <p>In the <code>lam</code> case, the temporary identifier is re-bound to <code>ob</code>. An error is returned if the <code>lam</code> is unbound. In the <code>id</code> case, <code>STO</code> attempts to replace the contents of the variable with <code>ob</code>. If a variable with that name was not found, a new variable is created. <code>STO</code> assumes that <code>ob</code> is not a primitive code object in the <code>id</code> case.</p>
<code>@</code>	<pre>(id → ob TRUE)</pre> <pre>(id → FALSE)</pre> <pre>(lam → ob TRUE)</pre> <pre>(lam → FALSE)</pre> <p>In the <code>lam</code> case, return the contents of the temporary variable along with <code>TRUE</code> or just <code>FALSE</code> if no temporary variable could be found with that name. In the <code>id</code> case does the same for global variables, starting from the current directory and working up through parent directories if necessary.</p>

One problem with `STO` and `@` is that if you give, for example, `SIN` as the (`ob`) argument, `STO` will copy the entire body of the program into a variable, and then `@` would recall the undecompileable program. Because of this, it is preferred to use `SAFESTO` and `SAFE@`, which work like `STO` and `@`, but automatically convert ROM bodies into XLIB names.

Some other words related are:

Word	Stack and action
?STO_HERE	<pre>(ob id →) (ob lam →)</pre> <p>This is the system version of user <code>STO</code>. Works like <code>SAFESTO</code>, but only stores in the current directory and does not over-write directories.</p>
SAFE@_HERE	<pre>(id → ob TRUE) (id → FALSE) (lam → ob TRUE) (lam → FALSE)</pre> <p>Like <code>SAFE@</code>, but searches only in the current directory.</p>
PURGE	<pre>(id →)</pre> <p>Purges variable specified by <code>id</code>, does no type check on stored object.</p>
?PURGE_HERE	<pre>(id →)</pre> <p>Like <code>PURGE</code>, but works only in current directory.</p>
XEQRCL	<pre>(id → ob) (id → error!)</pre> <p>System version of user word <code>RCL</code>: same as <code>SAFE@</code>, but errors if variable does not exist.</p>
REPLACE	<pre>(newob oldob → newob)</pre> <p>Like <code>STO</code>, but instead of the name of the variable takes the previous contents as argument. For example:</p> <pre>\$ "OLD" ' ID ABC STO (creates var ABC) \$ "NEW" ID ABC (2:NEW, 1:OLD inside ABC) REPLACE ID ABC (returns NEW)</pre> <p>Be sure that the object in level one is inside a variable, otherwise you will probably get a memory clear. But using <code>REPLACE</code> is worth the risk, because it is much faster than <code>STO</code>. Instead of using “<code>anobject ' ID avar STO</code>” use “<code>anobject ID avar REPLACE</code>”.</p>
CREATEDIR	<pre>(id →)</pre> <p>Creates an empty directory inside the actual. Does <code>?PURGE_HERE</code> first to delete the original.</p>
XEQPGDIR	<pre>(id →)</pre> <p>Purges a directory, making all possible checks before.</p>
XEQORDER	<pre>({ var1 var2 ... } →)</pre> <p>Orders the variables in current directory. Does checks first.</p>

Part II

Advanced RPL

Chapter 10

Error handling

When an error occurs in a System RPL program, what normally happens is that the program is aborted and the error message is shown at the top of the display. However, sometimes it is desired for the program to trap the error and if possible continue execution, or perhaps show that an error happened in a different way.

But sometimes, the programs need to *generate* an error. For example, if the user gave invalid input for the program, it should abort with a “Invalid Argument Type” error, instead of risking crashing the machine.

10.1 Trapping errors

You can intercept the execution of the error handling subsystem, i.e., trap an error generated by your program, using the following structure:

```
:: ... ERRSET <suspect object> ERRTRAP <if-error object> ... ;
```

It works like this: if the `<suspect object>` generates an error, the execution continues at `<if-error object>`. Otherwise, it continues past it.

The action of `<if-error object>` is completely flexible. Normally, it will handle the error and then continue or exit the program. The current error number can be recalled with `ERROR@`, and then your program can do different actions on different kinds of errors. The error messages and numbers can be found on Appendix B of the HP48G Series User Manual. Other functions related to error handling are on Chapter 32.

10.1.1 The protection word

Each temporary environment (see Chapter 5) and DO/LOOP environment (see Chapter 8) has a protection word. The purpose of this is to allow the error handling sub-system to distinguish which environments were created before the error trap, and which were created after. This way, all environments that were created after the error trap was set will be deleted. For example, consider the following code:

```
::  
...  
1LAMBIND  
...  
TEN ZERO_DO  
  ERRSET ::  
  ...
```

```

1LAMBIND
...
FIVE ONE_DO
    <suspect object is here>
LOOP
ABND
;
ERRTRAP ::
    <error handling>
;
LOOP
...
ABND
;

```

If an error is generated, then the error will be trapped. The inner DO/LOOP and temporary environments will be deleted, thanks to the protection word.

The word `ERRSET` increments the protection word in the topmost temporary environment and topmost DO/LOOP environment. Thus, these environments now have a non-zero protection word. (The words `DO` and `BIND` had initialized the protection word to zero).

The words `ERRTRAP` and `ERRJMP` delete temporary and DO/LOOP environments (from the innermost to the outermost) until, in both cases, they find one with a non-zero protection word (which is then decremented). These environments were the ones that already existed before the setting of the error trap. This way, all environments created after the setting of the trap are deleted.

10.2 Generating errors

The error handling sub-system is invoked by the word `ERRJMP`. If an error trap was set, the error handler will be executed. If none was set, then the default one will be run (i.e., the error will be shown on the status line, together with a beep, etc.).

Normally, before calling `ERRJMP`, you must define which error you want to generate (most of the time when you generate errors, they are to be handled by the default error handler – such as displaying “Bad Argument Value” if the input is invalid, which is better than a crash). This can be done with the word `ERRORSTO`. It expects a bint as argument: the number of the error. The errors are listed on Appendix B of the HP48G Series User Manual.

There are some words that automate this process, automatically generating some common errors. They are listed on Chapter 32.

Chapter 11

Keyboard control

A System RPL program can get input from the user in five different ways:

- From the stack;
- Waiting keystrokes from the keyboard;
- Using the internal `INPUT`
- Using the internal `INFORM`;
- Setting up a Parameterized Outer Loop.

You have already seen how to get input directly from the stack. Using `InputLine`, `ParOuterLoop` and input forms will be seen on the following chapters. So, in this chapter you will learn how to read keystrokes from the keyboard.

11.1 Key locations

In User RPL, key representations have the form `%rc.p`. In System RPL, they are represented by two binary integers: `#KeyCode`, which goes from one to 49, and represents each key, in order, from left to right and top to bottom; and `#Plane`, which represents the modifier states, according to the table below:

#Plane	Modifiers	#Plane	Modifiers
ONE	None	FOUR	Alpha
TWO	Left-shift	FIVE	Alpha, left-shift
THREE	Right-shift	SIX	Alpha, right-shift

You can convert from one mode to another using:

```
Ck&DecKeyLoc ( %rc.p → #KeyCode #Plane )
```

```
CodePl>%rc.p ( #KeyCode #Plane → %rc.p )
```

11.2 Waiting for a key

The best function used to wait for a key is `WaitForKey`. This word puts the HP48 in a low-power state and waits until a key is pressed. It then returns the key code and plane. There are other words, which can be found on Chapter 40.

Chapter 12

Using InputLine

The word `InputLine` is the system equivalent to the user word `INPUT`. Its use is similar, and it does the same:

- Displays a prompt in the top of the screen;
- Starts the keyboard entry modes;
- Initializes the edit line;
- Accepts input until ENTER is pressed;
- Parses, evaluates, or just returns the user input;
- Returns `TRUE` if it was exited by ENTER or `FALSE` if aborted by ON/CANCEL.

The stack must contain the following parameters:

Name	Description
\$Prompt	The prompt to be displayed during input.
\$EditLine	The initial edit line.
CursorPos	The initial cursor position, specified as a binary integer character number or a two-element list of binary integer row and column. In both cases, #0 represents the end of edit line, row or column.
#Ins/Rep	The initial insert/replace mode: <ul style="list-style-type: none"> • #0 current mode • #1 insert mode • #2 replace mode
#Entry	The initial entry mode: <ul style="list-style-type: none"> • #0 current entry plus program entry • #1 program/immediate entry • #2 program/algebraic entry
#Alphalock	The initial alpha mode: <ul style="list-style-type: none"> • #0 current mode • #1 alpha enabled • #2 alpha disabled
ILMenu	The initial menu, in the format specified in section 14.5.
#ILMenu	The initial menu row number (normally ONE).
AttnAbort?	A flag: <ul style="list-style-type: none"> • <code>TRUE</code> CANCEL aborts the input • <code>FALSE</code> CANCEL just clears the edit line
#Parse	How to process the edit line: <ul style="list-style-type: none"> • #0 return edit line as a string • #1 return edit line as a string <i>and</i> a parsed object • #2 parse and evaluate edit line

If `AttnAbort?` is `TRUE`, if the user presses `CANCEL` the edition is aborted. If it is `FALSE`, `CANCEL` just clears the edit line. If it was already empty, then it aborts the edit.

Depending on the value of `#Parse`, different values are returned, according to the table:

#Parse	Stack	Description
#0	<code>\$Editline TRUE</code>	Edit line only
#1	<code>\$Editline obs TRUE</code>	Edit line and parsed object(s)
#3	<code>ob1 ... obn TRUE</code>	Resulting object(s)
	<code>FALSE</code>	<code>CANCEL</code> pressed to abort

12.1 An example

Here is an example of `InputLine`, which prompts for your name, and if the edition was not aborted, displays it.

```
::
$ "Your name:"      ( prompt )
NULL$              ( initial edit line )
#ZERO#ONE          ( cursor at end, insert mode )
ONEONE             ( prog/immed mode, alpha enabled )
NULL{ }            ( no menu )
ONE                ( menu row )
FALSE              ( CANCEL clears )
ZERO              ( returns string )
InputLine
NOT?SEMI           ( exit if FALSE )
$ "Your name is "
SWAP&$             ( concatenate string & name )
CLEARLCD           ( clear display )
DISPROWL           ( display string on 1st line )
SetDAsTemp         ( freeze display )
;
```

Chapter 13

Creating input forms

In User RPL, creating input forms is not one of the easiest tasks to do. As expected, in System RPL it is even more difficult. But there are two advantages: in User RPL you can only have text fields, in System RPL you can have all four kinds of fields (see below). And in System RPL they have the extra advantage of being *fast*.

There are four kinds of fields: data fields (DF), the normal user type input fields; extended data fields (EDF), which allow you to enter text or select a variable; list fields (LF), which have a pre-determined set of values; and check fields (CF), which you can select yes or no, for example.

The command `DoInputForm` needs the following arguments:

Parameter	Description
label1 ... labeln	Label definitions
field1 ... field2	Field definitions
#labels	Number of labels
#fields	Number of fields
MessageHandler	Normally 'DROPFALSE
"Title String"	Title string

13.1 Label definitions

Each label definitions is three arguments:

Parameter	Description
"label"	Label string
#x_offset	X coordinate
#y_offset	Y coordinate

The string will be displayed (with the small font) on the specified coordinates of the screen. The coordinates are two bints. The upper-left corner of the screen has the coordinates $x=0$ and $y=0$, and these values increase as the position goes down or right.

13.2 Field definitions

Each field definition is thirteen arguments:

Parameter	Notes
MessageHandler	Normally 'DROPFALSE
#x_offset	X coordinate
#y_offset	Y coordinate (normally label Y coordinate + 2)
#Length	Length of field
#Height	Height of field (usually NINE)
#FieldType	Specified filed type
#AllowedTypes	List of user types for DF/EDF, or MINUSONE for LF/CF
Decompile	See below
"HelpString"	Help string
ChooseData	See below.
ChooseDecompile	See below.
ResetValue	Reset value: DF/EDF: any/MINUSONE; LF: { "label" foo }; CF: TRUE/FALSE
RnitValue	Initial value, same as above

The message handler is normally specified as 'DROPFALSE, which means the programs does not hande what the user enters.

The x and y positions specify where the field will appear. They work similarly to the x and y positions of label definitions. Then length are also two binary integers, which specify the size of the field.

The field type is a bint which defines the field type:

Decimal value	Type
1	Text fields: DF or EDF
3	Algebraic fied: like DF, but automatically inserts ticks (').
12	List field
32	Check field

The allowed types is a list of system binary integers, where each represent a object type allowed for the field. The list of object types and numbers can be found on Chapter 4. If the field is a LF or CF, specify MINUSONE instead.

The decompile object can either be a secondary that takes an object and returns a string, or a bint. Here is an example of secondary that could be used for decompiling:

```
' ::
  DUP MINUSONE EQUAL casedrop NULL$
  DUPTYPECSTR? ?SEMI EDITDECOMP$
;
```

It first checks if the field is empty (i.e., its value is `MINUSONE`, in this case a null string is returned. Then, if the object is not already a string, the function `EDITDECOMP$` is called for changing the object into a string.

You can also specify a binary integer, each bit represents something different:

Bit	Meaning if set
0	No decompile – only expects strings
1	Decompile object to have stack appearance (using current number settings)
2	Decompile object to have stack appearance (but using STD mode)
3	Gets first character of a string (expects only strings)
4	Get first object of composite and decompile it
5	Get second object of composite and decompile it

Normally, for data and extended data fields, it is desired to display the object, if it is a number, using STD mode, so you would set only bit two. This results in 2^2 , or `FOUR`. When using list fields, most of the time this value will be set to `SEVENTEEN`, which is $2^0 + 2^4$. This means that each object in the choose box is a list. The first object is a string, which will be shown. The meaning of the others will be explained in the description of the `ChooseData` value. For check fields, specify `MINUSONE`.

The help string is just a string that will be shown above the menu bar when the field is selected.

The choose data field is only used for extended data and list fields. Other types should specify `MINUSONE` as this parameter. For EDFs, it is a string that will display in the title of the choose box of the variables. The current directory name will be appended, so normally it is a string like “Reals in ”, which will then show as something like “Reals in HOME”.

For LFs, it is a list with all the possible choices. The format depends on the `Decompile` parameter. If it was specified as `SEVENTEEN`, then it is a list of the format

```
{ { "label1" <foo> } { "label2" <bar> } { ... } ... }
```

The first object in the sub-list will be shown, and the whole list will be pushed to the stack. However, if `<foo>` is something like `DROP <bar>`, then you just need to execute `COMPEVAL` with the list as argument to evaluate the program `<bar>`.

The `ChooseDecompile` parameter follows the same rules as the `Decompile` one. Normally, specify `SEVENTEEN` for a LF, or `MINUSONE` for any other kind of field.

The reset and initial values specify the values which will show when the form is first displayed, or when it is reset. For DFs and EDFs, it is any object or `MINUSONE` if empty. For LFs, it is an object like in the `ChooseData` parameter (normally { "label" <foo> }). And for CFs, it is either `TRUE` or `FALSE`.

13.3 Label and field counts

These are two bints, representing the number of labels and fields defined. Note that since they are different values, you can have labels which just show some kind of information to the user, or fields without any label definition.

13.4 The message handler

The message handler is, again, a secondary (which is pushed in the stack, not evaluated) or simply `'DROPFALSE`, if your program does not test or change the input entered by the user.

13.5 The title string

This is a string that will be shown on the top of the display, with the small font. If it is longer than 131 characters (the width of the screen), it will be truncated and a “...” will be appended.

13.6 Results of the input form

The stack output, if the user exited the input form by ENTER is:

```
N+1: field1
N: field2
...
2: fieldN
1: TRUE
```

If CANCEL was used to exit the form, then just `FALSE` is returned.

For each field type the output is:

Field Type	Return value
DF/EDF	any (or <code>MINUSONE</code> if empty)
CF	<code>TRUE/FALSE</code>
LF	{ "Label" foo }

13.7 An example

Here is a program that displays an input form for an Equation Solver program. Then, if the screen was exited by ENTER, it puts some strings in the stack showing to the user selections. A *real* program would solve the equation, but this is only an example, so do not expect very much from it.

```

::
    CK0NOLASTWD          ( no args required )

* Label definitions
    $ "EQ:"              ONE    TEN
    $ "VAR:"              ONE    NINETEEN
    $ "TOL:"              SIXTY  NINETEEN
    $ "COMPLEX ROOTS"     EIGHT  TWENTYEIGHT
    $ "ANGLE MODE:"       ONE    THIRTYSEVEN

* EQ field definition
    'DROPPFALSE          ( no message handler )
    FIFTEEN              ( x position )
    EIGHT                ( y position )
    # 61                 ( length )
    NINE                 ( standard height - NINE )
    TWENTYTHREE          ( EDF )
    { NINE }             ( symbolics allowed )
    FOUR                 ( decomp with STD mode )
    $ "Enter equation or press CHOOSE" ( help string )
    $ "Equations in "    ( string to show in CHOOSE box )
    MINUSONE             ( choose decompile )
    DUPDUP              ( no reset/init value )

* VAR field definition
    'DROPPFALSE          ( no message handler )
    EIGHTEEN             ( x position )
    SEVENTEEN            ( y position )
    THIRTYFIVE           ( length )
    NINE                 ( height )
    ONE                  ( DF )
    { SIX }              ( ID's allowed )
    FOUR                 ( decomp with STD mode )
    $ "Enter variable"   ( help string )
    MINUSONE             ( no choose data )
    DUP                  ( no choose decompile )
    DUPDUP              ( no reset/init value )

* TOL field definition
    'DROPPFALSE          ( no message handler )
    # 4D                 ( x position )
    SEVENTEEN            ( y position )
    THIRTYFIVE           ( length )
    NINE                 ( height )
    ONE                  ( DF )
    { ZERO }             ( reals allowed )
    FOUR                 ( decomp with STD mode )
    $ "Enter tolerance"  ( help string )
    MINUSONE             ( no choose data )
    DUP                  ( no choose decomp )
    % .0001              ( reset value )
    DUP                  ( init value )

* Checkmark field definition
    'DROPPFALSE          ( no message handler )
    ONE                  ( x position )
    TWENTYSIX            ( y position )
    SIX                  ( checkmark length is SIX )
    NINE                 ( height )
    THIRTYTWO           ( CF )

```

```

MINUSONE          ( no types in CF )
DUP               ( CF )
$ "Use [+/-] to mark" ( help string )
MINUSONE          ( no choose data )
DUP               ( nor choose decomp )
FalseFalse       ( reset/init value: not checked )

* ANGLE MODE field definition
'DROPPFALSE       ( no message handler )
FORTYNINE         ( x position )
THIRTYFIVE        ( y position )
THIRTY            ( length )
NINE              ( height )
TWELVE            ( LF )
MINUSONE          ( no types in LF )
SEVENTEEN         ( display first object in comp )
$ "Use CHOOSE or [+/-] to select" ( help string )
* Field contents. The first will be shown, the entire
* list will be pushed to the stack
{ { "RAD" DROP SETRAD }
  { "DEG" DROP SETDEG }
  { "GRAD" DROP SETGRAD } }
SEVENTEEN         ( choose decomp )
{ "RAD" DROP SETRAD } ( reset value )
DUP               ( init value )

* General info
FIVE              ( five labels )
DUP               ( same number of fields )
'DROPPFALSE       ( default message handler )
$ "Solve Equation" ( title string )
DoInputForm       ( do the form )
NOT?SEMI          ( exit if CANCEL pressed )
COMPEVAL          ( set angle mode )

* If this were a real program, solving would
* probably start here. But we will just show
* what the user selected
FOUR reversym DROP
SWAP $ "Solve for " SWAP
ID>$ &$ $ " in" &$SWAP
$ "Tol.: " 4ROLL
DO>STR &$
4ROLL
ITE
$ "Real&complex roots"
$ "Real roots only"
;

```

The picture below shows how the screen will look like.

```

██████████ SOLVE EQUATION ██████████
EQ: ████████████████████████████████
VAR:                TOL: .0001
 _COMPLEX ROOTS
ANGLE MODE: RAD

ENTER EQUATION OR PRESS CHOOSE
EDIT CHOOSE ██████████ CANCEL OK

```

Chapter 14

The parameterized outer loop

The parameterized outer loop is a System RPL structure that allows you to create a complete application, which receives keystrokes and does different actions, based on the key that was pressed. This is repeated s many times as necessary, until an exit condition happens. Most of the times, there is a key that stops the loop, like CANCEL or DROP. Generally, it is used with programs that work with the display.

To set up a parameterized outer loop, nine parameters are necessary:

Parameter name	Description
AppDisplay	This object is evaluated before each key evaluation. “AppDisplay” should handle display updating not handled by the keys themselves, and should also perform special handling of errors.
AppKeys	The hard key assignments, in the format described below.
NonAppKeyOK?	A flag: if <code>TRUE</code> , then the hard keys not assigned perform their normal actions. Otherwise, they are canceled.
DoStdKeys?	A flag: if <code>TRUE</code> , then standard key definitions are used for non-application keys instead of default key processing.
AppMenu	The menu specification, in the format described below, or <code>FALSE</code> .
#AppMenuRow	The initial menu row. Normally <code>ONE</code> .
SuspendOK?	A flag: if <code>TRUE</code> , any user command that would create a suspended environment and restart the system outer loop will instead generate an error.
ExitCond	This object is evaluated before each display update and key evaluation. If the result is <code>TRUE</code> , the loop is exited.
AppError	The error-handling object to be evaluated in an error occurs during key evaluation.

After those arguments are in the stack, run `ParOuterLoop`. This word does not generate any results itself, but any of the key assignments can return results to the stack or any other form desired.

14.1 Parameterized outer loop words

The parameterized outer loop is formed by calls (with proper error handling) to calls to the following words. None of them return anything, and only `POLSetUI` takes arguments: the nine required by `ParOuterLoop`.

Word	Action
<code>POLSaveUI</code>	Saves the current user interface in a temporary environment.
<code>POLSetUI</code>	Sets the current user interface, according to the parameters given.
<code>POLKeyUI</code>	Displays, reads and evaluates keys. Handles errors, and exits according to the user interface specified by <code>POLSetUI</code> .
<code>POLRestoreUI</code>	Restores the user interface saved by <code>POLSaveUI</code> and abandons temporary environment.
<code>POLResUI&Err</code>	Restores the user interface and errors. This is used when there is an error not handled within the parameterized outer loop.

The word `ParOuterLoop` decompiles to:

```
::
POLSaveUI      ( save the current user interface )
ERRSET ::      ( start error trap )
  POLSetUI      ( set new user interface )
  POLKeyUI      ( handle keypresses )
;
ERRTRAP
  POLResUI&Err  ( if an error happened, restore the saved )
                ( interface and error )
POLRestoreUI    ( restore saved user interface )
;
```

If you use the words above instead of `ParOuterLoop`, you *must* provide the same level of error protection as the code above.

One note: the parameterized outer loop creates a temporary environment when it saves its current user interface, and it abandons the interface when it restores a saved user interface. This means that you cannot use words that operate on the topmost temporary environment, like `1GETLAM` within the loop, unless the variable was created *after* calling `POLSaveUI`, and it is abandoned *before* calling `POLRestoreUI`. For temporary environments created before `POLSaveUI`, *named* temporary variables should be used.

14.2 The display

In the parameterized outer loop, the user is responsible for setting up the display and updating it; there is no default display.

The display can be updated in two ways: with the parameter “AppDisplay” or with key assignments. For example, when the user presses a key to move the cursor, the key assignment can either pass information to “AppDisplay” (often implicitly), so that it handles the code, or the key assignment object can handle the display itself. Which method is more efficient depends on the situation. In our example below, AppKeys just sets the positions of the grob, which is draw by AppDisplay.

14.3 Error handling

If an error occurs during the key processing, AppError is executed. This parameter is responsible for processing any errors generated while the parameterized outer loop is running. AppError should determine the specific word and act accordingly. Or you can just specify ERRJMP as AppError, which means your application does not handle any errors.

14.4 Hard key assignments

In the parameterized outer loop, any key in any of the six planes (see section 11.1) can be assigned for a new function. The parameter AppKeys specifies which keys to assign and their new assignments.

If a key is not assigned by the application, and the NonAppKeyOK? parameter is TRUE, the standard key definition is executed if the DoStdKeys? parameter is TRUE, or, if available, the USER key assignment, if it is FALSE. If NonAppKeyOK? is FALSE, a warning beep is produced, and nothing else is done.

Most of the time, NonAppKeysOK? should be set to FALSE.

The AppKeys parameter is a secondary, which must take as argument the keycode and plane, and return either the desired key definition and TRUE or FALSE if the application does not handle it. Specifically, the stack diagram is as follows:

```
( #KeyCode #Plane → KeyDef TRUE )
( #KeyCode #Plane → FALSE      )
```

The suggested form for the key assignments is:

```
ONE #=casedrop :: (process unshifted plane ) ;
TWO #=casedrop :: (process left-shifted plane ) ;
...
2DROP FALSE
```

Each plane handler normally has the form

```
SEVEN      ?CaseKeyDef :: TakeOver <process MTH key> ;
NINETEEN   ?CaseKeyDef :: TakeOver <process TAN key> ;
...
DROP FALSE
```

The word `?CaseKeyDef` replaces `#=casedrop :: ' <keydef> TRUE ;`. It should be used because it saves code and makes the definitions more legible. Specifically, `?CaseKeyDef` is used in the form:

```
... #KeyCode #TestKeyCode ?CaseKeyDef <keydef> ...
```

If `#TestKeyCode` equals `#KeyCode`, `?CaseKeyDef` drops both of them, pushes `<KeyDef>` and `TRUE` to the stack, and exits the secondary. Otherwise, it drops only `#TestKeyCode`, skips `<KeyDef>` and continues.

14.5 Menu key assignments

Any application can specify an initial menu via the `AppMenu` parameter to be displayed when the parameterized outer loop starts. All menu keys can have assignments to the unshifted, left-shifted and right-shifted planes. When the loop exits, the previous menu is restored intact.

You can specify `FALSE` as `AppMenu`. This indicates that the current menu is to be left intact. If you specify an empty list, then a menu of six null-menu keys is created.

Note: hard key assignments have priority over menu key assignments. So, if you are planning to include a menu key handler, you must put the following line in the hard key assignments parameter:

```
DUP#<7 casedrpfls
```

The parameter `AppMenu` is a list of the form:

```
{
  Menu Key 1 Definition
  Menu Key 2 Definition
  ...
  Menu Key n Definition
}
```

Each menu key definition is one the following:

1. `NullMenuKey`
2. `{ LabelObj :: TakeOver <action> ; }`
3. `{ LabelObj {
 :: TakeOver <primary action> ;
 :: TakeOver <l-shift action> ;
 } }`

```

4. { LabelObj {
      :: TakeOver <primary action> ;
      :: TakeOver <l-shift action> ;
      :: TakeOver <r-shift action> ;
    } }

```

A `LabelObj` may be any object, but it is normally a string or a 21x8 grob. The word `NullMenuKey` inserts a blank menu key that just beeps when pressed.

The word `TakeOver` indicates that the program will be executed even if the command line is active.

14.6 Preventing suspended environments

Your application may require the evaluation of arbitrary commands and user arguments, but do not want the current environment to be suspended by `HALT` or `PROMPT` commands. The parameter `SuspendOK?`, when `FALSE`, will cancel these and any other commands that would suspend the environment and generate a “HALT Not Allowed” error, which `AppError` can handle. If the parameter is `TRUE`, the application must be prepared to handle the consequences. “The dangers here are many and severe”, as it is written on `RPLMAN.DOC`.

All foreseeable application should set `FALSE` as the `SuspendOK?` parameter.

14.7 The exit condition

The parameter `ExitCond` is an object that is evaluated before each key evaluation. If it evaluates to `TRUE`, the loop is exited, otherwise it continues. You could define, for example, `ExitCond` as `' LAM exit`. When the “quit” key is pressed, you just have to use `TRUE ' LAM exit STO` and the loop will be exited. Of course, you must create the `lam` and initialize it with `FALSE` before.

14.8 An example

The following program is an example of an application that uses a parameterized outer loop to create an environment where the user may move a little graphic over the screen. You can use the arrow keys to move, or the menu keys. In both cases, if you press left-shift before, the graphic moves ten steps instead of one. I added code so that the graphic does not go off the screen boundaries.

```

::
* Defines names for used keys. Makes things easier and
* more readable
  DEFINE kpNoShift      ONE
  DEFINE kpLeftShift    TWO
  DEFINE kcUpArrow      ELEVEN
  DEFINE kcLeftArrow     SIXTEEN
  DEFINE kcDownArrow    SEVENTEEN
  DEFINE kcRightArrow   EIGHTEEN
  DEFINE kcLeftShift    THIRTYFIVE
  DEFINE kcOn           FORTYFIVE

* Prepare display
  RECLAIMDISP      ( clear and resize display )
  CldrDAIsStat     ( temporarily disable clock )

* Smiling face grob. The below must be in one line only.
  GROB 7C 310003100008F000060300810C004000104000102000202401201000
4010004010004010004011044021042026032048F010400010810C0006030008F0
00
  FIFTYSIX      ( initial x coordinate for box )
  EIGHTEEN      ( initial y coordinate for box )
  FALSE         ( initial exit condition )
  {
    LAM MrSmile
    LAM x
    LAM y
    LAM exit?
  } BIND      ( binds local variables )
* The following composite is the display update object. It
* clears the screen and draws the smiling face grob to it.
' ::
  CLEARVISP      ( clear display )
  LAM MrSmile     ( recall smiling face grob )
  HARDBUFF       ( recall current display )
  LAM x LAM y     ( smile coordinates )
  GROB!          ( REPL )
  DispMenu.1     ( display menu )
;
* The following composite is the key action handler.
' ::
  kpNoShift #=>casedrop ::
    DUP#<7 casedrpfls ( enable softkeys )
    kcUpArrow ?CaseKeyDef
      :: TakeOver LAM y DUP ONE #<ITE :: DROP ERBEEP ; :: #1- '
        LAM y STO ; ;
    kcDownArrow ?CaseKeyDef
      :: TakeOver LAM y DUP THIRTYFIVE #>ITE :: DROP ERBEEP ; ::
        #1+ ' LAM y STO ; ;
    kcLeftArrow ?CaseKeyDef
      :: TakeOver LAM x DUP ONE #<ITE :: DROP ERBEEP ; :: #1- '
        LAM x STO ; ;
    kcRightArrow ?CaseKeyDef
      :: TakeOver LAM x DUP # 6F #>ITE :: DROP ERBEEP ; :: #1+
        ' LAM x STO ; ;
    kcOn ?CaseKeyDef
      :: TakeOver TRUE ' LAM exit? STO ;
    kcLeftShift #=>casedrpfls
    DROP 'DoBadKeyT
;

```

```

kpLeftShift #=casedrop ::
  DUP#<7 casedrpfls ( enable softkeys )
  kcUpArrow ?CaseKeyDef
    :: TakeOver LAM y DUP TEN #<ITE :: DROPZERO ERBEEP ; ::
    TEN #- ; ' LAM y STO ;
  kcDownArrow ?CaseKeyDef
    :: TakeOver LAM y DUP TWENTYSEVEN #>ITE :: DROP
    THIRTYSEVEN ERBEEP ; #10+ ' LAM y STO ;
  kcLeftArrow ?CaseKeyDef
    :: TakeOver LAM x DUP TEN #<ITE :: DROPZERO ERBEEP ; ::
    TEN #- ; ' LAM x STO ;
  kcRightArrow ?CaseKeyDef
    :: TakeOver LAM x DUP # 66 #>ITE :: DROP # 70 ERBEEP ;
    #10+ ' LAM x STO ;
  kcLeftShift #=casedrpfls
  DROP 'DoBadKeyT
;
2DROP 'DoBadKeyT
;
TrueTrue ( key definitions )
* Menu specification
{ { "Up" {
  :: TakeOver LAM y DUP ONE #<ITE :: DROP ERBEEP ; ::
  #1- ' LAM y STO ; ;
  :: TakeOver LAM y DUP TEN #<ITE :: DROPZERO ERBEEP ;
  :: TEN #- ; ' LAM y STO ;
}
}
{ "Down" { :: TakeOver LAM y DUP THIRTYSIX #>ITE :: DROP
ERBEEP ; :: #1+ ' LAM y STO ; ;
:: TakeOver LAM y DUP TWENTYSEVEN #>ITE :: DROP
THIRTYSEVEN ERBEEP ; #10+ ' LAM y STO ;
}
}
{ "Left" { :: TakeOver LAM x DUP ONE #<ITE :: DROP ERBEEP ;
:: #1- ' LAM x STO ; ;
:: TakeOver LAM x DUP TEN #<ITE :: DROPZERO ERBEEP
; :: TEN #- ; ' LAM x STO ;
}
}
{ "Right" { :: TakeOver LAM x DUP # 6F #>ITE :: DROP ERBEEP ;
:: #1+ ' LAM x STO ; ;
:: TakeOver LAM x DUP # 66 #>ITE :: DROP # 70
ERBEEP ; #10+ ' LAM x STO ;
}
}
NullMenuKey
{ "Quit" :: TakeOver TRUE ' LAM exit? STO ; }
}
ONEFALSE ( first row, no suspended envs )
' LAM exit? ( exit condition )
'ERRJMP ( error handler )
ParOuterLoop ( run the par outer loop )
RECLAIMDISP ( resize and clear display )
ClrDasOK ( redraw display )
;

```

Chapter 15

The display

There are two screens available to the programmer while programming in System-RPL: the graphics screen, which is visible, for example, in the Plot application, and the text screen, which is the graphic visible in the standard stack environment. Whenever possible, the latter should be used, leaving the graphics screen as a “owned” resource.

15.1 Display organization

The HP48 system RAM contains three dedicated graphic objects (subsequently called grobs) used for display purposes.

Pointer	Grob
HARDBUFF2	Menu labels
ABUFF	Text grob (stack)
GBUFF	Graphics grob (PICT)

The text and graphic grobs may be enlarged, and may be scrolled. The menu label grob has a fixed size of 131x8 pixels.

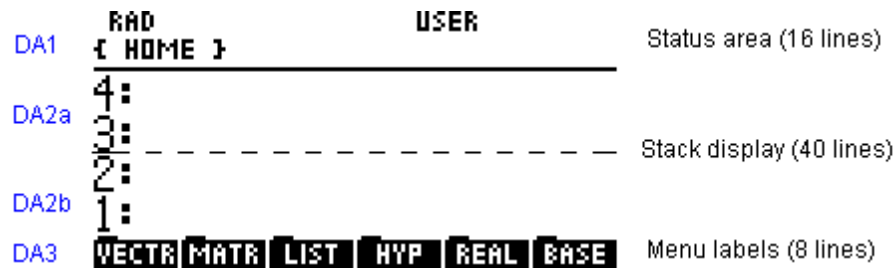
The word `TOADISP` makes the text grob visible; the word `TOGDISP` makes the graphic grob visible.

The following words return display grobs to the stack:

Word	Stack
ABUFF	(→ textgrob)
GBUFF	(→ graphgrob)
HARDBUFF2	(→ menugrob)
HARDBUFF	(→ HBgrob)
	Returns whichever grob is visible (text or graph)
HBUFF_X_Y	(→ HBgrob #x1 #y1)

One thing to note is that the words above return just pointer to the grob, so if you alter the grob, the display will also be altered automatically. If you do not want that behavior, after using any of the words above, call `TOTEMPOB` to make a unique copy in temporary memory.

The text grob is divided in three regions. The display areas are numbered one, two and three. In many words you will find “DA”, which means “Display Area”.



Display area 2 is actually divided in two areas: 2a and 2b. The boundary between the two areas can move, but the overall size of all three areas is fixed.

15.2 Preparing the display

Two words establish control over the text display: `RECALIMDISP` and `ClrDA1IsStat`. The first does the following:

- Assures the current display is the text one;
- Clears the text display;
- If necessary, resizes the text display to the default size of 131x56 pixels.

This word works very similarly to the user word `CLLCD`, the difference is that `CLLCD` never resizes the text display.

The word `ClrDA1IsStat` is optional, but most of the time it should be used. It suspends the ticking clock display temporarily. When input is expected from the user from the keyboard, using `WaitForKey` or the parameterized outer loop, the ticking clock would botch the display. `ClrDA1IsStat` avoids this problem.

When the menu is not necessary, use the word `TURNMENUOFF` to hide the menu and enlarge the text grob to 131x64 pixels. It is turned on again with `TURNMENUON`.

The suggested framework for an application that uses the text display is:

```
::
  ClrDA1IsStat      ( suspend clock )
  RECLAIMDISP      ( set, clear and resize text display )
  TURNMENUOFF      ( turns off menu )

  <application>

  ClrDAsOK          ( redraw LCD )
  -or-
  SetDAsTEMP        ( freeze the whole display )
;
```

15.3 Controlling display refresh

In some programs, it is desired that, after the application ends, the screen is not redrawn, but continues frozen so that the user can see the results. In User RPL the word `FREEZE` is responsible for this. But sometimes, it is desired that the display is returned back to normal. In System RPL, several words serve those purposes. The most used ones are listed below; the whole list is on Chapter 41.

Word	Action
SetDA1Temp	Freezes display area 1.
SetDA2aTemp	Freezes display area 2a.
SetDA2bTemp	Freezes display area 2b.
SetDA2OKTemp	Freezes display areas 2a and 2b.
SetDA3Temp	Freezes display area 3.
SetDA12Temp	Freezes display areas 1 and 2.
SetDAsTemp	Freezes the whole display.
ClrDA1OK	Redraws display area 1.
ClrDA2aOK	Redraws display area 2a.
ClrDA2bOK	Redraws display area 2b.
ClrDA2OK	Redraws display areas 2a and 2b.
ClrDA3OK	Redraws display area 3.
ClrDAsOK	Redraws the whole display.

15.4 Clearing the display

The following words clear either the whole or part of `HARDBUFF`. Remember that `HARDBUFF` refers to the currently displayed grob, either the text or the graph display. Except from `BLANKIT`, no words take or return arguments.

Word	Action
BLANKIT	(#startrow #rows →) Clears #rows from <code>HARDBUFF</code>
CLEARVDISP	Clears entire <code>HARDBUFF</code> .
BlankDA1	Clears display area 1
BlankDA2	Clears display area 2
BlankDA12	Clears display areas 1 and 2
Clr16	Clears top 16 rows
Clr8	Clears top 8 rows
Clr8-15	Clears rows 8 to 15 (second status line)
CLCD10	Clears status and stack area
CLEARLCD	Clears entire display

15.5 Annunciator control

You can control the left-shift, right-shift and alpha annunciators directly, using the words listed below and on Chapter 41. However, be careful to return them to the state they were before after your application finishes.

Word	Action
ClrAlphaAnn	Clears the alpha annunciator.
SetAlphaAnn	Sets the alpha annunciator.
ClrLeftAnn	Clears the left-shift annunciator.
SetLeftAnn	Sets the left-shift annunciator.
ClrRightAnn	Clears the right-shift annunciator.
SetRightAnn	Sets the right-shift annunciator.

15.6 Display coordinates

The upper-left pixel of the display has the coordinates $x=0$ and $y=0$, and the bottom-right pixel has coordinates $x=130$ and $y=63$.

Subgrobs are taken from the upper left coordinate to the pixel below and to the right of the lower right corner coordinate. The terms $\#x1$ and $\#y1$ refer to the upper left coordinates, and $\#x2$ and $\#y2$ refer to the pixel below and to the right of the lower right corner.

15.6.1 Display subgrobs

The words described below return `HARDBUFF` and coordinates for the stack in a form suitable for a subsequent call to `SUBGROB` in order to get a portion of the current display. $\#x1$ and $\#y1$ refer to the upper left corner of the window on the currently displayed grob. If the grob has been scrolled (see section 15.6), these will *not* be $\#0$ and $\#0$.

When `HARDBUFF` has been scrolled, some of the words below may not be appropriate since they depend on the upper left corner being $\#0$, $\#0$. The LCD is then called window and the terms $\#x1$ and $\#y1$ refer to the upper left corner of the window. The word `HBUFF_X_Y` returns `HARDBUFF` and these coordinates. The word `WINDOWCORNER` returns just the coordinates.

Word	Stack
TOP8	(\rightarrow HBgrob $\#x1$ $\#y1$ $\#x1+131$ $\#y1+8$)
TOP16	(\rightarrow HBgrob $\#x1$ $\#y1$ $\#x1+131$ $\#y1+16$)
Rows8-15	(\rightarrow HBgrob $\#x1$ $\#y1+8$ $\#x1+131$ $\#y1+16$)

15.6.2 Scrolling the display

To scroll the display, use the words `SCROLLUP`, `SCROLLDOWN`, `SCROLLLEFT` and `SCROLLRIGHT`. These words scroll the display in the specified

direction one pixel, and check if their corresponding arrow key is being held down. If it is, they repeat their action until the key is released or the edge of the display is reached.

To go directly to one edge of the display, use one of these words: JUMPTOP, JUMPBOT, JUMPLEFT or JUMPRIGHT.

15.7 Displaying text

The HP48 has three fonts available: the large font (5x9), the medium font (5x7) and the small font. The small font is variable width; the medium and large ones are fixed width.

In User RPL, the programmer can only use the medium font directly (with `DISP`). In System RPL, it is possible to display text directly with the large and medium fonts. If you want to use the small font or place the text exactly where you want, that must be done graphics, which will be described later.

15.7.1 Medium font

To display text with the medium font, use one of the words below. These words should only be used when the text grob is the current display and the screen has not been scrolled. Strings longer than 22 characters will be truncated to 21 characters and a trailing ellipsis (...) will be added. Shorter strings will be blank filled.

Word	Stack and description
DISPROW1	(\$ →) Displays string on first line.
DISPROW2	(\$ →) Displays string on second line.
DISPROW3	(\$ →) Displays string on third line.
DISPROW4	(\$ →) Displays string on fourth line.
DISPROW5	(\$ →) Displays string on fifth line.
DISPROW6	(\$ →) Displays string on sixth line.
DISPROW7	(\$ →) Displays string on seventh line.
DISPROW8	(\$ →) Displays string on eight line. Only works if menu is off.

Word	Stack and description
DISPN	(\$ #row →) Displays string on specified line.
DISP5x7	(\$ #start #max →) Display string starting at #start row, spanning up to #max rows. The string must have embedded carriage returns to show where to break to next line. If any line segment is longer than 22 characters, it is truncated and an ellipsis is added.

You can use the words `DISPROW1*` and `DISPROW2*` when the display has been scrolled. These words display in the first and second lines the given string, but the difference is that they display relative to the window corner. Unfortunately, there are no equivalents for the other five rows.

15.7.2 Large font

The words below can be used to display text in the large font. The same rules for long strings of the medium fonts are applied to the large font.

Word	Stack and description
BIGDISPROW1	(\$ →) Displays string on first line.
BIGDISPROW2	(\$ →) Displays string on second line.
BIGDISPROW3	(\$ →) Displays string on third line.
BIGDISPROW4	(\$ →) Displays string on fourth line.
BIGDISPN	(\$ #row →) Displays string on specified row.

15.7.3 Displaying warnings

The word `FlashWarning` is used to display a warning message. It displays the given string on the status area, beeps and pauses for approximately three seconds. To display a message in the status area, it uses the word `DISPSTATUS2`, which takes a string with a line break in it, and displays it using the two lines of the status area. After that, the display is returned to the state it was before.

A variation of `FlashWarning` is `FlashMsg`, which does not beep and pauses for a shorter time.

15.8 The menu line

The menu line consists of six grobs eight pixels high and 21 pixels wide. The starting columns for each menu key label in `HARBDUFF2` is:

Column (Hex)	Softkey	Column (hex)	Softkey
0	First softkey (A)	42	Fourth softkey (D)
16	Second softkey (B)	58	Fifth softkey (E)
2C	Third softkey (C)	6E	Sixth softkey (F)

The word `DispMenu.1` redisplay the current menu; and `DispMenu` redisplay the current menu and calls `SetDA3Valid` to freeze the menu display area (display area 3).

The words below convert several kinds of objects to menu labels and display them at the specified column:

Word	Stack and action
<code>Str>Menu</code>	(#col \$ →) Makes and displays a standard menu label.
<code>Id>Menu</code>	(#col id →) Recalls <code>id</code> and displays standard or directory label, depending on the contents.
<code>Grob>Menu</code>	(#col grob →) Displays a grob as a menu label.
<code>Seco>Menu</code>	(#col :: →) Evaluates secondary and uses results to create and display appropriate menu label.

The words below convert strings to the four different kinds of grobs available. All of them take a string and return a grob as arguments

Word	Action
<code>MakeStdLabel</code>	Makes a black label (standard).
<code>MakeBoxLabel</code>	Makes label with a box inside.
<code>MakeDirLabel</code>	Makes directory label (bar above)
<code>MakeInvLabel</code>	Makes white label (like in Solver).

15.9 Using graphic objects

Following, there are words for creating, manipulating and displaying graphic objects. The ones listed below are just the most used or important. You can find a complete list on Chapter 42

When dealing with graphics, keep two things in mind:

1. Some grob operations work directly on the grob without making a copy. So, all pointers to that object in the stack will be modified. You can use the word `CKREF` to ensure an object is unique. This kind of operation is denominated “bang-type”, and the commands normally have an exclamation point to indicate that, like `GROB!` or `GROB!ZERO`. These operations also have no error checking, so improper or out-of-range parameters may corrupt memory.
2. To place a grob in the display grob, it is better to use the word `XYGROBDISP`. This word checks if the grob to be placed in `HARDBUFF` would exceed its boundaries, and if so `HARDBUFF` is enlarged so that the grob fits.

15.9.1 Grob tools

The words below are used for creating or modifying graphic objects. The complete list is on Chapter 42

Word	Stack an description
GROB!	(grob1 grob2 #x #y →) Stores grob1 into grob2. Bang type.
GROB!ZERO	(grob #x1 #y1 #x2 #y2 → grob') Blanks a rectangular region of the grob. Bang type.
GROB!ZERODRP	(grob #x1 #y1 #x2 #y2 →) Blanks a rectangular region of the grob. Assumes text or graph grob. Bang type.
SUBGROB	(grob #x1 #y1 #x2 #y2 → grob') Returns specified portion of grob.
XYGROBDISP	(#row #col grob →) Stores grob in <code>HARDBUFF</code> , expanding if necessary.
GROB>GDISP	(grob →) Stores new graph grob.
MAKEGROB	(#height #width → grob) Creates a blank grob.
INVGROB	(grob → grob') Inverts grob data bits. Bang type.
PIXON	(#x #y →) Sets pixel in text grob.

Word	Stack an description
PIXOFF	(#x #y →) Clears pixel in text grob.
PIXON?	(#x #y → flag) Is pixel in text grob on?
PIXON3	(#x #y →) Sets pixel in graph grob.
PIXOFF3	(#x #y →) Clears pixel in graph grob.
PIXON?3	(#x #y → flag) Is pixel in graph grob on?
ORDERXY#	(#x1 #y1 #x2 #y2 → #x1' #y1' #x2' #y2') To draw lines, #x2 must be greater than #x1. This function orders the coordinates so that the above condition is met.
LINEON	(#x1 #y1 #x2 #y2 →) Draws a line in text grob.
LINEOFF	(#x1 #y1 #x2 #y2 →) Clears a line in text grob.
TOGLINE	(#x1 #y1 #x2 #y2 →) Toggles a line in text grob.
LINEON3	(#x1 #y1 #x2 #y2 →) Draws a line in graph grob.
LINEOFF3	(#x1 #y1 #x2 #y2 →) Clears a line in graph grob.
TOGLINE3	(#x1 #y1 #x2 #y2 →) Toggles a line in graph grob.
\$>BIGGROB	(\$ → grob) Makes grob of the string using the large font (5x9).
\$>GROB	(\$ → grob) Makes grob of the string using the medium font (5x7).
\$>grob	(\$ → grob) Makes grob of the string using the small font.
Symb>HBuf f	(symb →) Displays symbolic in HARDBUFF in Equation Writer form. Enlarges HARDBUFF if necessary, so use RECLAIMDISP after.

15.9.2 Grob dimensions

The words below return or verify size information of grobs.

Word	Stack and description
GROBDIM	(grob → #height #width) Returns dimensions of the grob.
GROBDIMw	(grob → #width) Returns width of the grob.
DUPGROBDIM	(grob → grob #height #width) Does DUP, then GROBDIM.
GBUFFGROBDIM	(→ #height #width) Returns dimensions of graphic grob.
CKGROBFITS	(grob1 grob2 #row #col → grob1 grob2' #row #col) Shrinks grob2 if it wouldn't fit in grob1, starting at specified row and column.

Part III

Reference

Chapter 16

Notes

In the following chapters, there is a list of (almost) all commands available in System RPL to the programmer. As observed earlier, there are two kinds of commands: supported and unsupported. Supported are the ones the HP design team guarantees to stay always at the same address in all ROM versions, so they can be used safely. Unsupported ones don't have that guarantee, so you must be careful when using them, or your program might not run in all ROM versions, and could even cause a crash. Unsupported commands have their names listed between ()'s. They must be called by their addresses, using `PTR <address>`, because their names are not in the entry point list of the compilers.

However, it is better to assign names to those address (which will be valid only for that project). This way, you can call them by their names, and not by their addresses. This also makes your code more readable, and if the function is going to be called several times, you have one more reason to do that. Just include in your source file:

```
ASSEMBLE
=<name> EQU #<address>
...
RPL
```

You can include as many of these as needed. Just call the function using `<name>`.

In some chapters, there are conversion functions. The listed functions are the ones for converting to the data type described in the chapter only. So, to find a function to convert from a real number to a binary integer, look at the chapter about binary integers, and the opposite function will be found in the chapter about real numbers. The exceptions are the functions that convert complex numbers to reals representing the real and imaginary parts. These are listed in the chapter about complex numbers.

These are the object types and their representation in stack diagrams:

Type	Address	Prolog	Object Type	Stack
1	02933	DOREAL	Real number	%
2	02977	DOCMP	Complex number	C%
3	02A2C	DOCSTR	Character string	\$
4	029E8	DOARRY	Array	[]
5	02A74	DOLIST	List	{}
6	02E48	DOIDNT	Global identifier	id
7	02E6D	DOLAM	Local identifier	lam
8	02D9D	DOCOL	Secondary	::

Type	Address	Prolog	Object Type	Stack
9	02AB8	DOSYMB	Algebraic object	symb
B	02A4E	DOHSTR, DOHXS	Hexadecimal string	hxs
C	02B1E	DOGROB	Graphics object	grob
D	02AFC	DOTAG	Tagged object	tag
E	02ADA	DOEXT	Unit	unit
0F	02E92	DOROMP	ROM Pointer	romptr
1F	02911	DOBINT	Binary integer	#
2F	02A96	DORRP	Directory	rrp
3F	02955	DOEREL	Extended real number	%%
4F	0299D	DOECMP	Extended complex number	C%%
5F	02A0A	DOLNKARRY	Linked array	[L]
6F	029BF	DOCHAR	Character	chr
7F	02DCC	DOCODE	Code object	code
8F	02B40	DOLIB	Library	lib
9F	02B62	DOBAK	Backup	bak
AF	02B88	DOEXT0	Library data	libdat
BF	02BAA	DOACPTR	Access pointer (GX only)	acptr
CF	02BCC	DOEXT2	External type 2	ext2
DF	02BEE	DOEXT3	External type 3	ext3
EF	02C10	DOEXT4	External type 4	ext4

The stack representations below have special meanings:

Representation	Meaning
comp	Composite: either symb or :: or { }
sym	Symbolic class: either symb or id or lam
symf	Either symb or id or lam or % or C%
pco	Primitive code object
ROMPTR	ROMPTR object
romptr	Contents of ROMPTR (preceded by property fields)
romp	ROMPTR or romptr
meta or M	Meta object (see Chapter 26)
F%	Either % or C%
L%	Either %% or C%
B%	Either % or C% or %% or C%
ob	Any object
nob	Next object in the runstream

Chapter 17

Binary integers

17.1 Built-in binary integers

Dec	Hex	Address	Name(s)
0	0	03FEF	ZERO
1	1	03FF9	ONE, real, MEMERR
2	2	04003	TWO
3	3	0400D	THREE, str
4	4	04017	FOUR
5	5	04021	FIVE, list
6	6	0402B	SIX, id, idnt
7	7	04035	SEVEN
8	8	0403F	EIGHT, seco
9	9	04049	NINE, symb
10	A	04053	TEN, sym
11	B	0405D	ELEVEN
12	C	04067	TWELVE
13	D	04071	THIRTEEN
14	E	0407B	FOURTEEN, EXT
15	F	04085	FIFTEEN
16	10	0408F	SIXTEEN, REALOB
17	11	04099	SEVENTEEN, 2REAL, REALREAL
18	12	040A3	EIGHTEEN
19	13	040AD	NINETEEN
20	14	040B7	TWELVE
21	15	040C1	TWENTYONE
22	16	040CB	TWENTYTWO
23	17	040D5	TWENTYTHREE
24	18	040DF	TWENTYFOUR
25	19	040E9	TWENTYFIVE
26	1A	040F3	TWENTYSIX, REALSYM
27	1B	040FD	TWENTYSEVEN
28	1C	04107	TWENTYEIGHT
29	1D	04111	TWENTYNINE
30	1E	0411B	THIRTY, REALEXT
31	1F	04125	THIRTYONE
32	20	0412F	THIRTYTWO
33	21	04139	THIRTYTHREE
34	22	04143	THIRTYFOUR
35	23	0414D	THIRTYFIVE

Dec	Hex	Address	Name(s)
36	24	04157	THIRTYSEX
37	25	04161	THIRTYSEVEN
38	26	0416B	THIRTYEIGHT
39	27	04175	THIRTYNINE
40	28	0417F	FORTY, FOURTY
41	29	04189	FORTYONE
42	2A	04193	FORTYTWO
43	2B	0419D	FORTYTHREE
44	2C	64B12	FORTYFOUR
45	2D	64B1C	FORTYFIVE
46	2E	64B26	FORTYSIX
47	2F	64B30	FORTYSEVEN
48	30	64B3A	FORTYEIGHT
49	31	64B44	FORTYNINE
50	32	64B4E	FIFTY
51	33	64B58	FIFTYONE
52	34	64B62	FIFTYTWO
53	35	64B6C	FIFTYTHREE, STRLIST
54	36	64B76	FIFTYFOUR
55	37	64B80	FIFTYFIVE
56	38	64B8A	FIFTYSIX
57	39	64B94	FIFTYSEVEN
58	3A	64B9E	FIFTYEIGHT
59	3B	64BA8	FIFTYNINE
60	3C	64BB2	SIXTY
61	3D	64BBC	SIXTYONE
62	3E	64BC6	SIXTYTWO
63	3F	64BD0	SIXTYTHREE
64	40	64BDA	SIXTYFOUR, YHI, BINT_40h
65	41	64BE4	BINT_65d, ARRYREAL
66	42	64BEE	FOUR TWO
67	43	64BF8	FOURTHREE
68	44	64C02	SIXTYEIGHT
69	45	64C0C	FOURFIVE
70	46	64C16	SEVENTY
74	4A	64C20	SEVENTYFOUR
79	4F	64C2A	SEVENTYNINE
80	50	64C34	EIGHTY
81	51	64C3E	EIGHTYONE, LISTREAL
82	52	64C48	LISTCMP
83	53	64C52	FIVETHREE
84	54	64C5C	FIVEFOUR
85	55	64C66	2LIST
86	56	64C70	FIVESIX
87	57	64C7A	LISTLAM
91	5B	64C84	BINT_91d
96	60	64C8E	BINT_96d
97	61	64C98	IDREAL
100	64	64CAC	ONEHUNDRED
111	6F	64CC0	char

Dec	Hex	Address	Name(s)
115	73	64CE8	BINT_115d
116	74	64CF2	BINT_116d
122	7A	64D06	BINT_122d
128	80	64D10	BINT80h
130	82	64D1A	BINT130d, XHI-1
131	83	64D24	BINT131d, XHI
145	91	64D38	SYMBREAL
158	9E	64D56	SYMBUNIT
160	A0	64D6A	SYMOB
161	A1	64D74	SYMREAL
166	A6	64D92	SYMID
167	A7	64D9C	SYMLAM
170	AA	64DB0	SYMSYM
174	AE	64DBA	SYMEXT
175	AF	1CD69	(BINT_AFh)
192	C0	64DD8	BINTC0h
204	CC	64DE2	2GROB
208	D0	64DEC	TAGGEDANY
225	E1	64DF6	EXTREAL
234	EA	64E00	EXTSYM
238	EE	64E0A	2EXT
240	F0	64E14	ROMPANY
253	FD	64E1E	BINT253
255	FF	64E28	BINT255d
256	100	64E32	REALOBOB
258	102	64E3C	#_102
273	111	64E64	3REAL
279	117	15D6F	(BINT_117h)
337	151	64F04	INTEGER337
2563	A03	34301	ATTN#
2563	A03	64FC2	ATTNERR
3082	C0A	6506C	Connecting
3584	E00	6508A	EXTOBOB
10547	2933	03F8B	TYPEREAL
10581	2955	03FDB	(TYPEEREL)
10568	2948	03FA9	TYPEIDNT
10615	2977	03F95	(TYPECMP)
10868	2A74	03F9F	(TYPELIST)
10902	2A96	03FC7	(TYPERRP)
10936	2AB8	03FBD	(TYPESYMB)
10970	2ADA	03FE5	(TYPEEXT)
11677	2D9D	03FB3	(TYPECOL)
11885	2E6D	03FD1	(TYPELAM)
458752	70000	65094	#EXITERR
1048575	FFFFFF	6509E	MINUSONE

These words either put more than one bint in the stack or do some kind of stack manipulation:

Address	Name	Stack
641FC	ZEROZERO	(→ #0 #0)
64209	#ZERO#ONE	(→ #0 #1)
6427A	#ZERO#SEVEN	(→ #0 #7)
63AC4	ONEONE	(→ #1 #1)
Also called ONEDUP.		
6428A	#ONE#27	(→ #1 #27d)
6429D	#TWO#ONE	(→ #2 #1)
642AF	#TWO#TWO	(→ #2 #2)
642BF	#TWO#FOUR	(→ #2 #4)
642D1	#THREE#FOUR	(→ #3 #4)
642E3	#FIVE#FOUR	(→ #5 #4)
64309	ZEROZEROZERO	(→ #0 #0 #0)
6431D	ZEROZEROONE	(→ #0 #0 #1)
64331	ZEROZEROTWO	(→ #0 #0 #2)
62535	DROPZERO	(ob → #0)
64449	(3DROPZERO)	(ob1 ob2 ob3 → #0)
6254E	2DROP00	(ob1 ob2 → #0 #0)
62946	DROPONE	(ob → #1)
63A88	DUPZERO	(ob → ob #0)
63A9C	DUPONE	(ob → ob #1)
63AD8	DUPTWO	(ob → ob #2)
63AB0	SWAPONE	(ob1 ob2 → ob2 ob1 #1)
62E3A	ZEROSWAP	(ob → #0 ob)
63079	ZEROOVER	(ob → ob #0 ob)
6351F	ZEROFALSE	(→ #0 FALSE)
62E67	ONESWAP	(ob → #1 ob)
63533	ONEFALSE	(→ #1 FALSE)

17.2 Conversion words

Address	Word	Stack and notes
18CEA	COERCE	(% → #)
62CE1	COERCEDUP	(% → # #)
62E7B	COERCESWAP	(ob % → # ob)
194F7	COERCE2	(% % → # #)
18CD7	%ABSCOERCE	(% → #)
2EC11	%IP>#	(% → #)
Does ABS too.		
193DA	(COERCE{ } 2)	({ % } → { # })
		({ % % } → { # # })
05A03	HXS>#	(hxs → #)
4F3D1	(2HXS>#)	(hxs hxs → # #)

Address	Word	Stack and notes
05A51	CHR>#	(chr → #)
51532	2HXSLIST?	({ hxs1 hxs2 } → #1 #2)
Converts list of two hxs to two bints. Generates “Bad Argument Value” for invalid input.		

17.3 Arithmetic functions

In the following table, the object #1 and #2 represent two binary integers, and not the binary integers 1 and 2.

Address	Name	Stack and comments
03DBC	#+	(#1 #2 → #1+#2)
25B0B	(#+OVF)	(#1 #2 → #1+#2)
0 = result = FFFFF		
03DEF	#1+	(# → #+1)
03E2D	#2+	(# → #+2)
6256A	#3+	(# → #+3)
6257A	#4+	(# → #+4)
6258A	#5+	(# → #+5)
6259A	#6+	(# → #+6)
625AA	#7+	(# → #+7)
625BA	#8+	(# → #+8)
625CA	#9+	(# → #+9)
625DA	#10+	(# → #+10)
625EA	#12+	(# → #+12)
03DE0	#-	(#1 #2 → #1-#2)
03E0E	#1-	(# → #-1)
03E4E	#2-	(# → #-2)
625FA	#3-	(# → #-3)
6260A	#4-	(# → #-4)
6261A	#5-	(# → #-5)
6262A	#6-	(# → #-6)
03EC2	#*	(#1 #2 → #1*#2)
191B9	#*OVF	(#1 #2 → #1*#2)
0 = result = FFFFF		
03E6F	#2*	(# → #*2)
62691	#6*	(# → #*6)
62674	#8*	(# → #*8)
6264E	#10*	(# → #*10)
03EF7	#/	(#1 #2 → #rem #quo)
03E8E	#2/	(# → #/2)
Rounded down		
637CC	#-+1, #1--	(#1 #2 → (#1-#2)+1)
63808	#+-1, #1-+	(#1 #2 → (#1+#2)-1)
624FB	#-#2/	(#1 #2 → (#1-#2)/2)
627D5	#+DUP	(#1 #2 → #1+#2 #1+#2)

Address	Name	Stack and comments
62DFE	#+SWAP	(ob #1 #2 → #1+#2 ob)
63051	#+OVER	(ob #1 #2 → ob #1+#2 ob)
627F8	#-DUP	(#1 #2 → #1-#2 #1-#2)
62E12	#-SWAP	(ob #1 #2 → #1-#2 ob)
63065	#-OVER	(ob #1 #2 → ob #1-#2 ob)
62809	#1+DUP	(# → #+1 #+1)
62E26	#1+SWAP	(ob # → #+1 ob)
1DABB	#1+ROT	(ob1 ob2 # → ob2 #+1 ob1)
6281A	#1-DUP	(# → #-1 #-1)
5E4A9	#1-SWAP	(ob # → #-1 ob)
62FD9	#1-ROT	(ob1 ob2 # → ob2 #-1 ob1)
28558	#1-UNROT	(ob1 ob2 # → #-1 ob1 ob2)
62E4E	#1-1SWAP	(# → 1 #-1)
Returns the bint ONE and the result		
628EB	DUP#1+	(# → # #+1)
626F7	DUP#2+	(# → # #+2)
6292F	DUP#1-	(# → # #-1)
63704	2DUP#+	(#1 #2 → #1 #2 #1+#2)
Also called DUP3PICK#+		
637F4	DROP#1-	(# ob → #-1)
62794	SWAP#-	(#1 #2 → #2-#1)
62904	SWAP#1+	(# ob → ob #+1)
Also called SWP1+		
51843	SWAP#1+SWAP	(# ob → #+1 ob)
637E0	SWAP#1-	(# ob → ob #-1)
51857	SWAP#1-SWAP	(# ob → #-1 ob)
5EAF4	(SWAPDROP#1-)	(ob # → #-1)
637A4	SWAPOVER#-	(#1 #2 → #2 #1-#2)
6272C	OVER#+	(#1 #2 → #1 #2+#1)
6377C	OVER#-	(#1 #2 → #1 #2-#1)
63718	ROT#+	(#1 ob #2 → ob #2+#1)
63768	ROT#-	(#1 ob #2 → ob #2-#1)
637B8	ROT#1+	(# ob1 ob2 → ob1 ob2 #+1)
5FB76	ROT#1+UNROT	(# ob1 ob2 → #+1 ob1 ob2)
62DCC	ROT#+SWAP	(#1 ob #2 → #2+#1 ob)
Also called ROT+SWAP		
63740	3PICK#+	(#1 ob #2 → #1 ob #2+#1)
63754	4PICK#+	(#1 ob ob #2 → #1 ob ob #2+#1)
62DE5	4PICK#+SWAP	(#1 ob ob #2 → #1 ob #2+#1 ob)
Also called 4PICK+SWAP		
624BA	#MIN	(#1 #2 → #)
624C6	#MAX	(#1 #2 → #)
03EB1	#AND	(#1 #2 → #)
Bitwise AND		

17.4 Tests

The words below return `TRUE` or `FALSE` depending on the condition. Their names should be enough to understand the use. `2#=OR` returns `TRUE` if either of the two arguments is `ZERO`. Note that `ONE_EQ` does an `EQ` check, not `EQUAL` check (see Chapter 33). Another name for `#>1` is `ONE#>`.

Address	Name	Address	Name	Address	Name
03D19	#=	636C8	#2<>	6289B	2DUP#<
03CA6	#0=	03CE4	#<	63687	DUP#<7
622A7	#1=	63673	#<3	628D1	2DUP#>
6229A	#2=	03D83	#>	63385	ONE_EQ
62289	#3=	636F0	#>1	620EB	OVER#=
636B4	#5=	628B5	2DUP#=	6364B	OVER#0=
03D4E	#<>	62266	DUP#0=	6365F	OVER#<
03CC7	#0<>	622C5	DUP#1=	636DC	OVER#>
622B6	#1<>	622D4	DUP#0<>	6362D	2#0=OR

Chapter 18

Real numbers

18.1 Built-in real numbers

Number	Address	Real word	Address	Extended real word
-9.99E499	2A487	%-MAXREAL		--
-9	2A42E	%-9		--
-8	2A419	%-8		--
-7	2A404	%-7		--
-6	2A3EF	%-6		--
-5	2A3DA	%-5		--
-4	2A3C5	%-4		--
-3	2A3B0	%-3		--
-2	2A39B	%-2		--
-1	2A386	%-1		--
-1E-499	2A4B1	%-MINREAL		--
0	2A2B4	%0	2A4C6	%%0
1E-499	2A49C	%MINREAL		--
$\pi/180$ (0,0174...)		--	2A62C	PI/180
.1	494B1	%.1	2A562	%%.1
.15	495AA	(%.15)		--
.4		--	2B3DD	%%.4
.5	650BD	%.5	2A57C	%%.5
.555...		--	10E68	cfF
1	2A2C9	%1	2A4E0	%%1
			10E82	cfC (also %%1)
2	2A2DE	%2	2A4FA	%%2
e (2.7183...)	650A8	%e		--
3	2A2F3	%3	2A514	%%3
π (3.1416...)	2A443	%PI	2A458	(%%PI)
4	2A308	%4	2A52E	%%4
5	2A31D	%5	2A548	%%5
6	2A332	%6		--
2π (6.2832..)	514EB	(%2PI)	0F688	%%2PI
7	2A347	%7	2B1FF	%%7
8	2A35C	%8		--
9	2A371	%9		--
10	650E7	%10	2A596	%%10
11	1CC03	%11		--
12	1CC1D	%12	2B2DC	%%12
13	1CC37	%13		--

Number	Address	Real word	Address	Extended real word
14	1CC51	%14		--
15	1CC85	%15		--
16	1CD3A	%16		--
17	1CD54	%17		--
18	1CDF2	%18		--
19	1CE07	%19		--
20	1CC6B	%20		--
21	1CCA4	%21		--
22	1CCC3	%22		--
23	1CCE2	%23		--
24	1CD01	%24		--
25	1CD20	%25		--
26	1CD73	%26		--
27	1CD8D	%27		--
60		--	2B300	%%60
80	320B1	%80		--
100	415F1	%100		--
180	650FC	%180		--
273.15		--	10E9C	(%KZERO)
360	65126	%360		--
459.67		--	10EB6	(%RZERO)
1200	22352	(%1200)		--
2400	22367	(%2400)		--
4800	2237C	(%4800)		--
8192	0EFEE	(%TICKSsec)		--
9600	22391	(%9600)		--
491520	0F003	(%TICKSmin)		--
29491200	0F018	(%TICKShour)		--
707788800	0F02D	(%TICKSday)		--
4954521600	0F042	(%TICKSweek)		--
9.99E499	2A472	%MAXREAL		--

These words combine stack manipulation with built-in real numbers:

Address	Word	Address	Word
5198F	(DROP%0)	1CA0D	(DROP%1)
54B1E	(DROP%0ABND)	54A9C	(DROP%1ABND)
1F047	(2DROP%0)		
56AFB	(4DROP%0)		
50A3B	(UNROT2DROP%0)		

18.2 Conversion words

Address	Word	Stack
245C1	%>%%	(% → %%)
62E8F	%>%%SWAP	(ob % → %% ob)
2A5B0	%%>%	(%% → %)
2B45C	2%>%%	(% % → %% %%)
2B470	2%%>%	(%% % → % %)
5435D	HXS>%	(hxs → %)
18DBF	UNCOERCE	(# → %)
1950B	UNCOERCE2	(# # → % %)
63B96	UNCOERCE%%	(# → %%)
19529	(UNCOERCE{ } 2)	({ # } → { % })
		({ # # } → { % % })

18.3 Real functions

In the table below, all CK% functions work like the user versions. For example, CK%SQRT returns a complex number if its argument is negative. %SQRT error if its argument is negative. The same happen for the other CK% words.

Address	Real function	Address	Ex. Real function
2A974	%+	2A943	%%+
2A981	%-	2A94F	%%-
2A95B	%>%%-		--
2A9BC	%*	2A99A	%%*
2A9FE	%/	2A9E8	%%/
	--	63B82	%%/>%
2AA70	%^	2AA5F	%%^
2A900	%ABS	2A8F0	%%ABS
2A920	%CHS	2A910	%%CHS
2AAAF	%1/	2AA92	%1/
2AA9E	%>%%1/		--
2AB09	%SQRT	2AAEA	%%SQRT
2AAF6	%>%%SQRT		--
1B3F5	(CK%SQRT)		--
1B47B	(%SQ)		--
2AB2F	%EXP	2AB1C	%%EXP
2AB42	%EXPM1		--
2AB6E	%LN	2AB5B	%LN
1B995	(CK%LN)		--
2ABA7	%LNP1	2AB94	%%LNP1
2AB81	%LOG		--
1BA0C	(CK%LOG)		--
2ABBA	%ALOG		--
2ABEF	%SIN	2AC06	%%SIN

Address	Real function	Address	Ex. Real function
	--	2AC27	%%SINRAD
	--	2AC17	%%SINDEG
2AC40	%COS	2AC57	%%COS
	--	2AC78	%%COSRAD
	--	2AC38	%%COSDEG
2AC91	%TAN		--
	--	2ACA8	%%TANRAD
2ACC1	%ASIN		--
1B6EA	(CK%ASIN)		--
	--	2ACD8	%%ASINRAD
2ACF1	%ACOS		--
1B775	(CK%ACOS)		--
	--	2AD08	%%ACOSRAD
2AD21	%ATAN		
2ADAE	%SINH	2AD95	%%SINH
2ADDA	%COSH	2ADC7	%%COSH
2ADED	%TANH		--
2AE00	%ASINH		--
2AE13	%ACOSH		--
1B86C	(CK%ACOSH)		--
2AE26	%ATANH		--
1B8DE	(CK%ATANH)		--
2A930	%MANTISSA		--
2AE39	%EXPONENT		--
2AF4D	%FP		--
2AF60	%IP		--
2AF86	%FLOOR	2AF99	%%FLOOR
2AF73	%CEIL		--
2ABDC	%MOD		--
2AFAC	(%INT)	2AF99	%%INT
1B30D	(%ARG)		--
2AD38	%ANGLE	2AD4F	%%ANGLE
2AD3G	%>%%ANGLE		--
	--	2AD7C	%%ANGLERAD
	--	2AD6C	%%ANGLEDEG
2B529	RNDXY		--
	(%number %places → %number')		
2B53D	TRCXY		--
	(%number %places → %number')		
2AE62	%COMB		--
2AE75	%PERM		--
2AE4C	%NFACT		--
	Calculates factorial of number.		
2B0C4	%FACT		--
	Calculates $\Gamma(x+1)$		
2AA81	%NROOT		--
	Equivalent to user function XROOT		
2A70E	%MIN		--
2A6F5	%MAX	2A6DC	%%MAX

Address	Real function	Address	Ex. Real function
62D81	%MAXorder (% % → %max %min)		--
51AB7	(%MAXIMIZE) (% → %0, %MAXREAL or % -MAXREAL)		--
2AFC2	%RAN Returns next random number.		--
2B044	%RANDOMIZE		--
2B07B	System level RDZ. DORANDOMIZE		--
	Stores given number as random number seed.		
2A9C9	%OF		--
2AA0B	%T		--
2AA30	%CH		--
2A622	%D>R		--
2A655	%R>D		--
2B48E	%REC>%POL (x y → radius angle)	2B498	%%R>P
2B4BB	%POL>%REC (radius angle → x y)	2B4C5	%%P>R
2B4F2	%SPH>%REC (%r %th %ph → %x %y %z)		--
	--	51A94	(%%SQR)
	(%%1 %%2 → %%1^2+%%2^2)		
51A71	(2%>%%SQR) Does 2%>%% and then %%SQR		--
	--	520B2	(2DUP%%R)
	(%%1 %%2 → %%1 %%2 %% (sqrt(%%1^2+%%2^2)))		
50262	%1+		--
50276	%1-		--
62BF1	%10*		--
51BE4	%+SWAP		--
	--	62EA3	%%*SWAP
	--	62FED	%%*ROT
	--	63C18	%%*UNROT
	--	63BBE	SWAP%%/

18.4 Tests

Address	Word	Address	Word	Address	Word	Address	Word
2A8C1	%=	2A8B6	%<=	2A75A	%%0=	2A87F	%%>
2A76B	%0=	2A738	%0<	2A7BB	%%0<>	2A895	%%>=
63BAA	DUP%0=	2A88A	%>	2A81F	%%<	2A788	%%0>
2A8CC	%<>	2A8A0	%>=	2A8AB	%%<=	2A7E3	%%0>=
2A7CF	%0<>	2A799	%0>	2A727	%%0<		
2A871	%<	2A7F7	%0>=	2A80B	%%0<=		

Chapter 19

Complex numbers

19.1 Built-in complex numbers

Number	Address	Word	Number	Address	Word
(0,0)	524AF	C%0	(%%1,%%0)	5193B	C%%1
(1,0)	524F7	C%1	(0,1)	5267F	(C%i)
(-1,0)	5196A	C%-1	(0,-1)	526AE	(C%-i)

19.2 Conversion words

Address	Word	Stack
519F8	C%%>C%	(C%% → C%)
05C27	%>C%	(%re %im → C%)
632A9	SWAP%>C%	(%im %re → C%)
51A37	Re>C%	(%re → C%)
05D2C	C%>%	(C% → %re %im)
519A3	C>Re%	(C% → %re)
519B7	C>Im%	(C% → %im)
05C72	(%%>C%%)	(%%re %%im → C%%)
05DBC	C%%>%%	(C%% → %%re %%im)
51A07	%%>C%	(%%re %%im → C%)
519CB	C%>%%	(C% → %%re %%im)
519DF	C%>%%SWAP	(C% → %%im %%re)
51C6B	(SWAP2C%>%)	(C%2 C%1 → re1 im1 re2 im2)
51C84	(SWAP2C%%>%%)	(C%%2 C%%1 → re1 im2 re2 im2)

19.3 Complex Functions

Functions requiring two arguments:

Address	Word	Address	Word	Address	Word
51C16	(C%C+C)	51BD0	(C%C+R)	51BF8	(C%R+C)
51C3E	(C%%C+C)	51C9D	(C%%C+R)	51CB1	(C%%R+C)
51CFC	(C%C-C)	51CE8	(C%C-R)	51CD4	(C%R-C)

Address	Word	Address	Word	Address	Word
51D10	(C% C-C)	51D38	(C% C-R)	51D24	(C% R-C)
51D88	(C% C*C)	51D4C	(C% C*R)	51D60	(C% R*C)
51DE2	(C% C*C)	51DAB	(C% C*C*R)	51DBF	(C% R*C)
51EC8	(C% C/C)	51E64	(C% C/R)	51E19	(C% R/C)
51F13	(C% C/C)	51F7C	(C% C/R)	51F3B	(C% R/C)
52374	C% C^C	52360	C% C^R	52342	C% R^C
51A4A	(C% *i)				
51A5F	(C/i)				

Functions that require just one argument:

Address	Complex Word	Address	Ex. Complex Word
52062	C%ABS	52080	(C%ABS)
51B70	C%CHS	51B91	C%CHS
51EFA	C%1/		--
52107	C%SQRT		--
1B48F	(C%SQ)		--
520CB	C%SGN		--
51BB2	C%CONJ	51BC1	C%CONJ
52099	C%ARG		--
52193	C%EXP		--
521E3	C%LN		--
522BF	C%LOG		--
52305	C%ALOG		--
52530	C%SIN		--
52571	C%COS		--
525B7	C%TAN		--
52804	C%ASIN		--
52863	C%ACOS		--
52675	C%ATAN		--
5262F	C%SINH		--
52648	C%COSH		--
5265C	C%TANH		--
5281D	C%ASINH		--
52836	C%ACOSH		--
527EB	C%ATANH		--

19.4 Tests

There are only two tests available for complex numbers:

Address	Word	Address	Word
51B43	C%0=	51B2A	C%%0=

19.5 Reals and complex numbers functions

Address	Name	Stack and description
35B47	(SWITCHFLOATS)	(B% → ?) Dispatches action based on type. The order is %, C%, %, C%. For example, to change the sign of any float: :: SWITCHFLOATS %CHS C%CHS %%CHS C%%CHS ;
35B88	(SWITCH2FLOATS)	(L% L% → ?) Works similarly to the above function. The order is %% %, C%% %, %% C%, C%% C%.
37D19	(F%>L%)	(% → %%) (C% C%%) Converts float to long float.
37BE9	(L%+)	(L% L% → L%) Adds long real or complex numbers.
37C0C	(L%-)	(L% L% → L%) Subtracts long real or complex numbers.
37C2F	(L%*)	(L% L% → L%) Multiplies long real or complex numbers.
37C52	(L%/)	(L% L% → L%) Divides long real or complex numbers.
37CD3	(B%NEG)	(B% → B%') Changes sign of any number.
37C75	(B%ABS)	(B% → B%') Absolute value of any number.
37DF6	(B%0=)	(B% → flag) Compares any number to zero.

Chapter 20

Character strings

20.1 Built-in characters

Address	Word	Address	Word	Address	Word
65464	CHR_0	6556E	CHR_X	654B1	CHR_;
6546B	CHR_1	65575	CHR_Y	65441	CHR_+
65472	CHR_2	6557C	CHR_Z	6544F	CHR_-
65479	CHR_3	65583	CHR_a	6543A	CHR_*
65480	CHR_4	6558A	CHR_b	6545D	CHR_/
65487	CHR_5	65591	CHR_c	654BF	CHR_=
6548E	CHR_6	65598	CHR_d	656BE	CHR_<>
65495	CHR_7	6559F	CHR_e	654B8	CHR_<
6549C	CHR_8	655A6	CHR_f	654C6	CHR_>
654A3	CHR_9	655AD	CHR_g	656B0	CHR_<=
654CD	CHR_A	655B4	CHR_h	656B7	CHR_>=
654D4	CHR_B	655BB	CHR_i	65694	CHR_['
654DB	CHR_C	655C2	CHR_j	6569B	CHR_]'
654E2	CHR_D	655C9	CHR_k	656A2	CHR_{
654E9	CHR_E	655D0	CHR_l	656A9	CHR_}'
654F0	CHR_F	655D7	CHR_m	65639	CHR_->
654F7	CHR_G	655DE	CHR_n	65640	CHR_<<
654FE	CHR_H	655E5	CHR_o	65647	CHR_>>
65505	CHR_I	655EC	CHR_p	65433	CHR_#
6550C	CHR_J	655F3	CHR_q	65425	CHR_...
65513	CHR_K	655FA	CHR_r	6541E	CHR_00
6551A	CHR_L	65601	CHR_s	6564E	CHR_Angle
65521	CHR_M	65608	CHR_t	6542C	CHR_DblQuote
65528	CHR_N	6560F	CHR_u	65655	CHR_Deriv
6552F	CHR_O	65616	CHR_v	6565C	CHR_Integral
65536	CHR_P	6561D	CHR_w	65663	CHR_LeftPar
6553D	CHR_Q	65624	CHR_x	6566A	CHR_Newline
65544	CHR_R	6562B	CHR_y	65671	CHR_Pi
6554B	CHR_S	65632	CHR_z	65678	CHR_Rightpar
65552	CHR_T	65456	CHR_.	6567F	CHR_Sigma
65559	CHR_U	65448	CHR_,'	65686	CHR_Space
65560	CHR_V	654AA	CHR_:'	6568D	CHR_UndScore
65567	CHR_W				

20.2 Built-in character strings

Address	Word	Representation
055DF	NULL\$	"" - Empty string
65254	SPACE\$	" " - Also called tok_
65238	NEWLINE\$	"\0a" - Newline
2E4F0	CRLF\$	"\0d\0a" - Carriage return and line feed
65797	\$_RAD	"RAD"
657A7	\$_GRAD	"GRAD"
656E5	\$_XYZ	"XYZ"
656D5	\$_R<Z	"R\80Z" - "R<angle>Z"
656C5	\$_R<<	"R\80\80" - "R<angle><angle>"
65769	\$_EXIT	"EXIT"
65757	\$_ECHO	"ECHO"
6577B	\$_Undefined	"Undefined"
656F5	\$_<>>	"\AB\BB" - "« »"
65703	\$_{ }	"{}"
65711	\$_[]	"[]"
6571F	\$_' '	"'" - Two single quotes
6572D	\$_::	"::"
6573B	\$_LRParens	"()"
65749	\$_2DQ	""" - Two double quotes
414BD	(\$_:)	". "
65290	tok,	" , "
65284	tok'	"'" - One single quote
652FC	tok-	" - "
6529C	tok.	". "
0FA69	tok_g	"g"
0FA8E	tok_m	"m"
0FACE	tok_s	"s"
65176	tok{	"{"
651D6	tok<<	"<<"
65308	tok=	"="
25446	tok->	"->"
6534C	tok0	"0"
65358	tok1	"1"
653AC	tok8	"8"
653B8	tok9	"9"
651BE	tokESC	"\1B" - Escape character
651E2	tokexponent	"E"
65278	tokquote	""" - One double quote
6518E	toksharp	"#"
65212	(14spc\$)	" " - String of 14 spaces.

Combinations of NULL\$ with other functions:

Address	Word	Action
62D59	NULL\$SWAP	NULL\$, then SWAP.
04D3E	DROPNULL\$	DROP then NULL\$.
04D57	(2DROPNULL\$)	2DROP then NULL\$.
1613F	NULL\$TEMP	Creates null string in temporary memory (NULL\$, then TOTEMPOB).

20.3 Conversion words

Address	Word	Stack and notes
6475C	CHR>\$	(chr → \$) Converts a character into a string.
167E4	#>\$	(# → \$) Creates string from the bint.
167D8	#:>\$	(# → \$) Creates string from the bint and appends a colon and a space. Ex: "1: "
162B8	a%>\$	(% → \$) Converts real number into string using current display mode.
162AC	a%>\$,	(% → \$) Same as above, but does not use commas.
05BE9	ID>\$	(id/lam → \$) Converts identifier into string.
540BB	hxs>\$	(hxs → \$) Uses current display mode and wordsize.
54061	HXS>\$	(hxs → \$) Does hxs>\$ and then appends base character.
140F1	DOCHR	(% → \$) Creates string of the character with the number specified.
1410F	(DONUM)	(\$ → %) Returns number of first character of string.

Address	Word	Stack and notes
15B13	DECOMP\$	(ob → \$) Convert object into its string representation for stack display.
15A0E	EDITDECOMP\$	(ob → \$) Convert object into its string representation for edition.
14088	DO>STR	(ob → \$) Internal version of ->STR.
14137	DOSTR>	(\$ → ?) Internal version of STR->.
238A4	palparse	(\$ → ob TRUE) (\$ → \$ # \$' FALSE) Tries parsing a string into an object. If successful, returns object and TRUE, otherwise returns position of error and FALSE.
49079	PromptIdUtil	(id ob → \$) Creates string of the form "id: ob".

20.4 Management words

Address	Word	Stack and action
05636	LEN\$	(\$ → #) Returns length in bytes.
1CA26	(LEN\$>%)	(\$ → %) LEN\$ then UNCOERCE.
627BB	DUPLEN\$	(\$ → \$ #) DUP then LEN\$.
05622	OVERLEN\$	(\$ ob → \$ ob #) OVER then LEN\$.
1782E	(2LEN\$#+)	(\$ \$ → \$ \$ #) Returns sum of length of two strings.
127CA	(DROPDUPLEN\$1+)	(\$ ob → \$ #len+1) Does DROP, then DUP, then LEN\$ and finally #1+.

Address	Word	Stack and action
050ED	CAR\$	(\$ \rightarrow chr) Returns first character of string, or NULL\$ for null string.
0516C	CDR\$	(\$ \rightarrow \$') Returns string without first character, or NULL\$ for null string.
05733	SUB\$	(\$ #start #end \rightarrow \$') Returns substring between specified positions.
62D6D	SUB\$SWAP	(ob \$ #start #end \rightarrow \$ ob) SUB\$ then SWAP.
1C8BB	(XEQSUB\$)	(\$ %start %end \rightarrow \$) Same as SUB\$ but uses real numbers as arguments.
63245	#1-SUB\$	(\$ #start #end \rightarrow \$') Does #1- and then SUB\$.
63259	1_#1-SUB\$	(\$ #end \rightarrow \$') Returns substring from the first character to the character before the specified position. Also called 1_#1-SUB
6326D	LAST\$	(\$ #start \rightarrow \$') Returns substring from the specified start position to the end (inclusive).
63281	#1+LAST\$	(\$ #start \rightarrow \$') Returns substring from the specified start position to the end (exclusive).
30805	SUB\$1#	(\$ #pos \rightarrow #char) Returns specified character as a bint.
05193	&\$	(\$1 \$2 \rightarrow \$1+\$2) Concatenates two strings.
622EF	SWAP&\$	(\$1 \$2 \rightarrow \$2+\$1) Concatenates two strings.
63F6A	&\$SWAP	(ob \$1 \$2 \rightarrow \$1+\$2 ob) &\$ then SWAP.

Address	Word	Stack and action
62376	!append\$	(\$1 \$2 → \$1+\$2) Tries &\$, if not enough memory does !!ap- pend\$?.
622E5	!insert\$	(\$1 \$2 → \$2+\$1) Does SWAP then !append\$.
62F2F	!append\$SWAP	(ob \$1 \$2 → \$1+\$2 ob) !append\$ then SWAP.
62312	!!append\$?	(\$1 \$2 → \$1+\$2) Attempts append “in place” if target is in tempob.
623A0	!!append\$	(\$1 \$2 → \$1+\$2) Tries appending “in place”.
62394	!!insert\$	(\$1 \$2 → \$2+\$1) Tries inserting “in place”.
0525B	>H\$	(\$ chr → \$') Prepends character to string
052EE	>T\$	(\$ chr → \$') Appends character to string.
62BB0	APPEND_SPACE	(\$ → \$') Appends space to string.
63191	NEWLINE\$&\$	(\$ → \$') Appends new line character to string. Also called NEWLINE&\$.
2E4DC	APNDCRLF	(\$ → \$') Appends carriage return and line feed to string.
61C1C	EXPAND	(\$ #nibbles → \$') Appends null characters to the string. Since refers to the number of nibbles, you must use a number twice as large as the number of null characters you want appended.
645B1	POS\$	(\$search \$find #start → #pos) Search for \$find in \$start, start at position #start. Returns position of \$find or 0 if not found.
645B1	POSCHR	(\$search chr #start → #pos) The same entry as above.

Address	Word	Stack and action
15EF6	(ONEPOS\$)	(\$search \$find/chr → #pos) POS\$ with #start = 1.
1CAD7	(XEQPOS\$)	(\$search \$find/chr → %pos) POS\$ with #start = 1 and followed by UNCOERCE.
645BD	POS\$REV	(\$search \$find #limit → #pos) Searches from end backwards until #limit.
645BD	POSCHRREV	(\$seach chr #limit → #pos) Same entry as above.
12770	COERCE\$22	(\$ → \$') If the string is longer than 22 characters, truncates it to 21 characters and appends “... ”.
127A7	SEP\$NL	(\$ → \$2 \$1) Separates string at first newline character. First string is second line, second string is first line.
45676	Blank\$	(# → \$) Creates a string with the specified number of spaces.
188D2	(NOT\$)	(\$ → \$') Logical NOT. Makes copy of the string first. These and the other words below work character by character of the string(s), doing the operation on the numerical representa- tion of the character.
18961	(!NOT\$)	(\$ → \$') Logical NOT “in place”.
18873	AND\$	(\$1 \$2 → \$') Logical AND. Errors if strings are not the same length.
188E6	(!AND\$)	(\$1 \$2 → \$') Logical AND. Does not check if strings are the same length.
18887	OR\$	(\$1 \$2 → \$') Logical OR. Errors if strings are not the same length.

Address	Word	Stack and action
188F5	(!OR\$)	(\$1 \$2 → \$') Logical OR, does not check if strings are the same length.
1889B	XOR\$	(\$1 \$2 → \$') Logical XOR. Errors if strings are not the same length.
18904	(!XOR\$)	(\$1 \$2 → \$') Logical XOR. Does not check if strings are the same length.

20.5 Tests

The string comparison test below (<, >, etc.) first check the length of the strings. If they are equal, then the character numbers are compared. Thus, “ABA” is smaller than “AAB”, which is smaller than “AAA”. However, “B” is smaller than “a”, because the uppercase characters have lower numbers. If you need to search ignoring case, you’ll need to develop a routine to change everything to upper case or everything to lower case.

Address	Word	Stack
0556F	NULL\$?	(\$ → flag)
63209	DUPNULL\$?	(\$ → \$ flag)
142A6	(\$<\$?)	(\$1 \$2 → %flag)
1420A	(\$>\$?)	(\$1 \$2 → %flag)
142E2	(\$<=\$?)	(\$1 \$2 → %flag)
142BA	(\$>=\$?)	(\$1 \$2 → %flag)
42C3D	CkChr00	(\$ → \$ flag)

Returns FALSE if string contains any null characters.

Chapter 21

Hexadecimal strings

21.1 Conversion words

Address	Word	Stack and notes
059CC	#>HXS	(# → hxs) Length will be five.
543F9	%>#	(% → hxs) Converts real number into hxs.

21.2 General functions

Address	Word	Stack and description
0EDE1	MAKEHXS	(#nibbles → hxs) Makes blank hxs of specified size.
3742D	(!MAKEHXS)	(#nibbles → hxs) Makes hxs filled with random data.
055D5	NULLHXS	(→ hxs) Puts a null hxs in the stack.
0518A	&HXS	(hxs1 hxs2 → hxs') Appends hxs2 to hxs1.
61C1C	EXPAND	(hxs #nibbles → hxs') Appends #nibbles zeros to the hxs.
05616	LENHXS	(hxs → #) Returns length in nibbles.
05815	SUBHXS	(hxs #start #end → hxs') Returns sub hxs string.

21.3 Arithmetic functions

All functions below assume a wordsize less than or equal to 64 bits. The resulting hxs length will be current wordsize.

Address	Word	Stack and description
53EA0	bit+	(hxs1 hxs2 → hxs) Adds two hxs.
54330	bit%#+	(% hxs → hxs') Adds real to hxs, returns hxs.
54349	bit#%+	(hxs % → hxs') Adds real to hxs, returns hxs.
53EB0	bit-	(hxs1 hxs2 → hxs) Subtracts hxs2 from hxs1.
542FE	bit%#-	(% hxs → hxs') Subtracts hxs from real, returns hxs.
5431C	bit#%-	(hxs % → hxs') Subtracts real from hxs, returns hxs.
53ED3	bit*	(hxs1 hxs2 → hxs) Multiplies two hxs.
542D1	bit%#*	(% hxs → hxs') Multiplies real by hxs, returns hxs.
542EA	bit#%*	(hxs % → hxs') Multiplies hxs by real, returns hxs.
53F05	bit/	(hxs1 hxs2 → hxs) Divides hxs1 by hxs2.
5429F	bit%#/	(% hxs → hxs') Divides real by hxs, returns hxs.
542BD	bit#%/	(hxs % → hxs') Divides hxs by real, returns hxs.
53EC3	(bitNEG)	(hxs → hxs') Changes sign of hxs.
53D4E	bitNOT	(hxs → hxs') Bitwise NOT.
53D04	bitAND	(hxs1 hxs2 → hxs) Bitwise AND.

Address	Word	Stack and description
53D15	bitOR	(hxs1 hxs2 → hxs) Bitwise OR.
53D26	bitXOR	(hxs1 hxs2 → hxs) Bitwise XOR.
53D5E	bitSL	(hxs → hxs') Shifts one bit to the left.
53D6E	bitSLB	(hxs → hxs') Shifts one byte to the left.
53D81	bitSR	(hxs → hxs') Shifts one bit to the right.
53D91	bitSRB	(hxs → hxs') Shifts one byte to the right.
53E0C	bitRL	(hxs → hxs') Shifts circularly one bit to the left.
53E3B	bitRLB	(hxs → hxs') Shifts circularly one byte to the left.
53DA4	bitRR	(hxs → hxs') Shifts circularly one bit to the right.
53DE1	bitRRB	(hxs → hxs') Shifts circularly one byte to the right.
53E65	bitASR	(hxs → hxs') Arithmetic shift one bit to the right. The most significant bit (the sign) does not change.

21.4 Tests

For all words below, the stack diagram is (hxs1 hxs2 → flag), except for NULLHXS?, whose stack diagram is (hxs → flag).

Address	Test	Description
05566	(NULLHXS?)	Returns TRUE if the input is a null hxs.
544D9	HXS==HXS	== test
544EC	HXS#HXS	≠ test
54552	HXS<HXS	< test
54500	HXS>HXS	> test
5453F	HXS<=HXS	≤ test
5452C	HXS>=HXS	≥ test

Chapter 22

Tagged objects

Address	Word	Stack and notes
05E81	>TAG	(ob \$ → tagged)
225F5	USER\$>TAG	(ob \$ → tagged)
		Maximum of 255 characters in string.
22618	%>TAG	(ob % → tagged)
05F2E	ID>TAG	(ob id/lam → tagged)
05E9F	({ }>TAG)	({ id ob } → tagged)
647BB	TAGOBS	(ob \$ → tagged)
		(ob1 ... obn { \$1 ... \$n } → tagged1 ... taggedn)
05EC9	(TAG>)	(tagged → ob \$tag)
64775	STRIPTAGS	(tagged → ob)
647A2	STRIPTAGS12	(tagged ob' → ob ob')
		The two words above remove all tags from an object.

Chapter 23

Arrays

In the stack diagrams below, {dims} represents a list of two bints, representing the array dimensions or the position of a element of the array.

23.1 General operations

Address	Function	Stack and description
03562	ARSIZE	([] → #elements) Returns number of elements as a bint.
63141	OVERARSIZE	([] ob → [] ob #elements) Does OVER then ARSIZE.
035A9	DIMLIMITS	([] → {dims}) Returns list of array dimensions.
357A8	MDIMS	([] → #m FALSE) ([] → #m #n TRUE) If it is a vector, returns number of elements and FALSE. If it is an array (including arrays with only one line), returns dimensions and TRUE.
62F9D	MDIMSDROP	([] → #m) ([] → #m #n) MDIMS followed by DROP.
03442	MAKEARRAY	({dims} ob → [ob]) Makes array with all elements initialized to ob.
19294	(>ARRAY)	(F% ... F% #n [F%] → [F%]) Copies floats into array.
1D054	XEQ>ARRAY	(F% ... F% {%dims} → [F%]) Makes array with specified dimensions and elements. Does checks first.
1D02C	(XEQ>VECTOR)	(F% ... F% %n → [F%]) Creates a vector.

Address	Function	Stack and description
1D0AB	(DOARRAY>)	([] → F% ... F% {%dims}) Explodes array. Only works for arrays of (normal) real and complex numbers.
35D35	(MATIDN)	([F%] → [F%]') Creates identity matrix. Errors if input is not a square matrix.
3745E	SWAPROWS	([] #m #n → []' #m #n) Swaps two rows. Does not make copy in tempob.
37508	(SWAPCOLUMNS)	([] #m #n → []' #m #n) Swaps two columns. Does not make copy in tempob.
0358F	(TYPEARRAY@)	([] → #prolog) Returns address of the prolog of the array element type.
03685	(ARRAYEL?)	({dims} [] → #element TRUE) ({dims} [] → FALSE) Returns TRUE if array element exists.
0371D	GETATELN	(# [] → ob TRUE) (# [] → FALSE) Get one element from array.
0C501	(GETEL)	(#index [] → ob TRUE) (#index [] → FALSE) (#index #addr_of_array → ob TRUE) (#index #addr_of_array → FALSE) Gets one element from array.
355B8	PULLREALEL	([%] #n → [%] %) Gets real element.
355C8	PULLCMPEL	([C%] #n → [C%] C%) Gets complex element.
3558E	(PULLEL)	([F%] #n → [F%] F%) Gets real or complex element.
35602	(PULLEREALEL)	([%] #n → [%] %%) Gets real element then converts to long real.
355D8	(PULLLONGEL)	([F%] #n → [F%] L%) Gets element then converts to long.

Address	Function	Stack and description
35628	PUTEL	([F%] B% #n → [F%]) Puts element at specified position. Converts to “short” before.
3566F	PUTREAL	([%] % #n → [%]) Puts real element at specified position.
356F3	PUTCMPEL	[C%] C% #n → [C%] Puts complex element at specified position.

23.2 Calculations

Address	Word	Stack and description
36115	(MAT+)	([F%] [F%] → [F%]) Adds two arrays.
36278	(MAT-)	([F%] [F%] → [F%]) Subtracts two arrays.
3644E	(MAT*)	([F%] [F%] → [F%]) Multiplies two arrays.
36AC3	(MAT/)	([F%] [F%] → [F%]) Divides two arrays.
362DC	(MATFLOAT*)	([F%] F% → [F%]) (F% [F%] → [F%]) Multiplies matrix by float.
363DB	(MATFLOAT/)	([F%] F% → [F%]) Divides matrix by float.
36444	(MATSQ)	([F%] → [F%]) Squares matrix.
35F30	(MATCONJ)	([F%] → [F%]) If a complex array, does the conjugate of all elements. If a real array, does nothing.
35DEB	(MATNEG)	([F%] → [F%]) Changes sign of all elements of array.
36A99	(MATINV)	([F%] → [F%]) Reciprocal of all elements of array.

Address	Word	Stack and description
3811F	MATTRN	([F%] → [F%]') Transposes matrix.
37E0F	METREDIM	([F%] {dims} → [F%]') Redimensions matrix. Removes elements or adds zeros as necessary.
35CAE	MATCON	([F%] F% → [F%]') Replace all elements of [F%] by F%.
35FA3	(DUP%0CON)	([F%] → [F%] [0%]) DUP then creates a matrix of the same size filled with zeros.
36A48	(MATDET)	([F%] → F%) Calculates determinant of matrix. Generates “Invalid Dimension” error for non-square matrices.
369E9	(MATABS)	([F%] → F%) Returns the scalar magnitude of array.
36705	(MATDOT)	([F%] [F%] → [F%]) Returns the dot product of two vectors.
36791	(MATCROSS)	([F%] [F%] → [F%]) Returns the cross product of two vectors. Generates a “Invalid Dimension” error if inputs are not vectors.
365BB	(MATRSD)	([F%] [F%] [F%] → [F%]) Calculates residuals of solutions of a linear system.
368F4	(MATRNRM)	([F%] → F%) Row norm.
3690D	(MATCNRM)	([F%] → F%) Column norm.
36039	(MATR>C)	([%re] [%im] → [C%]) Creates complex matrix from real and imaginary parts.
360B6	(MATC>R)	([C%] → [%re] [%im]) Explodes complex matrix into real and imaginary parts.

Address	Word	Stack and description
35F8F	(MATRE)	([F%] → [F%]') Returns (real) matrix with real part of complex numbers. Does nothing if the input is a real matrix.
35FEE	(MATIM)	([F%] → [F%]') Returns (real) matrix with imaginary part of complex numbers. Returns an array of zeros if input is a real matrix.
35E2C	(MATRND)	([F%] % → [F%]') RND on all elements of matrix.
35EA9	(MATTRNC)	([F%] % → [F%]') TRNC on all elements of matrix.
35C2C	(DOARRYPRG1)	(seco [F%] → [F%]') Evaluates seco for each element in array, then builds array again. Argument for seco will be L%. See examples below.
35C63	(DOARRYPRG2)	(seco [m1] [m2] → [F%]') Same as above, but seco has two arguments: one from m1 and another from m2. Arrays must be F%. Arguments for seco will be L%. See examples below.

Examples of DOARRYPRG1 and DOARRYPRG2:

```

* MATCHS - Changes sign of all elements of array
::
' NEGF% SWAP DOARRYPRG1 ;

* MATADD - Adds two matrices
::
' L%+ UNROT DOARRYPRG2 ;

```

23.3 Statistics

All functions below operate on the variables Σ_{DAT} and Σ_{PAR} . They work like their user versions. The `Column` words take as arguments a bint and an array. They then calculate the function for the specified column of the array.

Address	Word	Address	Word
2C22F	STATCLST	2C706	(STATGETYCOL)
	Clears ΣDAT	2C8E6	(STATCOV)
2C270	(STATRCL)	2C940	(STATX)
2C1F3	(STATSTO)	2C959	(STATY)
2C2D9	STATSADD%	2C972	(STATXX)
	Σ+ with real	2C99A	(STATYY)
2C535	STATN	2C9C2	(STATXY)
2C58A	STATSMIN	2CA0D	(STATLR)
2C558	STATSMAX	2CB4D	(STATPREDX)
2C571	STATSMEAN	2CADA	(STATPREDY)
2C5A3	STATSTDEV	2CCD3	(ColumnMIN)
2C5BC	STATTOT	2CCBA	(ColumnMAX)
2C5D5	STATVAR	2CCEE	(ColumnMEAN)
2C675	(STATCOL)	2CD09	(ColumnTDEV)
2C6B6	(STATXCOL)	2CCDF	(ColumnTOT)
2C6CF	(STATYCOL)	2CCFD	(ColumnVAR)
2C6F2	(STATGETXCOL)	2C83C	(STATCORR)

Chapter 24

Unit objects

24.1 Creating units

Address	Word	Description
10B5E	um*	* marker
10B68	um/	/ marker
10B72	um^	^ marker
10B7C	umP	Character prefix operator
10B86	umEND	Unit end operator

24.2 General operations

Address	Word	Stack and description
0F371	UMCONV	(unit1 unit2 → unit1') Change units of unit1 to units of unit2.
0F945	UMSI	(unit → unit') Equivalent to user word UBASE.
197C8	(UMFACT)	(unit1 unit2 → unit) Equivalent to user word UFACT.
0F34E	UMU>	(unit → % unit') Returns number and normalized part of unit.
10047	U>nbr	(unit → %) Returns number part of unit.
0F33A	UM>U	(% unit → unit') Replaces number part of unit.
10065	Unbr>U	(unit % → unit') Replaces number part of unit.
0F218	UNIT>\$	(unit → \$) Converts unit to string.

Address	Word	Stack and description
0FE44	U>NCQ	(unit → n%% cf%% []) Returns the number, conversion factor to base units and a vector in the form: [kg m A s K cd mol r sr ?] where each element represents the exponent of that unit. For example, 1_N U>NCQ would return: %%1 %%1 [1 1 0 -2 0 0 0 0 0] since it is equivalent to 1_kg*m/s^2

24.3 Arithmetic functions

Binary arithmetic operations: (unit1 unit2 → unit)

Address	Word	Address	Word	Address	Word
0F6A2	UM+	0FB8D	UMMIN	0FCCD	UM%T
0F77A	UM-	0FB6F	UMMAX	0FC3C	UM%CH
0F792	UM*	0F8FA	UMXROOT	0FD0E	UM%
0F823	UM/				

Unary arithmetic operations: (unit → unit')

Address	Word	Address	Word	Address	Word
0F5FC	UMABS	0FCE6	UMSIGN	0FD68	UMRND
0F615	UMCHS	0FCFA	UMIP	0FD8B	UMTRC
0F841	(UMINV)	0FD0E	UMFP	0F62E	UMSIN
0F913	UMSQ	0FD22	UMFLOOR	0F660	UMCOS
0F92C	UMSQRT	0FD36	UMCEIL	0F674	UMTAN
0FD4A	(UMOPER:)	Evaluates next object with numeric unit part, then builds unit again. For example: :: UMOPER: %1/ ;			

24.4 Tests

(unit1 unit2 → %flag)

Address	Word	Address	Word	Address	Word
0F584	UM=?	0F5AC	UM<?	0F5DA	UM<=?
0F598	UM#?	0F5C0	UM>?	0F5E8	UM>=?

Chapter 25

Composites

25.1 General operations

Address	Word	Stack and description
0521F	&COMP	(comp1 comp2 \rightarrow comp1+comp2) Concatenates two composites.
052FA	>TCOMP	(comp ob \rightarrow comp+ob) Adds ob to tail (end) of composite.
052C6	>HCOMP	(comp ob \rightarrow ob+comp) Adds ob to head (beginning) of composite.
1AC93	(SWAP>HCOMP)	(ob comp \rightarrow ob+comp) Does SWAP then >HCOMP.
05089	CARCOMP	(comp \rightarrow ob) (nullcomp \rightarrow nullcomp) Returns first object of the composite, or a null composite if the argument is a null composite.
6317D	?CARCOMP	(comp flag \rightarrow comp) (comp flag \rightarrow ob) If the flag is TRUE, does CARCOMP.
05153	CDRCOMP	(comp \rightarrow comp') Returns the composite minus its first object, or a null composite if the argument is a null composite.
0567B	LENCOMP	(comp \rightarrow #n) Returns length of composite (number of objects).
63231	DUPLENCOMP	(comp \rightarrow comp #n) Does DUP then LENCOMP.
1CA3A	(LENCOMP>%)	(comp \rightarrow %n) Returns length of composite as a real number.

Address	Word	Stack and description
05821	SUBCOMP	(comp #start #end → comp') Returns a sub-composite. Makes all possible index checks first.
056B6	NTHELCOMP	(comp #i → ob TRUE) (comp #i → FALSE) Returns specified element of composite and TRUE, or just FALSE if it could not be found.
62B9C	NTHCOMPDROP	(comp #i → ob) (comp #i →) Does NTHELCOMP then DROP.
62D1D	NTHCOMDDUP	(comp #i → ob ob) Does NTHCOMPDROP then DUP.
6480B	NEXTCOMPOB	(comp #offset → comp #next_offset ob TRUE) (comp #offset → FALSE) Returns object at specified nibble offset from start. If the object is SEMI (i.e., the end of the composite has been reached) returns FALSE. To get the first element, use FIVE as offset value (to skip the prolog).
055B7	NULLCOMP?	(comp → flag) If the composite is empty, returns TRUE.
6321D	DUPNULLCOMP?	(comp → comp flag) Does DUP then NULLCOMP?
643EF	matchob?	(ob comp → TRUE) (ob comp → ob FALSE) Returns TRUE if ob is EQUAL to any element of the composite.
64426	POSCOMP	(comp ob test → #i) (comp ob test → #0) Evaluates test for all elements of composite and ob, and returns index of first object which the test is TRUE, if no one returned TRUE, returns #0. For example, the program below returns #4: :: { %1 %2 %3 %-4 %-5 %6 %7 } %0 ' %< POSCOMP ;
644A3	EQUALPOSCOMP	(comp ob → #i) (comp ob → #0) POSCOMP with test = EQUAL.

Address	Word	Stack and description
644BC	NTHOF	(ob comp → #1) (ob comp → #0) Does SWAP then EQUALPOSCOMP.
6448A	#=POSCOMP	(comp # → #i) (comp # → #0) POSCOMP with test = #=.
644D0	FindlstTrue	(comp test → ob TRUE) (comp test → FALSE) Tests every element for test. The first one that returns TRUE if put into the stack along with TRUE. If no object returned TRUE, FALSE is put into the stack. For example, the program below returns -4 and TRUE: :: { %1 %2 %2 %-4 %-5 %6 } ' %0< FindlstTrue ;
6452F	Lookup	(ob test comp → nextob TRUE) (ob test comp → FALSE) Does matching in groups of two. If first matches, second is returned with TRUE; if there was no match, FALSE is returned. For example, the program below returns 6 and TRUE: :: %0 { %1 %2 %3 %-4 %-5 %6 } ' %< Lookup ;
64593	EQLookup	(ob comp → nextob TRUE) (ob comp → FALSE) Lookup with test = EQ.
64127	Embedded?	(ob1 ob2 → flag) Returns TRUE if ob2 is embedded in, or is the same as, ob1. Otherwise returns FALSE.

25.2 Building

Address	Word	Stack and notes
05331	(>COMP)	(meta #prolog → comp)
05459	{ }N	(meta → { })
05445	::N	(meta → seco)
0546D	SYMBN	(meta → symb)

Address	Word	Stack
54CEF	(SYMBN:)	(meta → symb&nob) Creates symbolic from meta contents and next object in the runstream
5E661	(ONESYMBN)	(ob → symb)
05481	EXTN	(meta → unit)
5E0DA	P{ }N	(meta → { })
5E111	(P::N)	(meta → seco)
5E0A3	(PSYMBN)	(meta → symb)

The P words first try the low level version, and if it errors (insufficient memory) the composite is built one element at a time. This allows garbage collection to happen while the composite is being built so that you can build larger composites. Use these words if you expect the composite to be very large.

25.3 Exploding

Address	Word	Stack
054AF	INNERCOMP	(comp → meta)
613E1	DUPINCOMP	(comp → comp meta)
631F5	SWAPINCOMP	(comp ob → ob meta)
62B88	INCOMPDROP	(comp → obn ... ob1)
62C41	INNERDUP	(comp → meta #n)
1C973	(INNERCOMP>%)	(comp → obn ... ob1 %n)
636A0	INNER#1=	(comp → obn ... ob1 flag)
5E585	(INNERtop&)	(meta comp → meta') Adds composite objects to meta object.

25.4 Lists

Address	Word	Stack and notes
055E9	NULL{ }	(→ { }) Puts a null list to the stack.
63A6F	DUPNULL{ }?	({ } → { } flag)
23EED	ONE{ }N	(ob → { })
631B9	TWO{ }N	(ob1 ob2 → { })
631CD	THREE{ }N	(ob1 ob2 ob3 → { })
631A5	#1-{ }N	(ob1 ... obn #n+1 → { })
1DC00	PUTLIST	({ } ob #n → { }') Replaces object at specified position. Assumes valid #i.
0E461	(INSERTN{ })	({ } ob #n → { }') Insert ob at nth position. Assumes valid #n.

Address	Word	Stack and notes
0E4DE	(REMOVEN{ })	({ } #n → { }') Removes nth ob. Assumes valid #n.
49CD6	(ROLL{ })	({ } → { }') Rolls list elements.
35491	apndvarlst	({ } ob → { }') Appends ob to list if not already there.

25.5 Secondaries

Address	Word	Stack and description
055FD	NULL::	(→ seco) Returns null secondary.
63FE7	Ob>Seco	(ob → seco) Does ONE then ::N.
63FCE	?Ob>Seco	(ob → seco) If the object is not a composite, does Ob>Seco.
63FFB	2Ob>Seco	(ob1 ob2 → seco) Does TWO then ::N.
632D1	::NEVAL	(meta → ?) Does ::N then EVAL.
5E8DE	(argum)	(seco → seco #args) Returns argument count for secondary. Checks first command, if it is different from CK0, CK1&Dispatch, etc. #5 is returned.
5E9A7	(infarg?)	(seco → seco flag) Is first command in secondary CKINFARGS?

Chapter 26

Meta objects

A meta object is actually a collection of n object and their count (a bint). The word `INNERCOMP` produces a meta object from a composite. A null meta is `ZERO`.

26.1 Stack functions

Address	Word	Stack and notes
5E35C	(dup)	(M → M M)
0326E	NDROP	(M →)
		Should be called <code>drop</code> .
63FA6	DROPNDROP	(M ob →)
		Should be called <code>DROPdrop</code> .
62F75	N+1DROP	(ob M →)
		Should be called <code>dropDROP</code> .
5EB1C	psh	(M1 M2 → M2 M1)
		Should be called <code>swap</code> .
5EB58	(rot)	(M1 M2 M3 → M2 M3 M1)
5EBDB	(unrot)	(M1 M2 M3 → M3 M1 M2)
5EBC6	(4roll)	(M1 M2 M3 M4 → M2 M3 M4 M1)
5EBEA	(4unroll)	(M1 M2 M3 M4 → M4 M1 M2 M3)
5ED45	(5roll)	(M1 M2 M3 M4 M5 → M2 M3 M4 M5 M1)
5ED5A	(5unroll)	(M1 M2 M3 M4 M5 → M5 M1 M2 M3 M4)
5EBFC	(N+1roll)	(M1 ... Mn+1 #n → M2 ... Mn+1 M1)
5ED6C	(N+1unroll)	(M1 ... Mn+1 #n → Mn+1 M1 ... Mn)
63911	SWAPUnNDROP	(M1 M2 → M2)
		Should be called <code>swapdrop</code> .
5E857	(rotswap)	(M1 M2 M3 → M2 M1 M3)
63F1A	metaROTDUP	(M1 M2 M3 → M2 M3 M1 M1)
		Should be called <code>rotdup</code> .
5E870	(4rollunrot)	(M1 M2 M3 M4 → M2 M1 M3 M4)

26.2 Combining functions

Address	Word	Stack and notes
5E415	top&	(M1 M2 → M1&M2)
5E4D1	pshtop&	(M1 M2 → M2&M1)

Address	Word	Stack and notes
63F2E	ROTUntop&	(M1 M2 M3 → M2 M3&M1)
63F42	roll2top&	(M1 M2 M3 → M3 M1&M2)
Also called rolltwotop&.		
5E3C0	(over&)	(M1 M2 M3 → M1&M2 M3)
5E3AC	psh&	(M1 M2 M3 → M1&M3 M2)
5E843	(overev&)	(M1 M2 M3 → M2&M1 M3)
5E490	(2top&)	(M1 M2 M3 → M1&M2&M3)
5B861	(top&pshtop&)	(M1 M2 M3 → M2&M3&M1)
5E585	(INNERtop&)	(M1 comp → M1&comp)
Explodes composite and adds to meta: INNERCOMP top&		

26.3 Meta and object operations

Address	Word	Stack and notes
5FC24	(pick1)	(ob M → ob M ob)
61305	get1	(ob M → M ob)
63105	OVER#2+UNROL	(M ob → ob M)
5E3E8	(pshml)	(M ob → ob #1 M)
62904	SWAP#1+	(M ob → M&ob)
Also called SWP1+		
5E401	psh1top&	(M ob → ob&M)
5E4A9	pull	(M&ob → M ob)
5EAF4	(pulldrop)	(M&ob → M)
5E6BB	(pullpshml)	(M&ob → ob #1 M)
5E4BD	pullrev	(ob&M → M ob)
6119E	DUP#1+PICK	(ob&M → ob&M ob)
5FA45	(pulldroppull)	(M&ob1&ob2 → M ob1)
5CC12	(2pulldrop)	(M&ob1&ob2 → M)
60F0E	ROTDROPSWAP	(M&ob1 ob2 → M&ob2)
5FA63	(revpulldrop)	(M&ob1 ob2 → M ob2)
548AA	(revpull&psh)	(M&ob1 ob2 → ob1&ob2 M)
5E706	psh1&	(M1 M2 ob → ob&M1 M2)
5E7A5	psh1&rev	(M1 M2 ob → ob&M1 M2)
57432	(addtpsh)	(M1 M2 ob → M1&ob M2)
10ADB	(rot1)	(ob M1 M2 → M1 M2 ob)
10AF9	(unrot1)	(M1 M2 ob → ob M1 M2)
5E4EA	pullpsh1&	(M1 M2&ob → ob&M1 M2)
5E503	(pullrev1&)	(M1 M2&ob → M1&ob M2)
5D6FA	(pshpullpsh1&)	(M1&ob M2 → ob&M2 M1)
5E67A	pshzer	(M → #0 M)
638FD	SWAPUnDROP	(ob M → M)
25322	(4psh)	(M1 ob1 ob2 ob3 ob4 → M2 M1)
M2 = ob1&ob2&b3&ob4		
554B3	(repl%1)	(M&ob → M&%1)

Address	Word	Stack and notes
55607	(repl%-1)	(M&ob → M&%-1)
5483C	(COLAkeep1st)	Returns and (M&ob → ob)
5FC38	(%1pshml)	(M → %1 #1 M)

The words below take as argument the next object(s) in the runstream:

Address	Word	Stack
5E51C	(addt:)	(M → M&ob)
5E530	(addt2:)	(M → M&ob1&ob2)
5E59E	(repl:)	(M&ob → M&ob')
5E549	(psh1&rev:)	(M1 M2 → M1&ob M2)
5E562	(psh1&rev2:)	(M1 M2 → M1&ob1&ob2 M2)
5DD65	(2psh1&rev:)	(M1 M2 → M1&ob M2&ob)

The words below take as argument 1LAM and/or 3LAM:

Address	Word	Stack
55288	1GETLAMSWP1+	(M → M&LAM1)
55477	(replfunc)	(M&ob → M&LAM1)
560ED	xssgneral	(M1 M2 → M1&M2&LAM1)
56101	xnsgeneral	(M → LAM3&M&LAM1)
5611F	xsngeneral	(M → M&LAM3&LAM1)
562BE	(dropaddoper)	(M1 M2 M3 → M1&M2&LAM1)
56309	(MetaUnCalc)	(M ob → LAM3 #1)

26.4 Other operations

Address	Word	Stack and description
64345	SubMetaOb	(meta #start #end → meta') Gets a sub-meta. Does range checks.
643BD	SubMetaOb1	(ob1...obi...obn #n #i #n #i → ob1...obi #n #i) This function can be used to take the first i objects of a meta, if you follow it with SWAPDROP. Example: :: %1 %2 %3 %4 %5 FIVE THREE FIVE THREE SubMetaOb1 ; → Results: %1 %2 %3 #5 #3
5F996	(tailpsh)	(meta #n → meta1 meta2) Pushes n-1 last objects in meta to meta1.

Address	Word	Stack and description
28296	metatail	(ob1...obn-i...obn #i #n+1 → ob1...ob...obn-i #n-i obn-i+1...obn #i) #n is the count of the objects in meta. Takes the last #i elements of meta and creates a new one. Example: :: %1 %2 %3 %4 %5 TWO SIX metatail ; → Results: %1 %2 %3 #3 %4 %5 TWO
584B2	(MEQU?)	(M1 M2 → M1 M2 flag) If the metas are equal (i.e., same count and equal objects) returns TRUE.
5768A	(ObInMeta?)	(M ob → M ob FLAG) Returns TRUE if ob is equal to some ob in meta.
55314	(?addinver:)	(meta&Nob → meta) (meta → meta&1LAM) If next object in the runstream is equal to first object of meta, drops that object. Otherwise, adds 1LAM to meta.
5540E	(?addrevert)	(meta&1LAM → meta&1LAM) (meta → meta&1LAM) Adds 1LAM to meta, if not already there.
5613D	(?addsimir)	(meta meta → meta) (meta1 meta2 → meta1&meta2&1LAM)
58715	(NoIdsInMeta?)	(M → M flag) If meta has any ids, lams or secondaries starting with CK0, returns FALSE.
5AD08	(dvars?)	(meta → meta flag) Returns TRUE if meta contains any LAM dvar.
5670F	(>dvars)	(meta1 meta2 → meta1&meta2') All ids in meta2 matching lam 'dvar contents are changed to LAM_'dvar. (meta1 can be #0).
5AC86	(dvars>)	(meta → meta') Lam 'dvars: are changed to 1LAM)

Dropping plus other actions:

Address	Word	Stack
169A5	NDROPFALSE	(M → FALSE)
50F60	(dropDROPf)	(ob M → FALSE)

Address	Word	Stack							
57419	(DROP2dropf)	(M1	M2	ob	→	FALSE)
57405	(2DROP2dropf)	(M1	M2	ob1	ob2	→	FALSE)
5551C	(Repl0)	(M	→	%0	#1)
55535	(Repl1)	(M	→	%1	#1)
5554E	(Repl-1)	(M	→	%-1	#1)
56183	(2Repl0)	(M1	M2	→	%0	#1)
561D8	(2Repl-1)	(M1	M2	→	%-1	#1)
5643A	(DropRepl0)	(M	ob	→	%0	#1)
5499F	(Repl0ABND)	(M	→	%0)

Chapter 27

Symbolics

27.1 General operations

Address	Word	Stack and description
055F3	(NULLSYMB)	(→ symb) Puts a null algebraic in the stack.
5E067	(SINNER)	(symb → meta) (ob → ob #1) If the argument is a symbolic, does INNERCOMP, otherwise ONE. Note that ob #1 is a meta object with only one object.
5E30C	(2SINNER)	(ob1 ob2 → meta1 meta2) SINNER for two objects.
5E2F8	(2SINNERtop&)	(ob1 ob2 → meta) Does 2SINNER then top&.
5E32A	(SINNERMETA)	(meta → meta') Explodes each object in meta with SINNER and merges the result with top&.
5F2A3	(EXPLODE)	(ob → meta) Uses recursive calls to SINNER to explode object.
5F2EE	(IMPLODE)	(meta → ob) Builds symbolic obeying VUNS properties (UNSYM element), checking fcnapply, etc. Does not build symbolic if result is a single object valid in symbolics.
5E652	symcomp	(ob → ob') If ob is symbolic, does nothing, otherwise ONESYMBN.
5E085	(CKSYMBN)	(meta → ob) If size is not one, does SYMBN, else DROPSYM.

Address	Word	Stack and description
5F384	(DROPSYM)	(ob1 ob2 → ob) Drops ob2, if ob1 if symf does nothing, else does ONESYMBN.
1CF2E	(EQ>)	(symb → arg1 arg2) Internal version of EQ->.
1CFD0	(EXPR>)	(symb → arg1 ... argn %n ob) Internal version of OBJ->.
1578D	CRUNCH	(ob → %) Internal version of ->NUM
22F68	(SYMCRUNCH1)	(ob → %) If id does XEQRCCL, then does CRUNCH for all object types.
22F86	(SYMCRUNCH2)	(ob1 ob2 → % ob2) SYMCRUNCH1 for the object in level two.
353AB	(FINDVARS)	(symb → { }) Returns a list of the variables of the equation.
5A036	uncrunch	(→) Clears numeric results flag (system flag 3) for the next command only. Example: SYMCOLCT = :: uncrunch colct ;
545A0	cknumdsptchl	(sym → symf) Used by one argument functions to evaluate a symbolic or numeric routine according to numeric results flag. Usage: :: ccknumdsptchl <sym> <num> ; If numeric mode, the object in level one is CRUNCHED and <num> is COLAd. If symbolic mode, cksevall: is called. Example: SYMRE = :: ccknumdsptchl MetaRE xRE ;
54DBC	(cksevall:)	(symf' → symf') Binds next two objects in the runstream to LAMxSYMfcn and LAMxfnc. Explodes symf, then evaluates next on Meta, then builds ob with CKSYMBN. If symf is equation next is evaluated on both sides, then equation is rebuilt (ckeval- leq1).

Address	Word	Stack and description
54E2A	(ckevaleq1)	(meta&= → symf) Evaluates 2LAM on both sides of equation, re-builds symbolic and abandons temporary environment.
558DC	sscknum2	(sym sym → symf) Used by two argument functions to evaluate function according to current numeric mode. Usage: :: sscknum2 <sym> <num> ; In numeric mode both arguments are CRUNCHED and <num> is COLAD. Else, cksseval2: is called. Example: SYM+ = :: sncknum2 Meta+ x+ ;
558F5	sncknum2	(sym % → symf) Usage: :: sncknum2 <sym> <num> ; In symbolic mode uses cksneval2:. Example: SYM+O = :: sncknum2 Meta+Con x+ ;
5590E	nscknum2	(% sym → symf) Usage: :: nscknum2 <sym> <num> ; In symbolic mode uses cknseval2:. Example: O+SYM = :: nscknum2 Con+Meta x+ ;
55657	(cknum2:)	(symf symf → symf) Used by the three above functions to determine (and possibly to CRUNCH) the program to COLA.
557EC	(cksseval2:)	(sym sym → symf) Binds next two objects in the runstream to LAMxSYMfcn and LAMxfcn. Explodes the objects in the stack, and evaluates next object in the runstream. If either is an equation, ckevaleq2 is called. Rebuilds one symbolic.
5576F	(cksneval2:)	(sym % → symf) Binds % and next two objects in the runstream to LAMsc1, LAMxSYMfcn and LAMxfcn. Explodes sym, evaluates LAMxSYMfnc, rebuilds symbolic. If sym is equation, ckevaleq1 is called.
5575B	(cknseval:)	(% sym → symf) Does SWAP then cknseval2:.

Address	Word	Stack and description
58CE4	(parameval)	(sym param → ?) Ensures sym is symbolic (using symcomp), then executes param on each element of symbolic. param is bound to 1LAM during the loop. param should return a flag. If TRUE, or if the object in level 1 is not an operator the loop continues, else possible COLCT property is executed. (Better return TRUE always).
58CEE	(eval)	(sym → ?) Like parameval, but without binding of a new param. Use this for recursive evaluation with the same parameter. (See SHOWLS and show-param for examples).
5918A	(evalTRUE)	(sym → ? TRUE) Used for recursive parameval.

27.2 Mathematical operations

One argument meta functions assume function in 1LAM. Two argument meta functions for constants also assume Con in 3LAM.

Address	Symbolic funct.	Address	Meta funct.
55F2B	(SYM+O)	56543	(Meta+Con)
55F44	(O+SYM)	56331	(Con+Meta)
55F5D	(SYM+)	56160	(Meta+)
55F76	(SYM-O)	56566	(Meta-Con)
55F85	(O-SYM)	56359	(Con-Meta)
55F8F	(SYM-)	56174	(Meta-)
55FC1	(SYM*O)	56589	(Meta*Con)
55FDA	(O*SYM)	56390	(Con*Meta)
55FF3	(SYM*)	561BA	(Meta*)
5600C	(SYM/O)	565CF	(Meta/Con)
56025	(O/SYM)	563DB	(Con/Meta)
5603E	(SYM/)	56214	(Meta/)
55EE0	(SYM^O)	5645D	(Meta^Con)
55EF9	(O^SYM)	562FA	(Con^Meta)
55F12	(SYM^)		--
56057	(SYM%MOD)	5660B	(MetamodCon)
56070	(%SYMMOD)	5642B	(ConmodMeta)
56089	(SYMMOD)	56250	(Metamod)
55E95	(SYM%MIN)		--
55EAE	(%SYMMIN)		--
55EC7	(SYMMIN)		--
55E4A	(SYM%MAX)		--

Address	Symbolic funct.	Address	Meta funct.
55E63	(%SYMMAX)		--
55E7C	(SYMMAX)		--
55C3D	(SYM%%OF)		--
55C56	(%SYM%OF)		--
55C6F	(SYM%OF)		--
55C88	(SYM%%CH)		--
55CA1	(%SYM%CH)		--
55CBA	(SYM%CH)		--
55CD3	(SYM%%T)		--
55CEC	(%SYM%T)		--
55D05	(SYM%T)		--
55D1E	(SYM%COMB)		--
55D37	(%SYMCOMB)		--
55D50	(SYMCOMB)		--
55D69	(SYM%PERM)		--
55D82	(%SYMPERM)		--
55D9B	(SYMPERM)		--
55DB4	(SYM%RND)		--
55DCD	(SYMRND)		--
55DE6	(RNDSYM)		--
55DFF	(SYM%TRNC)		--
55E18	(TRCNYM)		--
55E31	(SYMTRCN)		--
560A2	(SYM%XROOT)		--
560BB	(%SYMXROOT)		--
560D4	(SYMXROOT)		--
54EEB	(SYMNEG)	553D2	(MetaNEG)
54F04	(SYMABS)	555B2	(MetaABS)
54F68	(SYMSIGN)		--
54F36	(SYMINV)	553EB	(MetaINV)
5518E	(SYMIP)		--
551A7	(SYMFP)		--
551C0	(SYMFLOOR)		--
551D9	(SYMCEIL)		--
551F2	(SYMEXPONENT)		--
5520B	(SYMMANT)		--
54AE0	(SYMRE)	5542C	(MetaRE)
54EB9	(SYMIM)	55495	(MetaIM)
54F1D	(SYMCONJ)	55567	(MetaCONJ)
54ED2	(SYMNOT)		--
54F9A	(SYMSQ)	555E9	(MetaSQ)
54F81	(SYMSQRT)		--
54FB3	(SYMSIN)	5533C	(MetaSIN)
54FCC	(SYMCOS)	55378	(MetaCOS)
54FE5	(SYMTAN)	553A5	(MetaTAN)
55049	(SYMASIN)		--
55062	(SYMACOS)		--
5507B	(SYMATAN)		--

Address	Symbolic funct.	Address	Meta funct.
54FFE	(SYMSINH)	5529C	(MetaSINH)
55017	(SYMCOSH)	552B0	(MetaCOSH)
55030	(SYMTANH)	552C4	(MetaTANH)
55094	(SYMASINH)		--
550AD	(SYMACOSH)		--
550C6	(SYMATANH)		--
550F8	(SYMLN)		--
55143	(SYMLNPL)		--
550DF	(SYMEXP)	552D8	(MetaEXP)
5515C	(SYMEXPM)	55300	(MetaEXPM)
55111	(SYMLOG)		--
5512A	(SYMALOG)	552EC	(MetaALOG)
55175	(SYMFACT)		--
55224	(SYMD>R)		--
5523D	(SYMR>D)		--
54F4F	(SYMARG)		--
55256	(SYMUBASE)		--
5226F	(SYMUVAL)		--
5599A	(SYM%AND)		--
559B3	(%SYMAND)		--
559CC	(SYMAND)		--
559E5	(SYM%OR)		--
559FE	(%SYMOR)		--
55A17	(SYMOR)		--
55A30	(SYM%XOR)		--
55A49	(%SYM XOR)		--
55A62	(SYM XOR)		--
55A7B	(SYMFLOAT==)		--
55A94	(FLOATSYM==)		--
55AAD	(SYM==)		--
55AC6	(SYMFLOAT<>)		--
55ADF	(FLOATSYM<>)		--
55AF8	(SYM<>)		--
55B11	(SYM%<)		--
55B2A	(%SYM<)		--
55B43	(SYM<)		--
55B5C	(SYM%>)		--
55B75	(%SYM>)		--
55B8E	(SYM>)		--
55BA7	(SYM%<=)		--
55BC0	(%SYM<=)		--
55BD9	(SYM<=)		--
55BF2	(SYM%>=)		--
55C0B	(%SYM>=)		--
55C24	(SYM>=)		--

27.2.1 Collection

Address	Word	Stack and description
57D90	SYMCOLCT	(symf → symf) :: uncrunch colct ;
57DA4	(colct)	(symf → symf) Basic collection function, does not check numeric results flag. Disassembly: :: EXPLODE pshzer colfac pshzer colrev ATTNFLG@ #0<> case :: CKSYMBN CK0NOLASTWD ?ATTNQUIT ; pshzer colunfac SYMN COLA coleval ;
587AA	(colfac)	(meta1 meta2 → meta') Appends objects in meta2 tail to meta1 tail replacing all -, /, NEG, INV and SQ with +, *, ^, and -1 as a possible factor. Example rules: <div style="margin-left: 100px;"> 'SQ(A)' → 'A^2' '-A' → '-1*A' 'A-B' → 'A+-1*b' 'A/B' → 'A*B^-1' </div>
57E08	(colrev)	(meta1 meta2 → meta') Appends objects in meta2 to tail of meta1 collecting numeric factors, ordering terms according to a comparison function, collecting numeric terms to front. Only + and * factors are checked. Sub-routines used by this function: <div style="margin-left: 100px;"> 58511 (MetaLess?) (M1 M2 → M1 M2 flag) 58525 (MetaMore?) (M1 M2 → M1 M2 flag) 585A7 (BodyMore?) (ob1 ob2 → flag) </div>
58A20	(colunfac)	(meta1 meta2 → meta') Appends objects in meta2 to head of meta1 converting ^, + and * to / and - when suitable.
58CDA	(coleval)	(ob → ob') Passes FALSE as parameter to parameval. Thus eval uses ?COLCT to check special evaluation.

27.2.2 Expansion

Address	Word	Stack and description
57A0C	(SYMEXPAN)	(symf → symf) Expands symbolic or float
57A48	(expan)	(meta1 meta2 meta3 → meta) Expands meta3. Successful part is added to tail of meta2. Calls <code>expan1</code> and <code>larg</code> until meta3 becomes empty.
57AA2	(expan1)	(meta → meta1 meta2) Expands meta. Meta1 is the unsuccessful part, meta2 the successful part (could be just and operator). Sub-expanders:
57B63	(?expan^)	If ^ then expands (returns if successful.)
57AB6	(expansq)	Expands <code>SQ</code> .
5BFD8	(MetaD->)	
5C0B9	(Meta<-D)	
5C2CE	(MetaE^)	
5C348	(MetaL*)	
57B4C	(?expanneginv)	Prevents <code>Meta->()</code> from expanding <code>[Expr INV NEG]</code> .
5C137	(Meta->())	
57B01	(?expanapp)	If <code>xFCNAPPLY</code> then tries calling <code>?EXPAN</code> .
57C71	(expansum^)	Expands $(A+B)^2$ or $(A-B)^2$
57CF8	(NXTPOT%)	Returns next number when expanding ^.
	(% → flag %' TRUE)	
	(% → % FALSE)	
		The flag indicates wheter %0>. Do not use for %0.

27.2.3 Integration

Address	Word	Stack and notes
1F201	(XEQINTEGRID)	(ob ob ob id/lam → symf)
1F27A	(XEQINTEG)	(ob ob ob QN → symf)
5AAC7	(SYMINTEG)	(symf symf symf QN → symf)

Address	Word	Stack and notes
5662E	(NUMINTEG)	(symf QN symf_lo symf_hi → %)
52C36	(CALCINTEG)	(seco %accuaracy %lo %hi → %integral %error) Low level numeric integration. If %low = %hi returns %0 %0. Checks that $1E-12 \leq \%accuracy \leq 1$. seco gets % as input and should return one value.
5ACC7	(intg)	(#0 #0 meta → meta_ok meta_fail) Integrates meta where variable of integration has been changed to LAMdvar. Meta objects should be merged by addition. Use colunfac to resume /, -, etc. from *, +.
5D0C2	(forceadd)	(meta → meta') Forces top level operators to be +, NEG when possible by changing from -, +, NEG. Attempts to arrange rightmost term to be second argument for top +. Example: 'A+(B+C)' → 'A+B+-C'
5B659	(forcemul?aga)	(meta → meta') Recursive Meta<-D, MetaD-> and forcemul calling. If any operation was successful AGAIN is executed.
5B717	(forcemul)	(meta → meta') Forces top level operator to be + and NEG when possible by changing from / and INV. LAMdvar is ordered specially.
5AFAB	(intg1)	(M_ok M_fail M_temp meta → M_ok' M_fail' M_temp) Integrates meta, ok part is adds to meta1 (meta3 is the next part to integrate in the top level loop.)
5B0FA	(intg1ok)	(M1 M2 M3 M4 → M1' M2 M3 TRUE) Adds M4 to M1. (Successful intg1).
5B09B	(intg1fail)	(M1 M2 M3 M4 → M1 M2' M4 TRUE) Adds M4 to M2. (Unsuccessful intg1).
5B0CD	(intgconst)	(M_ok M_fail M_temp meta → M_ok' M_fail' M_temp) Integrates constant to meta. (dvars? gives FALSE).

Address	Word	Stack and notes
5B131	(intglinear)	(M1 M2 M3 M4 → M1' M2 M3) Integrates linear term (M4).
5B140	(intgaddlin)	(meta #loc → meta') Adds $2^{1/2}$ to LAMdvar in meta at stack level #loc.
5AD80	(linear?)	(meta #level → meta' TRUE) (meta #level → meta' #loc FALSE) Is meta linear in LAMdvar? #level is first location of LAMdvar obtained from dvars? :: linear DUP IT SWAPDROP ;
5AD9E	(linear)	(meta #level → meta' #loc flag)
5AD6C	(linear!)	(meta #level → meta' flag) :: linear SWAPDROP ;

27.2.4 Other functions

Address	Word	Stack and notes
1F38B	(SYMWHERE)	(symf { } → symf)
1F439	(XEQSYMBWHERE)	(symf QN1 id1 ... QNn idn → symf)
1F43E	(CKWHEREARGS)	Checks pairs of quoted names/ids.
547B5	SYMBWHERE	(symf QN1 id1 ... QNn idn #2n+1 → symf)
547E2	(WHERE1)	(QN1 id1 ... QNn idn #n meta1 → symf) Used when meta size is 1.
54887	(WHERE1)	(QN1 id1 QNn idn #2 metan → symf)
58D75	SYMSHOW	(sym id/lam → symf)
20B00	XEQSHOWLS	(sym { } → symf)
5910B	(SHOWLS)	(sym {names} → symf) See this for a good example of recursive parameval.
1A4A3	(%IFTE)	(% ob1 ob2 → ?)
54564	(SYMIFTE)	(sym symf symf → symf) Uses cknumdsptchl with: 54609 (MetaIFTE) 54653 (NumIFTE)
591AD	(SYMQUAD)	(sym id → symf) Avoids the obvious in solving a quadratic equation.
595DD	(SYMTAYLR)	(sym id % → symf) Calculates taylor polynomial.

Address	Word	Stack and notes
57293	(SYMISOL)	(sym id \rightarrow symb) Isolate a variable.
1F113	(XEQSYMDERCON)	(QN %/C%/unit \rightarrow symf)
1F0F5	(XEQSYMDERSTEP)	(QN sym \rightarrow symf)
54977	(SYMDERSTEP)	(QN sym \rightarrow symf) No CKSYMBTYPE check.
54954	(SYMDER)	(sym sym \rightarrow symf)
56949	(SYMSUM)	(sym sym sym ob \rightarrow symf)
56A06	(SYM%SUM)	(sym sym % ob \rightarrow symf)
56A4C	(%SYMSUM)	(sym % sym ob \rightarrow symf)
56AC9	(%SUM)	(sym % % ob \rightarrow symf)

27.3 Meta symbolics functions

27.3.1 Adding operators

Address	Word	Stack
5BC94	(addt+)	(meta \rightarrow meta&+)
5BC67	(addt-)	(meta \rightarrow meta&-)
5CD16	(addt*)	(meta \rightarrow meta&*)
5CD2A	(addtNEG)	(meta \rightarrow meta&NEG)
5CD3E	(addtINV)	(meta \rightarrow meta&INV)
5BCC1	(repl/)	(meta&ob \rightarrow meta&/)
5BCEE	(repl*)	(meta&ob \rightarrow meta&*)

27.3.2 Changing operators

Address	Word	Stack
5ACD6	(M1st+?Drp)	(meta&+ \rightarrow meta) (meta \rightarrow meta)
5BC5D	(meta+)	(meta&NEG \rightarrow meta&-) (meta \rightarrow meta&+)
5BC8A	(meta-)	(meta&NEG \rightarrow meta&+) (meta \rightarrow meta&-)
5BCB7	(meta*)	(meta&INV \rightarrow meta&/) (meta \rightarrow meta&*)
5BCE4	(meta/)	(meta&INV \rightarrow meta&*) (meta \rightarrow meta&/)
5BD3E	(drpmeta+)	(meta&NEG&ob \rightarrow meta&-) (meta&ob \rightarrow meta&+)

Address	Word	Stack
5BD57	(drpmeta-)	(meta&NEG&ob → meta&+)
		(meta&ob → meta&-)
5BD70	(drpmeta*)	(meta&INV&ob → meta&/)
		(meta&ob → meta&*)
5BD89	(drpmeta/)	(meta&INV&ob → meta&*)
		(meta&ob → meta&/)
5BBE5	(metaneg)	(meta&NEG → meta)
		(meta → meta&NEG)
5BC3F	(metainv)	(meta&INV → meta)
		(meta → meta&INV)
5BC03	(metaneglft)	(meta → meta')
		metaneg on left sub-expression.
5BC21	(metainvlft)	(meta → meta')
		metainv on left sub-expression.

27.3.3 Splitting algebraic metas

Address	Word	Stack and description
5EA9F	pshzerpsharg	(meta → M_last M_rest) Pushes last sub-expression in meta. If meta is a valid expression M_rest will be empty.
63F92	pZpargSWAPUn	(meta → M_rest M_last) pshzerpsharg then psh.
63F56	plDRPpZparg	(meta&ob → M_last M_rest) Drops ob then calls pshzerpsharg.
5E68E	(pargop)	(meta → M_last&op M_rest) Pushes last sub-expression ignoring first object in meta. Thus op is +, -, etc. and M_last is their second argument.
5EAC2	(larg)	(meta → M_rest M_last) Splits last sub-expression from meta.
5E6F2	(parg&)	(meta1 meta2 → meta1&M_last M_rest)
5CCEE	(larg&)	(meta1 meta2 → meta1&M_rest M_last)
5CBF9	(drppargtop&)	(meta&ob → M_last&M_rest)
57F4B	(swappargunrot)	(meta1 meta2 → M_rest meta2 M_last)
1CF42	(drppargsym)	(meta&ob → 'M_Rest' 'MetaLast') Buids objects with PSYMBN. Will give invalid expressions if ob is not a two-argument function.

Address	Word	Stack and description
5F926	(splitup)	(meta #n #m → meta #level) Calculates stack level of last object to be included when splitting last m sub-expressions from meta starting from stack level n. (2 1 would give level of first object in the last sub-expression.)
5F96E	(splitdown)	(meta #n #m → meta #lowlevel #args+1) Seeks stack level n-1 downwards for extra operators for m expressions. #lowlevel is the stack level of the extra operator. #args indicates how many expressions the lowlevel operator is still missing.
558BE	(?spliteq)	(meta1&meta2&= → meta2 meta1) (meta → meta meta) If meta contains =, splits two sides, otherwise DUP.

27.3.4 Miscellaneous

Address	Word	Stack and description
58C02	(count+)	(meta → meta #0) (meta &+&+...&+ → meta #n)
58C0E	(count*)	Same as above for *.

27.3.5 Rules menu operations

Stack diagrams for functions below: (meta → meta')

For matching patterns the corresponding operation is COLAD, thus current stream is dropped in such cases.

Address	Word	Action
5BE56	(MetaMulInv)	Simplify combinations of INV and * (using /). Sub-functions: 58A61 (colinv1) [expr1 INV expr2 INV *] → [expr1 expr2 /] 58A93 (colinv2) [expr INV *] → [expr /] 58AAC (colinv3) [expr1 INV expr2 *] → [expr1 expr2 /]
5971D	(MetaDNEG)	Double negation.
5976B	(MetaDINV)	Double inversion.

Address	Word	Action
597B5	(Meta*1)	Multiply by one.
5983B	(Meta^1)	Raise to power of one.
59885	(Meta1/)	Divide by one.
5990F	(Meta+1-1)	Add one and subtract one.
596D3	(MetaRCOLCT)	Restricted collection.
5C6D9	(Meta<-T)	Move nearest right term to the left.
5C68D	(MetaT->)	Move nearest left term to the right.
5C623	(Meta(()))	Put parentheses over nearest term.
5C589	(Meta(<-)	Include left term.
5C5D6	(Meta->))	Include right term.
5BE81	(Meta<-->)	Commute terms.
5BECE	(Meta<-A)	Associate left term.
5BF53	(MetaA->)	Associate right term.
5C137	(Meta->())	Remove prefix.
5C0B9	(Meta<-D)	Delete left term (via expansion).
5C102	(Meta<-D!)	Delete left term (above - ^ expansion).
5BFD8	(MetaD->)	Delete right term (via expansion).
5C3C2	(Meta<-M)	Merge common factor on left side.
5C4CF	(MetaM->)	Merge common factor on right side.
5C261	(Meta-())	Double negate, then remove prefix.
5C204	(Meta1/())	Double inversion, then remove prefix.
5C348	(MetaL*)	Transform $\text{LN}(A^B)$ to $\text{LN}(A) * B$.
5C375	(MetaL())	Transform $\text{LN}(A) * B$ to $\text{LN}(A^B)$.
5C2CE	(MetaE^)	Transform $\text{EXP}(A * B)$ to $\text{EXP}(A)^B$.
5C31B	(MetaE())	Transform $\text{EXP}(A)^B$ to $\text{EXP}(A * B)$.
5C670	(Meta->TRG)	Change EXP to trigonometric functions.
5C53C	(MetaAF)	Add fractions.
5C845	(Meta->DEF)	Define function (SIN, SINH, ASIN...)
5C91D	(MetaTRG*)	Expand trigonometric function of a sum.
5C73D	(Meta->()C%)	Remove first RE, IM or CONJ.

Words for repeated evaluation:

Address	Word	Address	Word
5CDF2	(Meta<-Dall)	5CEF1	(MetaD->all)
5CE15	(Meta<-Aall)	5CE4C	(MetaA->all)
5CFF5	(Meta<-Mall)	5D009	(MetaM->all)
5CF5A	(Meta<-Tall)	5CF23	(MetaT->all)
5CEBA	(Meta(<-all)	5CE83	(Meta->)all)
5CF91	(Meta->()all)	5CFC3	(Meta->()C%all)

The repeater words:

Address	Word	Stack and description
5CD52	(evalcase:)	(meta → ?) Evaluates next object. If it drops current stream then continue, else <code>SKIP</code> next. Example: :: evalcase: Meta<-D Meta<-Daga ;
5CD7A	(revalcase:)	(meta → ?) Evaluates next object for sub-expressions until current stream is not dropped by ob. Example: Meta<-Daga = :: revalcase: Meta<-D COLA RDROP ; (COLA RDROP is there to makr successful oper.)

Chapter 28

Library and backup objects

28.1 Port operations

Address	Word	Stack and description
0AAB2	PORTSTATUS	(#port → present? writeable? merged? #size #addr?) Returns information for port.
0AB22	(PORTEND)	(#port → #addr) Gets end address of port.
0AB82	NEXTLIBBAK	(#addr → backup/library #nextaddr TRUE/FALSE) Gets next library or backup.
0B409	(MERGE)	(#port →) Merges specified port. Only works for port one. Checks if wrong port number was entered.

28.2 ROM pointers

Address	Word	Stack and description
07E50	#>ROMPTR	(#lib #cdm → ROMPTR) Creates ROM pointer.
08CCC	ROMPTR>#	(ROMPTR → #lib #cmd) Splits ROM pointer.
07EE9	ROMPTR@	(ROMPTR → ob TRUE) (ROMPTR → FALSE) Recall contents of ROM pointer.
62C19	DUPROMPTR@	(ROMPTR → ROMPTR ob TRUE) (ROMPTR → ROMPTR FALSE) Does DUP then ROMPTR@.

Address	Word	Stack and description
02FEF	(DoRomptr)	(ROMPTR → ?) Recalls contents of ROM pointer and EVAL. Generates “Undefined XLIB Error” if not found.
62A61	?>ROMPTR	(ob → ob') If ROM_WORD? and TYPECOL? then RPL@.
62A84	?ROMPTR>	(ob → ob') If TYPEROMP? and contents exit INHARDROM? then return contents.
62BD8	RESOROMP	(→ ob) Recalls contents of next object in the run-stream (which must be a ROM pointer).
07E76	(PTR>ROMPTR)	(ob → ROMPTR TRUE) (ob → FALSE) If the object is a library command, returns its ROM pointer and TRUE, if not just FALSE.
081FB	(ROMPTRDECOMP)	(ROMPTR → id TRUE) (ROMPTR → FALSE) If the library command exists and has a name, returns that name and TRUE, otherwise FALSE.
081E3	(PTR>ID)	(ob → id TRUE) (ob → FALSE) If the object is a library command and has a name, returns its name and TRUE, if not returns just FALSE.
07C18	(ID>CMD)	(id → id TRUE) (id → ROMPTR TRUE) (id → FALSE) Searches id in current path, if found returns TRUE. Else searches attached libraries. If nothing was found, return FALSE.

28.3 Libraries

Address	Word	Stack and description
07709	TOSRRP	(# →) Attaches library to HOME directory.
076AE	OFFSRRP	(# →) Detaches library from HOME directory.

Address	Word	Stack and description
08157	(GETCONFIG)	(#libnum → ob TRUE) (#libnum → FALSE) Gets specified library's configuration routine.
07F86	(ROMPART)	(rrp → { #lib1 ... #libn } TRUE) (ROMPTR → #libnum) Gets library's number.

28.4 Backup objects

Address	Word	Stack and description
081D9	BAKNAME	(bak → id TRUE) Returns backup's name
0948E	BAK>OB	(bak → ob) Get's backup object.
21674	(>BAK)	(id ob → bak) Creates backup object with specified name and contents.

Chapter 29

Stack operations

Address	Word	Stack and notes
0314C	DEPTH	(1 ... n → 1 ... n #n)
5DE7D	reversym	(1 ... n #n → n ... 1)
03188	DUP	(1 → 1 1)
62CB9	DUPDUP	(1 → 1 1 1)
5E370	NDUPN	(ob #n → ob ... ob #n)
		(ob #0 → #0)
62FB1	DUPROT	(1 2 → 2 2 1)
630F1	DUPROLL	(1 ... n #n → 1 3 ... n #n 2)
61380	DUPUNROT	(1 2 → 2 1 2)
		Also called SWAPOVER.
61099	DUP4UNROLL	(1 2 3 → 3 1 2 3)
611F9	DUP3PICK	(1 2 → 1 2 2 1)
		Also called 2DUPSWAP.
630DD	DUPPICK	(n ... 1 #n → n ... 1 #n n-1)
6119E	DUP#1+PICK	(n ... 1 #n → n ... 1 #n n)
5FC24	(DUP#2+PICK)	(n ... 1 #n → n ... 1 #n n+1)
031AC	2DUP	(1 2 → 1 2 1 2)
611F9	2DUPSWAP	(1 2 → 1 2 2 1)
		Also called DUP3PICK.
63C40	2DUP5ROLL	(1 2 3 → 2 3 2 3 1)
031D9	NDUP	(1 ... n #n → 1 ... n 1 ... n)
03244	DROP	(1 →)
627A7	DROPDUP	(1 2 → 1 1)
63FA6	DROPNDROP	(1 ... n #n ob →)
6270C	DROPSWAP	(1 2 3 → 2 1)
62726	DROPSWAPDROP	(1 2 3 → 2)
		Also called ROT2DROP and $XYX>Y$.
62FC5	DROPROT	(1 2 3 4 → 2 3 1)
63029	DROPOVER	(1 2 3 → 1 2 1)
03258	2DROP	(1 2 →)
60F4B	3DROP	(1 2 3 →)
		Also called $XYZ>$.
60F7E	4DROP	(1 ... 4 →)
		Also called $XYZW>$.
60F72	5DROP	(1 ... 5 →)
60F66	6DROP	(1 ... 6 →)
60F54	7DROP	(1 ... 7 →)
0326E	NDROP	(1 ... n #n →)
62F75	N+1DROP	(ob 1 ... n #n →)
		Also called #1+NDROP.

Address	Word	Stack and notes
03223	SWAP	(1 2 → 2 1)
62747	SWAPDUP	(1 2 → 2 1 1)
6386C	SWAP2DUP	(1 2 → 2 1 2 1)
60F9B	SWAPDROP	(1 2 → 2)
Also called XY>Y.		
62830	SWAPDROPDUP	(1 2 → 2 2)
6284B	SWAPDROPSWAP	(1 2 3 → 3 1)
Also called XYZ>ZX.		
60F33	SWAPROT	(1 2 3 → 3 2 1)
Also called UNROTSWAP and XYZ>ZYX.		
63C2C	SWAP4ROLL	(1 2 3 4 → 2 4 3 1)
Also called XYZW>YWZX.		
61380	SWAPOVER	(1 2 → 2 1 2)
Also called DUPUNROT.		
63C54	SWAP3PICK	(1 2 3 → 1 3 2 1)
62001	2SWAP	(1 2 3 4 → 3 4 1 2)
03295	ROT	(1 2 3 → 2 3 1)
62775	ROTDUP	(1 2 3 → 2 3 1 1)
62C7D	ROT2DUP	(1 2 3 → 2 3 1 3 1)
60F21	ROTDROP	(1 2 3 → 2 3)
Also called XYZ>YZ.		
60F0E	ROTDROPSWAP	(1 2 3 → 3 2)
62726	ROT2DROP	(1 2 3 → 2)
Also called DROPSWAPDROP and XYZ>Y.		
60EE7	ROTSWAP	(1 2 3 → 2 1 3)
Also called XYZ>YXZ.		
6112A	ROTROT2DROP	(1 2 3 → 3)
Also called UNROT2DROP and XYZ>Z.		
62CA5	ROTOVER	(1 2 3 → 2 3 1 3)
60FBB	4ROLL	(1 2 3 4 → 2 3 4 1)
Also called FOURROLL and XYZW>YZWX.		
62864	4ROLLDROP	(1 2 3 4 → 2 3 4)
62ECB	4ROLLSWAP	(1 2 3 4 → 2 3 1 4)
63001	4ROLLROT	(1 2 3 4 → 2 4 1 3)
Also called FOURROLLROT.		
630A1	4ROLLOVER	(1 2 3 4 → 2 3 4 1 4)
60FD8	5ROLL	(1 2 3 4 5 → 2 3 4 5 1)
Also called FIVEROLL.		
62880	5ROLLDROP	(1 2 3 4 5 → 2 3 4 5)
61002	6ROLL	(1 ... 6 → 2 ... 6 1)
Also called SIXROLL.		
6106B	7ROLL	(1 ... 7 → 2 ... 7 1)
Also called SEVENROLL.		
6103C	8ROLL	(1 ... 8 → 2 ... 8 1)
Also called EIGHTROLL.		
03325	ROLL	(1 ... n #n → 2 ... n 1)
62F89	ROLLDROP	(1 ... n #n → 2 ... n)
62D45	ROLLSWAP	(1 ... n #n → 2 ... n-1 1 n)
612F3	#1+ROLL	(ob 1 ... n #n → 1 ... n ob)

Address	Word	Stack and notes
61318	#2+ROLL	(a b 1 ... n #n → b 1 ... n a)
612DE	#+ROLL	(1 ... n+m #n #m → 2 ... n+m 1)
612CC	#-ROLL	(1 ... n-m #n #m → 2 ... n-m 1)
60FAC	UNROT	(1 2 3 → 3 1 2) Also called 3UNROLL and XYZ>ZYX
62CF5	UNROTDUP	(1 2 3 → 3 1 2 1)
6284B	UNROTDROP	(1 2 3 → 3 1) Also called SWAPDROPSWAP and XYZ>ZX.
6112A	UNROT2DROP	(1 2 3 → 3) Also called ROTROT2DROP and XYZ>Z.
60F33	UNROTSWAP	(1 2 3 → 3 2 1) Also called SWAPROT and XYZ>ZXY.
60F0E	UNROTSWAPDRO	(1 2 3 → 3 2) Also called XYZ>ZY.
6308D	UNROTOVER	(1 2 3 → 3 1 2 1)
6109E	4UNROLL	(1 2 3 4 → 4 1 2 3) Also called FOURUNROLL and XYZW>WXYZ.
62D09	4UNROLLDUP	(1 2 3 4 → 4 1 2 3 3)
6113C	4UNROLL3DROP	(1 2 3 4 → 4) Also called XYZW>W.
63015	4UNROLLROT	(1 2 3 4 → 4 3 2 1)
610C4	5UNROLL	(1 2 3 4 5 → 5 1 2 3 4) Also called FIVEUNROLL.
610FA	6UNROLL	(1 ... 6 → 6 1 ... 5) Also called SIXUNROLL.
62BC4	7UNROLL	(1 ... 7 → 7 1 ... 6) Also called SEVENUNROLL.
63119	8UNROLL	(1 ... 8 → 8 1 ... 7)
6312D	10UNROLL	(1 ... 10 → 10 1 ... 9)
0339E	UNROLL	(1 ... n #n → n 1 ... n-1)
61353	#1+UNROLL	(ob 1 ... n #n → n ob 1 ... n-1)
61365	#2+UNROLL	(a b 1 ... n #n → n a b 1 ... n-1)
6133E	#+UNROLL	(1 ... n+m #n #m → n+m 1 ... n+m-1)
6132C	#-UNROLL	(1 ... n-m #n #m → n-m 1 ... n+m-1)
032C2	OVER	(1 2 → 1 2 1)
62CCD	OVERDUP	(1 2 → 1 2 1 1)
62D31	OVERSWAP	(1 2 → 1 1 2) Also called OVERUNROT.
63105	OVER#2+UNROLL	(1 ... n #n ob → ob 1 ... n #n)
63C90	OVER5PICK	(1 2 3 4 → 1 2 3 4 3 1)
63FBA	2OVER	(1 2 3 4 → 1 2 3 4 1 2)
611FE	3PICK	(1 2 3 → 1 2 3 1)
62EDF	3PICKSWAP	(1 2 3 → 1 2 1 3)
630B5	3PICKOVER	(1 2 3 → 1 2 3 1 3)
63C68	3PICK3PICK	(1 2 3 → 1 2 3 1 2)
6121C	4PICK	(1 2 3 4 → 1 2 3 4 1)
62EF3	4PICKSWAP	(1 2 3 4 → 1 2 3 1 4)
63069	4PICKOVER	(1 2 3 4 → 1 2 3 4 1 4)
6123A	5PICK	(1 2 3 4 5 → 1 2 3 4 5 1)

Address	Word	Stack and notes
6125E	6PICK	(1 ... 6 → 1 ... 6 1)
61282	7PICK	(1 ... 7 → 1 ... 7 1)
612A9	8PICK	(1 ... 8 → 1 ... 8 1)
032E2	PICK	(1 ... n #n → 1 ... n 1)
611A3	#1+PICK	(1 ... n #n-1 → 1 ... n 1)
611BE	#2+PICK	(1 ... n #n-2 → 1 ... n 1)
611D2	#3+PICK	(1 ... n #n-3 → 1 ... n 1)
611E1	#4+PICK	(1 ... n #n-4 → 1 ... n 1)
61184	#+PICK	(1 ... n+m #n #m → 1 ... n+m 1)
61172	#-PICK	(1 ... n-m #n #m → 1 ... n-m 1)

Chapter 30

Checking for arguments

Address	Word	Description
18AE1	CK0	Saves current command to LASTCKCMD. Verify that there are at least <n> objects in the stack, if not generates a “Too Few Arguments” error. Saves stack mark to STACKMARK. If Last Arg is enabled then saves the arguments.
18AA5	CK1	
18A80	CK2	
18A5B	CK3	
18B92	CK4	
18B6D	CK5	
16C34	CKN	Checks for a real in level one. Then checks for that number of arguments.
18A15	CK0NOLASTWD	Like the above, but does not save current command.
18AB2	CK1NOLASTWD	
18A8D	CK2NOLASTWD	
18A68	CK3NOLASTWD	
18B9F	CK4NOLASTWD	
18B7A	CK5NOLASTWD	
18C4A	CKNNOLASTWD	
18F9D	CK&DISPATCH0	Dispatches on stack argument. See Chapter 4 for instructions on how to use.
18FB2	CK&DISPATCH1	Dispatches on stack arguments, stripping tags if necessary.
18ECE	CK1&Dispatch	Combines CK<n> with CK&DISPATCH1.
18EDF	CK2&Dispatch	
18EF0	CK3&Dispatch	
18F01	CK4&Dispatch	
18F12	CK5&Dispatch	
5EA09	CKINFARGS	Gets meta as argument and checks its length (using DEPTH), and errors if it is too short. Collects the arguments to a list, does CK1NOLASTWD, and explodes the meta again.
1884D	0LASTOWDOB!	Clears command save by last CK<n> command. Also called 0LastRowWord!.
40BC9	AtUserStack	:: CK0NOLASTWD 0LASTOWDOB! ;
1592D	CK1NoBlame	:: 0LASTOWDOB! CK1NOLASTWD ;

Address	Word	Description
62474	'RSAVEWORD	Stores first object in the composite above the actual to LASTCKCMD. Also called 'RSaveRomWrd.
18F23	EvalNoCK	(comp \rightarrow ?) Evaluates composite without saving as current command. If first command is CK<n>&Dispatch it is replaced by CK&DISPATCH1. If first command is CK<n> it is skipped.
18F6A	(EvalNoCK:)	EvalNoCK with the next object in the run-stream as argument.

30.1 Type checking

The words below check for a specified argument type, and call SETTTYPEERR if it is of the wrong type, i.e., a “Bad Argument Type” error is generated.

Address	Word	Stack and description
63B2D	CKREAL	(ob \rightarrow ob) Checks for real.
193C1	(CKARRY)	(ob \rightarrow ob) Checks for array.
194BB	(CKRARRY)	(ob \rightarrow ob) Checks for real array.
194D9	(CKCARRY)	(ob \rightarrow ob) Checks for complex array.
19443	(CKLIST)	(ob \rightarrow ob) Checks for list.
20BE0	(CKNAMELIST)	(ob \rightarrow ob) Checks for non-empty list of names.
1945C	(CKLISTTYPE)	(ob #prolog \rightarrow ob) Checks for non-empty list of certain type.
1F05B	CKSYMBTYPE	(ob \rightarrow ob) Checks for quoted name (name as symbolic).

Address	Word	Stack and description
54C63	nmetasym	(meta → meta) Checks for meta containing %, C%, unit, id, lam or symb.
19207	(CKNFLOATS)	(ob1 ... obn any #n → ob1 ... obn any #n zero) Checks for #n floats (F%) zero = C%0 if at least one float was complex, otherwise it is %0.

Chapter 31

Temporary environments

31.1 Built-in ids and lams

Address	Word	Disassembly
15777	NULLID	Null identifier
34D30	NULLLAM	Null lam
211B4	(ID_CST)	ID CST
225A4	(ID_S)	ID S
3FACF	(ID_SKEY)	ID SKEY
3FAE8	(LAM_SKEY)	LAM SKEY
4AB1C	ID_X	ID X
4744F	'IDX	:: ' ID X ;
4AB59	ID_Y	ID Y
41A39	('idUserKeys)	:: ' ID UserKeys ;
41A43	(ID_UserKeys)	ID UserKeys
41A5F	('idUserKeys.)	:: ' UserKeys.CRC ;
41A69	(ID_UserKeys.)	ID UserKeys.CRC
1576C	(ID_EQ)	ID EQ
2C1FD	(ID_SIGMADAT)	ID \85DAT (Σ DAT)
549DB	(lam'dvar)	LAM 'dvar
5127E	('ID_PPAR)	:: ' ID PPAR ;

31.2 Conversion words

Address	Word	Stack
05B15	\$>ID	(\$ → id)
63295	DUP\$>ID	(\$ → \$ id)
05AED	(ID>LAM)	(id → lam)
05B01	(LAM>ID)	(lam → id)

31.3 Temporary environments words

Address	Word	Stack and description
074D0	BIND	(obn ... ob1 { lamn ... lam1 } →) Binds n objects to n differently named lams.
074E4	DOBIND	(obn ... ob1 lamn ... lam1 #n →) Binds n objects to n differently named lams.
634CF	1LAMBIND	(ob →) Binds one object to a null named lam.
634CA	DUP1LAMBIND	(ob → ob) Does DUP then 1LAMBIND.
07497	ABND	(→) Abandons topmost temporary environment.
61CE9	CACHE	(obn ... ob1 #n lam →) Binds all objects under the same name. 1LAM has the count.
61EA7	DUMP	(NULLAM → ob1 ... obn #n) Inverse of CACHE. Always does garbage collection.
61D41	SAVESTACK	(→) Caches stack to SAVELAM.
61F8F	undo	(→) Dumps SAVELAM.
07943	@LAM	(lam → ob TRUE) (lam → FALSE) Tries recalling object from lam. If successful, returns object and TRUE, otherwise returns just FALSE.
07D1B	STOLAM	(ob lam →) Tries storing object in lam. Generates “Undefined Local Name” error if not found.
02FD6	(DoLam)	(lam → ob) (lam → !error!) Tries recalling object from lam, generates “Undefined Local Name” error if not found.

Address	Word	Stack and description
61695	12PUTLAM	
616A5	13PUTLAM	
616B5	14PUTLAM	
616C5	15PUTLAM	
616D5	16PUTLAM	(ob →)
616E5	17PUTLAM	These words store a new contents to specified
616F5	18PUTLAM	lam.
61705	19PUTLAM	
61715	20PUTLAM	
61725	21PUTLAM	
61735	22PUTLAM	
61610	DUP4PUTLAM	(ob → ob) Does DUP then 4PUTLAM.
634B6	1GETABND	(→ 1lamob) Does 1GETLAM then ABND.
62DB3	1ABND_SWAP	(ob → 1lamob ob) Does 1GETABND then SWAP.
62F07	1GET_SWAP	(ob → 1lamob ob) Does 1GETLAM then SWAP.
55288	1GETLAMSWP1+	(# → 1lamob #+1) Does 1GETLAM then SWAP#+.
632E5	2GETEVAL	(→ ?) Does 2GETLAM then EVAL.
617D8	GETLAMPAIR	(#n → #n ob lam FALSE) (#n → #n TRUE) Gets lam contents and name (10 = 1lam, 20 = 2lam, etc.)
34D30	NULLLAM	(→ NULLLAM) Puts NULLLAM in the stack.
34D2B	1NULLLAM{ }	(→ { }) Puts a list with one NULLLAM in the stack.
37DB9	(2NULLLAM{ })	(→ { }) Puts a list with two NULLLAMs in the stack.
37B17	(3NULLLAM{ })	(→ { }) Puts a list with three NULLLAMs in the stack.
52D26	4NULLLAM{ }	(→ { }) Puts a list with four NULLLAMs in the stack.

Address	Word	Stack and description
3306C	(7NULLLAM{ })	(→ { }) Puts a list with seven NULLLAMs in the stack.
10E36	(8NULLLAM{ })	(→ { }) Puts a list with eight NULLLAMs in the stack.

Chapter 32

Error handling

32.1 General words

Address	Word	Stack and description
141E5	ERRBEEP	(→) Beeps.
04CE6	ERROR@	(→ #) Returns current error number.
04D06	ERRORSTO	(# →) Stores new error number.
6383A	ERROROUT	(# →) Stores new error number and does ERRJMP.
04D33	ERRORCLR	(→) Stores zero as new error number.
04ED1	ERRJMP	(→) Invokes error handling sub-system.
04E07	GETEXITMSG	(→ \$) Gets EXITMSG (user defined error message).
04E37	EXITMSGSTO	(\$ →) Stores \$ as EXITMSG.
1502F	DO#EXIT	(# →) Stores new error number, does AtUserStack and then ERRJMP.
15007	DO%EXIT	(% →) Same as above, but takes real number as argument.
1501B	(DOHXSEXIT)	(hxs →) Same as above functions, but input is hxs.

32.2 Error generating words

Number	Address	Word	Error message
001	04FB6	SETMEMERR	Insufficient memory
002	04FC2	(SETDIRRECUR)	Directory Recursion
003	04FCE	(SETUNDEFLAM)	Undefined Local Name
004	05016	SETROMPERR	Undefined XLIB Name
006	04FAA	(SETPOWERLOST)	Power Lost
008	04FDA	(SETINVCARD)	Invalid Card Data
009	04FE6	(SETOBINUSE)	Object In Use
00A	04FF2	SETPORTNOTAV	Port Not Available
00B	04FFE	(SETNOROOM)	No Room in Port
00C	0500A	(SETOBNOTIN)	Object Not In Port
102	10F54	(NULLCHARERR)	Can't Edit Null Char.
103	10F64	(INVFUNCERR)	Invalid User Function
104	10F74	(NOEQERR)	No Current Equation
106	10F86	SYNTAXERR	Invalid Syntax
124	10FE6	(LASTSTKERR)	LAST STACK Disabled
125	10FF6	(LASTCMDERR)	LAST CMD Disabled
126	10FC6	NOHALTERR	HALT Not Allowed
128	11006	(ARGNUMERR)	Wrong Argument Count
129	11016	SETCIRCERR	Circular Reference
12A	11026	(DIRARGERR)	Directory Not Allowed
12B	11036	(EMPTYDIRERR)	Non-Empty Directory
12C	11046	(INVDEFERR)	Invalid Definition
12D	11056	(MISLIBERR)	Missing Library
12E	10F96	(SETINVPPAR)	Invalid PPAR
12F	10FA6	(SETNONERAL)	Non-Real Result
130	10FB6	(SETISOLERR)	Unable to Isolate
13C	11066	(IDCONFERR)	Name Conflict
201	18CC2	SETSTACKERR	Too Few Arguments
202	18CB2	SETTYPEERR	Bad Argument Type
203	18CA2	SETSIZEERR	Bad Argument Value
204	18C92	SETNONEXTERR	Undefined Name
301	29DCC	(POSFLOWERR)	Positive Underflow
302	29DDC	(NEGFLOWERR)	Negative Underflow
303	29DEC	(OVERFLOWERR)	Overflow
304	29DFC	SETIVLERR	Undefined Result
305	29E0C	(INFRESERR)	Infinite Result
B01	10EEA	(INVUNITERR)	Invalid Unit
B02	10EFA	(CONSTUNITERR)	Inconsistent Units
C12	2EC34	SetIOPARerr	Invalid IOPAR
D04	0CBAE	(NOALARMERR)	Nonexistent alarm

Chapter 33

Flags and tests

33.1 Flags

Address	Word	Stack and notes
5380E	COERCEFLAG	(flag → %1/%0) Converts user flag to system flag, drops current stream.
2A7CF	%0<>	Can be used as the opposite function.
03A81	TRUE	(→ TRUE)
0BBED	TrueTrue	(→ TRUE TRUE)
634F7	TrueFalse	(→ TRUE FALSE) Also called TRUEFALSE.
03AC0	FALSE	(→ FALSE)
6350B	FalseTrue	(→ FALSE TRUE) Also called FALSETRUE.
2F934	FalseFalse	(→ FALSE FALSE)
0BC01	failed	(→ FALSE TRUE)
62103	DROPTRUE	(ob → TRUE)
2F542	(2DROPTRUE)	(ob1 ob2 → TRUE)
5F657	(3DROPTRUE)	(ob1 ob2 ob3 → TRUE)
10029	(4DROPTRUE)	(ob1 ob2 ob3 ob4 → TRUE)
6210C	DROPFALSE	(ob → FALSE)
62B0B	2DROPFALSE	(ob1 ob2 → FALSE)
5F5E4	(4DROPFALSE)	(ob1 ob2 ob3 ob4 → FALSE)
5F6B1	(5DROPFALSE)	(ob1 ... ob5 → FALSE)
169A5	NDROPFALSE	(ob1 ... obn #n → FALSE)
4F1D8	SWAPTRUE	(ob1 ob2 → ob2 ob1 TRUE)
21660	SWAPDROPTRUE	(ob1 ob2 → ob2 TRUE)
62EB7	XYZ>ZTRUE	(ob1 ob2 ob3 → ob3 TRUE)
5DE41	(COLATRUE)	(→ TRUE) Puts TRUE in the stack and drops rest of current stream.
5DE55	RDROPFALSE	(→ FALSE) Puts TRUE in the stack and drops rest of current stream.
03AF2	NOT	(flag → flag') Returns FALSE if the input is TRUE, and vice-versa.
03B46	AND	(flag1 flag2 → flag) Returns TRUE if both flags are TRUE.

Address	Word	Stack and notes
03B75	OR	(flag1 flag2 → flag) Returns TRUE if either flag is TRUE.
03ADA	XOR	(flag1 flag2 → flag) Returns TRUE if flags are different.
63B50	ORNOT	(flag1 flag2 → flag) Returns FALSE if either flag is TRUE.
62C55	NOTAND	(flag1 flag2 → flag) Returns TRUE if flag1 is TRUE and flag2 is FALSE.
62C91	ROTAND	(flag1 ob flag2 → ob flag) Returns TRUE if either flag is TRUE.

33.2 General tests

Address	Word	Stack and description
03B2E	EQ	(ob1 ob2 → flag) Returns TRUE if both objects are the same, i.e., they occupy the same physical space in memory. Only the addresses of the objects are tested.
635D8	2DUPEQ	(ob1 ob2 → ob1 ob2 flag) Does 2DUP then EQ.
63605	EQOR	(flag ob1 ob2 → flag') Does EQ then OR.
6303D	EQOVER	(ob3 ob1 ob2 → ob3 flag ob3) Does EQ then OVER.
635F1	EQ:	(ob → flag) EQ with the next object in the current stream.
635EC	DUPEQ:	(ob → ob flag) Does DUP then EQ:.
03B97	EQUAL	(ob1 ob2 → flag) Returns TRUE if the objects are equal (but not necessarily the same), i.e., their prologs and contents are the same.
635C4	EQUALNOT	(ob1 ob2 → flag) Returns TRUE if the objects are different.
63619	EQUALOR	(flag ob1 ob2 → flag') Does EQ then OR.

33.3 Object type tests

General object type tests:

Address	Word	Stack and description
03C64	TYPE	(ob → #prolog) Returns address of prolog of object.
1CB90	XEQTYPE	(ob → ob %type) System version of user word TYPE.

Specific object type tests:

Object	Address	No copy	Address	With copy
Real number	6216E	TYPEPEREAL?	62169	DUPTYPEPEREAL?, DTYPEPEREAL?
Complex number	62183	TYPEPCMP?	6217E	DUPTYPEPCMP?
String	62159	TYPEPCSTR?	62154	DUPTYPEPCSTR?, DTYPEPCSTR?
Array	62198	TYPEPCARRY?	62193	DUPTYPEPCARRY?, DTYPEPCARRY?
Real array	6223B	TYPEPCARRY?		Not available
Complex array	62256	TYPEPCARRY?		Not available
List	62216	TYPEPLIST?	62211	DUPTYPEPLIST?, DTYPEPLIST?
Global identifier	6203A	TYPEPIDNT?	62035	DUPTYPEPIDNT?
Local identifier	6221A	TYPEPLAM?	62115	DUPTYPEPLAM?
Symbolic	621D7	TYPEPSYMB?	621D2	DUPTYPEPSYMB?
Hex string	62144	TYPEPHSTR?	6213F	DUPTYPEPHSTR?
Grob	62201	TYPEPGROB?	621FC	DUPTYPEPGROB?
Tagged	6222B	TYPEPTAGGED?	62226	DUPTYPEPTAG?
Unit	6204F	TYPEPEXT?	6204A	DUPTYPEPEXT?
ROM Pointer	621AD	TYPEPEROMP?	621A8	DUPTYPEPEROMP?
Binary integer	6212F	TYPEPBINT?	621DB	DUPTYPEPBINT?
Directory	621C2	TYPEPRRP?	621D2	DUPTYPEPRRP?
Character	62025	TYPEPCCHAR?	62020	DUPTYPEPCCHAR?
Program	621EC	TYPEPCOL?	621E7	DUPTYPEPCOL?, DTYPEPCOL?

Chapter 34

Runstream control

Note: see Chapter 6 for more detailed explanations of some commands listed below.

Address	Word	Stack and description
06E86	NOP	(→) No operation.
06EEB	'R	(→ ob) Pushes next object in return stack (i.e., the first object in the composite above this one) to the stack (skipping it). If top return stack is empty (contains SEMI), a null secondary is pushed and the pointer is not advanced.
06F66	'REVAL	(→ ?) Does 'R then EVAL.
639DE	'R'R	(→ ob1 ob2) Does 'R twice.
61B89	ticR	(→ ob TRUE) (→ FALSE) Pushes next object in return stack to stack and TRUE, of just FALSE if the top return stack body is empty. In this case, it is dropped.
06F9F	>R	(:: →) Pushes :: to top of return stack (skips prolog, i.e., the composite will be executed automatically).
0701F	R>	(→ ::) Creates and pops a secondary from top return stack body to stack.
07012	R@	(→ ::) Like the above, but the return stack is not popped.

Address	Word	Stack and description
0716B	IDUP	(→) Pushes top body into return stack.
06F86	EVAL	(ob → ?) Evaluates object.
18EBA	COMPEVAL	(comp → ?) EVAL just pushes a list back, this one executes it.
61B45	2@REVAL	(→ ?) EVALS first object in the stream above the previous one.
61B55	3@REVAL	(→ ?) EVALS first object in the stream above the stream above the previous one.
619CB	GOTO	(→) Jumps to next address in stream. Address is a five-nibble address, not a system binary. Can only be used to jump to the middle of programs, cannot jump to a program prolog.
619E0	?GOTO	(flag →) If TRUE, jumps, else skips five nibbles.
619F3	NOT?GOTO	(flag →) If FALSE jumps, else skips five nibbles.
14EA5	RDUP	(→) Duplicates top return stack level.
06FB7	RDROP	(→) Pops the return stack.
6114E	2RDROP	(→) Pops two return stack levels.
61160	3RDROP	(→) Pops three return stack levels.
632F9	DROPRDROP	(ob →) Does DROP then RDROP.
62958	RDROPCOLA	(→) Does RDROP then COLA.

Address	Word	Stack and description
54C4F	(RDROPCOLATRUE)	(→ TRUE) Does RDROP then COLATRUE.
60EBD	RSWAP	(→) Swap in the return stack.
14F2A	(RROLL)	(#n →) Rolls nth return stack level to top of return stack.
63880	RSKIP	(→) Skips first object in the return stack (i.e., the first object in the composite above this one).

34.1 Quoting objects

Address	Word	Stack and description
06E97	'	(→ nob) Pushes next object in the stream to the stack (skipping it.)
63925	DUP '	(ob → ob nob) Does DUP then '.
6394D	DROP '	(ob → nob) Does DROP then '.
63939	SWAP '	(ob1 ob2 → ob2 ob1 nob) Does SWAP then '.
63961	OVER '	(ob1 ob2 → ob1 ob2 ob1 nob) Does OVER then '.
63975	STO '	(ob id/lam → nob) Does STO then '.
63989	TRUE '	(→ TRUE nob) Pushes TRUE and the next object to the stack.
639B6	FALSE '	(→ FALSE nob) Pushes FALSE and the next object to the stack.

Address	Word	Stack and description
6399D	ONEFALSE '	(→ #1 FALSE nob) Pushes ONE, FALSE and the next object to the stack.
639CA	#1+ '	(# → #+1 nob) Does #1+ then '.
632BD	'NOP	(→ NOP) Pushes NOP to the stack.
63155	'ERRJMP	(→ ERRJMP) Pushes ERRJMP to the stack.
3A9B8	'DROPFALSE	(→ DROPFALSE) Pushes DROPFALSE to the stack.
3FDFE	'DoBadKey	(→ DoBadKey) Pushes DoBadKey to the stack.
3FE12	'DoBadKeyT	(→ DoBadKey TRUE) Pushes DoBadKey and TRUE to the stack.
63B5A	'x*	(→ x*) Pushes x* (user word *) to the stack.
63B6E	'xDER	(→ xDER) Pushes xDER (user word ?) to the stack.
5129C	'IDFUNCTION	(→ xFUNCTION) Pushes xFUNCTION (user word FUNCTION) to the stack.
512C4	'IDPOLAR	(→ xPOLAR) Pushes xPOLAR (user word POLAR) to the stack.
512B0	('xCONIC)	(→ xCONIC) Pushes xCONIC (user word CONIC) to the stack.
512D8	'IDPARAMETER	(→ xPARAMETRIC) Pushes xPARAMETRIC (user word PARAMETRIC) to the stack.
512EC	('xTRUTH)	(→ xTRUTH) Pushes xTRUTH (user word TRUTH) to the stack.
51300	('xSCATTER)	(→ xSCATTER) Pushes xSCATTER (user word SCATTER) to the stack.

Address	Word	Stack and description
51314	('xHISTOGRAM)	(\rightarrow xHISTOGRAM) Pushes xHISTOGRAM (user word HISTOGRAM) to the stack.
51328	('xBAR)	(\rightarrow xBAR) Pushes xBAR (user word BAR) to the stack.

34.2 Skipping objects

Address	Word	Description
06FD1	COLA	Evaluates next object and drops remainder of this stream.
63A15	ONECOLA	Does ONE then COLA.
63312	SWAPCOLA	Does SWAP then COLA.
63326	XYZ>ZCOLA	Does UNROT2DROP then COLA.
61A6D	COLA_EVAL	Returns and EVAL. (EVALs first object in previous stream).
6296D	COLACOLA	Drops the remainder of the current stream and does COLA in the above one.
0714D	SKIP	Skips next object in the runstream.
0715C	(2SKIP)	Skips next two objects in the runstream.
283D8	(3SKIP)	Skips next three objects in the runstream.
21C47	(MEMSKIP)	(ob \rightarrow #nextaddress)
62CEE	skipcola	Does SKIP then COLA.
626E5	2skipcola	Does 2SKIP then COLA.
626DC	3skipcola	Does 3SKIP then COLA.
626AE	5skipcola	Skips five objects, then does COLA.
633B2	COLASKIP	Drops remainder of current stream and skips first object in the above stream.
283C4	(COLAskipcola)	Drops remainder of current stream, executes skipcola in the above one.

Chapter 35

Conditionals

How to read the diagrams below: what's on the left side of the arrow represents the stack. *ob1* and *ob2* are different objects. *f1* and *f2* are different flags, *T* represents `TRUE` and *F*, `FALSE`. *#m* and *#n* represent two binary integers, *#m* being smaller than *#n*. *#set* means the a flag is set, *#clr* means it is cleared. On the right of the arrow, the objects which will be executed are represented. The initial stream has the form:

```
:: <test> <ob1> ... <obn> ;
```

In the diagrams, *<rest>* represents everything after the object before.

For the `case` words, most of the times their names are enough to show their actions. Their names have up to three parts: the initial actions, written before `case`, which represent what is done before the test. Thus, `NOTcase` is equivalent to `NOT` then `case`. The final part can be of two types: the first type is written with lowercase letters, like `casedrop`. Those actions are done if the test is `TRUE`, along with the other object(s) specified by the user. So, the code below

```
:: ... casedrop <IfTrueAction> <IfFalseAction> ... ;
```

can be rewritten as

```
:: ... case :: DROP <IfTrueAction> ; <IfFalseAction> ... ;
```

The second type is written with uppercase letters. It means that the object(s) following are the conditions to be executed if the test is `TRUE`. For example, the code below

```
:: ... caseDROP <IfFalseAction> ... ;
```

is equivalent to

```
:: ... case DROP <IfFalseAction> ... ;
```

35.1 Flag tests

Address	Word	Action
61A3B	?SEMI	(T → :: ;) (F → :: <ob1> <rest> ;)
61A2C	NOT?SEMI	(T → :: <ob1> <rest> ;) (F → :: ;)
638E4	?SEMIDROP	(ob T → :: ob ;) (ob F → :: <ob1> <rest> ;)
61B72	NOT?DROP	(ob T → :: ob <ob1> <rest> ;) (ob F → :: <ob1> <rest> ;)
62F1B	?SWAP	(ob1 ob2 T → :: ob2 ob1 <ob1> <rest> ;) (ob1 ob2 F → :: ob1 ob2 <ob1> <rest> ;)
62D9F	?SKIPSWAP	(ob1 ob2 T → :: ob1 ob2 <ob1> <rest> ;) (ob1 ob2 F → :: ob2 ob1 <ob1> <rest> ;)
62F5C	?SWAPDROP	(ob1 ob2 T → :: ob1 <ob1> <rest> ;) (ob1 ob2 F → :: ob2 <ob1> <rest> ;)
62F43	NOT?SWAPDROP	(ob1 ob2 T → :: ob2 <ob1> <rest> ;) (ob1 ob2 F → :: ob1 <ob1> <rest> ;)
070FD	RPIT	(T ob → :: ob <ob1> <rest> ;) (F ob → :: <ob1> <rest> ;)
070C3	RPITE	(T ob1 ob2 → :: ob2 <ob1> <rest> ;) (F ob1 ob2 → :: ob1 <ob1> <rest> ;)
61A86	COLARPITE	(T ob1 ob2 → :: ob1 ;) (F ob1 ob2 → :: ob2 ;)
61AE9	2'RCOLARPITE	Returns to composite above and does ITE there.
619BC	IT	(T → :: <ob1> <rest> ;) (F → :: <ob2> <rest> ;)
0712A	?SKIP or NOT_IT	(T → :: <ob2> <rest> ;) (F → :: <ob1> <rest> ;)
61AD8	ITE	(T → :: <ob1> <ob3> <rest> ;) (F → :: <ob2> <rest> ;)

Address	Word	Action
6381C	COLAITE	(T → :: <ob1> ;) (F → :: <ob2> ;)
61A58	ITE_DROP	(ob T → :: <ob2> <rest> ;) (ob F → :: ob <ob1> <rest> ;)
63E61	ANDITE	(f1 f2 → :: <ob1> <ob3> <rest> ;) (f1 f2 → :: <ob2> <rest> ;)
61993	case	(T → :: <ob1> ;) (F → :: <ob2> <rest> ;)
619AD	NOTcase	(T → :: <ob2> <rest> ;) (F → :: <ob1> ;)
63CEA	ANDcase	(f1 f2 → :: <ob1> ;) (f1 f2 → :: <ob2> <rest> ;)
63DDF	ANDNOTcase	(f1 f2 → :: <ob1> ;) (f1 f2 → :: <ob2> <rest> ;)
629BC	ORcase	(f1 f2 → :: <ob1> ;) (f1 f2 → :: <ob2> <rest> ;)
618F7	casedrop	(ob T → :: <ob1> ;) (ob F → :: ob <ob2> <rest> ;)
618E8	NOTcasedrop	(ob T → :: ob <ob2> <rest> ;) (ob F → :: <ob1> ;)
6191F	case2drop	(ob1 ob2 T → :: <ob1> ;) (ob1 ob2 F → :: ob1 ob2 <ob2> <rest> ;)
61910	NOTcase2drop	(ob1 ob2 T → :: ob1 ob2 <ob2> <rest> ;) (ob1 ob2 F → :: <ob1> ;)
6194B	caseDROP	(ob T → :: ;) (ob F → :: ob <ob1> <rest> ;)
61960	NOTcaseDROP	(ob T → :: ob <ob1> <rest> ;) (ob F → :: ;)
638B2	casedrptru	(ob T → TRUE) (ob F → :: ob <ob1> <rest> ;)

Note: should be caseDRPTRU.

Address	Word	Action
6356A	casedrpfls	(ob T → FALSE) (ob F → :: ob <ob1> <rest> ;) Note: should be caseDRPFLLS.
63AEC	NOTcsdrpfls	(ob T → :: ob <ob1> <rest> ;) (ob F → FALSE) Note: should be NOTcaseDRPFLLS.
61970	case2DROP	(ob1 ob2 T → :: ;) (ob1 ob2 F → :: ob1 ob2 <ob2> <rest> ;)
61984	NOTcase2DROP	(ob1 ob2 T → :: ob1 ob2 <ob1> <rest> ;) (ob1 ob2 F → :: ;)
63583	case2drpfls	(ob1 ob2 T → FALSE) (ob1 ob2 F → :: ob1 ob2 <ob1> <rest> ;) Note: should be case2DRPFLLS.
634E3	caseTRUE	(T → TRUE) (F → :: <ob1> <rest> ;)
638CB	NOTcaseTRUE	(T → :: <ob1> <rest> ;) (F → TRUE)
6359C	caseFALSE	(T → FALSE) (F → :: <ob1> <rest> ;)
5FB49	NOTcaseFALSE	(T → :: <ob1> <rest> ;) (F → FALSE)

35.2 Binary number tests

Address	Word	Action
6336C	#=?SKIP	(#m #m → :: <ob2> <rest> ;) (#m #n → :: <ob1> <rest> ;)
63399	#>?SKIP	(#m #n → :: <ob1> <rest> ;) (#n #m → :: <ob2> <rest> ;)
62C2D	#=ITE	(#m #m → :: <ob1> <ob3> <rest> ;) (#m #n → :: <ob2> <rest> ;)

Address	Word	Action
63E9D	#<ITE	(#m #n → :: <ob1> <ob3> <rest> ;) (#n #m → :: <ob2> <rest> ;)
63EB1	#>ITE	(#m #n → :: <ob2> <rest> ;) (#n #m → :: <ob1> <ob3> <rest> ;)
6186C	#=case	(#m #m → :: <ob1> ;) (#m #n → :: <ob2> <rest> ;)
6187C	OVER#=case	(#m #m → :: #m <ob1> ;) (#m #n → :: #m <ob2> <rest> ;)
618D3	#=casedrop	(#m #m → :: <ob1> ;) (#m #n → :: #m <ob2> <rest> ;) Note: Should be OVER#=casedrop.
63547	#=casedrpfls	(#m #m → FALSE) (#m #n → :: #m <ob1> <rest> ;) Should be OVER#=caseDRPFLS.
63D3A	#<>case	(#m #m → :: <ob2> <rest> ;) (#m #n → :: <ob1> ;)
63D12	#<case	(#m #n → :: <ob1> ;) (#n #m → :: <ob2> <rest> ;)
63D67	#>case	(#m #n → :: <ob2> <rest> ;) (#n #m → :: <ob1> ;)
61A18	#0=?SEMI	(#0 → :: ;) (# → :: <ob1> <rest> ;)
6333A	#0=?SKIP	(#0 → :: <ob2> <rest> ;) (# → :: <ob1> <rest> ;)
63E89	#0=ITE	(#0 → :: <ob1> <ob3> <rest> ;) (# → :: <ob2> <rest>)
63E48	DUP#0=IT	(#0 → :: #0 <ob1> <rest> ;) (# → :: # <ob2> <rest> ;)
63EC5	DUP#0=ITE	(#0 → :: #0 <ob1> <ob3> <rest> ;) (# → :: # <ob2> <rest>)
61896	#0=case	(#0 → :: <ob1> ;) (# → :: <ob2> <rest> ;)

Address	Word	Action
61891	DUP#0=case	(#0 → :: #0 <ob1> ;) (# → :: # <ob2> <rest> ;)
618A8	DUP#0=csedrp	(#0 → :: <ob1> ;) (# → :: # <ob2> <rest> ;)
63CBD	DUP#0=csDROP	(#0 → :: ;) (# → :: # <ob1> <rest> ;)
63D26	#1=case	(#1 → :: <ob1> ;) (# → :: <ob2> <rest> ;)
63353	#1=?SKIP	(#1 → :: <ob2> <rest> ;) (# → :: <ob1> <rest> ;)
63D4E	#>2case	(#0/#1/#2 → :: <ob2> <rest> ;) (# → :: <ob1> ;)

35.3 Real and complex numbers tests

The function `num0=case` means either `%0` or `C%0`. And so does the other ones. All the words except `j%0=case` make a copy of the object first.

Address	Word	Action
5F127	%0=case	(%0 → :: %0 <ob1> ;) (ob → :: ob <ob2> <rest> ;)
63D7B	j%0=case	(%0 → :: <ob1> ;) (ob → :: <ob2> <rest> ;)
5F13B	C%0=case	(C%0 → :: C%0 <ob1> ;) (ob → :: ob <ob2> <rest> ;)
5F0FA	num0=case	(0 → :: 0 <ob1> ;) (ob → :: ob <ob2> <rest> ;)
5F181	%1=case	(%1 → :: %1 <ob1> ;) (ob → :: ob <ob2> <rest> ;)
5F19F	C%1=case	(C%1 → :: C%1 <ob1> ;) (ob → :: ob <ob2> <rest> ;)
5F154	num1=case	(1 → :: 1 <ob1> ;) (ob → :: ob <ob2> <rest> ;)

Address	Word	Action
5F1EA	%2=case	(%2 → :: %2 <obl> ;) (ob → :: ob <ob2> <rest> ;)
5F208	C%2=case	(C%2 → :: C%2 <obl> ;) (ob → :: ob <ob2> <rest> ;)
5F1BD	num2=case	(2 → :: 2 <obl> ;) (ob → :: ob <ob2> <rest> ;)
5F267	%-1=case	(%-1 → :: %-1 <obl> ;) (ob → :: ob <ob2> <rest> ;)
5F285	C%-1=case	(C%-1 → :: C%-1 <obl> ;) (ob → :: ob <ob2> <rest> ;)
5F23A	num-1=case	(-1 → :: -1 <obl> ;) (ob → :: ob <ob2> <rest> ;)
5EEDB	(REALNEGcase)	(%<0 → :: % <obl> ;) (ob → :: ob <ob2> <rest> ;)

35.4 Meta object tests

Address	Word	Action
5FBE6	(pick1#0=case)	(#0 M → COLA) (ob M → SKIP)
5EFD9	MEQ1stcase	Meta&ob1 ob2 → ob1=ob2 ? case
5EF15	AEQ1stcase	Meta&ob → ob=nob ? case
5EFF9	MEQopscase	Meta1&ob1 Meta2&ob2 ob3 →
5F048	AEQopscase	Meta1&ob1 Meta2&ob2 →
5F061	Mid1stcase	Meta&ob → ob is id or lam ? case
549EC	(MetaConcase)	(Meta → Meta) COLA if meta contains no ids, lams, syms or romptrs. Else SKIP.
5EF2E	(M1st+case)	Meta&+ ? case
5EF41	(M1st-case)	Meta&- ? case
5EF54	(M1st*case)	Meta&* ? case
5EF67	(M1st/case)	Meta&/ ? case
5EFA0	(M1st^case)	Meta&^ ? case
58ADE	(M-1potcase)	Meta&-1&^ ? case
5EFB3	(M1stSQcase)	Meta&SQ ? case
5EF7A	(M1stNEGcase)	Meta&NEG ? case
5EF8D	(M1stINVcase)	Meta&INV ? case
5EFC6	(M1stFNCcase)	Meta&FCNAPPLY ? case
5EE10	M-1stcasechs	Meta&NEG → Meta COLA ; Meta → Meta SKIP Meta&(%<0) → Meta&ABS(%) COLA

35.5 General object tests

Address	Word	Action
63E2F	EQIT	(ob1 ob1 → :: <ob1> <rest> ;) (ob1 ob1 → :: <ob2> <rest> ;)
63E75	EQITE	(ob1 ob1 → :: <ob1> <ob3> <rest> ;) (ob1 ob2 → :: <ob2> <rest> ;)
63CD6	jEQcase	(ob1 ob1 → :: <ob1> ;) (ob1 ob2 → :: <ob2> <rest> ;)
61933	EQcase	(ob1 ob1 → :: ob1 <ob1> ;) (ob1 ob2 → :: ob1 <ob2> <rest> ;) Should be called OVEREQcase.
618BA	EQcasedrop	(ob1 ob2 → :: <ob1> ;) (ob1 ob2 → :: ob1 <ob2> <rest> ;)
63CFE	EQUALcase	(ob1 ob1 → :: <ob1> ;) (ob1 ob2 → :: <ob2> <rest> ;)
63DF3	EQUALNOTcase	(ob1 ob1 → :: <ob2> <rest> ;) (ob1 ob2 → :: <ob1> ;)
63CA4	EQUALcasedrp	(ob ob1 ob2 → :: <ob1> ;) (ob ob1 ob2 → :: ob <ob2> <rest> ;)
517FE	EQUALcasedro	(ob1 ob2 → :: <ob1> ;) (ob1 ob2 → :: ob1 <ob2> <rest> ;) Should be OVEREQUALcasedrp.
5E984	nonopcase	(seco → :: seco <ob2> <rest> ;) (ob → :: ob <ob1> ;)
5F0AA	idntcase	(id → :: id <ob1> ;) (ob → :: ob <ob2> <rest> ;)
63E07	dIDNTNcase	(id → :: id <ob2> <rest> ;) (ob → :: ob <ob1> ;)
5F0CD	idntlamcase	(id/lam → :: id <ob1> ;) (ob → :: ob <ob2> <rest> ;)
63D8F	REALcase	(% → :: <ob1> ;) (ob → :: <ob2> <rest> ;)

Address	Word	Action
63E1B	dREALNcase	(% → :: % <ob2> <rest> ;) (ob → :: ob <obl> ;)
63DA3	dARRAYcase	([] → :: [] <obl> ;) (ob → :: ob <ob2> <rest> ;)
63DB7	dLISTcase	({ } → :: { } <obl> ;) (ob → :: ob <ob2> <rest> ;)
27244	NOTLISTcase	({ } → :: { } <ob2> <rest> ;) (ob → :: ob <obl> ;)
18E45	(DNOTSymb?SEMI)	(symb → :: symb <obl> <rest> ;) (ob → :: ob ;)
27254	NOTSECOcase	(seco → :: seco <ob2> <rest> ;) (ob → :: ob <obl> ;)
27264	NOTROMPcase	(romp → :: romp <ob2> <rest> ;) (ob → :: ob <obl> ;)
27224	(DNOTBAKcase)	(bak → :: bak <ob2> <rest> ;) (ob → :: ob <obl> ;)
27234	(DNOTLIBcase)	(lib → :: lib <ob2> <rest> ;) (ob → :: ob <obl> ;)
5EDFC	numblstcase	If %, C%, [] or [L] then COLA, else SKIP.

35.6 Miscellaneous

Address	Word	Action
63ED9	UserITE	(#set → :: <obl> <ob3> <rest> ;) (#clr → :: <ob2> <rest> ;)
63EED	SysITE	(#set → :: <obl> <ob3> <rest> ;) (#clr → :: <ob2> <rest> ;)
63BEB	caseDoBadKey	(T → :: DoBadKey ;) (F → :: <obl> <rest> ;) Also called caseDEADKEY.
63BD2	caseDrpBadKy	(ob T → :: DoBadKey ;) (ob F → :: ob <obl> <rest> ;)
63169	caseERRJMP	(T → :: ERRJMP ;) (F → :: <obl> <rest> ;)

Address	Word	Action
63B05	caseSIZEERR	(T → :: SIZEERR ;) (F → :: <obl> <rest> ;)
63B19	NcaseSIZEERR	(T → :: <obl> <rest> ;) (F → :: SIZEERR ;)
63B46	NcaseTYPEERR	(T → :: <obl> <rest> ;) (F → :: TYPEERR ;)

Chapter 36

Loops

If you are using the HP Tools, you must take special care when you use the words marked with an asterisk below. The compiler does not recognize them, so if you simply use them, you'll get errors like "DO without LOOP". When you use one of them, insert `DO` or `LOOP` between `()`'s just after the word. This way, the compiler will not generate any errors. For example:

```
::
...
ZERO_DO
...          *** WRONG ***
DROPLoop
...
;

::
...
ZERO_DO (DO)
...          *** RIGHT ***
DROPLoop (LOOP)
...
;
```

If you use JAZZ, you do not need to worry about that. Note that there is no space after the parenthesis.

36.1 Indefinite loops

Address	Word	Stack and description
0716B	IDUP	(→) Pushes top body into return stack.
071A2	BEGIN	(→) Pushes top body into return stack.
071AB	AGAIN	(→) Copies return stack to top.
071E5	REPEAT	(→) Copies return stack to top.

Address	Word	Stack and description
071C8	UNTIL	(flag →) If FALSE then AGAIN, otherwise RDROP.
633C6	* NOT_UNTIL	(flag →) NOT then UNTIL.
62B6F	#0=UNTIL	(# → #) Actually, should be DUP#0=UNTIL.
071EE	WHILE	(flag →) If TRUE does nothing, otherwise RDROP then 2SKIP.
633DF	NOT_WHILE	(flag →) NOT then WHILE.
633F8	DUP#0<>WHILE	(# →) Try to guess what it does.

36.2 Definite loops

Address	Word	Stack and notes
073F7	DO	(#stop #start →)
073C3	* ZERO_DO	(#stop →)
6347F	* DUP#0_DO	(#stop → #stop)
073CE	* ONE_DO	(#stop →)
073DB	* #1+_ONE_DO	(#stop →)
63498	toLEN_DO	({} → {}) From ONE to #elements+1.
37BCB	(ONE_DO_ARRAY)	([] → []) From ONE to #elements+1.
07334	LOOP	(→)
073A5	+LOOP	(# →) Increments index by specified number.
63466	* DROPLOOP	(ob →)
6344D	* SWAPLOOP	(ob1 ob2 → ob2 ob1)
54CB3	(SWAPDROPLOOP)	(ob1 ob2 → ob2)
07321	(STOPLOOP)	(→) Destroys topmost loop environment.
07221	INDEX@	(→ #) Recalls topmost loop counter value.
63411	DUPINDEX@	(ob → ob #)
63425	SWAPINDEX@	(ob1 ob2 → ob2 ob1 #)
63439	OVERINDEX@	(ob1 ob2 → ob1 ob2 ob1 #)

Address	Word	Stack and notes
63790	INDEX@-	(# → #')
07270	INDEXSTO	(# →)
		Stores new topmost loop counter value.
07249	ISTOP@	(→ #)
		Recalls topmost loop stop value.
07295	ISTOPSTO	(# →)
		Stores new topmost loop stop value.
5182F	ISTOP-INDEX	(→ #)
07258	JINDEX@	(→ #)
		Recalls second topmost loop counter value.
072AD	JINDEXSTO	(# →)
		Stores new second topmost loop counter value.
07264	JSTOP@	(→ #)
		Recalls second topmost loop stop value.
072C2	JSTOPSTO	(# →)
		Stores new second topmost loop stop value.
6400F	ExitAtLoop	(→)
		Does not exit loop immediately. Just stores zero as the stop value, so all objects until the next LOOP will be evaluated.
		Also called ZEROISTOPSTO.
3F78C	(DUPExitAtLOOP)	(ob → ob ob)
3F7EB	(ExitAtLOOPDUP)	(ob → ob ob)
4334F	(DRPExitAtLOOP)	(ob →)

Chapter 37

Memory operations

37.1 Temporary memory

Address	Word	Stack and description
06657	TOTEMPOB	(ob → ob') Copies object to TEMPOB and returns pointer to the new copy.
62C69	TOTEMPSWAP	(ob1 ob2 → ob2' ob1') Does TOTEMPOB then SWAP.
37B44	CKREF	(ob → ob') If object is in TEMPOB, is not embedded in a composite and not referenced, does nothing. Else copies it to TEMPOB and returns the copy.
63F7E	SWAPCKREF	(ob1 ob2 → ob2' ob1') Does SWAP then CKREF.
06B4E	INTEMNOTREF?	(ob → ob flag) If the object is in TEMPOB area, is not embedded in a composite and is not referenced, returns the object and TRUE, otherwise returns the object and FALSE.
06B3E	(INTEMP?)	(ob → ob flag) Tests if object is in TEMPOB area and not in a composite.
065D9	(PTRREF?)	(ob → ob flag) Tests if object is referenced.
065E5	(REFERENCED?)	(ob → ob flag) Tests if object is referenced or in composite.
06BC2	(NOTREF?)	(ob → ob flag) Tests if object is not referenced or in composite. (:: REFERENCED? NOT ;)

Address	Word	Stack and description
06DDE	(>TOPTEMPOB)	(ob → ob') Moves object to top ob TEMPOB area. Does not garbage collection.
064BD	(UNREFOB)	(ob → ob ob') Makes a standalone copy by moving references to a new copy.
064D6	(UNREFTEMP)	(ob1 ob2 → ob1 ob') Moves references from ob2 to ob1 (ob1 in TEMPOB area).
064E2	(UNREFINTEMP)	(ob1 ob2 → ob1 ob') Moves references from ob2 to ob1 (ob1 in TEMPOB area). References to body of ob2 are moved too.

37.2 Recalling, storing and purging

Address	Word	Stack and description
0797B	@	(id/lam → ob TRUE) (id/lam → FALSE) Basic recalling function.
62C05	DUP@	(id/lam → id/lam ob TRUE) (id/lam → id/lam FALSE) Does DUP then @.
62A34	SAFE@	(id/lam → ob TRUE) (id/lam → FALSE) For lams does @. For ids does ?ROMPTR> to the ob found.
5E5EE	(SAFE@NOT)	(id → ob FALSE) (id → TRUE) Does SAFE@ then NOT.
62A2F	DUPSAFE@	(id/lam → id/lam ob TRUE) (id/lam → id/lam FALSE) Does DUP then SAFE@.
1853B	SAFE@_HERE	(id/lam → ob TRUE) (id/lam → FALSE) Same as SAFE@, but works only in the current directory.

Address	Word	Stack and description
20B81	XEQRCL	(id → ob) Same as SAFE@, but errors if variable is not found. Also works for lams, but you get the wrong error.
20B9A	LISTRCL	({ path id } → ob) Recalls from specified path.
5E602	(@LAMNOT)	(lam → ob FALSE) (lam → lam TRUE) Recalls lam contents if possible.
5E616	(ROMPTR@NOT)	(ROMPTR → ob FALSE) (ROMPTR → ROMPTR TRUE) Recalls contents of ROMPTR if possible.
5E5B7	(@OBNOT)	(ob → ob' FALSE) (ob → ob TRUE) Recalls contents if input is id, lam or ROMPTR. For other types returns TRUE.
072D7	STO	(ob id/lam →) For ids: <ul style="list-style-type: none"> Assumes ob is not pco; If replacing some object that object is copied to TEMPOB and pointers are updated. For lams: <ul style="list-style-type: none"> Errors if lam is unbound.
18513	XEQSTOID	(ob id/lam →) Same as SAFESTO, but will only store in the current directory and will not overwrite a directory. Also called ?STO_HERE.
085D3	REPLACE	(newob oldob → newob) Replaces oldob (in memory) with newob. See Chapter 9 for detailed description.
08C27	PURGE	(id →) Purges variable. Does no type check first.
1854F	?PURGE_HERE	(id →) Like PURGE, but only works in current directory.
08696	CREATE	(ob id →) Creates a variable in the current directory. Errors if id is or contains current directory. Assumes id is not a pco.

Address	Word	Stack and description
185C7	DoHere:	(→) Next object in the runstream is evaluated for the current directory only.

37.3 Directories

Address	Word	Stack and description
077E4	(CRDIR#)	(#libnum → rrp) Creates an empty directory.
08DF2	(!CREATEDIR)	(id →) Creates an empty directory. Does not check if the name is already used. (:: # 7FF CRDIR# SWAP CREATE ;)
184E1	CREATEDIR	(id →) Creates an empty directory. Calls ?PURGE_HERE first to delete the original.
08326	LASTRAM-WORD	(rrp → ob TRUE) (rrp → FALSE) Recalls first object in directory.
18621	LastNonNull	(rrp → ob TRUE) (rrp → FALSE) Recalls first object in directory (not null named).
08376	PREVRAM-WORD	(ob → ob' TRUE) (ob → FALSE) Recalls next object in directory.
1863A	PrevNonNull	(ob → ob' TRUE) (ob → FALSE) Recalls next object in directory (not null named).
18653	(CkNonNull)	(ob → ob TRUE) (ob → FALSE) Checks that the variable (ob) has a name.
082E3	RAM-WORDNAME	(ob → id) Recalls name of object in current directory.

Address	Word	Stack and description
18595	XEQPGDIR	(id →) Purges a directory. Checks references, etc. first.
20FF2	XEQORDER	({ id1 id2 ... } →) Orders directory.
18779	DOVARS	(→ { id1 id2 ... }) Returns list of variables from current directory.
1867F	(DODIRPRG)	(ob :: → { }) Executes seco (can be single object) on all directory variables. At execution: ob :: id_contents { } id To be returned: ob :: id_contents { } ob flag If flag is TRUE, ob is added with >TCOMP to list, else it is dropped.
1848C	PATHDIR	(→ { HOME dir1 dir2 ... }) Returns current path.
1A16F	UPDIR	(→) Goes to parent directory.
08309	(GETUPDIR)	(rrp → rrp' TRUE) (rrp → FALSE) Gets parent directory.
08DD4	(HOMEDIR?)	(rrp → flag) Is the directory the HOME directory?
05D5A	CONTEXT@	(→ rrp) Recalls current directory.
08D08	CONTEXT!	(rrp →) Sets new current directory.
08D82	(LCONTEXT@)	(→ rrp) Recalls last directory.
08D4A	(LCONTEXT!)	(rrp →) Stores new last directory.
08D92	HOMEDIR	(→) Sets HOME as current directory. Also called SYSCONTEXT.

Address	Word	Stack and description
08DC4	(SYSLCONTEXT)	(→) Sets HOME as last directory.
640A0	SaveVarRes	(→) Binds current and last directories to two null-named lams.
640FA	RestVarRest	(→) First sets HOME as both the current and last directories (in case an error happens). Then, restores the current and last directories from 1LAM and 2LAM.

37.4 The hidden directory

Address	Word	Stack and description
640BE	SetHiddenRes	(→) Sets the hidden directory as the current and last directories.
64037	WithHidden	(→ ?) Executes next command in hidden directory.
64023	RclHiddenVar	(id → ob TRUE) (→ FALSE) Recalls variable in hidden directory. (:: WithHidden @ ;)
64078	StoHiddenVar	(ob id →) Stores variable in hidden directory. (:: WithHidden STO ;)
6408C	PuHiddenVar	(id →) Purges variable in hidden directory. (:: WithHidden PURGE ;)

Chapter 38

Time and alarms

The internal alarms list has the format: { { hxs action } { ... } ... }. The length of the hxs is 24 nibbles. The least significant 13 nibbles represent the tick value for the time and date, the next 10 nibbles the repeat interval (if any), and the most significant nibble represents the status of the alarm (pending, acknowledged, etc.).

Address	Word	Stack and description
40EE7	SLOW	(→) 15 millisecond delay.
40F02	VERYSLOW	(→) 300 millisecond delay.
40F12	VERYVERYSLOW	(→) 3 second delay.
1A7ED	(wait)	(hxs →) Wait specified number of ticks (there are 8192 ticks in a second).
1A7C9	dowait	(%secs →) Waits specified number of seconds.
1A7B5	(dowait/quit?)	(%secs →) Waits specified number of seconds, exits program if CANCEL is pressed.
2A673	%>HMS	(% → %hms) Converts from decimal to H.MMSS format.
2AF27	%%H>HMS	(%% → %%hms) Same function but for long reals.
2A68C	%HMS>	(%hms → %) Converts from H.MMSS format to decimal.
2A6A0	%HMS+	(%hms1 %hms1 → %hms) Adds time in hms format.
2A6C8	%HMS-	(%hms1 %hms2 → %hms) Subtracts time in hms format.

Address	Word	Stack and description
0CBFA	TOD	(→ %time) Returns current time.
0CD53	(>TIME)	(%time →) Sets time.
0CD3F	(CLKADJ)	(%time →) Also sets time.
0CC0E	DATE	(→ %date) Returns current date.
0CD2B	(>DATE)	(%date →) Sets date.
0CC5B	DATE+DAYS	(%date %days → %date') Adds specified number of days to date.
0CC39	DDADYS	(%date1 %date2 → %days) Returns number of days between two dates.
0E235	ALARMS@	(→ { }) Returns internal alarms list.
0E6ED	STOALM	(%date %time action %repeat → %) Stores an alarm. %repeat is the number of ticks between every repetition. Since there are 8192 ticks in a second, 60 seconds in a minute, and 60 minutes in an hour, to make an alarm that repeats every hour, %repetition would be 8192*60*0 = 29491200. Returns real number representing the position of the alarm in the list.
0E54D	(STOALARMLS)	({ } → %) Stores an alarm. List contents: { %date %time action %repeats } You may omit %repeats and action. In this case, the alarm has no repetition and no message is displayed. Returns real number representing the position of the alarm in the list.
0E510	(STOALARM%)	(%time → %) Store an alarm at specified time today, with no message and no repetition. Returns real number representing the position of the alarm in the list.

Address	Word	Stack and description
0EF45	(>ALRMLS)	(\$ %date %time %rpt → { }) Generates list (of the internal type) representing the alarm.
0E1D8	(ALRMLS>)	({ } → { }') Converts list of internal format to list in the format of STOALARMLS.
0E402	(RCLALM)	(#n → { } TRUE) (#n → FALSE) Recalls nth alarm. List is in the internal format.
0E3DF	(RCLALARM%)	(%n → { }) Recalls nth alarm. List is in the format of STOALARMLS.
0EAD7	(FINDALARM%)	(%date → %) Returns position in the internal alarm list of the first alarm of that day (or in any following day).
0EB31	(FINDALARMLS)	({ } → %) Takes a list of the format: { %date %time } Returns real represent the position of the specified alarm in the alarm list, or 0 if not found.
0E724	(DELALARM)	(%n →) Deletes nth alarm.
422A1	ALARM?	(→ flag) Returns TRUE if an alarm is due.
0DDC1	(ACKALM)	(→ flag) Tries acknowledging first alarm due. Returns TRUE if no due alarm was found, or FALSE if a due alarm has been found and acknowledged.
0DDA8	(ACKALLALMS)	(→) Acknowledges all due alarms.
0EB81	CLKTICKS	(→ hxs) Returns tick count. Also called SysTime.

Address	Word	Stack and description
0D304	TIMESTR	(%date %time → \$) Returns string representation of time, using current format. Example: "WED 06/24/98 10:00:45A"
0CFD9	Date>d\$	(%date → \$) Returns string representation of date, using current format.
0D2F0	(Date>wd\$)	(%date → \$weekday) Returns weekday: "SUN", "MON", etc.
0CF5B	(Ticks>wd\$)	(hxs → \$weekday) Same function but using clock ticks.
0D06A	TOD>t\$	(%time → \$) Returns string represent the time, using current format.
0EE50	Date>hxs13	(%date → hxs) Converts date to ticks.
0D156	(Ticks>Date)	(hxs → %date) Returns date from hxs of internal alarm list format.
0EE83	(TOD>Ticks)	(%time → hxs) Converts time to ticks.
0D143	(Ticks>TOD)	(hxs → %time) Returns time from hxs of internal alarm list format.
0D169	(Ticks>Rpt)	(hxs → %rpt) Converts hxs in internal alarm list format to repetition interval.
0EE26	(Date+Time)	(hxs_d hxs_t → hxs) Takes two hxs representing the date and the time, and joins them into only one hxs.

Chapter 39

System functions

39.1 User and system flags

Address	Word	Stack and notes
53731	SetSysFlag	(# →)
53761	ClrSysFlag	(# →)
53784	TestSysFlag	(# → flag)
		Returns TRUE if flag is set.
1C4EC	(TestSysClr)	(# → flag)
		Clears flag after testing.
3EDA2	(TogSysFlag)	(# →)
		Toggles system flag.
53725	SetUserFlag	(# →)
53755	ClrUserFlag	(# →)
53778	TestUserFlag	(# → flag)
		Returns TRUE if flag is set.
1C4CE	(TestUserClr)	(# → flag)
		Clears flag after testing.
1C637	RCLSYSF	(→ hxs)
		Recalls system flags.
1C731	(STOSYSF)	(hxs →)
		Stores system flags.
1C6E3	DOSTOSYSF	(hxs →)
		Stores system flags, checking for changes in LASTARG flag.
1C64E	RCLUSERF	(→ hxs)
		Recalls user flags.
1C6F7	(STOUSERF)	(hxs →)
		Stores user flags.
1C6CF	(STOALLF)	(hxs hxs →)
		Stores user and system flags. First is user flags, second is system flags.
1C6A2	DOSTOALLF	({ } →)
		Stores user and system flags. Expects a list of two hxs, first is user flags, second is system flags.
53CAA	dostws	(# →)
53C96	(XEQSTWS)	(% →)
54039	WORDSIZE	(→ #)
53CF0	(XEQRCWS)	(→ %)

Address	Word	Stack and notes
53C37	DOHEX	(→)
54C5B	DODEC	(→)
53C43	DOBIN	(→)
53C4F	DOOCT	(→)
54050	BASE	(→ #) Returns #10h, #10d, #10b or #10o. In decimal terms, 16 for hexadecimal base, 10 for decimal base, 8 for octal base or 2 for binary base.
5407A	(BASECHAR)	(→ char) Returns "h", "d", "b" or "o".
16707	DOSTD	(→)
166E3	DOFIX	(# →)
166EF	DOSCI	(# →)
166FB	DOENG	(# →)
53B61	(NumbMode)	(→ #) Returns 0 for STD mode, 1 for FIX mode, 2 for SCI mode or 3 for ENG mode.
2A5F0	SETRAD	(→)
53BDD	RAD?	(→ flag)
2A5D2	SETDEG	(→)
53BC9	(DEG?)	(→ flag)
2A604	SETGRAD	(→)
167BF	DPRADIX?	(→ flag) Returns TRUE if current radix is ".".
53C23	(PRSOL?)	(→ flag) Returns TRUE if general solutions flag (1) is set.
53C0A	(NOTCONST?)	(→ flag) Returns TRUE if symbolic constants flag (2) is cleared.
53B9C	(SETNUM)	(→) Sets numeric results flag (3).
53B88	(CLRNUM)	(→) Clears numeric results flag (3).
53BB0	(NOTNUM?)	(→ flag) Returns TRUE if numeric results flag (3) is cleared.

39.2 General functions

Address	Word	Stack and description
1415A	DOBEEP	(%freq %dur →) Beeps. Analog to user function BEEP.
141B2	setbeep	(#MHz #ms →) Also beeps.

Address	Word	Stack and description
05944	OCRC	(ob → # hxs) Returns size in nibbles and checksum as hxs.
1A1FC	OCRC%	(ob → hxs %) Returns checksum and size in bytes.
1A265	VARSIZE	(id → hxs %) Returns checksum and size in bytes of specified variable.
1A2DA	INHARDROM?	(ob → ob flag) Is object address < #70000h?
19350	(NOTINHARDROM?)	(ob → ob flag) Is object address ≥ #70000h?
05AB3	CHANGETYPE	(ob #prolog → ob') Changes prolog of object, does TOTEMPOB.
05ACC	(!CHANGETYPE)	(ob #prolog → ob') Changes prolog of object.
6595A	getnibs	(hxs hxs → hxs') Peek. First hxs is data, second is address. The data is overwritten for its length (maximum 16) with nibbles starting from specified address.
6594E	putnibs	(hxs hxs →) Poke. First hxs is data, second is address. Works like above function.

Chapter 40

Keyboard control

40.1 User keys

If no keys are assigned, the internal key assignments list is an empty list. If there is one or more assignments, the list contains 49 sub-lists, each one representing one key. Each sub-list is either empty, if that key has no assignments; or contains six elements: each representing the assignment of one plane (in the order: unshifted, left-shifted, right-shifted, alpha, alpha and left-shift and finally alpha and right-shift). For planes with no assignment, an empty list is entered.

Address	Word	Stack and description
41F3F	GetUserKeys	(→ {}) Returns user keys list (internal format).
41C02	(XEQRclKeys)	(→ {}) Recalls all key assignments (in user format) plus status of non defined keys.
41B28	(XEQAsnKey)	(ob %rc.p →) Assigns an object to a key, specified in user format.
41E78	(AsnKey)	(ob #kc #p →) Assigns an object to a key, specified in system format.
41F2C	(UserKeys!)	({} →) Stores user keys (list is in internal format).
41E32	(StoUserKeys)	({} →) Like the above, but also recalculates CRC.
41AA1	(Ck&AsnUKeys)	({} →) Stores user keys (list in user format), recalculates CRC.
41B8C	(DelKey)	(#kc #plane →) Deletes that key assignment, recalculates CRC.

Address	Word	Stack and description
41B3C	(XEQDelKeys)	({ } →) Deletes specified keys (in user format).
41B69	(Ck&ClrUKey)	(0 →) (%rc.p →) System version of user word <code>DELKEYS</code> : if 0, deletes all keys, otherwise deletes specified key.
41F52	(PgUserKeys)	(→) Deletes all user keys.
41F13	(ClrUserKeys)	(→) Deletes all user keys and recalculates CRC.
3FF75	(NonUsrKeyOK?)	(→ flag) Returns <code>TRUE</code> if the keys not defined do their normal actions.
3FF86	(SetNUsrKeyOK)	(→) Keys not defined do their normal actions.
3FF97	(ClrNUsrKeyOK)	(→) Keys not defined just beep when pressed.

40.2 Waiting for keys

Address	Word	Stack and description
41CA2	Ck&DecKeyLoc	(%rc.p → #kc #p) Converts from user key representation format to system.
41D92	CodePl>%rc.p	(#kc #p → %rc.p) Inverse transformation.
00D71	FLUSHKEYS	(→) Flushes the key buffer. Also called <code>FLUSH</code> .
04708	CHECKKEY	(→ #kc TRUE) (→ FALSE) Returns next key in the key buffer, but does not pop it.

Address	Word	Stack and description
04714	GETTOUCH	(→ #kc TRUE) (→ FALSE) Pops next key from key buffer.
047C7	REPKEY?	(#kc → flag) Returns TRUE if the key is being pressed.
42402	KEYINBUFFER?	(→ flag) Returns TRUE if there is at least a key in the key buffer.
41F65	WaitForKey	(→ #kc #p) Returns next full key press.
1A738	Wait/GetKey	(% → ?) Internal WAIT command.
42262	ATTN?	(→ flag) Returns TRUE if CANCEL has been pressed.
4243E	?ATTN_QUIT	(→) If CANCEL has been pressed, ABORTS program.
4245C	NoAttn?Semi	(→) If CANCEL has been not pressed, drops the rest of the stream.
05040	ATTNFLG@	(→ #) Recalls CANCEL key counter.
05068	ATTNFLGCLR	(→) Clears CANCEL key counter. Does not affect the key buffer.

Chapter 41

The display

41.1 Display organization

Address	Word	Action
1314D	TOADISP	Sets the text display as the active.
13135	TOGDISP	Sets the graphic display as the active.
13167	(GDISPON?)	Returns a flag indicating whether the graphic display is active.
12655	ABUFF	Returns the text grob to the stack.
12665	GBUFF	Returns the graphic grob to the stack.
12635	HARDBUFF	Returns the current grob to the stack.
12645	HARDBUFF2	Returns the menu grob to the stack.
0E128	HBUF_X_Y	(→ HBgrob #x #y)
12B6C	HARDHEIGHT	(→ #height)
12B58	(HBUFFDIMw)	(→ #width)
5187F	GBUFFGROBDIM	(→ #height #width)
		Returns dimensions of graphic grob.

41.2 Preparing the display

Address	Word	Stack and description
130AC	RECLAIMDISP	(→) Activates the text grob, clears it and sets the default size.
39531	ClrDA1IsStat	(→) Suspends clock display.
4E2CF	TURNMENUOFF	(→) Turns off menu display, enlarges ABUFF to fill screen.
4E347	TURNMENUON	(→) Turns menu grob on.
4E360	MENUOFF?	(→ flag) Returns TRUE if the menu grob is off.

Address	Word	Stack and description
130CA	(RSZVDISP)	(→) Sets standard size for currently displayed grob.
1297D	(BROADENHBUFF)	(#cols →) Broadens currently displayed grob.
12964	(HEIGHTENHBUFF)	(#rows →) Heightens currently displayed grob.
12BB7	(BROADENGROB)	(grob #cols →) Broadens graph or text grob.
12DD1	HEIGHTENGROB	(gorb #rows →) Heightens graph or text grob.
13043	(KILLADISP)	(→) Clears text display.
13061	KILLGDISP	(→) Clears graph display.

41.3 Controlling d isplay refresh

Address	Word	Notes
390CC	ClrDA1OK	(ClrDA1ValidF ClrDA1TempF ClrDA1NoCh)
38DAC	DA1OK?	
38F28	DA1OK?NOTIT	Does DA1OK?, NOT then IT.
390E5	ClrDA2aOK	(ClrDA2aValidF ClrDA2aTempF ClrDA2aNoCh)
38DFC	(DA2aOK?)	
38F41	DA2aOK?NOTIT	Does DA2aOK?, NOT then IT.
390FE	ClrDA2bOK	(ClrDA2bValidF ClrDA2bTempF ClrDA2bNoCh)
38E4C	(DA2bOK?)	
38F5A	DA2bOK?NOTIT	Does DA2bOK?, NOT then IT.
39117	ClrDA2OK	(ClrDA2aOK ClrDA2bOK)
38E9C	(DA2OK?)	
3912B	ClrDA3OK	(ClrDA3ValidF ClrDA3TempF ClrDA3NoCh)
38EB5	DA3OK?	
38F73	DA3OK?NOTIT	Does DA3OK?, NOT then IT.
39144	ClrDAsOK	(ClrDA1OK ClrDA2OK ClrDA3OK)
38F05	(DAsOK?)	
3902C	SetDA1Temp	(SetDA1TempF ClrDA1Bad ClrDA1IsStat)
39045	SetDA2aTemp	(SetDA2aTempF ClrDA2aBad)
39059	SetDA2bTemp	(SetDA2bTempF ClrDA2bBad ClrDA2bEdit)

Address	Word	Notes
39207	SetDA2OKTemp	(SetDA2aTemp SetDA2bTemp)
3921B	SetDA12Temp	(SetDA1Temp SetDA2OKTemp)
39072	SetDA3Temp	(SetDA3TempF ClrDA3Bad)
3922F	SetDAsTemp	(SetDA1Temp SetDA2OKTemp SetDA3Temp)
3932B	(SetDA1TempF)	
39339	(ClrDA1TempF)	
3931D	(DA1TempF)	
39355	(SetDA2aTempF)	
39363	(ClrDA2aTempF)	
39347	(DA2aTempF?)	
3937F	(SetDA2bTempF)	
3938D	(ClrDA2bTempF)	
39371	(DA2bTempF?)	
393A9	(SetDA3TempF)	
393B7	(ClrDA3TempF)	
3939B	(DA3TempF?)	
393D3	SetDA1NoCh	
393E1	(ClrDA1NoCh)	
393C5	(DA1NoCh?)	
393FD	SetDA2aNoCh	
3940B	(ClrDA2NoCh)	
393EF	(DA2aNoCh?)	
39427	SetDA2bNoCh	
39335	(ClrDA2bNoCh)	
39419	DA2bNoCh?	
3918A	SetDA2NoCh	
3919E	SetDA12NoCh	
39451	SetDA3NoCh	
3945F	(ClrDA3NoCh)	
39443	(DA3NoCh?)	
391C6	SetDA13NoCh	
391B2	SetDA23NoCh	
391DA	(SetDA12a3NoCh)	
391EE	SetDA123NoCh	
38FD2	SetDA1Valid	(SetDA1ValidF ClrDA1Bad ClrDA1IsStat)
38FEB	SetDA2aValid	(SetDA2aValidF ClrDA2aBad)
38FFF	SetDA2bValid	(SetDA2bValidF ClrDA2bBad ClrDA2bEdit)
3915D	SetDA2Valid	(SetDA2aValid SetDA2bValid)
39018	SetDA3Valid	(SetDA3ValidF ClrDA3Bad)
39171	(SetDAsValid)	(SetDA1Valid SetDA2Valid SetDA3Valid)
39283	(SetDA1ValidF)	
39291	(ClrDA1ValidF)	
39275	(DA1ValidF?)	
392AD	(SetDA2aValidF)	
392BB	(ClrDA2aValidF)	
3929F	(DA2aValidF?)	
392D7	(SetDA2bValidF)	
392E5	(ClrDA2bValidF)	

Address	Word	Notes
392C9	(DA2bValidF?)	
39301	SetDA3ValidF	
3930F	(ClrDA3ValidF)	
392F3	(DA3ValidF?)	
3947B	SetDA1Bad	
390A4	MENoP&FixDA1	(SetDA1Bad)
38DE8	(SetDA1BadT)	(SetDA1Bad TRUE)
39489	ClrDA1Bad	
3946D	(DA1Bad?)	
394A5	SetDA2aBad	
38E38	(SetDA2aBadT)	(SetDA2aBad TRUE)
394B3	ClrDA2aBad	
39497	DA2aBad?	
394CF	SetDA2bBad	
38E88	(SetDA2bBadT)	(SetDA2bBad TRUE)
394DD	ClrDA2bBad	
394C1	(DA2bBad?)	
390B3	MENP&FixDA12	(SetDA1Bad SetDA2aBad SetDA2bBad)
394F9	SetDA3Bad	
38EF1	(SetDA3BadT)	(SetDA3Bad TRUE)
39507	ClrDA3Bad	
394EB	(DA3Bad?)	
39248	(DAsBad?)	Is any DA "Bad"?
39523	SetDA1IsStat	
39531	ClrDA1IsStat	
39515	(DA1IsStat?)	
3954D	(SetDA2bEdit)	
3955B	(ClrDA2bEdit)	
3953F	(DA2bEdit?)	
3957A	SetNoRollDA2	
3958B	ClrNoRollDA2	
39569	(NoRollDA2?)	
39086	(?SetEditRoll)	(EditExst?NOT ITE SetDA2RollF SetDA2aNoCh)
38F8C	(InitDispModes)	(SetDAsBad ClrDA2RollF ClrDA1IsStat)
38FB9	DA2aLess1OK?	(DA2RollF? DA2aBad? NOTAND)

41.4 Clearing the display

Address	Word	Action
126DF	BLANKIT	(#startrow #rows →) Clears #rows from HARDBUFF, starting at #startrow.
134AE	CLEARVDISP	Clears HARDBUFF.
0E083	Clr8	Clears top eight rows (first status line).
0E097	Clr8-15	Clears second status line.

Address	Word	Action
0E06F	Clr16	Clears top 16 rows (both status lines).
3A546	BlankDA1	Clears status area from HARDBUFF.
3A591	BlankDA2a	Clears DA2a.
3A55F	BlankDA2	Clears DA2a and DA2b.
3A578	BlankDA12	Clears DA1 and DA2.
01F6D	CLCD10	Clears status and stack area.
01FA7	CLEARLCD	Clears entire display.
5046A	DOCLLCD	Like user word CLLCD.

41.5 Annunciator and modes control

Address	Word	Action
11361	SetLeftAnn	Sets left-shift annunciator.
1136E	ClrLeftAnn	Clears left-shift annunciator.
11347	SetRightAnn	Sets right-shift annunciator.
11354	ClrRightAnn	Clears right-shift annunciator.
1132D	SetAlphaAnn	Sets alpha annunciator.
1133A	ClrAlphaAnn	Clears alpha annunciator.
11543	(SetLock)	Sets alpha mode.
1156C	(ClrLock)	Clears alpha mode.
40D25	LockAlpha	Sets alpha mode, annunciators, etc.
40D39	UnLockAlpha	Clears alpha mode, annunciators, etc.
11501	(Lock?)	Is alpha mode set?
11320	(ClrPrgmAnn)	Clears program-entry annunciator.
11533	SetPrgmEntry	Sets program-entry mode.
1155C	(ClrPrgmEntry)	Clears program-entry mode.
11511	PrgmEntry?	Is program-entry mode set?
3EDF2	(Do1st/1st+:)	If in program mode, executes only the next object after it. If not, execution continues normally.
3EE1A	Do1st/2nd+:	If in program mode, executes the next object after it. If not in program mode, executes the rest of the stream starting at the second object after it.
53976	SetAlgEntry	Sets algebraic-entry mode.
53984	ClrAlgEntry	Clears algebraic-entry mode.
53968	AlgEntry?	Is algebraic-entry mode set?
408AA	ImmedEntry?	Returns TRUE if immediate-entry mode (program and algebraic-entry modes cleared).

41.6 Window coordinates

Address	Word	Stack and description
0E0D3	TOP8	(→ HBgrob #x1 #y #x1+131 #y1+8) Returns coordinates of first status line.
0E0FB	Rows8-15	(→ HBgrob #x1 #y1+8 #1+131 #y1+16) Returns coordinates of second status line.
0E0AB	TOP16	(→ HBgrob #x1 #y1 #x1+131 #y1+16) Returns coordinates of status area.
137B6	WINDOWCORNER	(→ #x #y) Gets coordinates of corner of window.
0E128	HBUFF_X_Y	(→ HBgrob #x #y) Returns current grob and window coordinates.
515FA	LEFTCOL	(→ #x) Gets x-coordinate of left column.
516E0	RIGHTCOL	(→ #x) Gets x-coordinate of right column.
515A0	TOPROW	(→ #y) Gets y-coordinate of top row.
515B4	BOTROW	(→ #y) Gets y-coordinate of bottom row.
13679	WINDOWXY	(#x #y →) Sets corner coordinates.
13695	(REDISPHBUFF)	(→) Sets #0 and #0 as window corner coordinates.

41.6.1 Scrolling the display

Address	Word	Action
131C8	WINDOWUP	Moves display one pixel up.
13220	WINDOWDOWN	Moves display one pixel down.
134E4	WINDOWLEFT	Moves display one pixel left.
1357F	WINDOWRIGHT	Moves display one pixel right.
4D132	SCROLLUP	Moves display one pixel up, checks for corresponding key being pressed.

Address	Word	Action
4D16E	SCROLLDOWN	Moves display one pixel down, checks for corresponding key being pressed.
4D150	SCROLLLEFT	Moves display one pixel left, checks for corresponding key being pressed.
4D18C	SCROLLRIGHT	Moves display one pixel right, checks for corresponding key being pressed.
51690	JUMPTOP	Jumps to top of display.
516AE	JUMPBOT	Jumps to bottom of display.
516E5	JUMPLEFT	Jumps to left of display.
51703	JUMPRIGHT	Jumps to right of display.
5162C	WINDOWTOP?	Is window at the top?
51645	WINDOWBOT?	Is window at the bottom?
5165E	WINDOWLEFT?	Is window at the left?
51677	WINDOWRIGHT?	Is window at the right?
12996	(ScreenUpN)	(#n →) Moves stack display up #n lines.
12A4A	(ScreenDnN)	(#n →) Moves stack display down #n lines.
12A0D	(ScreenUp)	Moves stack display up one line.
12AF6	(ScreenDn)	Moves stack display down one line.

41.7 Displaying text

41.7.1 Medium font

Address	Word	Stack and notes
1245B	DISPROW1 or DISP@01	(\$ →)
12725	DISPROW1*	(\$ →)
		Displays relative to window corner.
0E029	DISPROW1*!	(\$ →)
		Does Clr8 then DISPROW1*.
1246B	DISPROW2 or DISP@09	(\$ →)
12748	DISPROW2*	(\$ →)
		Displays relative to window corner.
1247B	DISPROW3 or DISP@17	(\$ →)
1248B	DISPROW4 or DISP@25	(\$ →)
1249B	DISPROW5	(\$ →)
124AB	DISPROW6	(\$ →)
124BB	DISPROW7	(\$ →)
124CB	DISPROW8	(\$ →)
		Only works if menu is off.
12429	DISPN	(\$ #row →)

Address	Word	Stack and notes
3A4CE	Disp5x7	(\$ #start #max →) Displays string on multiple lines, starting at #start and no using more than #max rows. New lines must be manually specified. Segments longer than 22 characters are truncated and appended with "...".

41.7.2 Large font

Address	Word	Stack
12415	BIGDISPROW1	(\$ →)
12405	BIGDISPROW2	(\$ →)
123F5	BIGDISPROW3	(\$ →)
123E5	BIGDISPROW4	(\$ →)
123C8	BIGDISPN	(\$ #max →)

41.7.3 Displaying warnings

Address	Word	Stack and description
0E047	(Save16)	(→ grob) Returns top 16 rows.
0E05B	(Restore16)	(grob →) Restores top 16 rows.
1270C	DISPSTATUS2	(\$ →) Displays message in status area using two lines.
12B85	FlashMsg	(\$ →) Displays message in status area, then restores it to normal.
38926	FlashWarning	(\$ →) Displays message in status area, beeps, then restores it to normal.
38908	DoWarning	(\$ →) Displays message, beeps and freezes status area.

Chapter 42

Graphics

42.1 Built-in grobs

Address	Word	Dims.	Grob
505B2	(NULLGROB)	0x0	Null grob
13D8C	CURSOR1	6x10	Insert cursor (arrow)
13DB4	CURSOR2	6x10	Replace cursor (solid box)
66EF1	SmallCursor	4x6	Cursor (box outline)
66ECD	MediumCursor	6x8	Cursor (box outline)
66EA5	BigCursor	6x10	Cursor (box outline)
5053C	CROSSGROB	5x5	Cross cursor (“+”)
5055A	MARKGROB	5x5	Mark symbol (“x”)
39B2D	(LineGrob)	131x2	Line (status area divider)
3A337	(StdLabelGrob)	21x8	Normal menu key
3A399	(BoxLabelGrob)	21x8	Menu key with box.
3A3FB	(DirLabelGrob)	21x8	Directory menu key.
3A45D	(InvLabelGrob)	21x8	Inverse menu key (solver)

42.2 Dimensions

Address	Word	Stack and notes
50578	GROBDIM	(grob → #height #width)
5179E	DUPGROBDIM	(grob → grob #height #width)
63C04	GROBDIMw	(grob → #width)
4F7E6	CKGROBFITS	(g1 g2 #n #m → g1 g2' #n #m) Shrinks g2 if it does not fit in grob1.

42.3 Grob handling

Address	Word	Stack an description
11679	GROB!	(grob1 grob2 #x #y →) Stores grob1 into grob2. Bang type.
11A6D	GROB!ZERO	(grob #x1 #y1 #x2 #y2 → grob') Blanks a rectangular region of the grob. Bang type.
6389E	GROB!ZERODRP	(grob #x1 #y1 #x2 #y2 →) Blanks a rectangular region of the grob. Assumes text or graph grob. Bang type.
1192F	SUBGROB	(grob #x1 #y1 #x2 #y2 → grob') Returns specified portion of grob.
128B0	XYGROBDISP	(#row #col grob →) Stores grob in HARDBUFF, expanding it if necessary.
12F94	GROB>GDISP	(grob →) Stores new graph grob.
1158F	MAKEGROB	(#height #width → grob) Creates a blank grob.
122FF	INVGROB	(grob → grob') Inverts grob data bits. Bang type.
1384A	PIXON	(#x #y →) Sets pixel in text grob.
1383B	PIXOFF	(#x #y →) Clears pixel in text grob.
13992	PIXON?	(#x #y → flag) Is pixel in text grob on?
13825	PIXON3	(#x #y →) Sets pixel in graph grob.
1380F	PIXOFF3	(#x #y →) Clears pixel in graph grob.
13986	PIXON?3	(#x #y → flag) Is pixel in graph grob on?

Address	Word	Stack an description
51893	ORDERXY#	(#x1 #y1 #x2 #y2 → #x1' #y1' #x2' #y2') To draw lines, #x2 must be greater than #x1. This function orders the coordinates so that the above condition is met.
518CA	ORDERXY%	(%x1 %y1 %x2 %y2 → %x1' %y1' %x2' %y2') ORDERXY# with real numbers.
50B17	LINEON	(#x1 #y1 #x2 #y2 →) Draws a line in text grob.
50B08	LINEOFF	(#x1 #y1 #x2 #y2 →) Clears a line in text grob.
50AF9	TOGLINE	(#x1 #y1 #x2 #y2 →) Toggles a line in text grob.
50AEA	LINEON3	(#x1 #y1 #x2 #y2 →) Draws a line in graph grob.
50ACC	LINEOFF3	(#x1 #y1 #x2 #y2 →) Clears a line in graph grob.
50ADB	TOGLINE3	(#x1 #y1 #x2 #y2 →) Toggles a line in graph grob.
11CF3	\$>BIGGROB	(\$ → grob) Makes grob of the string using the large font (5x9).
11D00	\$>GROB	(\$ → grob) Makes grob of the string using the medium font (5x7).
11F80	\$>grob	(\$ → grob) Makes grob of the string using the small font.
39632	Blank&GROB!	(\$ #x #x1 #x2 →) Clears HARDBUFF between (#x1, 0) and (#x2, 6). Converts string to grob with small characters and displays it at (#x, 0).
503D4	DOLCD>	(→ grob) Returns current display.
50438	DO>LCD	(grob →) Grob to display.

Address	Word	Stack an description
659DE	Symb>HBuff	(symb →) Displays symbolic in HARDBUFF in Equation Writer form. Enlarges HARDBUFF if necessary, so use RECLAIMDISP after.
1200C	RIGHT\$3x6	(\$ #n → flag grob) Transforms string into grob, then take all characters starting after column #n. flag is FALSE if #n is greater than the width of the grob. In this case, the whole grob is returned.
1215E	CENTER\$3x5	(grob #x #y \$ #w → grob') Creates grob from string (3x5) and embeds it at specified position (#x, #y). #w represents the maximum width of the grob created. Bang-type.

42.4 Creating menu label grobs

Address	Word	Stack and description
3A328	MakeStdLabel	(\$ → grob) Makes standard menu label.
3A38A	MakeBoxLabel	(\$ → grob) Makes label with a box.
3A3EC	MakeDirLabel	(\$ → grob) Makes directory label.
3ED6B	(DirLabel:)	(→ grob) Makes directory label with next string in the stream. Usage: :: DirLabel: \$;
3A44E	MakeInvLabel	(\$ → grob) Makes inverse label.
3EC99	Box/StdLabel	(\$ flag → grob) If TRUE makes box label, otherwise makes standard label.
3ECB2	Box/StdLbl:	(→ grob) Does Box/StdLabel with the next two objects from the stream. Usage: :: Box/StdLbl: \$ <test> ;

Address	Word	Stack and description
3ECE	(FBox/StdLbl:)	(→ grob) Takes a string and a bint from the runstream. Tests the system flag specified, does Box/StdLabel. Usage: :: FBox/StdLbl: \$ #flag ;
3ED25	(BBox/StdLbl:)	(→ grob) Takes a string and a bint from the runstream. Does BASE then EQ, and finally Box/StdLabel. Usage: :: BBox/StdLabel: \$ #base ;
3ED48	(MBox/StdLbl:)	(→ grob) Takes a string and a bint from the runstream. Does NumbMode and EQ, then Box/StdLabel. Usage: :: MBox/StdLbl: \$ #mode ;
3ED0C	Std/BoxLabel	(\$ flag → grob) If TRUE makes standard label, otherwise makes box label.
3ECD0	(FStd/BoxLbl:)	(→ grob) Takes a string and a bint from the runstream. Tests the system flag specified, does Std/BoxLabel. Usage: :: FStd/BoxLbl: \$ #flag ;
3A297	Grob>Menu	(#col grob →) Displays grob as menu label.
3A2B5	Str>Menu	(#col \$ →) Displays string as menu label.
3A2DD	Id>Menu	(#col id →) Displays id as menu label.
3A2C9	Seco>Menu	(#col :: →) Does EVAL then DoLabel.
41904	DoLabel	(#col ob →) If ob is of one of the supported types, displays a menu label. If not, generates a "Bad Argument Type" error.
3A260	(>MENU)	(#col grob →) (#col \$ →) (#col id →) (#col :: →) Works by dispatching the object type.

Chapter 43

User functions

Commands of library 2:

#	Address	Word	#	Address	Word
000	1957B	(xASR)	026	1A1D9	(xBYTES)
001	1959B	(xRL)	027	1A2BC	(xNEWOB)
002	195BB	(xRLB)	028	1A303	(xKILL)
003	195DB	(xRR)	029	1A31E	(xOFF)
004	195FB	(xRRB)	02A	1A339	(xDOERR)
005	1961B	(xSL)	02B	1A36D	(xERR0)
006	1963B	(xSLB)	02C	1A388	(xERRN)
007	1965B	(xSR)	02D	1A3A3	(xERRM)
008	1967B	(xSRB)	02E	1A3BE	(xEVAL)
009	1969B	(xR>B)	02F	1A3FE	xIFTE
00A	196BB	(xB>R)	030	1A4CD	(xIFT)
00B	196DB	(xCONVERT)	031	1A52E	(xSYSEVAL)
00C	1971B	(xUVL)	032	1A584	(xDISP)
00D	1974F	(x>UNIT)	033	1A5A4	(xFREEZE)
00E	19771	(xUBASE)	034	1A5C4	(xBEEP)
00F	197A5	(xUFACT)	035	1A5E4	(x>NUM)
010	197F7	(xTIME)	036	1A604	(xLAST)
011	19812	(xDATE)	037	1A71F	(xWAIT)
012	1982D	(xTICKS)	038	1A858	(xCLLCD)
013	19848	(xWSLOG)	039	1A873	(xKEY)
014	19863	(xACKALL)	03A	1A8BB	(xCONT)
015	1987E	(xACK)	03B	1A8D8	(x=)
016	1989E	(xSETDATE)	03C	1A995	(xNEG)
017	198BE	(xSETTIME)	03D	1AA1F	(xABS)
018	198DE	(xCLKADJ)	03E	1AA6E	(xCONJ)
019	198FE	(xSTOALARM)	03F	1AABD	(xPI)
01A	19928	(xRCLALARM)	040	1AADF	(xMAXR)
01B	19948	(xFINDLARM)	041	1AB01	(xMINR)
01C	19972	(xDELALARM)	042	1AB23	(xCONSTANTe)
01D	19992	(xTSTR)	043	1AB45	(xi)
01E	199B2	(xDDAYS)	044	1AB67	x+
01F	199D2	(xDATE+)	045	1ACDD	xNEGNEG
020	1A105	(xCRDIR)	046	1AD09	x-
021	1A125	(xPATH)	047	1ADEE	x*
022	1A140	(xHOME)	048	1AF05	x/
023	1A15B	(xUPDIR)	049	1B02D	x^
024	1A194	(xVARS)	04A	1B185	rpnXROOT
025	1A1AF	(xTVARS)	04B	1B1CA	xxROOT

#	Address	Word	#	Address	Word
04C	1B278	(xINV)	07C	1C060	(x%)
04D	1B2DB	(xARG)	07D	1C0D7	(x%T)
04E	1B32A	(xSIGN)	07E	1C149	(x%CH)
04F	1B374	(xSQRT)	07F	1C1B9	(xRAND)
050	1B426	(xSQ)	080	1C1D4	(xRDZ)
051	1B4AC	(xSIN)	081	1C1F6	(xCOMB)
052	1B505	(xCOS)	082	1C236	(xPERM)
053	1B55E	(xTAN)	083	1C274	(xSF)
054	1B5B7	(xSINH)	084	1C2D5	(xCF)
055	1B606	(xCOSH)	085	1C313	(xFS?)
056	1B655	(xTANH)	086	1C360	(xFC?)
057	1B6A4	(xASIN)	087	1C399	(xDEG)
058	1B72F	(xACOS)	088	1C3B4	(xRAD)
059	1B79C	(xATAN)	089	1C3CF	(xGRAD)
05A	1B7EB	(xASINH)	08A	1C3EA	(xFIX)
05B	1B830	(xACOSH)	08B	1C41E	(xSCI)
05C	1B8A2	(xATANH)	08C	1C452	(xENG)
05D	1B905	(xEXP)	08D	1C486	(xSTD)
05E	1B94F	(xLN)	08E	1C4A1	(xFS?C)
05F	1B9C6	(xLOG)	08F	1C520	(xFC?C)
060	1BA3D	(xALOG)	090	1C559	(xBIN)
061	1BA8C	(xLNP1)	091	1C574	(xDEC)
062	1BAC2	(xEXPM)	092	1C58F	(xHEX)
063	1BB02	xFACT	093	1C5AA	(xOCT)
064	1BB41	preFACT	094	1C5C5	(xSTWD)
065	1BB6D	(xIP)	095	1C5FE	(xRCWS)
066	1BBA3	(xFP)	096	1C619	(xRCLF)
067	1BBD9	(xFLOOR)	097	1C67F	xSTOF
068	1BC0F	(xCEIL)	098	1C783	(x>LIST)
069	1BC45	(xXPON)	099	1C79E	(xR>C)
06A	1BC71	(xMAX)	09A	1C7CA	(xRE)
06B	1BCE3	(xMIN)	09B	1C819	(xIM)
06C	1BD55	(xRND)	09C	1C85C	(xSUB)
06D	1BDD1	(xTRNC)	09D	1C8EA	(xREPL)
06E	1BE4D	(xMOD)	09E	1C95A	(xLIST>)
06F	1BE9C	(xMANT)	09F	1C98E	(xC>R)
070	1BEC8	(xD>R)	0A0	1C9B8	xSIZE
071	1BEF4	(xR>D)	0A1	1CAB4	(xPOS)
072	1BF1E	(x>HXS)	0A2	1CB0B	(x>STR)
073	1BF3E	(xHMS>)	0A3	1CB26	(xSTR>)
074	1BF5E	(xHMS+)	0A4	1CB46	(xNUM)
075	1BF7E	(xHMS-)	0A5	1CB66	(xCHR)
076	1BF9E	(xRNRM)	0A6	1CB86	(xTYPE)
077	1BFBE	(xCNRM)	0A7	1CE28	(xVTYPE)
078	1BFDE	(xDET)	0A8	1CE33	(xEQ>)
079	1BFFE	(xDOT)	0A9	1CF7B	xOBJ>
07A	1C01E	(xCROSS)	0AA	1D009	(x>ARRY)
07B	1C03E	(xRSD)	0AB	1D092	(xARRY>)

#	Address	Word	#	Address	Word
0AC	1D0DF	(xRDM)	0DC	1E661	xFUNCTION
0AD	1D186	(xCON)	0DD	1E681	(xCONIC)
0AE	1D2DC	(xIDN)	0DE	1E6a1	xPOLAR
0AF	1D392	(xTRN)	0DF	1E6C1	xPARAMETRIC
0B0	1D407	(xPUT)	0E0	1E6E1	(xTRUTH)
0B1	1D5DF	(xPUTI)	0E1	1E701	(xSCATTER)
0B2	1D7C6	(xGET)	0E2	1E721	(xHISTOGRAM)
0B3	1D8C7	(xGETI)	0E3	1E741	(xBAR)
0B4	1DD06	(xV>)	0E4	1E761	(xsME)
0B5	1DE66	(x>V2)	0E5	1E783	(xAND)
0B6	1DEC2	(x>V3)	0E6	1E809	(xOR)
0B7	1E04A	(xINDEP)	0E7	1E88F	(xNOT)
0B8	1E07E	(xPMIN)	0E8	1E8F6	(xXOR)
0B9	1E09E	(xPMAX)	0E9	1E972	(x==)
0BA	1E0BE	(xAXES)	0EA	1EA9D	(x#?)
0BB	1E0E8	(xCENTR)	0EB	1EBBE	x<?
0BC	1E126	(xRES)	0EC	1EC5D	x>?
0BD	1E150	(x*H)	0ED	1ECFC	(x<=?)
0BE	1E170	(x*W)	0EE	1ED9B	(x>=?)
0BF	1E190	(xDRAW)	0EF	1EE38	(xOLDPTR)
0C0	1E1AB	(xAUTO)	0F0	1EE53	(xPR1)
0C1	1E1C6	(xDRAX)	0F1	1EE6E	(xPRSTC)
0C2	1E1E1	(xSCALE)	0F2	1EE89	(xPRST)
0C3	1E201	(xPDIM)	0F3	1EEA4	(xCR)
0C4	1E22B	(xDEPND)	0F4	1EEBF	(xPRVAR)
0C5	1E25F	(xERASE)	0F5	1EF43	(xDELAY)
0C6	1E27A	(xPX>C)	0F6	1EF63	(xPRLCD)
0C7	1E29A	(xC>PX)	0F7	1EF7E	rpnDER
0C8	1E2BA	(xGRAPH)	0F8	1EFD2	xDER
0C9	1E2D5	(xLABEL)	0F9	1F133	(xRCEQ)
0CA	1E2F0	(xPVIEW)	0FA	1F14E	(xSTEQ)
0CB	1E31A	(xPIXON)	0FB	1F16E	(xROOT)
0CC	1E344	(xPIXOFF)	0FC	1F1DA	rpnINTG
0CD	1E36E	(xPIX?)	0FD	1D223	xINTEGRAL
0CE	1E398	(xLINE)	0FE	1F2C9	(xSUM)
0CF	1E3C2	(xTLINE)	0FF	1F354	(rpnWHERE)
0D0	1E3EC	(xBOX)	100	1F3F3	(xWHERE)
0D1	1E416	(xBLANK)	101	1F500	(xQUOTE)
0D2	1E436	(xPICT)	102	1F55D	rpnAPPLY
0D3	1E456	(xGOR)	103	1F5C5	xAPPLY
0D4	1E4E4	(xGXOR)	104	1F640	xFCNAPPLY
0D5	1E572	(xLCD>)	105	1F996	(COMPLEXDUMMY)
0D6	1E58D	(x>LCD)	106	1F9AE	(POLARDUMMY)
0D7	1E5AD	(x>GROB)	107	1F9C4	(x->Q)
0D8	1E5D2	(xARC)	108	1F9E9	(x->QPI)
0D9	1E606	(xTEST)	109	1FA59	(xMATCHUP)
0DA	1E621	(xXRNG)	10A	1FA8D	(xMATDHDN)
0DB	1E641	(xYRNG)	10B	1FAEB	xFORMUNIT

#	Address	Word	#	Address	Word
10C	1FB5D	(xPREDIV)	13C	20133	(xBARPLOT)
10D	1FB87	(xDUP)	13D	20167	(xHISTPLOT)
10E	1FBA2	(xDUP2)	13E	2018C	(xSCATRPLT)
10F	1FBBD	(xSWAP)	13F	201B1	(xLINFIT)
110	1FBD8	(xDROP)	140	201D6	(xLOGFIT)
111	1FBF3	(xDROP2)	141	201FB	(xEXPFIT)
112	1FC0E	(xROT)	142	20220	(xPWRFIT)
113	1FC29	(xOVER)	143	2025E	(xBESTFIT)
114	1FC44	(xDEPTH)	144	202CE	(xSINV)
115	1FC64	(xDROPN)	145	2034D	(sSNeg)
116	1FC7F	(xDUPN)	146	203CC	(xSCONJ)
117	1FC9A	(xPICK)	147	2044B	(xSTO+)
118	1FCB5	(xROLL)	148	20538	(xSTO-)
119	1FCD0	(xROLLD)	149	2060C	(STO/)
11A	1FCEB	(xCLEAR)	14A	20753	(xSTO*)
11B	1FD0B	(xSTOSIGMA)	14B	208F4	(xINCR)
11C	1FD2B	(xCLSIGMA)	14C	209AA	(xDECR)
11D	1FD46	(xRCLSIGMA)	14D	20A15	(xCOLCT)
11E	1FD61	(xSIGMA+)	14E	20A49	(xEXPAN)
11F	1FD8B	(xSIGMA-)	14F	20A7D	(xRULES)
120	1FDA6	(xNSIGMA)	150	20A93	(xISOL)
121	1FDC1	(xCORR)	151	20AB3	(xQUAD)
122	1FDDC	(xCOV)	152	20AD3	(xSHOW)
123	1FDF7	(xSUMX)	153	20B20	(xTAYLR)
124	1FE12	(xSYMY)	154	20B40	(xRCL)
125	1FE2D	(xSYMX2)	155	20CC4	(xSTO)
126	1FE48	(xSUMY2)	156	20D65	(xDEFINE)
127	1FE63	(xSUMXY)	157	20EFE	(xPURGE)
128	1FE7E	(xMAXSIGMA)	158	20FAA	xMEM
129	1FE99	(xMEAN)	159	20FD9	(xORDER)
12A	1FEB4	(xMINSIGMA)	15A	210FC	(xCLUSR)
12B	1FECF	(xSDEV)	15B	2115D	(xTMENU)
12C	1FEEA	(xTOT)	15C	21196	(xMENU)
12D	1FF05	(xVAR)	15D	211E1	(xRCLMENU)
12E	1FF20	(xLR)	15E	211FC	(xPVARs)
12F	1FF7	(xPREDV)	15F	2123A	(xPGDIR)
130	1FF9A	(xPREDY)	160	2125A	(xARCHIVE)
131	1FFBA	(xPREDX)	161	2133C	(xRESTORE)
132	1FFDA	(xXCOL)	162	2137F	(xMERGE)
133	1FFFA	(xYCOL)	163	213D1	xFREE
134	2001A	(xUTPC)	164	214D2	(xLIBS)
135	2003A	(xUTPN)	165	21448	(xATTACH)
136	2005A	(xUTPF)	166	2147C	(xDETACH)
137	2007A	(xUTPT)	167	21E75	(xXMIT)
138	2009A	(xSIGMACOL)	168	21E95	(xSRECV)
139	200C4	(xSCLSIGMA)	169	21EB5	(xOPENIO)
13A	200F3	(xSIGMALINE)	16A	21ED5	(xCLOSEIO)
13B	2010E	(xBINHS)	16B	21EF0	(xSEND)

#	Address	Word
16C	21F24	(xKGET)
16D	21F62	(xRECN)
16E	21F96	(xRECV)
16F	21FB6	(xFINISH)
170	21FD1	(xSERVER)
171	21FEC	(xCKSM)
172	2200C	(xBAUD)
173	2202C	(xPARITY)
174	2204C	(xTRANSIO)
175	2206C	(xKERRM)
176	22087	(xBUFLEN)

#	Address	Word
177	220A2	(xSTIME)
178	220C2	(xSBRK)
179	220DD	(xPKT)
17A	22ACA	(xINPUT)
17B	224F4	(xASN)
17C	22514	(xSTOKEYS)
17D	22548	(xDELKEYS)
17E	22586	(xRCLKEYS)
17F	225BE	(x->TAG)
180	22633	(xDTAG)

Commands of library 1792:

#	Address	Word
00	22EC3	(xIF)
01	22EFA	(xTHEN)
02	22F5B	(xELSE)
03	22FD5	xIFEND
04	22FEB	xALG->
05	22033	(xWHILE)
06	2305D	(xREPEAT)
07	230C3	(xDO)
08	230ED	(xUNTIL)
09	23103	(xSTART)
0A	231A0	(xSTARVAR)
0B	2324C	(xNEXT)
0C	23380	(xSTEP)
0D	233DF	(xIFERR)
0E	23472	(xHALT)

#	Address	Word
0F	2349C	(xSILENT')
10	234C1	xRPN->
11	235FE	x>>ABND
12	2361E	x<<
13	23639	x>>
14	23654	(x')
15	23679	xENDTIC
16	23694	xWHILEEND
17	236B9	xENDDO
18	2371F	xERRTHEN
19	2378D	(xCASE)
1A	237A8	xTHENCASE
1B	23813	(xDIR)
1C	23824	(xPROMPT)

Chapter 44

Commands by name

The “see” column tells you which section to look for a reference of that command. If nothing is listed, then you are out of luck. Try disassembling the entry point with JAZZ. Sometimes it is enough to understand the action.

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
06E97	'	34.1	03CC7	#0<>	17.4	6258A	#5+	17.3
623A0	!!append\$	20.4	03CA6	#0=	17.4	636B4	#5=	17.4
62312	!!append\$?	20.4	61A18	#0=?SEMI	35.2	6262A	#6-	17.3
62394	!!insert\$	20.4	6333A	#0=?SKIP	35.2	62691	#6*	17.3
50E59	!#1+IF<dim-1		61896	#0=case	35.2	6259A	#6+	17.3
50EA5	!#1-IF>0		63E89	#0=ITE	35.2	625AA	#7+	17.3
0BCCF	!*triland		62B6F	#0=UNTIL	36.1	62674	#8*	17.3
0BC6F	!*trior		03E0E	#1-	17.3	625BA	#8+	17.3
62376	!append\$	20.4	637CC	#1--	17.3	625CA	#9+	17.3
62F2F	!append\$SWAP	20.4	631A5	#1-{ }N	25.4	03EB1	#AND	17.3
1795A	!DcompWidth		03DEF	#1+	17.3	627F8	#-DUP	17.3
622E5	!insert\$	20.4	63808	#1-+	17.3	65094	#EXITERR	17.1
03DE0	#-	17.3	639CA	#1+'	34.1	642E3	#FIVE#FOUR	17.1
624FB	#-#2/	17.3	073DB	#1+_ONE_DO	36.2	624C6	#MAX	17.3
03EC2	#*	17.3	62809	#1+DUP	17.3	624BA	#MIN	17.3
191B9	#*OVF	17.3	63281	#1+LAST\$	20.4	6428A	#ONE#27	17.1
03EF7	#/	17.3	62F75	#1+NDROP	29	63065	#-OVER	17.3
167D8	#:>\$	20.3	611A3	#1+PICK	29	61172	#-PICK	29
64E3C	#_102	17.1	612F3	#1+ROLL	29	03DC7	#PUSHA-	
03DBC	#+	17.3	1DABB	#1+ROT	17.3	612CC	#-ROLL	17.3
637CC	#-+1	17.3	62E26	#1+SWAP	17.3	62E12	#-SWAP	17.3
63808	#+-1	17.3	61353	#1+UNROLL	17.3	642D1	#THREE#FOUR	17.1
627D5	#+DUP	17.3	622B6	#1<>	17.4	642BF	#TWO#FOUR	17.1
63051	#+OVER	17.3	622A7	#1=	17.4	6429D	#TWO#ONE	17.1
61184	#+PICK	17.3	63353	#1=?SKIP	35.2	642AF	#TWO#TWO	17.1
612DE	#+ROLL	17.3	63D26	#1=case	35.2	6132C	#-UNROLL	29
62DFE	#+SWAP	17.3	6264E	#10*	17.3	64209	#ZERO#ONE	17.1
6133E	#+UNROLL	17.3	625DA	#10+	17.3	6427A	#ZERO#SEVEN	17.1
03CE4	#<	17.4	62E4E	#1-1SWAP	17.3	6571F	\$_''	20.2
03D4E	#<>	17.3	625EA	#12+	17.3	6572D	\$_::	20.2
63D3A	#<>case	17.4	6281A	#1-DUP	17.3	65711	\$_[]	20.2
63673	#<3	17.3	62FD9	#1-ROT	17.3	65703	\$_{ }	20.2
63D12	#<case	35.2	63245	#1-SUB\$	20.4	656F5	\$_<<>>	20.2
63E9D	#<ITE	35.2	5E4A9	#1-SWAP	17.3	65749	\$_2DQ	20.2
03D19	#=	17.4	28558	#1-UNROT	17.3	65757	\$_ECHO	20.2
6336C	#=?SKIP	35.2	03E4E	#2-	17.3	65769	\$_EXIT	20.2
6186C	#=case	35.2	03E6F	#2*	17.3	657A7	\$_GRAD	20.2
618D3	#=casedrop	35.2	03E8E	#2/	17.3	6573B	\$_LRParens	20.2
63547	#=casedrpfls	35.2	03E2D	#2+	17.3	656C5	\$_R<<	20.2
62C2D	#=ITE	35.2	611BE	#2+PICK	29	656D5	\$_R<Z	20.2
6448A	#=POSCOMP	25.1	61318	#2+ROLL	29	65797	\$_RAD	20.2
03D83	#>	17.4	61365	#2+UNROLL	29	6577B	\$_Undefined	20.2
167E4	#>\$	20.3	636C8	#2<>	17.4	656E5	\$_XYZ	20.2
5435D	#>%	19.2	6229A	#2=	17.4	11CF3	\$>BIGGROB	42.3
63399	#>?SKIP	35.2	625FA	#3-	17.3	11D00	\$>GROB	42.3
636F0	#>1	17.4	6256A	#3+	17.3	11F80	\$>grob	42.3
63D4E	#>2case	35.2	611D2	#3+PICK	29	05B15	\$>ID	31.2
63D67	#>case	35.2	62289	#3=	17.4	11D8F	\$5x7	41.7
05A75	#>CHR	20.3	6260A	#4-	17.3	2A981	%-	18.3
059CC	#>HXS	21.1	6257A	#4+	17.3	2A94F	%%-	18.3
63EB1	#>ITE	35.2	611E1	#4+PICK	29	2A99A	%%*	18.3
07E50	#>ROMPTR	28.2	6261A	#5-	17.3	62FED	%%*ROT	18.3

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
62EA3	%%*SWAP	18.3	2A8C1	%=	18.4	2ACF1	%ACOS	18.3
63C18	%%*UNROT	18.3	2A88A	%>	18.4	2AE13	%ACOSH	18.3
2A562	%%.1	18.1	543F9	%>#	17.2	2ABBA	%ALOG	18.3
2B3DD	%%.4	18.1	2A5C1	%>%%	18.2	2AD38	%ANGLE	18.3
2A57C	%%.5	18.1	2A95B	%>%%-	18.3	2ACC1	%ASIN	18.3
2A9E8	%%/	18.3	2AA9E	%>%1/	18.3	2AE00	%ASINH	18.3
63B82	%%/>%	18.3	2AD5B	%>%ANGLE	18.3	2AD21	%ATAN	18.3
2AA5F	%%^	18.3	2AAF6	%>%SQRT	18.3	2AE26	%ATANH	18.3
2A943	%%+	18.3	62E8F	%>%SWAP	18.3	2AF73	%CEIL	18.3
2A81F	%%<	18.4	2A8A0	%>=	18.4	2AA30	%CH	18.3
2A8AB	%%<=	18.4	05C27	%>C%	19.2	2A920	%CHS	18.3
2A87F	%%>	18.4	2A673	%>HMS	38	2AE62	%COMB	18.3
2A5B0	%%>%	18.4	22618	%>TAG	22	2AC40	%COS	18.3
2A895	%%>=	18.4	2A2B4	%0	18.1	2ADDA	%COSH	18.3
51A07	%%>C%	19.2	2A738	%0<	18.4	2A622	%D>R	18.3
2A4C6	%%0	18.1	2A7CF	%0<>	18.4	650A8	%e	18.1
2A727	%%0<	18.4	2A76B	%0=	18.4	2AB2F	%EXP	18.3
2A80B	%%0<=	18.4	5F127	%0=case	35.3	2AB42	%EXPM1	18.3
2A7BB	%%0<>	18.4	2A799	%0>	18.4	2AE39	%EXPONENT	18.3
2A75A	%%0=	18.4	2A7F7	%0>=	18.4	2B0C4	%FACT	18.3
2A788	%%0>	18.4	2A2C9	%1	18.1	2AF86	%FLOOR	18.3
2A7E3	%%0>=	18.4	2A386	%-1	18.1	2AF4D	%FP	18.3
2A4E0	%1	18.1	50276	%1-	18.3	2A6C8	%HMS-	38
2AA92	%1/	18.3	2AAAF	%1/	18.3	2A6A0	%HMS+	38
2A596	%10	18.1	50262	%1+	18.3	2A68C	%HMS>	38
2B2DC	%12	18.1	5F181	%1=case	35.3	2AF60	%IP	18.3
2A4FA	%2	18.1	5F267	%-1=case	35.3	2EC11	%IP>#	17.2
0F688	%2PI	18.1	650E7	%10	18.1	2AB6E	%LN	18.3
2A514	%3	18.1	62BF1	%10*	18.3	2ABA7	%LNP1	18.3
2A52E	%4	18.1	415F1	%100	18.1	2AB81	%LOG	18.3
2A548	%5	18.1	1CC03	%11	18.1	2A930	%MANTISSA	18.3
2B300	%60	18.1	1CC1D	%12	18.1	2A6F5	%MAX	18.3
2B1FF	%7	18.1	1CC37	%13	18.1	62D81	%MAXorder	18.3
2A8F0	%ABS	18.3	1CC51	%14	18.1	2A472	%MAXREAL	18.1
2AD08	%ACOSRAD	18.3	1CC85	%15	18.1	2A487	%-MAXREAL	18.1
2AD4F	%ANGLE	18.3	1CD3A	%16	18.1	2A70E	%MIN	18.3
2AD6C	%ANGLEDEG	18.3	1CD54	%17	18.1	2A49C	%MINREAL	18.1
2AD7C	%ANGLERAD	18.3	1CDF2	%18	18.1	2A4B1	%-MINREAL	18.1
2ACD8	%ASINRAD	18.3	650FC	%180	18.1	2ABDC	%MOD	18.3
2A910	%CHS	18.3	1CE07	%19	18.1	2AE4C	%NFACT	18.3
2AC57	%COS	18.3	2A2DE	%2	18.1	2AA81	%NROOT	18.3
2AC68	%COSDEG	18.3	2A39B	%-2	18.1	2A9C9	%OF	18.3
2ADC7	%COSH	18.3	5F1EA	%2=case	35.3	2AE75	%PERM	18.3
2AC78	%COSRAD	18.3	1CC6B	%20	18.1	2A443	%PI	18.1
2AB1C	%EXP	18.3	1CCA4	%21	18.1	2B4BB	%POL>%REC	18.3
2AF99	%FLOOR	18.3	1CCC3	%22	18.1	2A655	%R>D	18.3
2AF27	%H>HMS	38	1CCE2	%23	18.1	2AFC2	%RAN	18.3
2AF99	%INT	18.3	1CD01	%24	18.1	2B044	%RANDOMIZE	18.3
2AB5B	%LN	18.3	1CD20	%25	18.1	2B48E	%REC>%POL	18.3
2AB94	%LNP1	18.3	1CD73	%26	18.1	2A8D7	%SGN	18.3
2A6DC	%MAX	18.3	1CD8D	%27	18.1	2ABEF	%SIN	18.3
2B4C5	%P>R	18.3	2A2F3	%3	18.1	2ADAE	%SINH	18.3
2B498	%R>P	18.3	2A3B0	%-3	18.1	2B4F2	%SPH>%REC	18.3
2AC06	%SIN	18.3	65126	%360	18.1	2AB09	%SQRT	18.3
2AC17	%SINDEG	18.3	2A308	%4	18.1	2AA0B	%T	18.3
2AD95	%SINH	18.3	2A3C5	%-4	18.1	2AC91	%TAN	18.3
2AC27	%SINRAD	18.3	2A31D	%5	18.1	2ADED	%TANH	18.3
2AAEA	%SQRT	18.3	2A3DA	%-5	18.1	05193	&\$	20.4
2ACA8	%TANRAD	18.3	2A332	%6	18.1	63F6A	&\$SWAP	20.4
2A9BC	%*	18.3	2A3EF	%-6	18.1	0521F	&COMP	25.1
494B4	%.1	18.1	2A347	%7	18.1	0518A	&HXS	21.2
650BD	%.5	18.1	2A404	%-7	18.1	05445	::N	25.5
2A9FE	%/	18.3	2A35C	%8	18.1	632D1	::NEVAL	25.5
2AA70	%^	18.3	2A419	%-8	18.1	62A61	?>ROMPTR	28.2
2A974	%+	18.3	320B1	%80	18.1	715B1	?ACCPTR>	
51BE4	%+SWAP	18.3	2A371	%9	18.1	4243E	?ATTNQUIT	40.2
2A871	%<	18.4	2A42E	%-9	18.1	42078	?BlinkCursor	
2A8B6	%<=	18.4	2A900	%ABS	18.3	6317D	?CARCOMP	25.1
2A8CC	%<>	18.4	18CD7	%ABSCOECE	18.3	3FF1B	?CaseKeyDef	14.4

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
3FF48	?CaseRomptr@		648BD	>LASTRAM-WOR		64DE2	2GROB	17.1
40E3D	?ClrAlg		06F9F	>R	34	51532	2HXSLLIST?	17.2
40E5B	?ClrAlgSetPr		41422	>Review\$		64C66	2LLIST	17.1
3A1CA	?DispMenu		3E35F	>SkipKey		63FFB	2Ob>Seco	25.5
39B85	?DispStack		052EE	>T\$	20.4	63FBA	2OVER	29
3959C	?DispStatus		05E81	>TAG	22	615F0	2PUTLAM	31.3
386D8	?FlashAlert	39.2	052FA	>TCOMP	25.1	61AE9	2'RCOLARPITE	35.1
619E0	?GOTO	34	1884D	0LASTOWDOB!	30	6114E	2RDROP	34
3FA7A	?Key>UKeyOb		1884D	0LastRomWrd!	30	04099	2REAL	17.1
63FCE	?Ob>Seco	25.5	2B789	1/X15		626E5	2skipcola	34.2
26A2D	?OKINALG		63259	1_#1-SUB	20.4	62001	2SWAP	29
1854F	?PURGE_HERE	37.2	63259	1_#1-SUB\$	20.4	61B55	3@REVAL	34
39BF3	?RollUpDA2		614AC	10GETLAM	31.3	60F4B	3DROP	29
62A84	?ROMPTR>	28.2	61675	10PUTLAM	31.3	6140E	3GETLAM	31.3
61A3B	?SEMI	34	6312D	10UNROLL	29	611FE	3PICK	29
638E4	?SEMIDROP	34	614BC	11GETLAM	31.3	63740	3PICK#+	17.3
0712A	?SKIP	35.1	61685	11PUTLAM	31.3	63C68	3PICK3PICK	29
62D9F	?SKIPSWAP	35.1	614CC	12GETLAM	31.3	630B5	3PICKOVER	29
18513	?STO_HERE	37.2	61695	12PUTLAM	31.3	62EDF	3PICKSWAP	29
62F1B	?SWAP	35.1	614DC	13GETLAM	31.3	61600	3PUTLAM	31.3
62F5C	?SWAPDROP	35.1	616A5	13PUTLAM	31.3	61160	3RDROP	34
448C1	?TogU/LCase		614EC	14GETLAM	31.3	64E64	3REAL	17.1
0797B	@	37.2	616B5	14PUTLAM	31.3	626DC	3skipcola	34.2
07943	@LAM	31.3	614FC	15GETLAM	31.3	60FAC	3UNROLL	29
24EA6	{ }>DIR		616C5	15PUTLAM	31.3	60F7E	4DROP	29
05459	{ }N	25.4	6150C	16GETLAM	31.3	60F83	4DropLoop	
100E0	~BRbrowse		616D5	16PUTLAM	31.3	61438	4GETLAM	31.3
450E0	~BRDispItems		6151C	17GETLAM	31.3	52D26	4NULLLAM{ }	31.3
130E0	~BRdone		616E5	17PUTLAM	31.3	6121C	4PICK	29
530E0	~BRGetItem		6152C	18GETLAM	31.3	63754	4PICK#+	17.3
490E0	~BRinverse		616F5	18PUTLAM	31.3	62DE5	4PICK#+SWAP	17.3
120E0	~BRoutput		6153C	19GETLAM	31.3	62DE5	4PICK+SWAP	17.3
180E0	~BRRclC1		61705	19PUTLAM	31.3	630C9	4PICKOVER	29
170E0	~BRRclCurRow		3AA0A	1A/LockA		62EF3	4PICKSWAP	29
030E0	~BRStoC1		62DB3	1ABND SWAP	31.3	61615	4PUTLAM	31.3
520E0	~BRViewItem		634B6	1GETABND	31.3	60FBB	4ROLL	29
000B3	~Choose		613B6	1GETLAM	31.3	62864	4ROLLDROP	29
050B3	~ChooseMenu0		55288	1GETLAMSWP1+	31.3	630A1	4ROLLOVER	29
060B3	~ChooseMenu1		62F07	1GETSWAP	31.3	63001	4ROLLROT	29
070B3	~ChooseMenu2		634CF	1LAMBIND	31.3	62ECB	4ROLLSWAP	29
2D0B3	~DoCKeYCance		34D2B	1NULLLAM{ }	31.3	6109E	4UNROLL	29
2B0B3	~DoCKeYChAll		615E0	1PUTLAM	31.3	6113C	4UNROLL3DROP	29
2A0B3	~DoCKeYCheck		514DC	1REV		62D09	4UNROLLEDUP	29
2E0B3	~DoCKeYOK		15978	1stkdecomp\$w		63015	4UNROLLROT	29
2C0B3	~DoCKeYUnChA		6362D	2#0=OR	17.4	60F72	5DROP	29
590B0	~DoKeyCancel		2B470	2%>%	18.2	6145C	5GETLAM	31.3
5A0B0	~DoKeyOK		2B45C	2%>%%	18.2	6123A	5PICK	29
090B1	~DoMKeyOK		61B45	2@REVAL	34	61625	5PUTLAM	31.3
000B1	~DoMsgBox		6154C	20GETLAM	31.3	60FD8	5ROLL	29
C80B0	~GetFieldVal		61715	20PUTLAM	31.3	62880	5ROLLDROP	29
C50B0	~gFldVal		6155C	21GETLAM	31.3	626AE	5skipcola	34.2
850B0	~grobAlertIc		61725	21PUTLAM	31.3	610C4	5UNROLL	29
860B0	~grobCheckKe		6156C	22GETLAM	31.3	60F66	6DROP	29
050B0	~IFMenuRow1		61735	22PUTLAM	31.3	6146C	6GETLAM	31.3
060B0	~IFMenuRow2		03258	2DROP	29	6125E	6PICK	29
360B3	~LEDISPItem		6254E	2DROP00	17.1	61635	6PUTLAM	31.3
350B3	~LEDISPList		3FDBD	2DropBadKey		61002	6ROLL	29
300B3	~LEDISPComp		62B0B	2DROPFALSE	33.1	610FA	6UNROLL	29
120E4	~MESRclEqn		031AC	2DUP	29	60F54	7DROP	29
020B1	~MsgBoxMenu		63704	2DUP#+	17.3	6147C	7GETLAM	31.3
630E3	~PCunpack		6289B	2DUP#<	17.4	61282	7PICK	29
580E7	~UTTYPEEXT0?		628B5	2DUP#=	17.4	61645	7PUTLAM	31.3
110E7	~UTVUNSlArg		628D1	2DUP#>	17.4	6106B	7ROLL	29
073A5	+LOOP	36.2	63C40	2DUP5ROLL	29	62BC4	7UNROLL	29
3E3E1	<DelKey		635D8	2DUPEQ	33.2	6148C	8GETLAM	31.3
3E2DD	<SkipKey		611F9	2DUPSWAP	29	612A9	8PICK	29
3E4CA	>DelKey		64E0A	2EXT	17.1	61655	8PUTLAM	31.3
0525B	>H\$	20.4	632E5	2GETEVAL	31.3	6103C	8ROLL	29
052C6	>HCOMP	25.1	613E7	2GETLAM	31.3	63119	8UNROLL	29

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
6149C	9GETLAM	31.3	0D5E5	ASRW5		3A591	BlankDA2a	41.4
61665	9PUTLAM	31.3	34301	Attn#	17.1	126DF	BLANKIT	41.4
162B8	a%>\$	20.3	42262	ATTN?	40.2	515B4	BOTROW	41.6
162AC	a%>\$,	20.3	0CA60	ATTNchk		3EC99	Box/StdLabel	42.4
39FB0	AbbrevStack?		64FC2	ATTNERR	17.1	3ECB2	Box/StdLbl:	42.4
07497	ABND	31.3	05040	ATTNFLG@	40.2	2E108	BUILDKPACKET	
04EA4	ABORT	32.1	05068	ATTNFLGCLR	40.2	05DBC	C%>%%	19.2
12655	ABUFF	41.1	420F5	ATTNxcp		519F8	C%>C%	19.2
30914	ACK_INIT		40BC9	AtUserStack	30	51B2A	C%0=	19.4
2B7DC	ADDF		491D5	AUTOSCALE		5193B	C%1	19.1
1265A	addrADISP		0948E	BAK>OB	28.4	51B91	C%CHS	19.3
4226A	addrATTNFLG		081D9	BAKNAME	28.4	51BC1	C%CONJ	19.3
0E7D3	addrClkOnNib		70601	BANKMTHDS		4F408	C%>#	
00D48	addrKEYSTATE		54050	BASE	39.1	05D2C	C%>%	19.2
136AC	addrLINECNTg		0010D	BAU		519CB	C%>%%	19.2
0188D	addrORghost		071A2	BEGIN	36.1	519DF	C%>%%SWAP	19.2
04E66	addrTEMPENV		66EA5	BigCursor	42.1	524AF	C%0	19.1
179E8	addrTEMPTOP		123C8	BIGDISPN	41.7	51B43	C%0=	19.4
1263A	addrVDISP		12415	BIGDISPROW1	41.7	5F13B	C%0=case	35.3
1264A	addrVDISP2		12405	BIGDISPROW2	41.7	524F7	C%1	19.1
1605F	addtics		123F5	BIGDISPROW3	41.7	5196A	C%-1	19.1
42EC7	AdjEdModes		123E5	BIGDISPROW4	41.7	51EFA	C%1/	19.3
069F7	ADJMEM		074D0	BIND	31.3	5F19F	C%1=case	35.3
312DA	adr_uart_han		44F42	BindMatVars		5F285	C%-1=case	35.3
047CF	adrDISABLE_K		64CE8	BINT_115d	17.1	5F208	C%2=case	35.3
32CB6	adrGraphPrth		64CF2	BINT_116d	17.1	52062	C%ABS	19.3
047DD	adrKEYBUFFER		64D06	BINT_122d	17.1	52863	C%ACOS	19.3
42284	adrTIMEOUTCL		64D1A	BINT_130d	17.1	52836	C%ACOSH	19.3
5EF15	AEQ1stcase	35.4	64D24	BINT_131d	17.1	52305	C%ALOG	19.3
5F048	AEQopscase	35.4	64BE4	BINT_65d	17.1	52099	C%ARG	19.3
071AB	AGAIN	36.1	64C84	BINT_91d	17.1	52804	C%ASIN	19.3
2B770	aH>HMS		64C8E	BINT_96d	17.1	5281D	C%ASINH	19.3
0115A	AINRTN		64E1E	BINT253	17.1	52675	C%ATAN	19.3
422A1	ALARM?	38	64E28	BINT255d	17.1	527EB	C%ATANH	19.3
0E235	ALARMS@		64BDA	BINT40h	17.1	52374	C%C^C	19.3
42113	ALARMxcp		64D10	BINT80h	17.1	52360	C%C^R	19.3
53968	AlgEntry?	41.5	64DD8	BINTC0h	17.1	51B70	C%CHS	19.3
3981B	AlgEntryStat		53EB0	bit-	21.3	51BB2	C%CONJ	19.3
1568F	ALGeq?		5431C	bit#%-	21.3	52571	C%COS	19.3
001FF	allkeys		542EA	bit#%*	21.3	52648	C%COSH	19.3
010E5	AllowIntr		542BD	bit#%/	21.3	52193	C%EXP	19.3
324A6	AllowPrlcdCl		54349	bit#%+	21.3	521E3	C%LN	19.3
2B67D	aMODF		542FE	bit#%#-	21.3	522BF	C%LOG	19.3
03B46	AND	33.1	542D1	bit#%*	21.3	52342	C%R^C	19.3
18873	AND\$	20.4	5429F	bit#%/	21.3	520CB	C%SGN	19.3
63CEA	ANDcase	35.1	54330	bit#%#+	21.3	52530	C%SIN	19.3
63E61	ANDITE	35.1	53ED3	bit*	21.3	5262F	C%SINH	19.3
63DDF	ANDNOTcase	35.1	53F05	bit/	21.3	52107	C%SQRT	19.3
39673	AngleStatus		53EA0	bit+	21.3	525B7	C%TAN	19.3
0010B	ANNCTRL		53D04	bitAND	21.3	5265C	C%TANH	19.3
2E4DC	APNDCTRLF	20.4	53E65	bitASR	21.3	519B7	C>Im%	19.2
35491	apndvarlst	25.4	008E6	BITMAP		519A3	C>Re%	19.2
38C08	AppDisplay!		53D4E	bitNOT	21.3	61CE9	CACHE	31.3
38C18	AppDisplay@		00100	BITOFFSET		40CE9	CacheStack	
62BB0	APPEND_SPACE	20.4	53D15	bitOR	21.3	050ED	CAR\$	20.4
38C98	AppError!		53E0C	bitRL	21.3	05089	CARCOMP	25.1
38CAB	AppError@		53E3B	bitRLB	21.3	0010E	CARDCTL	
38C68	AppExitCond!		53DA4	bitRR	21.3	0010F	CARDSTAT	
38C78	AppExitCond@		53DE1	bitRRB	21.3	61993	case	35.1
38C38	AppKeys!		53D5E	bitSL	21.3	6191F	case2drop	35.1
38C58	AppKeys0		53D6E	bitSLB	21.3	61970	case2DROP	35.1
38CFB	AppMode?		53D81	bitSR	21.3	63583	case2drpfls	35.1
47984	APPprompt1!		53D91	bitSRB	21.3	63BEB	caseDEADKEY	35.6
479A7	APPprompt2		53D26	bitXOR	21.3	63BEB	caseDoBadKey	35.6
00202	argtypeerr		45676	Blank\$	20.4	618F7	casedrop	35.1
00203	argvalerr		39632	Blank&GROB!	42.3	6194B	caseDROP	35.1
64BE4	ARRAYREAL	17.1	3A546	BlankDA1	41.4	63BD2	caseDrpBadKy	35.6
03562	ARSIZE	23.1	3A578	BlankDA12	41.4	6356A	casedrpfls	35.1
0D5F6	ASLW5		3A55F	BlankDA2	41.4	638B2	casedrptru	35.1

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
63169	caseERRJMP	35.6	654E9	CHR_E	20.1	18A8D	CK2NOLASTWD	30
6359C	caseFALSE	35.1	6559F	CHR_e	20.1	18A5B	CK3	30
63B05	caseSIZEERR	35.6	654F0	CHR_F	20.1	18EF0	CK3&Dispatch	30
634E3	caseTRUE	35.1	655A6	CHR_f	20.1	18A68	CK3NOLASTWD	30
2BEE1	CCSB1		654F7	CHR_G	20.1	18B92	CK4	30
0516C	CDR\$	20.4	655AD	CHR_g	20.1	18F01	CK4&Dispatch	30
05153	CDRCOMP	25.1	654FE	CHR_H	20.1	18B9F	CK4NOLASTWD	30
1215E	CENTER\$3x5	42.3	655B4	CHR_h	20.1	18B6D	CK5	30
10E82	cfC	18.1	65505	CHR_I	20.1	18F12	CK5&Dispatch	30
10E68	cfF	18.1	655BB	CHR_i	20.1	18B7A	CK5NOLASTWD	30
05AB3	CHANGETYPE	39.2	6565C	CHR_Integral	20.1	42C3D	CkChr00	20.5
64CC0	char	17.1	6550C	CHR_J	20.1	49C54	CkEQUUtil	
444EE	Char>Edit		655C2	CHR_j	20.1	4F7E6	CKGROBFITS	42.3
42D82	CharEdit		65513	CHR_K	20.1	0A00E	CKLBCRC	
27D00	check_pdata		655C9	CHR_k	20.1	18C34	CKN	30
511E3	CHECKHEIGHT		6551A	CHR_L	20.1	18C4A	CKNNOLASTWD	30
04708	CHECKKEY		655D0	CHR_l	20.1	545A0	cknumdsptchl	27.1
4E266	CHECKMENU		65663	CHR_LeftPar	20.1	51148	CKPICT	30.1
41111	CheckMenuRow		65521	CHR_M	20.1	63B2D	CKREAL	30.1
51166	CHECKPICT		655D7	CHR_m	20.1	37B44	CKREF	37.1
4A9AF	CHECKPVARs		65528	CHR_N	20.1	1F05B	CKSYMBTYPE	30.1
32CAF	ChkGrHook		655DE	CHR_n	20.1	0D9C7	CKTIME	
325AA	ChkLowBat		6566A	CHR_Newline	20.1	01F6D	CLCD10	41.4
30B1D	CHOOSE_INIT		6552F	CHR_O	20.1	01FA7	CLEARLCD	41.4
65448	CHR_	20.1	655E5	CHR_o	20.1	51125	CLEARMENU	
6544F	CHR_-	20.1	65536	CHR_P	20.1	134AE	CLEARVDISP	41.4
65433	CHR_#	20.1	655EC	CHR_p	20.1	018E2	clkspd	
6543A	CHR_*	20.1	65671	CHR_Pi	20.1	0EB81	CLKTICKS	38
65456	CHR_.	20.1	6553D	CHR_Q	20.1	0D7A1	CLKUTL1	
65425	CHR_...	20.1	655F3	CHR_q	20.1	315C6	CLOSEUART	
6545D	CHR_/	20.1	65544	CHR_R	20.1	315F9	CloseUart	
654AA	CHR_:	20.1	655FA	CHR_r	20.1	0E06F	Clr16	41.4
654B1	CHR_;	20.1	65678	CHR_RightPar	20.1	0E083	Clr8	41.4
65694	CHR_[20.1	6554B	CHR_S	20.1	0E097	Clr8-15	41.4
6569B	CHR_]	20.1	65601	CHR_s	20.1	39FD2	ClrAbbrevStk	
656A2	CHR_{	20.1	6567F	CHR_Sigma	20.1	53984	ClrAlgEntry	41.5
656A9	CHR_}	20.1	65686	CHR_Space	20.1	1133A	ClrAlphaAnn	41.5
65441	CHR_+	20.1	65552	CHR_T	20.1	38D17	ClrAppMode	
654B8	CHR_<	20.1	65608	CHR_t	20.1	38D9B	ClrAppSuspOK	
65640	CHR_<<	20.1	65559	CHR_U	20.1	39489	ClrDA1Bad	41.3
656B0	CHR_<=	20.1	6560F	CHR_u	20.1	39531	ClrDA1IsStat	41.2
656BE	CHR_<>	20.1	6568D	CHR_UndScore	20.1	390CC	ClrDA1OK	41.3
654BF	CHR_=	20.1	65560	CHR_V	20.1	394B3	ClrDA2aBad	41.3
654C6	CHR_>	20.1	65616	CHR_v	20.1	390E5	ClrDA2aOK	41.3
65639	CHR_>	20.1	65567	CHR_W	20.1	394DD	ClrDA2bBad	41.3
656B7	CHR_>=	20.1	6561D	CHR_w	20.1	39435	ClrDA2bNoCh	41.3
65647	CHR_>>	20.1	6556E	CHR_X	20.1	390FE	ClrDA2bOK	41.3
65464	CHR_0	20.1	65624	CHR_x	20.1	39117	ClrDA2OK	41.3
6541E	CHR_00	20.1	65575	CHR_Y	20.1	39507	ClrDA3Bad	41.3
6546B	CHR_1	20.1	6562B	CHR_y	20.1	3912B	ClrDA3OK	41.3
65472	CHR_2	20.1	6557C	CHR_Z	20.1	39144	ClrDAsOK	41.3
65479	CHR_3	20.1	65632	CHR_z	20.1	2BBE2	CLRFRC	
65480	CHR_4	20.1	05A51	CHR>#	17.2	1136E	ClrLeftAnn	41.5
65487	CHR_5	20.1	6475C	CHR>\$	20.3	53A90	ClrNewEditL	
6548E	CHR_6	20.1	01160	CINRTN		3958B	ClrNoRollDA2	41.3
65495	CHR_7	20.1	41CA2	Ck&DecKeyLoc	40.2	11354	ClrRightAnn	41.5
6549C	CHR_8	20.1	18F9D	CK&DISPATCH0	30	2D9B2	ClrServMode	
654A3	CHR_9	20.1	18FB2	CK&DISPATCH1	30	53761	ClrSysFlag	39.1
654CD	CHR_A	20.1	18FA9	CK&DISPATCH2	30	423D3	clrtimeout	
65583	CHR_a	20.1	142FB	Ck&Freeze		53755	ClrUserFlag	39.1
6564E	CHR_Angle	20.1	18A1E	CK0	30	40C94	CMDSTO	
654D4	CHR_B	20.1	23768	CK0ATTNABORT		0010A	CMODE	
6558A	CHR_b	20.1	18A15	CK0NOLASTWD	30	41D92	CodePl>%rc.p	40.2
654DB	CHR_C	20.1	18AA5	CK1	30	18CEA	COERCE	17.1
65591	CHR_c	20.1	18ECE	CK1&Dispatch	30	12770	COERCE\$22	20.4
654E2	CHR_D	20.1	1592D	CK1NoBlame	30	194F7	COERCE2	17.1
65598	CHR_d	20.1	18AB2	CK1NOLASTWD	30	62CE1	COERCEDUP	17.1
6542C	CHR_DblQuote	20.1	18A80	CK2	30	5380E	COERCEFLAG	35.1
65655	CHR_Deriv	20.1	18EDF	CK2&Dispatch	30	62E7B	COERCESWAP	17.1

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
06FD1	COLA	34.2	7DD17	D/DCONJ		39958	DispDir?Tim2	
61A6D	COLA_EVAL	34.2	7DD35	D/DCOS		3988B	DispDir?Time	
62986	COLAcase		7DD40	D/DCOSH		3A00D	DispEditLine	
6296D	COLACOLA	34.2	7DDF0	D/DDER		430CF	DispILPrompt	
6381C	COLAITE		7DD4B	D/DEXP		3A1E8	DispMenu	
629A1	COLANOTcase		7DD4B	D/DEXPM1		3A1FC	DispMenu.1	
61A8E	COLARPITE	35.1	7DD82	D/DIFTE		12429	DISPN	
633B2	COLASKIP	34.2	7DE06	D/DINTEGRAL		01BBD	DispOff	
5573D	COLAthexFCN		7DD56	D/DINV		01B8F	DispOn	
01FD3	Coldstart		7DD61	D/DLN		133AB	disprange	
09B73	COMPCONFRCRC		7DD6C	D/DLNP1		1245B	DISPROW1	41.7
18EBA	COMPEVAL	34	7DD77	D/DLOG		12725	DISPROW1*	41.7
1F996	COMPLEXDUMMY		7DD8D	D/DSIN		1246B	DISPROW2	41.7
396C8	ComVecStatus		7DD98	D/DSINH		12748	DISPROW2*	41.7
6506C	Connecting	17.1	7DDA3	D/DSQ		1247B	DISPROW3	41.7
00B02	constuniterr		7DDAE	D/DSQRT		1248B	DISPROW4	41.7
08D08	CONTEXT!	37.3	7DE11	D/DSUM		1249B	DISPROW5	41.7
08D5A	CONTEXT@	37.3	7DDB9	D/DTAN		124AB	DISPROW6	41.7
00101	CONTRAST		7DDC4	D/DTANH		124BB	DISPROW7	41.7
256E4	convertbase		7DDFB	D/DWHERE		124CB	DISPROW8	41.7
4651C	CopyColsLeft		6384E	D0=DSKTOP		395BA	DispStatus	
4677E	CopyColsRight		01C31	D0->Row1		1270C	DISPSTATUS2	41.7
46625	CopyRowsDown		01C58	D0->Sft1		39B0A	DispStsBound	
46409	CopyRowsUp		6385D	D1=DSKTOP		39AF1	DispTimeReq?	
7DF87	COPYVAR		3946D	DA1Bad?	41.3	153FC	DispVarsUtil	
137DC	corner		38DAC	DA1OK?	41.3	2BBB5	DIV2	
00104	CRC		38F28	DA1OK?NOTIT	41.3	06A8E	DIV5	
08696	CREATE	37.2	39497	DA2aBad?	41.3	2B977	DIVF	
184E1	CREATEDIR	37.3	38FB9	DA2aLess1OK?	41.3	63DB7	dLISTcase	30.1
06AD8	CREATETEMP		38F41	DA2aOK?NOTIT	41.3	073F7	DO	36.2
00113	CRER		39419	DA2bNoCh?	41.3	1502F	DO#EXIT	32.1
2E4F0	CRLF\$	20.2	38F5A	DA2bOK?NOTIT	41.3	15048	DO\$EXIT	32.1
4DA0D	CROSS_HAIRS		38EB5	DA3OK?	41.3	15007	DO%EXIT	32.1
4DA76	CROSS_OFF		38F73	DA3OK?NOTIT	41.3	50438	DO>LCD	42.3
5053C	CROSSGROB	42.1	25223	DaGNTc		14088	DO>STR	20.3
4ECAD	CROSSMARKON		63DA3	dARRAYcase	30.1	3EE1A	Do1st/2nd+:	
1578D	CRUNCH	27.1	0CC0E	DATE	38	02BAA	DOACPTR	16
15941	CRUNCHNoBlam		0CC5B	DATE+DAYS	38	029E8	DOARRY	16
0D618	CSLW5		0CFD9	Date>d\$	38	3FDD1	DoBadKey	
0D607	CSRW5		0EE50	Date>hxsl3	38	3FDfE	'DoBadKey	34.1
4248E	CtlAlarm!		0D4AD	DAY#		3FE12	'DoBadKeyT	34.1
4E442	CURRENTMARK?		0D744	Day>Date		02B62	DOBAK	16
427AF	Cursor&Disp		0D62F	DCHXW		2EC84	DOBAUD	
444A5	CURS0R_END?		17980	DcompWidth@		1415A	DOBEEP	39.2
13E85	CURS0R_OFF		424DA	DCursor		53C43	DOBIN	39.1
13D8C	CURS0R1	42.1	0CC39	DDAYS	38	074E4	DOBIND	
13DB4	CURS0R2	42.1	009A5	Debounce		02911	DOBINT	16
13D55	cursorblink-		30D31	DECODE		2EDE1	DOBUFLN	
13D28	cursorblink+		15B13	DECOMP\$		4F179	DOC>PX	
7DBF8	D/D-		041ED	DEEPSLEEP	39.2	42475	DoCAlarmKey	
7DBE2	D/D*		04292	DeepSleep		029BF	DOCHAR	16
7DC03	D/D/		2512D	delimcase		140F1	DOCHR	20.3
7DDCF	D/D^		0314C	DEPTH	29	5046A	DOCLLCD	41.4
7DDDA	D/D^X		7DC54	derprod1		02977	DOCMP	16
7DDE5	D/D^Y		7DC0E	derquot		02DCC	DOCODE	16
7DBED	D/D+		63E07	dIDNTNcase	30.1	02D9D	DOCOL	16
7DC72	D/D=		035A9	DIMLIMITS	23.1	31854	docr	
7DC7D	D/DABS		01115	DisableIntr		31868	DOCR	
7DCA1	D/DACOS		1245B	DISP@01	41.7	05981	DoCRC	
7DCAC	D/DACOSH		1246B	DISP@09	41.7	0597E	DoCRCc	
7DC72	D/Dalg=		1247B	DISP@17	41.7	02A2C	DOCSTR	16
7DCB7	D/DALOG		1248B	DISP@25	41.7	53C5B	DODEC	39.1
7DE1C	D/DAPPLY		00120	DISP1CTL		31FFD	DODELAY	
7DCC2	D/DARG		00130	DISP2CTL		40DD4	DoDelim	
7DCCD	D/DASIN		3A4CE	Disp5x7	41.7	40DF7	DoDelims	
7DCD8	D/DASINH		13B51	DISPCHAR+PC		140AB	DODISP	
7DCE3	D/DATAN		4E6EF	DispCoord1		0299D	DOECMP	16
7DCEE	D/DATANH		4A055	DISPCOORD2		166FB	DOENG	39.1
7DCF9	D/DCHS		398F4	DispDir?Tim1		4B60C	DOERASE	

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
02955	DOEREL	16	4E582	DRAWBOX#		6321D	DUPNULLCOMP?	25.1
02ADA	DOEXT	16	50758	DRAWLINE#3		63A9C	DUPONE	17.1
02B88	DOEXT0	16	4C639	drax		630DD	DUPPICK	29
02BAA	DOEXT1	16	63E1B	dREALNcase	35.5	630F1	DUPROLL	29
02BCC	DOEXT2	16	03244	DROP	29	62C19	DUPROMPTR@	28.2
02BEE	DOEXT3	16	6394D	DROP'	34.1	61FA9	DUPROM-WORD?	28.2
02C10	DOEXT4	16	637F4	DROP#1-	17.3	62FB1	DUPROT	29
620DC	DOFALSE		3FDC7	DropBadKey		62A2F	DUPSAFE@	37.2
2E876	DOFINISH		4D11E	DROPDEADTRUE		61745	DUPTEMPENV	31.3
3B211	DoFirstRow		627A7	DROPDUP	29	63AD8	DUPTWO	17.1
166E3	DOFIX	39.1	6210C	DROPFALSE	33.1	62193	DUPTYPEARRY?	33.3
0554C	DOGARBAGE		3A9B8	'DROFFALSE	34.1	6212A	DUPTYPEBINT?	33.3
4CE6F	DOGRAPHIC		03249	DropLoop		62020	DUPTYPECHAR?	33.3
02B1E	DOGROB	16	63466	DROPLOOP	36.2	6217E	DUPTYPECMP?	33.3
185C7	DoHere:		63FA6	DROPNDROP	29	621E7	DUPTYPECOL?	33.3
53C37	DOHEX	39.1	04D3E	DROPNULL\$	20.2	62154	DUPTYPECSTR?	33.3
02A4E	DOHSTR	16	62946	DROPONE	17.1	6204A	DUPTYPEEXT?	33.3
02A4E	DOHXS	16	63029	DROPOVER	29	621FC	DUPTYPEGROB?	33.3
02E48	DOIDNT	16	632F9	DROPRDROP	34	6213F	DUPTYPEHSTR?	33.3
199EB	DoInputForm	13	62FC5	DROPROT	29	62035	DUPTYPEIDNT?	33.3
2EDA6	DOKERRM		6270C	DROPSWAP	29	62115	DUPTYPELAM?	33.3
40454	DoKeyOb		62726	DROPSWAPDROP	29	62211	DUPTYPELIST?	33.3
41904	DoLabel	42.4	2DDC4	DropSysErr\$		62169	DUPTYPEREAL?	33.3
02E6D	DOLAM	16	18308	DropSysObs		621A8	DUPTYPEROMP?	33.3
503D4	DOLCD>	42.3	62103	DROPTTRUE	33.1	621BD	DUPTYPERRRP?	33.3
02B40	DOLIB	16	62535	DROPZERO	17.1	621D2	DUPTYPESYMB?	33.3
02A74	DOLIST	16	00103	DSPCTL		62226	DUPTYPETAG?	33.3
02A0A	DOLNKARRY	16	00102	DTEST		61380	DUPUNROT	29
40DC0	DoMenuKey		62193	DTYPEARRY?	33.3	63A88	DUPZERO	17.1
41934	DoMenuKeyNS		621E7	DTYPECOL?	33.3	63A29	dvarlsBIND	
40350	DoNameKeyLRS		62154	DTYPECSTR?	33.3	00003	DZP	
40337	DoNameKeyRS		62211	DTYPELIST?	33.3	7DC88	easyabs	
44C31	DoNewMatrix		62169	DTYPEREAL?	33.3	42BD4	Echo\$Key	
3A71C	DoNextRow		61EA7	DUMP	31.3	42BB6	Echo\$NoChr00	
53C4F	DOOCT	39.1	03188	DUP	29	3EE47	Echo2Macros	
44FE7	DoOldMatrix		63925	DUP'	34.1	42AE4	EchoChrKey	
2EB37	DOOPENIO		63687	DUP#<7	17.4	039EF	ECUSER	
2ECCA	DOPARITY		6347F	DUP#0_DO	36.2	15A40	ederr	
2E8D1	DOPKT		622D4	DUP#0<>	17.4	15A0E	EDITDECOMP\$	20.3
3ADED	DoPlotMenu		633F8	DUP#0<>WHILE	36.1	15B31	editdecomp\$w	
3A735	DoPrevRow		62266	DUP#0=	17.4	63DCB	EditExstCase	
31EE2	DOPRLCD		61891	DUP#0=case	35.2	42D32	EditLevell	
4FOAC	DOPX>C		63CBD	DUP#0=csDROP	35.2	53A4A	EditLExists?	
2B07B	DORANDOMIZE	18.3	618A8	DUP#0=csedrp	35.2	44683	EDITLINE\$	
1572B	DORCLE		63E48	DUP#0=IT	35.2	3BDFA	EditMenu	
02933	DOREAL	16	63EC5	DUP#0=ITE	35.2	44730	EDITPARTS	
02E92	DOROMP	16	6292F	DUP#1-	17.3	443CB	EditString	
02A96	DORRP	16	628EB	DUP#1+	17.3	0403F	EIGHT	17.1
2EE18	DOSBRK		6119E	DUP#1+PICK	29	040A3	EIGHTEEN	17.1
166EF	DOSCI	39.1	622C5	DUP#1=	17.4	6103C	EIGHTROLL	17.1
18CA7	DOSIZEERR	32.2	626F7	DUP#2+	17.3	64C34	EIGHTY	17.1
3BE54	DoSolvrMenu		63295	DUP\$>ID	31.2	64C3E	EIGHTYONE	17.1
2EE97	DOSRECV		63BAA	DUP\$0=	18.4	0405D	ELEVEN	17.1
16707	DOSTD	39.1	62C05	DUP@	37.2	64127	Embedded?	
1C6A2	DOSTOALLF	39.1	634CA	DUP1LAMBIND	31.3	30BD7	ENCODE	
15717	DOSTOE		611F9	DUP3PICK	29	30BBE	ENCODE1PKT	
1C6E3	DOSTOSYSF	39.1	63704	DUP3PICK#+	17.3	00019	ENTERCODE	
14137	DOSTR>	20.3	61610	DUP4PUTLAM	31.3	03B2E	EQ	33.2
53CAA	dostws	39.1	61099	DUP4UNROLL	29	635F1	EQ:	33.2
02AB8	DOSYMB	16	641CC	DupAndThen		61933	EQcase	35.5
02AFC	DOTAG	16	62CB9	DUPDUP	29	618BA	EQcasedrop	35.5
2ED10	DOTRANSIO		635EC	DUPEQ:	33.2	4E46A	EQCURSOR?	
620C3	DOTRUE		5179E	DUPGROBDIM	42.2	63E2F	EQIT	35.5
186E8	DOTVARS%		631E1	DUPINCOMP	25.3	63E75	EQITE	35.5
18779	DOVARS	37.3	63411	DUPINDEX@	36.2	152FF	EqList?	
1A7C9	dowait	38	627BB	DUPLen\$	20.4	64593	EQLookup	25.1
38908	DoWarning	41.7	63231	DUPLenCOMP	25.1	63605	EQOR	33.2
0DB51	dowutil		63209	DUPNULL\$?	20.5	6303D	EQOVER	33.2
167BF	DPRADIX?	39.1	63A6F	DUPNULL{ }?	25.4	03B97	EQUAL	33.2

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
63CFE	EQUALcase	35.5	38926	FlashWarning	41.7	4ADB0	GETSCALE	
517F3	EQUALcasedro	35.5	0D6D8	FLOAT		314E5	GETSERIAL	
63CA4	EQUALcasedrp	35.5	00D57	Flush		2FFBA	GetStrLen	
635C4	EQUALNOT	33.2	00D71	FLUSH	40.2	2FFB7	GetStrLenC	
63DF3	EQUALNOTcase	35.5	00D8E	FlushAttn		2FFB4	GetStrLenStk	
63619	EQUALOR	33.2	00D71	FLUSHKEYS	40.2	039BE	GETTEMP	
644A3	EQUALPOSCOMP	25.1	3136C	FLUSHRSBUF		04D64	GETTHEMESG	32.1
15744	EQUATION		0417F	FORTY	17.1	012EE	GetTimChk	
141E5	ERRBEEP	32.1	64B3A	FORTYEIGHT	17.1	0130E	GetTime++	
32B08	ErrFixEIRU		64B1C	FORTYFIVE	17.1	04714	GETTOUCH	
04ED1	ERRJMP	32.1	64B12	FORTYFOUR	17.1	41F3F	GetUserKeys	
05023	Errjmp		64B44	FORTYNINE	17.1	4B139	GETXMAX	
63155	'ERRJMP	34.1	04189	FORTYONE	17.1	4B10C	GETXMIN	
10F80	ErrjmpC		64B30	FORTYSEVEN	17.1	505C6	GETXPOS	
04CE6	ERROR@	32.1	64B26	FORTYSIX	17.1	505E4	getxpos	
04D33	ERRORCLR	32.1	0419D	FORTYTHREE	17.1	4B14D	GETYMAX	
6383A	ERROROUT	32.1	04193	FORTYTWO	17.1	4B120	GETYMIN	
04D0E	ERRORSTO	32.1	04017	FOUR	17.1	5068D	GETYPOS	
04E5E	ERRSET	32.1	64C0C	FOURFIVE	17.1	506AB	getypos	
0CBC4	ErrTime		60FBB	FOURROLL	29	619CB	GOTO	34
04EB8	ERRTRAP	32.1	63001	FOURROLLROT	29	10F40	GPErrjmpC	
06F8E	EVAL	34	0407B	FOURTEEN	17.1	065AA	GPMEMERR	
1583C	EVALCRUNCH		64BF8	FOURTHREE	17.1	03672	GPOverWrALp	
18F23	EvalNoCK	30	64BEE	FOUR TWO	17.1	62096	GPOverWrFLp	
2BE99	EXAB0		0417F	FOURTY	17.1	0366F	GPOverWrR0Lp	
2BEA7	EXAB2		6109E	FOURUNROLL	29	62073	GPOverWrT/FL	
2D517	EXCHINITPK		05F42	GARBAGE	37.1	62076	GPOverWrTLp	
419C4	ExitAction!		0613E	GARBAGECOL		54266	GPPushA	
6400F	ExitAtLOOP	36.2	12665	GBUFF	41.1	620D2	GPPushFLoop	
4CF68	ExitFcn		5187F	GBUFFGROBDIM	41.1	620B6	GPPushT/FLp	
4CE4C	EXITFCNsto		4CF05	GDISPCENTER		620B9	GPPushTLoop	
04E37	EXITMSGSTO	32.1	61305	get1		4CEE7	GraphicExit	
61C1C	EXPAND	21.2	2BFFD	GETAB0		11679	GROB!	42.3
0407B	EXT	17.1	2BFE3	GETAB1		11A6D	GROB! ZERO	42.3
05481	EXTN	25.2	0371D	GETATELN		6389E	GROB! ZERODRP	42.3
25C41	Extobcode		0B809	getBPOFF		4F78C	GROB+#	
6508A	EXTOBOB	17.1	21922	GetBVars		12F94	GROB>GDISP	42.3
64DF6	EXTREAL	17.1	2C031	GetCD0		3A297	Grob>Menu	42.4
64E00	EXTSYM	17.1	3205C	GetChkPRTPAR		50578	GROBDIM	42.2
0BC01	failed	33.1	04A41	GETDF		63C04	GROBDIMw	42.2
03AC0	FALSE	33.1	45D1F	GetElt		1518D	GsstFIN	
639B6	FALSE'	34.1	4A0AA	GetEqN		3FE44	H/W>KeyCode	
2F934	FalseFalse	33.1	04E07	GETEXITMSG	32.1	3FE26	H/WKey>KeyOb	
6350B	FALSETRUE	33.1	4AF63	GETINDEP		12635	HARDBUFF	41.1
6350B	FalseTrue	33.1	2EA4F	GetIOPAR		12645	HARDBUFF2	41.1
49BA5	FcnUtilEnd		2F39C	GetKermPkt#		12B6C	HARDHEIGHT	41.1
04085	FIFTEEN	17.1	42159	GETKEY		0E128	HBUFF_X_Y	41.1
64B4E	FIFTY	17.1	420A0	GETKEY*		12DD1	HEIGHTENGROB	41.2
64B9E	FIFTYEIGHT	17.1	4203C	GetKeyOb		53860	HISTON?	
64B80	FIFTYFIVE	17.1	307E2	GETKP		08D92	HOMEDIR	37.3
64B76	FIFTYFOUR	17.1	075A5	GETLAM	31.3	4B5AD	HSCALE	
64BA8	FIFTYNINE	17.1	617D8	GETLAMPAR		0DB91	HXDCW	
64B58	FIFTYONE	17.1	45AE0	GetMat/Vec		544EC	HXS#HXS	21.4
64B94	FIFTYSEVEN	17.1	25452	getmatchtok		5453F	HXS<=HXS	21.4
64B8A	FIFTYSIX	17.1	415C9	GetMenu%		54552	HXS<HXS	21.4
64B6C	FIFTYTHREE	17.1	2E7EF	GETNAME		544D9	HXS==HXS	21.4
64B62	FIFTYTWO	17.1	26162	GetNextToken		05A03	HXS>#	17.2
644EE	FindlstT.1		6595A	getnibs	39.2	54061	HXS>\$	20.3
644D0	FindlstTrue	25.1	4B364	GETPARAM		540BB	hxs>\$	20.3
0EBD5	FindNext		4B0DA	GETPMIN&MAX		5435D	HXS>%	18.2
04021	FIVE	17.1	04A0B	GETPROC		5452C	HXS>=HXS	21.4
64C5C	FIVEFOUR	17.1	067D2	GETPTR		54500	HXS>HXS	21.4
60FD8	FIVEROLL	29	26FAE	GETPTRFALSE		4A95A	ICMPDRPRTDRP	
64C70	FIVESIX	17.1	05143	GETPTRLOOP		0402B	id	17.1
64C52	FIVETHREE	17.1	25CE1	GETPTRTRUE		4AB1C	ID_X	31.1
610C4	FIVEUNROLL	29	4B062	GETPTYPE		4AB59	ID_Y	31.1
32B1A	FixEIRU		4AFDB	GETRES		05BE9	ID>\$	20.3
17ADB	FixRRP		4B7D8	GetRes		3A2DD	Id>Menu	42.4
12B85	FlashMsg	41.7	514AF	GETRHS		05F2E	ID>TAG	22

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
5129C	'IDFUNCTION	34.1	00C0D	kermssendmsg		05B7D	MAKE\$N	
0402B	idnt	17.1	3FB1A	Key>StdKeyOb		03442	MAKEARRY	23.1
5F0AA	idntcase	35.5	3FA57	Key>U/SKeyOb		017A6	makebeep	39.2
5F0CD	idntlamcase	35.5	04999	KeyInBuff?		3A38A	MakeBoxLabel	42.4
512D8	'IDPARAMETER	34.1	42402	KEYINBUFFER?	40.2	3A3EC	MakeDirLabel	42.4
512C4	'IDPOLAR	34.1	40A6F	KeyOb!		1158F	MAKEGROB	42.3
64C98	IDREAL	17.1	40A82	KeyOb@		115B3	makegrob	
0716B	IDUP	34	40A95	KeyOb0		3A44E	MakeInvLabel	42.4
4744F	'IDX	31.1	13061	KILLGDISP	41.2	3A4AB	MakeLabel	42.4
408AA	ImmedEntry?	41.5	3016B	KINVISLF		4B323	MAKEPICT#	
62B88	INCOMPDROP	25.3	2FEDD	KVIS		4AAEA	MAKEPVARS	
2F383	IncrLAMPKNO		2FEC9	KVISLF		3A328	MakeStdLabel	42.4
510AD	INDEPVAR		418F4	LabelDef!		7DF0E	MANMENU*/	
07221	INDEX@	36.2	2D45A	LAMKLIST		7DF19	MANMENU^	
63790	INDEX@#-	36.2	2F211	LAMKML		7DF03	MANMENU+-	
07270	INDEXSTO	36.2	2D46D	LAMKMODE		7DF66	MANMENUATG	
00305	infreserr		2D3EE	LAMKP		7DF3A	MANMENUCSIV	
1A2DA	INHARDROM?	39.2	2D441	'LamKPSto		7DF50	MANMENUCX	
44277	InitEd&Modes		2D493	LAMKRM		7DF45	MANMENUEQ	
4428B	InitEdLine		2D3FB	LAMLNAME		7DF24	MANMENUEXP	
44394	InitEdModes		63A3D	'LAMLNAMESTO		7DF2F	MANMENULN	
0970A	InitEnab		2D40E	LAMOBJ		7DF5B	MANMENUTRG	
2E6BE	InitIOEnv		2D41D	LAMOPOS		5055A	MARKGROB	
40F86	InitMenu		2D3B1	LAMPACKET		643EF	matchob?	
41679	InitMenu%		2D3A0	LAMPKNO		643F9	matchob?Lp	
45023	InitOldMat		2D3C6	LAMRETRY		35CAE	MATCON	23.2
385E8	InitSysUI		6326D	LAST\$	20.4	37E0F	MATREDIM	23.2
41741	InitTrack:		426F1	LastErow?		3811F	MATTRN	23.2
636A0	INNER#1=	25.3	419E4	LastMenuDef!		2D396	MAXRETRY	
054AF	INNERCOMP	25.3	419F4	LastMenuDef@		357A8	MDIMS	23.1
62C41	INNERDUP	25.3	4186E	LastMenuRow!		62F9D	MDIMSDROP	23.1
43200	InputLATtn		41881	LastMenuRow@		66ECD	MediumCursor	42.1
43179	InputLEnter		18621	LastNonNull		05F61	MEM	39.2
42F44	InputLine		50D78	LASTPT?		03FF9	MEMERR	17.1
53A3C	INSERT?		08326	LASTRAM-WORD	37.3	390A4	MENOp&FixDA1	41.3
53A2E	INSERT_MODE		0011D	LBR		390B3	MENP&FixDA12	41.3
03F24	IntDiv		0011C	LCR		418A4	MenuDef@	
64F04	INTEGER337	17.1	4256B	LCursor		40828	MenuKey	
06B4E	INTENNOTREF?	37.1	515FA	LEFTCOL	41.6	41944	MenuKeyLS!	
00A03	intrptderr		05636	LEN\$	20.4	41914	MenuKeyNS!	
122FF	INVGROB	42.3	0567B	LENCOMP	25.1	41924	MenuKeyNS@	
00110	IOC		05616	LENHXS	21.1	41964	MenuKeyRS!	
2EC25	IOCheckReal		4E37E	LINECHANGE		407FB	MenuMaker	
0011F	IRAM@		00128	LINECOUNT		4E2AC	MENUOFF	
0011A	IRC		00125	LINENIBS		4E360	MENUOFF?	41.2
3E5CD	IStackKey		50B08	LINEOFF	42.3	41848	MenuRow!	
07249	ISTOP@	36.2	50ACC	LINEOFF3		4185B	MenuRow@	
5182F	ISTOP-INDEX	36.2	50B17	LINEON		418D4	MenuRowAct!	
07295	ISTOPSTO	36.2	50AEA	LINEON3		5EFD9	MEQ1stcase	35.4
619BC	IT	35.1	39F6F	LINESOFSTACK		5EFF9	MEQopscase	35.4
61AD8	ITE	35.1	04021	list	17.1	63F1A	metaROTDUP	26.1
61A58	ITE_DROP	35.1	24C0D	List		28296	metatail	26.4
63D7B	j%0=case	35.3	64C48	LISTCMP	17.1	5F061	Mid1stcase	35.4
63CD6	jEQcase	35.5	64C7A	LISTLAM	17.1	6509E	MINUSONE	17.1
07258	JINDEX@	36.2	20B9A	LISTRCL	37.2	4085A	Modifier	
072AD	JINDEXSTO	36.2	64C3E	LISTREAL	17.1	3FE7B	ModifierKey?	
04D87	JstGETTHEMSG	32.1	04929	liteslp		0670C	MOVEDOWN	
07264	JSTOP@	36.2	41175	LoadTouchTbl		06A1D	MOVEDSD	
072C2	JSTOPSTO	36.2	40D25	LockAlpha	41.5	069C5	MOVEDSU	
516AE	JUMPBOT	41.6	15E83	longhxs		06992	MOVERSD	
516E5	JUMPLEFT	41.6	6452F	Lookup	25.1	06A53	MOVERSU	
51703	JUMPRIGHT	41.6	64548	Lookup.1		066B9	MOVEUP	
51690	JUMPTOP	41.6	07334	LOOP	36.2	7DF7C	MOVEVAR	
2D730	KDispRow2		2D564	Loop		0D8AE	mpopl%	
2D74E	KDispStatus2		00108	LPD		53EE4	MPY	
1553B	KeepUnit		00109	LPE		03991	MUL#	
2EAE2	KERMOPEN		5EE10	M-1stcasechs	35.4	2B91E	MULTF	
00C10	kermpktmsg		62ABB	MACRODCMP		62F75	N+1DROP	29
00C0E	kermrecvmsg		05B79	MAKE\$		63B19	NcaseSIZEERR	35.6

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
63B46	NcaseTYPEERR	35.6	5FB49	NOTcaseFALSE	35.1	6377C	OVER#-	17.3
7DEED	nCOLCTQUOTE		638CB	NOTcaseTRUE	35.1	6372C	OVER#+	17.3
7DEE2	nCustomMenu		63AEC	NOTcsdrpfls	35.1	6365F	OVER#<	17.4
0326E	NDROP	29	27244	NOTLISTcase	35.5	620EB	OVER#	17.4
169A5	NDROPFALSE	33.1	27264	NOTROMPcase	35.5	6187C	OVER#=case	35.2
031D9	NDUP	29	27254	NOTSECOCcase	35.5	636DC	OVER#>	17.4
5E370	NDUPN	29	5590E	nscknum2	27.1	6364B	OVER#0=	17.4
255FB	need'case		62D1D	NTHCOMDDUP	25.1	63105	OVER#2+UNROL	29
00302	negunferr		62B9C	NTHCOMPDROP	25.1	63C90	OVER5PICK	29
25632	newBASE		056B6	NTHCOMP	25.1	63141	OVERARSIZE	23.1
4C09B	NEWINDEP		644BC	NTHOF	25.1	62CCD	OVERDUP	29
65238	NEWLINE\$	20.2	055DF	NULL\$	20.2	63439	OVERINDEX@	36.2
63191	NEWLINE\$&\$	20.4	0556F	NULL\$?	20.5	05622	OVERLEN\$	20.4
63191	NEWLINE&\$	20.4	62D59	NULL\$SWAP	20.2	62D31	OVERSWAP	29
4E4B0	NEWMARK		1613F	NULL\$TEMP	20.2	62D31	OVERUNROT	29
6480B	NEXTCOMPOB	25.1	055FD	NULL:	25.5	6207D	OverWrF/TLp	
0AB82	NEXTLIBBAK	28.1	055E9	NULL{}	25.4	620A0	OverWrFLoop	
6443A	nextpos		055B7	NULLCOMP?	25.1	6209D	OverWrT/FLp	
179D0	NEXTRRPOB		055D5	NULLHXS	21.2	62080	OverWrTLoop	
4BFAE	NEXTSTEP		15777	NULLID	31.1	5E0DA	P{ }N	25.4
29A8D	nextsym'R		34D30	NULLLAM	31.1	29E46	PACK	
255BD	ngsizecase		3EC71	NullMenuKey	14.5	29E21	PACKSB	
04049	NINE	17.1	2534A	nultrior		238A4	palparse	20.3
040AD	NINETEEN	17.1	5F0FA	num0=case	35.3	62B1F	PALPTRDCMP	
7DEA0	nINTGACOS		5F154	num1=case	35.3	62B5B	palrompdcmp	
7DECC	nINTGALOG		5F23A	num-1=case	35.3	38985	ParOuterLoop	14
7DE95	nINTGASIN		5F1BD	num2=case	35.3	40AD9	Parse.1	
7DEAB	nINTGATAN		5EDFC	numblstcase		40B2E	ParseFail	
7DE5E	nINTGCOS		3303F	NUMSOLVE		1848C	PATHDIR	37.3
7DE7F	nINTGCOSH		7DBAB	nWHEREEDER		39971	PathStatus	
7DED7	nINTGEXPM		7DBD7	nWHEREFCNAPP		2A62C	PI/180	18.1
7DE27	nINTGINV		7DBA0	nWHEREIFTE		032E2	PICK	29
7DEB6	nINTGLN		7DBB6	nWHEREINTG		20CAD	PICTRCL	
7DEC1	nINTGLOG		7DBC1	nWHEREISUM		1383B	PIXOFF	42.3
7DE32	nINTGSIGN		7DBCC	nWHEREWHERE		1380F	PIXOFF3	42.3
7DE53	nINTGSIN		216D8	OB>BAKcode		1384A	PIXON	42.3
7DE74	nINTGSINH		63FE7	Ob>Seco	25.5	13992	PIXON?	42.3
7DE48	nINTGSQ		42DC8	ObEdit		13986	PIXON?3	42.3
7DE3D	nINTGSQRT		05944	OCRC	39.2	13825	PIXON3	42.3
7DE69	nINTGTAN		1A1FC	OCRC%	39.2	63F56	plDRPpZparg	27.3
7DE8A	nINTGTANH		076AE	OFFSRRP	0	4B6D9	PLOTERR	
54C63	nmetasyms	30.1	00303	ofloerr		50DA5	PlotOneMore?	
4245C	NoAttn?Semi	40.2	03FF9	ONE	17.1	4B765	PLOTPREP	
538F8	NOBLINK		636F0	ONE#>	17.4	49AD3	PointDerivUt	
40D93	NoEdit?case		073CE	ONE_DO	36.2	49F06	PointMoveCur	
4488A	NoEditLine?		63385	ONE_EQ		1F9AE	POLARDUMMY	
3EC85	NoExitAction		23EED	ONE{ }N	25.4	38B45	POLErrorTrap	
14483	nohalt		63A15	ONECOLA	34.2	38AEB	POLKeyUI	14.1
10FC6	NOHALTERR	32.2	63AC4	ONEDUP	17.1	38B90	POLRestoreUI	14.1
53AE4	NoIgnoreAlm		63533	ONEFALSE	33.1	38B77	POLResUI&Err	14.1
5E984	nonopcase	35.5	6399D	ONEFALSE '	34.1	389BC	POLSaveUI	14.1
06E8E	NOP	34	64CAC	ONEHUNDRED	17.1	38A64	POLSetUI	14.1
632BD	'NOP	34.1	63AC4	ONEONE	17.1	06641	POP#	
01FDA	norecCSseq		62E67	ONESWAP	17.1	29FDA	POP1%	
01FBD	norecPWLseq		00C74	OnKeyDown?		29FD0	POP1%SPLITA	
03AF2	NOT	33.1	00C80	OnKeyStable?		03F5D	POP2#	
61B72	NOT?DROP	35.1	2EB62	OpenIO		2A002	POP2%	
619F3	NOT?GOTO	34	3187C	OpenIOPrT		3251C	PopASavptr	
61A2C	NOT?SEMI	35.1	3161E	OpenUartClr		0D92C	POPDATE%	
62F43	NOT?SWAPDROP	35.1	03B75	OR	33.1	61A02	popflag	
0712A	NOT_IT	35.1	18887	OR\$	20.4	04840	POPKEY	40.2
633C6	NOT_UNTIL	36.1	629BC	ORcase	35.1	3251F	PopSavptr	
633DF	NOT_WHILE	36.1	51893	ORDERXY#	42.3	0D948	POPTIME%	
62C55	NOTAND	33.1	518CA	ORDERXY%	42.3	31289	POPUART	
619AD	NOTcase	35.1	635B0	ORNOT	33.1	0000A	portnotaverr	
61910	NOTcase2drop	35.1	05902	OSIZE	39.2	0AAB2	PORTSTATUS	28.1
61984	NOTcase2DROP	35.1	30ED2	OUTUART		645B1	POS\$	20.4
618E8	NOTcasedrop	35.1	032C2	OVER	29	645BD	POS\$REV	20.4
61960	NOTcaseDROP	35.1	63961	OVER '	34.1	645B1	POSCHR	20.4

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
645BD	POSCHREV	20.4	4B189	PUTYMIN		61FB6	ROM-WORD?	28.2
64426	POSCOMP	25.1	4AB2A	PvarsC%0		61FCF	Rom-Word?	
00301	posunferr		63F92	pZpargSWAPUn	27.3	06806	ROOM	
1BB41	preFACT	43	06EEB	'R	34	49BD2	RootUtil	
4E497	PREMARKON		07012	R@	34	03295	ROT	29
1863A	PrevNonNull	37.3	0701F	R>	34	63768	ROT#-	17.3
08376	PREVRAM-WORD	37.3	53BDD	RAD?	39.1	63718	ROT#+	17.3
39853	PrgmEntrStat		2B7B0	RADD1		62DCC	ROT#+SWAP	17.3
11511	PrgmEntry?	41.5	2B7CA	RADDF		637B8	ROT#1+	17.3
32161	PRINT		082E3	RAM-WORDNAME	37.3	5FB76	ROT#1+UNROT	17.3
32B74	PrintGrob		2520A	'Rapndit		62DCC	ROT+SWAP	17.3
32387	PRINTxNLF		00114	RBR		62726	ROT2DROP	29
028FC	PRLG		2BEB5	RCAB0		62C7D	ROT2DUP	29
49709	PromptIdUtil	20.3	2BEC0	RCAB2		62C91	ROTAND	33.1
00C13	prtparerr		2BECB	RCCD0		60F21	ROTDROP	29
5EB1C	psh	26.1	2BED6	RCCD2		60F0E	ROTDROPSWAP	29
5E3AC	psh&	0	01AD7	RCKBp		62775	ROTDUP	29
5E706	pshl&	26.3	42E86	Rcl&Do:		62CA5	ROTOVER	29
5E7A5	pshl&rev	26.3	64023	RclHiddenVar	37.4	6112A	ROTROT2DROP	29
5E401	pshltop&	26.3	1C637	RCLSYSF	39.1	60EE7	ROTSWAP	29
5E4D1	pshtop&	26.3	1C64E	RCLUSERF	39.1	63F2E	ROTUntop&	0
5E67A	pshzer	26.3	00111	RCS		45C2F	RowElt#	
5EA9F	pshzerpsharg	27.3	06FB7	RDROP	34	4489E	ROWNUM	
323E9	PSubErr		62958	RDROPCOLA	34	0E0FB	Rows8-15	41.6
5133C	PtoR		5DE55	RDROFFALSE	33.1	070FD	RPIT	35.1
7DF71	PTYPE>PINFO		14EA5	RDUP	34	070C3	RPITE	35.1
6408C	PuHiddenVar	37.4	51A37	Re>C%	19.2	234C1	RPN->	43
5E4A9	pull	26.3	03FF9	real	17.1	1F55D	rpnAPPLY	43
355C8	PULLCMPEL	23.1	63D8F	REALcase	35.3	1EF7E	rpnDER	43
5E4EA	pullpshl&	26.3	0411B	REALEXT	17.1	1F1D4	rpnINTG	43
355B8	PULLREALEL	23.1	0408F	REALOB	17.1	1F354	rpnWHERE	43
5E4BD	pullrev	26.3	64E32	REALOBOB	17.1	1B185	rpnXROOT	43
0F3E4	puretemp?		265ED	realPACode		71DB2	RPTRACC	
08C27	PURGE	37.2	04099	REALREAL	17.1	639DE	'R'R	34
06537	PUSH#		040F3	REALSYM	17.1	639FC	'RRDROP	34
0357C	PUSH#ALoop		130AC	RECLAIMDISP	41.2	62474	'RSaveRomWrd	
2E31F	Push#FLoop		510D5	RECORDX&YC%		62474	'RSAVEWORD	
0357F	PUSH#LLoop		2F989	RecvNextPkt		63880	RSKIP	34
036F7	Push#TLoop		323F9	REMAP		15A60	rstfmt1	
2A188	PUSH%		071E5	REPEAT	36.1	2B7A7	RSUB1	
2A23D	PUSH%LOOP		40E88	REPEATER		60EBD	RSWAP	
06529	PUSH2#		51735	REPEATERCH		62A34	SAFE@	37.2
627EB	Push2#aLoop		047C7	REPKEY?	40.2	1853B	SAFE@_HERE	37.2
03F14	Push2#Loop		085D3	REPLACE	37.2	0A532	SAFESKIPOB	
03A86	PUSHA		53A20	REPLACE_MODE		62A02	SAFESTO	37.2
620C0	PushF/TLoop		629D0	REQcase		0000F	sALLOWINTR	
620DC	PushFLoop		629E9	REQcasedrop		15A8B	savefmt1	
5422C	PUSHhxs		4B710	RESETDEPTH		61D3A	SAVELAM	
0596D	PUSHhxsLoop		62BD8	RESOROMP		40C76	SaveLastEdit	
620D9	PushT/F		0C147	restoreiram		4139B	SaveLastMenu	
620D9	PushT/FLoop		640FA	RestVarRes		61D41	SAVESTACK	31.3
620C3	PushTLoop		06F66	'REVAL	34	640A0	SaveVarRes	
2C04B	PUTAB0		5DE7D	reversym	29	0679B	SAVPTR	
356F3	PUTCMPEL	23.1	41984	ReviewKey!		01307	SavPtrTime*	
35628	PUTEL	23.1	0C506	rGETATELN		00008	sBEG	
4AF77	PUTINDEP		1200C	RIGHT\$3x6	42.3	00004	sBPOFF	
4AF8B	PUTINDEPLIST		5160E	RIGHTCOL	41.6	4D16E	SCROLLDOWN	41.6
075E9	PUTLAM	31.3	2BEEC	RNDC[B]		4D150	SCROLLLEFT	41.6
1DC00	PUTLIST	25.4	2B529	RNDXY	18.3	4D18C	SCROLLRIGHT	41.6
6594E	putnibs		71C3B	rNTHELCOMP		4D132	SCROLLUP	41.6
4B076	PUTPTYPE		03325	ROLL	29	0403F	seco	17.1
3566F	PUTREALEL	23.1	42E27	Roll&Do:		3A2C9	Seco>Menu	42.4
4B012	PUTRES		63F42	roll2top&	0	0312B	SEMI	
4AE3C	PUTSCALE		62F89	ROLLDROP	29	61A47	SEMILOOP	
31444	PUTSERIAL		62D45	ROLLSWAP	29	2D5E1	SEND_PACKET	
30E4E	PutSerialEck		63F42	rolltwotop&	0	2FEA1	SENDACK	
4B1AC	PUTXMAX		64E14	ROMPANY		2D58C	SENDEOT	
4B166	PUTXMIN		07E99	ROMPTR@	28.2	2E0C7	SENDERORR	
4B1CF	PUTYMAX		08CCC	ROMPTR>#	28.2	2E6EB	SENDLIST	

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
2E0A9	SENDNAK		3A9E7	SetSomeRow		4019D	StdMenuKeyNS	
2FEB5	SENDNULLACK		18CC2	SETSTACKERR	32.2	31F4A	StdPRTPAR	
2E0F4	SENDPKT		53731	SetSysFlag	39.1	159EB	stkdecomp\$w	
2EEC4	SendSetup		410C6	SetThisRow		07D27	STO	37.2
127A7	SEP\$NL	20.4	423BB	settimeout		63975	STO'	34.1
07661	SET		18CB2	SETTYPEERR	32.2	0E6ED	STOALM	38
39FC1	SetAbbrevStk		53725	SetUserFlag	39.1	47467	STOAPPLDATA	
53976	SetAlgEntry	41.5	04035	SEVEN	17.1	64078	StoHiddenVar	37.4
1132D	SetAlphaAnn	41.5	6106B	SEVENROLL	29	2E9CB	StoIOPAR	
38D09	SetAppMode		04099	SEVENTEEN	17.1	07D1B	STOLAM	31.3
38D8A	SetAppSuspOK		64C16	SEVENTY	17.1	31F7D	StoPRTPAR	
141B2	setbeep	39.2	64C20	SEVENTYFOUR	17.1	0400D	str	17.1
11016	SETCIRCERR	32.2	64C2A	SEVENTYNINE	17.1	3A2B5	Str>Menu	42.4
4325A	SetCursor		00001	Sfkey1		64775	STRIPTAGS	22
391EE	SetDA123NoCh	41.3	00006	Sfkey6		647A2	STRIPTAGS12	22
3919E	SetDA12NoCh	41.3	00002	sFLUSH		64B6C	STRLIST	17.1
3921B	SetDA12Temp	41.3	0426A	ShowInvRomp	39.2	00001	sTRUNC	
391C6	SetDA13NoCh	41.3	16671	Shrink\$		05733	SUB\$	20.4
3947B	SetDA1Bad	41.3	64190	Sig?ErrJump		30805	SUB\$1#	20.4
39523	SetDA1IsStat	41.3	0402B	SIX	17.1	62D6D	SUB\$SWAP	20.4
393D3	SetDA1NoCh	41.3	61002	SIXROLL	29	05821	SUBCOMP	25.1
3902C	SetDA1Temp	41.3	0408F	SIXTEEN	17.1	1192F	SUBGROB	42.3
38FD2	SetDA1Valid	41.3	64BB2	SIXTY	17.1	05815	SUBHXS	21.2
391B2	SetDA23NoCh	41.3	64C02	SIXTYEIGHT	17.1	64345	SubMetaOb	26.4
394A5	SetDA2aBad	41.3	64BDA	SIXTYFOUR	17.1	643BD	SubMetaOb1	26.4
39086	SetDA2aEcho	41.3	64BBC	SIXTYONE	17.1	29BC2	subpdcdptch	
393FD	SetDA2aNoCh	41.3	64BD0	SIXTYTHREE	17.1	1446F	SuspendOK?	
39045	SetDA2aTemp	41.3	64BC6	SIXTYTWO	17.1	03223	SWAP	29
38FEB	SetDA2aValid	41.3	610FA	SIXUNROLL	29	63939	SWAP'	34.1
394CF	SetDA2bBad	41.3	0714D	SKIP	34.2	62794	SWAP#-	
39427	SetDA2bNoCh	41.3	626EE	skipcola	34.2	637E0	SWAP#1-	17.3
39059	SetDA2bTemp	41.3	03019	SKIPOB		62904	SWAP#1+	17.3
38FFF	SetDA2bValid	41.3	42131	SLEEPxcp		51843	SWAP#1+SWAP	17.3
3918A	SetDA2NoCh	41.3	40EE7	SLOW	38	51857	SWAP#1-SWAP	17.3
39207	SetDA2OKTemp	41.3	66EF1	SmallCursor	42.1	63BBE	SWAP%/%	18.3
3915D	SetDA2Valid	41.3	558F5	sncknum2	27.1	632A9	SWAP%>C%	19.2
394F9	SetDA3Bad	41.3	00002	sNEGATE		622EF	SWAP&\$	20.4
39451	SetDA3NoCh	41.3	151A6	SolvMenuInit		6386C	SWAP2DUP	29
39072	SetDA3Temp	41.3	48FF9	SORTASLOW		63C54	SWAP3PICK	29
39018	SetDA3Valid	41.3	65254	SPACE\$	20.2	63C7C	SWAP4PICK	29
39301	SetDA3ValidF	41.3	2BC4A	SPLITA		63C2C	SWAP4ROLL	29
3922F	SetDAsTemp	41.3	7DEF8	SPLITWHERE		63F7E	SWAPCKREF	37.1
2A5D2	SETDEG	39.1	2BCA0	SPLTAC		63312	SWAPCOLA	34.2
38D5D	SetDoStdKeys		2BA0F	SQRF		5A01D	SWAPcompSWAP	
3252B	SetEcma94		00118	SRQ1		60F9B	SWAPDROP	29
53B31	setflag		00119	SRQ2		62830	SWAPDROPDUP	29
2A604	SETGRAD	39.1	0131D	srvc_timer2		6284B	SWAPDROPSWAP	29
07638	SETHASH	0	007B5	SrvckbdAB		21660	SWAPDROPTRUE	33.1
640BE	SetHiddenRes	37.4	558DC	sscknum2	27.1	62747	SWAPDUP	29
2EC34	SetIOPARErr	32.2	14C17	sstDISP		631F5	SWAPINCOMP	25.3
539F9	SetISysFlag		2BE61	STAB0		63425	SWAPINDEX@	36.2
29DFC	SETIVLERR		2BE6F	STAB2		6344D	SWAPLOOP	36.2
3FCAF	SetKeysNS		1686A	stackitw		63AB0	SWAPONE	17.1
11361	SetLeftAnn	41.5	41008	StartMenu		61380	SWAPOVER	29
4CF41	SETLOOPENV		3858E	StartupProc		637A4	SWAPOVER#-	17.3
04FB6	SETMEMERR	32.2	2C22F	STATCLST	23.3	60F33	SWAPROT	29
0764E	SETMSG	0	2C571	STATMEAN	23.3	3745E	SWAPROWS	23.1
38D33	SetNAppKeyOK		2C535	STATN	23.3	4F1D8	SWAPTRUE	33.1
18C92	SETNONEXTERR	32.2	2C2D9	STATSADD%	23.3	638FD	SWAPUnDROP	29
3957A	SetNoRollDA2	41.3	2C558	STATSMAX	23.3	63911	SWAPUnNDROP	26.1
04FF2	SETPORTNOTAV	32.2	2C58A	STATSMIN	23.3	62904	SWP1+	17.3
11533	SetPrgmEntry	41.5	2C5A3	STATSTDEV	23.3	04053	sym	17.1
2A5F0	SETRAD	39.1	2C5BC	STATTOT	23.3	04049	symp	17.1
417F3	SetRebuild		2C5D5	STATVAR	23.3	659DE	Symb>HBuf	42.3
11347	SetRightAnn	41.5	2BE7D	STCD0		0546D	SYMBN	25.2
21CBA	SETROMPART		2BE8B	STCD2		5A310	sympn	
05016	SETROMPERR	32.2	3ED0C	Std/BoxLabel	42.4	32FF9	SYMBNUMSOLVE	
2D9A1	SetServMode		2E99E	StdIOPAR		64D38	SYMBREAL	17.1
18CA2	SETSIZEERR	32.2	401D4	StdMenuKeyLS		64D56	SYMBUNIT	17.1

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
547B5	SYMBWHERE	27.2	2856C	tok=casedrop		10047	U>nbr	24.2
57D90	SYMCOLCT	27.2	25446	tok->	20.2	0FE44	U>NCQ	24.2
5E652	symcomp	27.1	6534C	tok0	20.2	42249	UART?	
64DBA	SYMEXT	17.1	65358	tok1	20.2	3133B	UARTBUFLEN	
64D92	SYMID	17.1	653AC	tok8	20.2	42145	UARTxcp	
64D9C	SYMLAM	17.1	0BD54	tok8cktrior		42660	UCursor	
64D6A	SYMOB	17.1	0BD60	tok8trior		0F774	UM-	24.3
64D74	SYMREAL	17.1	653B8	tok9		0F598	UM#?	24.4
58D75	SYMSHOW	27.2	651BE	tokESC	20.2	0FBAB	UM%	24.3
64DB0	SYMSYM	17.1	651E2	tokexponent	20.2	0FC3C	UM%CH	24.3
10F86	SYNTAXERR	32.2	65278	tokquote	20.2	0FCCD	UM%T	24.3
2EA6A	Sys@		6518E	toksharp	20.2	0F792	UM*	24.3
08D92	SYSCONTEXT	37.3	63498	toLEN_DO	36.2	10B5E	um*	24.1
386A1	SysDisplay		5E415	top&	0	0F823	UM/	24.3
38728	SysErrorTrap		63F01	top&Cr		10B68	um/	24.1
63EED	SysITE	35.6	0E0AB	TOP16	41.6	10B72	um^	24.1
3866F	SysMenuCheck		0E0D3	TOP8	41.6	0F6A2	UM+	24.3
08D66	SysPtr@		4272D	TopERow?		0F5AC	UM<?	24.4
2E9E6	SysSTO		515A0	TOPROW	41.6	0F5D4	UM<=?	24.4
40792	SystemLevel?		07709	TOSRRP	0	0F584	UM=?	24.4
0EB81	SysTime	38	06657	TOTEMPOB	37.1	0F5C0	UM>?	24.4
64DEC	TAGGEDANY	17.1	62C69	TOTEMPSWAP	37.1	0F5E8	UM>=?	24.4
647BB	TAGOB5	22	2B53D	TRCXY	18.3	0F33A	UM>U	24.2
40788	TakeOver	14.5	2EFD7	TRPACKETFAIL		0F5FC	UMABS	24.3
00116	TBR		03A81	TRUE	33.1	0FD36	UMCEIL	24.3
00112	TCS		63989	TRUE'	34.1	0F615	UMCHS	24.3
0F41B	TempConv		634F7	TRUEFALSE	33.1	0F371	UMCONV	24.3
04053	TEN	17.1	634F7	TrueFalse	33.1	0F660	UMCOS	24.3
53784	TestSysFlag	39.1	0BBED	TrueTrue	33.1	10B86	umEND	24.1
53778	TestUserFlag	39.1	2BD76	TST15		0FD22	UMFLOOR	24.3
04071	THIRTEEN	17.1	4E2CF	TURNMENUOFF	41.2	0FD0E	UMFP	24.3
0411B	THIRTY	17.1	4E347	TURNMENUON	41.2	0FCFA	UMIP	24.3
0416B	THIRTYEIGHT	17.1	041A7	TurnOff		0FB6F	UMMAX	24.3
0414D	THIRTYFIVE	17.1	3A9CE	TurnOffKey		0FB8D	UMMIN	24.3
04143	THIRTYFOUR	17.1	04067	TWELVE	17.1	10B7C	umP	24.1
04175	THIRTYNINE	17.1	040B7	TWENTY	17.1	0FD68	UMRND	24.3
04125	THIRTYONE	17.1	04107	TWENTYEIGHT	17.1	0F945	UMSI	24.2
04161	THIRTYSEVEN	17.1	040E9	TWENTYFIVE	17.1	0FCE6	UMSIGN	24.3
04157	THIRTYSIX	17.1	040DF	TWENTYFOUR	17.1	0F62E	UMSIN	24.3
04139	THIRTYTHREE	17.1	04111	TWENTYNINE	17.1	0F913	UMSQ	24.3
0412F	THIRTYTWO	17.1	040C1	TWENTYONE	17.1	0F92C	UMSQRT	24.3
0400D	THREE	17.1	040FD	TWENTYSEVEN	17.1	0F674	UMTAN	24.3
631CD	THREE{ }N	25.4	040F3	TWENTYSIX	17.1	0FD8B	UMTRC	24.3
61B89	ticR	34	040D5	TWENTYTHREE	17.1	0F34E	UMU>	24.2
4227F	TIMEOUT?		040CB	TWENTYTWO	17.1	0F8FA	UMKROOT	24.3
00137	TIMER1		04003	TWO	17.1	10065	Unbr>U	24.2
00138	TIMER2		631B9	TWO{ }N	25.4	18DBF	UNCOERCE	18.2
0012E	TIMERCTRL.1		03C64	TYPE	33.3	63B96	UNCOERCE%%	18.2
0012F	TIMERCTRL.2		62198	TYPEARRY?	33.3	1950B	UNCOERCE2	18.2
0D304	TIMESTR	38	6212F	TYPEBINT?	33.3	5A036	uncrunch	27.1
1314D	TOADISP	41.1	62256	TYPECARRY?	33.3	61F8F	undo	31.3
0CBFA	TOD	38	62025	TYPECHAR?	33.3	538DC	UNDO_OFF	
0D06A	TOD>t\$	38	62183	TYPECMP?	33.3	538CE	UNDO_ON	
13135	TOGDISP	41.1	621EC	TYPECOL?	33.3	538C0	UNDO_ON?	
5072B	TOGGLELINE#3		62159	TYPECSTR?	33.3	0F218	UNIT>\$	24.2
3E586	TogInsertKey		6204F	TYPEEXT?	33.3	40D39	UnLockAlpha	41.5
50AF9	TOGLINE	42.3	62201	TYPEGROB?	33.3	0339E	UNROLL	29
50ADB	TOGLINE3	42.3	62144	TYPEHSTR?	33.3	60FAC	UNROT	29
65290	tok,	20.2	03FA9	TYPEIDNT	17.1	6112A	UNROT2DROP	29
65284	tok'	20.2	6203A	TYPEIDNT?	33.3	6284B	UNROTDROP	29
652FC	tok-	20.2	6211A	TYPELAM?	33.3	62CF5	UNROTDUP	29
6529C	tok.	20.2	62216	TYPELIST?	33.3	6308D	UNROTOVER	29
65254	tok_	20.2	6223B	TYPERARRY?	33.3	60F33	UNROTSWAP	29
0FA69	tok_g	20.2	03F8B	TYPEREAL	17.1	60F0E	UNROTSWAPDRO	29
0FA8E	tok_m	20.2	6216E	TYPEREAL?	33.3	071C8	UNTIL	36.1
0FACE	tok_s	20.2	621AD	TYPEROMP?	33.3	1A16F	UPDIR	37.3
65176	tok{	20.2	621C2	TYPERRP?	33.3	225F5	USER\$>TAG	24.1
651D6	tok<<	20.2	621D7	TYPESYMB?	33.3	39748	UserFlagStat	
65308	tok=	20.2	6222B	TYPETAGGED?	33.3	63ED9	UserITE	35.6

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
41A8D	UserKeys?		1F5C5	xAPPLY	43	231A0	xSTARTVAR	43
397BB	UserKeysStat		1D186	xCON	43	1C67F	xSTOF	43
6416D	UStackDepth		1EFD2	xDER	43	237A8	xTHENCASE	43
1A265	VARSIZE	39.2	63B6E	'xDER	34.1	1F3F3	xWHERE	43
0E66A	VerifyTOD		63A56	'xDEREQ		23694	xWHILEEND	43
30794	VERSTRING		236B9	xENDDO	43	1B1CA	xxROOT	43
40F02	VERYSLOW	38	23679	xENDTIC	43	60F9B	XY>Y	29
40F12	VERYVERYSLOW	38	1D054	XEQ>ARRAY	23.1	2BE53	XYEX	
42D46	ViewLevel1		21B5A	XEQIOBACKUP		128B0	XYGROBDISP	42.3
17B86	VLM		20FF2	XEQORDER	37.3	60F4B	XYZ>	29
4B553	VSCALE		18595	XEQPGDIR	37.3	62726	XYZ>Y	29
1165A	w->W		20F8A	XEQPURGEPICT		60EE7	XYZ>YXZ	29
1A738	Wait/GetKey	40.2	20B81	XEQRCL	37.2	60F21	XYZ>YZ	29
41F65	WaitForKey	40.2	21C6F	XEQSETLIB		6112A	XYZ>Z	29
01FBD	Warmstart		20B00	XEQSHOWLS	27.2	63326	XYZ>ZCOLA	34.2
071EE	WHILE	36.1	18513	XEQSTOID	37.2	62EB7	XYZ>ZTRUE	33.1
4F052	WINDOW#		40F22	XEQStoKey		6284B	XYZ>ZX	29
51645	WINDOWBOT?	41.6	1CB90	XEQTYPE	30.1	60FAC	XYZ>ZXY	29
137B6	WINDOWCORNER		2371F	xERRTHEN	43	60F0E	XYZ>ZY	29
13220	WINDOWDOWN	41.6	1BB02	xFACT	43	60F33	XYZ>ZYX	29
134E4	WINDOWLEFT	41.6	1F640	xFCNAPPLY	43	60F7E	XYZW>	29
5165E	WINDOWLEFT?	41.6	1FAEB	xFORMUNIT	43	6113C	XYZW>W	29
1357F	WINDOWRIGHT	41.6	213D1	xFREE	43	6109E	XYZW>WXYZ	29
51677	WINDOWRIGHT?	41.6	1E661	xFUNCTION	43	63C2C	XYZW>YWZX	29
5162C	WINDOWTOP?	41.6	64D24	XHI	17.1	60FBB	XYZW>YZWX	29
131C8	WINDOWUP	41.6	64D1A	XHI-1	17.1	2BD32	Y<=X	
13679	WINDOWXY		22FD5	xIFEND	43	64BDA	YHI	17.1
136AA	WindowXY		1A3FE	xIFTE	43	0DB3A	YMD>Ticks	
64037	WithHidden	37.4	1F223	xINTEGRAL	43	4E776	Z-BOX	
54039	WORDSIZE	39.1	20FAA	xMEM	43	03FEF	ZERO	17.1
1AD09	x-	43	1ACDD	xNEGNEG	43	073C3	ZERO_DO	36.2
1ADEE	x*	43	56101	xnsgeneral	27.1	6351F	ZEROFALSE	33.1
63B5A	'x*	34.1	1CF7B	xOBJ>	43	6400F	ZEROISTOPSTO	36.2
1AF05	x/	43	03ADA	XOR	33.1	63079	ZEROOVER	17.1
0931B	X@		1889B	XOR\$	20.4	62E3A	ZEROSWAP	17.1
1B02D	x^	43	1E6C1	xPARAMETRIC	43	641FC	ZEROZERO	17.1
1AB67	x+	43	1E6A1	xPOLAR	43	6431D	ZEROZEROONE	17.1
1EBBE	x<?	43	1D5DF	xPUTI	43	64331	ZEROZEROTWO	17.1
2361E	x<<	43	1D0DF	xRDM	43	64309	ZEROZEROZERO	17.1
1EC5D	x>?	43	2349C	xSILENT'	43			
23639	x>>	43	1C9B8	xSIZE	43			
235FE	x>>ABND	43	5611F	xssgeneral	27.1			
22FEB	xALG->	43	560ED	xssgeneral	27.1			

Chapter 45

Commands by address

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
00001	Sfkey1		00C0D	kermSENDmsg		02BCC	DOEXT2	16
00001	sTRUNC		00C0E	kermrecvmsg		02BEE	DOEXT3	16
00002	sFLUSH		00C10	kermPKTmsg		02C10	DOEXT4	16
00002	sNEGATE		00C13	prtparerr		02D9D	DOCOL	16
00003	DZP		00C74	OnKeyDown?		02DCC	DOCODE	16
00004	SBPOFF		00C80	OnKeyStable?		02E48	DOIDNT	16
00006	Sfkey6		00D48	addrKEYSTATE		02E6D	DOLAM	16
00008	SBEG		00D57	Flush		02E92	DOROMP	16
0000A	portnotaverr		00D71	FLUSH	40.2	03019	SKIPOB	
0000F	sALLOWINTR		00D71	FLUSHKEYS	40.2	030E0	~BRStoC1	
00019	ENTERCODE		00D8E	FlushAttn		0312B	SEMI	
000B1	~DoMsgBox		010E5	AllowIntr		0314C	DEPTH	29
000B3	~Choose		01115	DisableIntr		03188	DUP	29
00100	BITOFFSET		0115A	AINRTN		031AC	2DUP	29
00101	CONTRAST		01160	CINRTN		031D9	NDUP	29
00102	DTEST		012EE	GetTimChk		03223	SWAP	29
00103	DSPCTL		01307	SavPtrTime*		03244	DROP	29
00104	CRC		0130E	GetTime++		03249	DropLoop	
00108	LPD		0131D	srvc_timer2		03258	2DROP	29
00109	LPE		017A6	makebeep	39.2	0326E	NDROP	29
0010A	CMODE		0188D	addrORghost		03295	ROT	29
0010B	ANNCTRL		018E2	clkspd		032C2	OVER	29
0010D	BAU		01AD7	RCKBp		032E2	PICK	29
0010E	CARDCTL		01B8F	DispOn		03325	ROLL	29
0010F	CARDSTAT		01BBD	DispOff		0339E	UNROLL	29
00110	IOC		01C31	D0->Row1		03442	MAKEARRY	23.1
00111	RCS		01C58	D0->Sft1		03562	ARSize	23.1
00112	TCS		01F6D	CLCD10	41.4	0357C	PUSH#ALoop	
00113	CRER		01FA7	CLEARLCD	41.4	0357F	PUSH#LOOP	
00114	RBR		01FBD	norecPWLseq		035A9	DIMLIMITS	23.1
00116	TBR		01FBD	Warmstart		0366F	GPOverWrR0Lp	
00118	SRQ1		01FD3	Coldstart		03672	GPOverWrALp	
00119	SRQ2		01FDA	norecCSseq		036F7	Push#TLoop	
0011A	IRC		020B1	~MsgBoxMenu		0371D	GETATELN	
0011C	LCR		028FC	PRLG		03991	MUL#	
0011D	LBR		02911	DOBINT	16	039BE	GETTEMP	
0011F	IRAM@		02933	DOREAL	16	039EF	ECUSER	
00120	DISP1CTL		02955	DOEREL	16	03A81	TRUE	33.1
00125	LINENIBS		02977	DOCMP	16	03A86	PUSHA	
00128	LINECOUNT		0299D	DOECMP	16	03AC0	FALSE	33.1
0012E	TIMERCTRL.1		029BF	DOCHAR	16	03ADA	XOR	33.1
0012F	TIMERCTRL.2		029E8	DOARRY	16	03AF2	NOT	33.1
00130	DISP2CTL		02A0A	DOLNKARRY	16	03B2E	EQ	33.2
00137	TIMER1		02A2C	DOCSTR	16	03B46	AND	33.1
00138	TIMER2		02A4E	DOHSTR	16	03B75	OR	33.1
001FF	allkeys		02A4E	DOHXS	16	03B97	EQUAL	33.2
00202	argtypeerr		02A74	DOLIST	16	03C64	TYPE	33.3
00203	argvalerr		02A96	DORRP	16	03CA6	#0=	17.4
00301	posunferr		02AB8	DOSYMB	16	03CC7	#0<>	17.4
00302	negunferr		02ADA	DOEXT	16	03CE4	#<	17.4
00303	ofloerr		02AFC	DOTAG	16	03D19	#=	17.4
00305	infreserr		02B1E	DOGROB	16	03D4E	#<>	17.3
007B5	SrvckbdAB		02B40	DOLIB	16	03D83	#>	17.4
008E6	BITMAP		02B62	DOBAK	16	03DBC	#+	17.3
009A5	Debounce		02B88	DOEXT0	16	03DC7	#PUSHA-	
00A03	intrptderr		02BAA	DOACPTR	16	03DE0	#-	17.3
00B02	constuniterr		02BAA	DOEXT1	16	03DEF	#1+	17.3

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
03E0E	#1-	17.3	0417F	FOURTY	17.1	05815	SUBHXS	21.2
03E2D	#2+	17.3	04189	FORTYONE	17.1	05821	SUBCOMP	25.1
03E4E	#2-	17.3	04193	FORTYTWO	17.1	05902	OSIZE	39.2
03E6F	#2*	17.3	0419D	FORTYTHREE	17.1	05944	OCRC	39.2
03E8E	#2/	17.3	041A7	TurnOff		0596D	PUSHhxsLoop	
03EB1	#AND	17.3	041ED	DEEPSLEEP	39.2	0597E	DoCRCC	
03EC2	#*	17.3	0426A	ShowInvRomp	39.2	05981	DoCRC	
03EF7	#/	17.3	04292	DeepSleep		059CC	#>HXS	21.1
03F14	Push2#Loop		04708	CHECKKEY		05A03	HXS>#	17.2
03F24	IntDiv		04714	GETTOUCH		05A51	CHR>#	17.2
03F5D	POP2#		047C7	REPKEY?	40.2	05A75	#>CHR	20.3
03F8B	TYPEREAL	17.1	047CF	adrDISABLE_K		05AB3	CHANGETYPE	39.2
03FA9	TYPEIDNT	17.1	047DD	adrKEYBUFFER		05B15	\$>ID	31.2
03FEF	ZERO	17.1	04840	POPKEY	40.2	05B79	MAKE\$	
03FF9	MEMERR	17.1	04929	liteslp		05B7D	MAKE\$N	
03FF9	ONE	17.1	04999	KeyInBuff?		05BE9	ID>\$	20.3
03FF9	real	17.1	04A0B	GETPROC		05C27	%>C%	19.2
04003	TWO	17.1	04A41	GETDF		05D2C	C%>%	19.2
0400D	str	17.1	04CE6	ERROR@	32.1	05DBC	C%>%%	19.2
0400D	THREE	17.1	04D0E	ERRORSTO	32.1	05E81	>TAG	22
04017	FOUR	17.1	04D33	ERRORCLR	32.1	05F2E	ID>TAG	22
04021	FIVE	17.1	04D3E	DROPNULL\$	20.2	05F42	GARBAGE	37.1
04021	list	17.1	04D64	GETTHEMSG	32.1	05F61	MEM	39.2
0402B	id	17.1	04D87	JstGETTHEMSG	32.1	060B0	~IFMenuRow2	
0402B	idnt	17.1	04E07	GETEXITMSG	32.1	060B3	~ChooseMenu1	
0402B	SIX	17.1	04E37	EXITMSGSTO	32.1	0613E	GARBAGECOL	
04035	SEVEN	17.1	04E5E	ERRSET	32.1	06529	PUSH2#	
0403F	EIGHT	17.1	04E66	addrTEMPENV		06537	PUSH#	
0403F	seco	17.1	04EA4	ABORT	32.1	065AA	GPMEMERR	
04049	NINE	17.1	04EB8	ERRTRAP	32.1	06641	POP#	
04049	symb	17.1	04ED1	ERRJMP	32.1	06657	TOTEMPOB	37.1
04053	sym	17.1	04FB6	SETMEMERR	32.2	066B9	MOVEUP	
04053	TEN	17.1	04FF2	SETPORTNOTAV	32.2	0670C	MOVEDOWN	
0405D	ELEVEN	17.1	05016	SETROMPERR	32.2	0679B	SAVPTR	
04067	TWELVE	17.1	05023	Err jmp		067D2	GETPTR	
04071	THIRTEEN	17.1	05040	ATTNFLG@	40.2	06806	ROOM	
0407B	EXT	17.1	05068	ATTNFLGCLR	40.2	06992	MOVERSD	
0407B	FOURTEEN	17.1	05089	CARCOMP	25.1	069C5	MOVEDSU	
04085	FIFTEEN	17.1	050B0	~IFMenuRow1		069F7	ADJMEM	
0408F	REALOB	17.1	050B3	~ChooseMenu0		06A1D	MOVEDSD	
0408F	SIXTEEN	17.1	050ED	CAR\$	20.4	06A53	MOVERSU	
04099	2REAL	17.1	05143	GETPTRLOOP		06A8E	DIV5	
04099	REALREAL	17.1	05153	CDRCOMP	25.1	06AD8	CREATETEMP	
04099	SEVENTEEN	17.1	0516C	CDR\$	20.4	06B4E	INTENNOTREF?	37.1
040A3	EIGHTEEN	17.1	0518A	&HXS	21.2	06E8E	NOP	34
040AD	NINETEEN	17.1	05193	&\$	20.4	06E97	'	34.1
040B7	TWENTY	17.1	0521F	&COMP	25.1	06EEB	'R	34
040C1	TWENTYONE	17.1	0525B	>H\$	20.4	06F66	'REVAL	34
040CB	TWENTYTWO	17.1	052C6	>HCOMP	25.1	06F8E	EVAL	34
040D5	TWENTYTHREE	17.1	052EE	>T\$	20.4	06F9F	>R	34
040DF	TWENTYFOUR	17.1	052FA	>TCOMP	25.1	06FB7	RDROP	34
040E9	TWENTYFIVE	17.1	05445	: :N	25.5	06FD1	COLA	34.2
040F3	REALSYM	17.1	05459	{ }N	25.4	07012	R@	34
040F3	TWENTYSIX	17.1	0546D	SYMBN	25.2	0701F	R>	34
040FD	TWENTYSEVEN	17.1	05481	EXTN	25.2	070B3	~ChooseMenu2	
04107	TWENTYEIGHT	17.1	054AF	INNERCOMP	25.3	070C3	RPITE	35.1
04111	TWENTYNINE	17.1	0554C	DOGARBAGE		070FD	RPIT	35.1
0411B	REALEXT	17.1	0556F	NULL\$?	20.5	0712A	?SKIP	35.1
0411B	THIRTY	17.1	055B7	NULLCOMP?	25.1	0712A	NOT_IT	35.1
04125	THIRTYONE	17.1	055D5	NULLHXS	21.2	0714D	SKIP	34.2
0412F	THIRTYTWO	17.1	055DF	NULL\$	20.2	0716B	IDUP	34
04139	THIRTYTHREE	17.1	055E9	NULL{ }	25.4	071A2	BEGIN	36.1
04143	THIRTYFOUR	17.1	055FD	NULL: :	25.5	071AB	AGAIN	36.1
0414D	THIRTYFIVE	17.1	05616	LENHXS	21.1	071C8	UNTIL	36.1
04157	THIRTYSIX	17.1	05622	OVERLEN\$	20.4	071E5	REPEAT	36.1
04161	THIRTYSEVEN	17.1	05636	LEN\$	20.4	071EE	WHILE	36.1
0416B	THIRTYEIGHT	17.1	0567B	LENCOMP	25.1	07221	INDEX@	36.2
04175	THIRTYNINE	17.1	056B6	NTHELCOMP	25.1	07249	ISTOP@	36.2
0417F	FORTY	17.1	05733	SUB\$	20.4	07258	JINDEX@	36.2

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
07264	JSTOP@	36.2	0D607	CSRW5		0FD8B	UMTRC	24.3
07270	INDEXSTO	36.2	0D618	CSLW5		0FE44	U>NCQ	24.2
07295	ISTOPSTO	36.2	0D62F	DCHXW		10047	U>nbr	24.2
072AD	JINDEXSTO	36.2	0D6D8	FLOAT		10065	Unbr>U	24.2
072C2	JSTOPSTO	36.2	0D744	Day>Date		100E0	~BRbrowse	
07334	LOOP	36.2	0D7A1	CLKUTL1		10B5E	um*	24.1
073A5	+LOOP	36.2	0D809	getBPOFF		10B68	um/	24.1
073C3	ZERO_DO	36.2	0D8AE	mpopl%		10B72	um^	24.1
073CE	ONE_DO	36.2	0D92C	POPDATe%		10B7C	umP	24.1
073DB	#1+_ONE_DO	36.2	0D948	POPTIME%		10B86	umEND	24.1
073F7	DO	36.2	0D9C7	CKTIME		10E68	cFF	18.1
07497	ABND	31.3	0DB3A	YMD>Ticks		10E82	cFC	18.1
074D0	BIND	31.3	0DB51	dowutil		10F40	GPErrjmpC	
074E4	DOBIND		0DB91	HXDCW		10F80	ErrjmpC	
075A5	GETLAM	31.3	0E06F	Clr16	41.4	10F86	SYNTAXERR	32.2
075E9	PUTLAM	31.3	0E083	Clr8	41.4	10FC6	NOHALTERR	32.2
07638	SETHASH	0	0E097	Clr8-15	41.4	11016	SETCIRCERR	32.2
0764E	SETMESG	0	0E0AB	TOP16	41.6	110E7	~UTVUNSlArg	
07661	SET		0E0D3	TOP8	41.6	1132D	SetAlphaAnn	41.5
076AE	OFFSRRP	0	0E0FB	Rows8-15	41.6	1133A	ClrAlphaAnn	41.5
07709	TOSRRP	0	0E128	HBUFF_X_Y	41.1	11347	SetRightAnn	41.5
07943	@LAM	31.3	0E235	ALARMS@		11354	ClrRightAnn	41.5
0797B	@	37.2	0E66A	VerifyTOD		11361	SetLeftAnn	41.5
07D1B	STOLAM	31.3	0E6ED	STOALM	38	1136E	ClrLeftAnn	41.5
07D27	STO	37.2	0E7D3	addrClkOnNib		11511	PrgmEntry?	41.5
07E50	#>ROMPTR	28.2	0EB81	CLKTICKS	38	11533	SetPrgmEntry	41.5
07E99	ROMPTR@	28.2	0EB81	SysTime	38	1158F	MAKEGROB	42.3
081D9	BAKNAME	28.4	0EBD5	FindNext		115B3	makegrob	
082E3	RAM-WORDNAME	37.3	0EE50	Date>hxsl3	38	1165A	w->W	
08326	LASTRAM-WORD	37.3	0F218	UNIT>\$	24.2	11679	GROB!	42.3
08376	PREVRAM-WORD	37.3	0F33A	UM>U	24.2	1192F	SUBGROB	42.3
085D3	REPLACE	37.2	0F34E	UMU>	24.2	11A6D	GROB! ZERO	42.3
08696	CREATE	37.2	0F371	UMCONV	24.3	11CF3	\$>BIGGROB	42.3
08C27	PURGE	37.2	0F3E4	puretemp?		11D00	\$>GROB	42.3
08CCC	ROMPTR>#	28.2	0F41B	TempConv		11D8F	\$5x7	41.7
08D08	CONTEXT!	37.3	0F584	UM=?	24.4	11F80	\$>grob	42.3
08D5A	CONTEXT@	37.3	0F598	UM#?	24.4	1200C	RIGHT\$3x6	42.3
08D66	SysPtr@		0F5AC	UM<?	24.4	120E0	~BRoutput	
08D92	HOMEDIR	37.3	0F5C0	UM>?	24.4	120E4	~MESRclEqn	
08D92	SYSCONTEXT	37.3	0F5D4	UM<=?	24.4	1215E	CENTER\$3x5	42.3
090B1	~DoMKeyOK		0F5E8	UM>=?	24.4	122FF	INVGROB	42.3
0931B	X@		0F5FC	UMABS	24.3	123C8	BIGDISPN	41.7
0948E	BAK>OB	28.4	0F615	UMCHS	24.3	123E5	BIGDISPROW4	41.7
0970A	InitEnab		0F62E	UMSIN	24.3	123F5	BIGDISPROW3	41.7
09B73	COMPCONFRCRC		0F660	UMCOS	24.3	12405	BIGDISPROW2	41.7
0A00E	CKLBRCRC		0F674	UMTAN	24.3	12415	BIGDISPROW1	41.7
0A532	SAFESKIPOB		0F688	%2PI	18.1	12429	DISPN	
0AAB2	PORTSTATUS	28.1	0F6A2	UM+	24.3	1245B	DISP@01	41.7
0AB82	NEXTLIBBAK	28.1	0F774	UM-	24.3	1245B	DISPROW1	41.7
0BBED	TrueTrue	33.1	0F792	UM*	24.3	1246B	DISP@09	41.7
0BC01	failed	33.1	0F823	UM/	24.3	1246B	DISPROW2	41.7
0BC6F	!*trior		0F8FA	UMXROOT	24.3	1247B	DISP@17	41.7
0BCCF	!*triand		0F913	UMSQ	24.3	1247B	DISPROW3	41.7
0BD54	tok8cktrior		0F92C	UMSQRT	24.3	1248B	DISP@25	41.7
0BD60	tok8trior		0F945	UMSI	24.2	1248B	DISPROW4	41.7
0C147	restoreiram		0FA69	tok_g	20.2	1249B	DISPROW5	41.7
0C506	rGETATELN		0FA8E	tok_m	20.2	124AB	DISPROW6	41.7
0CA60	ATTNchk		0FACE	tok_s	20.2	124BB	DISPROW7	41.7
0CBC4	ErrTime		0FB6F	UMMAX	24.3	124CB	DISPROW8	41.7
0CBFA	TOD	38	0FB8D	UMMIN	24.3	12635	HARDBUFF	41.1
0CC0E	DATE	38	0FBAB	UM%	24.3	1263A	addrVDISP	
0CC39	DDAYS	38	0FC3C	UM%CH	24.3	12645	HARDBUFF2	41.1
0CC5B	DATE+DAYS	38	0FCCD	UM%T	24.3	1264A	addrVDISP2	
0CFD9	Date>d\$	38	0FCE6	UMSIGN	24.3	12655	ABUFF	41.1
0D06A	TOD>t\$	38	0FCFA	UMIP	24.3	1265A	addrADISP	
0D304	TIMESTR	38	0FD0E	UMFP	24.3	12665	GBUFF	41.1
0D4AD	DAY#		0FD22	UMFLOOR	24.3	126DF	BLANKIT	41.4
0D5E5	ASRW5		0FD36	UMCEIL	24.3	1270C	DISPSTATUS2	41.7
0D5F6	ASLW5		0FD68	UMRND	24.3	12725	DISPROW1*	41.7

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
12748	DISPROW2*	41.7	15A60	rstfmt1		18EDF	CK2&Dispatch	30
12770	COERCE\$22	20.4	15A8B	savefmt1		18EF0	CK3&Dispatch	30
127A7	SEP\$NL	20.4	15B13	DECOMP\$		18F01	CK4&Dispatch	30
128B0	XYGROBDISP	42.3	15B31	editdecomp\$w		18F12	CK5&Dispatch	30
12B6C	HARDHEIGHT	41.1	15E83	longhxs		18F23	EvalNoCK	30
12B85	FlashMsg	41.7	1605F	addtics		18F9D	CK&DISPATCH0	30
12DD1	HEIGHTENGROB	41.2	1613F	NULL\$TEMP	20.2	18FA9	CK&DISPATCH2	30
12F94	GROB>GDISP	42.3	162AC	a%>\$,	20.3	18FB2	CK&DISPATCH1	30
13061	KILLGDISP	41.2	162B8	a%>\$	20.3	191B9	*OVF	17.3
130AC	RECLAIMDISP	41.2	16671	Shrink\$		194F7	COERCE2	17.1
130E0	~BRdone		166E3	DOFIX	39.1	1950B	UNCOERCE2	18.2
13135	TOGDISP	41.1	166EF	DOSCI	39.1	199EB	DoInputForm	13
1314D	TOADISP	41.1	166FB	DOENG	39.1	1A16F	UPDIR	37.3
131C8	WINDOWUP	41.6	16707	DOSTD	39.1	1A1FC	OCRC%	39.2
13220	WINDOWDOWN	41.6	167BF	DPRADIX?	39.1	1A265	VARSIZE	39.2
133AB	disprange		167D8	#:>\$	20.3	1A2DA	INHARDROM?	39.2
134AE	CLEARVDISP	41.4	167E4	#>\$	20.3	1A3FE	xIFTE	43
134E4	WINDOWLEFT	41.6	1686A	stackitw		1A738	Wait/GetKey	40.2
1357F	WINDOWRIGHT	41.6	169A5	NDROPFALSE	33.1	1A7C9	dowait	38
13679	WINDOWXY		170E0	~BRRclCurRow		1AB67	x+	43
136AA	WindowXY		1795A	!DcompWidth		1ACDD	xNEGNEG	43
136AC	addrLINECNTg		17980	DcompWidth@		1AD09	x-	43
137B6	WINDOWCORNER		179D0	NEXTRRPOB		1ADEE	x*	43
137DC	corner		179E8	addrTEMPTOP		1AF05	x/	43
1380F	PIXOFF3	42.3	17ADB	FixRRP		1B02D	x^	43
13825	PIXON3	42.3	17B86	VLM		1B185	rpnXROOT	43
1383B	PIXOFF	42.3	180E0	~BRRclC1		1B1CA	xXROOT	43
1384A	PIXON	42.3	18308	DropSysObs		1BB02	xFACT	43
13986	PIXON?3	42.3	1848C	PATHDIR	37.3	1BB41	preFACT	43
13992	PIXON?	42.3	184E1	CREATEDIR	37.3	1C637	RCLSYSF	39.1
13B51	DISPCHAR+PC		18513	?STO_HERE	37.2	1C64E	RCLUSERF	39.1
13D28	cursorblink+		18513	XEQTSTOID	37.2	1C67F	xSTOF	43
13D55	cursorblink-		1853B	SAFE@_HERE	37.2	1C6A2	DOSTOALLF	39.1
13D8C	CURSOR1	42.1	1854F	?PURGE_HERE	37.2	1C6E3	DOSTOSYSF	39.1
13DB4	CURSOR2	42.1	18595	XEQPGDIR	37.3	1C9B8	xSIZE	43
13E85	CURSOR_OFF		185C7	DoHere:		1CB90	XEQTYPE	30.1
14088	DO>STR	20.3	18621	LastNonNull		1CC03	%11	18.1
140AB	DODISP		1863A	PrevNonNull	37.3	1CC1D	%12	18.1
140F1	DOCHR	20.3	186E8	DOTVARS%		1CC37	%13	18.1
14137	DOSTR>	20.3	18779	DOVARS	37.3	1CC51	%14	18.1
1415A	DOBEEP	39.2	1884D	0LASTOWDOB!	30	1CC6B	%20	18.1
141B2	setbeep	39.2	1884D	0LastRomWrd!	30	1CC85	%15	18.1
141E5	ERRBEEP	32.1	18873	AND\$	20.4	1CCA4	%21	18.1
142FB	Ck&Freeze		18887	OR\$	20.4	1CCC3	%22	18.1
1446F	SuspendOK?		1889B	XOR\$	20.4	1CCE2	%23	18.1
14483	nohalt		18A15	CK0NOLASTWD	30	1CD01	%24	18.1
14C17	sstDISP		18A1E	CK0	30	1CD20	%25	18.1
14EA5	RDUP	34	18A5B	CK3	30	1CD3A	%16	18.1
15007	DO%EXIT	32.1	18A68	CK3NOLASTWD	30	1CD54	%17	18.1
1502F	DO#EXIT	32.1	18A80	CK2	30	1CD73	%26	18.1
15048	DO\$EXIT	32.1	18A8D	CK2NOLASTWD	30	1CD8D	%27	18.1
1518D	GsstFIN		18AA5	CK1	30	1CDF2	%18	18.1
151A6	SolvMenuInit		18AB2	CK1NOLASTWD	30	1CE07	%19	18.1
152FF	EqList?		18B6D	CK5	30	1CF7B	xOBJ>	43
153FC	DispVarsUtil		18B7A	CK5NOLASTWD	30	1D054	XEQ>ARRAY	23.1
1553B	KeepUnit		18B92	CK4	30	1D0DF	xRDM	43
1568F	ALGeq?		18B9F	CK4NOLASTWD	30	1D186	xCON	43
15717	DOSTOE		18C34	CKN	30	1D5DF	xPUTI	43
1572B	DORCLE		18C4A	CKNNOLASTWD	30	1DABB	#1+ROT	17.3
15744	EQUATION		18C92	SETNONEXTERR	32.2	1DC00	PUTLIST	25.4
15777	NULLID	31.1	18CA2	SETSIZEERR	32.2	1E661	xFUNCTION	43
1578D	CRUNCH	27.1	18CA7	DOSIZEERR	32.2	1E6A1	xPOLAR	43
1583C	EVALCRUNCH		18CB2	SETTYPEERR	32.2	1E6C1	xPARAMETRIC	43
1592D	CK1NoBlame	30	18CC2	SETSTACKERR	32.2	1EBBE	x<?	43
15941	CRUNCHNoBlam		18CD7	%ABSCOERCE	18.3	1EC5D	x>?	43
15978	1stkdecomp\$w		18CEA	COERCE	17.1	1EF7E	rpnDER	43
159EB	stkdecomp\$w		18DBF	UNCOERCE	18.2	1EFD2	xDER	43
15A0E	EDITDECOMP\$	20.3	18EBA	COMPEVAL	34	1F05B	CKSYMBTYPE	30.1
15A40	ederr		18ECE	CK1&Dispatch	30	1F1D4	rpnINTG	43

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
1F223	xINTEGRAL	43	29E21	PACKSB		2A87F	%%>	18.4
1F354	rpnWHERE	43	29E46	PACK		2A88A	%>	18.4
1F3F3	xWHERE	43	29FD0	POP1%SPLITA		2A895	%%>=	18.4
1F55D	rpnAPPLY	43	29FDA	POP1%		2A8A0	%>=	18.4
1F5C5	xAPPLY	43	2A002	POP2%		2A8AB	%%<=	18.4
1F640	xFCNAPPLY	43	2A0B3	~DoCKeyCheck		2A8B6	%<=	18.4
1F996	COMPLEXDUMMY		2A188	PUSH%		2A8C1	%=	18.4
1F9AE	POLARDUMMY		2A23D	PUSH%LOOP		2A8CC	%<>	18.4
1FAEB	xFORMUNIT	43	2A2B4	%0	18.1	2A8D7	%SGN	18.3
20B00	XEQSHOWLS	27.2	2A2C9	%1	18.1	2A8F0	%%ABS	18.3
20B81	XEQRCL	37.2	2A2DE	%2	18.1	2A900	%ABS	18.3
20B9A	LISTRCL	37.2	2A2F3	%3	18.1	2A910	%%CHS	18.3
20CAD	PICTRCL		2A308	%4	18.1	2A920	%CHS	18.3
20F8A	XEQPURGEPICT		2A31D	%5	18.1	2A930	%MANTISSA	18.3
20FAA	xMEM	43	2A332	%6	18.1	2A943	%%+	18.3
20FF2	XEQORDER	37.3	2A347	%7	18.1	2A94F	%%-	18.3
213D1	xFREE	43	2A35C	%8	18.1	2A95B	%>%-	18.3
21660	SWAPDROPTTRUE	33.1	2A371	%9	18.1	2A974	%+	18.3
216D8	OB>BAKcode		2A386	%-1	18.1	2A981	%-	18.3
21922	GetBVars		2A39B	%-2	18.1	2A99A	%%*	18.3
21B5A	XEQIOBACKUP		2A3B0	%-3	18.1	2A9BC	%*	18.3
21C6F	XEQSETLIB		2A3C5	%-4	18.1	2A9C9	%OF	18.3
21CBA	SETROMPART		2A3DA	%-5	18.1	2A9E8	%%/	18.3
225F5	USER\$>TAG	24.1	2A3EF	%-6	18.1	2A9FE	%/	18.3
22618	%>TAG	22	2A404	%-7	18.1	2AA0B	%T	18.3
22FD5	xIFEND	43	2A419	%-8	18.1	2AA30	%CH	18.3
22FEB	xALG->	43	2A42E	%-9	18.1	2AA5F	%%^	18.3
231A0	xSTARTVAR	43	2A443	%PI	18.1	2AA70	%^	18.3
2349C	xSILENT'	43	2A472	%MAXREAL	18.1	2AA81	%NROOT	18.3
234C1	RPN->	43	2A487	%-MAXREAL	18.1	2AA92	%%1/	18.3
235FE	x>>ABND	43	2A49C	%MINREAL	18.1	2AA9E	%>%1/	18.3
2361E	x<<	43	2A4B1	%-MINREAL	18.1	2AAAF	%1/	18.3
23639	x>>	43	2A4C6	%%0	18.1	2AAEA	%%SQRT	18.3
23679	xENDTIC	43	2A4E0	%%1	18.1	2AAF6	%>%SQRT	18.3
23694	xWHILEEND	43	2A4FA	%%2	18.1	2AB09	%SQRT	18.3
236B9	xENDDO	43	2A514	%%3	18.1	2AB1C	%%EXP	18.3
2371F	xERRTHEN	43	2A52E	%%4	18.1	2AB2F	%EXP	18.3
23768	CK0ATTNABORT		2A548	%%5	18.1	2AB42	%EXPM1	18.3
237A8	xTHENCASE	43	2A562	%%.1	18.1	2AB5B	%%LN	18.3
238A4	palparse	20.3	2A57C	%%.5	18.1	2AB6E	%LN	18.3
23EED	ONE{ }N	25.4	2A596	%%10	18.1	2AB81	%LOG	18.3
24C0D	List		2A5B0	%%>%	18.4	2AB94	%LNP1	18.3
24EA6	{ }>DIR		2A5C1	%>%%	18.2	2ABA7	%LNP1	18.3
2512D	delimcase		2A5D2	SETDEG	39.1	2ABBA	%ALOG	18.3
2520A	'Rapndit		2A5F0	SETRAD	39.1	2ABDC	%MOD	18.3
25223	DaDGNTc		2A604	SETGRAD	39.1	2ABEF	%SIN	18.3
2534A	nultrior		2A622	%D>R	18.3	2AC06	%%SIN	18.3
25446	tok->	20.2	2A62C	PI/180	18.1	2AC17	%%SINDEG	18.3
25452	getmatchtok		2A655	%R>D	18.3	2AC27	%%SINRAD	18.3
255BD	ngsizecase		2A673	%>HMS	38	2AC40	%COS	18.3
255FB	need'case		2A68C	%HMS>	38	2AC57	%%COS	18.3
25632	newBASE		2A6A0	%HMS+	38	2AC68	%%COSDEG	18.3
256E4	convertbase		2A6C8	%HMS-	38	2AC78	%%COSRAD	18.3
25C41	Extobcode		2A6DC	%%MAX	18.3	2AC91	%TAN	18.3
25CE1	GETPTRTRUE		2A6F5	%MAX	18.3	2ACA8	%%TANRAD	18.3
26162	GetNextToken		2A70E	%MIN	18.3	2ACC1	%ASIN	18.3
265ED	realPacode		2A727	%0<	18.4	2ACD8	%%ASINRAD	18.3
26A2D	?OKINALG		2A738	%0<	18.4	2ACF1	%ACOS	18.3
26FAE	GETPTRFALSE		2A75A	%%0=	18.4	2AD08	%%ACOSRAD	18.3
27244	NOTLISTcase	35.5	2A76B	%0=	18.4	2AD21	%ATAN	18.3
27254	NOTSECOcase	35.5	2A788	%%0>	18.4	2AD38	%ANGLE	18.3
27264	NOTROMPcase	35.5	2A799	%0>	18.4	2AD4F	%%ANGLE	18.3
27D00	check_pdata		2A7BB	%%0<>	18.4	2AD5B	%>%ANGLE	18.3
28296	metatail	26.4	2A7CF	%0<>	18.4	2AD6C	%%ANGLEDEG	18.3
28558	#1-UNROT	17.3	2A7E3	%%0>=	18.4	2AD7C	%%ANGLERAD	18.3
2856C	tok=casedrop		2A7F7	%0>=	18.4	2AD95	%%SINH	18.3
29A8D	nextsym'R		2A80B	%%0<=	18.4	2ADAE	%SINH	18.3
29BC2	subpdcdptch		2A81F	%%<	18.4	2ADC7	%%COSH	18.3
29DFC	SETIVLERR		2A871	%<	18.4	2ADDA	%COSH	18.3

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
2ADED	%TANH	18.3	2C535	STATN	23.3	2FEC9	KVISLFL	
2AE00	%ASINH	18.3	2C558	STATSMAX	23.3	2FEDD	KVIS	
2AE13	%ACOSH	18.3	2C571	STATMEAN	23.3	2FFB4	GetStrLenStk	
2AE26	%ATANH	18.3	2C58A	STATSMIN	23.3	2FFB7	GetStrLenC	
2AE39	%EXPONENT	18.3	2C5A3	STATSTDEV	23.3	2FFBA	GetStrLen	
2AE4C	%NFACT	18.3	2C5BC	STATTOT	23.3	300B3	~LEDispPromp	
2AE62	%COMB	18.3	2C5D5	STATVAR	23.3	3016B	KINVISLFL	
2AE75	%PERM	18.3	2D0B3	~DoCKeyCance		30794	VERSTRING	
2AF27	%%H>HMS	38	2D396	MAXRETRY		307E2	GETKP	
2AF4D	%FP	18.3	2D3A0	LAMPKNO		30805	SUB\$1#	20.4
2AF60	%IP	18.3	2D3B1	LAMPACKET		30914	ACK_INIT	
2AF73	%CEIL	18.3	2D3C6	LAMRETRY		30B1D	CHOOSE_INIT	
2AF86	%FLOOR	18.3	2D3EE	LAMKP		30BBE	ENCODelPKT	
2AF99	%%FLOOR	18.3	2D3FB	LAMLNAME		30BD7	ENCODE	
2AF99	%%INT	18.3	2D40E	LAMOBJ		30D31	DECODE	
2AFC2	%RAN	18.3	2D41D	LAMOPOS		30E4E	PutSerialEck	
2B044	%RANDOMIZE	18.3	2D441	'LamKPSto		30ED2	OUTUART	
2B07B	DORANDOMIZE	18.3	2D45A	LAMKLIST		31289	POPUART	
2B0B3	~DoCKeyChAll		2D46D	LAMKMODE		312DA	adr_uart_han	
2B0C4	%FACT	18.3	2D493	LAMKRM		3133B	UARTBUFLen	
2B1FF	%%7	18.1	2D517	EXCHINITPK		3136C	FLUSHRSBUF	
2B2DC	%%12	18.1	2D564	Loop		31444	PUTSERIAL	
2B300	%%60	18.1	2D58C	SENDEOT		314E5	GETSERIAL	
2B3DD	%%.4	18.1	2D5E1	SEND_PACKET		315C6	CLOSEUART	
2B45C	2%>%%	18.2	2D730	KDispRow2		315F9	CloseUart	
2B470	2%>%%	18.2	2D74E	KDispStatus2		3161E	OpenUartClr	
2B48E	%REC>%POL	18.3	2D9A1	SetServMode		31854	docr	
2B498	%%R>P	18.3	2D9B2	ClrServMode		31868	DOCR	
2B4BB	%POL>%REC	18.3	2DDC4	DropSysErr\$		3187C	OpenIOPrnt	
2B4C5	%%P>R	18.3	2E0A9	SENDNAK		31EE2	DOPRLCD	
2B4F2	%SPH>%REC	18.3	2E0B3	~DoCKeyOK		31F4A	StdPRTPAR	
2B529	RNDXY	18.3	2E0C7	SENDERROR		31F7D	StoPRTPAR	
2B53D	TRCXY	18.3	2E0F4	SENDEPKT		31FFD	DODELAY	
2B67D	aMODF		2E108	BUILDKPACKET		3205C	GetChkPRTPAR	
2B770	aH>HMS		2E31F	Push#FLoop		320B1	%80	18.1
2B789	1/X15		2E4DC	APNDLCRLF	20.4	32161	PRINT	
2B7A7	RSUB1		2E4F0	CRLF\$	20.2	32387	PRINTxNLF	
2B7B0	RADD1		2E6BE	InitIOEnv		323E9	PSubErr	
2B7CA	RADDF		2E6EB	SENDLIST		323F9	REMAP	
2B7DC	ADDF		2E7EF	GETNAME		324A6	AllowPrldC1	
2B91E	MULTF		2E876	DOFINISH		3251C	PopASavptr	
2B977	DIVF		2E8D1	DOPKT		3251F	PopSavptr	
2BA0F	SQRF		2E99E	StdIOPAR		3252B	SetEcma94	
2BBB5	DIV2		2E9CB	StoIOPAR		325AA	ChkLowBat	
2BBE2	CLRFRC		2E9E6	SysSTO		32B08	ErrFixEIRU	
2BC4A	SPLITA		2EA4F	GetIOPAR		32B1A	FixEIRU	
2BCA0	SPLTAC		2EA6A	Sys@		32B74	PrintGrob	
2BD32	Y<=X		2EAE2	KERMOPEN		32CAF	ChkGrHook	
2BD76	TST15		2EB37	DOOPENIO		32CB6	adrGraphPrth	
2BE53	XYEX		2EB62	OpenIO		32FF9	SYMBNUMSOLVE	
2BE61	STAB0		2EC11	%IP>#	17.2	3303F	NUMSOLVE	
2BE6F	STAB2		2EC25	IOCheckReal		34301	Attn#	17.1
2BE7D	STCD0		2EC34	SetIOPARErr	32.2	34D2B	1NULLLAM{}	31.3
2BE8B	STCD2		2EC84	DOBAUD		34D30	NULLLAM	31.1
2BE99	EXAB0		2ECCA	DOPARITY		350B3	~LEDispList	
2BEA7	EXAB2		2ED10	DOTRANSIO		35491	apndvarlst	25.4
2BEB5	RCAB0		2EDA6	DOKERRM		355B8	PULLREALEL	23.1
2BEC0	RCAB2		2EDE1	DOBUFLen		355C8	PULLCMPEL	23.1
2BECB	RCCD0		2EE18	DOSBRK		35628	PUTEL	23.1
2BED6	RCCD2		2EE97	DOSRECV		3566F	PUTREALEL	23.1
2BEE1	CCSB1		2EEC4	SendSetup		356F3	PUTCMPEL	23.1
2BEEC	RNDC[B]		2EFD7	TRPACKETFAIL		357A8	MDIMS	23.1
2BFE3	GETAB1		2F211	LAMKML		35CAE	MATCON	23.2
2BFFD	GETAB0		2F383	IncrLAMPKNO		360B3	~LEDispItem	
2C031	GETCD0		2F39C	GetKermPkt#		3745E	SWAPROWS	23.1
2C04B	PUTAB0		2F934	FalseFalse	33.1	37B44	CKREF	37.1
2C0B3	~DoCKeyUnChA		2F989	RecvNextPkt		37E0F	MATREDIM	23.2
2C22F	STATCLST	23.3	2FEA1	SENDACK		3811F	MATTRN	23.2
2C2D9	STATSADD%	23.3	2FEB5	SENDNULLACK		3858E	StartupProc	

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
385E8	InitSysUI		3946D	DA1Bad?	41.3	3EC71	NullMenuKey	14.5
3866F	SysMenuCheck		3947B	SetDA1Bad	41.3	3EC85	NoExitAction	
386A1	SysDisplay		39489	ClrDA1Bad	41.3	3EC99	Box/StdLabel	42.4
386D8	?FlashAlert	39.2	39497	DA2aBad?	41.3	3ECB2	Box/StdLbl:	42.4
38728	SysErrorTrap		394A5	SetDA2aBad	41.3	3ED0C	Std/BoxLabel	42.4
38908	DoWarning	41.7	394B3	ClrDA2aBad	41.3	3EE1A	Do1st/2nd+:	
38926	FlashWarning	41.7	394CF	SetDA2bBad	41.3	3EE47	Echo2Macros	
38985	ParOuterLoop	14	394DD	ClrDA2bBad	41.3	3FA57	Key>U/SKeyOb	
389BC	POLSaveUI	14.1	394F9	SetDA3Bad	41.3	3FA7A	?Key>UKeyOb	
38A64	POLSetUI	14.1	39507	ClrDA3Bad	41.3	3FB1A	Key>StdKeyOb	
38AEB	POLKeyUI	14.1	39523	SetDA1IsStat	41.3	3FCAF	SetKeysNS	
38B45	POLErrorTrap		39531	ClrDA1IsStat	41.2	3FDBD	2DropBadKey	
38B77	POLResUI&Err	14.1	3957A	SetNoRollDA2	41.3	3FDC7	DropBadKey	
38B90	POLRestoreUI	14.1	3958B	ClrNoRollDA2	41.3	3FDD1	DoBadKey	
38C08	AppDisplay!		3959C	?DispStatus		3FDFF	'DoBadKey	34.1
38C18	AppDisplay@		395BA	DispStatus		3FE12	'DoBadKeyT	34.1
38C38	AppKeys!		39632	Blank&GROB!	42.3	3FE26	H/WKey>KeyOb	
38C58	AppKeys0		39673	AngleStatus		3FE44	H/W>KeyCode	
38C68	AppExitCond!		396C8	ComVecStatus		3FE7B	ModifierKey?	
38C78	AppExitCond@		39748	UserFlagStat		3FF1B	?CaseKeyDef	14.4
38C98	AppError!		397BB	UserKeysStat		3FF48	?CaseRomptr@	
38CAB	AppError@		3981B	AlgEntryStat		4019D	StdMenuKeyNS	
38CFB	AppMode?		39853	PrgmEntrStat		401D4	StdMenuKeyLS	
38D09	SetAppMode		3988B	DispDir?Time		40337	DoNameKeyRS	
38D17	ClrAppMode		398F4	DispDir?Tim1		40350	DoNameKeyLRS	
38D33	SetNAppKeyOK		39958	DispDir?Tim2		40454	DoKeyOb	
38D5D	SetDoStdKeys		39971	PathStatus		40788	TakeOver	14.5
38D8A	SetAppSuspOK		39AF1	DispTimeReq?		40792	SystemLevel?	
38D9B	ClrAppSuspOK		39B0A	DispStsBound		407FB	MenuMaker	
38DAC	DA1OK?	41.3	39B85	?DispStack		40828	MenuKey	
38EB5	DA3OK?	41.3	39BF3	?RollUpDA2		4085A	Modifier	
38F28	DA1OK?NOTIT	41.3	39F6F	LINESOFSTACK		408AA	ImmedEntry?	41.5
38F41	DA2aOK?NOTIT	41.3	39FB0	AbbrevStack?		40A6F	KeyOb!	
38F5A	DA2bOK?NOTIT	41.3	39FC1	SetAbbrevStk		40A82	KeyOb@	
38F73	DA3OK?NOTIT	41.3	39FD2	ClrAbbrevStk		40A95	KeyOb0	
38FB9	DA2aLess1OK?	41.3	3A00D	DispEditLine		40AD9	Parse.1	
38FD2	SetDA1Valid	41.3	3A1CA	?DispMenu		40B2E	ParseFail	
38FEB	SetDA2aValid	41.3	3A1E8	DispMenu		40BC9	AtUserStack	30
38FFF	SetDA2bValid	41.3	3A1FC	DispMenu.1		40C76	SaveLastEdit	
39018	SetDA3Valid	41.3	3A297	Grob>Menu	42.4	40C94	CMDSTO	
3902C	SetDA1Temp	41.3	3A2B5	Str>Menu	42.4	40CE9	CacheStack	
39045	SetDA2aTemp	41.3	3A2C9	Seco>Menu	42.4	40D25	LockAlpha	41.5
39059	SetDA2bTemp	41.3	3A2DD	Id>Menu	42.4	40D39	UnLockAlpha	41.5
39072	SetDA3Temp	41.3	3A328	MakeStdLabel	42.4	40D93	NoEdit?case	
39086	SetDA2aEcho	41.3	3A38A	MakeBoxLabel	42.4	40DC0	DoMenuKey	
390A4	MENoP&FixDA1	41.3	3A3EC	MakeDirLabel	42.4	40DD4	DoDelim	
390B3	MENP&FixDA12	41.3	3A44E	MakeInvLabel	42.4	40DF7	DoDelims	
390CC	ClrDA1OK	41.3	3A4AB	MakeLabel	42.4	40E3D	?ClrAlg	
390E5	ClrDA2aOK	41.3	3A4CE	Disp5x7	41.7	40E5B	?ClrAlgSetPr	
390FE	ClrDA2bOK	41.3	3A546	BlankDA1	41.4	40E88	REPEATER	
39117	ClrDA2OK	41.3	3A55F	BlankDA2	41.4	40EE7	SLOW	38
3912B	ClrDA3OK	41.3	3A578	BlankDA12	41.4	40F02	VERYSLOW	38
39144	ClrDAasOK	41.3	3A591	BlankDA2a	41.4	40F12	VERYVERYSLW	38
3915D	SetDA2Valid	41.3	3A71C	DoNextRow		40F22	XEQStoKey	
3918A	SetDA2NoCh	41.3	3A735	DoPrevRow		40F86	InitMenu	
3919E	SetDA12NoCh	41.3	3A9B8	'DROPFALSE	34.1	41008	StartMenu	
391B2	SetDA23NoCh	41.3	3A9CE	TurnOffKey		410C6	SetThisRow	
391C6	SetDA13NoCh	41.3	3A9E7	SetSomeRow		41111	CheckMenuRow	
391EE	SetDA123NoCh	41.3	3AA0A	1A/LockA		41175	LoadTouchTbl	
39207	SetDA2OKTemp	41.3	3ADED	DoPlotMenu		4139B	SaveLastMenu	
3921B	SetDA12Temp	41.3	3B211	DoFirstRow		41422	>Review\$	
3922F	SetDAsTemp	41.3	3BDFA	EditMenu		415C9	GetMenu%	
39301	SetDA3ValidF	41.3	3BE54	DoSolvrMenu		415F1	%100	18.1
393D3	SetDA1NoCh	41.3	3E2DD	<SkipKey		41679	InitMenu%	
393FD	SetDA2aNoCh	41.3	3E35F	>SkipKey		41741	InitTrack:	
39419	DA2bNoCh?	41.3	3E3E1	<DelKey		417F3	SetRebuild	
39427	SetDA2bNoCh	41.3	3E4CA	>DelKey		41848	MenuRow!	
39435	ClrDA2bNoCh	41.3	3E586	TogInsertKey		4185B	MenuRow@	
39451	SetDA3NoCh	41.3	3E5CD	IStackKey		4186E	LastMenuRow!	

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
41881	LastMenuRow@		44730	EDITPARTS		4CE4C	EXITFCNsto	
418A4	MenuDef@		4488A	NoEditLine?		4CE6F	DOGRAPHIC	
418D4	MenuRowAct!		4489E	ROWNUM		4CEE7	GraphicExit	
418F4	LabelDef!		448C1	?TogU/LCase		4CF05	GDISPCENTER	
41904	DoLabel	42.4	44C31	DoNewMatrix		4CF41	SETLOOPENV	
41914	MenuKeyNS!		44F42	BindMatVars		4CF68	ExitFcn	
41924	MenuKeyNS@		44FE7	DoOldMatrix		4D11E	DROPDEADTRUE	
41934	DoMenuKeyNS		45023	InitOldMat		4D132	SCROLLUP	41.6
41944	MenuKeyLS!		450E0	~BRDispItems		4D150	SCROLLLEFT	41.6
41964	MenuKeyRS!		45676	Blank\$	20.4	4D16E	SCROLLDOWN	41.6
41984	ReviewKey!		45AE0	GetMat/Vec		4D18C	SCROLLRIGHT	41.6
419C4	ExitAction!		45C2F	RowElt#		4DA0D	CROSS_HAIRS	
419E4	LastMenuDef!		45D1F	GetElt		4DA76	CROSS_OFF	
419F4	LastMenuDef@		46409	CopyRowsUp		4E266	CHECKMENU	
41A8D	UserKeys?		4651C	CopyColsLeft		4E2AC	MENUOFF	
41CA2	Ck&DecKeyLoc	40.2	46625	CopyRowsDown		4E2CF	TURNMENUOFF	41.2
41D92	CodePl>%rc.p	40.2	4677E	CopyColsRght		4E347	TURNMENUON	41.2
41F3F	GetUserKeys		4744F	'IDX	31.1	4E360	MENUOFF?	41.2
41F65	WaitForKey	40.2	47467	STOAPPLDATA		4E37E	LINECHANGE	
4203C	GetKeyOb		47984	APPprompt1!		4E442	CURRENTMARK?	
42078	?BlinkCursor		479A7	APPprompt2		4E46A	EQCURSOR?	
420A0	GETKEY*		48FF9	SORTASLOW		4E497	PREMARKON	
420F5	ATTNxcp		490E0	~BRinverse		4E4B0	NEWMARK	
42113	ALARMxcp		491D5	AUTOSCALE		4E582	DRAWBOX#	
42131	SLEEPxcp		494B4	%.1	18.1	4E6EF	DispCoord1	
42145	UARTxcp		49709	PromptIdUtil	20.3	4E776	Z-BOX	
42159	GETKEY		49AD3	PointDerivUt		4ECAD	CROSSMARKON	
42249	UART?		49BA5	FcnUtilEnd		4F052	WINDOW#	
42262	ATTN?	40.2	49BD2	RootUtil		4F0AC	DOPX>C	
4226A	addrATTNFLG		49C54	CkeQUtil		4F179	DOC>PX	
4227F	TIMEOUT?		49F06	PointMoveCur		4F1D8	SWAPTRUE	33.1
42284	adrTIMEOUTCL		4A055	DISPCOORD2		4F408	C%>#	
422A1	ALARM?	38	4A0AA	GetEqN		4F78C	GROB+#	
423BB	settimeout		4A95A	ICMPDRPRTDRP		4F7E6	CKGROBFITS	42.3
423D3	clrtypeout		4A9AF	CHECKPVAR		50262	%1+	18.3
42402	KEYINBUFFER?	40.2	4AAEA	MAKEPVAR		50276	%1-	18.3
4243E	?ATTNQUIT	40.2	4AB1C	ID_X	31.1	503D4	DOLCD>	42.3
4245C	NoAttn?Semi	40.2	4AB2A	PvarsC%0		50438	DO>LCD	42.3
42475	DoCAAlarmKey		4AB59	ID_Y	31.1	5046A	DOCLLCD	41.4
4248E	CtlAlarm!		4ADB0	GETSCALE		5053C	CROSSGROB	42.1
424DA	DCursor		4AE3C	PUTSCALE		5055A	MARKGROB	
4256B	LCursor		4AF63	GETINDEP		50578	GROBDIM	42.2
42660	UCursor		4AF77	PUTINDEP		505C6	GETXPOS	
426F1	LastERow?		4AF8B	PUTINDEPLIST		505E4	getxpos	
4272D	TopERow?		4AFDB	GETRES		5068D	GETYPOS	
427AF	Cursor&Disp		4B012	PUTRES		506AB	getypos	
42AE4	EchoChrKey		4B062	GETPTYPE		5072B	TOGGLELINE#3	
42BB6	Echo\$NoChr00		4B076	PUTPTYPE		50758	DRAWLINE#3	
42BD4	Echo\$Key		4B0DA	GETPMIN&MAX		50ACC	LINEOFF3	
42C3D	CkChr00	20.5	4B10C	GETXMIN		50ADB	TOGLINE3	42.3
42D32	EditLevel1		4B120	GETYMIN		50AEA	LINEON3	
42D46	ViewLevel1		4B139	GETXMAX		50AF9	TOGLINE	42.3
42D82	CharEdit		4B14D	GETYMAX		50B08	LINEOFF	42.3
42DC8	ObEdit		4B166	PUTXMIN		50B17	LINEON	
42E27	Roll&Do:		4B189	PUTYMIN		50D78	LASTPT?	
42E86	Rcl&Do:		4B1AC	PUTXMAX		50DA5	PlotOneMore?	
42EC7	AdjEdModes		4B1CF	PUTYMAX		50E59	!#1+IF<dim-1	
42F44	InputLine		4B323	MAKEPICT#		50EA5	!#1-IF>0	
430CF	DispILPrompt		4B364	GETPARAM		510AD	INDEPVAR	
43179	InputLEnter		4B553	VSCALE		510D5	RECORDX&YC%	
43200	InputLAttn		4B5AD	HSCALE		51125	CLEARMENU	
4325A	SetCursor		4B60C	DOERASE		51148	CKPICT	30.1
44277	InitEd&Modes		4B6D9	PLOTERR		51166	CHECKPICT	
4428B	InitEdLine		4B710	RESETDEPTH		511E3	CHECKHEIGHT	
44394	InitEdModes		4B765	PLOTPREP		5129C	'IDFUNCTION	34.1
443CB	EditString		4B7D8	GetRes		512C4	'IDPOLAR	34.1
444A5	CURSOR_END?		4BFAE	NEXTSTEP		512D8	'IDPARAMETER	34.1
444EE	Char>Edit		4C09B	NEWINDEP		5133C	PtoR	
44683	EDITLINE\$		4C639	drax		514AF	GETRHS	

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
514DC	1REV		53731	SetSysFlag	39.1	545A0	cknumdsptchl	27.1
51532	2HXSLIST?	17.2	53755	ClrUserFlag	39.1	547B5	SYMBWHERE	27.2
515A0	TOPROW	41.6	53761	ClrSysFlag	39.1	54C63	nmetasyms	30.1
515B4	BOTROW	41.6	53778	TestUserFlag	39.1	55288	1GETLAMSWP1+	31.3
515FA	LEFTCOL	41.6	53784	TestSysFlag	39.1	5573D	COLAthexFCN	
5160E	RIGHTCOL	41.6	5380E	COERCEFLAG	35.1	558DC	sscknum2	27.1
5162C	WINDOWTOP?	41.6	53860	HISTON?		558F5	sncknum2	27.1
51645	WINDOWBOT?	41.6	538C0	UNDO_ON?		5590E	nscknum2	27.1
5165E	WINDOWLEFT?	41.6	538CE	UNDO_ON		560ED	xssgeneral	27.1
51677	WINDOWRIGHT?	41.6	538DC	UNDO_OFF		56101	xnsgeneral	27.1
51690	JUMPTOP	41.6	538F8	NOBLINK		5611F	xsnsgeneral	27.1
516AE	JUMPBOT	41.6	53968	AlgEntry?	41.5	57D90	SYMCOLCT	27.2
516E5	JUMPLEFT	41.6	53976	SetAlgEntry	41.5	580E7	~UTYPEEXT0?	
51703	JUMPRIGHT	41.6	53984	ClrAlgEntry	41.5	58D75	SYMSHOW	27.2
51735	REPEATERCH		539F9	SetISysFlag		590B0	~DoKeyCancel	
5179E	DUPGROBDIM	42.2	53A20	REPLACE_MODE		5A01D	SWAPcompSWAP	
517F3	EQUALcasedro	35.5	53A2E	INSERT_MODE		5A036	uncrunch	27.1
5182F	ISTOP-INDEX	36.2	53A3C	INSERT?		5A0B0	~DoKeyOK	
51843	SWAP#1+SWAP	17.3	53A4A	EditLExists?		5A310	symbn	
51857	SWAP#1-SWAP	17.3	53A90	ClrNewEditL		5DE55	RDROPFALSE	33.1
5187F	GBUFFGROBDIM	41.1	53AE4	NoIgnoreAlm		5DE7D	reversym	29
51893	ORDERXY#	42.3	53B31	setflag		5E0DA	P{ }N	25.4
518CA	ORDERXY%	42.3	53BDD	RAD?	39.1	5E370	NDUPN	29
5193B	C%1	19.1	53C37	DOHEX	39.1	5E3AC	psh&	0
5196A	C%-1	19.1	53C43	DOBIN	39.1	5E401	pshltop&	26.3
519A3	C>Re%	19.2	53C4F	DOOCT	39.1	5E415	top&	0
519B7	C>Im%	19.2	53C5B	DODEC	39.1	5E4A9	#1-SWAP	17.3
519CB	C%>%%	19.2	53CAA	dostws	39.1	5E4A9	pull	26.3
519DF	C%>%%SWAP	19.2	53D04	bitAND	21.3	5E4BD	pullrev	26.3
519F8	C%>C%	19.2	53D15	bitOR	21.3	5E4D1	pshtop&	26.3
51A07	%%>C%	19.2	53D26	bitXOR	21.3	5E4EA	pullpshl&	26.3
51A37	Re>C%	19.2	53D4E	bitNOT	21.3	5E652	symcomp	27.1
51B2A	C%0=	19.4	53D5E	bitSL	21.3	5E67A	pshzer	26.3
51B43	C%0=	19.4	53D6E	bitSLB	21.3	5E706	pshl&	26.3
51B70	C%CHS	19.3	53D81	bitSR	21.3	5E7A5	pshl&rev	26.3
51B91	C%CHS	19.3	53D91	bitSRB	21.3	5E984	nonopcase	35.5
51BB2	C%CONJ	19.3	53DA4	bitRR	21.3	5EA9F	pshzerpsharg	27.3
51BC1	C%CONJ	19.3	53DE1	bitRRB	21.3	5EB1C	psh	26.1
51BE4	%+SWAP	18.3	53E0C	bitRL	21.3	5EDFC	numblstcase	
51EFA	C%1/	19.3	53E3B	bitRLB	21.3	5EE10	M-1stcasechs	35.4
52062	C%ABS	19.3	53E65	bitASR	21.3	5EF15	AEQ1stcase	35.4
52099	C%ARG	19.3	53EA0	bit+	21.3	5EFD9	MEQ1stcase	35.4
520CB	C%SGN	19.3	53EB0	bit-	21.3	5EFF9	MEQopscase	35.4
520E0	~BRViewItem		53ED3	bit*	21.3	5F048	AEQopscase	35.4
52107	C%SQRT	19.3	53EE4	MPY		5F061	Mid1stcase	35.4
52193	C%EXP	19.3	53F05	bit/	21.3	5F0AA	idntcase	35.5
521E3	C%LN	19.3	54039	WORDSIZE	39.1	5F0CD	idntlamcase	35.5
522BF	C%LOG	19.3	54050	BASE	39.1	5F0FA	num0=case	35.3
52305	C%ALOG	19.3	54061	HXS>\$	20.3	5F127	%0=case	35.3
52342	C%R^C	19.3	540BB	hxs>\$	20.3	5F13B	C%0=case	35.3
52360	C%C^R	19.3	5422C	PUSHhxs		5F154	num1=case	35.3
52374	C%C^C	19.3	54266	GPPushA		5F181	%1=case	35.3
524AF	C%0	19.1	5429F	bit%#/	21.3	5F19F	C%1=case	35.3
524F7	C%1	19.1	542BD	bit%#%	21.3	5F1BD	num2=case	35.3
52530	C%SIN	19.3	542D1	bit%#*	21.3	5F1EA	%2=case	35.3
52571	C%COS	19.3	542EA	bit%#*	21.3	5F208	C%2=case	35.3
525B7	C%TAN	19.3	542FE	bit%#-	21.3	5F23A	num-1=case	35.3
5262F	C%SINH	19.3	5431C	bit%#-	21.3	5F267	%-1=case	35.3
52648	C%COSH	19.3	54330	bit%#+	21.3	5F285	C%-1=case	35.3
5265C	C%TANH	19.3	54349	bit%#+	21.3	5FB49	NOTcaseFALSE	35.1
52675	C%ATAN	19.3	5435D	#>%	19.2	5FB76	ROT#1+UNROT	17.3
527EB	C%ATANH	19.3	5435D	HXS>%	18.2	60EBD	RSWAP	
52804	C%ASIN	19.3	543F9	%>#	17.2	60EE7	ROTSWAP	29
5281D	C%ASINH	19.3	544D9	HXS==HXS	21.4	60EE7	XYZ>YXZ	29
52836	C%ACOSH	19.3	544EC	HXS#HXS	21.4	60F0E	ROTDROPSWAP	29
52863	C%ACOS	19.3	54500	HXS>HXS	21.4	60F0E	UNROTSWAPDRO	29
52D26	4NULLLAM{ }	31.3	5452C	HXS>=HXS	21.4	60F0E	XYZ>ZY	29
530E0	~BRGetItem		5453F	HXS<=HXS	21.4	60F21	ROTDROP	29
53725	SetUserFlag	39.1	54552	HXS<HXS	21.4	60F21	XYZ>YZ	29

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
60F33	SWAPROT	29	613E7	2GETLAM	31.3	61A18	#0=?SEMI	35.2
60F33	UNROTSWAP	29	6140E	3GETLAM	31.3	61A2C	NOT?SEMI	35.1
60F33	XYZ>ZYX	29	61438	4GETLAM	31.3	61A3B	?SEMI	34
60F4B	3DROP	29	6145C	5GETLAM	31.3	61A47	SEMILOOP	
60F4B	XYZ>	29	6146C	6GETLAM	31.3	61A58	ITE_DROP	35.1
60F54	7DROP	29	6147C	7GETLAM	31.3	61A6D	COLA_EVAL	34.2
60F66	6DROP	29	6148C	8GETLAM	31.3	61A8E	COLARPITE	35.1
60F72	5DROP	29	6149C	9GETLAM	31.3	61AD8	ITE	35.1
60F7E	4DROP	29	614AC	10GETLAM	31.3	61AE9	2'RCOLARPITE	35.1
60F7E	XYZW>	29	614BC	11GETLAM	31.3	61B45	2@REVAL	34
60F83	4DropLoop		614CC	12GETLAM	31.3	61B55	3@REVAL	34
60F9B	SWAPDROP	29	614DC	13GETLAM	31.3	61B72	NOT?DROP	35.1
60F9B	XY>Y	29	614EC	14GETLAM	31.3	61B89	ticR	34
60FAC	3UNROLL	29	614FC	15GETLAM	31.3	61C1C	EXPAND	21.2
60FAC	UNROT	29	6150C	16GETLAM	31.3	61CE9	CACHE	31.3
60FAC	XYZ>ZXY	29	6151C	17GETLAM	31.3	61D3A	SAVELAM	
60FBB	4ROLL	29	6152C	18GETLAM	31.3	61D41	SAVESTACK	31.3
60FBB	FOURROLL	29	6153C	19GETLAM	31.3	61EA7	DUMP	31.3
60FBB	XYZW>YZWX	29	6154C	20GETLAM	31.3	61F8F	undo	31.3
60FD8	5ROLL	29	6155C	21GETLAM	31.3	61FA9	DUPROM-WORD?	28.2
60FD8	FIVEROLL	29	6156C	22GETLAM	31.3	61FB6	ROM-WORD?	28.2
61002	6ROLL	29	615E0	1PUTLAM	31.3	61FCF	Rom-Word?	
61002	SIXROLL	29	615F0	2PUTLAM	31.3	62001	2SWAP	29
6103C	8ROLL	29	61600	3PUTLAM	31.3	62020	DUPTYPECHAR?	33.3
6103C	EIGHTROLL	17.1	61610	DUP4PUTLAM	31.3	62025	TYPECHAR?	33.3
6106B	7ROLL	29	61615	4PUTLAM	31.3	62035	DUPTYPEIDNT?	33.3
6106B	SEVENROLL	29	61625	5PUTLAM	31.3	6203A	TYPEIDNT?	33.3
61099	DUP4UNROLL	29	61635	6PUTLAM	31.3	6204A	DUPTYPEEXT?	33.3
6109E	4UNROLL	29	61645	7PUTLAM	31.3	6204F	TYPEEXT?	33.3
6109E	FOURUNROLL	29	61655	8PUTLAM	31.3	62073	GPOverWrT/FL	
6109E	XYZW>WXYZ	29	61665	9PUTLAM	31.3	62076	GPOverWrTLp	
610C4	5UNROLL	29	61675	10PUTLAM	31.3	6207D	OverWrF/TLp	
610C4	FIVEUNROLL	29	61685	11PUTLAM	31.3	62080	OverWrTLoop	
610FA	6UNROLL	29	61695	12PUTLAM	31.3	62096	GPOverWrFLp	
610FA	SIXUNROLL	29	616A5	13PUTLAM	31.3	6209D	OverWrT/FLp	
6112A	ROTROT2DROP	29	616B5	14PUTLAM	31.3	620A0	OverWrFLoop	
6112A	UNROT2DROP	29	616C5	15PUTLAM	31.3	620B6	GPPushT/FLp	
6112A	XYZ>Z	29	616D5	16PUTLAM	31.3	620B9	GPPushTLoop	
6113C	4UNROLL3DROP	29	616E5	17PUTLAM	31.3	620C0	PushF/TLoop	
6113C	XYZW>W	29	616F5	18PUTLAM	31.3	620C3	DOTRUE	
6114E	2RDROP	34	61705	19PUTLAM	31.3	620C3	PushTLoop	
61160	3RDROP	34	61715	20PUTLAM	31.3	620D2	GPPushFLoop	
61172	#-PICK	29	61725	21PUTLAM	31.3	620D9	PushT/F	
61184	#+PICK	17.3	61735	22PUTLAM	31.3	620D9	PushT/FLoop	
6119E	DUP#1+PICK	29	61745	DUPTEMPENV	31.3	620DC	DOFALSE	
611A3	#1+PICK	29	617D8	GETLAMPAIR		620DC	PushFLoop	
611BE	#2+PICK	29	6186C	#=case	35.2	620EB	OVER#=	17.4
611D2	#3+PICK	29	6187C	OVER#=case	35.2	62103	DROPTTRUE	33.1
611E1	#4+PICK	29	61891	DUP#0=case	35.2	6210C	DROPTFALSE	33.1
611F9	2DUPSWAP	29	61896	#0=case	35.2	62115	DUPTYPELAM?	33.3
611F9	DUP3PICK	29	618A8	DUP#0=csedrp	35.2	6211A	TYPELAM?	33.3
611FE	3PICK	29	618BA	EQcasedrop	35.5	6212A	DUPTYPEBINT?	33.3
6121C	4PICK	29	618D3	#=casedrop	35.2	6212F	TYPEBINT?	33.3
6123A	5PICK	29	618E8	NOTcasedrop	35.1	6213F	DUPTYPEHSTR?	33.3
6125E	6PICK	29	618F7	casedrop	35.1	62144	TYPEHSTR?	33.3
61282	7PICK	29	61910	NOTcase2drop	35.1	62154	DTYPECSTR?	33.3
612A9	8PICK	29	6191F	case2drop	35.1	62154	DUPTYPECSTR?	33.3
612CC	#-ROLL	17.3	61933	EQcase	35.5	62159	TYPECSTR?	33.3
612DE	#+ROLL	17.3	6194B	caseDROP	35.1	62169	DTYPEREAL?	33.3
612F3	#1+ROLL	29	61960	NOTcaseDROP	35.1	62169	DUPTYPEREAL?	33.3
61305	get1		61970	case2DROP	35.1	6216E	TYPEREAL?	33.3
61318	#2+ROLL	29	61984	NOTcase2DROP	35.1	6217E	DUPTYPECMP?	33.3
6132C	#-UNROLL	29	61993	case	35.1	62183	TYPECMP?	33.3
6133E	#+UNROLL	17.3	619AD	NOTcase	35.1	62193	DTYPEARRY?	33.3
61353	#1+UNROLL	17.3	619BC	IT	35.1	62193	DUPTYPEARRY?	33.3
61365	#2+UNROLL	29	619CB	GOTO	34	62198	TYPEARRY?	33.3
61380	DUPUNROT	29	619E0	?GOTO	34	621A8	DUPTYPEROMP?	33.3
61380	SWAPOVER	29	619F3	NOT?GOTO	34	621AD	TYPEROMP?	33.3
613B6	1GETLAM	31.3	61A02	popflag		621BD	DUPTYPERRP?	33.3

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
621C2	TYPERRP?	33.3	6281A	#1-DUP	17.3	62E4E	#1-1SWAP	17.3
621D2	DUPTYPESYMB?	33.3	62830	SWAPDROPDUP	29	62E67	ONESWAP	17.1
621D7	TYPESEYMB?	33.3	6284B	SWAPDROPDOWN	29	62E7B	COERCESWAP	17.1
621E7	DTYPECOL?	33.3	6284B	UNROTDROP	29	62E8F	%>%SWAP	18.3
621E7	DUPTYPECOL?	33.3	6284B	XYZ>ZX	29	62EA3	%%*SWAP	18.3
621EC	TYPECOL?	33.3	62864	4ROLLDROP	29	62EB7	XYZ>ZTRUE	33.1
621FC	DUPTYPEGROB?	33.3	62880	5ROLLDROP	29	62ECB	4ROLLSWAP	29
62201	TYPEGROB?	33.3	6289B	2DUP#<	17.4	62EDF	3PICKSWAP	29
62211	DTYPELIST?	33.3	628B5	2DUP#	17.4	62EF3	4PICKSWAP	29
62211	DUPTYPELIST?	33.3	628D1	2DUP#>	17.4	62F07	1GETSWAP	31.3
62216	TYPELIST?	33.3	628EB	DUP#1+	17.3	62F1B	?SWAP	35.1
62226	DUPTYPETAG?	33.3	62904	SWAP#1+	17.3	62F2F	!append\$SWAP	20.4
6222B	TYPEFLAGGED?	33.3	62904	SWP1+	17.3	62F43	NOT?SWAPDROP	35.1
6223B	TYPERARRY?	33.3	6292F	DUP#1-	17.3	62F5C	?SWAPDROP	35.1
62256	TYPECARRY?	33.3	62946	DROPONE	17.1	62F75	#1+NDROP	29
62266	DUP#0=	17.4	62958	RDROPCOLA	34	62F75	N+1DROP	29
62289	#3=	17.4	6296D	COLACOLA	34.2	62F89	ROLLDROP	29
6229A	#2=	17.4	62986	COLAcase		62F9D	MDIMSDROP	23.1
622A7	#1=	17.4	629A1	COLANOTcase		62FB1	DUPROT	29
622B6	#1<>	17.4	629BC	ORcase	35.1	62FC5	DROPROT	29
622C5	DUP#1=	17.4	629D0	REQcase		62FD9	#1-ROT	17.3
622D4	DUP#0<>	17.4	629E9	REQcasedrop		62FED	%%*ROT	18.3
622E5	!insert\$	20.4	62A02	SAFESTO	37.2	63001	4ROLLROT	29
622EF	SWAP&\$	20.4	62A2F	DUPSAFE@	37.2	63001	FOURROLLROT	29
62312	!!append\$?	20.4	62A34	SAFE@	37.2	63015	4UNROLLROT	29
62376	!append\$	20.4	62A61	?>ROMPTR	28.2	63029	DROPOVER	29
62394	!!insert\$	20.4	62A84	?ROMPTR>	28.2	6303D	EQOVER	33.2
623A0	!!append\$	20.4	62ABB	MACRODCMP		63051	#+OVER	17.3
62474	'RSaveRomWrd		62B0B	2DROPFALSE	33.1	63065	#-OVER	17.3
62474	'RSAVEWORD		62B1F	PALPTRDCMP		63079	ZEROOVER	17.1
624BA	#MIN	17.3	62B5B	palrompdcmp		6308D	UNROTOVER	29
624C6	#MAX	17.3	62B6F	#0=UNTIL	36.1	630A1	4ROLLOVER	29
624FB	#-#2/	17.3	62B88	INCOMPDROP	25.3	630B5	3PICKOVER	29
62535	DROPZERO	17.1	62B9C	NTHCOMPDROP	25.1	630C9	4PICKOVER	29
6254E	2DROP00	17.1	62BB0	APPEND_SPACE	20.4	630DD	DUPPICK	29
6256A	#3+	17.3	62BC4	7UNROLL	29	630E3	~PCunpack	
6257A	#4+	17.3	62BD8	RESOROMP		630F1	DUPROLL	29
6258A	#5+	17.3	62BF1	%10*	18.3	63105	OVER#2+UNROL	29
6259A	#6+	17.3	62C05	DUP@	37.2	63119	8UNROLL	29
625AA	#7+	17.3	62C19	DUPROMPTR@	28.2	6312D	10UNROLL	29
625BA	#8+	17.3	62C2D	#=ITE	35.2	63141	OVERARSIZE	23.1
625CA	#9+	17.3	62C41	INNERDUP	25.3	63155	'ERRJMP	34.1
625DA	#10+	17.3	62C55	NOTAND	33.1	63169	caseERRJMP	35.6
625EA	#12+	17.3	62C69	TOTEMPSWAP	37.1	6317D	?CARCOMP	25.1
625FA	#3-	17.3	62C7D	ROT2DUP	29	63191	NEWLINE\$&\$	20.4
6260A	#4-	17.3	62C91	ROTAND	33.1	63191	NEWLINE&\$	20.4
6261A	#5-	17.3	62CA5	ROTOVER	29	631A5	#1-{}N	25.4
6262A	#6-	17.3	62CB9	DUPDUP	29	631B9	TWO{}N	25.4
6264E	#10*	17.3	62CCD	OVERDUP	29	631CD	THREE{}N	25.4
62674	#8*	17.3	62CE1	COERCEDUP	17.1	631E1	DUPINCOMP	25.3
62691	#6*	17.3	62CF5	UNROTUP	29	631F5	SWAPINCOMP	25.3
626AE	5skipcola	34.2	62D09	4UNROLLDUP	29	63209	DUPNULL\$?	20.5
626DC	3skipcola	34.2	62D1D	NTHCOMDDUP	25.1	6321D	DUPNULLCOMP?	25.1
626E5	2skipcola	34.2	62D31	OVERSWAP	29	63231	DUPLINCOMP	25.1
626EE	skipcola	34.2	62D31	OVERUNROT	29	63245	#1-SUB\$	20.4
626F7	DUP#2+	17.3	62D45	ROLLSWAP	29	63259	1_#1-SUB	20.4
6270C	DROPSWAP	29	62D59	NULL\$SWAP	20.2	63259	1_#1-SUB\$	20.4
62726	DROPSWAPDROP	29	62D6D	SUB\$SWAP	20.4	6326D	LAST\$	20.4
62726	ROT2DROP	29	62D81	%MAXorder	18.3	63281	#1+LAST\$	20.4
62726	XYZ>Y	29	62D9F	?SKIPSWAP	35.1	63295	DUP\$>ID	31.2
62747	SWAPDUP	29	62DB3	1ABNDSWAP	31.3	632A9	SWAP%>C%	19.2
62775	ROTDUP	29	62DCC	ROT#+SWAP	17.3	632BD	'NOP	34.1
62794	SWAP#-		62DCC	ROT+SWAP	17.3	632D1	::NEVAL	25.5
627A7	DROPDUP	29	62DE5	4PICK#+SWAP	17.3	632E5	2GETEVAL	31.3
627BB	DUPLIN\$	20.4	62DE5	4PICK+SWAP	17.3	632F9	DROPRDROP	34
627D5	#+DUP	17.3	62DFE	#+SWAP	17.3	63312	SWAPCOLA	34.2
627EB	Push2#aLoop		62E12	#-SWAP	17.3	63326	XYZ>ZCOLA	34.2
627F8	#-DUP	17.3	62E26	#1+SWAP	17.3	6333A	#0=?SKIP	35.2
62809	#1+DUP	17.3	62E3A	ZEROSWAP	17.1	63353	#1=?SKIP	35.2

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
6336C	#=?SKIP	35.2	6389E	GROB!ZERODRP	42.3	63E1B	dREALNcase	35.5
63385	ONE_EQ		638B2	casedrptru	35.1	63E2F	EQIT	35.5
63399	#>?SKIP	35.2	638CB	NOTcaseTRUE	35.1	63E48	DUP#0=ITE	35.2
633B2	COLASKIP	34.2	638E4	?SEMI DROP	34	63E61	ANDITE	35.1
633C6	NOT_UNTIL	36.1	638FD	SWAPUnDROP	29	63E75	EQITE	35.5
633DF	NOT_WHILE	36.1	63911	SWAPUnNDROP	26.1	63E89	#0=ITE	35.2
633F8	DUP#0<>WHILE	36.1	63925	DUP'	34.1	63E9D	#<ITE	35.2
63411	DUPINDEX@	36.2	63939	SWAP'	34.1	63EB1	#>ITE	35.2
63425	SWAPINDEX@	36.2	6394D	DROP'	34.1	63EC5	DUP#0=ITE	35.2
63439	OVERINDEX@	36.2	63961	OVER'	34.1	63ED9	UserITE	35.6
6344D	SWAPLOOP	36.2	63975	STO'	34.1	63EED	SysITE	35.6
63466	DROPLOOP	36.2	63989	TRUE'	34.1	63F01	top&Cr	
6347F	DUP#0_DO	36.2	6399D	ONEFALSE'	34.1	63F1A	metaROTDUP	26.1
63498	toLEN_DO	36.2	639B6	FALSE'	34.1	63F2E	ROTUntop&	0
634B6	lGETABND	31.3	639CA	#1+'	34.1	63F42	roll2top&	0
634CA	DUP1LAMBIND	31.3	639DE	'R'R	34	63F42	rolltwotop&	0
634CF	1LAMBIND	31.3	639FC	'RRDROP	34	63F56	plDRPpZparg	27.3
634E3	caseTRUE	35.1	63A15	ONECOLA	34.2	63F6A	&\$SWAP	20.4
634F7	TRUEFALSE	33.1	63A29	dvarlsBIND		63F7E	SWAPCKREF	37.1
634F7	TrueFalse	33.1	63A3D	'LAMLNAMESTO		63F92	pZpargSWAPUn	27.3
6350B	FALSETRUE	33.1	63A56	'xDEREQ		63FA6	DROPNDROP	29
6350B	FalseTrue	33.1	63A6F	DUPNULL{ }?	25.4	63FBA	2OVER	29
6351F	ZEROFALSE	33.1	63A88	DUPZERO	17.1	63FCE	?Ob>Seco	25.5
63533	ONEFALSE	33.1	63A9C	DUPONE	17.1	63FE7	Ob>Seco	25.5
63547	#=casedrpfls	35.2	63AB0	SWAPONE	17.1	63FFB	2Ob>Seco	25.5
6356A	casedrpfls	35.1	63AC4	ONEDUP	17.1	6400F	ExitAtLOOP	36.2
63583	case2drpfls	35.1	63AC4	ONEONE	17.1	6400F	ZEROISTOPSTO	36.2
6359C	caseFALSE	35.1	63AD8	DUP TWO	17.1	64023	RclHiddenVar	37.4
635B0	ORNOT	33.1	63AEC	NOTcsdrpfls	35.1	64037	WithHidden	37.4
635C4	EQUALNOT	33.2	63B05	caseSIZEERR	35.6	64078	StoHiddenVar	37.4
635D8	2DUPEQ	33.2	63B19	NcaseSIZEERR	35.6	6408C	PuHiddenVar	37.4
635EC	DUPEQ:	33.2	63B2D	CKREAL	30.1	640A0	SaveVarRes	
635F1	EQ:	33.2	63B46	NcaseTYPEERR	35.6	640BE	SetHiddenRes	37.4
63605	EQOR	33.2	63B5A	'x*	34.1	640FA	RestVarRes	
63619	EQUALOR	33.2	63B6E	'xDER	34.1	64127	Embedded?	
6362D	2#0=OR	17.4	63B82	%%/>%	18.3	6416D	UStackDepth	
6364B	OVER#0=	17.4	63B96	UNCOERCE%%	18.2	64190	Sig?ErrJmp	
6365F	OVER#<	17.4	63BAA	DUP#0=	18.4	641CC	DupAndThen	
63673	#<3	17.3	63BBE	SWAP%%/	18.3	641FC	ZEROZERO	17.1
63687	DUP#<7	17.4	63BD2	caseDrpBadKy	35.6	64209	#ZERO#ONE	17.1
636A0	INNER#1=	25.3	63BEB	caseDEADKEY	35.6	6427A	#ZERO#SEVEN	17.1
636B4	#5=	17.4	63BEB	caseDoBadKey	35.6	6428A	#ONE#27	17.1
636C8	#2<>	17.4	63C04	GROBDIMw	42.2	6429D	#TWO#ONE	17.1
636DC	OVER#>	17.4	63C18	%%*UNROT	18.3	642AF	#TWO#TWO	17.1
636F0	#>1	17.4	63C2C	SWAP4ROLL	29	642BF	#TWO#FOUR	17.1
636F0	ONE#>	17.4	63C2C	XYZW>YWZX	29	642D1	#THREE#FOUR	17.1
63704	2DUP#+	17.3	63C40	2DUP5ROLL	29	642E3	#FIVE#FOUR	17.1
63704	DUP3PICK#+	17.3	63C54	SWAP3PICK	29	64309	ZEROZEROZERO	17.1
63718	ROT#+	17.3	63C68	3PICK3PICK	29	6431D	ZEROZEROONE	17.1
6372C	OVER#+	17.3	63C7C	SWAP4PICK	29	64331	ZEROZEROTWO	17.1
63740	3PICK#+	17.3	63C90	OVER5PICK	29	64345	SubMetaOb	26.4
63754	4PICK#+	17.3	63CA4	EQUALcasedrp	35.5	643BD	SubMetaOb1	26.4
63768	ROT#-	17.3	63CBD	DUP#0=csDROP	35.2	643EF	matchob?	
6377C	OVER#-	17.3	63CD6	jEQcase	35.5	643F9	matchob?Lp	
63790	INDEX@#-	36.2	63CEA	ANDcase	35.1	64426	POSCOMP	25.1
637A4	SWAPOVER#-	17.3	63CFE	EQUALcase	35.5	6443A	nextpos	
637B8	ROT#1+	17.3	63D12	#<case	35.2	6448A	#=POSCOMP	25.1
637CC	#-+1	17.3	63D26	#1=case	35.2	644A3	EQUALPOSCOMP	25.1
637CC	#1--	17.3	63D3A	#<>case	17.4	644BC	NTHOF	25.1
637E0	SWAP#1-	17.3	63D4E	#>2case	35.2	644D0	Find1stTrue	25.1
637F4	DROP#1-	17.3	63D67	#>case	35.2	644EE	Find1stT.1	
63808	#+-1	17.3	63D7B	j#0=case	35.3	6452F	Lookup	25.1
63808	#1-+	17.3	63D8F	REALcase	35.3	64548	Lookup.1	
6381C	COLAITE		63DA3	dARRYcase	30.1	64593	EQLookup	25.1
6383A	ERROROUT	32.1	63DB7	dLISTcase	30.1	645B1	POS\$	20.4
6384E	D0=DSKTOP		63DCB	EditExstCase		645B1	POSCHR	20.4
6385D	D1=DSKTOP		63DDF	ANDNOTcase	35.1	645BD	POS\$REV	20.4
6386C	SWAP2DUP	29	63DF3	EQUALNOTcase	35.5	645BD	POSCHRRREV	20.4
63880	RSKIP	34	63E07	dIDNTNcase	30.1	6475C	CHR>\$	20.3

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
64775	STRIPTAGS	22	64DE2	2GROB	17.1	654E9	CHR_E	20.1
647A2	STRIPTAGS12	22	64DEC	TAGGEDANY	17.1	654F0	CHR_F	20.1
647BB	TAGOBBS	22	64DF6	EXTREAL	17.1	654F7	CHR_G	20.1
6480B	NEXTCOMPOB	25.1	64E00	EXTSYM	17.1	654FE	CHR_H	20.1
648BD	>LASTRAM-WOR		64E0A	2EXT	17.1	65505	CHR_I	20.1
64B12	FORTYFOUR	17.1	64E14	ROMPANY		6550C	CHR_J	20.1
64B1C	FORTYFIVE	17.1	64E1E	BINT253	17.1	65513	CHR_K	20.1
64B26	FORTYSIX	17.1	64E28	BINT255d	17.1	6551A	CHR_L	20.1
64B30	FORTYSEVEN	17.1	64E32	REALOBOB	17.1	65521	CHR_M	20.1
64B3A	FORTYEIGHT	17.1	64E3C	#_102	17.1	65528	CHR_N	20.1
64B44	FORTYNINE	17.1	64E64	3REAL	17.1	6552F	CHR_O	20.1
64B4E	FIFTY	17.1	64F04	INTEGER337	17.1	65536	CHR_P	20.1
64B58	FIFTYONE	17.1	64FC2	ATTNERR	17.1	6553D	CHR_Q	20.1
64B62	FIFTYTWO	17.1	6506C	Connecting	17.1	65544	CHR_R	20.1
64B6C	FIFTYTHREE	17.1	6508A	EXTOBOB	17.1	6554B	CHR_S	20.1
64B6C	STRLIST	17.1	65094	#EXITERR	17.1	65552	CHR_T	20.1
64B76	FIFTYFOUR	17.1	6509E	MINUSONE	17.1	65559	CHR_U	20.1
64B80	FIFTYFIVE	17.1	650A8	%e	18.1	65560	CHR_V	20.1
64B8A	FIFTYSIX	17.1	650BD	%.5	18.1	65567	CHR_W	20.1
64B94	FIFTYSEVEN	17.1	650E7	%10	18.1	6556E	CHR_X	20.1
64B9E	FIFTYEIGHT	17.1	650FC	%180	18.1	65575	CHR_Y	20.1
64BA8	FIFTYNINE	17.1	65126	%360	18.1	6557C	CHR_Z	20.1
64BB2	SIXTY	17.1	65176	tok{	20.2	65583	CHR_a	20.1
64BBC	SIXTYONE	17.1	6518E	toksharp	20.2	6558A	CHR_b	20.1
64BC6	SIXTYTWO	17.1	651BE	tokESC	20.2	65591	CHR_c	20.1
64BD0	SIXTYTHREE	17.1	651D6	tok<<	20.2	65598	CHR_d	20.1
64BDA	BINT40h	17.1	651E2	tokexponent	20.2	6559F	CHR_e	20.1
64BDA	SIXTYFOUR	17.1	65238	NEWLINE\$	20.2	655A6	CHR_f	20.1
64BDA	YHI	17.1	65254	SPACE\$	20.2	655AD	CHR_g	20.1
64BE4	ARRYREAL	17.1	65254	tok_	20.2	655B4	CHR_h	20.1
64BE4	BINT_65d	17.1	65278	tokquote	20.2	655BB	CHR_i	20.1
64BEE	FOUR TWO	17.1	65284	tok'	20.2	655C2	CHR_j	20.1
64BF8	FOURTHREE	17.1	65290	tok,	20.2	655C9	CHR_k	20.1
64C02	SIXTYEIGHT	17.1	6529C	tok.	20.2	655D0	CHR_l	20.1
64C0C	FOURFIVE	17.1	652FC	tok-	20.2	655D7	CHR_m	20.1
64C16	SEVENTY	17.1	65308	tok=	20.2	655DE	CHR_n	20.1
64C20	SEVENTYFOUR	17.1	6534C	tok0	20.2	655E5	CHR_o	20.1
64C2A	SEVENTYNINE	17.1	65358	tok1	20.2	655EC	CHR_p	20.1
64C34	EIGHTY	17.1	653AC	tok8	20.2	655F3	CHR_q	20.1
64C3E	EIGHTYONE	17.1	653B8	tok9		655FA	CHR_r	20.1
64C3E	LISTREAL	17.1	6541E	CHR_00	20.1	65601	CHR_s	20.1
64C48	LISTCMP	17.1	65425	CHR_...	20.1	65608	CHR_t	20.1
64C52	FIVETHREE	17.1	6542C	CHR_DblQuote	20.1	6560F	CHR_u	20.1
64C5C	FIVEFOUR	17.1	65433	CHR_#	20.1	65616	CHR_v	20.1
64C66	2LIST	17.1	6543A	CHR_*	20.1	6561D	CHR_w	20.1
64C70	FIVESIX	17.1	65441	CHR_+	20.1	65624	CHR_x	20.1
64C7A	LISTLAM	17.1	65448	CHR_	20.1	6562B	CHR_y	20.1
64C84	BINT_91d	17.1	6544F	CHR_-	20.1	65632	CHR_z	20.1
64C8E	BINT_96d	17.1	65456	CHR_.	20.1	65639	CHR_->	20.1
64C98	IDREAL	17.1	6545D	CHR_/	20.1	65640	CHR_<<	20.1
64CAC	ONEHUNDRED	17.1	65464	CHR_0	20.1	65647	CHR_>>	20.1
64CC0	char	17.1	6546B	CHR_1	20.1	6564E	CHR_Angle	20.1
64CE8	BINT_115d	17.1	65472	CHR_2	20.1	65655	CHR_Deriv	20.1
64CF2	BINT_116d	17.1	65479	CHR_3	20.1	6565C	CHR_Integral	20.1
64D06	BINT_122d	17.1	65480	CHR_4	20.1	65663	CHR_LeftPar	20.1
64D10	BINT80h	17.1	65487	CHR_5	20.1	6566A	CHR_Newline	20.1
64D1A	BINT_130d	17.1	6548E	CHR_6	20.1	65671	CHR_Pi	20.1
64D1A	XHI-1	17.1	65495	CHR_7	20.1	65678	CHR_RightPar	20.1
64D24	BINT_131d	17.1	6549C	CHR_8	20.1	6567F	CHR_Sigma	20.1
64D24	XHI	17.1	654A3	CHR_9	20.1	65686	CHR_Space	20.1
64D38	SYMBREAL	17.1	654AA	CHR_:	20.1	6568D	CHR_UndScore	20.1
64D56	SYMBUNIT	17.1	654B1	CHR_;	20.1	65694	CHR_[20.1
64D6A	SYMOB	17.1	654B8	CHR_<	20.1	6569B	CHR_]	20.1
64D74	SYMREAL	17.1	654BF	CHR_=	20.1	656A2	CHR_{	20.1
64D92	SYMID	17.1	654C6	CHR_>	20.1	656A9	CHR_}	20.1
64D9C	SYMLAM	17.1	654CD	CHR_A	20.1	656B0	CHR_<=	20.1
64DB0	SYMSYM	17.1	654D4	CHR_B	20.1	656B7	CHR_>=	20.1
64DBA	SYMEXT	17.1	654DB	CHR_C	20.1	656BE	CHR_<>	20.1
64DD8	BINTC0h	17.1	654E2	CHR_D	20.1	656C5	\$_R<<	20.2

Addr.	Name	See	Addr.	Name	See	Addr.	Name	See
656D5	\$_R<Z	20.2	7DC7D	D/DABS		7DE48	nINTGSQ	
656E5	\$_XYZ	20.2	7DC88	easyabs		7DE53	nINTGSIN	
656F5	\$_<<>>	20.2	7DCA1	D/DACOS		7DE5E	nINTGCOS	
65703	\$_{ }	20.2	7DCAC	D/DACOSH		7DE69	nINTGTAN	
65711	\$_[]	20.2	7DCB7	D/DALOG		7DE74	nINTGSINH	
6571F	\$_' '	20.2	7DCC2	D/DARG		7DE7F	nINTGCOSH	
6572D	\$_::	20.2	7DCCD	D/DASIN		7DE8A	nINTGTANH	
6573B	\$_LRParens	20.2	7DCD8	D/DASINH		7DE95	nINTGASIN	
65749	\$_2DQ	20.2	7DCE3	D/DATAN		7DEA0	nINTGACOS	
65757	\$_ECHO	20.2	7DCEE	D/DATANH		7DEAB	nINTGATAN	
65769	\$_EXIT	20.2	7DCF9	D/DCHS		7DEB6	nINTGLN	
6577B	\$_Undefined	20.2	7DD17	D/DCONJ		7DEC1	nINTGLOG	
65797	\$_RAD	20.2	7DD35	D/DCOS		7DECC	nINTGALOG	
657A7	\$_GRAD	20.2	7DD40	D/DCOSH		7DED7	nINTGEXPM	
6594E	putnibs		7DD4B	D/DEXP		7DEE2	nCustomMenu	
6595A	getnibs	39.2	7DD4B	D/DEXPM1		7DEED	nCOLCTQUOTE	
659DE	Symb>HBuff	42.3	7DD56	D/DINV		7DEF8	SPLITWHERE	
66EA5	BigCursor	42.1	7DD61	D/DLN		7DF03	MANMENU+-	
66ECD	MediumCursor	42.1	7DD6C	D/DLNP1		7DF0E	MANMENU*/	
66EF1	SmallCursor	42.1	7DD77	D/DLOG		7DF19	MANMENU^	
70601	BANKMTHDS		7DD82	D/DIFTE		7DF24	MANMENUEXP	
715B1	?ACCPTR>		7DD8D	D/DSIN		7DF2F	MANMENULN	
71C3B	rNTHELCOMP		7DD98	D/DSINH		7DF3A	MANMENUCSIV	
71DB2	RPTRACC		7DDA3	D/DSQ		7DF45	MANMENUEQ	
7DBA0	nWHEREIFTE		7DDAE	D/DSQRT		7DF50	MANMENU CX	
7DBAB	nWHEREEDER		7ddb9	D/DTAN		7DF5B	MANMENUTRG	
7DBB6	nWHEREINTG		7DDC4	D/DTANH		7DF66	MANMENUATG	
7DBC1	nWHERESUM		7DDCF	D/D^		7DF71	PTYPE>PINFO	
7DBCC	nWHEREWHERE		7DDDA	D/D^X		7DF7C	MOVEVAR	
7DBD7	nWHEREFCNAPP		7DDE5	D/D^Y		7DF87	COPYVAR	
7DBE2	D/D*		7DDF0	D/DDER		850B0	~grobAlertIc	
7DBED	D/D+		7DDFB	D/DWHERE		860B0	~grobCheckKe	
7DBF8	D/D-		7DE06	D/DINTEGRAL		C50B0	~gFldVal	
7DC03	D/D/		7DE11	D/DSUM		C80B0	~GetFieldVal	
7DC0E	derquot		7DE1C	D/DAPPLY				
7DC54	derprod1		7DE27	nINTGINV				
7DC72	D/D=		7DE32	nINTGSIGN				
7DC72	D/Dalg=		7DE3D	nINTGSQRT				

Appendices

Appendix A

Tools for programming in System RPL

As said before, you cannot create System RPL programs only using your calculator. Until some years ago, it was necessary a PC to create System RPL programs. Now, you can create them using just your calculator and special programs.

A.1 HP Tools

The HP Tools was the first package for creating System RPL programs. It was created by the HP staff, but it is not supported by them. They are a set of four utilities: a RPL compiler (`RPLCOMP.EXE`), a Saturn Assembler (`SASM.EXE`), a Saturn Linker (`SLOAD.EXE`) and a library creator (`MAKEROM.EXE`). There is also a program to convert a directory into a library (which will be explained in Appendix C), `USRLIB.EXE`.

If you want to use the HP Tools to compile the examples of this book or your programs, all source files must look like this:

```
ASSEMBLE
      NIBASC   /HHP48-X/
RPL

::
* Your program here
;
```

This is because all HP files have the header `HHP48-X` (actually `X` can be any letter of your choice). Since the program file was created in a computer, the header is not added. Without it, the file won't load correctly in the HP48. So, the instructions above add the header so that your program can be transferred to the HP48 correctly.

Compiling a System RPL program with the HP48 requires several steps:

1. Create your source file (do not forget to add the above!), and save it with the extension `.S`. We will suppose that your program is called `MYPROG`. So, the file with the source code would be called `MYPROG.S`.
2. Run `RPLCOMP MYPROG.S MYPROG.A`. This will generate the file `MYPROG.A`, which is the Saturn assemble source code.

3. Now, run `SASM MYPROG.A`. This will create two files `MYPROG.L` and `MYPROG.O`.
4. Create a file called `MYPROG.M` that looks like this (see the file `SLOAD.DOC` for information on what the commands mean):

```
TITLE My First Program
REL MYPROG.O
OUTPUT MYPROG
LLIST MYPROG.LR
SEARCH ENTRIES.O
SUPPRESS XREF
END
```

5. Now, run `SLOAD -H MYPROG.M`. This will create the files `MYPROG.LR`, the listing, and `MYPROG`, which is the program. Check the file `MYPROG.LR`. If there are no errors, your program is ready. If not, edit the source file, and restart from step 2.

The HP Tools can be found on <ftp://hpcvbbs.external.hp.com/dist/ms-dos/tools.exe>, or in Goodies Disk 4. The Goodies Disk are a compilation of several programs for your HP48. They can be found at Joe Horn's home page at <http://user.kcyb.com/joehorn/>.

Also designed for programming in System RPL with a computer, there are the GNU Tools by Mario Mikocevic. They are like the HP Tools, with some of JAZZ enhancements. Their source code is available (in C), and they can be run on other platforms. You can download the source code at Mario's home page at <http://www.zems.fer.hr/~mozgy/jwz/hp48.html>. You can get them compiled for MS-DOS and Linux at Andre Schrool's home page at <http://www.engr.uvic.ca/~aschrool/>.

A.2 JAZZ

Some years ago, Detlef Mueller and Raymond Hellstern developed the RPL48 package. This program allowed the user to create System RPL programs directly in the HP48. Later, Mika Heiskanen developed JAZZ. JAZZ is a library which allows you to create System RPL programs in your calculator, disassemble and debug them. It includes several enhancements, like the lambda variable generation described in section 5.3

The examples listed in this book are designed for JAZZ. Should you prefer using tools in your computer, you'll need to make some changes (see the above section). If you prefer using JAZZ, just type them (or upload the file, if provided), and run the `ASS` command. You'll get your program.

Other useful functions of JAZZ are the disassembler (commands `DIS`, `DISXY` and `DOB`), the debugger (`SDB`) and the entries catalog (`EC`). Consult the JAZZ documentation for more information.

You can get JAZZ at <http://www.hut.fi/~mheiskan/>.

Appendix B

Creating libraries

The biggest advantage of libraries is that the user can only access the commands the creator of the library wants to be accessed. Sub-routines do not show in the menu, and cannot be directly accessed. This means that only the “main” programs need to have protection against bad input, for example. Since only they will call the sub-routines, you do not need to waste code putting this kind of check in the sub-routines, unless you are going to pass invalid parameters to them.

There are two ways to create libraries: converting a previously existent directory into one, or creating one directly. The first method is easier.

B.1 Converting a directory into a library

To create a library from a directory, the only thing you need is to create some special variables in it, and run a program to convert it. There are programs for PC's (included in the HP and GNU Tools), or for the HP48 (which can be found on Mika Heiskanen's Hack library, downloadable from <http://www.hut.fi/~mheiskan/>).

The special variables recognized by library converters are:

Variable	Description
\$ROMID	Required. Contains a real or (user) binary representing the library number. The number should be in the range of 769-1792 (#301h-#6FFh).
\$TITLE	Contains a string representing the title of the library. The first five characters are used to create the softkey in the Library menu. The library does not need to have a title. But if it doesn't, it will not show in the Library menu.
\$CONFIG	Contains the program to be executed at configuration time. Normally, this is a program to attach the library, in the form « 1234 ATTACH » or :: 1234 TOSRRP ;.
\$MESSAGE	Specifies the message table. Its format varies from program to program, so check the documentation. In USRLIB, it is a list of variable names (which contain strings). Those strings will form the message table. In other tools, it is a list of strings.
\$VISIBLE	Contains a list of the variables that will be converted to user-accessible commands. If not present, then all commands will be accessible.
\$HIDDEN	Contains a list of the variables that will not be converted to user-accessible commands. If \$HIDDEN and \$VISIBLE are present, only \$HIDDEN will be considered.

Variable	Description
\$VARS	Contains a list of the variables which will not be converted to library commands, instead, they will remain in RAM. For example, if your program stores something in a variable, the name of this variable should be specified in this list.

Once you have created the directory with your application and the above variables, the only thing you need is to run the converter program.

B.2 Creating libraries directly

Creating libraries directly is a bit more complicated. Especially if you use the HP Tools or GNU Tools. The instructions here apply for JAZZ only. The process with HP Tools or GNU Tools is similar, but there are a few differences. See their documentation for instructions.

To create a library, before any code you must use the `xROMID <numb>` statement to tell the compiler you are creating a library and it's number. Following, if your library is to appear on the menu, you must put the statement to define the title: `xTITLE <title>`. The other commands, described below, can be anywhere in the file, but it is a good practice to put them all together in the beginning. The ideal form for the library is:

```
xROMID    <library number>
xTITLE    <library title>
xCONFIG   <configuration object name>
xMESSAGE  <message table name>
EXTERNAL  <command 1 name>
...
EXTERNAL  <command N name>
```

```
* Main code
```

Then, each command that should be accessible is defined with the token `xNAME <label>`. Commands that should not be accessible are defined with `NULLNAME <label>`.

All visible commands must be preceded by a property field. This is a series of flags that indicate that the command has some special properties. A standard command with no special properties is defined with `CON(1) 8`.

Let's see an example of a library that calculates the greatest common factor and least common multiple. It has three commands: `GCF`, which calculates the greatest common factor; `LCM`, which calculates the least common multiple; and `subGCF`. The latter one is not accessible to the user. It is a subroutine used to calculate the `gcf`.

```
* Library number: 1566
xROMID    #61E
* Library title
xTITLE    MYLIB: My first library
```

```

* Name of configuration routine
xCONFIG TheConfig

* Command declarations.
EXTERNAL GCF
EXTERNAL LCM
EXTERNAL subGCF

* Configuration routine
LABEL TheConfig
::
    # 61E TOSRRP      ( Attach library )
;

* User-accessible command to calculate the GCF
ASSEMBLE
    CON(1)  8
RPL
xNAME GCF
::
    CK2&Dispatch      ( req. two arguments )
    2REAL subGCF       ( two reals? then calculate )
;

* User-accessible command to calculate the LCM
ASSEMBLE
    CON(1)  8
RPL
xNAME LCM
::
    CK2&Dispatch      ( req. two arguments )
    2REAL ::          ( two reals? )
        2DUP
        %*
        UNROT
        subGCF
        %/
    ;
;

* Sub-routine used to calculate the GCF
NULLNAME subGCF
::
    DUP%0=
    caseDROP
    DUPUNROT
    %MOD
    COLA
    subGCF
;

```