

User's Guide



PocketProfessional[®]
Software for HP 48 Graphic Expandable Calculators

M_{aubert} D_{evelopment} G_{roup}

META KERNEL



Manual

ROM Version 2.10 – Feb. 1998

Credits

Jean-Yves Avenard
Cyrille de Brebisson
Christian Bourgeois
Etienne de Foras
Gérald Squelart

Thanks to

Jake Schwartz (without whom this English manual wouldn't exist!)
Paul Courbis (the 'Voyage' is our bible)
Hewlett Packard (for the HP48 and the RPLMAN doc)

da Vinci Technologies Group, Inc. has secured world wide rights to market MetaKernel software and documentation.

899 NW Grant Avenue, Corvallis, OR 97330.

Tel: (541) 757-8416; Fax: (541) 753-7821

Technical support is provided via Internet at support@sparcom.com

© 1996-1998 MDG Corp.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of MDG Corp.

Printed in USA.

Contents

USING THE META KERNEL	15
1 INSTALLATION.....	16
2 GENERAL PRESENTATION	17
3 STACK DISPLAY	18
4 CONFIGURATION	21
5 INTERACTIVE STACK.....	25
6 COMMAND LINE	26
7 COMMAND LIBRARY.....	31
8 EQUATION EDITOR.....	32
9 MATRIX EDITOR	42
10 GROB EDITOR (PICTURE2).....	44
11 FILER.....	49
UTILITIES	53
1 FONT EDITOR (EDF)	54
2 MACHINE LANGUAGE COMPILER (MASD).....	56
3 DISASSEMBLER.....	75
4 MISCELLANEOUS UTILITIES	76
APPENDIXES	83
1 INTRODUCTION TO ASSEMBLY LANGUAGE.....	84
2 INTRODUCTION TO SYSTEM RPL.....	94
3 TIPS AND TRICKS	125
4 THE MK FOR PROGRAMMERS.....	129
INDEX.....	147

Table of Contents

USING THE META KERNEL	15
1	INSTALLATION..... 16
2	GENERAL PRESENTATION 17
3	STACK DISPLAY 18
3.1	<i>Display</i> 18
3.1.1	Status area..... 18
3.1.2	Stack 18
3.1.3	Menu labels 18
3.2	<i>Display configuration</i> 19
3.2.1	Changing the display font..... 19
3.2.2	Modifying the current font..... 20
3.2.3	Changing the mini font..... 20
4	CONFIGURATION 21
4.1	<i>MK variables</i> 21
4.2	<i>MK user and system flags (MKUF, MKSF)</i> 21
5	INTERACTIVE STACK..... 25
6	COMMAND LINE 26
6.1	<i>Generalities</i> 26
6.2	<i>Editing menu</i> 26
6.2.1	Major options..... 26
6.2.2	Operations on the selection..... 26
6.2.3	Search operations..... 26
6.2.4	Miscellaneous 27
6.3	<i>Styles</i> 27
6.3.1	Generalities..... 27
6.3.2	Style menu..... 27
6.3.3	Using styles 28
6.3.4	Using fonts..... 28
6.4	<i>Miscellaneous command line options</i> 29
6.4.1	Full screen mode..... 29
6.4.2	Auto-indent mode..... 29
6.4.3	Special keystokes for System RPL 30
6.4.4	Big strings editing..... 30
7	COMMAND LIBRARY..... 31
8	EQUATION EDITOR..... 32
8.1	<i>First view of EQW</i> 32
8.1.1	Introduction 32
8.1.2	Modes 32
8.1.3	Editing 33
8.2	<i>Creating equations</i> 33
8.2.1	Running EQW 33

8.2.2	Exiting EQW	33
8.2.3	Numbers and names.....	33
8.2.4	Additions, subtractions, and multiplications.....	34
8.2.5	Complex numbers and expressions.....	34
8.2.6	Moving in the equation.....	34
8.2.7	Divisions.....	34
8.2.8	Exponents	34
8.2.9	Square root.....	35
8.2.10	Nth square root.....	35
8.2.11	Parenthesized-argument functions	35
8.2.12	User functions	35
8.2.13	Parenthesized terms.....	35
8.2.14	Differentiations	35
8.2.15	Integrations	35
8.2.16	Summations.....	36
8.2.17	Where function 	36
8.3	<i>Manipulating equations</i>	36
8.3.1	Modifying equations with EQW.....	36
8.3.2	Modifications.....	36
8.3.3	Temporary exit to the stack	36
8.3.4	Copy paste functions.....	36
8.3.5	Deletions.....	37
8.3.6	Evaluations	37
8.3.7	Custom program evaluation.....	37
8.4	<i>Cursor mode</i>	37
8.4.1	Switching to cursor mode	37
8.4.2	Movement.....	37
8.4.3	Exiting cursor mode.....	37
8.5	<i>Miscellaneous functions</i>	38
8.5.1	Menu use	38
8.5.2	Zoom	38
8.5.3	-->Grob.....	38
8.6	<i>Error messages</i>	38
8.7	<i>Examples</i>	39
8.8	<i>Key reference for EQW</i>	40
8.8.1	Selection mode	40
8.8.2	Edit mode.....	41
8.8.3	Selection and edit mode - common keys	41
8.8.4	Cursor mode	41
9	MATRIX EDITOR	42
9.1	<i>Presentation</i>	42
9.2	<i>Usage</i>	42
9.3	<i>Reference</i>	43
9.3.1	Keys.....	43
9.3.2	Menu keys	43
10	GROB EDITOR (PICTURE2).....	44

10.1	Usage	44
10.2	Reference.....	46
10.2.1	Commands	46
10.2.2	Keys	47
10.2.3	ALPHA keys.....	48
11	FILER.....	49
11.1	Ports and directories screen.....	49
11.2	File selection screen.....	49
11.3	Menu.....	50
11.4	HEXA Editor	51
11.4.1	Menu	51
11.4.2	Keys	51
11.4.3	Edit mode.....	51

UTILITIES 53

1	FONT EDITOR (EDF)	54
1.1	Font editing.....	54
1.2	CHARS mode.....	54
1.3	A session with CHARS.....	54
2	MACHINE LANGUAGE COMPILER (MASD).....	56
2.1	Generalities on ML (Machine Language).....	56
2.2	Launching Masd.....	56
2.3	Generalities on Masd Syntax.....	56
2.4	Links	57
2.5	Using labels.....	57
2.6	Using constants	58
2.7	Expressions	59
2.8	Skips	59
2.9	Macros	61
2.10	Filename conventions.....	61
2.11	Units	62
2.12	Library creation	62
2.13	SysRPL mode.....	62
2.13.1	Instructions.....	62
a)	Constants	63
b)	External (in entry points table)	63
c)	Tokens	63
d)	Decimal value (System Binary).....	64
e)	Unnamed local variables.....	64
2.13.2	Special keystrokes in the command line.....	65
2.13.3	Program example	65
2.14	SATURN instructions syntax	66
2.14.1	Assigning 0 to a register.....	66
2.14.2	Loading a value in A or C.....	66

2.14.3	Loading a register value into another register	67
2.14.4	Exchange between two registers	67
2.14.5	Addition	67
2.14.6	Subtraction	67
2.14.7	Increment and decrement	67
2.14.8	Right nibbles shifting (divide by 16).....	67
2.14.9	Left nibbles shifting (multiply by 16)	68
2.14.10	Right bit shifting (divide by 2).....	68
2.14.11	Right circular nibble shifting	68
2.14.12	Left circular nibble shifting.....	68
2.14.13	Logical AND.....	68
2.14.14	Logical OR.....	68
2.14.15	Logical NOT	68
2.14.16	Mathematical NOT	68
2.14.17	Loading value into a R Register.....	68
2.14.18	Loading value into A or C from a R register.....	68
2.14.19	Exchange between A or C and a R register.....	68
2.14.20	Memory write (POKE)	69
2.14.21	Memory read (PEEK)	69
2.14.22	D0 and D1 modifications	69
a)	Loading D0 and D1	69
b)	Exchanges between A or C and D0 or D1	69
♦	Loading A or C, field A, into D0 or D1	69
♦	Loading the four low nibbles of A or C into D0 or D1	69
♦	Exchanging A or C, field A, and D0 or D1	69
♦	Exchanging the 4 first nibbles of A or C and D0 or D1	69
c)	Increment and decrement of D0 and D1	69
2.14.23	Working registers tests.....	70
a)	Equality and inequality tests	70
b)	Lower and greater tests	70
c)	Nullity tests.....	70
2.14.24	Working with some bits of A or C register	70
2.14.25	Operations on PC	71
2.14.26	Working with the Hardware Status Register	71
2.14.27	Working with P	71
2.14.28	Jump instructions	71
2.14.29	Exchanges between C and RSTK	71
2.14.30	Input / output instructions	71
2.14.31	Processor control instructions	72
2.14.32	New instructions of Masd	72
2.15	<i>Masd directives</i>	73
2.16	<i>Error messages</i>	73
3	DISASSEMBLER.....	75
4	MISCELLANEOUS UTILITIES	76
4.1	-->H H+	76
4.2	H-> H+	76

4.3	<i>S</i> → <i>H</i> S+H	76
4.4	<i>H</i> → <i>S</i> H+S	76
4.5	<i>CD</i> → CD+	76
4.6	--> <i>CD</i> +CD	76
4.7	--> <i>A</i> +A	76
4.8	<i>A</i> → A+	77
4.9	<i>PEEK</i> PEEK	77
4.10	<i>POKE</i> POKE	77
4.11	<i>SREV</i> SREV	77
4.12	<i>PRG~LST</i> PRG~L	77
4.13	--> <i>RAM</i> +RAM	77
4.14	<i>APEEK</i> APEEK	77
4.15	<i>R~SB</i> R~SB	77
4.16	--> <i>GROB2</i> +GROB	78
4.17	--> <i>HEADER</i> +HEAD	78
4.18	<i>HEADER</i> → HEADE	78
4.19	<i>INPUT2</i> INPUT	78
4.20	<i>DISP2</i> DISPE	78
4.21	<i>HALT2</i> HALTE	78
4.22	--> <i>S2</i> +S2	78
4.23	--> <i>S4</i> +S4	78
4.24	<i>CHOOSE2</i> CHOOS	79
4.25	<i>EDIT</i> EDIT	79
4.26	<i>VISIT</i> VISIT	79
4.27	<i>EDITB</i> EOCTB	79
4.28	<i>VISITB</i> VISIT	79
4.29	<i>EQW</i> EQW	79
4.30	<i>FILER</i> FILER	79
4.31	<i>ASM</i> ASH	79
4.32	<i>ER</i> ER	79
4.33	<i>ASM</i> → ASH+	79
4.34	--> <i>NDISP</i> +NDIS	80
4.35	--> <i>FONT</i> +FONT	80
4.36	<i>FONT</i> → FONT+	80
4.37	<i>FONT8, FONT6</i> FONT8 FONT6	80
4.38	<i>KERNEL?</i> KERNE	80
4.39	<i>SF2, CF2, FS?2, FC?2, FS?C2, FC?C2, STOF2, RCLF2</i> SF2 CF2 FS?2 FC?2 FS?C2 FC?C2 STOF2 RCLF2	80
4.40	<i>SREPL</i> SREPL	80

4.41	<i>AR~LST</i> AR~LS	80
4.42	<i>DIMS</i> DIMS	80
4.43	<i>-->MINIFONT</i> MINIF	81
4.44	<i>MINIFONT-></i> MINIF	81
4.45	<i>FNT2GRB.PRG</i>	81
4.46	<i>GRB2FNT.PRG</i>	81

APPENDIXES 83

1	INTRODUCTION TO ASSEMBLY LANGUAGE	84
1.1	<i>Introduction</i>	84
1.2	<i>What is assembly language (asm) ?</i>	84
1.3	<i>The Saturn processor</i>	84
1.3.1	Generalities	84
1.3.2	Working and saving registers	84
1.3.3	P register	85
1.3.4	Flags register	85
1.3.5	Return stack	85
1.3.6	Memory pointers	85
a)	Memory	85
b)	Pointers	85
c)	Memory accesses	86
1.4	<i>Starting and stopping a Program</i>	86
1.5	<i>Working with the RPL stack</i>	86
1.6	<i>Useful routines</i>	87
1.6.1	Stack exchanges	87
1.6.2	Calculations	88
1.6.3	Miscellaneous	88
1.6.4	Memory operations	88
1.7	<i>Saturn instruction set</i>	89
2	INTRODUCTION TO SYSTEM RPL	94
2.1	<i>Generalities</i>	94
2.2	<i>External "Theory"</i>	94
2.3	<i>Writing a System RPL program</i>	94
2.4	<i>Instructions</i>	95
2.5	<i>Control structures</i>	96
2.5.1	Booleans	96
2.5.2	Tests	96
2.5.3	Loops	96
2.6	<i>Unnamed local variables</i>	96
2.7	<i>Argument control</i>	97
2.8	<i>Entry points list</i>	98
2.8.1	Notations	99
2.8.2	Binary Integers	99
a)	Built-in Binary Integers	99

b)	Binary Integer Manipulation.....	100
♦	Arithmetic Functions.....	100
♦	Conversion Functions.....	101
2.8.3	Character Constants.....	101
2.8.4	Hex & Character Strings.....	102
a)	Character Strings.....	102
b)	Hex Strings.....	103
2.8.5	Real Numbers.....	104
a)	Built-in Reals.....	104
b)	Real Number Functions.....	104
2.8.6	Complex Numbers.....	106
a)	Built-in Complex Numbers.....	106
b)	Conversion Words.....	106
c)	Complex Functions.....	106
2.8.7	Arrays.....	106
2.8.8	Composite Objects.....	107
2.8.9	Tagged Objects.....	108
2.8.10	Unit Objects.....	108
2.8.11	Temporary Variables and Temporary Environments.....	109
2.8.12	Checking Arguments.....	110
a)	Number of Arguments.....	110
b)	Dispatching on Argument Type.....	110
2.8.13	Loop Control Structures.....	111
a)	Indefinite Loops.....	111
b)	Definite Loops.....	112
2.8.14	Error Generation & Trapping.....	112
2.8.15	Test and Control.....	112
a)	Flags and Tests.....	112
♦	General Object Tests.....	113
♦	Binary Integer Comparisons.....	114
♦	Decimal Number Tests.....	114
b)	Words that Operate on the Runstream.....	114
c)	If/Then/Else.....	115
d)	CASE words.....	115
2.8.16	Stack Operations.....	116
2.8.17	Memory Operations.....	117
a)	Temporary Memory.....	117
b)	Variables.....	118
c)	Directories.....	118
d)	The Hidden Directory.....	119
e)	Additional Memory Utilities.....	119
2.8.18	Display Management & Graphics.....	119
a)	Display Organization.....	119
b)	Preparing the Display.....	119
c)	Controlling Display Refresh.....	119
d)	Clearing the Display.....	119
e)	Annunciator Control.....	119



f)	Display Coordinates	120
g)	Displaying Text	120
h)	Graphics Objects.....	120
♦	Graphics Tools.....	120
♦	Grob Dimensions	121
♦	Built-in Grobs	121
♦	Menu Display Utilities	121
i)	Scrolling the Display	121
2.8.19	Keyboard Control.....	122
a)	Key Locations.....	122
b)	Waiting for a Key	122
c)	InputLine	122
d)	The Parameterized Outer Loop.....	123
2.8.20	System Commands.....	123
3	TIPS AND TRICKS	125
3.1	<i>Meta Kernel and UFL</i>	125
3.2	<i>STARTED and EXITED examples</i>	125
3.2.1	Remove the header display	125
3.2.2	Edit compressed files.....	125
3.3	<i>Meta Kernel and Erable</i>	127
3.3.1	STARTOFF and TOFF example	128
4	THE MK FOR PROGRAMMERS.....	129
4.1	<i>Customizing the Filer</i>	129
4.1.1	Overview	129
4.1.2	FILER.CUSTOM Format	129
4.1.3	Explanation of each component.....	129
a)	System Binary 1 and GROB.....	129
b)	System binary 2	129
c)	System binary 3	130
d)	Internal calls	130
e)	Custom program	130
f)	Explanation for the string of addresses	131
g)	Explanations about the name	132
h)	Warning and very important points	132
4.2	<i>Machine Language entry points list</i>	132
4.2.1	MINI_DISP	132
4.2.2	MINI_FONT_ADDRESS.....	132
4.2.3	DISPLAY_SBR.....	132
4.2.4	EDIT_SBR	133
4.2.5	FILER.ADR.....	133
4.2.6	SCAN_FONT	133
4.2.7	RECONFIG	133
4.2.8	DECONFIG.....	133
4.2.9	CMD_SIZE.....	133
4.2.10	GET_PATH	134
4.2.11	DECONFIG_RAM	134

4.2.12	RECONFIG_RAM.....	134
4.2.13	GET_FONT.....	134
4.2.14	INIT_DISPLAY_LINE.....	134
4.2.15	CHANGE_FLAG.....	134
4.2.16	GET_ASCII_KEY.....	134
4.2.17	GET_KEY.....	135
4.2.18	RUN_KEY.....	135
4.2.19	MINI_DISP_VAL.....	135
4.2.20	MINI_DISP_AWP.....	135
4.2.21	TRUP.....	135
4.2.22	TRDN.....	135
4.2.23	ZEROM.....	136
4.2.24	SCAN_KEY.....	136
4.2.25	NEW_ADR.....	136
4.2.26	RESIZE_PLUS.....	136
4.2.27	C=IN3.....	136
4.2.28	A=IN3.....	136
4.2.29	OUT=C=IN3.....	136
4.2.30	OUT=CA=IN3.....	136
4.2.31	DISP_DEC.....	136
4.2.32	MULT_BAC.....	137
4.2.33	GREY?.....	137
4.2.34	BEEP.....	137
4.2.35	KEY_REPEAT.....	137
4.2.36	KEY_NO.....	137
4.2.37	HEX_DEC.....	137
4.2.38	A_DIV_C.....	137
4.2.39	SET_BIT.....	137
4.2.40	CLEAR_BIT.....	137
4.2.41	BIT?.....	138
4.2.42	BZU.SBR.....	138
4.2.43	INV.ZONE.....	138
4.2.44	OFF.SBR.....	138
4.2.45	InitTable.....	138
4.2.46	GetAdr.....	139
4.2.47	GetText.....	139
4.2.48	GetFirst.....	139
4.2.49	GetNext.....	139
4.2.50	GetFirstAdr.....	140
4.2.51	GetNextAdr.....	140
4.2.52	RclPath.....	140
4.3	<i>SysRPL entry points</i>	140
4.3.1	GET.INDEX.....	140
4.3.2	GET.Y.I.....	141
4.3.3	GET.X.Y.I.....	141
4.3.4	PUT.X.Y.....	141
4.3.5	DIMS.....	141

4.3.6	FILER_FNAME	142
4.3.7	FILER_FRCL	142
4.3.8	FILER_NEXTADR	142
4.3.9	SURPRISE	142
4.3.10	INPUT	142
4.3.11	CTRL_LOOP	142
4.3.12	INIT_CMD	142
4.3.13	CMP_PLUS	142
4.3.14	DISP_CALL	142
4.3.15	BZU	143
4.3.16	KER_PARAM	143
4.3.17	MINI_EDITOR	143
4.3.18	KEY_PARAM	143
4.3.19	UPSTACK.ADR	144
4.3.20	WAITKEY.ADR	144
4.3.21	COMMANDLINE.ADR	144
4.3.22	MATRIX.ADR	144
4.3.23	SAVE.ARG	144
4.3.24	LOAD.ARG	144
INDEX		147

Using the Meta Kernel

1 Installation

Switch off your calculator by pressing  .

Put the ROM card into the slot 1 of the HP48 (it must be the port 1)

Turn the HP48 on.

After a short time, the **Meta Kernel** is installed:



2 General presentation

The **Meta Kernel** is an application card that changes the standard environment. A good knowledge of the standard environment is required, as the **MK** uses it, and adds several functions. Please read the HP48 GX User's Guide about it.

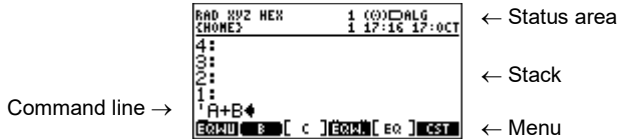
With the new stack display and command line, new utilities are provided:

- Equation editor (EQW)..... Page 32
- Matrix editor (MATR) Page 42
- Grob editor (PICTURE2)..... Page 44
- File utility (FILER) Page 49
- Symbolic assembler (MASD) Page 56
- Library maker..... Page 62
- System RPL Development Kit..... Pages 31, 62, 94

3 Stack display

3.1 Display

The display is divided in three parts: (the command line belongs to the stack)



The stack and status area can be resized in height, see below.

3.1.1 Status area

RAD; **RZ**; **RAD**; **RZ**; **RAD**; **RZ**; **RAD**; **RZ**; **(O)**; **#**; **(O)**; **#**; **(O)**; **#**; **(O)**; **#**; **(O)**; **#** The status area shows the current HP48 GX state.

Error messages, current path, calculator modes are displayed in this area.

Here are the new state flags:

DEG : Degrees	ALG : Algebraic entry mode	HEX : Hexadecimal base
RAD : Radians	PRG : Program entry mode	DEC : Decimal base
GRD : Grads	(O) : Beeper	OCT : Octal base
RPN : Rectangular coordinates	HLT : A program has been halted	BIN : Binary base
RPR : Polar coordinates	USR : User keyboard	(O) : ROM or write-protected RAM in port 2
RSP : Spherical coordinates	UVS : User keyboard for the next keystroke	(■) : Read-write RAM card in port 2

The status area occupies two text lines by default. Its height can be changed to one or zero lines with the **MK** command **→HEADER**, consult Page 78.

3.1.2 Stack

With the **MK**, up to nine lines can be viewed on the stack, as you can change the font height (6,7 or 8 pixels) and the status area height.

The maximum number of lines displayed by objects can be changed with **→NDISP**, consult Page 80.

More configuration capabilities are provided by the **Meta Kernel** system flags (**MKSF**), consult Page 21.

Algebraics and grobs may be displayed directly in the stack, use the **MKSF** to configure how and when they are displayed.



3.1.3 Menu labels

Menu-label style depends on the type of the stored object:

```

RAD XYZ HEX 1 (0)
^E EQUNUTY 16 17 17:0CT
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

DIR Répertoire	PROG Programme, Code
GROB Grob	[REAL] Réel, complexe, expression algébrique, matrice
LIST Liste, variable, entier binaire	STRIN Chaîne de caractères

By default, labels are written using the **MK** mini font. If the **MK** system flag `-20` is set (with `-20 SF2`), standard menu font is used instead:

```
DIR GROB LIST PROG [REAL] STRIN
```

3.2 Display configuration

New display fonts can be created and used. The number of lines displayed on the stack differs with the font height.

With a font8 (8 pixels height), 5 to 7 lines may be viewed; with a font7, 6 to 8 lines ; with a font6, 7 to 9 lines.

3.2.1 Changing the display font

To use a different font, put it on the stack, and execute `→FONT`, consult Page 80.

Fonts are displayed on the stack like this:

```

RAD XYZ HEX 1 (0)
^E EQUNUTY 16:24 17:0CT
5: Ft8_0:SYSTEM 8
4: Ft6_0:SYSTEM 6
3: Ft6_6:Robot6
2: Ft8_4:Minitel 8
1: Ft8_1:Carre 8
ROBOT ROBOT MINIT CARRE IOPAR EQUWU

```

Example: `Ft8_0:SYSTEM 8`, this is the standard display font for the **MK**.

The first number (**8**) is the font height in pixels, the second number (**0**) is the id of the font, then the name (`SYSTEM 8`).

Two examples with a font8 and a font6:

```

RAD XYZ HEX 1 (0)
^E EQUNUTY 1 14:30 29:JUL
1: "% GROB.SRC
GOSUBL EFFFACE
* REPLY
ST=0 5 ST=0 9 ST=1
"FONT 8
GROB [PCSR] SUBS: START MENU PUTIL

```

```

RAD XYZ HEX 1 (0)
^E EQUNUTY 1 14:29 29:JUL
1: "% GROB.SRC
GOSUBL EFFFACE
* REPLY
ST=0 5 ST=0 9 ST=1
"FONT 6
GROB [PCSR] SUBS: START MENU PUTIL

```


If the current font is a font6 (6 pixel-high), the **MK** system flags `-8` and `-9` force respectively the stack display and the command line to use the mini font:

```

RAD XYZ HEX 1 (0)
^E EQUNUTY 1 14 31 29:JUL
1: "% GROB.SRC
GOSUBL EFFFACE
* REPLY
ST=0 5 ST=0 9 ST=1 11 ST=0 14..
GOSUBL PXY
"MINI FONT !!!4
GROB [PCSR] SUBS: START MENU PUTIL

```

3.2.2 Modifying the current font

The current display font may be modified with the CHARS menu () . Consult Page 54.

On the floppy disk, there are two programs, named GRB2FNT.PRG and FNT2GRB.PRG, which convert **MK** fonts to or from common 8x2048 grobs. Consult Page 81.

3.2.3 Changing the mini font

It is possible to change the **MK** mini font, used in the status area and the menu labels.

Use the commands **MINIFONT→** and **→MINIFONT**. A mini font is a **Library Data** object. This format is very popular in the HP48's world, in particularly with programs which use UFL (Universal Font Library). In order to know how to use the UFL with the **Meta Kernel** check the column Tricks and Tips.

4 Configuration

4.1 MK variables

Some programs can be executed on some events, in place of the default programs. They must be stored in the **HOME** directory.

STARTUP	Executed at each reboot (ON) - (<input type="checkbox"/> c...).
STARTED	Executed just before entering in command line for editing an object. The object to be edited can be found on stack level 1. This program must leave the object to be edited on the stack level 1.
EXITED	Executed when the user stops the command line editing. Level 2 contains the object and level 1 TRUE if the user has pressed ENTER , or only FALSE for ON . For example, these two programs can be used to change the display while editing (try STARTED: * 0 →HEADER * and EXITED: * 2 →HEADER *).
STARTOFF	Executed when the automatic OFF is called, after a certain amount of idle time (You can make a screen saver! But remember the HP48 GX works on batteries...).
TOFF	Defines the amount of idle time before an automatic OFF (or STARTOFF). It must be a binary integer, in Ticks (1 s = 8192 Ticks).
STARTTEXT	Contains a string that is the path (using the MASD syntax) to the entries points list. Example: "4/DIR1/TABLE.EXT @" will refer to the table TABLE.EXT in Port 4, in the backup DIR1 For more details, please read page 62.
STARTSEND	Used by the filer for file transfer. By default, Kermit is used. If STARTSEND exists, it is called with the object on level 2 and the name on level 1. Consult page 49.
STARTERR	Used for the error messages display. The error is found in two character strings on levels 2 and 1.
STARTEQW	Called by EQW (equation editor) when CST is pressed.
FILER.CUSTOM	Called by the Filer when CST is pressed.

4.2 MK user and system flags (MKUF, MKSF)












The **Meta Kernel** adds a new series of flags.

Use **CF2**, **SF2**, **FC?2**, **FS?2**, **FC?C2**, **FS?C2** like the functions **CF**, **SF** ...

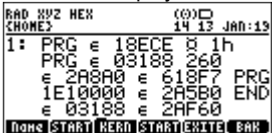







By default, all flags are cleared.

MK System flags -1 to -64 are used by the **MK** to change parameters.

MK User flags 1 to 64 can be used by any program that runs under the **MK**.

Flag	Cleared (CF2)	Set (SF2)
-1	Multiline for stack display at all levels. 	No multiline for stack display (except level 1). 
-2	Strings displayed on multiple lines. 	Strings displayed on only one line. 
-3	Digital watch in status area. 	Analog watch in status area. 
-4	No auto-indentation in command line editing.	Auto-indentation in command line editing.
-5	Cursor can not go out of the text.	Full page editing.
-6	Strings take only one line for →GROB2.	Multi-lines strings for →GROB2.
-7	ASM→ with addresses.	ASM→ without the addresses (makes labels).
-8	The current font is normally used in stack display. 	Mini-font in stack display (the current font must be a font6). 
-9	The current font is normally used in command line editing.	Mini-font in command line editing (the current font must be a font6). 
-10	Rightmost stack display format. 	Leftmost stack display format. 
-11	Silent keystrokes.	Keystrokes click.
-12	Purge confirmation in the Filer.	No purge confirmation in the Filer.
-13	(Reserved for internal use).	

-14	(Reserved for internal use).	
-15	On-stack algebraics display.	Standard stack display for algebraics.
-16	Current font for EQW stack display.	Mini-font for EQW stack display.
-17	Current font for algebraic →GROB2.	Mini-font for algebraic →GROB2.
-18	Current font for algebraics editing in EQW.	Mini-font for algebraics editing in EQW.
-19	On-stack grobs display.	Standard stack display for grobs.
-20	MK Mini-font menu labels.	Standard menu labels.
-21	Normal stack display	Display the stack using the HP SysRPL syntax
-22	→S2 adds " !NO CODE<CR> !RPL<CR>" at the beginning of the source, and "␣" at the end. Therefore it can be use directly by MASD	→S2 just convert an object into SystemRPL without adding any header.

<p>-23</p>	<p>The stack display is recursiv. When an object contains a reference to another object, the stack displays it.</p> 	<p>The stack display is not recursiv.</p> 
<p>-24</p>	<p>Object display in the command line is not recursiv.</p> 	<p>Object display in the command line is recursiv (See flag -23)</p> 
<p>-25</p>	<p>In the normal stack display (-21 CF2), displays the addresses of all the unknown objects:</p> 	<p>If the address is in the Entries point table, then use the mnemonic:</p> 
<p>-26</p>	<p>The CHOOSE2 boxes use the current font</p> 	<p>The CHOOSE2 boxes use the mini-font.</p> 
<p>-27</p>	<p>The symbolic matrix writer will give an ARRAY only for the array of real and complex</p>	<p>Matrix Writer gives an array of ? for any object, each time the matrix contains objects of the same type. But beware of memory lost if you try to use a RPL command on it.</p>
<p>-28 à -64</p>	<p>Not used, reserved for future versions.</p>	

5 Interactive stack

The interactive stack works the same way as the standard one, but at a greater speed, and it is possible to edit many objects simultaneously: in the command line editor, it is possible to go temporarily to the interactive stack, and to edit another object recursively. It is also possible to **HALT** an edition.

```

RAD XYZ HEX          (0)
<HOME>              13:53 17:0CT
000000000000000000 4338
000000000000000000 4339
000000000000000000 4400
000000000000000000 4401
000000000000000000 4402
000000000000000000 4403
000000000000000000 4404
ECHO INFO PICK ROLL ROLLD +LIST
  
```

Current stack level →

Interactive Stack Operations: (*n* indicates the current level)

ECHO	Copies the current level object on the command line, at the cursor position.
INFO	Shows information about the current level object (size, CRC...).
↶ INFO	Edits the current level object with the best environment (EDITB).
↷ INFO	Edits the object whose name is on the current level, with the best environment (VISITB).
PICK	Equivalent to <i>n</i> PICK .
ROLL	Equivalent to <i>n</i> ROLL .
ROLLD	Equivalent to <i>n</i> ROLLD .
+LIST	Equivalent to <i>n</i> +LIST .
DUPN	Equivalent to <i>n</i> DUPN .
DROPN	Equivalent to <i>n</i> DROPN .
LEVEL	Puts the number of the current level on level 1.
GOTO	Goes directly to a given level.
KEEP	Clears all the levels upper than the current level (Therefore it KEEPS the first stack levels up to the current one).
SLOW / FAST	Selects the cursor speed.

6 Command line

6.1 Generalities

The **Meta Kernel** improves editing with new functions: COPY/PASTE, FIND/REPLACE, text style (Bold, italic, underline, inverse).





Styles can only be used inside strings (they will be ignored and lost in any other object).

You can edit an object by pressing  ^{EDIT}  when the object is on stack level 1. There are other ways to edit an object (explained after).

6.2 Editing menu


There are four pages in the menu, plus the Style menu.

6.2.1 Major options

	Goes to a given line number.
 GOTO	Goes to a given position (in number of characters from the beginning).
 GOTO	Displays the menu of the text, which is composed of every line that begins with the character *, useful in assembly language.
+BEG	Goes to the beginning of the selection.
+END	Goes to the end of the selection.
DEL L	Clears the current line.
SLOW / FAST	Selects the cursor speed.
INFO	Displays information about the current text.
 INFO	Edits the selection with the best environment (EDITB).

6.2.2 Operations on the selection

Note: the selection marks are lost when the text is modified.

BEGIN	Sets the beginning of the selection.
END	Sets the end of the selection.
CUT	Copies the selection on the stack level 1, and clears it in the text.
COPY	Copies the selection on the stack level 1.
PASTE	Puts the object on the stack level 1 into the text at the cursor position.
 PASTE	Puts the object on the stack level 1 into the text at the cursor position, and deletes it from the stack.
DEL	Clears the selection.

6.2.3 Search operations

FIND	Displays a screen where you can type the search string. When you press enter, this string is put on stack level 1. The character ? stands for any
-------------	---

	character. Then the selection is placed on the first occurrence of this string after the cursor position.
REPL	Displays a screen where you can type the search string and the replace string. The search string is put on stack level 1, and the replace string on level 2. Then the selection is placed on the first occurrence of the search string. The change is not done yet !
NEXT	Selects the next occurrence of the search string (the string on stack level 1).
R	Replaces the selection by the replace string (the string on stack level 2).
R/N	Equivalent to R , then NEXT . So it makes one change, and goes to the next occurrence of the search string.
ALL	Equivalent to R and NEXT , repeated until the end of the text. So every occurrence of the search string is replaced by the replace string, from the cursor position to the end of the text, without confirmation.

6.2.4 Miscellaneous

←WORD	Goes to the beginning of the previous word.
→WORD	Goes to the beginning of the next word.
+DEL	Clears the text between the cursor and the beginning of the previous word.
DEL+	Clears the text between the cursor and the beginning of the next word.
STYLE	Displays the style menu.
+STK	Goes to the interactive stack. Consult Page 25.
↶ +STK	EVAL uates the selection and replaces it with the result. E.g. if 20 SIN is selected, it will be replaced by .342020143326 .
↷ +STK	Equivalent to HALT . The current editing is suspended, you can do what you want (except a reboot) and get back to the text with ↶ ^{CONT} ON .

6.3 Styles

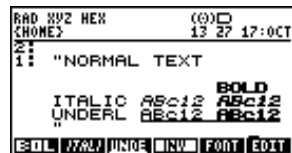
6.3.1 Generalities

Styles can be used inside text strings. There are four styles: Bold, italic, underlined, inversed.

Several styles can be used on the same portion of text.

More than one font can also be used in the same text (but they must have the same height).

6.3.2 Style menu



ESC	Sets/clears the bold style.
------------	------------------------------------

ITAL	Sets/clears the <i>italic</i> style.
UNDE	Sets/clears the <u>underlined</u> style.
INVS	Sets/clears the inverse style.
FONT	Displays the font list.
EDIT	Gets back to the Edit menu.

6.3.3 Using styles

You can modify the style of the current selected text by pressing a style menu key. If there is no selection, the text you will then type will be of the chosen style (If you move the cursor, the style will be the one under the cursor).

6.3.4 Using fonts

If you press the **FONT** menu key, a choose box shows all the fonts which have the same height as the current editing font.

If there is a selection, the font will be applied on this selection, otherwise it will be used for the text you will then type (If you move the cursor, the font will be the one under the cursor).

The **Meta Kernel** can use up to 247 fonts inside one text.

For a font to be recognized by the **MK** (so that it will appear in the **FONT** choose box), it just has to be saved in the **HOME** directory, or in a port (0, 2 to 31)

If several fonts use the same ID, the **MK** only takes the first one, in the order of the search (**HOME**, then 0, 2 ... 31). ID 0 is reserved for the system font. ID > 247 will be ignored.

Example:

Type the text: **THIS IS A SIMPLE EXAMPLE**

```

RAD HWZ HEX      2 (0)
<HOME>          2 13 40 17:0CT
3:
2:
1:
"THIS IS A SIMPLE
EXAMPLE"
IOPAR B [ c ]EOM:3[ E0 ]CST

```

Put the cursor on the 11th character (S in SIMPLE). Press the menu key **BEGIN**

```

RAD HWZ HEX      2 (0)
<HOME>          1 13 41 17:0CT
3:
2:
1:
"THIS IS A ◀SIMPLE
EXAMPLE"
BEGIN END CUT COPY PASTE DEL

```

Press **▶** six times. Press **END**

```

RAD HWZ HEX      2 (0)
<HOME>          1 13 42 17:0CT
3:
2:
1:
"THIS IS A SIMPLE◀
EXAMPLE"
BEGIN END CUT COPY PASTE DEL

```

Go to the **Style** menu and press **END**

```
RAD SVZ HEX      3 (9)
<HOME>          1 13:42 17:0CT
0:
1:
2:
3:
4:
5:
6:
7:
8:
9:
A:
B:
C:
D:
E:
F:
G:
H:
I:
J:
K:
L:
M:
N:
O:
P:
Q:
R:
S:
T:
U:
V:
W:
X:
Y:
Z:
[THIS IS A ◀▶◁▷◀▶◁▷
EXAMPLE"
[EDIT F7:7Z F10:0 F11:0 F12:0 F13:0 F14:0 F15:0 F16:0 F17:0 F18:0 F19:0 F20:0 F21:0 F22:0 F23:0 F24:0 F25:0 F26:0 F27:0 F28:0 F29:0 F30:0 F31:0 F32:0 F33:0 F34:0 F35:0 F36:0 F37:0 F38:0 F39:0 F40:0 F41:0 F42:0 F43:0 F44:0 F45:0 F46:0 F47:0 F48:0 F49:0 F50:0 F51:0 F52:0 F53:0 F54:0 F55:0 F56:0 F57:0 F58:0 F59:0 F60:0 F61:0 F62:0 F63:0 F64:0 F65:0 F66:0 F67:0 F68:0 F69:0 F70:0 F71:0 F72:0 F73:0 F74:0 F75:0 F76:0 F77:0 F78:0 F79:0 F80:0 F81:0 F82:0 F83:0 F84:0 F85:0 F86:0 F87:0 F88:0 F89:0 F90:0 F91:0 F92:0 F93:0 F94:0 F95:0 F96:0 F97:0 F98:0 F99:0 F100:0
```

Now, select the word **EXAMPLE**

```
RAD SVZ HEX      3 (9)
<HOME>          2 13:43 17:0CT
0:
1:
2:
3:
4:
5:
6:
7:
8:
9:
A:
B:
C:
D:
E:
F:
G:
H:
I:
J:
K:
L:
M:
N:
O:
P:
Q:
R:
S:
T:
U:
V:
W:
X:
Y:
Z:
[THIS IS A SIMPLE
EXAMPLE"
[EDIT F7:7Z F10:0 F11:0 F12:0 F13:0 F14:0 F15:0 F16:0 F17:0 F18:0 F19:0 F20:0 F21:0 F22:0 F23:0 F24:0 F25:0 F26:0 F27:0 F28:0 F29:0 F30:0 F31:0 F32:0 F33:0 F34:0 F35:0 F36:0 F37:0 F38:0 F39:0 F40:0 F41:0 F42:0 F43:0 F44:0 F45:0 F46:0 F47:0 F48:0 F49:0 F50:0 F51:0 F52:0 F53:0 F54:0 F55:0 F56:0 F57:0 F58:0 F59:0 F60:0 F61:0 F62:0 F63:0 F64:0 F65:0 F66:0 F67:0 F68:0 F69:0 F70:0 F71:0 F72:0 F73:0 F74:0 F75:0 F76:0 F77:0 F78:0 F79:0 F80:0 F81:0 F82:0 F83:0 F84:0 F85:0 F86:0 F87:0 F88:0 F89:0 F90:0 F91:0 F92:0 F93:0 F94:0 F95:0 F96:0 F97:0 F98:0 F99:0 F100:0
```

Go to the **Style** menu and press **FONT** (There may be other fonts than shown below)

```
RAD          Choose the font
<HOME>
0:          SYSTEM 8
1:          Robot 8
2:          Minitel 8
3:          Curative 1
4:          Carre 8
[EDIT F7:7Z F10:0 F11:0 F12:0 F13:0 F14:0 F15:0 F16:0 F17:0 F18:0 F19:0 F20:0 F21:0 F22:0 F23:0 F24:0 F25:0 F26:0 F27:0 F28:0 F29:0 F30:0 F31:0 F32:0 F33:0 F34:0 F35:0 F36:0 F37:0 F38:0 F39:0 F40:0 F41:0 F42:0 F43:0 F44:0 F45:0 F46:0 F47:0 F48:0 F49:0 F50:0 F51:0 F52:0 F53:0 F54:0 F55:0 F56:0 F57:0 F58:0 F59:0 F60:0 F61:0 F62:0 F63:0 F64:0 F65:0 F66:0 F67:0 F68:0 F69:0 F70:0 F71:0 F72:0 F73:0 F74:0 F75:0 F76:0 F77:0 F78:0 F79:0 F80:0 F81:0 F82:0 F83:0 F84:0 F85:0 F86:0 F87:0 F88:0 F89:0 F90:0 F91:0 F92:0 F93:0 F94:0 F95:0 F96:0 F97:0 F98:0 F99:0 F100:0
```

Choose the font you want (here **ROBOT 8**) with the up and down arrows and press **ENTER**.

```
RAD SVZ HEX      3 (9)
<HOME>          2 13:44 17:0CT
0:
1:
2:
3:
4:
5:
6:
7:
8:
9:
A:
B:
C:
D:
E:
F:
G:
H:
I:
J:
K:
L:
M:
N:
O:
P:
Q:
R:
S:
T:
U:
V:
W:
X:
Y:
Z:
[THIS IS A SIMPLE
◀▶◁▷◀▶◁▷"
[EDIT F7:7Z F10:0 F11:0 F12:0 F13:0 F14:0 F15:0 F16:0 F17:0 F18:0 F19:0 F20:0 F21:0 F22:0 F23:0 F24:0 F25:0 F26:0 F27:0 F28:0 F29:0 F30:0 F31:0 F32:0 F33:0 F34:0 F35:0 F36:0 F37:0 F38:0 F39:0 F40:0 F41:0 F42:0 F43:0 F44:0 F45:0 F46:0 F47:0 F48:0 F49:0 F50:0 F51:0 F52:0 F53:0 F54:0 F55:0 F56:0 F57:0 F58:0 F59:0 F60:0 F61:0 F62:0 F63:0 F64:0 F65:0 F66:0 F67:0 F68:0 F69:0 F70:0 F71:0 F72:0 F73:0 F74:0 F75:0 F76:0 F77:0 F78:0 F79:0 F80:0 F81:0 F82:0 F83:0 F84:0 F85:0 F86:0 F87:0 F88:0 F89:0 F90:0 F91:0 F92:0 F93:0 F94:0 F95:0 F96:0 F97:0 F98:0 F99:0 F100:0
```

6.4 Miscellaneous command line options

6.4.1 Full screen mode

By default, if you press the **▶** key at the end of a line, the cursor will be placed on the first character of the next line. With the **MK**, it is possible to go 'out' of the lines, with the **MK** system flag -5.

Type **-5 SF2** to enable the full screen mode (if you like this mode, a good idea is to place a **-5 SF2** in the program **STARTUP**).

If you go beyond the end of a line and press a character key, spaces will automatically be inserted before the character you type.

6.4.2 Auto-indent mode

By default, when you press **▶** **←** to insert a carriage return, the cursor is then always placed on the first column of the display. The **MK** implements the auto-indent mode, so that when you type a carriage return, the cursor will be placed under the first character of the upper line (Useful when programming).

Type **-4 SF2** to enable auto-indent mode.

6.4.3 Special keystokes for System RPL

In the command line, some keystokes help to type characters that are specific to System RPL, and also to easily find System RPL commands only by typing the first letters of the command.

These keystrokes are detailed on page 65.

6.4.4 Big strings editing

As specified in the HP48 GX User's Manual, you can not type character strings that are bigger than the half of the free memory.

The **MK** uses a new memory handler, so that when you edit strings (and only strings), you can have more characters than the half of the free memory, and still press **ENTER**, because the **MK** does not evaluate the string, but places it directly on the stack. You will never get an Insufficient Memory when validating a character string.

For other objects types, the **MK** built-in **→STR** will usually works if there is more or less as much free memory as the object size, so the object may be viewed. But when **ENTER** is pressed, the **STR→** may have Insufficient Memory.

7 Command library

A command library is a special library which extends the STR→ function. The MK command library adds new keywords (they can be typed on the command line and used in programs):

TRUE	External boolean true.
FALSE	External boolean false.
\$ binary	External of the given address. E.g. \$ 3188h (external DUP). To type \$, press .
€ Hexa	External of the given address. € 3188 is equivalent to \$ 3188h. To type €, press €.
⌘ binary	System binary of the given number. To type , press .
K\$ number	Character of the given ASCII code.
XLIB LibNumber PrgNumber	Program #PrgNumber of the library #LibNumber.
INCLUDE VarName	Includes a given variable content in the line.
PRG	External program beginning (\$ 02D9Dh), a END must close the program.
~	NoEval (\$ 06E97h), which causes the next object not to be evaluated. To type ~, press .
PROG	~ PRG (To edit an external program, begin it with PROG, so it won't be evaluated when is pressed).
CODE Hexa	Creates a CODE object with the following binary data. Length is calculated.
OBJ Hexa	Creates an object with the following binary data.
GREY X Y Hexa	Creates a four-grayscale grob XxY.
♠ Name	Puts the address of the corresponding name, if it exists in the entry points table. Consult page 63. To type ♠, press ♠.

8 Equation editor

The program EQW can easily create, edit and manipulate algebraic objects, in their hand-written style.

For example, here is a mathematical equation:

$$E = \int_0^{\frac{1}{X}} |F(t)| dt$$

Here is how it usually appears on the stack:

$$E = f(0, 1/X, ABS(F(t)), t)$$

And now, here is the same equation typed with EQW:

This EQW display is also used in the stack display:

8.1 First view of EQW

In this document, 'equation' stands for all algebraic objects.

8.1.1 Introduction

EQW is a special environment in which the keyboard is redefined and limited to specific operations.

8.1.2 Modes

EQW provides three modes of operation:

- Selection mode

or

- Editing mode

In most cases, selection and edit modes are used to enter equations.

- Cursor mode, used to navigate rapidly through the equation.

8.1.3 Editing

During editing, keys behave in different ways, depending on the current mode.



In selection mode, a part of the equation is inverted or inboxed. If a function key is pushed, the function is applied on the selection.

In edit mode, a flashing cursor is ready to insert new characters.

The next sections explain how to enter and edit an equation.

8.2 Creating equations

8.2.1 Running EQW

Type   **ENTER** to run EQW with a new equation.

8.2.2 Exiting EQW

To validate the current equation on the stack, press **ENTER**.





To get out of EQW, and lose all changes made on the equation, press **ON**.

There may be some times when EQW can't draw the equation as fast as you type, but all keystrokes are processed, none are lost. In this case, EQW draws the equation only when there is enough time to do so.

8.2.3 Numbers and names

Type real numbers and names the same way as in the command line.

In edit mode (with the flashing cursor), the character α is directly available by pressing **DEL**.



Note: in EQW, the  key locks the alpha mode by pressing it once (corresponding to **-60 SF**). Therefore, to type a name, either hold  and press the character keys, or press  once, then press the character keys and press  to unlock the alpha mode.

8.2.4 Additions, subtractions, and multiplications



To enter +, - and \cdot , press ,  and .

The operation is applied on the edited number or name, or the current selection.

To enter implicit multiply, do not press . An implicit multiply is automatically inserted in the following cases:

- A number followed by an alpha character or a prefix function (a function whose name is before arguments, for example  .
- An alpha character followed by a prefix function.








8.2.5 Complex numbers and expressions

To enter a complex number or expression, type in the real part, then   and then the imaginary part.

A complex expression is considered as a two-argument function (equivalent to +).





A complex number is stored directly in the final equation, whereas a complex expression like $\langle A, B \rangle$ is converted to $A+B*i$.

8.2.6 Moving in the equation




Before going further, here are the movement keys. In fact, entering an equation with EQW is not far from the RPN way: no parenthesis are needed. For example, to type $\langle X+5 \rangle \cdot 2$, press  -  \times  , then select $X+5$ by typing , and then type  . See the examples at the end of this chapter to handle more cases.

Arrow keys move the selection. They behave different ways, depending on the type of selection.

Here are the movement keys when the selection is inverted:


-  selects the first argument of the current function, or if the selection is already a number or a name, then selection becomes a box.
-  selects the upper function of the selection.
-  and  select the previous or next argument of the upper function.

Here are the movement keys when the selection is in a box:

-  goes back into inverted selection mode.
-  and  select the previous or next number or name, in hierarchical order. These keys are useful to rapidly move around the whole equation.

To rapidly select a part of the equation, even far from the selection, use the cursor mode (see the Cursor Mode section).

8.2.7 Divisions



Press  to insert a division.

Division is applied on the edited object or the current selection.



To 'go out' of the division, use  or  until the whole division is selected.

8.2.8 Exponents







 begins the exponent on the edited object or the selection.

Use  or  to go out of the exponent.




8.2.9 Square root

- In selection mode,  applies a square root on the selection.
- In edit mode,  inserts a square root, eventually with an implied multiply.

8.2.10 Nth square root

- In selection mode,   places the selection under the root. Press  and type the outside term. To exchange the two arguments, read the Modifications section.
- In edit mode,   inserts a root. Type the outside term. Press  and then type the inside expression.

8.2.11 Parenthesized-argument functions

- In selection mode, press the function key to apply it on the selection. If the function needs more than one argument, the selection is used as the first one. press  to go and type the next arguments.
- In edit mode, press the function key or type its name followed by  , and then the arguments.

8.2.12 User functions







These functions are created by the user and are not handled directly by the HP 48GX.

Type the function name, press   and type the first argument.
To add an argument, press .

8.2.13 Parenthesized terms





EQW draws equations with the fewest possible parentheses. To add explicit parentheses around the selection, press  .




8.2.14 Differentiations

- In selection mode, press   to insert a differentiate sign, the selection is the function to be differentiated. Press  and type the variable of differentiation.
- In edit mode, press   to insert a differentiate sign, type the variable of differentiation, press  and type the equation to be differentiated.








See the Evaluations section to discover how to differentiate a function without exiting from EQW.

8.2.15 Integrations









- In selection mode, press   to add an integral sign. The current selection is the equation to be integrated. Press  and  to move around the integral arguments and type them.

- In edit mode, press   to insert an integral sign. Type the arguments, press  to move to the next one.

8.2.16 Summations

- In selection mode, press   to add a summation sign. The selection is the function to be summed. Press  and  to move around the arguments and type them.
- In edit mode, press   to insert a summation sign. Type the arguments, press  to move to the next one.

8.2.17 Where function |



- In selection mode, press    to add a where function. The current selection is the base expression. Press  to move to the next argument (the variable and its value).
- In edit mode, press    to insert a where sign. Type the arguments, press  to move to the next one.

8.3 Manipulating equations

8.3.1 Modifying equations with EQW





To edit an equation with EQW, press  when the equation is on level 1 of the stack.

8.3.2 Modifications

To edit a variable name or a bigger expression with the command line editor, select it and press  .

It can also be edited with HP's EQUATION WRITER by typing  .

8.3.3 Temporary exit to the stack

It is possible to go temporarily to the stack by pressing  . Press   again to get back to EQW, in the same state as before halting.








8.3.4 Copy paste functions

Copy and paste functions are available. SUB copies the selected expression on the stack. REPL pastes the expression on level 1 either onto the selection, or where the edit cursor is. These commands are accessible in the menu or on keyboard:

Sub :  
Repl :  

8.3.5 Deletions

There are different ways to delete a part of an equation.

- In selection mode, typing numbers or characters suppresses the current selection. To enter a new expression that does not begin with a number or a name, pressing   deletes the selection and goes into edit mode.
- To delete a one-argument function, just select this function and press .
- To delete one argument of a two-argument function, select it and press 
.
- Press   to delete all arguments but the one selected.

8.3.6 Evaluations


Selected expressions can be processed by RPL commands **EVAL** , **COLCT** and **EXPAN** . The result replaces the old selection. These commands are accessible through menus or on keyboard:

 : **EVAL** ,

 :  ,

 :  .

8.3.7 Custom program evaluation


An external program may be applied on the current selection. By pressing , the program **STARTEQM** situated in the **HOMEDIR** is called, with the current selection on the stack level 1. This program must return the new expression on the stack level 1.

For example, the following program transforms all the **INV** functions by **1/** :



```
« { 'INV(&X)' '1/ &X' } 1MATCH DROP »
```

8.4 Cursor mode

8.4.1 Switching to cursor mode

To get into cursor mode, press .



8.4.2 Movement

Arrow keys direct the cursor movement. The movement is faster when  is held. To go directly to a border, hold  and press one arrow key.

When the cursor stops, a box is drawn around the underlying expression.

8.4.3 Exiting cursor mode

There are two ways to return to selection mode.

 returns to the same state prior to entering in cursor mode, whereas  makes the expression under the cursor (which appears in a box) the new selection. This is a fast way to select a particular part of an equation.

8.5 Miscellaneous functions

8.5.1 Menu use

To show the menu, press a menu key, or **NXT**. The menu may be hidden by pressing **→** **MENU** **NXT**. When switching to cursor mode, the menu is automatically hidden.

8.5.2 Zoom

EQW can draw the equation using the system mini-font, by pressing **PRG** or **Zoom**. This key can be used in any mode. A second keystroke returns to the stack font.

8.5.3 →GroB

The current selection may be pushed onto the stack as a graphic object by pressing **STO** or **→GroB**. This function is the same as the RPL command **→GROB2**.

8.6 Error messages

An error message remains displayed as long as a key is pressed, to keep from destroying the expression.

Incomplete Expression	One cannot exit from EQW, or perform certain functions when the equation is not complete.
Global Expected	For some functions (e.g. differentiate, integrate), one of the arguments must be a variable name.
First Argument Missing	Two-argument functions must be typed after the first argument.
Two Arguments Expected	← CLEAR DEL (One argument delete) must be applied on a two-arguments or user function.
Edit Mode	RPL commands (EVAL, COLCT...) require selection mode.
1 Arg Function Needed	DEL must be used on a one-argument function like SIN.
Select An Argument	→ CLEAR DEL can not be used on the whole equation.
Algebraic Expected	To paste an expression, the first level of the stack must contain an algebraic object.
Too Few Arguments	The stack is empty.
Command Forbidden	RPL commands like SWAP, DUP... are forbidden in equations.

8.7 Examples

$$E = \int_0^X |F(t)| dt$$

Press:

$E = \int \langle$ α $-$ \square_E \leftarrow $=$ 0 \rightarrow \cos
 $0, 1/X$ 0 \rightarrow 1 \div DEL \rightarrow
 $\text{ABS} \langle$ α $-$ \square_A \square_B SIN \leftarrow $\left(\frac{\quad}{\quad}\right)$
 $F \langle T$ α \square_F \leftarrow $\left(\frac{\quad}{\quad}\right)$ α $-$ \cos T \rightarrow
 dT α \cos T

$$E = \int_0^X |F(T)| dT$$

ENTER

1: E= \int (0, 1/X, ABS(F(T)), T)

$$\frac{\partial}{\partial x} \left(\frac{\cos x}{\sin^2 x} \right)$$

Type the expression:

$\text{COS} \langle X$ \cos DEL \rightarrow
 $\swarrow \text{SIN} \langle X$ \div SIN DEL
 $\wedge 2$ \rightarrow y^x 2
 ∂ \rightarrow \uparrow \rightarrow SIN DEL \uparrow \uparrow

$$\frac{\partial}{\partial X} \left(\frac{\text{COS}(X)}{\text{SIN}(X)^2} \right)$$

Save it on the stack:

MTH **NXT** **Sub**

Differentiate it:

EVAL **EVAL** **EVAL** **EVAL**

Simplify it:

Expan Expan Colct

Rebuild the expression:



Change INV into 1/:



8.8 Key reference for EQW

8.8.1 Selection mode

	Inverted selection	Boxed selection
	Previous argument.	Previous variable or number.
	Next argument (if on last argument, go to upper function).	Next variable or number (if on last argument, go to upper function).
	Upper function.	Inverted selection.
	First argument of the selected function.	
	First argument.	
	Last argument.	
	Argument left roll.	
	Argument right roll.	
	Whole equation selection.	
	First name or number of the selection, boxed selection.	
	SUB , copies the selection to the stack.	
	EXPAND s the selection.	
	COLCT s the selection.	
EQW	EVAL uates the selection.	
STO	→GROB on the selection.	
	Line command editing.	
	HP's Equation Writer editing.	
	Adds/removes parentheses.	
DEL	One-argument function deletion.	
	Argument deletion in a two-argument function or in a user function.	
	Deletes all but the selection.	
	Variable or name edition.	
	Clears selection, goes into edit mode.	
SIN , COS , TAN	Inserts a one-argument function.	
	Inserts a function, the selection becomes the first argument (except for \int and Σ where the selection becomes the main argument).	

SPC	Next argument, or create another argument in a user function.
	Creates a complex number, selection is the real part.
Digit, character	Deletes the selection, starts editing.

8.8.2 Edit mode

Digit, character	Adds a character; a number followed by an alpha inserts an implicit multiply.
DEL	Fast access to the character \times (α not needed).
	Creates user function.
SPC	Next argument, or creates another argument in a user function.

8.8.3 Selection and edit mode - common keys

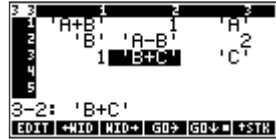
ENTER	Saves changes and exits.
ON	Discards changes and exits.
	HALT , temporary exit, a second returns to EQW, at the same place.
	OFF .
PRG	Switches between stack font and little font.
MTH	Display the first page of the menu.
NXT	Menu, next page.
	Menu, previous page.
	Turn off the menu.
	REPL aces the expression from the stack in the equation.

8.8.4 Cursor mode

Arrows , , , and	Movement.
Arrows with	Fast movement.
Arrows with	Border movement.
'	Back in selection mode, no changes to the selection.
ENTER	Back in selection mode, the new selection is the part of the equation that is pointed to by the cursor.
ON	Discards changes and exits.

9 Matrix editor

The Matrix Editor is a matrix-oriented environment.



9.1 Presentation


The **MK** Matrix Editor can be used like the standard Matrix Writer, but it adds new features. Not only real and complex matrixes can be edited, but also symbolic, character strings, program, etc., arrays.

For example, here is how the following symbolic matrix:

$$\begin{vmatrix} 'A+B' & 1 & 'A' \\ 'B' & 'A-B' & 2 \\ 1 & 'B+C' & 'C' \end{vmatrix}$$

is represented on the stack as a list of lists:

```
<< 'A+B' 1 'A' >  
< 'B' 'A-B' 2 >  
< 1 'B+C' 'C' >>
```

To view and edit it under the Matrix Editor, press the down arrow , or execute the command **EDITB**.





The symbolic Matrix Writer is powerful enough to manage any kind of data. For example you can edit your contacts list.

With some external tools you can also transform the Symbolic Matrix Writer into a spreadsheet.

9.2 Usage

Two modes are available, one to create a new matrix, the other to edit an existing matrix.

To create a new matrix, press  **ENTER**^{MATRIX}. An empty matrix is displayed, where any object can be entered. When  is pressed, if all the elements in the matrix are of the same type, an **Array of type** is pushed on the stack (the content is displayed for real and complex numbers and strings), otherwise a list of list is created (this is a multi-typed array).








In the second mode, when an existing array is edited, if a new element is entered outside the matrix, the other empty cells will be filled with "null" elements

corresponding to the matrix type (0 for reals, "" for strings, **NOVAL** for multi-typed arrays, etc.).
























*PS: For some security reasons, only arrays of real and complex are created. If you want to create an array of an other type, set the **MK** flag -27.*

9.3 Reference

9.3.1 Keys

Arrows  ,  ,  , and 	Movement.
Arrows with 	Page by page movement.
Arrows with 	Border movement.
	Clears the current cell (fills it with the "null" element).

9.3.2 Menu keys

	Edits the current cell in the command line editor.
	Edits the current cell in the best environment (EDITB).
	Makes the cells narrower and displays one additional column.
	Makes the cells wider and displays one less column.
 / 	If the box is visible, makes the cursor move to the next column after entry (on by default).
 / 	If the box is visible, makes the cursor move to the next row after entry.
 / 	Selects the cursor speed.
	Inserts a row of "null" elements.
	Deletes the current row.
	Inserts a column of "null" elements.
	Deletes the current column.
	Copies the current cell to stack level 1.
 	Copies the object on stack level 1, to the current cell
	Activates the interactive stack.
	Set the selection's upper left corner
	Set the selection's lower right corner
	Copies the selection on the stack level 1 as a symbolic matrix
	Copies the matrix on the stack level 1 to the current position
	Erase the selection

10 Grob editor (Picture2)

PICTURE2 creates and modifies graphic objects, similarly to PICTURE, but is more powerful for bitmap creation. Grobs can be in two or four-level grayscale, and every drawing is made in real-time, with or without zoom.

10.1 Usage

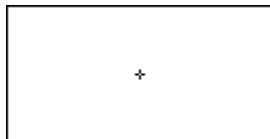
In this part, basis functions will be used to create a sample grob. See the reference part for a complete list of functions.

First create a four-level grayscale grob. Type **GREY 131 64** **ENTER**



The blank grob is displayed on the stack.

Then edit it. Press **▼**



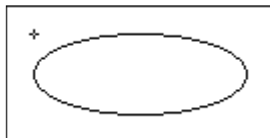
There are two ways to use PICTURE2 functions.

- Using the menu. To display the menu, press **-**, **NXT** and **PREV** **NXT** cycle through the menu pages.
- With Shortcut keys. Hold **α** and press the first letter of a function to launch it. When you don't know the letter corresponding to a function, search the function in the menu, the shortcut letter is displayed bold.

Now draw an ellipse, using the menu. Press **-** to display the menu if necessary. The ellipse is called by the menu key **ELLIP**. Press **-** to hide the menu.

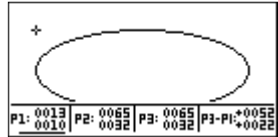
To call an ellipse with the shortcut keys, hold **α** and press **E** (E for Ellipse, remember the bold E on the menu key).

The ellipse can be shaped, using the arrow keys.



Press **EEX**_z anytime to Zoom. Press **EEX**_z a second time to get back to the normal view.

Press **+** to display the cursors coordinates.



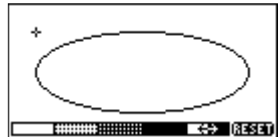
There are three coordinates shown: P1, P2 and P3. P1 is the current cursor position (it is underlined). To put the cursor on one coordinate, press **1**, **2** or **3**. In the case of the ellipse, P3 is the center of the ellipse, and P1 is a corner of the enclosing box. P2 is not used.

When the ellipse is correctly positioned (with P1 and P3), it is necessary to fix it on the grob. With no menu displayed, press:

- A** to fix it in white,
- B** to fix it in light gray (white for a two-level grayscale),
- C** to fix it in dark gray (black for a two-level grayscale),
- b** to fix it in black,
- E** to fix it in XOR mode (every pixel of the ellipse will be the opposite of the underlying pixel).

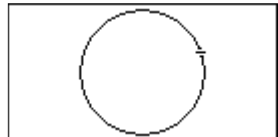
Or press **←** or **F** to cancel the ellipse.

These color keys are displayed in the third page of the menu:

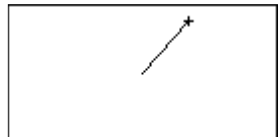


The other simple shapes are: (with the **α** shortcut keys):

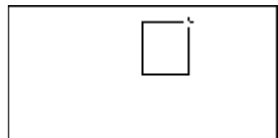
c Circle. P3 is the center and P1 is a point of the circle.



NXT L Line. P1 and P3 are the two ends.



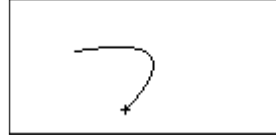
B Box. P1 and P3 are two opposite corners.



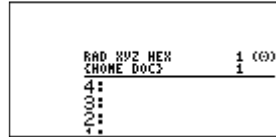
◀P Plan (filled box). P1 and P3 are two opposite corners.



◀SINs Spline curve. P1 and P3 are the ends, P2 is the tangential point.



◀MTHg GXOR. Merges another grob, see the reference part (here with a **LCD**→ grob).



A surface can be filled with a given color: Place the cursor on the surface to be filled, press **◀α** - **◀F** (Fill), and then, in case of a four-level grayscale grob, the corresponding color key **◀A**, **◀B**, **◀C** or **◀D**. A pattern can be used to fill a surface, see the reference part.

To leave a trail behind the cursor (like DOT+), press a color key **◀A**, **◀B**, **◀C**, **◀D** or **◀E**. To stop the trail, press **◀←**.

To change only one pixel, press **◀MTH**g, **◀PRG**h, **◀CST**i or **◀VAR**j, corresponding to the color keys **◀A**, **◀B**, **◀C**, **◀D**.

To exit, press **◀ON**, the modified grob will be put on the stack.

10.2 Reference

10.2.1 Commands












These commands can be typed in the command line, or called from RPL programs.

PICTURE2	The graphic editor which works with the PICT.
EDITB	The graphic editor which works with the grob on stack level 1.
→PICT	Stores the grob on stack level 1 in the PICT (like PICT STO , but it also works the four-level grayscale grobs).
PICT→	Puts the current PICT grob on the stack.
?GREY	If the grob on stack level one is in four-level grayscale, returns 1, else 0.
GBLANK	Equivalent to BLANK , but creates a four-level grayscale grob.
→GREY	Takes two B&W grobs on the stack and creates one four-level grayscale grob with them. The grob on level 2 defines the low-weighted bits (a pixel in this grob becomes a light gray pixel in the gray-grob, a pixel in the grob on level 1 becomes a dark gray pixel in the gray-grob; if both or set, they become a


	black pixel). To convert a two-level grayscale grob into a four-level one, type DUP →GREY .
GREY→	Contrary of →GREY , takes a four-level grayscale grob and returns two B&W grobs.


























10.2.2 Keys

ON	Exits PICTURE2.
	Switches off the calculator.
+	Switches on or off the menu display.
-	Switches on or off the coordinates display.
1	Position the cursor at P1, and set P1 as the current cursor.
1	Stores the current cursor position in P1.
1	Position the cursor at P1.
2	Position the cursor at P2, and set P2 as the current cursor.
2	Stores the current cursor position in P2.
2	Position the cursor at P2.
3	Position the cursor at P3, and set P3 as the current cursor.
3	Stores the current cursor position in P3.
3	Position the cursor at P3.
.	Inverts the color of the pixel under the cursor.
x	Copies P1 coordinates in P3.
÷	Switches between P1 and P3 as cursor (No effects if P2 is the current cursor).
ENTER	Puts the cursor coordinates on the stack, as a complex number.
ENTER	Puts the cursor coordinates on the stack, as a list of two binary numbers.
ENTER	Does a SUB. The grob enboxed between P1 and P3 is put on the stack, as a grob.
	Cancels the current shape (line, circle...) and returns to normal cursor mode.
CLEAR DEL	Clears the whole grob.
EEEX _z	Switches between the zoomed and the normal view.
STO	Puts the current grob on the stack.
Arrows and	Cursor movement.
and	
+Arrows	Page by page cursor movement.
+Arrows	Position the cursor on a border.
NXT	Menu keys.
-	
	In cursor mode, DOT+ in white. In shape mode, fixes the shape in white.
	In cursor mode, DOT+ in light gray (white in B&W).

	In shape mode, fixes the shape in light gray. In cursor mode, DOT+ in dark gray (black in B&W). In shape mode, fixes the shape in dark gray.
	In cursor mode, DOT+ in black. In shape mode, fixes the shape in black.
	In cursor mode, DOT+ in XOR mode. In shape mode, fixes the shape in XOR mode.
 	like  , but only inverts the high-weighted bit plan (try it!).
	Cancels the current shape and returns to normal cursor mode.
	Puts a white pixel.
	Puts a light gray pixel (white in B&W).
	Puts a dark gray pixel (black in B&W).
	Puts a black pixel.


10.2.3 ALPHA keys

To use these keys, hold  down.

	Suspends the current editing, and edits the grob found on stack level 1 (Useful to edit a pattern for a pattern fill). Press  to return to the other grob editing.
	Box. P1 and P3 are two opposite corners.
	Circle. P3 is the center, P1 is a point on the circle.
	Executes the program stored in DOPICT .
	Ellipse. P3 is the center, P1 is a corner of the enclosing box.
	Surface Fill. In B&W, inverts the surface color. In grayscale, the wanted color must be then defined with  ,  ,  , or  . Press  or  to cancel.
	Goes to the interactive stack. Press  to return to PICTURE2.
	Line. P1 and P3 are the ends.
	Pattern fill (<i>Motif</i>). The pattern must be a 8x8 grob found on the stack level 1, in B&W or in grayscale.
	Filled box (Plan). P1 and P3 are two opposite corners.
	REPL. Takes the grob on stack level 1. It may be positioned with the arrow keys. It is validated by pressing:  does a REPL and drops the grob.  does a GOR and drops the grob.  does a GXOR and drops the grob.  or  cancels.
	Spline curve. P1 and P3 are the ends, P2 defines the tangent lines.

11 Filer



The Filer is a file utility that eases directory and port access.

Press  ^{MEMORY}**VAR** to execute it. The first screen lets you choose the directory.

11.1 Ports and directories screen



The tree screen shows a list of ports and directories.

Press ,  to go throughout the list.

Press , **ENTER** or **OK** to go in the selected location.

Press **ON** or **CANCEL** to exit to Filer.

11.2 File selection screen




On this screen are shown: the free memory, the total size of the selected files, a list of files and the menu.

Press ,  to move in the file list.


In each line, there are five fields: the selection number (if the object is selected), an icon depending on the file type, the name, the type and the size of the file.

Press **ENTER** to select or deselect a file.

Press  on a directory or a library to go into it.

Press  to go out of the current directory or library.

For all the commands that require a list of files, if there is no selection, the current line will be taken as the selection.

You can change the header in order to display the number of objects available and the working path (here port 0 home):  ^{MENU}**NXT**














To hide the type and the size of every objects press **VAR**



11.3 Menu

Some commands work only in a special mode, for example only in the HOME directory. When you try to run these commands in an incorrect mode, a beep is produced.





	Edits the current object.
	Copies the selected objects. You define the destination with the tree screen: select the destination (a port or a directory) and press ENTER or OK .
	Moves the selected objects to the destination you selects in the tree screen.
	Puts the selected objects on the stack.
	Gives information about the current object: Name, type, size, address, CRC.
	To go to the tree screen.
	Clears the selected objects. If the MK flag -12 is clear (default) you will be prompt for a confirmation
	Renames the current object. This command is available only in the VAR
	Orders the selected files in the order of the selection. The other files follow.
	Evaluates the current object (Some programs that normally do not work in port 2, can work using this command).
	Sends the selected files with KERMIT, with the current I/O settings. If the variable STARTSEND exists in the HOME directory, it is used to send the files (level 2: the object, level 1: the name).
	Suspend the Filer and return to the RPL stack
	Goes to the hexadecimal editor, at the beginning of the current object.

  	Views the current file if it is a string.
  	Edits the current object with the best environment.
 	Starts the custom mode if the FILER.CUSTOM is defined
  	By default, the Filer displays the memory available and the selected objects's number. It can display the number of objects and the working path

11.4 HEXA Editor

The screen shows 64 nibbles and their ASCII value. Each line represents 16 nibbles (separated by 8).

On the bottom of the screen are written the assembly instruction, and the RPL object at the current address.







Press  and  to go one nibble backward or forward. Press  and  to move by 16 nibbles.

```





BD020: D9D20E16 32E02A22 @-3d44c"
BD030: 9E208057 001C4320 <==u64b
BD040: 6E201045 E162C28 @=T.6Ac
BD050: 20E10000 5E756373 ==2W67
BD060: 70E10028 656379c2 ==2W67
BD070: A239E208 057004E2 *S.PP#e+
BD080: A2F17A18 DBF1C2A2 *S.S2L.*
BD090: 09100035 56E64636 +=Sendi
BD020 C=B A PROG
FIND NEXT GOTO P308J P3ADR EXIT

```

11.4.1 Menu



	Searches an hexadecimal string
	Searches the next occurrence.
	Goes to a given address.
	Pushes the current object on the RPL stack. This command is very useful to recover datas in a corrupted card ((Invalid Card Data)
	Pushes the current address on the RPL stack as a binary integer
	Exits the Hexa Editor.

11.4.2 Keys

	Jumps to the next assembly instruction.
	Jumps to the next RPL object.
 	Goes in edit mode.


11.4.3 Edit mode

Very dangerous, use at your own risk !

Use the arrow keys to move, press a hexadecimal key from 0 to F to put the digit at the cursor position. Press   again to exit the Edit Mode.

Utilities

1 Font Editor (EDF)

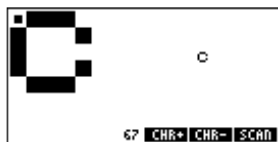
EDF is a font editor. It is accessed through the CHARS menu, where the current font can be edited. Any font can be edited by pushing it on the stack and pressing .

1.1 Font editing


When EDF is launched, scan mode is active, the font is displayed on the screen and the cursor is moved by the arrow keys. The current character number is displayed on the bottom of the screen (here 67: C).



MODIF Switches to edit mode. On the left of the screen, the character is displayed eight times bigger, on the right in its normal size.



SCAN gets back to the CHARS mode.

The cursor is in the big character view, on the left. It is moved with the arrow keys, and  inverts the pixel under it.

1.2 CHARS mode

The chars mode is the default mode when EDF is launched. The cursor is moved by the arrow keys.

ECHO pushes the current character on the command line. Press **ON** to exit.

ECHO1 pushes the current character on the command line and exits to the command line.

1.3 A session with CHARS

We will add a point inside the character **C**.



Press  **CHARS**  to enter in CHARS. Use the arrows to move to the **C** character:



Press **MODE** to edit it:



Use the arrows to move to the center of the character, press **◉** to put a point:



Press **ON** to return to the stack display. See how the **C** characters have changed.



Note that this only affects the current font. The next **→FONT** will destroy the changes. To make them permanent, do a **FONT→** and store the changed font.

2 Machine Language Compiler (Masd)

2.1 Generalities on ML (Machine Language)

As the Saturn processor directly executes ML, the operating system can not control what a ML program is doing.

On the HP 48 calculator, user data are stored in the same area as temporary data. When there is a bug in a ML program, you have best chance to lost your data's. So be very careful when programming in ML.

ML is a processor dependent language, so what you will learn on the HP 48 will not be useful on other processor. On the other hand, the programming techniques you will acquire are not dependent of the hardware and then will be reusable.

2.2 Launching Masd

To compile a program, put the string on the top of the stack and type **ASM** or use the **ASM** menu of the **MDGK** library.

2.3 Generalities on Masd Syntax

Masd expects a character string (called source) on the top of the stack.

A source is a set of instructions, comments, and separation characters and ends with a carriage return and an arobas **@**.

Masd is case sensitive, so be careful, as « **boucle** » and « **BOUCLE** » are two different labels.

Separation characters are those with an ASCII number below 32. They include spaces, tabs, line feed and carriage return.

Some instructions need a parameter, called field. Separation characters between an instruction and the field, are spaces, tabs, and points. Therefore **A+B.A** can be used instead of **A+B A**.

Comments can be placed everywhere between two instructions. They begin with **%** or **;** and finish at the end of the current line.

Directives change the way Masd interprets your source. Theses instructions begin with a **!** and will be explained later.

While Masd is working, it displays messages on the screen to explain what is processed.

Linking FileName	Compiles the FileName file. The main file is called Main.
Uplink	Ends the compilation of a file.
Compil. Glab	Compiles global jumps.
Compil. Exp	Calculates expressions.
Garbage	Reconfigures the memory.
Ghost LabelName	Detects an unused label.

If Masd detects one or more syntax error, it will push a list describing all errors on the stack. This list is used by **ER** to find the location of errors, and let the user correct them.

Errors may be notified at the wrong place. For example if a label is not correctly defined, the errors will be located on the calling instructions.

2.4 Links

Links are source files that can be linked during compile time (equivalent to the `{$!}` directive in PASCAL and `#include` in C).

When a link call is encountered, Masd stops compiling the current link, compiles the new one and then continues compiling the first one.

Syntax:

'FileName links the file called FileName.

Note 1: A link can call other links.

Note 2: You can not use more than 256 links in your project.

Note 3: To know how Masd looks for files, see the File search section.

Note 4: Links are useful to cut projects in independent parts to allow fast and easy access to source code.

2.5 Using labels

A label is a marker in the program. The principal use of labels is to determine jump destinations.

A label is a set of less than 128 characters different from space, '+', '-', '*', and '/'. A label begins with a star '*' and ends with a separation character.

***BigLoop** is the BigLoop label declaration.

Be careful about upper and lower cases!

Three types of labels can be used:

- Global labels

A global label is a label that can be used everywhere in the project, like global variables in Pascal or C.

- Local labels

A Local label is a label that is only accessible in a local section like local variables in Pascal or C.

A local section starts at the beginning of a source, after a global label, after a link (see link section) or after a **!LOCAL** directive.

A local section finishes at the end of a source, before a link, before a global label or before a **!LOCAL** directive.

A local label is identified by a '.' as the first character.

- Link labels

A link label is a label that exists only in the link where it is declared, like a private clause in Object Pascal.

A link label is identified by a '_' as the first character.

Note 1: In projects, using less global labels is better because a global label is longer to compile and because it gives a better program structure. A good habit is to use global labels to cut the program in subroutines, and to use local labels inside these subroutines.

Note 2: The **Meta Kernel** command line is able to find labels in a source. Press blue shift **GOTO**.

2.6 Using constants

It is possible to define constants. It is useful to identify a memory address by a name, rather by the address itself.

For example, instead of typing **D1=80100** every time it is needed, it is better to declare **DC Result 80100** at the beginning of the project and then to type **D1=<5>Result** when needed.

Constant declaration:

DCE CstName Expression or
EQUE CstName Expression or
DEFINE CstName Expression
DC CstName CstValue or **EQU CstName CstValue**

CstValue is a hexadecimal number. A decimal number can be typed with a leading **#** character.

DC Foo 10 is same as **DC Foo #16**

Note 1: A constant cannot be given the same name as a declared label.

Note 2: The name of a constant follows the same rules as the name of a label.

Note 3: A constant value is always stored on 5 nibbles.

Masd introduces a 'programming register' called CP (Constant Pointer) which helps to define constants. CP is defined by:

CP=Cst or **CPE=Expression**

CP is defined on 5 nibbles, its initial value is 80100.

DCCP Increment ConstantName

declares a constant with the current CP value and then increase CP by Increment.

Note: Increment is a hexadecimal value, to use a decimal value, put a leading **#**.

For example, if CP equals to \$10

DCCP 5 Foo

defines a Foo constant with a value of \$10 and then change the value of CP to \$15.

Several constants can be defined, starting from CP.

: Inc CstName0 CstName1 ... CstNameN-1 :

defines *N* constants CstName_x with a value of CP+x*Inc and then changes the CP value to CP+N*Inc.

By default, *Inc* is a decimal number. It can be typed in hexadecimal, with a leading **\$**.

2.7 Expressions

An expression is a mathematical operation that is calculated while compiling.

Terms of this operation are hexadecimal or decimal values, constants or labels.

An expression stops on a separation character.

DCCP 5 @Data

...

D1=<5>@Data+\$10/#2 D0=<5>\$5+DUP LC<5>"DUP"+#5

are correct expressions.

Notes:

- A hexadecimal value must begin with a **\$** and a decimal value must begin with a **#**.
- The **&** character equals to the address of the current instruction in absolute mode, or to the offset of the current instruction in standard mode (This value has no meaning in itself, but may be used to calculate the distance between a label and the current instruction).
- The value of a label is the address of the label in absolute mode, or the offset of the label in the program in normal mode (This value has no meaning in itself, but may be used to calculate the distance between a label and the current instruction).
- Entries from the STARTEXT table may be used. In an ambiguous case (**DUP+#5** may either be an addition **DUP + 5**, or an entry **'DUP+#5'**), add **" "** around the word: **"DUP"+#5**.
- There are no priorities (precedences) in operations. **\$1+\$2*\$3=\$9** instead of **\$7**. Use parenthesis to set precedences.
- You can't use more than three level of parenthesis.
- Calculations are done on 64 bits.
- X divide by 0 = **\$FFFFFFFFFFFFFFFF**.
- By default, expressions are calculated at the end of the compilation of the project.
- Masd can be forced to compile expressions using the **!COMPEXP** directive. But be careful, if an expression uses a label that is not already declared, this will cause an error on the expression and not on the **!COMPEXP** directive.
- Masd can be forced to compile the last expression using the **!COMPEX** directive with the same remarks than for **!COMPEXP**. This can be useful when local labs are used in an expression.

2.8 Skips

Skips are a first step from ML to a third generation language, even if they are only another way to write SATURN instructions.

The basement of Skips is the Block structure.

A block is enclosed in **{** and **}**, and can be inside another block.

The following instructions deal with blocks.

SKIPS instructions	Equivalents
SKIP { ... }	GOTO .S ... *.S
SKIPL { ... }	GOTOL .S ... *.S

SKIPC (...)	GOC .S ... *.S
SKC (...)	GOC .S ... *.S
SKIPNC (...)	GONC .S ... *.S
SKNC (...)	GONC .S ... *.S
Test SKIPYES (...)	Test GOYES .S ... *.S
Test (...)	Test GOYES .S ... *.S
Test+ (...)	/Test GOYES .S ... *.S
SKUB (...)	GOSUB .S ... *.S
SKUBL (...)	GOSUBL .S ... *.S
{ ... }	Defines a block (generates no code)
STRING (...)	\$/02A2C GOIN5 *.S ... *.S (to create a character string)
CODE (...)	\$/02DCC GOIN5 *.S ... *.S (to create a code object)
STROBJ <i>PROLOG</i> (...)	\$/ <i>PROLOG</i> GOIN5 .S ... *.S (to create a 'prolog - length' object)

/Test is the opposite of Test. For example if Test is ?A<C.A, /Test is ?A>=C.A. The test instructions dealing with the hardware register (?HST=0, ?MP=0, ?SR=0, ?XM=0 and ?SB=0) cannot be inverted.

Once blocks are defined, special instructions can be used in them. These instructions called EXIT and UP allow to jump to the end or to the beginning of a block.

These instructions	are equivalent to
{	*.Beginning
EXIT	GOTO.End
EXITC	GOC.End
EXITNC	GONC.End
?A=0.A EXIT	?A=0.A +.End
UP	GOTO.Beginning
UPC	GOC.Beginning
UPNC	GONC.Beginning
?A=0.A UP	?A=0.A +.Beginning
}	*.End

Note: do not make confusion between EXIT and UP instructions, which are GOTOs, and EXIT and UP after a test, which are GOYES's.

EXIT and UP can jump to the beginning or to the end of an upper-level block by specifying the number of blocks to exit, after the UP or EXIT instructions.

These instructions	Are equivalent to
{	*.Beg3
{	*.Beg2
{	*.Beg1
UP2	GOTO.Beg2
UP3	GOTO.Beg3
EXIT1	GOTO.End1
EXIT3	GOTO.End3
}	*.End1
}	*.End2
}	*.End3

Note: EXIT1 is equivalent to EXIT, and UP1 is equivalent to UP.

Using SKIPELSE, SKEC or SKENC instructions, two blocks create an IFNOT-THEN-ELSE structure.

These instructions	Are equivalent to	Or in high-level language
<pre>?A=0.A SKIPYES (EXIT UP) SKELSE (A+1.A EXIT UP)</pre>	<pre>?A=0.A GOYES.Beg2 *.Beg1 GOTO.End2 % and not End1 GOTO.Beg1 *.End1 GOTO.End2 *.Beg2 A+1.A GOTO.End2 GOTO.Beg2 *.End2</pre>	<pre>IF NOT A=0 THEN BEGIN END ELSE BEGIN END</pre>

2.9 Macros

If data are to be included in a project, they can be entered in a source file, using `$`. But a simpler way is to include data from an external file, which is a macro. The macro file must be a character string, a graphic, a code object or a list.

In case of a string or a code, Masd includes only the data part (after the length)

In case of a graphic, only the graphic data will be included (no length, no dimensions).

In case of a list, only the first object of the list will be included following the previous rules.

The syntax is:

`/$FileName`

Note: To know how Masd looks for the FileName file, see the following section.

2.10 Filename conventions

Masd sometimes needs to find a file in the HP 48 memory.

The file can be found either by specifying the complete file name and location, or only the file name to be search in the search path list.

The initial search path list contains the current directory, the upper directory and so on to the HOME directory.

Note: You can add a directory in the search path list using `!PATH+ RepName` where RepName identifies a directory name using the full pathname rules, explained below.

To specify a full path, use

`H/` to specify HOMEDIR as the root.

`x/`, where *x* is a port number, to specify a port as root.

This root is followed by a list of directories, ending with the name of the file.

`4/TOTO/TITI/TUTU` specifies the TUTU file in the TITI directory, stored in the TOTO backup of the fourth port.

`H/ME/YOU` specifies the YOU file in the ME directory, in the HOMEDIR.

2.11 Units

A unit is a part of a project, which can be compiled separately and stored in a file. The generated unit code can then be included in the project final code, without recompiling it.

A unit can use labels and constants defined in the calling program, and the calling program can use labels and constants defined in the unit.

Any source can be changed into a unit by placing a **!UNIT** directive in the source.

In the calling source file, type **USES FileName** to include a unit.

Note 1: A unit is smaller in memory than the source file it comes from.

Note 2: Using a unit is a very fast operation, faster than compiling the source.

Note 3: Tools libraries can be created and distributed without sharing the source code.

2.12 Library creation

Libraries can be created with Masd using the following instructions:

LIB, MESSAGE, VISIBLE, HIDDEN and LCONFIG.

Here is a source producing a library:

```
!NO CODE          % no code prolog $02DCC
LIB               % beginning of lib
( "library name" % library title
 1234            % library romid in decimal
MESSAGE          % message table
( "Message 1"
  "Message 2"
)
VISIBLE          % $visible list
( VisibleName1  % Name of File Names containing
  VisibleName2  % visible objects
)
HIDDEN           % $hidden list
( HiddenName1   % Name of File Names containing
  HiddenName2   % hiddens objects
)
LCONFIG ConfigName % Name of the file containing
                  % the Config Object
)
@
```

2.13 SysRPL mode

Masd can switch to SysRPL mode (also called System RPL or External) using the **!RPL** directive. In RPL mode, lots of things are changing.

Separation characters don't change (Characters of ASCII number below 32).

Comments begin with ***** and end at the end of the line, or begin with '**(**' and end with **)**'.

2.13.1 Instructions

In RPL mode, Masd interprets instructions in the following order.

a) Constants

If a constant exists with the same name, the constant value is used on 5 nibbles.

Example:

```
EQU TOTO 4E2CF
```

...

```
TOTO
```

will produce

```
FC2E4
```

b) External (in entry points table)

If an entry in the external table exists with the same name, the value associated with this entry is used.

```
BZUExt
```

will produce

```
DB10C
```

Note 1: Use the TableCreator program on your PC to create the table.

Note 2: This table can be stored anywhere in the HP 48 memory. Create a **STARTEXT** text file in the HOMEDIR with the file name of the External table on the first line and a @ on the second line.

Example of a STARTEXT file for a table stored in the Table backup to the fourth port:

```
"4/Table
```

```
@"
```

Note 3: Using an external table is much faster than using constants. On the other hand, constants are project dependant, which is not the case of an external table.

c) Tokens

::	Program prolog \$02D9D
;	List, Program or Algebraic end \$0312B
{	List prolog \$02A74
}	List end \$0312B
SYMBOLIC	Algebraic prolog \$02AB8
UNIT	Unit prolog \$02ADA
# <i>cst</i>	System Binary of <i>cst</i> value, given in hexadecimal.
# #<i>cst</i>	System Binary of <i>cst</i> value, given in decimal.
#<i>cst</i>	System Binary (hexadecimal value). If the SB exists in ROM, its address is used.
##<i>cst</i>	System Binary (decimal value). If the SB exists in ROM, its address is used.
PTR <i>cst</i>	Address. PTR 4E2CF generates FC2E4 .
ACPTR <i>cst1 cst2</i>	Extended pointer.
ROMPTR <i>LibN ObjN</i>	XLIB.
% <i>real</i>	Real number.
%% <i>real</i>	Long real number.
C% <i>real1 real2</i>	Complex number.
C%% <i>real1 real2</i>	Long complex number.
"..."	Character string. Special characters can be included by typing \ and the ascii value on two hexadecimal characters.

ID name	Global name.
LAM name	Local name.
TAG chrs	Tagged object.
XxlibName	XLIB identified by its name. If it is a standard HP48 command (like xDUP), the address is used instead of an XLIB object.
HXS Size Data	Binary integer (\$02A4E), Size is in hexadecimal and Data is a set of hexadecimal characters. Example: HXS 5 FFA21
GROB Size Data	GROB (\$02B1E).
LIBDAT Size Data	Library data (\$02B88).
BAK Size Data	Backup (\$02B62).
LIB Size Data	Library (\$02B40).
EXT1 Size Data	Extended1 (\$02BAA).
EXT2 Size Data	Extended2 (\$02BCC).
EXT3 Size Data	Extended3 (\$02BEE).
EXT4 Size Data	Extended4 (\$02C01).
ARRAY Size Data	Array (\$029E8).
LNKARRAY Size Data	Linked Array (\$02A0A).
CODE Size Data	Code object (\$02DCC).
NIBB Size Data or NIBHEX Data or CON(Size) Expr	Includes directly hexadecimal data (no prolog).
CHR x	Character object.
INCLOB FileName	Includes the content of the file <i>FileName</i> .
INCLUDE FileName	Includes the source of the file <i>FileName</i> to be compiled (Like ' in ASM mode).
LABEL label	Defines a label (like * in ASM mode).
EQU CstName Cst or EQUE CstName Exp or DEFINE CstNam Exp	Defines a constant (Like DC in ASM mode).
EQUCP Interleave CstName	Defines a constant (Like DCCP in ASM mode).

Note: A constant can be defined in ASM or RPL mode, and may be used in both modes.

d) Decimal value (System Binary)

If the instruction is not yet recognized, and if it is a decimal value, Masd generates a system binary.

e) Unnamed local variables

Then, Masd tries to match the instruction with declared local variables.

A local environment is set using:

```
{ { var1 var2 ... varN } } with N<23
```

These variables have names during compile time, but they are implemented as unnamed local variables, which are faster to access than named local variables.

A local variable is recalled by typing its name

Data can be stored in a local variable by typing its name, with a leading ! or =.

Note 1: Local variable are available until the next local definition.

Note 2: The local environment is not closed automatically, use **ABND** or other provided words.

Example:

{ { label1 label2 .. labelN } } will become :

' NULLLAM <#N> NDUPN DOBIND (or 1LAMBIND if there is only one variable)

And:

label1 → 1GETLAM

=label1 → 1PUTLAM

!label1 → 1PUTLAM

Program example

<pre>;; { { A B } } B A! ABND ;</pre>	<pre>;; ' NULLLAM TWO NDUPN DOBIND 2GETLAM 1PUTLAM ABND ;</pre>
---------------------------------------	---

Notes on RPL mode.


Masd switches back to ASM mode using the **!ASM** directive.


2.13.2 Special keystrokes in the command line

In the command line editor, some keystrokes help to enter characters that are specific to System RPL, and also to easily find a command (auto-completion).

 followed by  displays a comma , .


 **maintained** and  displays a semicolon ; .

In alpha mode,  followed by **VAR** displays a vertical bar | .

In alpha mode,  **maintained** and **VAR** displays an anti-slash \ .


Auto-completion works as follow:


Type the first characters of the command, for example **DU**.

Press  and maintain it. Press **SPC** once. On the first screen line, appears the first available command that begins with **DU**: **DUMP**.

To see the next possibilities, press **SPC** (always with  pressed): **DUP**, **DUPó**, **DUP#<?...**

To go back, and see previous choices, press .

If you see the right command, simply release , the command will be inserted in the text.

To cancel the search without inserting a command, press **+** once and then release .

2.13.3 Program example

To find or understand the syntax of a particular object, use **+S2** to produces an Masd syntax-source

As in ASM mode, a RPL source must end with a @.

```
"!NO CODE
!RPL
::
ONE (My first program)
!ASM
% Turn into ASM mode
CODE { SAVE LOADRPL }
!RPL (Turn into SysRPL mode)
TWO
:
@"
```

2.14 SATURN instructions syntax

In this section:

x is an integer number between 1 and 16.

h is a hexadecimal digit.

a is a 1 to 16 or a 0 to 15 number depending of the current mode (0-15 or 1-16)

f is a field A, B, X, XS, P, WP, M or S.

Reg is a working register A, B, C or D.

SReg is a save register R0, R1, R2, R3 or R4.

Exp is an expression.

Cst is a constant. The value is given in hexadecimal or decimal using a leading \$ or # respectively.

DReg is a pointer register D0 or D1.

Data is memory data pointed by D0 or D1. It means **DAT0** or **DAT1**.

Note: For instructions that use two working registers, only the pairs A-B, B-C, C-D and A-C are available.

For instructions like *Reg1=Reg1...* you can write only *Reg1...* Example: **A=A+C.A** is the same as **A+C.A**.

2.14.1 Assigning 0 to a register

Syntax: *Reg=0.f*

Example: **A=0.M**

2.14.2 Loading a value in A or C

LC and LA instructions allow to load a constant value into A or C register.

LC *hhh...hh* loads *x* nibbles into C.

LA *hhh...hh* loads *x* nibbles into A.

Example: **LC 80100**

Note: LC #12 allow to load 12 decimal into the 3 first nibbles of C. The number of nibbles used is the number of characters necessary to write the value (including the #). So #12 will take three nibbles.

LCASC(x) Characters loads the hexadecimal value of x characters into C. x must be between 1 and 8. **LAASC(x)** is the counterpart for A.

Example: **LCASC(7) HP_MASD**

LC(x) Exp or **LA(x) Exp** load the result of an expression into C or A, using x nibbles.

Example: **LC(5)@Buffer+DataOffset**

2.14.3 Loading a register value into another register

Syntax: *Reg1=Reg2.f*

Example: **A=B.X**

2.14.4 Exchange between two registers

Syntax: *Reg1Reg2EX.f*

Example: **CDEX.W**

2.14.5 Addition

Syntax: *Reg1=Reg1+Reg2.f* or *Reg1+Reg2.f*

Example: **C=C+A.A** or **C+A.A**

Note if *Reg1* and *Reg2* are same, this cause to multiply the register by two.

2.14.6 Subtraction

Syntax: *Reg1=Reg1-Reg2.f* or *Reg1-Reg2.f*

Example: **C=C-B.A** or **C-B.A**

Note: The following instructions are also available:

A=B-A.f **B=C-B.f** **C=A-C.f** **D=C-D.f**

2.14.7 Increment and decrement

Syntax: *Reg=Reg+Cst.f* or *Reg+Cst.f* *Reg=Reg-Cst.f* or *Reg-Cst.f*

Example: **A=A+10.A** or **A+10.A** **A=A-10.A** or **A-10.A**

Note 1: The Saturn processor is not able to add a constant greater than 16 to a register but if *cst* is greater than 16, Masd will generate as many instructions as needed.

Note 2: Even if adding constants to a register is very useful, big constants should be avoided because this will slow down execution, and generate a big program.

Note 3: Adding a constant greater than 1 to a P, WP, XS or S field is a bugged SATURN instruction (problem with carry propagation). Use these instructions with care.

Note 4: After adding a constant greater than 16 to a register, the carry should not be tested.

2.14.8 Right nibbles shifting (divide by 16)

Syntax: *RegSR.f*

Example: **ASR.W**

2.14.9 Left nibbles shifting (multiply by 16)

Syntax: *RegSL.f*

Example: **ASL.W**

2.14.10 Right bit shifting (divide by 2)

Syntax: *RegSRB.f*

Example: **ASRB.W**

2.14.11 Right circular nibble shifting

Syntax: *RegSRC.f*

Example: **ASRC.W**

2.14.12 Left circular nibble shifting

Syntax: *RegSLC.f*

Example: **ASLC.W**

2.14.13 Logical AND

Syntax: *Reg1=Reg1&Reg2.f* or *Reg1&Reg2.f*

Example: **C=C&B.A** or **C&B.A**

2.14.14 Logical OR

Syntax: *Reg1=Reg1!Reg2.f* or *Reg1!Reg2.f*

Example: **C=C!B.A** or **C!B.A**

2.14.15 Logical NOT

Syntax: *Reg1=-Reg1-1.f*

Example: **C=-C-1.A**

2.14.16 Mathematical NOT

Syntax: *Reg1=-Reg1.f*

Example: **C=-C.A**

2.14.17 Loading value into a R Register

Syntax: *RReg=Reg.f*

Example: **R0=A.W**

Note: *Reg* can only be A or C.

2.14.18 Loading value into A or C from a R register

Syntax: *Reg=RReg.f*

Example: **A=R1.X**

Note: *Reg* can only be A or C.

2.14.19 Exchange between A or C and a R register

Syntax: *RegRRegEX.f*

Example: **AR1EX.X**

Note: *Reg* can only be A or C.

2.14.20 Memory write (POKE)

These instructions write the value of A or C at the address pointed to by D0 or D1.

Syntax: *Data=Reg.f* or *Data=Reg.x*

Example: **DAT1=C.A** or **DAT0=A.10**

Note: *Reg* can only be A or C.

2.14.21 Memory read (PEEK)

These instructions load into A or C the data pointed to by D0 or D1.

Syntax: *Reg=Data.f* or *Reg=Data.x*

Example: **C=DAT1.A** or **A=DAT0.10**

Note: *Reg* can only be A or C.

2.14.22 D0 and D1 modifications

a) Loading D0 and D1

Syntax: *DReg=hhor* or *DReg=hhhh* or *DReg=hhhhh* or
DReg=(2)Exp or *DReg=(4)Exp* or *DReg=(5)Exp*

Example: **D0=FF D0=12345** **D1=(5)totot+\$5**

b) Exchanges between A or C and D0 or D1

- ◆ Loading A or C, field A, into D0 or D1

Syntax: *DReg=Reg*

Example: **D0=A**

Note: *Reg* can only be A or C.

- ◆ Loading the four low nibbles of A or C into D0 or D1

Syntax: *DReg=RegS*

Example: **D0=AS**

Note: *Reg* can only be A or C.

- ◆ Exchanging A or C, field A, and D0 or D1.

Syntax: *RegDRegEX*

Example: **AD1EX**

Note: *Reg* can only be A or C.

- ◆ Exchanging the 4 first nibbles of A or C and D0 or D1

Syntax: *RegDRegXS*

Example: **AD1XS**

Note: *Reg* can only be A or C.

c) Increment and decrement of D0 and D1

Syntax: *DReg=DReg+Cst* or *DReg-Cst*

Syntax: *DReg=DReg-Cst* or *DReg-Cst*

Example: **D0=D0+12** **D1-50**

Note 1: The Saturn processor is not able to add a constant greater than 16 to a register but if *cst* is greater than 16, Masd will generate as many instructions as needed.

Note 2: Even if adding constants to a register is very useful, big constants should be avoided because this will slow down execution, and generate a big program.

Note 3: After adding a constant greater than 16, the carry should not be tested.

2.14.23 Working registers tests

Notes:

- A test is always followed by **RTNYES**, **GOYES**, **SKIPYES**, **EXIT**, **UP**, **GOTO**, **GOTOL**, **GOVLNG**, **GOSUB**, **GOSUBL** or **GOSBYL**.
- **RTY** is the same as **RTNYES**.
- An arrow (→) may be followed by a label name, then replacing **GOYES**, or may be followed by a skip block, which is equivalent to the inverse of the test followed by **SKIPYES**, to reproduce a IF-THEN structure. Example: **?A=C.A → { }** is the same as **?A#C.A { }**.
- **SKIPYES** may be omitted if followed by a skip block (**{ }**).
- If the test is followed by a **GOTO**, **GOTOL**, **GOVLNG**, **GOSUB**, **GOSUBL** or **GOSBYL**, Masd compiles the inverse of the test, to reproduce a **GOYES** with a larger range. Example: **?A=C.A GOTO B** is the same as **?A#C.A { GOTO B }**.
- **GOTO**, **GOTOL**, **GOVLNG**, **GOSUB**, **GOSUBL**, **GOSBYL** or → **{ }** cannot be used after a HST test.
- A label name must follow a **GOYES**, **GOTO**, **GOTOL**, **GOVLNG**, **GOSUB**, **GOSUBL** or **GOSBYL**.

a) Equality and inequality tests

Syntax: *?Reg1=Reg2.f* *?Reg1#Reg2.f*

Example: **?A=C.B** **?C#D.A**

Note: The HP inequality character may be used.

b) Lower and greater tests

Syntax: *?Reg1<Reg2.f* *?Reg1<=Reg2.f*

Example: **?A<C.B** **?C>=D.A**

Note: The HP lower or equal and greater or equal characters may be used.

c) Nullity tests

Syntax: *?Reg=0.f* *?Reg#0.f*

Example: **?A=0.B** **?C#0.XS**

Note: The HP inequality character may be used.

2.14.24 Working with some bits of A or C register

RegBIT=v.a *?RegBIT=v.a* where Reg is A or C, v is 0 or 1 (reset or set), and a is the bit number.

Examples: **ABIT=0.5**, **?CBIT=1.3 GOYES TOTO**

2.14.25 Operations on PC

A=PC C=PC PC=A PC=C APCEX CPCEX PC=(A) PC=(C)

2.14.26 Working with the Hardware Status Register

**SB=0 XM=0 SR=0 MP=0 HST=*a*
?SB=0 ?XM=0 ?SR=0 ?MP=0 ?HST=*a***

2.14.27 Working with P

**P=*a*
P=P+1 P+1 P=P-1 P-1
?P=*a* ?P#*a*
P=C.*a* C=P.*a* CPEX.*a*
C=C+P+1 C+P+1**

2.14.28 Jump instructions

**GOTO *label*
GOTOL *label* or GOLONG *label*
GOVLNG *Cst* *Cst* is an hexadecimal number.
GOVLNG =*label* *label* is a constant, or a label in absolute mode
GOVLNG ="*COMMAND*" *Command* is an entry in the STARTEXT table.
GOSUB *label*
GOSUBL *label*
GOSBVL *Cst* *Cst* is a hexadecimal number.
GOSBVL =*label* *label* is a constant, or a label in absolute mode.
GOSBVL ="*COMMAND*" *COMMAND* is an entry in the STARTEXT table.**

**GOC *label*
GONC *label*
GOTOC *label* same as SKIPNC { GOTO *label* }
GOTONC *label* same as SKIPPC { GOTO *label* }**

**RTN RTNSXM RTNCC RTNSC RTI
RTNC RTNNC
RTNYES or RTY after a test.**

2.14.29 Exchanges between C and RSTK

C=RSTK and RSTK=C instructions allow to push to or pop data from the Saturn return stack.

2.14.30 Input / output instructions

OUT=CS, OUT=C, A=IN and C=IN

Note 1: **A=IN** and **C=IN** instructions are bugged (they only work on even addresses). So use **A=IN2** and **C=IN2**, which are ROM calls to **A=IN** and **C=IN** instructions.

Note 2: if the beginning of ROM is not usable (because it is recovered by RAM), use **A=IN3** and **C=IN3**, which are calls to **A=IN** and **C=IN** instructions in the **Meta Kernel Card**.

Note 3: **OUT=C=IN** is a ROM call that does **OUT=C C=IN**.

Note 4: **OUT=C=IN3** is the same, but in the **Meta Kernel card** (works even if lower ROM is recovered).

2.14.31 Processor control instructions

Working mode modification

SETDEC SETHEX

other instructions

UNCNGF CONFIG RESET C=ID

SHUTDN INTON INTOFF RSI

SREQ?

BUSCB BUSCC BUSCD

2.14.32 New instructions of Masd

GOINC <i>label</i>	Equivalent to LC<5>label-& . (& is the address of the instruction)
GOINA <i>label</i>	Equivalent to LA<5>label-& . (& is the address of the instruction)
« and »	Program prolog 02D9D and epillog 0312B.
~	Noeval
A <i>label</i>	In absolute mode, places the address of the label.
B <i>label</i>	In absolute mode, places the address of the label+5.
\$ <i>hhh...hhh</i> or NIBHEX <i>hhh...hhh</i>	Includes hexadecimal data. Example: \$12ACD545680B .
\$/ <i>hhh...hhh</i>	Includes hexadecimal data in reverse order. Example: \$/123ABC is equivalent to \$CBA321 .
\$(x) <i>Exp</i> or CON(x) <i>Exp</i> or EXP(x) <i>Exp</i>	Places the value of <i>Exp</i> in the code, on x nibbles.
¢ <i>Ascii</i>	Includes ASCII data. The end of the string is the next ¢ or carriage return. Example: ¢Hello¢ . To output a ¢ character, put it twice.
INCLUDE <i>FileName</i>	Includes directly the specified file as data.
€ <i>Cst</i>	Includes the value of <i>Cst</i> on 5 nibbles. €123 is equivalent to \$/00123 .
¤ <i>cst</i>	Includes a system binary of value <i>Cst</i> .
¤ <i>cst</i>	Includes a system binary of value <i>Cst</i> . If this number already exists in ROM, its address is used instead.
GOIN5 <i>lab G5 lab</i> GOIN4 <i>lab G4 lab</i> GOIN3 <i>lab G3 lab</i> GOIN2 <i>lab G2 lab</i>	Same as \$(x) label-& with x=5, 4, 3 or 2. Useful to create a jump table.
SAVE	Equivalent to GOSBVL 0679B
LOAD	Equivalent to GOSBVL 067D2
RPL	Equivalent to A=DAT0.A D0+5 PC=(A)
LOADRPL	Equivalent to GOVLNG 138B9
INTOFF2	Equivalent to GOSBVL 01115
INTON2	Equivalent to GOSBVL 010E5

ERROR_C	Equivalent to GOSBVL 10F80
RES_STR	Equivalent to GOSBVL 05B7D
RES_ROOM	Equivalent to GOSBVL 039BE
[...] Size	Makes the code inside the brackets fit the given size, by adding zeros if necessary. Only three levels of brackets can be used.

2.15 Masd directives

!PATH+ DirName	Add the specified directory in the search path list.
!LINK chr	Change the char identifying link labs to <i>chr</i> .
!OFF	Shut down the screen. This speeds up the HP48 by 13%.
!LOCAL chr	Change the char identifying local labs to <i>chr</i> .
!DISP message	Display the message on the screen.
!UNIT	The output file will be a unit.
!WAROFF	Masd will not display ghost labels.
!NO CODE	Masd will not generate a \$02DCC prolog but will directly output the data.
!1-16	Switch to 1-16 mode.
!1-15	Switch to 0-15 mode.
!RPL	Switch to RPL mode.
!ASM	Switch to ASM mode.
!FL=0.a	Clear the <i>a</i> compilation flag.
!FL=1.a	Set the <i>a</i> compilation flag.
!?FL=0.a	Compile the end of the line if flag <i>a</i> is set.
!?FL=1.a	Compile the end of the line if flag <i>a</i> is clear.
!ABSOLUT Addr	Switch to absolute mode. The program begins at the address <i>Addr</i> . Note: Masd always consider the prolog \$02DCC and code length to be the beginning of the program even if !NO CODE is set.
!ABSADR Addr	If in absolute mode, add whites nibbles to continue at the specified address. If not possible, errors.
!EVEN	In absolute mode, cause an error if the directive is not on an even address.
!ADR	Masd will generate a source defining all constants and labels used in the program instead of the program.
!COMPEXP	Cause Masd to calculate all previous expressions.
!COMPEX	Cause Masd to calculate last expression.

2.16 Error messages

Invalid File	The file is not a source or a macro. (must end with a @)
Too many links	Only 256 links are supported. May be due to a recursion.
Unknown Instr.	Unknown instruction.
Bad Field	Incorrect field.
0-15 Expected	An integer between 0 and 15 is expected.
1-16 Expected	An integer between 1 and 16 is expected.
Error Stack Size	You use more than three brackets.
Size Too Small	Program between brackets is too big.
Close Size	There are more] than [.

Label Expected	A label name is expected.
Const Expected	A hexadecimal number is expected.
Can't Find	Can't find the label or the file.
Lab. Already Dec.	A label or a constant is already declared with the same name.
Need {	An opening { was expected.
Need }	A closing } was expected.
Forbidden	Operation not allowed.
Bad Expression	Error in expression.
Jump too Long	No comments.

3 Disassembler

The disassembler converts binary code into a source string. There are two ways to use it: with a Code on the stack level 1 or with two addresses (in binary) that point to the beginning and the end of the code to be disassembled.

The syntax used is Masd syntax, in mode 0-15.

Each line contains an address and an instruction.

If the **MK** system flag -7 is set (with **-7 SF2**), addresses are not shown, except for the destinations of jumps. In this case, the resulting source may be then reassembled if needed.

Example:

-7 CF2 (default)	-7 SF2
AE734 GOSBYL 0679B	GOSBYL 0679B
AE73B LC 01000	LC 01000
AE742 C-A A	*AE742
AE744 GONC AE742	C-A A
AE747 GOVLNG 138B9	GONC AE742
	GOVLNG 138B9
	@

4 Miscellaneous Utilities

These utilities are found in the MDGKER library.

Some of the following commands are marked **Dangerous!**, which means that incorrect use of them can corrupt memory. Use them at your own risk!

4.1 `-->H`

Returns a character string containing the hexadecimal numbers that form the object on stack level 1 in memory.

Example: `'A' >H` returns `"84E201014"`

4.2 `H->`

Inverse of `->H`, it converts a hexadecimal character string to the objects coded in it. **Dangerous!**

Example: `"47A209C2A2B2130" H->` returns `{ 1 }`

4.3 `S->H`

Converts a character string into its hexadecimal form (every character is converted to two characters).

Example: `"HI" S->H` returns `"8494"`

4.4 `H->S`

Inverse of `S->H`. Every couple of hexadecimal characters gives an ASCII character.

Example: `"8405" H->S` returns `"HP"`

4.5 `CD->`

Does a `->H` on the content of a Code object.

4.6 `-->CD`

Inverse of `CD->`. A hexadecimal string is placed inside a Code object.

4.7 `-->A`

Returns the memory address of the object on stack level 1.

Example: `1 >A` returns `#2A2C9h` (address of the real number 1)

4.8 A→ **A→**

Puts the object pointed to by the given address on stack. **Dangerous!**

Example: `#1B4ACh A→` returns `SIN`

4.9 PEEK **PEEK**

Reads memory. Level 2 contains the starting address (a binary number), level 1 gives the number of nibbles to be read (a binary number). A hexadecimal string is returned on the stack.

Example: `#0 #10d PEEK` returns `"2369B108DA"`

4.10 POKE **POKE**

Writes memory. Level 1 contains the hexadecimal string, level 1 the start address. **Dangerous!**

Example: `#101h "C" POKE` changes the contrast.

4.11 SREV **SREV**

Reverses the order of the characters in a string.

Example: `"HP 48" SREV` returns `"84 PH"`

4.12 PRG~LST **PRG~L**

Transforms a program to a list and vice-versa.

Example: `{ SWAP DROP } PRG~LST` returns `PRG SWAP DROP END.`

4.13 -->RAM **->RAM**

Creates a new copy of the object in memory, equivalent to `NEWOB`, but it also works with ROM objects.

Example: `~ DUP ->RAM` returns `PRG € 18AA5 € 03188 END` (the `DUP` program in ROM).

4.14 APEEK **APEEK**

Reads the address at a given address (binary number on stack level 1).

Example: `#3188h APEEK` returns `#318Dh`

4.15 R~SB **R~SB**

Transforms a real number to a system binary and vice-versa.

Example: `32 R~SB` returns `¤ 20h`

4.16 -->GROB2 **→GROB**

Transforms an object to its graphical representation.

The object is on stack level 2. A real number at level 1 gives the desired height:

- 1: Mini-font
- 2: Stack font
- 3: Big font
- 0: Big font, but if the object is an algebraic, the **MK** equation editor is used to draw it.

If the system flag2 -6 is set, the carriage returns in a string are used to draw multiple lines, otherwise if this flag -6 is cleared, the string will only be drawn on one line.

Example: 'SIN(X)' 0 →GROB2 returns **Graphic 31 × 9**

4.17 -->HEADER **→HEAD**

Defines the number of lines of the status area, 0, 1 or 2.

4.18 HEADER→ **→HEAD**

Returns the current status area height.

4.19 INPUT2 **→INPUT**

Works like **INPUT**, but uses the **Meta Kernel** environment.

4.20 DISP2 **→DISP**

Works like **DISP**, but uses the current system font (defined with **→FONT**), so that up to 9 lines can be written.

4.21 HALT2 **→HALT**

Works like **HALT**, but returns to the **Meta Kernel** stack (**HALT** returns to the standard HP stack).

Note: **DEBUG**, **SST** and **NEXT** in the PRG-RUN menu have been rewritten to work under the **Meta Kernel**.

4.22 -->S2 **→S2**

Converts an object into a character string in Masd format, in RPL mode.

4.23 -->S4 **→S4**

Converts an object into a character string, like the standard **→STR.EDIT** uses it.

4.24 CHOOSE2 **CHOOSE2**

Works like **CHOOSE**, but uses the current system font.

4.25 EDIT **EDIT**

Starts editing the object on stack level 1, with the command line editor.

4.26 VISIT **VISIT**

Starts editing the object stored in the given variable name, with the command line editor.

4.27 EDITB **EDITB**

Equivalent to **EDIT**, but uses the best adapted environment (command line, equation editor, graphic editor...), depending on the object type.

4.28 VISITB **VISITB**

Equivalent to **VISIT**, but uses the best adapted environment (command line, equation editor, graphic editor...), depending on the object type.

4.29 EQW **EQW**

Launches the equation editor with the expression given on stack level 1.

4.30 FILER **FILER**

Launches the filer.

4.31 ASM **ASM**

Launches the assembler with the source string on stack.

4.32 ER **ER**

Edits the error after assembling. There must be a string at level 2 and the errors list at level 1.

4.33 ASM→ **ASM→**

Starts the disassembler. Consult Page 75.

4.34 -->NDISP **NDISP**

Defines the maximum number of lines taken by an object displayed on stack (including grobs and algebraics).

4.35 -->FONT **FONT**

Sets the font object on stack level 1 as the system font.

4.36 FONT-> **FONT**

Returns the current system font on the stack.

4.37 FONT8, FONT6 **FONT8 FONT6**

Returns the standard **Meta Kernel** fonts 6 and 8.

4.38 KERNEL? **KERNEL?**

Returns nothing. It is used to test if the **Meta Kernel** is present, if a program needs it.

4.39 SF2, CF2, FS?2, FC?2, FS?C2, FC?C2, STOF2, RCLF2

SF2 CF2 FS?2 FC?2 FS?C2 FC?C2 STOF2 RCLF2

Equivalent to **SF**, **CF**, **FS?**, **FC?**, **FS?C**, **FC?C**, **RCLF** and **STOF**, with the new **Meta Kernel** flags. Consult Page 21.

4.40 SREPL **SREPL**

Does a find/replace in a character string. The main string is at level 3, the find string at level 2 and the replace string at level 1. Every occurrence of the find string in the main string will be replaced by the replace string. The find and replace strings may have different lengths.

4.41 AR~LST **AR~LST**

Converts an array into a list of lists and vice-versa.

E.g. `[[1 2] [3 4]] ← AR~LST → {{ 1 2 } { 3 4 }}`.

Note: All the objects must be of the same type.

4.42 DIMS **DIMS**

Puts the size of the array contained in a list of lists on stack level 2 and 1 on stack level one. If it is **not** an array, returns 0.

`{ 1 { 2 3 } } DIMS → 0`

```
<< 1 'A' "ABC" >
< 2 NOVAL 3 >> DIMS →< 2 3 > 1
```

4.43 -->MINIFONT

Sets the mini-font object on stack level 1 as the system mini-font.

4.44 MINIFONT →

Returns the current system mini-font on the stack.

4.45 FNT2GRB.PRG

Converts an **MK** font to a grob font (common 6,7 or 8 x 2048 grob).

This program is on the floppy disk.

4.46 GRB2FNT.PRG

Converts a grob font to an **MK** font. The grob in at level 3, the name is a character string at level 2 and the identifier a real number at level 1.

This program is on the floppy disk.

Appendixes

1 Introduction to assembly language

1.1 Introduction

Our purpose here is to help you translate an algorithm in Saturn assembly language, using Masd. You are supposed to already know the programming process, and RPL language.

If you want to learn how to program, there are several books to do it, like Wirst's 'The Art of Programming'.

We will try to give you the basis to build little programs. If you want to gain more experience, look at other people's programs, and read more advanced books on this subject.

Many informations here are taken from the excellent French book 'Voyage au Centre de la HP48' by Paul Courbis (Angkor).

1.2 What is assembly language (asm) ?

Asm is the only language directly readable by the microprocessor. Therefore, it is the fastest and the most powerful language. Each processor has its own asm. Asm is composed of binary numbers interpreted by the processor, each group of numbers giving instructions to be processed.

To make it easily readable by humans, each group of numbers is represented by a mnemonic. Assemblers (like Masd) translate these mnemonics to binary code, that the processor can execute. They sometimes add macro instructions or even structures to ease the programmer's work.

1.3 The Saturn processor

1.3.1 Generalities

SATURN is the name of a processor made by NEC. It is used in the HP48. It is a 4-bit processor (it can process 4 bits at each clock tick), with 4 64-bit working registers, 5 64-bit saving registers, 2 20-bit memory pointers, an internal 8-level return stack, 16 software flags, 4 hardware flags, 1 field register, 1 output register, 1 input register and 1 20-bit program counter.

A bit is a binary digit, it can takes only two values 0 or 1.

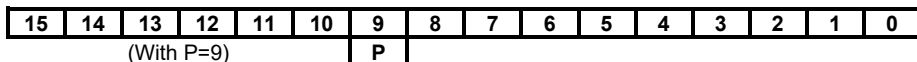
A nibble is a 4-bit value, from 0 to 15.

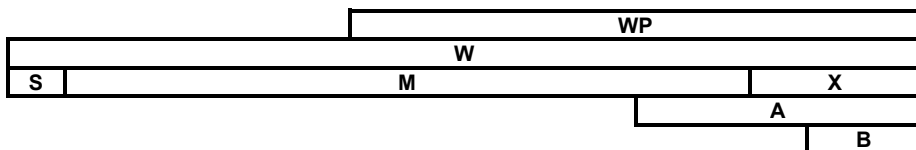
1.3.2 Working and saving registers

These registers are 64-bit (16-nibble) registers divided into fields.

A field is a part of a register; a group of nibbles. Two fields may overlap. They can have different lengths.

In a Saturn register, there are six fixed fields and two variable fields (depending on P register):





Example: Working with the A field only affects the 20 lowest bits of the register.
 In this text, Cb means register C, field B.

1.3.3 P register

P is a 4-bit register which indicates the WP field length, the P field location, and the first modified nibble for the instructions LA and LC.

1.3.4 Flags register

The flags register is a 16-bit register, composed of 16 flags, which can be only 0 or 1. Their use is similar to RPL flags (CF, SF...).

1.3.5 Return stack

The return stack is a set of eight 20-bit registers. Only one of them is accessible directly (like the RPL stack).

When a subroutine is called (with **GOSUB**), the return address (just after the **GOSUB**) is pushed onto this stack. And when a **RTN** is encountered, the Saturn pops this address back from the stack and jumps to it.

Because there are only eight levels, building large programs with a lot of subroutine levels must not overload the return stack.

It is also possible to push a value with **RSTK=C** and to pop a value with **C=RSTK**.

1.3.6 Memory pointers

The three memory pointers D0, D1 and PC are 20-bit registers, which contains addresses.

D0 and D1 are used to access memory. You can load any address in them and then read memory.

PC is the Program Counter, it holds the address of the next instruction to be processed. It is modified at every jump (**GOTO**, **GOSUB**...), or directly with instructions such as **PC=C**.

a) Memory

Before examining pointers, let's see how memory is seen through HP asm.

HP memory can be compared to a ring of cells. Each cell contains one nibble. They are numbered from 0 to 524287 (20 bits). It is a ring, so the cell 524287 is followed by the cell 0. The number of a cell is called its address.

b) Pointers

A pointer is a register that contains an address. On HP48, a pointer is 20 bits (5 nibbles) long. Therefore we can access any memory cell with one pointer (Whereas on PC, we need two pointers).

c) Memory accesses

On the HP48, memory inputs and outputs are made through the A and C registers, D0 and D1 indicating which cell will be accessed.

It is possible to access more than one cell at one time, as up to an entire register (16 nibbles) can be loaded.

A=DAT0 A reads 5 nibbles at the address pointed by D0, then the data are put in the A field of A (Aa).

DAT1=C 12 writes 12 nibbles at the address pointed by D1. The data written are the 12 lowest nibbles of C (nibbles 0 to 11).

Note that the Saturn processor inverts the order of the data read or written. E.g. if Ca contains 84571 and D1 contains 80200, **DAT1=C A** will fill memory as follow:

80200 1	80201 7	80202 5	80203 4	80204 8
------------	------------	------------	------------	------------

The inversion is done at each read or write, so if you write and then read at the same address, using the same field, you will retrieve the same data.

1.4 Starting and stopping a Program

Starting to write an asm program looks difficult to beginners, because it requires comprehension of the RPL system, in order to avoid destroying other data.

To start a program, all the system data must be saved somewhere in memory, where they will be read back, after your program has finished, in order to continue normal operation. These data are contained in the registers Ba, Da, D0 and D1.

This saving is done with the Masd instruction **SAVE** (or the standard Saturn **GOSBYL 0679B**). Do it only one time, before doing any other processes.

At the end of your program, you have to restore the system data, with the instruction **LOAD** (or **GOSBYL 067D2**).

You can then exit to the RPL system with **RPL** (or **A=DAT0 A D0=D0+5 PC=<A>**).

These two instructions can be compacted into the single instruction **LOADRPL** (or **GOVLNG 138B9**).

The shortest program is:

```
"SAVE
% This program does nothing !
LOADRPL
e"
```

1.5 Working with the RPL stack

The RPL stack (the stack displayed on the screen) is stored in memory as a stack of addresses, each address points to an object in RAM or ROM. In order to reach an object, its address must be first read.

The stack pointer is in D1, pointing to the address of the first stack level. D1+5 points to the second level, and so on. A 00000 marks the end of the stack. The command **SAVE** saves D1 (and other RPL registers) in reserved RAM, so if you modify D1, and then need to read the RPL stack, do a **LOAD** before, which

restores D1. And don't forget to do a **LOAD** before exiting from your program, otherwise D1 could point somewhere in memory, that may crash your HP.

Note: If a program needs to read the stack, it is recommended to do the argument tests in RPL before starting the asm code.

Example: To read the value of a system binary on the first stack level, do the following:

```

C=DAT1 A % Reads the address of the first level in Ca
CD1EX % D1 now points on the prolog of this object
% Note that D1 is saved in Ca
D1=D1+5 % D1 points on the value of the SB
A=DAT1 A % Reads the value in Aa
CD1EX % Restores D1
  
```

To do a DROP, you just have to make the stack start 5 nibbles upper:

```

D1=D1+5 % The stack starts upper, so lower levels are dropped
D=D+1 A % Da contains the number of free 5 nibbles-blocks,
% so you shall increment it
  
```

Da is also saved by **SAVE** and restored by **LOAD**.

If you do a **LOAD** after, the old value of D1 is retrieved. In order to effectively drop an element, do **LOAD D1+5 D+1.A** at the end of your program (before the **RPL** command), or do **LOAD D1+5 D+1.A SAVE** anywhere in your program.

To put an element on the stack, you just have to create a new stack level. E.g. if the object address is in Aa:

```

LOAD % loads the RPL registers
D=D-1.A SKIPNC % Tests free memory
< C=0.A LC 1 ERREUR_C % Insufficient mem
D1=D1-5 % Adds a stack level
DAT1=A A % Puts the object address
SAVE % Saves the RPL registers
  
```

1.6 Useful routines

Rf = field f of register R.

The functions ending with an EXIT (RPL return) shall be called with a GOVLNG, whereas the other function must be called with GOSBVL.

Note: you can replace **GOSBVL add RTN** by **GOVLNG add**, which is shorter and saves one RSTK level.

1.6.1 Stack exchanges

SB = System Binary (Prolog 02911).

Function	Address	Quick help	Modified registers
POP A	06641	Drops 1 SB, stores its value in Aa.	D1, Da, Ca, Aa
POP A C	03F5D	Drops 2 SB, stores their values in Aa and Ca.	D1, Da, Ca, Aa, 1 RSTK
PUSH R0	06537	Pushes R0a in a SB.	D0, D1, Aw, Ba, Ca, Da

PUSH R0, R1	06529	Pushes R0a and R1a in two SB.	D0, D1, Aw, Ba, Ca, Da, R0w
DROP EXIT	03249	Drops one object and exits.	
LOAD TRUE EXIT	25CE1	Loads RPL registers, pushes TRUE and exits.	
LOAD FALSE EXIT	26FAE	Loads RPL registers, pushes FALSE and exits.	
PUSH A EXIT	0357C	pushes Aa in a SB and exits.	
PUSH R0, R1 EXIT	03F14	pushes R0a and R1a in two SB and exits.	
PUSH TRUE EXIT	620C3	pushes TRUE and exits.	
PUSH FALSE EXIT	620DC	pushes FALSE and exits.	
PUSH CARRY EXIT	620D9	If carry is set, pushes a TRUE, else a FALSE; then EXITs.	

1.6.2 Calculations

Ba=Aa*Ca	03991		Aa, Ba, Ca
Cw,Aw=Aw*Cw	53EE4		Aw, Cw, Dw
Aw=Cw=Aw/Cw Bw=Remind.	65807		Aw, Bw, Cw, P
C=C/5	06A8E		A11, B11, C11, P
Aa=CRC(D0, Aa)	05981	Stores in Aa, the CRC of the memory area starting at D0, on Aa nibbles.	D1, D0, Aa, Ca, P

1.6.3 Miscellaneous

BEEP(Da,Ca)	017A6	Bips Ca ms, with a frequency of Da Hz.	D0, D1, P, Aw, Bw, Cw, Dw, R1w, R2w, R3w
EMPTYKBUF	00D57	Empties the keyboard buffer.	D1, Ca
BUSY ON	01BEB	Lights on the busy annunciator.	D0, Cb
BUSY OFF	42359	Lights off the busy annunciator.	D0, Cb
D0=@Screen	01C31	Stores the address of the first pixel of the screen in D0.	D0, Aa, Ca
D0=@Menu	01C58	stores the address of the first pixel of the menu in D0.	D0, Aa, Ca
DISP	11D8F	Displays Ca characters at D1 in a grob pointed by D0, which has a width of Da, starting at the Character number Ba.	XM, P, D0, D1, Aa, Ba, Cw, Da

1.6.4 Memory operations

DAT1=0 (Ca)	0675C	Resets to 0 a memory zone pointed by D1, on Ca nibbles.	D1, P, Aw, Ca
RES ROOM	039BE	Reserves Ca nibbles in temporary RAM. D0 indicates the starting.	ST0,1,10 D0, D1, P, Aw, Bw, Cw, Dw

RES STRING	05B7D	Reserves a character string of Ca nibbles in temporary RAM. R0a points on the prolog and D0 on the first character.	ST0,1,10 D0, D1, P, Aw, Bw, Cw, Dw
GARB COL	0313E	Does a garbage collect.	ST0,1,10, D0, D1, Aw, Bw, Cw, Dw
FREEROOM	16671	Resizes the object pointed by R0a (in temporary RAM), so that it ends at D0.	D0, D1, Aw, Ca, Ba, R1w
SKIP OBJ	03019	D0 pointing on an object, it is moved to the next object.	D0, ST1, Aa, Ca

1.7 Saturn instruction set

There are four columns in this table.

The first column may be empty, or contains a M or a star. M indicates that the calculation mode is used (**SETDEC** and **SETHex**). A star indicates that the carry is modified by the instruction.

The second column contains the instruction mnemonic.

The third column indicates the fields which can be used with the instruction. f is any field, a or b is any field but A, d is a nibble value (0-15 or 1-16), x is a nibble value (0-15 only).

The last column is the execution length in Saturn cycles. n is the number of nibbles in the used field. If the time is not integer, take IP(time) if the instruction is on an even address or IP(time)+1 if the instruction is on an odd address. For tests instructions, there are two times separated by a slash, the first is for a true test, the second for a false test. For memory instructions, the two times indicate the instruction length and the memory operation length.

This list is taken from the book 'Voyage au centre de la HP48 GX' by Paul Courbis, Angkor Editions.

*	?A#0	A	21.5/13.5
*	?A#0	a	16.5/8.5+q
*	?A#C	A	21.5/13.5
*	?A#C	a	16.5/8.5+q
*	?A<=B	A	21.5/13.5
*	?A<=B	b	16.5/8.5+q
*	?A<B	A	21.5/13.5
*	?A<B	b	16.5/8.5+q
*	?A=0	A	21.5/13.5
*	?A=0	a	16.5/8.5+q
*	?A=C	A	21.5/13.5
*	?A=C	a	16.5/8.5+q
*	?A>=B	A	21.5/13.5
*	?A>=B	b	16.5/8.5+q
*	?A>B	A	21.5/13.5
*	?A>B	b	16.5/8.5+q
*	?ABIT=0	d	20.5/12.5
*	?ABIT=1	d	20.5/12.5
*	?B#0	A	21.5/13.5
*	?B#0	a	16.5/8.5+q
*	?B#A	A	21.5/13.5
*	?B#A	a	16.5/8.5+q
*	?B<=C	A	21.5/13.5
*	?B<=C	b	16.5/8.5+q
*	?B<C	A	21.5/13.5
*	?B<C	b	16.5/8.5+q
*	?B=0	A	21.5/13.5
*	?B=0	a	16.5/8.5+q
*	?B=A	A	21.5/13.5
*	?B=A	a	16.5/8.5+q
*	?B>=C	A	21.5/13.5
*	?B>=C	b	16.5/8.5+q
*	?B>C	A	21.5/13.5
*	?B>C	b	16.5/8.5+q
*	?C#0	A	21.5/13.5
*	?C#0	a	16.5/8.5+q
*	?C#B	A	21.5/13.5
*	?C#B	a	16.5/8.5+q
*	?C#D	A	21.5/13.5
*	?C#D	a	16.5/8.5+q
*	?C<=A	A	21.5/13.5
*	?C<=A	b	16.5/8.5+q
*	?C<A	A	21.5/13.5
*	?C<A	b	16.5/8.5+q
*	?C=0	A	21.5/13.5
*	?C=0	a	16.5/8.5+q
*	?C=B	A	21.5/13.5
*	?C=B	a	16.5/8.5+q
*	?C=D	A	21.5/13.5
*	?C=D	a	16.5/8.5+q
*	?C>=A	A	21.5/13.5
*	?C>=A	b	16.5/8.5+q
*	?C>A	A	21.5/13.5
*	?C>A	b	16.5/8.5+q
*	?CBIT=0	d	20.5/12.5
*	?CBIT=1	d	20.5/12.5
*	?D#0	A	21.5/13.5
*	?D#0	a	16.5/8.5+q
*	?D<=C	A	21.5/13.5
*	?D<=C	b	16.5/8.5+q
*	?D<C	A	21.5/13.5
*	?D<C	b	16.5/8.5+q
*	?D=0	A	21.5/13.5

*	?D=0	a	16.5/8.5+q
*	?D>=C	A	21.5/13.5
*	?D>=C	b	16.5/8.5+q
*	?D>C	A	21.5/13.5
*	?D>C	b	16.5/8.5+q
*	?HST=0	x	15.5/7.5
*	?MP=0		15.5/7.5
*	?P#	n	15.5/7.5
*	?P=	n	15.5/7.5
*	?SB=0		15.5/7.5
*	?SR=0		15.5/7.5
*	?ST=0	d	16.5/8.5
*	?ST=1	d	16.5/8.5
*	?XM=0		15.5/7.5
*M	A=-A	a	4.5+q
*M	A=-A	A	8
*M	A=-A-1	a	4.5+q
*M	A=-A-1	A	8
	A=0	a	4.5+q
	A=0	A	8
	A=AIB	f	6+q
	A=AIC	f	6+q
	A=A&B	f	6+q
	A=A&C	f	6+q
*M	A=A+1	a	4.5+q
*M	A=A+1	A	8
*M	A=A+A	a	4.5+q
*M	A=A+A	A	8
*M	A=A+B	a	4.5+q
*M	A=A+B	A	8
*M	A=A+C	a	4.5+q
*M	A=A+C	A	8
*	A=A+x+1	f	8+q
*M	A=A-1	a	4.5+q
*M	A=A-1	A	8
*M	A=A-B	a	4.5+q
*M	A=A-B	A	8
*M	A=A-C	a	4.5+q
*M	A=A-C	A	8
*	A=A-x-1	f	8+q
	A=B	a	4.5+q
	A=B	A	8
*M	A=B-A	a	4.5+q
*M	A=B-A	A	8
	A=C	a	4.5+q
	A=C	A	8
	A=DAT0	A	2.5,3.5
	A=DAT0	a	20+p,1+q/2
	A=DAT0	B	19.5
	A=DAT0	x+1	19+q,1+q/2
	A=DAT1	A	23.5,3.5
	A=DAT1	a	20+q,1+q/2
	A=DAT1	B	19.5
	A=DAT1	x+1	19+q,1+q/2
	A=IN		8.5
	A=PC		11
	A=R0	f	9+q
	A=R0	W	20.5
	A=R1	f	9+q
	A=R1	W	20.5
	A=R2	f	9+q
	A=R2	W	20.5
	A=R3	f	9+q

A=R3	W	20.5
A=R4	f	9+q
A=R4	W	20.5
ABEX	a	4.5+q
ABEX	A	8
ABIT=0	d	7.5
ABIT=1	d	7.5
ACEX	a	4.5+q
ACEX	A	8
AD0EX		9.5
AD0XS		8.5
AD1EX		9.5
AD1XS		8.5
APCEX		19
AR0EX	f	9+q
AR0EX	W	20.5
AR1EX	f	9+q
AR1EX	W	20.5
AR2EX	f	9+q
AR2EX	W	20.5
AR3EX	f	9+q
AR3EX	W	20.5
AR4EX	f	9+q
AR4EX	W	20.5
* ASL	a	4.5+q
* ASL	A	8
ASLC		22.5
ASR	a	4.5+q
ASR	A	8
ASRB	f	8.5+q
ASRB	W	21.5
ASRC		22.5
*M B=-B	a	4.5+q
*M B=-B	A	8
*M B=-B-1	a	4.5+q
*M B=-B-1	A	8
B=0	a	4.5+q
B=0	A	8
B=A	a	4.5+q
B=A	A	8
B=BIA	f	6+q
B=BIC	f	6+q
B=B&A	f	6+q
B=B&C	f	6+q
*M B=B+1	a	4.5+q
*M B=B+1	A	8
*M B=B+A	a	4.5+q
*M B=B+A	A	8
*M B=B+B	a	4.5+q
*M B=B+B	A	8
*M B=B+C	a	4.5+q
*M B=B+C	A	8
* B=B+x+1	f	8+q
*M B=B-1	a	4.5+q
*M B=B-1	A	8
*M B=B-A	a	4.5+q
*M B=B-A	A	8
*M B=B-C	a	4.5+q
*M B=B-C	A	8
* B=B-x-1	f	8+q
B=C	a	4.5+q
B=C	A	8
*M B=C-B	a	4.5+q
*M B=C-B	A	8

BCEX	a	4.5+q
BCEX	A	8
* BSL	a	4.5+q
* BSL	A	8
BSLC		22.5
BSR	a	4.5+q
BSR	A	8
BSRB	f	8.5+q
BSRB	W	21.5
BSRC		22.5
BUSCB		10
BUSCC		8.5
BUSCD		10
* C+P+1		9.5
C=0	a	4.5+q
C=0	A	8
C=A	a	4.5+q
C=A	A	8
*M C=A-C	a	4.5+q
*M C=A-C	A	8
*M C=-C	a	4.5+q
*M C=-C	A	8
*M C=-C-1	a	4.5+q
*M C=-C-1	A	8
C=B	a	4.5+q
C=B	A	8
C=C!A	f	6+q
C=C!A	f	6+q
C=C!D	f	6+q
C=C&A	f	6+q
C=C&A	f	6+q
C=C&D	f	6+q
*M C=C+1	a	4.5+q
*M C=C+1	A	8
*M C=C+A	a	4.5+q
*M C=C+A	A	8
*M C=C+B	a	4.5+q
*M C=C+B	A	8
*M C=C+C	a	4.5+q
*M C=C+C	A	8
*M C=C+D	a	4.5+q
*M C=C+D	A	8
* C=C+x+1	f	8+q
*M C=C-1	a	4.5+q
*M C=C-1	A	8
*M C=C-A	a	4.5+q
*M C=C-A	A	8
*M C=C-B	a	4.5+q
*M C=C-B	A	8
*M C=C-D	a	4.5+q
*M C=C-D	A	8
* C=C-x-1	f	8+q
C=D	a	4.5+q
C=D	A	8
C=DAT0	A	23.5,3.5
C=DAT0	a	20+q,1+q/2
C=DAT0	B	19.5
C=DAT0	x+1	19+q,1+q/2
C=DAT1	A	23.5,3.5
C=DAT1	a	20+q,1+q/2
C=DAT1	B	19.5
C=DAT1	x+1	19+q,1+q/2
C=ID		13.5
C=IN		8.5

C=P	x	8
C=PC		11
C=R0	f	9+q
C=R0	W	20.5
C=R1	f	9+q
C=R1	W	20.5
C=R2	f	9+q
C=R2	W	20.5
C=R3	f	9+q
C=R3	W	20.5
C=R4	f	9+q
C=R4	W	20.5
C=RSTK		9
C=ST		7
CBIT=0	d	7.5
CBIT=1	d	7.5
CD0EX		9.5
CD0XS		8.5
CD1EX		9.5
CD1XS		8.5
CDEX	a	4.5+q
CDEX	A	8
CETDEC		4
CETHEX		4
CLRHST		4.5
CLRST		7
CONFIG		13.5
CPCEX		19
CPEX	x	8
CR0EX	f	9+q
CR0EX	W	20.5
CR1EX	f	9+q
CR1EX	W	20.5
CR2EX	f	9+q
CR2EX	W	20.5
CR3EX	f	9+q
CR3EX	W	20.5
CR4EX	f	9+q
CR4EX	W	20.5
* CSL	a	4.5+q
* CSL	A	8
CSLC		22.5
CSR	a	4.5+q
CSR	A	8
CSRB	f	8.5+q
CSRB	W	21.5
CSRC		22.5
CSTEX		7
D=0	a	4.5+q
D=0	A	8
D=C	a	4.5+q
D=C	A	8
*M D=C-A	a	4.5+q
*M D=C-A	A	8
*M D=D	a	4.5+q
*M D=D	A	8
*M D=D-1	a	4.5+q
*M D=D-1	A	8
D=DIC	f	6+q
D=D&C	f	6+q
*M D=D+1	a	4.5+q
*M D=D+1	A	8
*M D=D+C	a	4.5+q
*M D=D+C	A	8

*M D=D+D	a	4.5+q
*M D=D+D	A	8
* D=D+x+1	f	8+q
*M D=D-1	a	4.5+q
*M D=D-1	A	8
*M D=D-C	a	4.5+q
*M D=D-C	A	8
* D=D-x-1	f	8+q
D0=	pq	6
D0=	pqrs	9
D0=	pqrst	10.5
D0=A		9.5
D0=AS		8.5
D0=C		9.5
D0=CS		8.5
D0=D0-	n+1	8.5
* D0=D0+	n+1	8.5
D1=	pq	6
D1=	pqrs	9
D1=	pqrst	10.5
D1=A		9.5
D1=AS		8.5
D1=C		9.5
D1=CS		8.5
D1=D1-	n+1	8.8
* D1=D1+	n+1	8.5
DAT0=A	A	19.5
DAT0=A	a	19+q
DAT0=A	B	16.5
DAT0=A	x+1	18+q
DAT0=C	A	19.5
DAT0=C	a	19.5
DAT0=C	B	16.5
DAT0=C	x+1	18+q
DAT1=A	A	19.5
DAT1=A	a	19+q
DAT1=A	B	16.5
DAT1=A	x+1	18+q
DAT1=C	A	19.5
DAT1=C	a	19.5
DAT1=C	B	16.5
DAT1=C	x+1	18+q
DSL	a	4.5+q
DSL	A	8
DSL C		22.5
DSR	a	4.5+q
DSR	A	8
DSRB	f	8.5+q
DSRB	W	21.5
DSRC		22.5
GOC	xy	12.5/4.5
GOLONG	pqrs	17
GONC	xy	12.5/4.5
GOSBVL	pqrst	19.5
GOSUB	xyz	15
GOSUBL	pqrs	18
GOTO	xyz	14
GOVLNG	pqrst	18.5
HST=0	x	4.5
INTOFF		7
INTON		7
LA	h0..hx	(15+3q)/2
LC	h0..hx	3+3q/2
MP=0		4.5

OUT=C		7.5
OUT=CS		5.5
P=	n	3
P=C	x	8
* P=P+1		4
* P=P-1		4
PC=(A)		26,3.5
PC=(C)		26,3.5
PC=A		19
PC=C		19
R0=A	f	9+q
R0=A	W	20.5
R0=C	f	9+q
R0=C	W	20.5
R1=A	f	9+q
R1=A	W	20.5
R1=C	f	9+q
R1=C	W	20.5
R2=A	f	9+q
R2=A	W	20.5
R2=C	f	9+q
R2=C	W	20.5
R3=A	f	9+q
R3=A	W	20.5
R3=C	f	9+q

R3=C	W	20.5
R4=A	f	9+q
R4=A	W	20.5
R4=C	f	9+q
R4=C	W	20.5
RESET		7.5
RSI		8.5
RSTK=C		9
RTI		11
RTN		11
RTNC		12.5/4.5
* RTNCC		11
RTNNC		12.5/4.5
* RTNSC		11
RTNSXM		11
SB=0		4.5
SHUTDN		6.5
SR=0		4.5
SREQ?		9.5
ST=0	d	5.5
ST=1	d	5.5
ST=C		7
UNCNFG		14.5
XM=0		4.5

2 Introduction to System RPL

2.1 Generalities

The System RPL integrated in the HP48, but which is not directly accessible by users. The SystemRPL has the same structure as RPL.

The RPL language and the standard environment are mainly programmed in SystemRPL, which shows its power and complexity.

So SystemRPL can do much more things than RPL and, like asm, can easily cause "Memory Lost".

In this section, **12345** represents an external. To use it, type **\$ 12345h**, or **€ 12345**.

2.2 External "Theory"

In the standard environment, external instructions (or simply 'externals') are displayed as **External** (that's why we call the language 'External'). Note that every RPL instruction can also be considered as an external (it has the same structure, only the display changes).

An external is the address of an object.

An object always begins with a prolog, and a prolog is in fact the address of the ML program that handles this object type.

So an external is the address of the address of an asm program.

E.g.: SIN is the external of address 1B4AC. At this address, we find the prolog 02D9D, and finally at the address 02D9D is the asm program which executes every program object.

Note: There are also many externals which are 'entry points'. Their prolog is simply the address of the asm program (In ROM, this program is just after its address). E.g.: The external DUP (**03188**) points directly onto 0318D, where the asm DUP is.

2.3 Writing a System RPL program

There is a list of entry points at the end of this section. When you know control structures and how to find the right instructions in this list, programming in SystemRPL is as simple as RPL.

With the **MK**, programs can be directly edited in command line, using the command library. Bu for real projects, it is better to work with the Masd in two times.

First write the program in a character string,

To compile this program, put the string on the stack and type **ASM**.

The program is put on the stack, where it can be saved, tested...

Here is a simple external program:

```
!NO CODE
!RPL
:: (Program prolog)
```



```

500 (pushes the system binary <500d> on the stack)
ZERO_DO      * FOR 0 to SB1-1
  INDEX@     * Recall the loop index
  #>$       * SB->S, convert the system binary into a string
  DISPROW1   * 1 DISP
  LOOP      * NEXT
;           * End of the SysRpl program (colon)
@

```

Note that 'normal' RPL commands can be used in SystemRPL, but for all these commands, there are equivalent entry points.

In this program, there is a System Binary (or SB) which is an integer coded on five nibbles. It is the most used object, as lots of functions wait for them as arguments.

It is important to note that before the **DISPROW1**, there is a **#>\$** which converts a SB into a character string. In RPL, DISP converts automatically every object into a string to display it. In SysRpl, most functions do NOT test the arguments, so the programmer has to do every conversion as needed. The external **DISP** supposes there is a character string on the stack; if there is no objects, or if there is another object type, be sure to see the well known question **Try To Recover Memory?**, whereas in RPL, the program would have ended with a **Too Few Arguments** or a **Bad Argument Type** error.

ALWAYS CHECK THE ARGUMENTS!




2.4 Instructions

The entry points list is very long, with more than 1700 entry points and 6000 externals. It is impossible to know all of them, but it is useful to learn some most used of them (mainly stack manipulations and structures).

Every RPL command has one or more external equivalent, because RPL which can handle multiple object types (e.g **DISP**, **+**) have an entry points equivalent for every object type (there are 27 external **+**!).

A good way to find an entry points is to decompile a RPL command.

For example, let's try to find the entry points **DUP**.

Put a RPL **DUP** on the stack by typing **~ DUP**. To type **~**, press    v.

Copy the DUP program in temporary RAM with **→RAM**.

Now the DUP program is on the stack:

Transform it into SysRpl syntax using **→S2**

```

!NO CODE
!RPL
::
  CK1
  DUP
;
@"

```

Most RPL commands programs are made like this: a type control and the systemRpl command. So **CK1** controls there is at least one object on the stack.

Every RPL command can be decompiled like that, more or less simply depending on the number of arguments handled by the command, so it is important to know the control structures.

2.5 Control structures

2.5.1 Booleans

A boolean is a binary value (true or false). The **MK** defines two words for boolean values: **TRUE** and **FALSE**.

They are mainly used in control structures, because they are arguments of If-Then-Else, Do-Until and While-Repeat.

They are generated by typing their name, or as a result of a test function (equality, superiority...).

2.5.2 Tests

In SystemRpl, there are lots of control structures, but only the most used ones are detailed here: IF THEN, IF ELSE and IF THEN ELSE.

IF THEN: **IF**. Takes a boolean on the stack and evaluates the following object if the boolean is **TRUE**. Example:

```
:: IFT :: "True" DISPROW1 ; ;.
```

IF ELSE: **?SKIP**. Equivalent to a NOT IF THEN.

IF THEN ELSE: **ITE**. Takes a boolean on the stack, evaluates the following object if the boolean is **TRUE**, or the second following object if the boolean is **FALSE**.

```
::  
    ITE  
    "This is TRUE"  
    :: "False !" 17 337 setbeep ;  
    DISPROW1  
;
```

2.5.3 Loops

There are two common loops: FOR NEXT, WHILE REPEAT END.

FOR NEXT:

```
#45  
ZERO_DO      *FOR 0 to SB-1  
...          *Your loop here  
LOOP         *NEXT
```

WHILE REPEAT END :

```
BEGIN        *WHILE  
...          * program which pushes a boolean on the stack  
WHILE        *REPEAT  
...          * your program here  
REPEAT       *END
```

2.6 Unnamed local variables

There are two types of local variables: named and unnamed. Named variables are used like in RPL. Unnamed variables are faster.

They are indexed in the same order of their creation.

CACHE creates a local environment. The stack level 1 contains the character 0 (**K\$ 0**), level 2 is a SB which is the number of variables to create, and level 3 to n+2 are initialization values.

More than one **BIND** environment can be stacked, but only the last created one is active.

ABND leaves the top-most environment. All **BINDs** must be **UNBNDed** before the program ends, otherwise the system will crash.

To ease the use of unnamed variables, the **MASD** provide a new mechanism (Check page 62)

Example:

```
{
LAM Variable1      * Name of the Variable
LAM Variable2
...
LAM Variablen
}
BIND (Create the local variables)
LAM Variable1      (Recall the Variable1 content)
FALSE
' LAM Variable1 STOLAM
                    (Stores the FALSE boolean into the variable Variable1)

ABND (Purges the local variable)

{{ Nom1 Nom2 Nom3 }}
                    (Creates 3 unnamed local variable)

Nom1 (Recalls the No1 variable's content)
TRUE !Nom1 (Stores the boolean TRUE into the No1 variable)

1GETLAM (Recall the No1 variable's content)
FALSE
1PUTLAM (Store the FALSE boolean into the No1 local variable)
ABND (Purges the local variable)
```

2.7 Argument control

Every RPL program begins with the argument control. It is usually made in two steps: first check the number of arguments on stack, then branch in the program depending on the argument types.

Here are the externals which test that the required number of objects is on the stack (it also saves the program name, for error messages):

Checks 0 object (only saves the name)	CK0
Checks 1 object	CK1
Checks 2 objects	CK2
Checks 3 objects	CK3
Checks 4 objects	CK4
Checks 5 objects	CK5
Checks n objects (n is a SB on stack level 1)	CKN

Then, the following structure branches to different part of the program, depending on the object types.

```

::
  CK&&DISPATCH
  #SB_prg1
  :: program 1 ;
  #SB_prg2
  :: program 2 ;
  ...
  #SB_prgn
  :: program n ;
;

```

Each SB describes the object types found on stack, which are accepted by the program, with the following code:

Algebraic	#9
Array	#4
Backup	#9F
Binary integer	#B
Character	#6F
Code	#7F
Complex	#2
Directory	#2F
Extended pointer	#BF
Global name	#6
Graphic	#C
Library	#8F
Library data	#AF
Linked array	#5F
List	#5

Local name	#7
Long complex	#4F
Long real	#3F
Program	#8
Real	#1
Reserved 1	#CF
Reserved 2	#DF
Reserved 3	#EF
String	#3
System binary	#1F
Tagged	#D
Unit	#E
Xlib name	#0F
Any type	#0

In order to test the stack for three character strings, do as follows:

```

::
  CK3 *check 3 objects
  CK&&DISPATCH1 (dispatch)
  #333 (string, string, string)
  ::
    (Your program)
  ;
;

```

2.8 Entry points list

This list is not complete. Anyway, it may be sufficient to start programming, and to make good programs. If you don't find the entry points you need, remember that the best place to find it is in the HP48 ROM itself!

This list is extracted from the RPL Manual by Hewlett Packard, with their permission. The original RPL Manual, which contains much more explanations about SysRPL, can be found on the floppy disk (in different formats).

The following list is sorted by object types. The same list is also available, sorted by addresses and by function names, on the floppy disk.

2.8.1 Notations

Ob	Any object
Id	Identifier Object
Lam	Temporary Identifier Object
romptr	ROM Pointer Object
#	Binary Integer Object
%	Real Object
%%	Extended Real Object
C%	Complex Object
C%%	Extended Complex Object
array	Array Object
lnkarray	Linked Array Object
\$	Character String Object
hxs	Hex String Object
chr	Character Object
ext	External Object
code	Code Object
primcode	Primitive Code Object
::	Secondary Object
{}	List Object
symp	Symbolic Object
comp	Any Composite Object (list, secondary, symbolic)
rrp	Directory Object
tagged	Tagged Object
flag	TRUE/FALSE

2.8.2 Binary Integers

a) Built-in Binary Integers

Addr	Word	Stack diagram
64E0A	2EXT	→ #EE
64DE2	2GROB	→ #CC
64C66	2LIST	→ #55
04099	2REAL	→ #11
64E64	3REAL	→ #111
34301	Attn#	→ #A03
64CE8	BINT_115d	→ #73
64CF2	BINT_116d	→ #74
64D06	BINT_122d	→ #7A
64D1A	BINT_130d	→ #82
64D24	BINT_131d	→ #83
64BE4	BINT_65d	→ #41
64C84	BINT_91d	→ #5B
64C8E	BINT_96d	→ #60
64E1E	BINT253	→ #
64E28	BINT255d	→ #FF
64BDA	BINT40h	→ #40
64D10	BINT80h	→ #80
64DD8	BINTC0h	→ #C0
64CC0	char	→ #6F
6506C	Connecting	→ #C0A
0403F	EIGHT	→ #8
040A3	EIGHTEEN	→ #12
64C34	EIGHTY	→ #50
64C3E	EIGHTYONE	→ #51
0405D	ELEVEN	→ #B
0407B	EXT	→ #E
6508A	EXTOBOB	→ #E00
64DF6	EXTREAL	→ #E1
64E00	EXTSYM	→ #EA

04085	FIFTEEN	→ #F
64B4E	FIFTY	→ #32
64B9E	FIFTYEIGHT	→ #3A
64B80	FIFTYFIVE	→ #37
64B76	FIFTYFOUR	→ #36
64BA8	FIFTYNINE	→ #3B
64B58	FIFTYONE	→ #33
64B94	FIFTYSEVEN	→ #39
64B8A	FIFTYSIX	→ #38
64B6C	FIFTYTHREE	→ #35
64B62	FIFTYTWO	→ #34
642E3	FIVE	→ #5
64C5C	FIVEFOUR	→ #54
64C70	FIVESIX	→ #56
64C52	FIVETHREE	→ #53
0417F	FORTY	→ #28
64B3A	FORTYEIGHT	→ #30
64B1C	FORTYFIVE	→ #2D
64B44	FORTYNINE	→ #31
04189	FORTYONE	→ #29
64B30	FORTYSEVEN	→ #2F
64B26	FORTYSIX	→ #2E
0419D	FORTYTHREE	→ #2B
04193	FORTYTWO	→ #2A
642E3	FOUR	→ #4
64C0C	FOURFIVE	→ #45
0407B	FOURTEEN	→ #E
64BF8	FOURTHREE	→ #43
64BEE	FOUR TWO	→ #42
0417F	FOURTY	→ #28
0402B	id	→ #6

0402B	idnt	→ #6
64C98	IDREAL	→ #61
00305	infreserr	→ #305
64F04	INTEGER337	→ #
00A03	intrptderr	→ #a03
04021	list	→ #5
64C48	LISTCMP	→ #52
64C7A	LISTLAM	→ #57
64C3E	LISTREAL	→ #51
6509E	MINUSONE	→ #FFFF
04049	NINE	→ #9
040AD	NINETEEN	→ #13
00303	ofloerr	→ #303
073DB	ONE	→ #1
64CAC	ONEHUNDRED	→ #64
63AC4	ONEONE	→ #1 #1
03FF9	real	→ #1
0411B	REALEXT	→ #1E
0408F	REALOB	→ #10
64E32	REALOBOB	→ #100
04099	REALREAL	→ #11
040F3	REALSYM	→ #1A
64E14	ROMPANY	→ #F0
0403F	seco	→ #8
6427A	SEVEN	→ #7
04099	SEVENTEEN	→ #11
64C16	SEVENTY	→ #46
64C20	SEVENTYFOUR	→ #4A
64C2A	SEVENTYNINE	→ #4F
0402B	SIX	→ #6
0408F	SIXTEEN	→ #10
64BB2	SIXTY	→ #3C
64C02	SIXTYEIGHT	→ #44
64BDA	SIXTYFOUR	→ #40
64BBC	SIXTYONE	→ #3D
64BD0	SIXTYTHREE	→ #3F
64BC6	SIXTYTWO	→ #3E
0400D	str	→ #3
04053	sym	→ #A
04049	symp	→ #9

64D56	SYBUNIT	→ #9E
64DBA	SYMEXT	→ #AE
64D92	SYMD	→ #A6
64D9C	SYMLAM	→ #A7
64D6A	SYMOB	→ #A0
64D74	SYMREAL	→ #A1
64DB0	SYMSYM	→ #AA
64DEC	TAGGEDANY	→ #D0
04053	TEN	→ #A
04071	THIRTEEN	→ #D
0411B	THIRTY	→ #1E
0416B	THIRTYEIGHT	→ #26
0414D	THIRTYFIVE	→ #23
04143	THIRTYFOUR	→ #22
04175	THIRTYNINE	→ #27
04125	THIRTYONE	→ #1F
04161	THIRTYSEVEN	→ #25
04157	THIRTSIX	→ #24
04139	THIRTYTHREE	→ #21
0412F	THIRTYTWO	→ #20
642D1	THREE	→ #3
04067	TWELVE	→ #C
040B7	TWENTY	→ #14
04107	TWENTYEIGHT	→ #1C
040E9	TWENTYFIVE	→ #19
040DF	TWENTYFOUR	→ #18
04111	TWENTYNINE	→ #1D
040C1	TWENTYONE	→ #15
040FD	TWENTYSEVEN	→ #1B
040F3	TWENTYSIX	→ #1A
040D5	TWENTYTHREE	→ #17
040CB	TWENTYTWO	→ #16
642B7	TWO	→ #2
64D24	XHI	→ #83
64D1A	XHI-1	→ #82
64209	ZERO	→ #0
641FC	ZEROZERO	→ #0 #0
6431D	ZEROZEROONE	→ #0 #0 #1
64331	ZEROZEROTWO	→ #0 #0 #2
64309	ZEROZEROZERO	→ #0 #0 #0

b) Binary Integer Manipulation

◆ Arithmetic Functions

Addr	Word	Stack diagram
03EC2	#*	#2 #1 → #2*#1
03DBC	#+	#2 #1 → #2+#1
63808	#+-1	#2 #1 → #2+#1-1
03DE0	#-	#2 #1 → #2-#1
624FB	#-#2/	#2 #1 → (#2-#1)/2
637CC	#-+1	#2 #1 → (#2-#1)+1
03EF7	#/	#2 #1 → #remainder #quotient
03DEF	#1+	# → #+1
639CA	#1+'	# → #+1 (and quotes next runstream object)
62809	#1+DUP	# → #+1 #+1
03E0E	#1-	# → #-1
6264E	#10*	# → #*10
625DA	#10+	# → #+10
625EA	#12+	# → #+12
03E6F	#2*	# → #*2

03E2D	#2+	# → #+2
03E4E	#2-	# → #-2
03E8E	#2/	# → FLOOR(#/2)
6256A	#3+	# → #+3
625FA	#3-	# → #-3
6257A	#4+	# → #+4
6260A	#4-	# → #-4
6258A	#5+	# → #+5
6261A	#5-	# → #-5
62691	#6+	# → #*6
6259A	#6+	# → #+6
625AA	#7+	# → #+7
62674	#8*	# → #*8
625BA	#8+	# → #+8
625CA	#9+	# → #+9
624C6	#MAX	#2 #1 → MAX(#2,#1)
624BA	#MIN	#2 #1 → MIN(#2,#1)
63704	2DUP#+	#2 #1 → #2 #1 #1+#2

637F4	DROP#1-	# ob → #-1
628EB	DUP#1+	# → ##+1
6292F	DUP#1-	# → ##-1
63704	DUP3PICK#+	#2 #1 → #2 #1 #1+#2
6372C	OVER#+	#2 #1 → #2 #1+#2
6377C	OVER#-	#2 #1 → #2 #1-#2
63718	ROT#+	#2 ob #1 → ob #1+#2
62DCC	ROT#+SWAP	#2 ob #1 → #1+#2 ob
63768	ROT#-	#2 ob #1 → ob #1-#2
637B8	ROT#1+	# ob ob' → ob ob' #+1

62DCC	ROT+SWAP	#2 ob #1 → #1+#2 ob
62794	SWAP#-	#2 #1 → #1-#2
62904	SWAP#1+	# ob → ob #+1
51843	SWAP#1+SWAP	# ob → #+1 ob
637E0	SWAP#1-	# ob → ob #-1
51857	SWAP#1-SWAP	# ob → #-1 ob
637A4	SWAPOVER#-	#2 #1 → #1 #2-#1

◆ Conversion Functions

Addr	Word	Stack diagram	Explanation
18CEA	COERCE	% → #	If %<0 then # is 0, If %>FFFF then #=FFFF
194F7	COERCE2	%2 %1 → #2 #1	If %<0 then # is 0, If %>FFFF then #=FFFF
62CE1	COERCEDUP	% → ##	If %<0 then # is 0, If %>FFFF then #=FFFF
62E7B	COERCESWAP	ob % → # ob	If %<0 then # is 0, If %>FFFF then #=FFFF
18DBF	UNCOERCE	# → %	
63B96	UNCOERCE%%	# → %%	
1950B	UNCOERCE2	#2 #1 → %2 %1	

2.8.3 Character Constants

Addr	Word	Stack diagram	Explanation
05A51	CHR>#	chr → #	
05A75	#>CHR	# → chr	
6475C	CHR>\$	chr → \$	

Addr	Word
65433	CHR #
6544F	CHR -
6546B	CHR 1
65487	CHR 5
654A3	CHR 9
654BF	CHR =
654DB	CHR C
654F7	CHR G
65513	CHR K
6552F	CHR O
6554B	CHR S
65567	CHR W
65583	CHR a
6559F	CHR e
655BB	CHR i
655D7	CHR m
655F3	CHR q
6560F	CHR u
6562B	CHR y
6542C	CHR Db1Quote
6564E	CHR Angle
6566A	CHR Newline
65686	CHR Space
656A2	CHR {

656BE	CHR <>
6543A	CHR *
65456	CHR .
65472	CHR 2
6548E	CHR 6
654AA	CHR :
654C6	CHR >
654E2	CHR D
654FE	CHR H
6551A	CHR L
65536	CHR P
65552	CHR T
6556E	CHR X
6558A	CHR b
655A6	CHR f
655C2	CHR j
655DE	CHR n
655FA	CHR r
65616	CHR v
65632	CHR z
65639	CHR ->
65655	CHR Deriv
65671	CHR Pi
6568D	CHR UndScore
656A9	CHR }

65441	CHR +
6545D	CHR /
65479	CHR 3
65495	CHR 7
654B1	CHR ;
654CD	CHR A
654E9	CHR E
65505	CHR I
65521	CHR M
6553D	CHR Q
65559	CHR U
65575	CHR Y
65591	CHR c
655AD	CHR g
655C9	CHR k
655E5	CHR o
65601	CHR s
6561D	CHR w
6541E	CHR 00 (hex0)
65640	CHR <<
6565C	CHR Integral
65678	CHR RightPar
65694	CHR [
656B0	CHR <=

65448	CHR ,
65464	CHR 0
65480	CHR 4
6549C	CHR 8
654B8	CHR <
654D4	CHR B
654F0	CHR F
6550C	CHR J
65528	CHR N
65544	CHR R
65560	CHR V
6557C	CHR Z
65598	CHR d
655B4	CHR h
655D0	CHR l
655EC	CHR p
65608	CHR t
65624	CHR x
65640	CHR ...
65647	CHR >>
65663	CHR LeftPar
6567F	CHR Sigma
6569B	CHR]
656B7	CHR >=

Addr	Word	Stack diagram	Explanation
656C5	\$ R<<		\$ "R/80/80" "R<angle><angle>"
656D5	\$ R<Z		\$ "R/80Z" "R<angle>Z"
656E5	\$ XYZ		\$ "XYZ"
656F5	\$ <<>>		\$ "ABBB"
65703	\$ { }		\$ "{}"
65711	\$ []		\$ "[]"
6571F	\$ ' '		\$ ""
6572D	\$::		\$ "::"

6573B	\$ IRParens		\$ "("
65749	\$ 2DQ		\$ ""
65757	\$ ECHO		\$ "ECHO"
65769	\$ EXIT		\$ "EXIT"
6577B	\$ Undefined		\$ "Undefined"
65797	\$ RAD		\$ "RAD"
657A7	\$ GRAD		\$ "GRAD"
65238	NEWLINE\$		\$ "\0a"
65254	SPACE\$		\$ " "

2.8.4 Hex & Character Strings

a) Character Strings

Addr	Word	Stack diagram	Explanation
623A0	!append\$	\$1 \$2 → \$3	Same as &\$, except that it will attempt the concatenation "in place," if there is not enough memory for the new string, and the target is in tempob.
05B15	\$>ID	\$name → Id	Converts string object to name object
05193	&\$	\$1 \$2 → \$3	Appends \$2 to \$1
63F6A	&\$SWAP	ob \$1 \$2 → \$3 ob	Appends \$2 to \$1, then swaps result with ob
0525B	>H\$	\$ chr → '\$	Prepends chr to \$
052EE	>T\$	\$ chr → '\$	Appends chr to \$
63259	1_#1-SUB\$	\$ # → '\$	Where '\$ = chars 1 thru #-1 of \$
162B8	a%>\$	% → \$	Converts % to \$ using current display mode
162AC	a%>,\$	% → \$	Converts % to \$ using current display mode. Same as a%>\$, but with no commas
18873	AND\$	\$1 \$2 → \$1 AND \$2	Bitwise logical AND of two strings
62BB0	APPEND_SPACE	\$ → '\$	Appends space to \$
45676	Blank\$	# → \$	Creates a string of # spaces
050ED	CAR\$	\$ → chr \$	Returns 1st chr of \$ or NULL\$ if \$ is null
0516C	CDR\$	\$ → '\$	'\$ is \$ minus first character. Returns NULL\$ if \$ is null
12770	COERCE\$22	\$ → '\$	If \$ longer than 22 chars., truncates to 21 chars & appends "..."
15B13	DECOMP\$	ob → \$	Decompiles object for stack display
14088	DO>STR	ob → \$	Internal version of STR
04D3E	DROPNULL\$	ob → NULL\$	Drops object, returns zero-length string
63295	DUP\$>ID	\$name → \$name Id	Dups, converts string object to name object
627BB	DUPLEN\$	\$ → \$ #length	Returns \$ and its length
63209	DUPNULL\$?	\$ → \$ flag	Returns TRUE if \$ is zero-length
15A0E	EDITDECOMP\$	ob → \$	Decompile object for editing
05BB9	ID>\$	ID → \$name	Converts name object to a string
04D87	JstGETTHEMSG	# → \$	Fetches message from message table
63281	LAST\$	\$ # → '\$	Returns last # chrs of \$
627BB	LEN\$	\$ → #length	Returns length of \$
63191	NEWLINE&\$	\$ → '\$	Appends "\0a" to \$
04D3E	NULL\$	→ \$	Returns empty string
0556F	NULL\$?	\$ → flag	Returns TRUE if \$ is zero-length
62D59	NULL\$SWAP	ob → \$ ob	Swaps empty string into level 2
1613F	NULL\$TEMP	→ \$	Creates empty string in TEMPOB
18887	OR\$	\$1 \$2 → \$3	Bitwise logical OR of two strings
05622	OVERLEN\$	\$ ob → \$ ob #length	Returns length of \$ in level 2
238A4	palparse	\$ → ob TRUE \$ → \$ #pos \$' FALSE	Parse a string into an object and TRUE, or returns position of error and FALSE
645B1	POS\$	\$search \$find #start → #pos	Returns #pos (#0 if not found) of \$find within \$search starting at head of \$search
645BD	POS\$REV	\$search \$find #start → #pos	Returns #pos (#0 if not found) of \$find within \$search starting at tail of \$search
49709	PromptIdUtil	id ob → \$	Returns string in the form " id: ob "
127A7	SEP\$NL	\$ → \$2 \$1	Separate \$ at newline character
63245	SUB\$	\$ #start #end → '\$	Returns substring of \$
30805	SUB\$1#	\$ #pos → #	Returns bint with value of character in \$ at position #pos
62D6D	SUB\$SWAP	ob \$ #start #end → '\$ ob	Returns substring of \$ and swaps with ob
622EF	SWAP&\$	\$1 \$2 → "\$2\$1"	Appends \$1 to \$2

0D304	TIMESTR	%date %time → "WED 03/30/90 11:30:15A"	Returns string time and date (equivalent to the user word TSTR)
1889B	XOR\$	\$1 \$2 → \$3	Bitwise logical XOR of two strings

b) Hex Strings

Addr	Word	Stack diagram	Explanation
5435D	#>%	hxs → %	Converts hxs to real
543F9	%>#	% → hxs	Converts real to hxs
0518A	&HXS	hxs1 hxs2 → hxs3	Appends hxs2 to hxs1
51532	2HXSLIST?	{ hxs1 hxs2 } → #1 #2	Converts list of two hxs into two bins. Generates Bad Argument Value error for invalid input
544EC	HXS#HXS	hxs1 hxs2 → %flag	Returns %1 if hxs1 <> hxs2, otherwise %0
05A03	HXS>#	hxs → #	Converts lower 20 bits of hxs into a bint
54061	HXS>\$	hxs → \$	Does hxs>\$, then appends base character
5435D	HXS>%	hxs → %	Converts hex string to real number
54552	HXS<HXS	hxs1 hxs2 → %flag	Returns %1 if hxs1<hxs2, otherwise %0
54500	HXS>HXS	hxs1 hxs2 → %flag	Returns %1 if hxs1>hxs2, otherwise %0
5452C	HXS>=HXS	hxs1 hxs2 → %flag	Returns %1 if hxs1>=hxs2, otherwise %0
5453F	HXS<=HXS	hxs1 hxs2 → %flag	Returns %1 if hxs1<=hxs2, otherwise %0
05616	LENHXS	hxs → #length	Returns # of nibbles in hxs
055D5	NULHXS	→ hxs	Returns zero-length hex string
05815	SUBHXS	hxs #m #n → hxs'	Returns substring

User RPL binary integers are actually hex strings. The following words assume 64-bit or shorter hex strings, and return results according to the current wordsize:

Addr	Word	Stack diagram	Explanation
53F05	bit/	hxs1 hxs2 → hxs3	Divides hxs1 by hxs2
5429F	bit%#	% hxs → hxs'	Divides % by hxs, returns hxs
542BD	bit%#	hxs % → hxs'	Divides hxs by %, returns hxs
53ED3	bit*	hxs1 hxs2 → hxs3	Multiplies hxs1 by hxs2
542D1	bit%#*	% hxs → hxs'	Multiplies % by hxs, returns hxs
542EA	bit%#*	hxs % → hxs'	Multiplies hxs by %, returns hxs
53EA0	bit+	hxs1 hxs2 → hxs3	Adds hxs1 to hxs2
54330	bit%#+	% hxs → hxs'	Adds % to hxs, returns hxs
54349	bit%#+	hxs % → hxs'	Adds hxs to %, returns hxs
53EB0	bit-	hxs1 hxs2 → hxs3	Subtracts hxs2 from hxs1
542FE	bit%#-	% hxs → hxs'	Subtracts % from hxs, returns hxs
5431C	bit%#-	hxs % → hxs'	Subtracts hxs from %, returns hxs
53D04	bitAND	hxs1 hxs2 → hxs3	Bitwise logical AND
53E65	bitASR	hxs → hxs'	Arithmetic shift right one bit
53D15	bitOR	hxs1 hxs2 → hxs3	Bitwise logical OR
53D4E	bitNOT	hxs1 hxs2 → hxs3	Bitwise logical NOT
53E0C	bitRL	hxs → hxs'	Circular left shift by 1 bit
53E3B	bitRLB	hxs → hxs'	Circular left shift by 1 byte
53DA4	bitRR	hxs → hxs'	Circular right shift by 1 bit
53DE1	bitRRB	hxs → hxs'	Circular right shift by 1 byte
53D5E	bitSL	hxs → hxs'	Shift left by 1 bit
53D6E	bitSLB	hxs → hxs'	Shift left by 1 byte
53D81	bitSR	hxs → hxs'	Shift right by 1 bit
53D91	bitSRB	hxs → hxs'	Shift right by 1 byte
53D26	bitXOR	hxs1 hxs2 → hxs3	Bitwise logical XOR

Wordsize control:

Addr	Word	Stack diagram	Explanation
54039	WORDSIZE	→ #	Returns user binary integer wordsize
53CAA	dostws	# →	Stores binary wordsize
540BB	hxs>\$	hxs → \$	Converts hex string to chr string using the current display mode and wordsize

2.8.5 Real Numbers

a) Built-in Reals

Addr	Word
2A562	%%.1
2B3DD	%%.4
2A57C	%%.5
2A4C6	%%0
2A4E0	%%1
2A596	%%10
2B2DC	%%12
2A4FA	%%2
0F688	%%2PI
2A514	%%3
2A52E	%%4
2A548	%%5
2B300	%%60
2B1FF	%%7

2A562	%.1
2A57C	%.5
2A386	%-1
2A4C6	%0
2A4E0	%1
2A596	%10
415F1	%100
1CC03	%11
2B2DC	%12
1CC37	%13
1CC51	%14
1CC85	%15
1CD3A	%16
1CD54	%17
650FC	%180

2A39B	%-2
1CC6B	%20
1CCA4	%21
1CCC3	%22
1CCE2	%23
1CD01	%24
1CD20	%25
1CD73	%26
1CD8D	%27
2A3B0	%-3
65126	%360
2A3C5	%-4
2A3DA	%-5
2A3EF	%-6
2A404	%-7

2A419	%-8
2A42E	%-9
2A487	%-MAXREAL
2A4B1	%-MINREAL
2A4FA	%2
2A514	%3
2A52E	%4
2A548	%5
2B300	%6
2B1FF	%7
2A35C	%8
650A8	%e
2A472	%MAXREAL
2A49C	%MINREAL
2A443	%PI

b) Real Number Functions

Addr	Word	Stack diagram	Explanation
2A99A	%%*	%%1 %2 → %3	Multiplies two extended reals
62FED	%%*ROT	ob1 ob2 %%1 %2 → ob2 %3 ob1	Multiplies two extended reals, then does a ROT
62EA3	%%*SWAP	ob %%1 %2 → %3 ob	Multiplies two extended reals, then does a SWAP
63C18	%%*UNROT	ob1 ob2 %%1 %2 → %3 ob1 ob2	Multiplies two extended reals, then does an UNROT
2A943	%%+	%%1 %2 → %3	Adds two extended reals
2A94F	%%-	%%1 %2 → %3	Subtraction
2A8F0	%%ABS	%' → %'	Absolute value
2AD08	%%ACOSRAD	%' → %'	Arc-cosine using radians
2AD4F	%%ANGLE	%'x %y → %angle	Angle using current angle mode from %'x %y
2AD6C	%%ANGLEDEG	%'x %y → %angle	Angle using degrees from %'x %y
2AD7C	%%ANGLERAD	%'x %y → %angle	Angle using radians from %'x %y
2ACD8	%%ASINRAD	%' → %'	Arc-sine using radians
2A910	%%CHS	%' → %'	Change sign
2AC57	%%COS	%' → %'	Cosine
2AC68	%%COSDEG	%' → %'	Cosine using degrees
2ADC7	%%COSH	%' → %'	Hyperbolic cosine
2AC78	%%COSRAD	%' → %'	Cosine using radians
2AB1C	%%EXP	%' → %'	e^x
2AF99	%%FLOOR	%' → %'	Greatest integer <= x
2AF27	%%H>HMS	%' → %'	Decimal hours to hh.mmss
2AF99	%%INT	%' → %'	Integer part
2AB5B	%%LN	%' → %'	ln(x)
2AB94	%%LNPI	%' → %'	ln(x+1)
2A6DC	%%MAX	%%1 %2 → %3	Returns greater of two %s
2B4C5	%%P>R	%'radius %angle → %'x %y	Polar to rectangular conversion
2B498	%%R>P	%'x %y → %'radius %angle	Rectangular to polar conversion
2AC06	%%SIN	%' → %'	Sine
2AC17	%%SINDEG	%' → %'	Sine using degrees
2AD95	%%SINH	%' → %'	Hyperbolic sine
2AAEA	%%SQRT	%' → %'	Square root
2ACA8	%%TANRAD	%' → %'	Tangent using radians
2AA5F	%%^	%%1 %2 → %3	Exponential
2A943	%%+	%%1 %2 → %3	Addition
51BE4	%%+SWAP	ob %1 %2 → %3 ob	Addition, then SWAP
2A94F	%%-	%%1 %2 → %3	Subtraction
50262	%%1+	% → %+1	Adds one
50276	%%1-	% → %-1	Subtracts one
543F9	%%>#	% → hxs	Converts real to binary integer

2A5C1	%>%%	% → %%	Converts real to extended real
2A95B	%>%-	%1 %2 → %%3	Converts 2 % to %%, then subtracts
2A99E	%>%1	%x → %%	Converts % to %%, then does 1/x
2AD5B	%>%ANGLE	%x %y → %%angle	Angle in current angle mode
2AAF6	%>%SQRT	% → %%	Converts % to %%, then sqrt(x)
62E8F	%>%SWAP	ob % → %% ob	Converts % to %%, then SWAP
51A07	%>C%	%real %imag → C%	Real to complex conversion
2A673	%>HMS	% → %hh.mmss	Decimal hours to hh.mmss
2A8F0	%ABS	% → %'	Absolute value
18CD7	%ABSCOERCE	% → #	Absolute value, convert to bint
2AD08	%ACOS	% → %'	Arc cosine
2AE13	%ACOSH	% → %'	Hyperbolic arc cosine
2ABBA	%ALOG	% → %'	10^x
2AD4F	%ANGLE	%x %y → %angle	Angle using current angle mode from %x %y
2ACD8	%ASIN	% → %'	Arc sine
2AE00	%ASINH	% → %'	Hyperbolic arc sine
2AD21	%ATAN	% → %'	Arc tangent
2AE26	%ATANH	% → %'	Hyperbolic arc tangent
2AF73	%CEIL	% → %'	Next greatest integer
2A910	%CH	%1 %2 → %3	Percent change
2A910	%CHS	% → %'	Change sign
2AE62	%COMB	%m %n → %COMB(m,n)	Combinations of m items taken n at a time
2AC57	%COS	% → %'	Cosine
2ADC7	%COSH	% → %'	Hyperbolic cosine
2A622	%D>R	% → %'	Degrees to radians
2AB1C	%EXP	% → %'	e^x
2AB42	%EXPM1	% → %'	e^x-1
2AE39	%EXPONENT	% → %'	Returns exponent
2B0C4	%FACT	% → %!	Factorial
2AF99	%FLOOR	% → %'	Greatest integer <= x
2AF4D	%FP	% → %'	Fractional part
2A6A0	%HMS+	%1 %2 → %3	HH.MMSS addition
2A6C8	%HMS-	%1 %2 → %3	HH.MMSS subtraction
2A68C	%HMS>	% → %'	Convert hh.mmss to decimal hours
2AF60	%IP	% → %'	Integer part
2EC11	%IP>#	% → #	IP(ABS(x) converted to binary integer
2AB5B	%LN	% → %'	ln(x)
2AB94	%LNp1	% → %'	ln(x+1)
2AB81	%LOG	% → %'	Common log
2A930	%MANTISSA	% → %'	Returns mantissa
2A6DC	%MAX	%1 %2 → %	Returns larger of two reals
62D81	%MAXorder	%1 %2 → %larger %smaller	Orders two numbers
2A70E	%MIN	%1 %2 → %	Returns smaller of two reals
2ABDC	%MOD	%1 %2 → %3	Returns %1 MOD %2
2AE4C	%NFACT	% → %'	Factorial
2AA81	%NROOT	%1 %2 → %3	Nth root
2A9C9	%OF	%1 %2 → %3	Returns percentage of %1 that is %2
2AE75	%PERM	%m %n → %PERM(%m,%n)	Returns permutations of %m items taken %n at a time
2B4BB	%POL>%REC	%x %y → %radius %angle	Rectangular to polar conversion
2A655	%R>D	%radians → %degrees	Radians to degrees
2AFC2	%RAN	→ %random	Random number
2B044	%RANDOMIZE	%seed →	Updates random number seed, uses the system clock if %=0
2B48E	%REC>%POL	%radius %angle → %x %y	Polar to rectangular conversion
2A8D7	%SGN	% → %'	Sign: -1, 0 or 1 returned depending on the sign of the argument
2AC06	%SIN	% → %'	Sine
2AD95	%SINH	% → %'	Hyperbolic sine
2B4F2	%SPH>%REC	%r %th %ph → %x %y %z	Spherical to rectangular conversion
2AAEA	%SQRT	% → %'	Square root
2ACA8	%T	%1 %2 → %3	Percent total
2ACA8	%TAN	% → %'	Tangent
2ADED	%TANH	% → %'	Hyperbolic tangent
2AA5F	%^	%1 %2 → %3	Exponential

2B470	2%>%	%%1 %%2 → %1 %%2	Extended real to real conversion
2B45C	2%>%%	%1 %%2 → %%1 %%%2	Real to extended real conversion
05D2C	C%>%	C% → %real %imag	Complex to real conversion
0CC39	DDAYS	%date1 %date2 → %diff	Days between dates in DMY format
2B07B	DORANDOMIZE	% →	Updates random number seed
2B529	RNDXY	%number %places → %number'	Rounds %number to %places
2B53D	TRCXY	%number %places → %number'	Truncates %number to %places
632A9	SWAP%>C%	%imag %real → C%	Real to complex conversion

2.8.6 Complex Numbers

a) Built-in Complex Numbers

Addr	Word	Stack diagram
524AF	C%0	→ (0,0)
524F7	C%1	→ (1,0)
5196A	C%-1	→ (-1,0)
5193B	C%%1	→ (%%1,%%0)

b) Conversion Words

Addr	Word	Stack diagram
51A07	%>C%	%real %imag → C%
?	%%>C%%	%%real %%imag → C%%
51A07	%%>C%	%real %imag → C%
05D2C	C%>%	C% → %real %imag
05DBC	C%%>%%	C%% → %%real %%imag
519F8	C%%>C%	C%% → C%

519CB	C%>%%	C% → %%real %imag
519DF	C%>%%SWAP	C% → %%imag %real
519B7	C>Im%	C% → %imag
519A3	C>Re%	C% → %real

c) Complex Functions

Addr	Word	Stack diagram	Explanation
51EFA	C%1/	C% → C%'	Inverse
52062	C%ABS	C% → %	Returns SQRT(x ² +y ²) from (x,y)
52863	C%ACOS	C% → C%'	Arc cosine
52305	C%ALOG	C% → C%'	Common antilog
52099	C%ARG	C% →	Returns ANGLE(x,y) from (x,y)
52804	C%ASIN	C% → C%'	Arc sine
52675	C%ATAN	C% → C%'	Arc tangent
52374	C%^C	C%1 C%2 → C%3	Power
51B70	C%CHS	C% → C%'	Change sign
51B91	C%%CHS	C%% → C%%'	Change sign
51BB2	C%CONJ	C% → C%'	Conjugate
51BC1	C%%CONJ	C%% → C%%'	Conjugate
52571	C%COS	C% → C%'	Cosine
52648	C%COSH	C% → C%'	Hyperbolic cosine
52193	C%EXP	C% → C%'	e ^z
521E3	C%LN	C% → C%'	Natural logarithm
522BF	C%LOG	C% → C%'	Common logarithm
520CB	C%SGN	C% → C%'	Returns (x/SQRT(x ² +y ²),y/SQRT(x ² +y ²))
52530	C%SIN	C% → C%'	Sine
5262F	C%SINH	C% → C%'	Hyperbolic sine
52107	C%SQRT	C% → C%'	Square root
525B7	C%TAN	C% → C%'	Tangent
5265C	C%TANH	C% → C%'	Hyperbolic tangent

2.8.7 Arrays

Addr	Word	Stack diagram	Explanation
03562	ARSIZE	[array] → #elements [array] → {dims}	
0371D	GETATELN	# [array] → ob TRUE # [array] → FALSE	FALSE if no such element.
03442	MAKEARRY	{dims} ob → [array]	Creates an unlinked array having the same element type as ob.

35CAE	MATCON	[arry%] % → [arry%] [arryC%] C% → [arryC%]	All elements are initialized to ob. Sets all elements in array to % or C%.
37E0F	MATREDIM	[array] {dims} → [array]	
3811F	MATTRN	[array] → [array]	
357A8	MDIMS	[1-D array] → #m FALSE [2-D array] → #m #n TRUE	
62F9D	MDIMSDROP	[2-D array] → #m #n	Don't use MDIMSDROP on a vector!
63141	OVERARSIZE	[array] ob → [array] ob #elements	
355B8	PULLREALEL	[arry%] # → [arry%] %	
355C8	PULLCMPEL	[arryC%] # → [arryC%] C%	
35628	PUTEL	[arry%] % # → [arry%] [arryC%] C% # → [arryC%]	
3566F	PUTREALEL	[arry%] % # → [arry%]	
356F3	PUTCMPEL	[arryC%] C% # → [arryC%]	

2.8.8 Composite Objects

In the notation below, the term "comp" refers to either any composite object. The term "#n" refers to the number of objects in a composite object, and the term "#i" refers to the index of an object within a composite.

Addr	Word	Stack diagram	Explanation
0521F	&COMP	comp comp' → comp"	comp is concatenated to comp'
63FFB	2Ob>Seco	ob1 ob2 → :: ob1 ob2 ;	
05445	::N	obn ... ob1 #n → :: obn ... ob1 ;	
632D1	::NEVAL	obn ... ob1 #n → ?	Does ::N, then evaluates secondary
052FA	>TCOMP	comp ob → comp'	ob is added to the tail of comp
6317D	CARCOMP	comp → ob comp → comp	Returns first object in the core of the composite. Returns an null comp if the supplied composite is null.
05153	CDRCOMP	comp → comp' comp → comp	Returns the core of the composite minus the first object. Returns null comp if the supplied composite is null.
631E1	DUPINCOMP	comp → comp obn ... ob1 #n	
63231	DUPLENCOMP	comp → comp #n	
6321D	DUPNULLCOMP?	comp → comp flag	TRUE if comp is null.
63A6F	DUPNULL{ } ?	{list} → {list} flag	TRUE if {list} is null.
644A3	EQUALPOSCOMP	comp ob → #pos #0	Returns the index of the first object in comp matching (EQUAL) ob (see NTHOF also)
64127	Embedded?	ob1 ob2 → flag	Returns TRUE if ob2 is embedded in, or the same as, ob1; otherwise returns FALSE.
62B88	INCOMPDROP	comp → obn ... ob1	
054AF	INNERCOMP	comp → obn ... ob1 #n	
62C41	INNERDUP	comp → obn ... ob1 #n #n	
0567B	LENCOMP	comp → #n	
6480B	NEXTCOMPOB	comp #offset → comp #offset' ob TRUE comp #offset → comp FALSE	#offset is the nibble offset from the start of the list to the Nth object in the list. Returns a new #offset and the next object if the next object is not SEMI, otherwise returns the list and FALSE. Use #5 at the start of the list.
62D1D	NTHCOMDDUP	comp #i → ob ob	
62B9C	NTHCOMPDROP	comp #i → ob	
056B6	NTHELCOMP	comp #i → ob TRUE comp #i → FALSE	Returns FALSE if #i is out of range
644BC	NTHOF	ob comp → #i #0	Same as SWAP EQUALPOSCOMP.
055FD	NULL::	→ ::;	(Returns null secondary)
63A6F	NULL{ }	→ { }	(Returns null list)
23EED	ONE{ }N	ob → { ob }	
63FFB	Ob>Seco	ob → :: ob ;	
6448A	POSCOMP	comp ob pred → #i #0	If the specified object "matches" an element of the specified composite, where "match" is defined as the specified predicate returning TRUE when applied to an element of the comp and the object, then POSCOMP returns the left-to-right index of the element within the composite, or zero. For instance, to find the first real less than 5 in a list of reals: :: {list} 5 ' %< POSCOMP ;

1DC00	FUTLIST	ob #i {list} → {list}'	Assumes 0<#i<=#n
05821	SUBCOMP	comp #m #n → comp' (Returns subcomposite)	IF #m > #n THEN comp' is null IF #m=0 THEN #m is set to 1 IF #n=0 THEN #n is set to 1 IF #m > LEN(comp) THEN comp' is null IF #n > LEN(comp) THEN #n is set to LEN(comp)
631F5	SWAPINCOMP	comp obj → obj obn ... ob1 #n	
631CD	THREE{ }N	ob1 ob2 ob3 → { ob1 ob2 ob3 }	
631B9	TWO{ }N	ob1 ob2 → { ob1 ob2 }	
631A5	{ }N	obn ... ob1 #n → {list}	
35491	apndvarlst	{list} ob → {list}'	Adds ob to the list if ob is not found within the list
643EF	matchob?	ob comp → ob TRUE ob comp → FALSE	Determines if ob is equal (EQUAL) to any element of comp

2.8.9 Tagged Objects

Addr	Word	Stack diagram	Explanation
22618	%>TAG	ob % → tagged	Tags ob with %
22618	>TAG	ob \$ → tagged	Tags ob with \$
05F2E	ID>TAG	ob id/lam → tagged	Tags ob with id
64775	STRIPTAGS	tagged → ob	Removes all tags
647A2	STRIPTAGS12	tagged ob' → ob ob'	Strips tags from level 2 object
647BB	TAGOBS	ob \$ → tagged ob1 ... obn { \$1 ... \$n } → tagged1 ... taggedn	Tags one object, or several objects if a list of tags is in level 1
225F5	USER\$>TAG	ob \$ → tagged	Tags ob with \$ (up to 255 chrs valid)

2.8.10 Unit Objects

Addr	Word	Stack diagram	Explanation
0FE44	U>NCQ	unit → n% cf% qhxs	Returns number, conversion factor, and hex quantity string
0F584	UM=?	unit1 unit2 → %flag	Returns %1 if two unit obs are equal
0F598	UM#?	unit1 unit2 → %flag	Returns %1 if unit1 <> unit2
0F5AC	UM<?	unit1 unit2 → %flag	Returns %1 if unit1 < unit2
0F5C0	UM>?	unit1 unit2 → %flag	Returns %1 if unit1 > unit2
0F5D4	UM<=?	unit1 unit2 → %flag	Returns %1 if unit1 <= unit2
0F5E8	UM>=?	unit1 unit2 → %flag	Returns %1 if unit1 >= unit2
0F33A	UM>U	% unit → unit'	Replaces the number part of a unit object
0FBAB	UM%	unit %percentage → unit'	Returns a percentage of a unit object
0FC3C	UM%CH	unit1 unit2 → %	Returns percent difference
0FCCD	UM%T	unit1 unit2 → %	Returns percentage fraction
0F6A2	UM+	unit1 unit2 → unit3	Addition
0F774	UM-	unit1 unit2 → unit3	Subtraction
F792	UM*	unit1 unit2 → unit3	Multiply
0F823	UM/	unit1 unit2 → unit3	Divide
10B72	UM^	unit1 unit2 → unit3	Power
?	UM1/	unit → unit'	Inverse
0F5FC	UMABS	unit → unit'	Absolute value
0F615	UMCHS	unit → unit'	Change sign
0F371	UMCONV	unit1 unit2 → unit1'	Converts unit1 to units of unit2
0F660	UMCOS	unit → unit'	Cosine
0FB6F	UMMAX	unit1 unit2 → unit?	Returns larger of unit1 and unit2
0FB8D	UMMIN	unit1 unit2 → unit?	Returns smaller of unit1 and unit2
0F945	UMSI	unit → unit'	Convert to SI base units
0F62E	UMSIN	unit → unit'	Sine
F913	UMSQ	unit → unit'	Square
F92C	UMSQRT	unit → unit'	Square root
0F674	UMTAN	unit → unit'	Tangent
0F34E	UMU>	unit → % unit'	Returns number and normalized unit parts of a unit object
0FBFA	UMXROOT	unit1 unit2 → unit3	unit1^1/unit2
0F218	UNIT>\$	unit → \$	Decompiles a unit object with tics

2.8.11 Temporary Variables and Temporary Environments

Addr	Word	Stack diagram	Explanation
62DB3	1ABNSWAP	ob → lamob ob	Does :: 1GETLAM ABND SWAP ;
634B6	1GETABND	→ lamob	Does :: 1GETLAM ABND ;
613B6	1GETLAM	→ ob	Returns contents of 1st lam
613E7	2GETLAM	→ ob	Returns contents of 2nd lam
6140E	3GETLAM	→ ob	Returns contents of 3rd lam
61438	4GETLAM	→ ob	Returns contents of 4th lam
6145C	5GETLAM	→ ob	Returns contents of 5th lam
6146C	6GETLAM	→ ob	Returns contents of 6th lam
6147C	7GETLAM	→ ob	Returns contents of 7th lam
6148C	8GETLAM	→ ob	Returns contents of 8th lam
6149C	9GETLAM	→ ob	Returns contents of 9th lam
614AC	10GETLAM	→ ob	Returns contents of 10th lam
614BC	11GETLAM	→ ob	Returns contents of 11th lam
614CC	12GETLAM	→ ob	Returns contents of 12th lam
614DC	13GETLAM	→ ob	Returns contents of 13th lam
614EC	14GETLAM	→ ob	Returns contents of 14th lam
614FC	15GETLAM	→ ob	Returns contents of 15th lam
6150C	16GETLAM	→ ob	Returns contents of 16th lam
6151C	17GETLAM	→ ob	Returns contents of 17th lam
6152C	18GETLAM	→ ob	Returns contents of 18th lam
6153C	19GETLAM	→ ob	Returns contents of 19th lam
6154C	20GETLAM	→ ob	Returns contents of 20th lam
6155C	21GETLAM	→ ob	Returns contents of 21st lam
6156C	22GETLAM	→ ob	Returns contents of 22nd lam
62F07	1GETSWAP	ob → lamob ob	Does :: 1GETLAM SWAP ;
634CF	1LAMBIND	ob →	Does :: 1NULLLAM() BIND ;
34D2B	1NULLLAM{ }	→ { NULLLAM }	Returns list with one null lam
615E0	1PUTLAM	ob →	Stores ob into 1st lam
615F0	2PUTLAM	ob →	Stores ob into 2nd lam
61600	3PUTLAM	ob →	Stores ob into 3rd lam
61615	4PUTLAM	ob →	Stores ob into 4th lam
61625	5PUTLAM	ob →	Stores ob into 5th lam
61635	6PUTLAM	ob →	Stores ob into 6th lam
61645	7PUTLAM	ob →	Stores ob into 7th lam
61655	8PUTLAM	ob →	Stores ob into 8th lam
61665	9PUTLAM	ob →	Stores ob into 9th lam
61675	10PUTLAM	ob →	Stores ob into 10th lam
61685	11PUTLAM	ob →	Stores ob into 11th lam
61695	12PUTLAM	ob →	Stores ob into 12th lam
616A5	13PUTLAM	ob →	Stores ob into 13th lam
616B5	14PUTLAM	ob →	Stores ob into 14th lam
616C5	15PUTLAM	ob →	Stores ob into 15th lam
616D5	16PUTLAM	ob →	Stores ob into 16th lam
616E5	17PUTLAM	ob →	Stores ob into 17th lam
616F5	18PUTLAM	ob →	Stores ob into 18th lam
61705	19PUTLAM	ob →	Stores ob into 19th lam
61715	20PUTLAM	ob →	Stores ob into 20th lam
61725	21PUTLAM	ob →	Stores ob into 21st lam
61735	22PUTLAM	ob →	Stores ob into 22nd lam
632E5	2GETEVAL	→ ?	Recalls & evaluates ob in 2nd lam
07943	@LAM	id → ob TRUE id → FALSE	Recalls lam by name, returns ob and TRUE if id exists; FALSE otherwise
07497	ABND	→	Abandons topmost temp var env.
074D0	BIND	ob ... { id ... } →	Creates new temp var env.
61CE9	CACHE	obn ... ob1 n lam →	Saves away n objects plus the count n in a temporary environment, each object being bound to the same identifier lam. The last pair has the count.)
61EA7	DUMP	NULLLAM → ob1..obn n	DUMP is essentially the inverse of CACHE, BUT: it ONLY works with NULLLAM as the cached name, and it ALWAYS does a garbage collect.

634CA	DUP1LAMBIND	ob → ob	Does DUP, then 1LAMBIND
61610	DUP4PUTLAM	ob → ob	Does DUP, then 4PUTLAM
61745	DUPTEMPENV	→	Duplicates topmost temporary env., clearing the protection word.
075A5	GETLAM	#n → ob	Returns object in #nth temp var
34D30	NULLLAM	→ NULLLAM	Null temporary variable name
075E9	PUTLAM	ob #n →	Stores ob in #nth temp var
18513	STO	ob id →	Stores ob in named global/temp var
07D1B	STOLAM	ob id →	Stores ob in named temp var

2.8.12 Checking Arguments

a) Number of Arguments

The following words verify that required number of arguments are on the stack, and issue the "Too Few Arguments" error otherwise.

Addr	Word	Stack diagram	Explanation
18A1E	CK0	→	No arguments required. Should be the 1st object of a command.
18A15	CK0NOLASTWD	→	No arguments required. Doesn't record the command.
18AA5	CK1	ob → ob	One argument required. Should be the 1st object of a command.
18AB2	CK1NOLASTWD	ob → ob	One argument required. Doesn't record the command.
18A80	CK2	ob2 ob1 → ob2 ob1	Two arguments required. Should be the 1st object of a command.
18A8D	CK2NOLASTWD	ob2 ob1 → ob2 ob1	Two arguments required. Doesn't record the command.
18A5B	CK3	ob3 ob2 ob1 → ob3 ob2 ob1	Three arguments required. Should be the 1st object of a command.
18A68	CK3NOLASTWD	ob3 ob2 ob1 → ob3 ob2 ob1	Three arguments required. Doesn't record the command.
18B92	CK4	ob4 ob3 ob2 ob1 → ob4 ob3 ob2 ob1	Four arguments required. Should be the 1st object of a command.
18B9F	CK4NOLASTWD	ob4 ob3 ob2 ob1 → ob4 ob3 ob2 ob1	Four arguments required. Doesn't record the command.
18B6D	CK5	ob5 ob4 ob3 ob2 ob1 → ob5 ob4 ob3 ob2 ob1	Five arguments required. Should be the 1st object of a command.
18B7A	CK5NOLASTWD	ob5 ob4 ob3 ob2 ob1 → ob5 ob4 ob3 ob2 ob1	Five arguments required. Doesn't record the command.
40BC9	AtUserStack	→	:: CK0NOLASTWD 0LASTOWDOB! ;
1592D	CK1NoBlame	ob → ob	:: 0LASTOWDOB! CK1NOLASTWD ;
18C34	CKN	obn ... ob1 #n → obn ... ob1	n arguments required. Should be the 1st object of a command.
18C4A	CKNNOLASTWD	obn ... ob1 #n → obn ... ob1	n arguments required. Doesn't record the command.

b) Dispatching on Argument Type

Addr	Word	Stack diagram	Explanation
18A1E	CK&DISPATCH0	... →	Dispatches upon the next pairs of objects (#type action).
18B7A	CK&DISPATCH1	... →	Dispatches upon the next pairs of objects (#type action). If no match found, does a second pass, stripping any tags from stack objects.
18ECE	CK1&Dispatch	ob →	One argument required, then dispatches upon the next pairs of objects (#type action).
18EDF	CK2&Dispatch	ob1 ob2 →	Two arguments required, then dispatches upon the next pairs of objects (#type action).
18EF0	CK3&Dispatch	ob1 ob2 ob3 →	Three arguments required, then dispatches upon the next pairs of objects (#type action).
18F01	CK4&Dispatch	ob1 ob2 ob3 ob4 →	Four arguments required, then dispatches upon the next pairs of objects (#type action).
18F12	CK5&Dispatch	ob1 ob2 ob3 ob4 ob5 →	Five arguments required, then dispatches upon the next pairs of objects (#type action).

```

CK&DISPATCH1
  #type1 action1
  #type2 action2
  ...
  #typen action3
;

```

The object-pair sequence must be terminated by a SEMI (;).

A binary integer type is nominally encoded as follows:

```
#nnnnn
| | | | |
| | | | +--- Level 1 argument type
| | | +--- Level 2 argument type
| | +--- Level 3 argument type
| +--- Level 4 argument type
+--- Level 5 argument type
```

Each "n" is a hexadecimal digit representing an object type, as shown in the table below. Thus #00011 represents two real numbers; #000A0 indicates a symbolic class object (symb, id, or lam) in level 2 and any type of object in level 1. There are also two-digit object type numbers, ending in F; use of any of these consequently reduces the total number of arguments that can be encoded in a single type integer. For example, #13F4F represents a real number in level 3, an extended real in level 2, and an extended complex in level 1.

The following table shows the hex digit values for each argument type. The column "# name" shows the object pointer name for the corresponding binary integer that may be used for a single argument function. The "Binary Integers" chapter contains a list of built-in binary integers that may be used for various common two-argument combinations.

Value	Argument	# name	User TYPE
0	Any Object	any	
1	Real Number	real	0
2	Complex Number	cmp	1
3	Character String	str	2
4	Array	arry	3,4
5	List	list	5
6	Global Name	idnt	6
7	Local Name	lam	7
8	Secondary	seco	8
9	Symbolic	symb	9
A	Symbolic Class	sym	6,7,9
B	Hex String	hxs	10
C	Graphics Object	grob	11
D	Tagged Object	TAGGED	12
E	Unit Object	unitob	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	External object1		27
CF	External object2		28
DF	External object3		29
EF	External object4		30

2.8.13 Loop Control Structures

a) Indefinite Loops

Addr	Word	Stack diagram	Explanation
071A2	BEGIN	→	Copies the interpreter pointer (RPL variable I) onto the return stack. Also called IDUP .
62B6F	UNTIL	flag →	If flag is TRUE , drops the top pointer on the return stack, otherwise copies that pointer to the interpreter pointer.
633F8	WHILE	flag →	If the flag is TRUE , then does nothing. Else drops the first pointer from the return stack, and skips the interpreter pointer past the next two objects.
071E5	REPEAT	→	Copies the first pointer on the return stack to the interpreter pointer.

071AB	AGAIN	→	Copies the first pointer on the return stack to the interpreter pointer.
-------	-------	---	--

BEGIN <test clause> WHILE <loop object> REPEAT	BEGIN <loop clause> UNTIL	BEGIN <loop clause> AGAIN Terminating this loop requires an error event, or a direct manipulation of the return stack.
--	---------------------------------	---

b) Definite Loops

Addr	Word	Stack diagram	Explanation
073DB	#1+_ONE_DO	#finish →	Equivalent to #1+ ONE DO; commonly used to execute a loop #finish times.
073DB	DO	#finish #start →	Begins DO loop
63466	DROLOOP	ob →	Performs DROP, then LOOP
6347F	DUP#0_DO	# → #	Begins # . . . #0 DO loop
63411	DUPINDEX@	ob → ob ob #index	Does DUP, then returns value of index in topmost DoLoop env.
6400F	ExitAtLOOP	→	Stores zero in stopping value of topmost DoLoop environment
63411	INDEX@	→ #index	Returns index of topmost DoLoop environment
63790	INDEX@#-	# → #'	Subtracts index value of topmost DoLoop environment from #
07270	INDEXSTO	# →	Stores # as index of top DoLoop environment
07249	ISTOP@	→ #stop	Returns stop value of the topmost DoLoop environment
07295	ISTOPSTO	# →	Stores new stop value in the topmost DoLoop environment
07258	JINDEX@	→ #index	Returns index of second DoLoop environment
073A5	LOOP	→	End of loop structure
633C6	NOT_UNTIL	flag →	End of loop structure
073DB	ONE_DO	#finish →	Begins #1..#finish DO loop
63439	OVERINDEX@	ob1 ob2 → ob1 ob2 ob1 #index	Does OVER, then returns value of index in topmost DoLoop environment
63425	SWAPINDEX@	ob1 ob2 → ob2 ob1 #index	Does SWAP, then returns value of index in topmost DoLoop environment
6344D	SWAPLOOP	ob1 ob2 → ob2 ob1	Does SWAP, then LOOP
6400F	ZEROISTOPSTO	→	Stores zero as the stop value in the topmost DoLoop environment
073C3	ZERO_DO	#finish →	Begins DO loop from #0 to #finish
63498	toLEN_DO	{list} → {list}	Begins DO loop from #1 of elements in list to stop value #number-of-elements+1.

2.8.14 Error Generation & Trapping

Addr	Word	Stack diagram	Explanation
04EA4	ABORT	→	Does ERRORCLR and ERRJMP
1502F	DO#EXIT	msg# →	Stores a new error number and executes ERRJMP; also executes AtUserStack. Puts the object ERRJMP on the stack
141E5	ERRBEEP	→	Generates an error beep
63155	ERRJMP	→	Invokes error handling subsystem
04CE6	ERROR@	→ #	Returns the current error number
04D33	ERRORCLR	→	Stores zero as the error number
6383A	ERROROUT	# →	Stores a new error number and does ERRJMP
04D0E	ERRORSTO	# →	Stores new error number
04E5E	ERRSET	→	used in ERRSET <[suspect ob]> ERRTRAP <[if-error action]>
04EB8	ERRTRAP	→	Skips next object in runstream.

2.8.15 Test and Control

a) Flags and Tests

Addr	Word	Stack diagram	Explanation
03EB1	AND	flag1 flag2 → flag	If flag1 and flag2 are both TRUE then TRUE else FALSE.
03AC0	FALSE	→ FALSE	Puts the FALSE flag on the stack.
6350B	FALSETRUE	→ FALSE TRUE	
2F934	FalseFalse	→ FALSE FALSE	
6362D	OR	flag1 flag2 → flag	If either flag1 or flag2 is TRUE then TRUE else FALSE.

635B0	ORNOT	flag1 flag2 → flag3	Logical OR followed by logical NOT.
03AF2	NOT	flag → flag'	If flag is TRUE then FALSE else TRUE.
62C55	NOTAND	flag1 flag2 → flag3	Logical NOT, then logical AND.
62C91	ROTAND	flag1 ob flag2 → ob flag3	Does ROT, then logical AND.
03A81	TRUE	→ TRUE	Puts the TRUE flag on the stack.
634F7	TrueFalse	→ TRUE FALSE	
0BBED	TrueTrue	→ TRUE TRUE	
03ADA	XOR	flag1 flag2 → flag	If both flag1 and flag2 are either TRUE or FALSE then FALSE, else TRUE.
5380E	COERCEFLAG	TRUE → %1 FALSE → %0	Converts a system flag to a real number flag.

◆ General Object Tests

Addr	Word	Stack diagram	Explanation
03B2E	EQ	ob1 ob2 → flag	If objects ob1 and ob2 are the same object, i.e. occupy the same physical space in memory, then TRUE else FALSE.
03B97	EQUAL	ob1 ob2 → flag	where ob1 and ob2 are not primitive code objects. If objects ob1 and ob2 are the same then TRUE else FALSE (this word is the system equivalent of the user RPL command \$SAME)
635D8	2DUPEQ	ob1 ob2 → ob1 ob2 flag	Returns TRUE if ob1 and ob2 have the same physical address.
63605	EQOR	flag1 ob1 ob2 → flag2	Does EQ, then logical OR.
63619	EQUALOR	flag1 ob1 ob2 → flag2	Does EQUAL, the logical OR.
6303D	EQOVER	ob1 ob2 ob3 → ob1 flag ob1	Does EQ, then OVER.
635C4	EQUALNOT	ob1 ob2 → flag	Returns FALSE if ob1 is equal to ob2.
62198	TYPEARRY?	ob → flag	TRUE if the object is an array
62193	DTYPEARRY? DUPTYPEARRY?	ob → ob flag	Dups and TRUE if the object is an array
6212F	TYPEBINT?	ob → flag	TRUE if the object is a binary integer
6212A	DUPTYPEBINT?	ob → ob flag	Dups and TRUE if the object is a binary integer
62256	TYPECARRY?	ob → flag	TRUE if the object is a complex array
62025	TYPECHAR?	ob → flag	TRUE if the object is a character
62020	DUPTYPECHAR?	ob → ob flag	Dups and TRUE if the object is a character
62183	TYPECMP?	ob → flag	TRUE if the object is a complex number
6217E	DUPTYPECMP?	ob → ob flag	Dups and TRUE if the object is a complex number
621EC	TYPECOL?	ob → flag	TRUE if the object is a program
621E7	DTYPECOL? DUPTYPECOL?	ob → ob flag	Dups and TRUE if the object is a program
62159	TYPECSTR?	ob → flag	TRUE if the object is a character string
62154	DTYPECSTR? DUPTYPECSTR?	ob → ob flag	Dups and TRUE if the object is a character string
6204F	TYPEEXT?	ob → flag	TRUE if the object is an unit
6204A	DUPTYPEEXT?	ob → ob flag	Dups and TRUE if the object is an unit
62201	TYPEGROB?	ob → flag	TRUE if the object is a graphics object
621FC	DUPTYPEGROB?	ob → ob flag	Dups and TRUE if the object is a graphics object
62144	TYPEHSTR?	ob → flag	TRUE if the object is a hex string
6213F	DUPTYPEHSTR?	ob → ob flag	Dups and TRUE if the object is a hex string
6203A	TYPEIDNT?	ob → flag	TRUE if the object is an identifier (global name)
62035	DUPTYPEIDNT?	ob → ob flag	Dups and TRUE if the object is an identifier (global name)
6211A	TYPELAM?	ob → flag	TRUE if the object is a temporary identifier (local name)
62115	DUPTYPELAM?	ob → ob flag	Dups and TRUE if the object is a temporary identifier (local name)
62216	TYPELIST?	ob → flag	TRUE if the object is a list
62211	DTYPELIST? DUPTYPELIST?	ob → ob flag	Dups and TRUE if the object is a list
6223B	TYPERARRY?	ob → flag	TRUE if the object is a real array
6216E	TYPEREAL?	ob → flag	TRUE if the object is a real number
62169	DTYPEREAL? DUPTYPEREAL?	ob → ob flag	Dups and TRUE if the object is a real number
621AD	TYPEROMP?	ob → flag	TRUE if the object is a ROM pointer (XLIB name)
621A8	DUPTYPEROMP?	ob → ob flag	Dups and TRUE if the object is a ROM pointer (XLIB name)
621C2	TYPERRP?	ob → flag	TRUE if the object is a directory
621BD	DUPTYPERRP?	ob → ob flag	Dups and TRUE if the object is a directory
621D7	TYPESYMB?	ob → flag	TRUE if the object is a directory

621D2	DUPTYPESYMB?	ob → ob flag	Dups and TRUE if the object is a directory
6222B	TYPETAGGED?	ob → flag	TRUE if the object is a tagged
62226	DUPTYPETAG?	ob → ob flag	Dups and TRUE if the object is a tagged

◆ Binary Integer Comparisons

Equality is tested in the sense of EQUAL (not EQ). Ordering treats all binary integers as unsigned.

Addr	Word	Stack diagram	Explanation
03D19	##	## → flag	TRUE if # = #.
03D4E	#<>	## → flag	TRUE if # <> # (not equal).
03CA6	#0=	# → flag	TRUE if # = 0
03CC7	#0<>	# → flag	TRUE if # <> 0
03CE4	#<	## → flag	TRUE if # < #'
03D83	#>	## → flag	TRUE if # > #'
6289B	2DUP#<	## → ## flag	TRUE if # < #'
628B5	2DUP#=#	## → ## flag	TRUE if # = #'
62266	DUP#0=	# → # flag	TRUE if # = #0
622C5	DUP#1=	# → # flag	TRUE if # = #1
622D4	DUP#0<>	# → # flag	TRUE if # <> #0
622C5	DUP#1=	# → # flag	TRUE if # = #1
63687	DUP#<7	# → # flag	TRUE if # < #7
63BAA	DUP%0=	% → % flag	TRUE if % = %0
636F0	ONE#>	# → flag	TRUE if # > #1
63385	ONE_EQ	# → flag	TRUE if # is ONE
636DC	OVER#>	## → # flag	TRUE if # > #'
6364B	OVER#0=	# ob → # ob flag	TRUE if # is #0
6365F	OVER#<	## → # flag	TRUE if # > #'
620EB	OVER#=#	## → # flag	TRUE if # = #'
636DC	OVER#>	## → # flag	TRUE if # < #'

◆ Decimal Number Tests

Addr	Word	Stack diagram	Explanation
2A81F	%<	% %' → flag	TRUE if % < %'
2A8AB	%<=	% %' → flag	TRUE if % <= %'
2A8CC	%<>	% %' → flag	TRUE if % <> %'
2A8C1	%=	% %' → flag	TRUE if % = %'
2A87F	%>	% %' → flag	TRUE if % > %'
2A895	%>=	% %' → flag	TRUE if % >= %'
2A80B	%0<	% → flag	TRUE if % < 0
2A7BB	%0<>	% → flag	TRUE if % <> 0
2A75A	%0=	% → flag	TRUE if % = 0
2A788	%0>	% → flag	TRUE if % > 0
2A7E3	%0>=	% → flag	TRUE if % >= 0
2A80B	%%0<=	%% %%' → flag	TRUE if %% <= %%'
2A7BB	%%0<>	%% %%' → flag	TRUE if %% <> %%'
2A75A	%%0=	%% %%' → flag	TRUE if %% = 0
2A788	%%0>	%% %%' → flag	TRUE if %% > 0
2A7E3	%%0>=	%% %%' → flag	TRUE if %% >= 0
2A87F	%%>	%% %%' → flag	TRUE if %% > %%'
2A895	%%>=	%% %%' → flag	TRUE if %% >= %%'
2A8AB	%%<=	%% %%' → flag	TRUE if %% <= %%'
51B2A	C%#0=	C% %' → flag	TRUE if C% %' = (%%0,%%%0)
51B43	C%0=	C% → flag	TRUE if C% = (0,0)

b) Words that Operate on the Runstream

Addr	Word	Stack diagram	Explanation
06E97	' [MK: ~]	→ ob	Pushes the next object in the runstream on the stack (does not EVALUATE it).
06EEB	'R	→ ob	Pushes the next object in the return stack on the stack (does not EVALUATE it). If func contains :: 'R STO ;, then func ob1 ob2 will evaluate as ob1 STO ob2.

639DE	'R'R	→ ob ob	Pushes the next two objects in the return stack on the stack.
61B89	ticR	→ ob TRUE FALSE	This word works similarly to 'R, except that it returns a flag to indicate whether the end of the top return stack composite has been reached. That is, if the top return stack pointer points to an object pointer to SEMI, then ticR pops the return stack and returns only FALSE. Otherwise return the next object from the composite and TRUE, while advancing the return stack pointer to the next object.
07E50	>R	:: →	Inserts the body of :: into the runstream, just below the top one. (That is, pushes a pointer to the body of :: onto the return stack). An example of its use is :: ' :: <foo> ; >R <bar> ; which will, when executed, cause <bar> to be executed before <foo>.
2B498	R>	→ ::	Creates a program object from the composite body pointed to by the top return stack pointer, and pushes the program on the data stack and pops the return stack. Example: :: :: R> EVAL <foo> ; <bar> ; which, when executed, will cause <bar> to be executed before <foo>.
62C19	R@	→ ::	Same as R> except that the return stack is not popped.
06FB7	RDROP	→	Pops the return stack.
?	IDUP	→	Duplicates the top body in the runstream. (That is, pushes the RPL variable I onto the return stack).
06FD1	COLA	→	Drops the remainder past the next object in the runstream and executes this object.
61A3B	?SEMI	flag →	Exits the current program if flag is TRUE.
638E4	?SEMIDROP	ob TRUE → FALSE →	Drops ob if flag is TRUE; exits the current program if flag is FALSE.
0712A	?SKIP	flag →	If flag is TRUE, skips the next object following ?SKIP.
61A2C	NOT?SEMI	flag →	Exits the current program if flag is FALSE.

c) If/Then/Else

Addr	Word	Stack diagram	Explanation
070C3	RPITE	flag ob1 ob2 → ?	If flag is TRUE then drop flag and ob2 and EVALUATE ob1, else drop flag and ob1 and EVALUATE ob2.
070FD	RPIT	flag ob → ?	If flag is TRUE then drop flag and EVALUATE ob, else just drop flag and ob.
63E48	IT	flag →	If flag is TRUE then execute the next object in the runstream; otherwise skip that object.
63E89	ITE	flag →	If flag is TRUE the execute the next object in the runstream, and skip the second object; otherwise skip the next object and execute the second.
0712A	?SKIP	flag →	If flag is TRUE, skip the next object in the runstream; otherwise, execute it.
0714D	SKIP	→	Skips over the next object in the runstream and continues execution beyond it. The sequence SKIP ; is a NOP.
63E89	#0=ITE	# →	#0= ITE
63E9D	#<ITE	# →	#0< ITE
62C2D	#=ITE	# →	#= ITE
63EB1	#>ITE	# →	#> ITE
63E61	ANDITE	flag flag' →	AND ITE
63EC5	DUP#0=ITE	# → #	DUP #0= ITE
63E2F	EQIT	ob1 ob2 →	EQ IT
63E75	EQITE	ob ob' →	EQ ITE
63E48	DUP#0=IT	# → #	DUP #0= IT
63EED	SysITE	# →	
63ED9	UserITE	# →	

d) CASE words

Addr	Word	Stack diagram	Explanation
61993	case	flag →	if TRUE, executes the next object in the runstream and skips the remaining program; if FALSE, skips the next object in the runstream.
618D3	#=casedrop	# # → # # → #	Should be named OVER#=casedrop.

5F181	%1=case	% →	
5F127	%0=case	% → flag	
63DDF	ANDNOTcase	flag1 flag2 →	
63CEA	ANDcase	flag1 flag2 →	
6191F	case2drop	ob1 ob2 TRUE → FALSE →	
618D3	casedrop	ob TRUE → FALSE →	
61891	DUP#0=case	# → #	
618A8	DUP#0=csedrp	# → # # →	If # <> #0 If # = #0
63DF3	EQUALNOTcase	ob ob' →	
63CFE	EQUALcase	ob ob' →	
63CA4	EQUALcasedrp	ob ob' ob' → ob ob' ob" → ob	
61933	EQcase	ob1 ob2 →	
619AD	NOTcase	flag →	
618E8	NOTcasedrop	ob FALSE → TRUE →	
629BC	ORcase	flag1 flag2 →	
6187C	OVER#=case	## → #	
63BEB	caseDoBadKey	flag →	Exit via DoBadKey
63BD2	caseDrpBadKey	ob TRUE → FALSE →	Exit via DoBadKey
61970	case2DROP	ob1 ob2 TRUE → FALSE →	
6194B	caseDROP	ob TRUE → FALSE →	
6359C	caseFALSE	TRUE → FALSE FALSE →	
634E3	caseTRUE	TRUE → TRUE FALSE →	
63547	casedrpf1s	ob TRUE → FALSE FALSE →	
63583	case2drpf1s	ob1 ob2 TRUE → FALSE FALSE →	
638B2	casedrptru	ob TRUE → TRUE FALSE →	
63CBD	DUP#0=csDROP	#0 → # → #	If # <> 0.
638CB	NOTcaseTRUE	FALSE → TRUE TRUE →	

2.8.16 Stack Operations

Addr	Word	Stack diagram
03258	2DROP	ob1 ob2 →
6254E	2DROP00	ob1 ob2 → #0 #0
62B0B	2DROPFALSE	ob1 ob2 → FALSE
031AC	2DUP	ob1 ob2 → ob1 ob2 ob1 ob2
63C40	2DUP5ROLL	ob1 ob2 ob3 → ob2 ob3 ob2 ob3 ob1
611F9	2DUPSWAP	ob1 ob2 → ob1 ob2 ob2 ob1
63FBA	2OVER	ob1 ob2 ob3 ob4 → ob1 ob2 ob3 ob4 ob1 ob2
62001	2SWAP	ob1 ob2 ob3 ob4 → ob3 ob4 ob1 ob2
60F4B	3DROP	ob1 ob2 ob3 →
611FE	3PICK	ob1 ob2 ob3 → ob1 ob2 ob3 ob1
63C68	3PICK3PICK	ob1 ob2 ob3 → ob1 ob2 ob3 ob1 ob2
630B5	3PICKOVER	ob1 ob2 ob3 → ob1 ob2 ob3

62EDF	3PICKSWAP	ob1 ob3 ob1 ob2 ob3 → ob1 ob2 ob1 ob3
60FAC	3UNROLL	ob1 ob2 ob3 → ob3 ob1 ob2
60F7E	4DROP	ob1 ob2 ob3 ob4 →
6121C	4PICK	ob1 ob2 ob3 ob4 → ob1 ... ob4 ob1
630C9	4PICKOVER	ob1 ob2 ob3 ob4 → ob1 ob2 ob3 ob4 ob1 ob4
62EF3	4PICKSWAP	ob1 ob2 ob3 ob4 → ob1 ob2 ob3 ob1 ob4
60FEB	4ROLL	ob1 ob2 ob3 ob4 → ob2 ob3 ob4 ob1
6109E	4UNROLL	ob1 ob2 ob3 ob4 → ob4 ob1 ob2 ob3
6113C	4UNROLL3DROP	ob1 ob2 ob3 ob4 → ob4
62D09	4UNROLLDUP	ob1 ob2 ob3 ob4 → ob4 ob1 ob2 ob3 ob3

63015	4UNROLLROT	ob1 ob2 ob3 ob4 → ob4 ob3 ob2 ob1
60F72	5DROP	ob1 ... ob5 →
6123A	5PICK	ob1 ... ob5 → ob1 ... ob5 ob1
60FD8	5ROLL	ob1 ... ob5 → ob2 ... ob5 ob1
62880	5ROLLDROP	ob1 ... ob5 → ob2 ... ob5
610C4	5UNROLL	ob1 ... ob5 → ob5 ob1 ... ob4
60F66	6DROP	ob1 ... ob6 →
6125E	6PICK	ob1 ... ob6 → ob1 ... ob6 ob1
61002	6ROLL	ob1 ... ob6 → ob2 ... ob6 ob1
60F54	7DROP	ob1 ... ob7 →
61282	7PICK	ob1 ... ob7 → ob1 ... ob7 ob1
6106B	7ROLL	ob1 ... ob7 → ob2 ... ob7 ob1
612A9	8PICK	ob1 ... ob8 → ob1 ... ob8 ob1
6103C	8ROLL	ob1 ... ob8 → ob2 ... ob8 ob1
63119	8UNROLL	ob1 ... ob8 → ob8 ob1 ... ob7
0314C	DEPTH	ob1 ... obn → ob1 ... obn #n
03244	DROP	ob →
627A7	DROPDUP	ob1 ob2 → ob1 ob1
6210C	DROPPFALSE	ob → FALSE
63FA6	DROPNDROP	ob1 ... obn # ob →
62946	DROPONE	ob → #1
63029	DROPOVER	ob1 ob2 ob3 → ob1 ob2 ob1
632F9	DROPRDROP	ob → (and pops 1 return stk level)
62FC5	DROPROT	ob1 ob2 ob3 ob4 → ob2 ob3 ob1
6270C	DROPSWAP	ob1 ob2 ob3 → ob2 ob1
62726	DROPSWAPDROP	ob1 ob2 ob3 → ob2
62103	DROPTTRUE	ob → TRUE
62535	DROPZERO	ob → #0
627D5	DUP	ob → ob ob
6119E	DUP#1+PICK	... #n → ... #n obn
611F9	DUP3PICK	ob1 ob2 → ob1 ob2 ob2 ob1
61099	DUP4UNROLL	ob1 ob2 ob3 → ob3 ob1 ob2 ob3
62CB9	DUPDUP	ob → ob ob ob
63A9C	DUPONE	ob → ob ob #1
630DD	DUPPICK	... #n → ... #n obn-1
630F1	DUPROLL	... #n → ... #n obn-1
62FB1	DUPROT	ob1 ob2 → ob2 ob2 ob1
63AD8	DUPTWO	ob → ob ob #2
61380	DUPUNROT	ob1 ob2 → ob2 ob1 ob2
63A88	DUPZERO	ob → ob ob #2
62F75	N+1DROP	ob ob1 ... obn #n →
62F75	NDROP	ob1 ... obn #n →

031D9	NDUP	ob1 ... obn #n → ob1 ... obn ob1 ... obn
5E370	NDUPN	ob #n → ob ... ob
63533	ONFALSE	→ #1 FALSE
62E67	ONESWAP	ob → #1 ob
63051	OVER	ob1 ob2 → ob1 ob2 ob1
63C90	OVER5PICK	v w x y z → v w x y z y v
62CCD	OVERDUP	ob1 ob2 → ob1 ob2 ob1 ob1
62D31	OVERSWAP	ob1 ob2 → ob2 ob1 ob1
62D31	OVERUNROT	ob1 ob2 → ob1 ob1 ob2
61184	PICK	obn ... #n → ... obn
612DE	ROLL	obn ... #n → ... obn
62F89	ROLLDROP	obn ... #n → ...
62D45	ROLLSWAP	obn ... ob #n → ... obn ob
1DAB8	ROT	ob1 ob2 ob3 → ob2 ob3 ob1
62726	ROT2DROP	ob1 ob2 ob3 → ob2
62C7D	ROT2DUP	ob1 ob2 ob3 → ob2 ob3 ob1 ob3 ob1
60F21	ROTDROP	ob1 ob2 ob3 → ob2 ob3
60F0E	ROTDROPSWAP	ob1 ob2 ob3 → ob3 ob2
62775	ROTDUP	ob1 ob2 ob3 → ob2 ob3 ob1 ob1
62CA5	ROTOVER	ob1 ob2 ob3 → ob2 ob3 ob1 ob3
6112A	ROTRROT2DROP	ob1 ob2 ob3 → ob3
60EE7	ROTSWAP	ob1 ob2 ob3 → ob2 ob1 ob3
62F2F	SWAP	ob1 ob2 → ob2 ob1
6386C	SWAP2DUP	ob1 ob2 → ob2 ob1 ob2 ob1
63C54	SWAP3PICK	ob1 ob2 ob3 → ob1 ob3 ob2 ob1
63C7C	SWAP4PICK	ob1 ob2 ob3 ob4 → ob1 ob2 ob4 ob3 ob4
62F5C	SWAPDROP	ob1 ob2 → ob2
62830	SWAPDROPDUP	ob1 ob2 → ob2 ob2
6284B	SWAPDROPSWAP	ob1 ob2 ob3 → ob3 ob1
21660	SWAPDROPTTRUE	ob1 ob2 → ob2 TRUE
62747	SWAPDUP	ob1 ob2 → ob2 ob1 ob1
63AB0	SWAPONE	ob1 ob2 → ob2 ob1 #1
61380	SWAPOVER	ob1 ob2 → ob2 ob1 ob2
60F33	SWAPROT	ob1 ob2 ob3 → ob3 ob2 ob1
4F1D8	SWAPTRUE	ob1 ob2 → ob2 ob1 TRUE
6133E	UNROLL	... ob #n → ob ...
28558	UNROT	ob1 ob2 ob3 → ob3 ob1 ob2
6112A	UNROT2DROP	ob1 ob2 ob3 → ob3
6284B	UNROTDROP	ob1 ob2 ob3 → ob3 ob1
62CF5	UNROTDUP	ob1 ob2 ob3 → ob3 ob1 ob2 ob2
6308D	UNROTOVER	ob1 ob2 ob3 → ob3 ob1 ob2 ob1
60F33	UNROTSSWAP	ob1 ob2 ob3 → ob3 ob2 ob1
63079	ZEROOVER	ob → ob #0 ob
5DE7D	reversym	ob1 ... obn #n → obn ... ob1 #n

2.8.17 Memory Operations

a) Temporary Memory

Addr	Word	Stack diagram	Explanation
37B44	CKREF	ob → ob'	If ob is in TEMPOB, is not embedded in a composite object, and is not referenced, then does nothing. Otherwise copies ob to

06B4E	INTEMNOTREF?	ob → ob flag	TEMPOB and returns the copy. If the input object is in TEMPOB area, is not embedded in a composite object, and is not referenced, returns ob and TRUE, otherwise returns ob and FALSE.
06657	TOTEMPOB	ob → ob'	Copies ob into TEMPOB and returns pointer to the new ob.

b) Variables

Addr	Word	Stack diagram	Explanation
08696	CREATE	ob id →	Creates a RAM-WORD with ob as its object part and the NAME FORM from id as its name part, in the current directory. An error occurs if ob is or contains the current directory ("Directory Recursion"). Assumes that ob is not a primitive code object.
18513	STO	ob id → ob lam →	In the lam case, the temporary identifier lam is re-bound to ob. The binding is to the first such temporary identifier object matching lam in the Temporary Environment area (searching from the first temporary environment to the last). An error is returned if lam is unbound. In the id case, STO attempts to match id to the name part of a global variable. If resolution is unsuccessful, STO creates a variable with ob as its object part and the name form from id as its name part, in the current directory. If resolution is successful, then ob replaces the object part of the resolved variable. If any updatable system object pointers reference the object part of the resolved variable, then the object part is placed into the temporary object area prior to the replacement and all affected updatable system object pointers are adjusted to reference the copy of the object part in the temporary object area. For the id case, STO assumes that ob is not a primitive code object.
61B45	@	id → ob TRUE id → FALSE lam → ob TRUE lam → FALSE	In the lam case, @ attempts to match lam to the temporary identifier object part of a binding in the Temporary Environment area (searching from the first temporary environment to the last). If successful, then the object bound to lam is returned along with a TRUE flag; else, a FALSE flag is returned. In the id case, @ attempts to match id to the name part of a global variable, starting in the current directory, and working up through parent directories if necessary. If the resolution is unsuccessful, then a FALSE flag is returned. Otherwise, the object part of the resolved variable is returned with a TRUE flag.
18513	?STO_HERE	ob id → ob lam →	This is the system version of user STO. It is the same as SAFESTO, except that for global variables, it a) stores only in the current directory; and b) will not overwrite a stored directory.
1853B	SAFE@_HERE	id → ob TRUE id → FALSE lam → ob TRUE lam → FALSE	Same as SAFE@, except for global variables the search is restricted to the current directory.
1854F	PURGE	id →	Purges variable specified by id; does no type check on stored object.
20B81	XEQRCL	id → ob	Same as SAFE@ for global variables, but errors if variable is nonexistent
18513	XEQSTOID	ob id →	Alternate name for ?STO_HERE

c) Directories

Addr	Word	Stack diagram	Explanation
08D08	CONTEXT!	rrp →	Stores a pointer to a rooted directory as the current directory
08D5A	CONTEXT@	→ rrp	Recalls the current context directory.
184E1	CREATEDIR	id →	Creates a directory object in the current directory.
18779	DOVARS	→ { id1 id2 ... }	Returns list of variable names from the current directory.
08D92	HOMEDIR	→	Makes HOME the current directory.
1848C	PATHDIR	→ { HOME dir dir ... }	Returns the current path.
1A16F	UPDIR	→	Switches context to the parent of the current directory.
20FF2	XEQORDER	{ id1 id2 ... } →	ORDERs current directory.
18595	XEQPGDIR	id →	Purges a directory while respecting reference/garbage collection conventions.

d) The Hidden Directory

Addr	Word	Stack diagram	Explanation
6408C	PuHiddenVar	id →	Purges the hidden variable named id.
64023	RclHiddenVar	id → ob TRUE id → FALSE	Recalls (@) a hidden variable.
64078	StoHiddenVar	ob id →	Stores ob in hidden variable

e) Additional Memory Utilities

Addr	Word	Stack diagram	Explanation
05F42	GARBAGE	→	Forces garbage collection.
05F61	MEM	→ #	Returns the amount of free memory (a garbage collection is not forced)
05944	OCRC	ob → #nibbles checksum(hxs)	Returns size of object in nibbles and a hex string checksum
6595A	getnibs	hxsDATA hxsADDR → hxsDATA'	Internal RPL version of PEEK
6594E	putnibs	hxsDATA hxsADDR →	Internal RPL version of POKE

2.8.18 Display Management & Graphics

a) Display Organization

Addr	Word	Stack diagram	Explanation
12655	ABUFF	→ textgrob	
12665	GBUFF	→ graphgrob	
12635	HARDBUFF	→ Hbgrob	Returns whichever of the text or graph grob is currently displayed.
12645	HARDBUFF2	→ menugrob	
0E128	HBUFF_X_Y	→ HBgrob #x1 #y1	

b) Preparing the Display

Addr	Word	Stack diagram	Explanation
130AC	RECLAIMDISP	→	Makes the text grob the current display, clears it and resizes it to the standard size (131x56) if necessary.
39531	ClrDA1IsStat	→	Suspends the ticking clock display.
4E2CF	TURNMENUOFF	→	Removes the menu from the display and resizes the text grob to 131x64.
4E347	TURNMENUON	→	Restores the menu display.

c) Controlling Display Refresh

Addr	Word	Stack diagram	Explanation
3902C	SetDA1Temp	→	Freezes display area 1
?	SetDA2aTemp	→	Freezes display area 2a
?	SetDA2bTemp	→	Freezes display area 2b
?	SetDA3Temp	→	Freezes display area 3
3922F	SetDAsTemp	→	Freezes all display areas
39144	ClrDAsOK	→	Redraws the entire lod when program ends

d) Clearing the Display

Addr	Word	Stack diagram	Explanation
126DF	BLANKIT	#startrow #rows →	Clears #rows starting at #startrow
3A578	BlankDA12	→	Clears rows 0 through 56
3A55F	BlankDA2	→	Clears rows 16 through 56
134AE	CLEARVDISP	→	Zeros out all of HARDBUFF
0E06F	Clr16	→	Clears top 16 pixel rows
0E083	Clr8	→	Clears top 8 pixel rows
0E097	Clr8-15	→	Clears pixel rows 8-15

e) Annunciator Control

Addr	Word	Stack diagram	Explanation
1133A	ClrAlphaAnn	→	Clears the alpha annunciator

1136E	ClrLeftAnn	→	Clears the left-shift annunciator
11354	ClrRightAnn	→	Clears the right-shift annunciator
1132D	SetAlphaAnn	→	Sets the alpha annunciator
11361	SetLeftAnn	→	Sets the left-shift annunciator
11347	SetRightAnn	→	Sets the right-shift annunciator

f) Display Coordinates

Addr	Word	Stack diagram	Explanation
0E0FB	Rows8-15	→ HBgrob #x1 #y1+8 #x1+131 #y1+16	
0E0AB	TOP16	→ HBgrob #x1 #y1 #x1+131 #y1+16	
0E0D3	TOP8	→ HBgrob #x1 #y1 #x1+131 #y1+8	
137B6	WINDOWCORNER	→ #x #y	Returns pixel numbers of upperleft corner of the window
?	Save16	→ grob	Calls TOP16 and SUBGROB to produce a grob consisting of the top 16 rows of the current display

g) Displaying Text

When the text grob is the current display AND has not been scrolled (except for `DISPROWn*`), the following words may be used to display text. Long strings are truncated to 22 characters with a trailing ellipsis (...), and strings shorter than 22 characters are blank filled.

Addr	Word	Stack diagram	Explanation
1245B	DISPROW1	\$ →	Displays text in row 1 with the medium (5x7) font.
12725	DISPROW1*	\$ →	Displays text in row 1 with the medium (5x7) font, even if the display has been scrolled.
1246B	DISPROW2	\$ →	Displays text in row 2 with the medium (5x7) font.
12748	DISPROW2*	\$ →	Displays text in row 2 with the medium (5x7) font, even if the display has been scrolled.
1247B	DISPROW3	\$ →	Displays text in row 3 with the medium (5x7) font.
1248B	DISPROW4	\$ →	Displays text in row 4 with the medium (5x7) font.
1249B	DISPROW5	\$ →	Displays text in row 5 with the medium (5x7) font.
124AB	DISPROW6	\$ →	Displays text in row 6 with the medium (5x7) font.
124BB	DISPROW7	\$ →	Displays text in row 7 with the medium (5x7) font.
12429	DISPN	\$ #row →	Displays text in row #row with the medium (5x7) font.
3A4CE	Disp5x7	\$ #start #max →	Displays text starting at row #start with the medium (5x7) font, for a maximum of #max rows.
12415	BIGDISPROW1	\$ →	Displays text in row 1 with the big (5x9) font.
12405	BIGDISPROW2	\$ →	Displays text in row 2 with the big (5x9) font.
123F5	BIGDISPROW3	\$ →	Displays text in row 3 with the big (5x9) font.
123E5	BIGDISPROW4	\$ →	Displays text in row 4 with the big (5x9) font.
123C8	BIGDISPN	\$ #row →	Displays text in row #row with the big (5x9) font.
38926	FlashWarning	\$ →	Displays a warning message during 3 seconds.

h) Graphics Objects

◆ Graphics Tools

Addr	Word	Stack diagram	Explanation
11CF3	\$>BIGGROB	\$ → grob	5x9 font
11D00	\$>GROB	\$ → grob	5x7 font
11F80	\$>grob	\$ → grob	3x7 font
503D4	DOLCD>	→ 64x131grob	LCD→
11679	GROB!	grob1 grob2 #col #row →	Stores grob1 into grob2. This is a bang-type word with no error checks!
11A6D	GROB! ZERO	grob #x1 #y1 #x2 #y2 → grob'	Zeros out a rectangular section of a grob. NOTE: Bang-type operation.
6389E	GROB! ZERODRP	grob #x1 #y1 #x2 #y2 →	Zeros out a rectangular section of a grob. NOTE: Bang-type operation!
12F94	GROB>GDISP	grob →	Stores graph grob with new grob.
12635	HARDBUFF	→ Hbgrob	The current display grob
12DD1	HEIGHTENGROB	grob #rows →	Adds #rows to grob, unless grob is null. NOTE: Assumes text grob or graph grob!
122FF	INVGROB	grob → grob'	Invert grob data bits - bang-type.

50B08	LINEOFF	#x1 #y1 #x2 #y2 →	Clears pixels in a line in text grob. Note: #x2 must be > #x1 (use ORDERXY#)
50ACC	LINEOFF3	#x1 #y1 #x2 #y2 →	Clears pixels in a line in graph grob. Note: #x2 must be > #x1 (use ORDERXY#)
50B17	LINEON	#x1 #y1 #x2 #y2 →	Sets pixels in a line in text grob. Note: #x2 must be > #x1 (use ORDERXY#)
50AEA	LINEON3	#x1 #y1 #x2 #y2 →	Sets pixels in a line in graph grob. Note: #x2 must be > #x1 (use ORDERXY#)
1158F	MAKEGROB	#height #width → grob	
51893	ORDERXY#	#x1 #y1 #x2 #y2 → #x1 #y1 #x2 #y2	Orders two points for line drawing
1383B	PIXOFF	#x #y →	Clears a pixel in the text grob
1380F	PIXOFF3	#x #y →	Clears a pixel in the graph grob
1384A	PIXON	#x #y →	Sets a pixel in the text grob
13992	PIXON?	#x #y → flag	Returns TRUE if text grob pixel is set
13986	PIXON?3	#x #y → flag	Returns TRUE if graph grob pixel is set
13825	PIXON3	#x #y →	Sets a pixel in the graph grob
1192F	SUBGROB	grob #x1 #y1 #x2 #y2 → subgrob	
659DE	Symb>HBufF	symb →	Displays symb in HARDBUFF in Equation-Writer form. May enlarge HARDBUFF, so do RECLAIMDISP afterwards.
50AF9	TOGGLE	#x1 #y1 #x2 #y2 →	Toggles pixels in a line in text grob. Note: #x2 must be > #x1 (use ORDERXY#)
50ADB	TOGGLE3	#x1 #y1 #x2 #y2 →	Toggles pixels in a line in graph grob. Note: #x2 must be > #x1 (use ORDERXY#)

◆ Grob Dimensions

Addr	Word	Stack diagram	Explanation
4F7E6	CKGROBFITS	grob1 grob2 #n #m → grob1 grob2' #n #m	Truncates grob2 if it doesn't fit in grob1
5179E	DUPGROBDIM	grob → grob #height #width	
5187F	GBUFFGROBDIM	→ #height #width	Returns dimensions of graph grob
50578	GROBDIM	grob → #height #width	
63C04	GROBDIMw	grob → #width	

◆ Built-in Grobs

Addr	Word	Stack diagram	Explanation
66EA5	BigCursor	→ grob	5x9 Cursor (outline box)
5053C	CROSSGROB	→ grob	5x5 "+" symbol
13D8C	CURSOR1	→ grob	5x9 Insert Cursor (arrow)
13DB4	CURSOR2	→ grob	5x9 Replace Cursor (solid box)
5055A	MARKGROB	→ grob	5x5 "X" symbol
66ECD	MediumCursor	→ grob	5x7 Cursor (outline box)
66EF1	SmallCursor	→ grob	3x5 Cursor (outline box)

◆ Menu Display Utilities

Addr	Word	Stack diagram	Explanation
3A297	Grob>Menu	#col 8x21grob →	Displays an 8x21 (only!) grob
3A2DD	Id>Menu	#col Id →	Recalls Id and displays standard label or directory label, depending on the contents of Id.
3A2C9	Seco>Menu	#col seco →	Evaluates secondary and uses result to produce and display appropriate menu label
3A2B5	Str>Menu	#col \$ →	Makes and displays standard menu label
3A38A	MakeBoxLabel	\$ → grob	Box with bullet in it
3A3EC	MakeDirLabel	\$ → grob	Box with directory bar
3A44E	MakeInvLabel	\$ → grob	White label (solver)
3A328	MakeStdLabel	\$ → grob	Black label (standard)

i) Scrolling the Display

Addr	Word	Stack diagram	Explanation
4D16E	SCROLLDOWN	→	Scroll down one pixel with repeat
4D150	SCROLLLEFT	→	Scroll left one pixel with repeat

4D18C	SCROLLRIGHT	→	Scroll right one pixel with repeat
4D132	SCROLLUP	→	Scroll up one pixel with repeat
516AE	JUMPBOT	→	Move window to bottom edge of grob
516E5	JUMPLEFT	→	Move window to left edge of grob
51703	JUMPRIGHT	→	Move window to right edge of grob
51690	JUMPTOP	→	Move window to bottom edge of grob

2.8.19 Keyboard Control

a) Key Locations

Addr	Word	Stack diagram	Explanation
41CA2	Ck&DecKeyLoc	%rc.p → #KeyCode #Plane	WRITE returns %rc.p
41D92	CodePl>%rc.p	#KeyCode #Plane → %rc.p	

b) Waiting for a Key

Addr	Word	Stack diagram	Explanation
04708	CHECKKEY	→ #KeyCode TRUE → FALSE	Returns, but does not pop, the next key in the buffer.
00D71	FLUSHKEYS	→	Flushes the key buffer.
04714	GETTOUCH	→ #KeyCode TRUE → FALSE	Pops next key from buffer.
42402	KEYINBUFFER?	→ flag	Returns TRUE if a key is in the buffer, otherwise returns FALSE.
42262	ATTN?	→ flag	Returns TRUE if [ATTN] has been pressed
05068	ATTNFLAGCLR	→	Clears the attn key flag (does not flush attn key from buffer)
41F65	WaitForKey	→ #KeyCode #Plane	Returns next fully formed keystroke.

c) InputLine

Addr	Word	Stack diagram	Explanation
42F44	InputLine	\$prompt \$editline #cursorpos #insrep #entry #alphalock ILMENU #ilmenuraw flagattnabort? #parse → ? flag	Internal INPUT. Displays the prompt in display area 2a, sets the keyboard entry modes, initializes the edit line, accepts user input, parses, evaluates or returns the user-input edit line, returns TRUE if exited by Enter or FALSE if aborted by Attn.

The stack on entry must contain the following: (from level 9 to level 1)

\$Prompt	The prompt to be displayed during user input
\$EditLine	The initial edit line
CursorPos	The initial edit line cursor position, specified as a binary integer character number or a two-element list of binary integer row and column numbers. For all numbers, #0 indicates the end of the edit line, row, or column.
#Ins/Rep	The initial insert/replace mode: #0 current insert/replace mode #1 insert mode #2 replace mode
#Entry	The initial entry mode: #0 current entry mode plus program entry #1 program/immediate entry #2 program/algebraic entry
#AlphaLock	The initial alpha-lock mode: #0 current alpha lock mode #1 alpha lock enabled #2 alpha lock disabled
ILMenu	The initial InputLine menu in the format specified by ParOuterLoop
#ILMenuRow	The initial InputLine menu row number in the format specified by ParOuterLoop
AttnAbort?	A flag specifying whether pressing Attn while a non-null edit line exists should abort InputLine (TRUE) or just clear the edit line (FALSE)
#Parse	How to process the resulting edit line: #0 Return the edit line as a string #1 Return the edit line as a string AND as a parsed object #2 Parse and evaluate the edit line

InputLine returns different results, depending on the initial value of #Parse:

#0	→ \$EditLine TRUE	Edit line
#1	→ \$EditLine ob TRUE	Edit line and parsed edit line
#2	→ Ob1 ... Obn TRUE	Resulting object or objects
any	→ FALSE	Attn pressed to abort edit

d) The Parameterized Outer Loop

Addr	Word	Stack diagram	Explanation
38985	ParOuterLoop	AppDisplay AppKeys NonAppKeyOK? DoStdKeys? AppMenu #AppMenuRow SuspendOK? ExitCond AppError →	Parameterized outer loop.

The parameterized outer loop, `ParOuterLoop`, takes nine arguments, in order:

AppDisplay	The display update object to be 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, a secondary object in the format described below.
NonAppKeyOK?	A flag specifying whether the hard keys not assigned by the application should perform their default actions or be canceled.
DoStdKeys?	A flag used in conjunction with "NonAppKeyOK?" specifying whether standard key definitions are to be used for non-application keys instead of default key processing.
AppMenu	The menu key specification, a secondary or list in the format specified in the menu key assignments document, or FALSE.
#AppMenuRow	The initial application menu row number. For most applications, this should be binary integer one.
SuspendOK?	A flag specifying whether or not any user command that would create a suspended environment and restart the system outer loop should instead generate an error.
ExitCond	An object that evaluates to TRUE when the outer loop is to be exited, or FALSE otherwise. "ExitCond" is evaluated before each application display update and key evaluation.
AppError	The error-handling object to be evaluated if an error occurs during the key evaluation part of the parameterized outer loop.

2.8.20 System Commands

Addr	Word	Stack diagram	Explanation
422A1	ALARM?	→ flag	Returns TRUE if an alarm is due
40BC9	AtUserStack	→	Declares user ownership of all objects on the stack.
0EB81	CLKTICKS	→ hxs	Returns 13 nibble hex string reflecting the number of ticks since 01/01/0000. There are 8192 ticks per second.
53761	ClrSysFlag	# →	Clears system flag from #1 to #64
53755	ClrUserFlag	# →	Clears user flag from #1 to #64
0CC0E	DATE	→ %date	Returns real number date
1415A	DOBEEP	%freq %duration →	BEEP command
53C43	DOBIN	→	Set base mode to BINARY
53C5B	DODEC	→	Set base mode to DECIMAL
166FB	DOENG	# →	Set ENG display with # (0-11) digits
166E3	DOFIX	# →	Set FIX display with # (0-11) digits
53C37	DOHEX	→	Set base mode to HEXadecimal
53C4F	DOOCT	→	Set base mode to OCTal
166EF	DOSCI	# →	Set SCI display with # (0-11) digits
16707	DOSTD	→	Set STD display mode
167BF	DPRADIX?	→ flag	Returns TRUE if current radix is . Returns FALSE if current radix is .
2A5D2	SETDEG	→	Set DEGREES angle mode

2A604	SETGRAD	→	Set GRADS angle mode
2A5F0	SETRAD	→	Set RADIANS angle mode
40EE7	SLOW	→	15msec delay
0CBFA	TOD	→ %time	Returns time of day in h.ms form
53784	TestSysFlag	# → flag	Returns TRUE if system flag # is set
53778	TestUserFlag	# → flag	Returns TRUE if user flag # is set
40F02	VERYSLOW	→	300 msec delay
40F12	VERYVERYSLOW	→	3 sec delay
54039	WORDSIZE	→ #	Returns binary wordsize
53CAA	Dostws	# →	Stores binary wordsize
1A7C9	Dowait	%seconds →	Waits for %seconds in light sleep

3 Tips and tricks

3.1 Meta Kernel and UFL

UFL (Universal Font Library) is an attempt to normalize the use of fonts in HP48 programs. The idea is to allow all programs to share the same fonts, instead of having as many fonts as software. UFL is a standard library. It provides two different fonts, the normal font or the mini-font. The **Meta Kernel** is able to work only with the last one, since the **MK** provides many more powerful features in font management, such as more than 240 different fonts at the same time, while UFL can manage only one. Also, you can change the system font, without restarting your HP48 with an ON-C.

You will find more informations about the UFL at this URL:

<http://www.engr.uvic.ca/~aschoorl/ufl/>

The **Meta Kernel** uses its own mini-font, but may use the UFL mini-font. Just execute these few lines, or add them in the STARTUP file:

```
1 FNT →MINIFONT
```

3.2 STARTED and EXITED examples

3.2.1 Remove the header display

When you edit files, it may be useful to have the whole screen available. Using the STARTED and EXITED capabilities, it is very easy to remove the header while editing, and restore it after.

```
STARTED:
«
  0 →HEADER
»

EXITED:
«
  2 →HEADER
»
```

3.2.2 Edit compressed files

When you are short in memory, it may be useful to compress some files. BZ was written by Mika Heiskanen (<http://www.hut.fi/~mheiskan/>) it is a very efficient and very fast compressor.

You will find the BZPlus package on <http://hp48.ml.org/utills/compress/>

The problem with the compressor, is that if you want to edit the file, you must uncompress the file manually, edit it, compress it again, then store it.

With STARTED and EXITED, you won't have to do that anymore. The idea is to check if a file is a compressed file with STARTED, if yes uncompress it, and when leaving the command line, compress it with EXITED.

The main difficulty is that the **Meta Kernel** can edit more than one text at the same time. Then you must save the history of the edited files, in order to know if the file was compressed or not. The history will be saved in the HOME directory with the name CHECK.BZ

STARTED is called by the **Meta Kernel** just before editing an object, and finds the object on the first stack level. Sometimes the **MK** calls STARTED without any valid object, in this case, you find the object NULL\$ on the stack.

EXITED is called by the **Meta Kernel** just after exiting the command line by pressing ENTER or ON. When the user press ENTER, the **MK** pushes two objects on the stacks as follow:

2: Object

1: TRUE

If the user has pressed ON, only FALSE is pushed on the first stack level.

The following STARTED and EXITED use both SytemRPL and UserRPL, they may be enhanced very easily by using only SystemRPL. Anyway, it is a good example of how to use the MASD in RPL mode.

```

"!NO CODE
!RPL
::
    xDUP NULL$ EQ                (Check if it's the NULL$ object)
    ?SKIP                        (If yes do nothing)
    ::
        xPATH xHOME xSWAP        (Save the working directory)
        xIF xBZ?                 (Is it a compressed file?)
        xTHEN
            :: xBZU %1 ;         (If yes, uncompress it)
        xELSE %0
        xIFEND
        xIFERR                    (Check if the history file exists)
            :: ' ID CHECK.BZ xRCL ;
        xERRTHEN
            ::                    (If not, create it with a null list)
                xDROP {}
                ' ID CHECK.BZ xSTO
                %1
        ;
        xIFEND
        xDROP                    (Save the history entry)
        ' ID CHECK.BZ xSTO+
        xSWAP xEVAL              (Go to the previous directory)
    ;
;
@
EXITED:
"!NO CODE
!RPL
::
    xPATH xSWAP xHOME            (Save the working directory)
    ITE                          (If the user pressed ENTER)
    ::
        xSWAP                    (Get the entry in the history file)
        ID CHECK.BZ
        xIF xHEAD                (If it was a compressed file)
            ::                    (Compress it again)
                xIFERR xBZC xERRTHEN xIFEND
        ;
        xIFEND

```



```

                xSWAP TRUE
;
FALSE
ID CHECK.BZ           (Remove the entry in the history)
xTAIL
' ID CHECK.BZ xSTO    (Go to the previous directory)
xSWAP xEVAL
;
@"

```

3.3 Meta Kernel and Erable

Erable is a very powerful Algebra system for the HP48GX. Combined with the **Meta Kernel**, you will get one of the best handheld computer algebra system (far better than the TI-92). Here is an example of an STARTEQW file, which allow to use all Erable's functions from the Equation Writer, by pressing **CST**.

This STARTEQW was written by Erable's author: Bernard Parisse

You will find more information about Erable at:

<http://www-fourier.ujf-grenoble.fr/pp/parisse/english.html>

```

STARTEQW:
« "ERABLE"
{ "1.SIMP" "2.TRIG" "3.INTDER" "4.EXEC"
  "5.PLOT" "6.CFG" "7.ARIT" }
1 CHOOSE2
IF THEN
  DUP
  {
  { EXPA COLC EXPLIN SINCOS TEXPA LNCOLC TSIMP
    { "0.BACK" STARTEQW } }
  { TRIGLIN TRIGCOS TRIGSIN TAN2SC TAN2SC2 HALFTAN
    { "0.BACK" STARTEQW }
  }
  { der1 RISCH PF LAP ILAP { "0.BACK" STARTEQW } }
  {
  { "1.NEW VX" « DUP 'VX' STO » }
  { 2 "REPLACE" « "REPLACE" ( "'X='" 4 ALG V )
    INPUT2 STR→ EXEC » }
  { "3.COMMAND" « "COMMAND" ( "" 4 ALG V )
    INPUT2 STR→ EVAL » }
  { "4.ISOL VX" « VX EXEC » }
  { "0.BACK" STARTEQW }
  }
  {
  { "1.ERASE AND PLOT" « ERASE DUP STEQ
    # B4045h LIBEVAL » }
  { "2.ADD TO PLOT" « EQ DUP TYPE 5 ==
    IF THEN OVER + ELSE OVER 2 →LIST END
    STEQ # B4045h LIBEVAL » }
  { "3.PLOT MENU" « # B4045h LIBEVAL » }
  { "0.BACK" STARTEQW }
  }
  {
  { "1.COMPLEX MODE" « 13 SF » }
  { "2.REAL MODE" « 13 CF » }
  { "3.INTEGER ARIT" « 10 SF » }
  { "4.POLYN. ARIT" « 10 CF » }
  { "0.BACK" STARTEQW }
  }
  }

```

```

    { CHS re im conj ( "BACK" STARTEQW ) }
  }
  SWAP
  1 1 SUB STR→ GET 1 CHOOSE2
  IF THEN EVAL END
END
»

```

3.3.1 STARTOFF and TOFF example

Here is a completely useless program. But like all completely useless programs, it is completely essential.

This is a screen saver program, it will displays points in the screen until you press a key.

If you want the **Meta Kernel** to run this program after two minutes of idle time, put `#983040d` in the 'TOFF' file.

```

STARTOFF:
«
PICT→ LCD→ →PICT ( #0 #0 ) PVIEW
DO
RAND 131 * R→B
RAND 64 * R→B
2 →LIST PIXON
UNTIL KEY END DROP
TEXT →PICT
»

```

4 The MK for Programmers

4.1 Customizing the Filer

4.1.1 Overview

The filer is completely customizable. You can run your own programs from inside the filer. These programs can utilize all of the functionality of the filer, which the internal functions use. You can also access all internal functions. This customization is accomplished using a custom menu that displays labels for each of your custom programs. In the filer, if you press CST or CUSTOM in the built-in menu, the filer will try to find the file 'FILER.CUSTOM'. This file is used to create your custom menu.

4.1.2 FILER.CUSTOM Format

FILER.CUSTOM is a list with the following arguments :

```
{
GROB 131*n                This is the GROB that will be displayed in the
                           menu area when you press CST. You can
                           also use a program that gives a GROB as a
                           result.

System Binary 1           Number of pages for the menu
{
    System binary 2       Execution Type
    System binary 3       Exit type
    [Program]             Program which will be run by the filer
    [System binary 4]     Shortcut-key for the program
}
}
{
    ....An entry just as above for each custom program
}
}
```

4.1.3 Explanation of each component

a) System Binary 1 and GROB

This is an integer that tells the FILER how many pages the menu will use. For example, if you want two pages (12 labels) then *GROB 131*n* would be a 131*16 GROB and SB1 would be 2.

b) System binary 2

This integer allows you to control when the program will be started.

There are 5 possibilities:

SB	Program Control
0	You can run the program everywhere (VAR,PORT,BACKUP,LIB)
1	Only when you are in VAR
2	Won't run if you are in a library
3	Won't run if you are in a backup
4	Will run only in port (home of the port)

c) System binary 3

This integer tells the filer how the associated program will be runned. This also allows you to access all the filer's internal subroutines. There are two categories of custom programs: internal calls, and custom calls.

NOTE: You can use a Binary integer instead of System Binary (it is easier to manage with the User-RPL compiler).

d) Internal calls

The internal calls are included so that you can access the built-in functionalities of the filer from inside your custom menu. For these calls, you don't need to put a program in the file unless you want to allow a shortcut (Ex Left-Shift 2). If you use a shortcut then add any program you want because it won't be runned at all. You should use the smallest possible program (ex: TakeOver \$40788) internal empty program).

SB	Operation
0	Bip
1	Info
2	Hexa
3	View
4	Arbo
5	Up
6	MaxUp
7	Down
8	Maxdown
9	Select (same as ENTER)
A	Updir
B	Downdir
C	Previous menu
D	Next menu
E	EVAL
F	Swap header
18	Display or remove file's details (Type& size)
19	EDIT
1A	COPY
1B	MOVE
1C	RCL
1D	PURGE
1E	RENAME
1F	CRDIR
20	ORDER
21	SEND
22	HALT2
23	EDITB

e) Custom program

If you are going to run a custom program there are 7 ways to call the program. Every time, the filer will place the working path on the stack.

Examples:

`{ }` for HOME
`:nb:{ }` for a port

:02: { FOO.DIR } if you are working in port 2 in the backup FOO.DIR

If you launch the program in the VAR, then your program will start in the current directory. The possible calls are: (1: ect is the resulting stack that will appear just before your program is run)

SB	Type of call
10	Recalls only the path: 1: Path
11	Recalls the name and the object for the currently selected object: 3: Path 2: Object 1: Name
12	Multiple objects selected n: Path ... 5: Object 2 4: Name 2 3: Object 1 2: Name 1 1: Number of objects (System Binary)
13	The FILER will call your program after each object. The FILER will place the following on the stack each time and call your program for each object selected. 3: Path 2: Object 1 1: Name 1
14	Recalls the name of the current object only 2: Path 1: Name
15	Recalls all the names selected n: path ... 3: Name2 2: Name1 1: Number of names selected (System Binary)
16	Recalls the current object only in a string of addresses 2: Path 1: String
17	Recalls the selected objects in a string of addresses 2: Path 1: String

f) Explanation for the string of addresses

This string recalled by the filer is a list of addresses. You can use it only in ML or with FILER_FRCL, FILER_FNAME and FILER_NXTADR, for SysRPL use:

FILER_FRCL (C030F): Takes a String of addresses, and recalls the first object.
FILER_FNAME (C030A): Same thing but recalls the first name object,
FILER_NXTADR (C0314): Takes a string of addresses, and removes the first addresses. And FALSE if there are some addresses left. TRUE if not.

The string format is:

Prolog (02A2C) Size AdrName1 AdrObject1 AdrName2 AdrObject2 ...

Each address is on 5 nibbles.

g) Explanations about the name

For all calls except 14 and 15, the name is the real filename. If you call the program from a library, then the name will be the name of the library itself (Like 'Strurt: v4.31 (c)'). However, for 14 and 15, if you have selected a library, the name will be a real number that is the library number. For 13, if the object is a library, then the name will be a 'L' plus the Id number.

Example: For Library 1303, the name will be: 'L1303'

NOTE: You can browse every library, but can only run your programs from an attached library.

h) Warning and very important points

In order to save room, we used list of addresses. So, if you modify the directory structural in your programs (using STO, PURGE, etc.), then NEVER USE CALL 13. And if you are a very experienced programmer, you can use calls 16 and 17.

4.2 Machine Language entry points list

Here is a list of entry points inside the **Meta Kernel**. They can be used by experienced programmers for programs to take advantage of the **MK**. Of course, if these entry points are used, the program won't run without the **MK**.

Call a routine with **GOSBVL C0xxx**.

Rf means the R register with the field f. Example: Ca describes the register C, field A.

Input and output registers are described in the 'In' and 'out' sections.

4.2.1 MINI_DISP

Address	C0040
Description	Displays a Mini-font character string
In	
Ca	= number of characters
D1	= string address
D0	= display starting address (in a 131*x grob)
ST11	= inversion (white on black)

4.2.2 MINI_FONT_ADDRESS

Address	C0047
Description	Mini-font address in the card
E.g	D0=C0047 A=DAT0.A

4.2.3 DISPLAY_SBR

Address	C004C
Description	Main display routine
Call INIT_DISPLAY_LINE before using this routine.	
In	
R0b	= font identifier
R1s	= (font height) * 2 - 1
D0	= starting display address in a grob
D1	= string address
Da	= number of characters
R2a	= display grob width in nibbles

Ba = left margin in characters
 Ca = right margin in characters
 ST4 = display left shift
 ST5 = display right shift
 ST0,1,2,3 = text attributes (bold, italic, underlined, inversed)
 ST8 = display carriage returns as characters if set
 ST9 = mini-font display

Out

D0 = next line address in the grob
 Aa = last written character address
 Da = number of remaining characters
 ST8 = set if a CR has been read
 D1 = end of displayed text in the grob

4.2.4 EDIT_SBR

Address C0053

Description Mini-edit (used by **FIND** in the command line)
 Call INIT_DISPLAY_LINE before using this routine.

In

R4a = screen address
 Ca = maximum number of characters
 D1 = buffer address
 ST9 = mini-font display

Out

Carry = set if **ENTER** has been pressed, clear exit by **ON**

4.2.5 FILER.ADR

Address C0319

Description Address of the main FILER program

4.2.6 SCAN_FONT

Address C009A

Description Rebuild the font table
 This function forces the **MK** to search through all ports to build the font table.

4.2.7 RECONFIG

Address C00A1

Description Reconfigures the second card
 After a DECONFIG, this routine reconfigures the second port card at C0000 (hidden by the **MK** card), and the bank switcher at 7F000.

4.2.8 DECONFIG

Address C00AB

Description Deconfigures the second card
 This routine is used to gain direct access to the second card, without deconfiguring the **MK**.
 The second port card is moved from C0000 (where it is hidden by the **MK** card) to 40000. The bank switcher is moved from 7F000 to 3F000.

4.2.9 CMD_SIZE

Address C00BE

Description Computes the command line size
Out
Aa = number of characters.

4.2.10 GET_PATH

Address C00C5
Description Finds the relative path of a directory
In
D1 = buffer address
Ca = maximum length
If ST2 = 0, the path is taken from the HOME directory to the current directory
if ST2 = 1, D0 = starting directory address, Da = ending directory address

4.2.11 DECONFIG_RAM

Address C00CC
Description Moves the RAM from 80000 to 00000
Useful if you plan to make your own interrupt handler. If your RAM based program call this entry point, the return address (RTN) will be recalculated

4.2.12 RECONFIG_RAM

Address C00D3
Description Moves the RAM from 00000 back to 80000

4.2.13 GET_FONT

Address C00DA
Description Returns the address of a given font
Call SCAN_FONT once before using this routine.
In
Ab = font id
Out
R0a = font address

4.2.14 INIT_DISPLAY_LINE

Address C00E1
Description Initializes the display routines
This routine initializes the registers necessary for the MK display routines.
In
Ca = screen width in nibbles

4.2.15 CHANGE_FLAG

Address C00E8
Description Updates the ST flags 0 to 3 and R0 for the display routine
Computes some parameters for the display routines, when special characters are encountered (font control characters...)

4.2.16 GET_ASCII_KEY

Address C00F6
Description Keyboard handler
Gets an ASCII character or a key number if not in ALPHA mode. This routine handles the shifts and alpha displays.
Out
ST0 = 0 if ASCII

ST0 = 1 if key number.

4.2.17 GET_KEY

Address C00FD
Description Keyboard handler
Out

Carry = set if no character in the keyboard buffer.

Aa = key code (if carry cleared)

4.2.18 RUN_KEY

Address C0104
Description Key assignment
This routine associates an assembly code with a key.

Syntax:

GOSBVL =RUN_KEY

GOIN4 NO_KEY_ROUTINE

\$/01 GOIN4 KEY_A % 01 is the A key code

\$/xx GOIN4 KEY_x % xx key code (same as in External)

\$\$\$ GOIN4 NOT_HANDLED_KEY % a bip routine for example

4.2.19 MINI_DISP_VAL

Address C010B
Description Displays a hexa value in mini-font
In

D0 = screen address

Bw = number to be displayed

Ca = number of digits to be displayed

ST11 = inverse (white on black)

ST10 = 1 to display leading zeros

4.2.20 MINI_DISP_AWP

Address C0112
Description Displays a hexadecimal value in mini-font
In

D0 = screen address

Awp = number to be displayed

P = (number of digits) - 1

ST11 = inverse (white on black)

ST10 = 1 to display leading zeros

4.2.21 TRUP

Address C0119
Description Copy up

Equivalent to 066B9, but works even after a DECONFIG_RAM (when RAM is at 00000)

4.2.22 TRDN

Address C0120
Description Copy down

Equivalent to 0670C, but works even after a DECONFIG_RAM (when RAM is at 00000)

4.2.23 ZEROM

Address **C0127**
Description Resets a memory zone to 0
In
 D1 = zone starting address
 Ca = number of nibbles to blank

4.2.24 SCAN_KEY

Address **C012E**
Description Keyboard handler
Does a keyboard scan, updates the keyboard buffer and the keystate. Use it when interrupts are disabled.

4.2.25 NEW_ADR

Address **C0135**
Description Moves an object at the end of the temporary zone
Moves an object in the temporary RAM at the end of this zone, so that **16671** (object resize) and **RESIZE_PLUS** can be used on it.

4.2.26 RESIZE_PLUS

Address **C013C**
Description Resize up a character string
Resize up the last character string in the temporary RAM.
In
 R0a = string address
 D0 = new end of the character string

4.2.27 C=IN3

Address **C02F2**
Description Equivalent to C=IN2, but in the **MK** card

4.2.28 A=IN3

Address **C02FA**
Description Equivalent to A=IN2, but in the **MK** card

4.2.29 OUT=C=IN3

Address **C02EF**
Description Equivalent to OUT=C=IN, but in the **MK** card

4.2.30 OUT=CA=IN3

Address **C02F7**
Description Equivalent OUT=C A=IN, but in the **MK** card

4.2.31 DISP_DEC

Address **C0143**
Description Displays a decimal number
In
 D0 = screen address
 Ca = number to be displayed
 ST11 = inverse (white on black)
 ST10 = 1 to display leading zeros

4.2.32 MULT_BAC

Address C014A
Description Ba = Ca * Aa

4.2.33 GREY?

Address C0151
Description Tests if a grob is in grayscale
In
D1 = grob address
Out
Carry = set if grob is in grayscale

4.2.34 BEEP

Address C0158
Description Do a beep if the system flag -56 is clear

4.2.35 KEY_REPEAT

Address C015C
Description Keyboard handler routine
Repeats a routine associated with a key.
Syntax:
GOSBVL =KEY_REPEAT
\$/keymask % in/out mask e.g \$/02040 for Q (OUT=040, IN=02)
GOTO KEY_REPEAT_ROUTINE
GOTO END_KEY_REPEAT

4.2.36 KEY_NO

Address C0160
Description Waits for all keys to be released

4.2.37 HEX_DEC

Address C0164
Description Hexa to decimal conversion
In
Ca = hexa to be converted
Out
Aa = decimal result

4.2.38 A_DIV_C

Address C0168
Description Ba = Aa / Ca

4.2.39 SET_BIT

Address C016C
Description Sets a bit in a bit field
In
D1 = bit field address
Aa = bit number (to be set to 1)

4.2.40 CLEAR_BIT

Adresse FEC3F
Description Clears a bit in a bit field

In
 D1 = bit field address
 Aa = bit number (to be clear)

4.2.41 BIT?

Address C0173
Description Tests a bit in a bit field
In
 D1 = bit field address
 Aa = bit number (to be tested)
Out
 Carry = set to the value of the tested bit

4.2.42 BZU.SBR

Adresse C017A
Description Décompression BZU
 Uncompress data using Mika Heiskanen algorithm in his program BZ.
In
 D0 = Address of the object to decompress (strings without the B prolog **BZ**).
 D1 = Address where you want to uncompress.

4.2.43 INV.ZONE

Address C0186
Description Inverse a memory zone
In
 D0 = Address zone.
 Ca = Size in nibbles to inverse

4.2.44 OFF.SBR

Address C0192
Description Turns off the machine (real Off)
 OFF in ML, doesn't reboot if libraries have changed.

4.2.45 InitTable

Adresse FECA0
Description Initializes memory zones for SysRPL-table routines
 Initializes pointers used by the following routines, which deal with the System RPL entry point table.
 There are examples on the floppy disk, in the file EXAMPLES2.DIR.
Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.
Out
 Carry set if no table has been found.
 Uses RSTK(5) Aw Cw Ba D0 D1 Da, plus the following memory zones:
 (b=bytes, n=nibbles)
 DC @ResZone BE4A5 % on 256b, table name
 DC NbExternals BE422 % on 5n, number of entries
 DC @TableText BE427 % on 5n, address of the first entry
 DC @TableHash BE42C % on 5n, address of the hash table
 DC @Table→e BE431 % on 5n, address of the pointers table, sorted
 by addresses
 DC TablePort BE436 % on 2n, port number of the table
 DC RclPort BE3AF % on 2n, port number of the table

4.2.46 GetAdr

Adresse **FECAC**

Description Returns the address associated with a name

Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.

In

DO = Address of the text to find, followed by a separation character

Out

Carry = Set if name not found

Ca = Address

Uses Aw, Cw, Ba, Da, D0, D1, RSTK(5).

DC @ResZone BE4A5 % on 256b, name saved

4.2.47 GetText

Adresse **FECAC**

Description Returns a pointer to the name associated to an address

Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.

In

Ca = Address

Out

Carry = Set if address not found

D1 = Address of the text (2n for the length, followed by the characters)

Uses Aa, Ba, Ca, Da, D0, D1, RSTK(2).

4.2.48 GetFirst

Adresse **FECB2**

Description Finds the first name beginning with the given text

Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.

In

DO = Address of the text to find, followed by a separation character

Out

Carry = Set if no name found

D1 = Address of the text (2n for the length, followed by the characters)

Uses Aw, Ba, Cw, RSTK(4), D0, D1, Da

DC @ResZone BE4A5 % sur 256o sauvegarde du nom à rechercher

4.2.49 GetNext

Adresse **FECB8**

Description Finds the next name beginning with the given text

Use GetFirst first, values of Bb, Da and D1 must be preserved.

Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.

In

DO = Address of the text to find, followed by a separation character

Out

Carry = Set if no name found

D1 = Address of the text (2n for the length, followed by the characters)

Uses Aw, Ba, Cw, RSTK(4), D0, D1, Da

DC @ResZone BE4A5 % sur 256o sauvegarde du nom à rechercher

4.2.50 GetFirstAdr

Adresse **FECBE**

Description Returns a pointer to the first table entry

Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.

Out

D0 = Address of an entry structure:

Address (5n), name length (2n), name.

Uses D0, D1, Aa, Da, Ca, RSTK(2).

4.2.51 GetNextAdr

Adresse **FEC4**

Description Returns a pointer to the next table entry

Use GetFirstAdr first, values of Da, et D1 must be preserved.

The table is sorted by addresses.

Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.

Out

Carry = Set if no more entries

D0 = Address of an entry structure (see GetFirstAdr)

Uses D0, D1, Aa, Da, Ca, RSTK(2).

4.2.52 RclPath

Adresse **FEC39**

Description Finds the address of a file

Warning: Port 2 must be at address 40000. Use the DECONFIG and RECONFIG routines.

In

DC @ResZone BE4A5 % on 256b, filename (Masd syntax, ends with an ASCII 0)

Out

Carry = Set if no file found

D1 = Address of the file

Uses Aw, Ba, Cw, RSTK4, D0, D1

4.3 SysRPL entry points

Here is a list of entry points inside the **Meta Kernel**. They can be used by programmers for programs to take advantage of the **MK**. Of course, if these entry points are used, the program won't run without the **MK**.

Call a routine with **PTR** `xxxxxx` in **MASD**, or with **€** `xxxxxx`

4.3.1 GET.INDEX

Address **FEC96**

Description Create an index for a fast access

In

1 : Matrix (List of list)

Out

2 : Matrix (list of list)

1 : Index (Strings)

4.3.2 GET .Y.I

Address **C006A**
Description Recall a row from a matrix
In
 3 : Matrix (List of list)
 2 : Index (String)
 1 : Row nb (System Binary)
Out
 3 : Matrix (List of list)
 2 : Index (String)
 1 : List

4.3.3 GET .X.Y.I

Address **C0305**
Description Recall a cell
In
 4 : Matrix (List of list)
 3 : Index (String)
 2 : X (System Binary)
 1 : Y (System Binary)
Out
 3 : Matrix (List of list)
 2 : Index (String)
 1 : Object

4.3.4 PUT.X.Y

Address **FEC64**
Description Put an object into a cell
The matrix has to be splited using INNERCOMP (\$ 54AF) ex :
{ { 1 2 3 } { 4 5 6 } } INNERCOMP :
3 : { 1 2 3 }
2 : { 4 5 6 }
1 : <2>
In
 N : First row
 ...
 5 : Row n
 4 : Number of rows (System Binary)
 3 : X (System Binary)
 2 : Y (System Binary)
 1 : Object
Out
 N : First row
 ...
 2 : Row n
 1 : Number of rows (System Binary)

4.3.5 DIMS

Address **FEC9B**
Description Give the dimension of a matrix
In
 1 : Matrix (list of list)

Out

If it's a matrix

4 : Matrix

3 : X (System Binary)

2 : Y (System Binary)

1 : TRUE

or

1 : FALSE

4.3.6 FILER_FNAME**Address** C030A**Description** Voir la page 131**4.3.7 FILER_FRCL****Address** C030F**Description** Voir la page 131**4.3.8 FILER_NEXTADR****Address** C0314**Description** Voir la page 131**4.3.9 SURPRISE****Address** C031E**Description** Just try**4.3.10 INPUT****Address** C0080**Description** SysRPL INPUT, works like **InputLine** (42F44)

This INPUT command takes 10 arguments. Consult page 122.

4.3.11 CTRL_LOOP**Address** C0085**Description** Parameterized outer loop, like **ParOuterLoop** (38985)

Takes 9 arguments. Consult page 123.

4.3.12 INIT_CMD**Address** C008A**Description** Initializes the command line**4.3.13 CMP_PLUS****Address** C0090**Description** Adds a character to the command line string**4.3.14 DISP_CALL****Address** C0095**Description** Text display**In**

2: Character string

1: TRUE if carriage returns are to be interpreted

4.3.15 BZU

Address C018D

Description Uncompress with BZU

Uncompress the compressed strings using Mika Heiskanen algorithm in his program BZ.

In

1 : Strings (without its BZ prolog).

Out

1 : Object

4.3.16 KER_PARAM

Address C0181

Description OuterLoop

Internal version of the ParOuterLoop. This program doesn't create a local environment to save the menus, therefore you can use unnamed local variables.

4.3.17 MINI_EDITOR

Address C0199

Description Mini editor on one screen line

In

3: String (if boolean on level 2 is TRUE)

2: Boolean

1: Line number, where the command line has to be put.

Out

2: String

1: Boolean (TRUE if line validated by ENTER, else FALSE)

If the boolean is TRUE, the editor is initialised with the character string on level 2.

4.3.18 KEY_PARAM

Address FEC82

Description Recall a program associated to a key

Assigns a program to key. Recalls the corresponding program.

In

3: Key code

2: Shift code

1: program list (string)

The list is as follow in MASD format :

STRING C

\$/axx E xxxxx

% a is 1 if in alpha mode, else 0. xx is the code of the key, plus the ML shift code:

% RightShift=C0 ; LeftShift=40. For example for the LeftShift+ENTER, run the

%internal command DUP :

\$/059 EXP(5)DUP

%at the end

}

Out

If the key is in the list :

2: program

1: TRUE

Otherwise

1: FALSE

4.3.19 UPSTACK.ADR

Address FEC69
Description Internal programs list
Recalls the programs list used by the Interactive Stack. See KEY_PARAM.
Out
1: String

4.3.20 WAITKEY.ADR

Address FEC6E
Description Internal programs list
Recalls the programs list used by the Kernel. See KEY_PARAM.
Out
1: String

4.3.21 COMMANDLINE.ADR

Address FEC73
Description Internal programs list
Recalls the programs list used by the command line editor. See KEY_PARAM.
Out
1: String

4.3.22 MATRIX.ADR

Address FEC87
Description Internal programs list
Recalls the programs list used by the Matrix Writer. See KEY_PARAM.
Out
1: String

4.3.23 SAVE.ARG

Address FEC78
Description Save the LASTARG
This program is very useful, while you want to run a User RPL command, and you don't want this program to save the last arg. See LOAD.ARG too

Ex :
"!NO CODE
!RPL
EQU SAVE.ARG FEC78
EQU LOAD.ARG FEC7D
::
 SAVE.ARG
 xDUP
 LOAD.ARG
;
@"

In case of errors, the program will restore the good stack.

4.3.24 LOAD.ARG

Address FEC7D
Description Restore the LASTARG saved by SAVE.ARG
See above

Index

- !Directives, 73
 - !PATH, 61
 - !RPL, 62
 - !UNIT, 62
- !—**
- Matrix, 43
 - Big strings, 30
 - BIN, 18
 - BOL, 27
 - BZ, 125
- #—**
- ⌘, 18
 - \$, 31
 - ⌘, 31
 - ~, 31
 - ⌘, 31
- 1—**
- IUS, 18
- A—**
- A→, 77
 - A, 76
 - ALG, 18
 - ALL
 - Command line, 27
 - APEEK, 77
 - AR~LST, 80
 - ARBO
 - Filer, 50
 - ASM, 79
 - ASM→, 75, 79
 - Assembler, 56, 84
 - Assembler syntax, 56
 - Auto indentation, 29
 - Auto-completion, 65
 - Auto-indentation, 22
- B—**
- BEG
 - Command line, 26
 - BEGIN
 - Command line, 26
- C—**
- CD→, 76
 - CD, 76
 - Character, 31
 - CHARS, 54
 - CHOOSE2, 24, 79
 - CODE, 31, 60
 - COL
 - Matrix, 43
 - +COL
 - Matrix, 43
 - COLCT
 - EQW, 37
 - Command library, 31
 - Command line, 26
 - Compiler directives, 73
 - Completion, 65
 - Compression, 125
 - Constants, 58
 - COPY
 - Command line, 26
 - Filer, 50
 - Matrix, 43
 - CRLIB, 62
 - CUSTO
 - Filer, 51
 - CUT
 - Command line, 26
- D—**
- DEBUG, 78
 - DEC, 18
 - DEG, 18
 - DEL
 - Command line, 26
 - Matrix, 43
 - DEL L
 - Command line, 26

DEL→
 Command line, 27
←DEL
 Command line, 27
DIMS, 80
Directives, 73
Disassembler, 75
DISP2, 78
Display font, 19
DRPN
 Interactive stack, 25
DUPN
 Interactive stack, 25

—E—

ECHO
 Chars, 54
 Interactive stack, 25
ECHO1
 Chars, 54
EDIT, 28, 79
 Filer, 50
 Matrix, 43
EDITB, 79
 Filer, 51
END
 Command line, 26
 Matrix, 43
→END
 Command line, 26
Equation editor, 32
EQW, 32, 79
ER, 79
Erable, 127
Error messages
 EQW, 38
 Masd, 73
EVAL
 EQW, 37
 Filer, 50
EXIT
 Hexa editor, 51
EXITED, 21, 125
EXITs, 60
EXPAN
 EQW, 37
Expressions, 59
External, 31, 94

External entry points, 98

—F—

FALSE, 31, 96
FAST
 Command line, 26
 Interactive stack, 25
 Matrix, 43
File manager, 49
Filename conventions, 61
Filer, 49
FILER, 79
FILER.CUSTOM, 21, 129
FIND
 Command line, 26
 Hexa editor, 51
Flags, 21, 80
FNT2GRB.PRG, 81
Font, 19
FONT, 28
Font editor, 54
FONT→, 80
→FONT, 80
FONT6, 80
FONT8, 80
Fonts, 28
Full screen editing, 22, 29

—G—

GO→
 Matrix, 43
GOTO
 Command line, 26
 Hexa editor, 51
 Interactive stack, 25
GRB2FNT.PRG, 81
GRD, 18
GREY, 31
→GROB
 EQW, 38
GROB editor, 44
→GROB2, 78

—H—

H→, 76
H→S, 76

→H, 76
HALT
 Filer, 50
HALT2, 78
HEAD
 Filer, 51
HEADER→, 78
→HEADER, 78
HEX, 18
HEXA
 Filer, 50
Hexa editor, 51
HIDDEN, 62
HLT, 18

—I—

INCLUDE, 31
INFO
 Command line, 26
 Filer, 50
 Interactive stack, 25
INPUT2, 78
Installation, 16
Interactive stack, 25
INV, 28
ITALI, 28

—K—

K\$, 31
KEEP
 Interactive stack, 25
KERNEL?, 80

—L—

Labels, 57
LCONFIG, 62
LEVEL
 Interactive stack, 25
LIB, 62
Library creation, 62
Links, 57
→LIST
 Interactive stack, 25

—M—

Macros, 61
Masd, 56
Masd syntax, 56
MATRIX, 42
Memory editor, 51
Menu labels, 18, 23
MESSAGE, 62
Mini-font, 20
MINIFONT→, 81
→MINIFONT, 81
MK entry points, 132
ML entry points in MK, 132
MODIF
 Chars, 54
MOVE
 Filer, 50
Multiline, 22

—N—

→NDISP, 80
New Masd instructions, 72
NEXT, 78
 Command line, 27
 Hexa editor, 51
NoEval, 31

—O—

OBJ, 31
OCT, 18
ORDER
 Filer, 50

—P—

PASTE
 Command line, 26
 Matrix, 43
PATH, 61
PEEK, 77
PICK
 Interactive stack, 25
PICTURE2, 44
POKE, 77
PRG, 18, 31
PRG~LST, 77

PROG, 31
PSADR
 Hexa editor, 51
PSOBJ
 Hexa editor, 51
PURG
 Filer, 50

—R—

R
 Command line, 27
R/N
 Command line, 27
R~SB, 77
RAD, 18
→RAM, 77
RCL
 Filer, 50
Registers, 84
RENAM
 Filer, 50
REPL
 Command line, 27
 EQW, 36
ROLL
 Interactive stack, 25
ROLLD
 Interactive stack, 25
-ROW
 Matrix, 43
+ROW
 Matrix, 43
RPL mode, 62
RdZ, 18
RdA, 18

—S—

S→H, 76
→S2, 78
→S4, 78
Saturn, 84
Saturn instruction set, 66, 89
SCAN
 Chars, 54
 Selection, 26
SEND
 Filer, 50

SKIP→
 Command line, 27
←SKIP
 Command line, 27
SKIPs, 59
SLOW
 Command line, 26
 Interactive stack, 25
 Matrix, 43
SREPL, 80
SREV, 77
SST, 78
Stack display, 18
Stack display font, 22
STARTED, 21, 125
STARTEQW, 21, 37, 127
STARTERR, 21
STARTEXT, 21, 63
STARTOFF, 21, 128
STARTSEND, 21, 50
STARTUP, 21
Status area, 18
→STK
 Matrix, 43
↑STK
 Command line, 27
 Matrix, 43
STRING, 60
STROBJ, 60
Style, 27
 Command line, 27
SUB
 EQW, 36
 System binary, 31
 System flags, 21
 System RPL, 62, 94
 System RPL entry points, 98
 System RPL entry points in MK, 140

—T—

TOFF, 21, 128
TRUE, 31, 96

—U—

UFL, 125
UNDE, 28
Units, 62

UPs, 60
USES, 62
USR, 18

—V—

VIEW
 Filer, 51
VISIBLE, 62
VISIT, 79
VISITB, 79

—W—

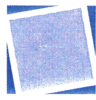
WID→
 Matrix, 43
←WID
 Matrix, 43

—X—

XLIB, 31
XYZ, 18

—Z—

ZOOM
 EQW, 38



PocketProfessional[®]

Software for HP 48 Graphic Expandable Calculators

Here are many of the HP 48 Calculator Software titles and accessories now available.

ENGINEERING

EE•Pro

Engineering Electronics Pac

Electrical Engineering Reference Pac

Solid State Electronics Pac

Spice48 Pac

Mechanical Engineering Pac

Mechanical Engineering Reference Pac

Structural Analysis

MATHEMATICS AND STATISTICS

Calculus Pac

Geometry/Surveying Pac

Math•Pro

Engineering Mathematics Pac

Mathematics Pac 1

Mathematics Pac 2

Mathematics Pac 2 - Professional Extension

Statistics Pac

NAVIGATION

Celestial Navigation Pac

Flight Navigation System w/GPS Connection

Navigare, Navigation Pac

SCIENCE

Chemistry Pac

Chemistry Reference Pac

Physics Pac

ACCESSORIES

AND PERIPHERALS

Business Pac

CalcWriter

Games48

Oil Cargo Calculations

Personal Information Manager

256K & 512K byte RAM Card; 48GX only

DISKETTE SOFTWARE AND SERIAL INTERFACE KITS

CalcWare Link - PC (HP 48G, GX, S, SX)

CalcWare Link - Mac (HP 48G, GX, S, SX)

Mathematics: Pre-Calculus

Mathematics: Calculus and Advanced Math

Physics: Mechanics & Thermodynamics

Physics: Electricity & Magnetism

Physics: Optics, Waves & Relativity

BOOKS FROM

ARMSTRONG PUBLISHING

HP 48 HandBook - An Overall Reference Book

HP 48 PocketBook - A Pocket Reference Guide

Intro to HP 48 System RPL - Most Advanced

Programmers Tool

For HP 48 owners in North America, Central and South America, Australia and Pacific Countries, call:

da Vinci Technologies Group, Inc.

899 NW Grant Avenue

Corvallis, Oregon 97330 USA

Phone (541) 757-8416 • Fax (541) 753-7821

Email: sales@sparcom.com

Website: <http://www.sparcom.com>

For HP 48 owners in Europe, Africa, Indian Subcontinent and Middle Eastern Countries, call:

BB Marketing Ans.

Ensjøveien. 12 b

Oslo, Norway

Phone (47) 22 67 11 57 • Fax (47) 22 68 05 05

Email: bbm@sn.no

Website: <http://www.sn.no/home/bbn>

© 1997 da Vinci Technologies Group, Inc. All rights reserved.

PocketProfessional and CalcWare are registered trademarks of da Vinci Technologies Group, Inc.

Printed in U.S.A.