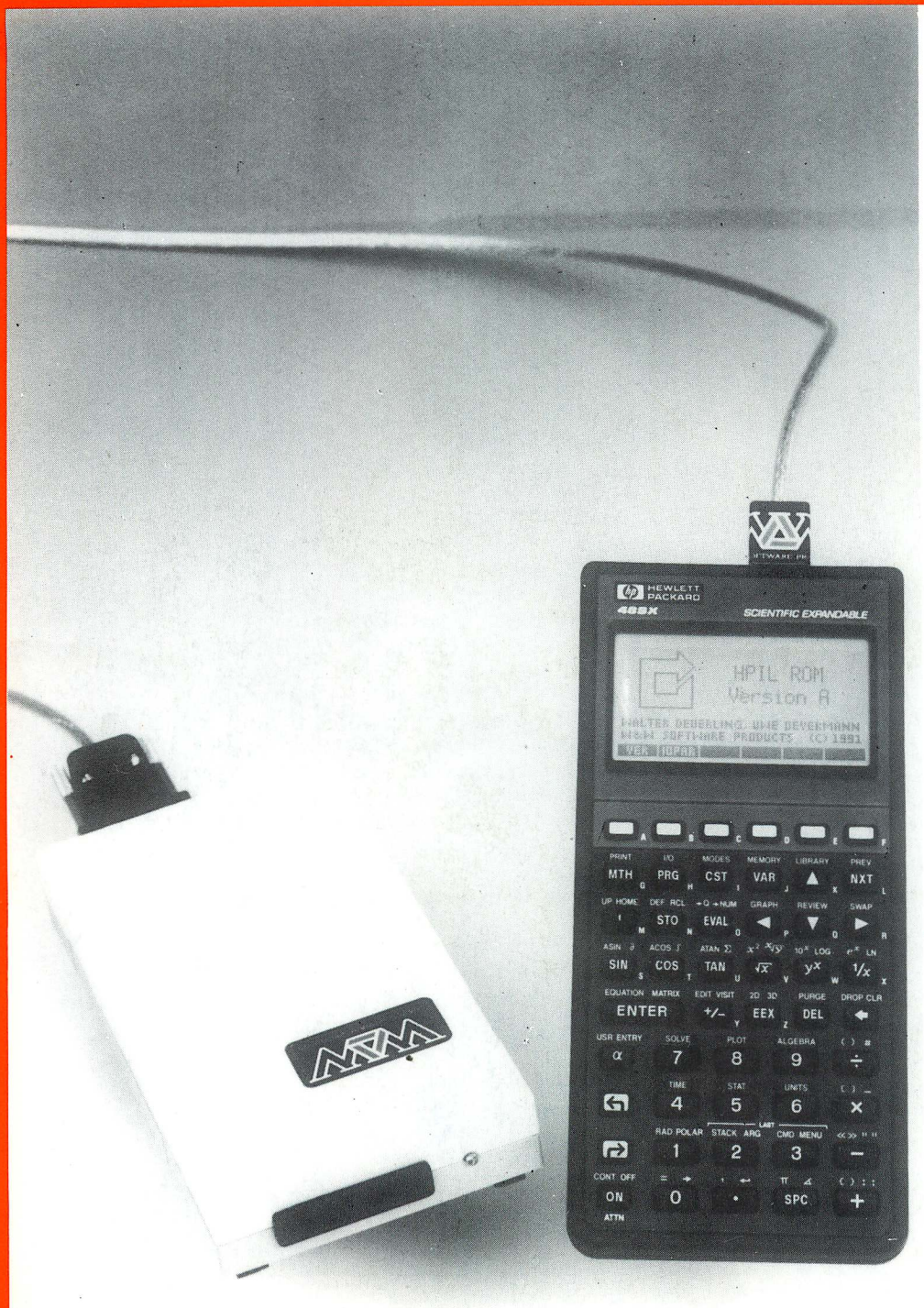
PASMA

Computerclub Deutschland e.V. • Postfach 11 04 11 • D-6000 Frankfurt am Main 1

März - Juni 1991 Nr. 2/3 D 2856 F



Das HP48-IL Interface ermöglicht die Verwendung der Peripherie von HP41/71 wie Kassettenlaufwerk oder Printer

W&W Software Products GmbH

Bei uns gibt's keine Probleme, nur **Lösungen!**

HEWLETT PACKARD COMPUTER PRODUKTE PARTNER-LÖSUNGEN

HP-41/71 RESTPOSTEN!!!

ALLGEMEINES ZUBEHÖR

Serie 10 Worksheets (50er Pack)	DM	35,-
Overlays (5er Pack)	DM	25,-
Hartlederhülle HP-Serie 10 Rechner	DM	99,-
Teakholz Tischständer	DM	110,-
Metallständer	DM	29,-
Holzständer für Aufklapprechner (HP-28 etc.)	DM	79,-
Ständer für HP-41, 21, 22, 32 u.ä. aus Acryl oder Holz	DM	49,-

SPEICHERERWEITERUNGEN FÜR DEN HP-41

Umbau von HP-41CX auf CY TURBO 64k (ab # 2549520801)	DM	599,-
Umbau von HP 41CX auf CY TURBO 32k (ab # 2549520801)	DM	499,-
Aufrüstung von 32k auf 64k	DM	136,80

MODULE FÜR DEN HP-41

HP X-Memory-Modul (82181A)	DM	150,-
HP X-Function-Modul (82180AD)	DM	150,-
HP Time-Modul (82182A)	DM	150,-
HP X-I/O-Modul (82183A)	DM	150,-
CCD-Modul A (bis # 2549520800 u. HP-41 CY)	DM	199,-
CCD-Modul B (ab # 2549520801)	DM	259,-
Grafik-ROM (16k-Modul für Geschäfts- u. Funktionsgrafiken)	DM	499,-

HP-41 ZUBEHÖR

Metall-Klappständer zur Montage unter dem HP-41	DM	35,-
Holzständer für HP-41	DM	35,-
Weiche HP-41 Echtlederhülle (schwarz)	DM	50,-
Hartlederhülle für HP-41 (offene Ausführung)	DM	99,-
HP-41 Overlays (Tastaturschablonen, schwarz, 10er Pack)	DM	25,-
HP-41 Staubschutzoverlay	DM	10,-
Zigarettenanzünderadapter für HP-Geräte	DM	59,-

SPEICHERERWEITERUNGEN FÜR DEN HP-71

CMT-71-64R (64kByte Speichererweiterungsmodul)	DM	730,-

SOFTWAREMODULE FÜR DEN HP-71

NAVSET 710 (Navigationsmodul)	DM	459,-
C71 Taschenrechnermodul (emuliert HP-11C, 12C und 16C)	DM	310,-
INVENT71 (Inventursoftware)	DM	1.500,-

EPROM-MODULE FÜR DEN HP-71

CMT-71-32KE (32kByte EPROM-Modul)	DM	299,-
CMT-71-64KE (64kByte EPROM-Modul)	DM	449,-

I/O-GERÄTE

SERIALPLUS II (Gehäuse mit 32k RAM und Zusatzdisplay)	DM	1.499,-
SERIALPLUS II (wie oben, aber incl. HP-71 und IL-Modul)	DM	2.199,-
SERIALPLUS II (Gehäuse mit 128k RAM und Zusatzdisplay)	DM	1.999,-
SERIALPLUS II (wie oben, aber incl. HP-71 und IL-Modul)	DM	2.999,-

SONSTIGES ZUBEHÖR

CMT-71 Wasserschutzgehäuse	DM	299,-
HP-71 Hartlederhülle	DM	110,-
HP-71 Holzständer	DM	55,-
HP-71 Staubschutzoverlay	DM	55,-

BÜCHER

Friedman: Control the world with HP-IL	DM	35,-
Loux/Coffif: An easy course in using the HP-28C	DM	35,-
Loux/Coffif: An easy course in using the HP-28S	DM	35,-
R. Hübner: Anwenderprogramme zum HP-28C/S	DM	25,-
Albers: HP-41 Barcodes mit dem HP-IL System	DM	25,-
Jarett: Erweiterte Funktionen des HP-41 leicht gemacht	DM	25,-
Jarett: Synthetisches Programmieren leicht gemacht	DM	20,-
Kraus: Der HP-41C/CV in Handwerk und Industrie	DM	5,-
Kruse: Optimales Peogrammieren mit dem HP-41	DM	15,-
Meschede: Plotten u. Drucken mit dem HP-41 Thermodrucker	DM	15,-
Prankel: Grafik mit dem HP-41	DM	15,-
Saul: The Chemestry Collection - Seven HP-41 Programs	DM	28,-
Synthetic Quick Reference Card	DM	2,-
Synthetic Quick Reference Guide	DM	10,-
Wickes: Synthetisches Programmieren auf dem HP-41	DM	15,-
Horn: HP-71-BASIC made easy	DM	18,-

LIEFERUNG SO LANGE VORRAT REICHT!

NEU HP-95LX auf Anfrage! **NEU**

HOTLINE: 02202/42021

W&W Software Products GmbH, Odenthaler Str. 214, 5060 Bergisch Gladbach 2, FAX: 32794

Inhalt

Die Redaktion zieht um

Da wir wegen einer bevorstehenden Gebäudesanierung unsere bisherigen Redaktionsräume in der Schwalbacher Straße im Frankfurter Gallusviertel verlassen müssen, starteten wir bereits vor vielen Wochen die intensive Suche nach neuen Räumlichkeiten. Diese Suche ist nun endlich erfolgreich abgeschlossen worden, und die PRISMA-Redaktion wird demnächst in ihre neuen Räume in den Frankfurter Stadtteil Rödelheim umziehen. Die genaue Anschrift wird nach dem erfolgten Umzug in PRISMA veröffentlicht. Bis dahin - und auch noch ein bischen länger - bitte ausschließlich an unser Postfach 11 04 11 schreiben.

Die hier vorliegende starke Doppelnummer ist also das letzte PRISMA, das in der alten Redaktion fertiggestellt wurde. Eine Doppelnummer deswegen, weil uns bei einer solchen Heftstärke die Veröffentlichung von Mammutartikeln wie SYSEVAL-Adressen vom HP48SX (beginnend auf Seite 26) besser angebracht erscheint und weil wir dadurch außerdem eine Vielzahl der eingesandten Artikeln fertig aufbereiten konnten, bevor wir in die allgemeine Umzugshektik übergehen werden.

Nützlicher Nebeneffekt: Wir kommen bei der PRISMA-Herstellung im zweiten Halbjahr wieder halbwegs in einen normalen Zeitablauf. Die Halbjahresbilanz mit über 130 redaktionellen Seiten in PRISMA kann sich trotz aller Verzögerung sehen lassen!

HP 95 LX

ein neuer Taschencomputer

Auf der diesjährigen COMDEX Spring '91 in Atlanta/USA wurde von Hewlett Packard der neue HP 95 LX vorgestellt und ich konnte ihn dort bereits persönlich in Augenschein nehmen. Der HP 95 LX ist ein Taschencomputer auf Basis eines Intel 8088-kompatiblen Prozessors mit dem Betriebssystem MS-DOS. Er ist ähnlich dem ATARI Portfolio aufklappbar mit Tastatur und mehrzeiligem LCD als Bildschirm. Unter den sieben integrierten Softwarelösungen fallen besonders die Tabellenkalkulation LOTUS 1-2-3 und der finanzmathematische HP19 auf. Weitere Informationen werden im kommenden PRISMA zu lesen sein, das dann bereits in unseren neuen Redaktionsräumen zusammengestellt wurde.

Bis dahin Happy Programming
Alf-Norman Tietze
(Chefredakteur)

Einladung zur Mitgliederversammlung 1991

Der Vorstand des Computerclubs Deutschland e.V. lädt ein zur Mitgliederversammlung 1991.

Ort: Frankfurt am Main, Intercity-Restaurant im Hauptbahnhof

Zeit: Samstag, den 7. Dezember 1991, 11:00 Uhr

Tagesordnung:

1. Begrüßung durch den Vorstand
2. Feststellung der Beschlußfähigkeit und andere Formalitäten
3. Bericht des Vorstandes
4. Bericht des Beirats
5. Bericht der Kassenprüfer
6. Entlastung des Vorstandes
7. Neuwahl des Beirates
8. Haushaltsplan 1991
9. PRISMA
10. Anträge
    a) Antrag auf Satzungsänderung
    b) Antrag betreffend Mitgliederversammlung 1992
    c) sonstige Anträge
11. Verschiedenes

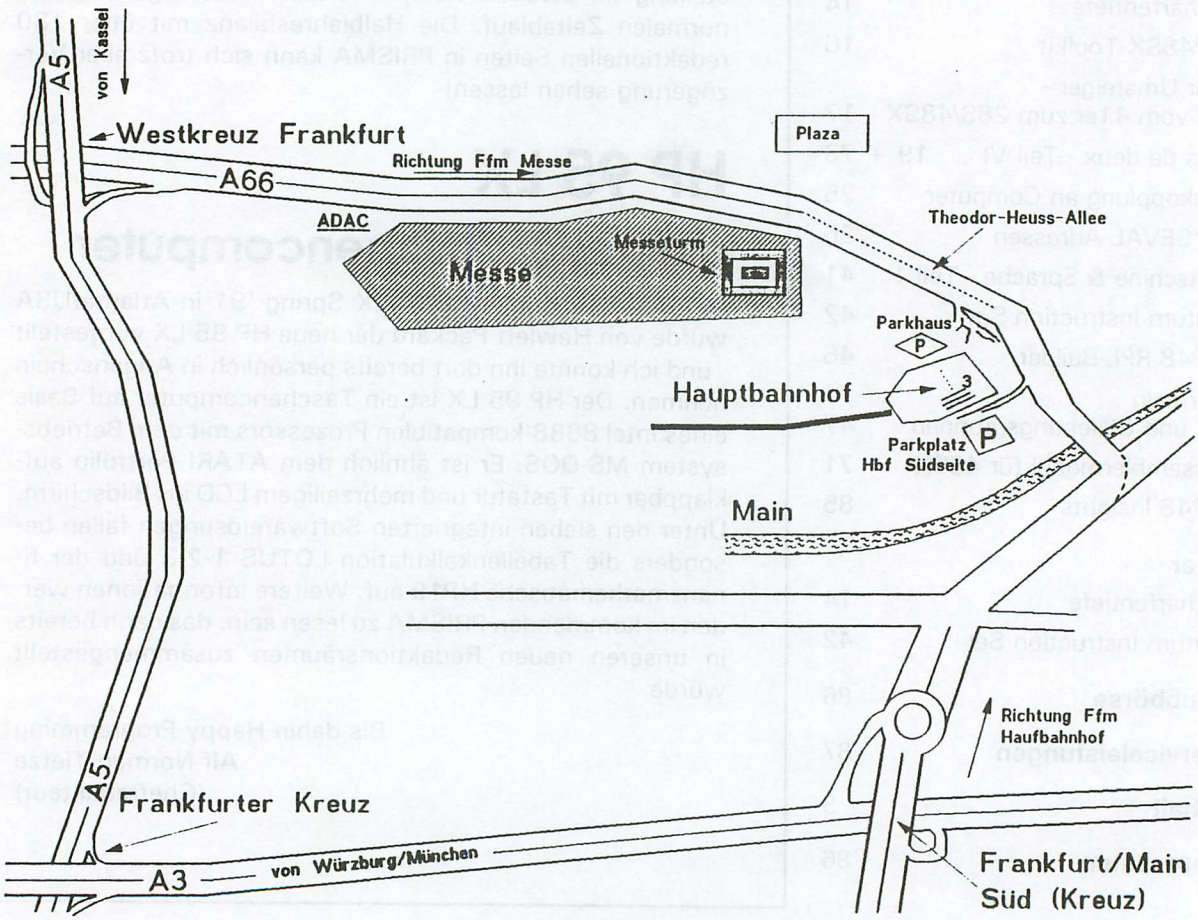
Auf Grund der Beschlußunfähigkeit der ordentlichen Mitgliederversammlung vom Samstag, dem 6. April 1991 mangels ausreichender Teilnahme ist die Mitgliederversammlung, zu welcher nunmehr eingeladen wird, gemäß § 14 Absatz 3 der Satzung des CCD e.V. ohne Rücksicht auf die Zahl der erschienenen Mitglieder beschlußfähig.

Der Vorstand wird zum Tagesordnungspunkt 10 a einen Vorschlag unterbreiten, wonach eine Mitgliederversammlung, die nach einer beschlußunfähigen Mitgliederversammlung stattfindet, am selben Tage stattfinden kann wie die beschlußunfähige Versammlung

Der Vorstand wird zum Tagesordnungspunkt 10 b vorschlagen, wegen zeitlicher Nähe zur Versammlung am 7. Dezember 1991 die Mitgliederversammlung 1992 ausfallen zu lassen.

Für den Vorstand:

Gerhard Link 1. Vorsitzender



Lotto

doch ein bißchen anders

von Dr.G.Heilmann

HP41CX, 1*XMemory, CCD-Modul, Streifendrucker, Diskettenlaufwerk HP9114B (oder HP41CX, 1*XMemory, Streifendrucker siehe Text)

Diese Ausrüstung ist zwar sehr aufwendig, dafür aber auch bequem. Grundsätzlich genügt der HP41CV, 1*XMemory und ein Streifendrucker. Dann müssen jedoch die angegebenen Programme geringfügig geändert werden. Es empfiehlt sich dann aber ein Extended-IL-Modul, mit dem der gesamte Speicherinhalt, auch mit XMemory, gesichert werden kann.

Der Grundgedanke besteht darin, in besonderer Weise die Anzahl der Ziffern zu begrenzen, die mit dem Zufallsgenerator ausgewählt werden. Zusätzlich verwirft das Programm alle solchen schon ermittelten Gruppen, die über zwei Jahre beim Samstagslotto und über ein halbes Jahr beim Mittwochslotto zu mehr als drei Treffern geführt haben.

Dazu sind für Samstagslotto in den Files Ki (i=1,2,3) die Lottozahlen von 9, 11 und 19 Wochen in Byteform aufgelistet und die Files I (I = 1, 2, 3) enthalten die Verteilung dieser Zahlen nach der Häufigkeit des Auftretens und in der zeitlichen Reihenfolge, dh. im k. Record die Zahlen, die im betrachteten Zeitraum k-mal vorgekommen sind. Das File 'E' enthält die Lottozahlen der letzten zwei Jahre für den Vergleich.

Beim Mittwochslotto sind die entsprechenden Files 'KK' und 'VV'. 'KK' ist für zehn Wochen eingerichtet. Das 'E' entsprechende File ist 'EE' und enthält 52 Sätze von Lottozahlen.

Wenn die eben beschriebenen Files vorliegen (siehe später), so werden sie von 'SL' bzw. 'ML' mit den letzten Lottozahlen auf den neuesten Stand gebracht. Die Ausdrücke der Verteilungen erleichtern dann die Wahl der Eingabe für das Zufallsprogramm.

Nun die Bedeutung im Einzelnen:

Samstag:

1. Aktualisieren der Files.

XEQ 'SL': Das Datum wird ausgedruckt und dann mit "Ni?" (i=1,...,6) zur Eingabe der aktuellen Lottozahlen aufgefordert. Quittung mit R/S. Sind sechs Zahlen eingegeben, so erfolgt der Ausdruck (siehe Muster). Bei Irrtum das Programm erneut starten. Mit R/S ergibt sich der Musterausdruck (1).

2. Auswahl der Lottozahlen.

XEQ 'ZSM': "P?" Seed für Zufallsgenerator als maximal 6-stellige ganze Zahl eingeben; R/S: VT? Im Alpha Modus 1, 2 oder 3 eingeben (Auswahl der Verteilungen, aus der ausgewählt werden soll); R/S: "N?" Anzahl der gewünschten Gruppen eingeben; R/S: "Z.X?" Z bestimmt den Record der Verteilung, Z-mal getroffen, X die Anzahl daraus zu wählender Zahlen.

Diese Aufforderung wiederholt sich, bis die Summe der X sechs beträgt. Immer verschiedene Z wählen, sonst geht der Rechner in eine endlose Schleife. Wird sechs überschritten, erscheint NONEXISTENT: das Programm neu starten.

Nach etwa 20 * N Minuten Rechenzeit Ende mit BEEP und Flag 02 gesetzt. Das Programm kann jetzt mit anderen Eingaben wiederholt werden, aber ohne den Seed zu verändern. Programmende: gegebenenfalls wieder einschalten und R/S. Der letzte Seedwert wird ausgedruckt, das System schaltet sich aus.

Mittwoch:

'ML' statt 'SL' ausführen. Mit BEEP wird zur Eingabe der B-Zahlen aufgefordert. Falls dabei ein Irrtum entstand, XEQ 'E' ausführen und weiter.

2. Auswahl der Lottozahlen wie bei Samstag, jedoch entfällt die Eingabe einer Verteilung. Flag 03 ist zu setzen.

Erzeugen der Files mit 'SMINIT':

Es sind zunächst die Filepaare 'I', 'KI' für 10, 13 und 20 Wochen und 'VV' und 'KK' für 11 Wochen zu erzeugen.

'SMINIT' mit Drucker starten, Anzahl der Wochen nach Aufforderung "WOCHEN?" eingeben und dann beginnend beim ältesten Zahlensatz wie oben bei 'SL' beschrieben vorgehen. Ist der Ausdruck fehlerhaft, dann mit XEQ 'E' wieder starten. Ausdruck nach Muster (2).

Im X-Register steht 0. Dies ist so oft durch Neustart zu wiederholen, bis die verlangte Anzahl von Wochen mit den letzten Ziehungszahlen erreicht ist. Man bestimme mit FILESIZE die Gröe von 'K' und 'VT' und speichere sie in die entsprechenden neu erzeugten Files ab.

Die Erzeugung von 'E' und 'EE' gelingt, wenn die Anzahl der Wochen negativ eingegeben wird. Ist man fertig, so erscheint 'ENDE!'. Der Inhalt von 'K' ist dann, wie eben beschrieben, in die 'E'-Files einzubringen. 'SMINIT' kann danach aus dem Aus dem Speicher entfernt werden.

Hält der Rechner irgend einmal mit der Meldung 'DUP FILE' an, dann ist über das Tastenfeld 'PURFL' auszuführen und das Programm mit R/S fortsetzen. WARNUNG! Auf keinen Fall den HP-ThinkJet als Drucker benutzen! Nach '27' folgende Zahlen werden als Escape Sequenzen interpretiert. Das führt zu unkontrollierten Störungen.

Der Verfasser berät gern und ist fast immer unter (02604)5100 zu erreichen.

Viel Glück.

```
01»LBL "ML"
02 ADV
03 CLX
04 X<>F
05 7
06 PSIZE
07 STO 02
08 2
09 STO 03
10 +
11 STO 04
12 ST/ X
13 STO 05
14 32
15 STO 06
16 SF 07
17 CF 22
18 SF 28
19 SF 29
20 FIX 6
21 CLA
22 DATE
23 ADATE
24 SF 12
25 AVIEW
26 CF 12
27 FIX 0
28 53
29 "EE"
30 SF 25
31 RCLPTA
32 FC?C 25
33 CRFLAS
34 SF 25
35 GETAS
36 GETAS
37»LBL E
38 ADV
39 SF 29
40 RCL 03
41 "H"
42 SF 25
43 CLFL
44 FC? 25
45 CRFLAS
46 1.006
47 RCL 04
48»LBL 00
49 "N"
50 ARCL Y
51 "}-?"
52 PROMPT
53 FC?C 22
54 GTO 00
55 " "
56 X<=Y?
57 ACA
58 CLA
59 ACX
60 XTOA
61 APPCHR
62 RDN
63 ISG Y
64 GTO 00
65 ADV
66 STOP
67 CLX
68 SEEKPT
69 GETREC
70 ASTO 00
71 "EE"
72 SEEKPTA
73 ASROOM
74 RCL 02
75 X>Y?
76 SF 00
77 CLX
78 SEEKPT
79 FS?C 00
80 DELREC
81 CLA
82 ARCL 00
83 APPREC
84 "EE"
85 SF 25
86 SAVEAS
87 SAVEAS
88 FC? 07
89 GTO 03
90 12
91 "K"
92 CRFLAS
93 "KK,K"
94 SF 25
95 GETAS
96 GETAS
97 RCL 03
98 -
99 "VT"
100 CRFLAS
101 "VV,VT"
102 SF 25
103 GETAS
104 GETAS
105»LBL 03
106 CF 29
107 ADV
108 CLX
109 "H"
110 SEEKPTA
111 GETREC
112 ASTO L
113 "K"
114 RCLPTA
115 CLA
116 ARCL L
117 APPREC
118 XEQ 15
119 FS? 07
120 BEEP
121 FS?C 07
122 GTO E
123 "K,KK"
124 SF 25
125 SAVEAS
126 SAVEAS
127 "VT,VV"
128 SF 25
129 SAVEAS
130 SAVEAS
131 "K"
132 PURFL
133 "VT"
134 PURFL
135 "H"
136 PURFL
137 ADV
138 ADV
139 ADV
140 ADV
141 PWRDN
142 OFF
143 RTN
144»LBL 15
145 ASTO 00
146 "VT"
147 RCLPTA
148»LBL 02
149 CLA
150 ARCL 00
151 ATOX
152 ASTO 00
153 CLA
154 X=0?
155 GTO 03
156 XTOA
157 CLX
158 SEEKPT
159 POSFL
160 RCL 05
161 DELCHR
162 FS? 04
163 CHS
164 +
165 INT
166 SEEKPT
167 APPCHR
168 GTO 02
169»LBL 03
170 "K"
171 FLSIZE
172 ASROOM
173 RCL 02
174 /
175 INT
176 RCL 05
177 +
178 -
179 "V"
180 ARCL X
181 "}-:"
182 ACA
183 PRBUF
184 SF 25
185 CLX
186 "VT"
187 SEEKPTA
188 RCL 04
189»LBL 04
190 ENTER^
191 CLX
192»LBL 05
193 GETREC
194 FC? 25
195 GTO 03
196 FC? 17
197 GTO 06
198 24
199 +
200 GTO 05
201»LBL 06
202 ALENG
203 +
204 RCL 05
205 -
206 " "
207 X<=Y?
208 ACA
209 ACX
210 RDN
211 GTO 04
212»LBL 03
213 PRBUF
214 FS? 04
215 GTO 03
216 CLX
217 "K"
218 SEEKPTA
219 GETREC
220 DELREC
221 SF 04
222 GTO 15
223»LBL 03
224 ADV
225 CF 04
226 CLX
227 "VT"
228 SEEKPTA
229 SF 25
230»LBL 07
231 GETREC
232 FC? 25
233 RTN
234 FC?C 00
235 ATOX
236 SF 01
237»LBL 08
238 RCL 04
239 ATOX
240 X=0?
241 GTO 03
242 CF 01
243 X>Y?
244 GTO 06
245 RCL 06
246 ACCHR
247 RDN
248»LBL 06
249 ACX
250 GTO 08
251»LBL 03
252 FS?C 01
253 GTO 07
254 SF 00
255 FS? 17
256 GTO 07
257 CF 00
258 "*"
259 ACA
260 PRBUF
261 GTO 07
262 END
```

✂✂✂✂✂✂✂✂

```
01»LBL "SL"
02 ADV
03 CLX
04 X<>F
05 7
06 PSIZE
07 STO 02
08 CF 22
09 SF 28
10 SF 29
11 FIX 6
12 CLA
13 DATE
14 ADATE
15 SF 12
16 AVIEW
17 CF 12
18 FIX 0
19 ADV
20 2
21 STO 03
22 STO 06
23 "H"
24 SF 25
25 CLFL
26 FC? 25
27 CRFLAS
28 1.006
29 9
30 STO 04
```

```
31»LBL 00
32 "N"
33 ARCL Y
34 "}-?"
35 PROMPT
36 FC?C 22
37 GTO 00
38 " "
39 X<=Y?
40 ACA
41 CLA
42 ACX
43 XTOA
44 APPCHR
45 RDN
46 ISG Y
47 GTO 00
48 ADV
49 ST/ X
50 STO 05
51 STOP
52 CLX
53 SEEKPT
54 GETREC
55 ASTO 00
56 105
57 "E"
58 SF 25
59 RCLPTA
60 FC?C 25
61 CRFLAS
62 SF 25
63 GETAS
64 GETAS
65 ASROOM
66 RCL 02
67 X>Y?
68 SF 00
69 CLX
70 SEEKPT
71 FS?C 00
72 DELREC
73 CLA
74 ARCL 00
75 APPREC
76 "E"
77 SF 25
78 SAVEAS
79 SAVEAS
80 3
81 STO 01
82 RCL 02
83 *
84 ST+ 06
85 "K"
86 CRFLAS
87 RCL 04
88 ST+ 06
89 -
90 "VT"
91 CRFLAS
92 CF 29
93»LBL 01
94 ADV
95 "K"
96 ARCL 01
97 "}-,K"
98 SF 25
99 GETAS
100 GETAS
101 CLA
102 ARCL 01
103 "}-,VT"
104 SF 25
105 GETAS
106 GETAS
107 CLX
108 "H"
109 SEEKPTA
110 GETREC
111 ASTO L
112 "K"
113 RCLPTA
114 CLA
115 ARCL L
116 APPREC
117 XEQ 15
118 "K,K"
119 ARCL 01
120 SF 25
121 SAVEAS
122 SAVEAS
123 "VT,"
124 ARCL 01
125 SF 25
126 SAVEAS
127 SAVEAS
128 DSE 01
129 GTO 01
130 "K"
131 PURFL
132 "VT"
133 PURFL
134 "H"
135 PURFL
136 ADV
137 ADV
138 ADV
139 ADV
140 PWRDN
141 OFF
142 RTN
143»LBL 15
144 ASTO 00
145 "VT"
146 RCLPTA
147»LBL 02
148 CLA
149 ARCL 00
150 ATOX
151 ASTO 00
152 CLA
153 X=0?
154 GTO 03
155 XTOA
156 CLX
157 SEEKPT
158 POSFL
159 RCL 05
160 DELCHR
161 FS? 04
162 CHS
163 +
164 INT
165 SEEKPT
166 APPCHR
167 GTO 02
168»LBL 03
169 "K"
170 FLSIZE
171 ASROOM
172 RCL 02
173 /
174 INT
175 RCL 05
176 +
177 -
178 "V"
179 ARCL X
180 "}-:"
181 ACA
182 PRBUF
183 SF 25
184 CLX
185 "VT"
186 SEEKPTA
187 RCL 04
188»LBL 04
189 ENTER^
190 CLX
191»LBL 05
192 GETREC
193 FC? 25
194 GTO 03
195 FC? 17
196 GTO 06
197 24
198 +
199 GTO 05
200»LBL 06
201 ALENG
202 +
203 RCL 05
204 -
205 " "
206 X<=Y?
207 ACA
208 ACX
209 RDN
210 GTO 04
211»LBL 03
212 PRBUF
213 FS? 04
214 GTO 03
215 CLX
216 "K"
217 SEEKPTA
218 GETREC
219 DELREC
220 SF 04
221 GTO 15
222»LBL 03
223 ADV
224 CF 04
225 CLX
226 "VT"
227 SEEKPTA
228 SF 25
229»LBL 07
230 GETREC
231 FC? 25
232 RTN
233 FC?C 00
234 ATOX
235 SF 01
236»LBL 08
237 RCL 04
238 ATOX
239 X=0?
240 GTO 03
241 CF 01
242 X>Y?
243 GTO 06
244 RCL 06
245 ACCHR
246 RDN
247»LBL 06
248 ACX
249 GTO 08
250»LBL 03
251 FS?C 01
252 GTO 07
253 SF 00
254 FS? 17
255 GTO 07
256 CF 00
257 "**"
258 ACA
259 PRBUF
260 GTO 07
261 END
```

✂✂✂✂✂✂✂✂

```
01»LBL"SMINIT"
02 CF 22
03 SF 28
04 FIX 0
05 ADV
06 SIZE?
07 7
08 X>Y?
09 PSIZE
10 STO 02
11 9
12 STO 03
13 13
14 STO 04
15 32
16 STO 05
17 ST/ X
18 STO 01
19»LBL 00
20 "WOCHEN?"
21 PROMPT
22 FC?C 22
23 GTO 00
24 X<0?
25 SF 08
26 +
27 "K"
28 CRFLAS
29 FS? 08
30 GTO E
31 10
32 "VT"
33 CRFLAS
34 CLA
35 94
36 XTOA
37 8
38»LBL 01
39 APPREC
40 DSE X
41 GTO 01
42 CLX
43 SEEKPT
44 1.049
45»LBL 02
46 CLA
47 XTOA
48 APPCHR
49 ISG X
50 GTO 02
51»LBL E
52 CF 03
53 SF 25
54 SF 29
55 2
56 "H"
57 CLFL
58 FC? 25
59 CRFLAS
60 1.006
61 RCL 03
```

```
62»LBL 03
63 "N"
64 ARCL Y
65 "}-?"
66 PROMPT
67 FC?C 22
68 GTO 03
69 " "
70 X<=Y?
71 ACA
72 CLA
73 ACX
74 XTOA
75 APPCHR
76 RDN
77 ISG Y
78 GTO 03
79 ADV
80 STOP
81 CF 29
82 CLX
83 SEEKPT
84 GETREC
85 ASTO L
86 "K"
87 RCLPTA
88 RCL 04
89 ASROOM
90 X>Y?
91 SF 03
92 CLA
93 ARCL L
94 APPREC
95 FS? 08
96 GTO J
97 XEQ 15
98 CF 03
99 "H"
100 PURFL
101 ADV
102 RTN
103»LBL J
104 FS? 03
105 GTO E
106 CF 08
107 "ENDE^"
108 PROMPT
109 GTO J
110»LBL 15
111 ASTO 00
112 "VT"
113 RCLPTA
114»LBL 04
115 CLA
116 ARCL 00
117 ATOX
118 ASTO 00
119 CLA
120 X=0?
121 GTO 05
122 XTOA
123 CLX
124 SEEKPT
125 POSFL
126 RCL 01
127 DELCHR
128 FS? 04
129 CHS
130 +
131 INT
132 SEEKPT
133 APPCHR
134 GTO 04
135»LBL 05
136 "K"
137 FLSIZE
138 ASROOM
139 RCL 02
140 /
141 INT
142 RCL 01
143 +
144 -
145 "V"
146 ARCL X
147 "}-:"
148 ACA
149 PRBUF
150 SF 25
151 CLX
152 "VT"
153 SEEKPTA
154 RCL 03
155»LBL 09
156 ENTER^
157 CLX
158»LBL 06
159 GETREC
160 FC? 25
161 GTO 05
162 FC? 17
163 GTO 07
164 24
165 +
166 GTO 06
167»LBL 07
168 ALENG
169 +
170 RCL 01
171 -
172 " "
173 X<=Y?
174 ACA
175 ACX
176 RDN
177 GTO 09
178»LBL 05
179 PRBUF
180 FS? 03
181 SF 04
182 FS? 04
183 GTO 05
184 CLX
185 "K"
186 SEEKPTA
187 GETREC
188 DELREC
189 SF 04
190 GTO 15
191»LBL 05
192 ADV
193 CF 04
194 CLX
195 "VT"
196 SEEKPTA
197 SF 25
198»LBL 08
199 GETREC
200 FC? 25
201 RTN
202 FC?C 00
203 ATOX
204 SF 01
205»LBL 10
206 9
207 ATOX
208 X=0?
209 GTO 05
210 CF 01
211 X>Y?
212 GTO 11
213 RCL 05
214 ACCHR
215 RDN
216»LBL 11
217 ACX
218 GTO 10
219»LBL 05
220 FS?C 01
221 GTO 08
222 SF 00
223 FS? 17
224 GTO 08
225 CF 00
226 "*"
227 ACA
228 PRBUF
229 GTO 08
230 END
```

✂✂✂✂✂✂✂✂✂

```
01»LBL "ZSM"
02 SF 25
03 20
04 "VT"
05 ASTO L
06 FS? 03
07 "VV"
08 PURFL
09 CRFLAS
10 FS? 03
11 GTO 00
12 "}-?"
13 AON
14 PROMPT
15 AOFF
16 "}-,"
17 ARCL L
18»LBL 00
19 SF 25
20 GETAS
21 GETAS
22 SF 28
23 SF 29
24 FIX 0
25 SIZE?
26 33
27 X>Y?
28 PSIZE
29 ΣREG 01
30 2
31 FS? 00
32 CLX
33 STO 00
34 6
35 ENTER^
36 ENTER^
37 ENTER^
38 STO 01
39 STO 29
40 +
41 STO 08
42 STO 28
43 +
44 STO 02
45 SIGN
46 STO 12
47 ST+ 28
48 ST+ X
49 FS? 00
50 /
51  E3
52 STO 30
53 1/X
54 ST* T
55 *
56 +
57 STO 31
58 SIGN
59 +
60 STO 32
61 "P?"
62 PROMPT
63 RCL 30
64 X^2
65 /
66 SEED
67 "N?"
68 PROMPT
69 STO 10
70»LBL 01
71 RCL 12
72 ST+ 02
73 ST+ 08
74 "Z,X?"
75 PROMPT
76 ENTER^
77 INT
78 SEEKPT
79 -
80  E1
81 *
82 STO IND 02
83 ST- 01
84 GETREC
85 ATOX
86 RCLPT
87 APPREC
88 ALENG
89 FS? 17
90 XEQ 00
91 STO IND 08
92 CLX
93 ENTER^
94 SIGN
95 X<>Y
96 X>NN?
97 /
98 X<NN?
99 GTO 01
100 SEEKPT
101»LBL 02
102 "^"
103 POSFL
104 X<0?
105 GTO 14
106 DELREC
107 GTO 02
108»LBL 00
109 RCLPT
110 INT
111 RCL Z
112 SEEKPT
113 GETREC
114 X<>Y
115 INT
116 SEEKPT
117 APPCHR
118 ALENG
119 R^
120 +
121 RTN
122»LBL 14
123 RCL 08
```

```
124 RCL 30
125 /
126 RCL 28
127 +
128 STO 09
129 FC? 01
130 PWRDN
131»LBL 33
132 RCL 09
133 STO 08
134 RCLFLAG
135 SIGN
136 STO d
137 X<> L
138 STOFLAG
139 CLX
140 STO 11
141 CLΣ
142»LBL 03
143 RCL 08
144 RCL 29
145 +
146 RCL 11
147 RCL 30
148 /
149 RCL 11
150 RCL IND Z
151 ST+ 11
152 +
153 +
154 STO 07
155»LBL 04
156 RNDM
157 FRC
158 RCL IND 08
159 *
160 INT
161 RCL 30
162 /
163 RCL 08
164 INT
165 RCL 28
166 -
167 +
168 SEEKPT
169 GETREC
170 RCL 07
171 ATOX
172»LBL 05
173 RCL IND Y
174 X=0?
175 GTO 14
176 X=Y?
177 GTO 04
178 RDN
179 DSE Y
180 GTO 05
181»LBL 14
182 RDN
183 STO IND Y
184 DSE Y
185 GTO 04
186 ISG 08
187 GTO 03
188 "E"
189 FS? 03
190 "EE"
191 RCLPTA
192 RCL 31
193 STO 25
194»LBL 06
195 RCL 12
196 CHS
197 STO 26
198 RCL IND 25
199 CLA
200 XTOA
201 ASTO 27
202»LBL 07
203 CLA
204 SF 25
205 RCL 26
206 RCL 12
207 +
208 SEEKPT
209 ARCL 27
210 POSFL
211 FS? 25
212 X<0?
213 GTO 00
214 SF 25
215 INT
216 STO 26
217 SEEKPT
218 GETREC
219 RCL 29
220 RCL 00
221»LBL 08
222 RCL IND Y
223 POSA
224 SIGN
225 +
226 DSE Y
227 GTO 08
228 X<=0?
229 GTO 07
230 GTO 14
231»LBL 00
232 DSE 25
233 GTO 06
234 CF 25
235»LBL 14
236 "VT"
237 FS? 03
238 "VV"
239 RCLPTA
240 FS? 25
241 GTO 33
242 RCL 32
243 SORT
244 CLA
245 FC? 01
246 PWRUP
247 4
248»LBL 09
249 RCL IND Y
250 ACAXY
251 ISG Y
252 GTO 09
253 PRBUF
254 FC? 01
255 PWRDN
256 DSE 10
257 GTO 33
258 "VT"
259 FS? 03
260 "VV"
261 PURFL
262 SF 21
263 RNDM
264 RCL 30
265 X^2
266 *
267 FC? 01
268 PWRUP
269 VIEW X
270 ADV
271 ADV
272 ADV
273 ADV
274 FC? 01
275 PWRDN
276 OFF
277 END
```

Beispiele

```
13.03.1991

 13. 16. 21. 22. 27. 43.

V20:
  4 13 11  5  8  6  2  0
V19:
  4 15  9  7  7  6  1  0

 28 20 35 46*
 45 12 14 39 10 29 48 44
  1  7 41  9 21 11 23*
 17 38 32  4 36 24 16 22
 27*
 18  3 49 31 30 19 26*
  8 47 42 33 40 34 25*
 37  2  6 15 13  5*
 43*

V13:
  7 21 12  4  4  1  0  0
V12:
  8 21 11  8  0  1  0  0

 45 28 20  1 12 14 46  7
*
 29 35 39 41 10 23  9 48
 31 17 25 11 44 24 32 36
 18 42 21 27 26*
  3 38 33 47  4 30 34 49
 13 16 22*
 15 19  2  6  8 37 40 43
*
  5*
```

```
V10:
 14 17 12  5  1  0  0  0
V9:
 16 17 11  5  0  0  0  0

 45 28 20  1 12 14 46  7
 24 25 18 29 32 35 17 23
*
 39 41 10  9 48 31 11 42
 44 26  3 36 13 21 27 40
 47*
 34 49 38 33 19 37  4 30
 16 22  8*
 15  2  6 43  5*



 12. 25. 26. 33. 41. 48.
 11. 17. 31. 32. 38. 48.
 16. 17. 32. 42. 44. 48.
                18,063.
```

Samstag

```
13.03.1991

  8. 11. 15. 22. 35. 39.

V11:
  7 16 15  6  4  0  1  0
V10:
  8 18 13  6  3  0  1  0
```

Muster 1

```
 40 16  3 29 30  6 18 44
*
 10 36 42  2 23 45 28  1
 37 47 41 31  7 32 35 12
 24 25*
 38 46 21  5 34 49 13  4
 17 26 14  8 43*
 48 19 27  9 11 33*
 20 22 39*
 15*


     1. 16. 23. 24. 43.
  1. 16. 19. 23. 24. 43.

V11:
  7 16 15  6  4  0  1  0
V10:
  8 19 11  7  3  0  1  0

 40  3 29 30  6 18 44 37
*
 10 36 42  2 45 28 47 41
 31  7 32 35 12 25 16  5
 13 21 38*
 46 34 49  4 17 26 14  8
  1 23 24*
 48 27  9 11 33 43 22*
 20 39 19*
 15*


 8. 10. 18. 30. 38. 41.
 7. 26. 29. 37. 38. 47.
 2.  5. 14. 29. 36. 37.
                272,213.
```

Mittwoch

```
  3.  5.  6. 15. 22. 41.
V1:
 43  6  0  0  0  0  0  0

  1  2  4  7  8  9 10 11
 12 13 14 16 17 18 19 20
 21 23 24 25 26 27 28 29
 30 31 32 33 34 35 36 37
 38 39 40 42 43 44 45 46
 47 48 49*
  3  5  6 15 22 41*

 15. 22. 30. 31. 44. 46.
V2:
 39  8  2  0  0  0  0  0

  1  2  4  7  8  9 10 11
 12 13 14 16 17 18 19 20
 21 23 24 25 26 27 28 29
 32 33 34 35 36 37 38 39
 40 42 43 45 47 48 49*
  3  5  6 41 30 31 44 46
*
 15 22*
```

Muster 2

```
  6. 16. 22. 25. 34. 39.
V10:
 12 22 10  3  1  1  0  0
V9:
 15 20 10  3  0  1  0  0

  2  9 19 21 24 26 28 36
 42 45 48 49  3  5 41*
 30 31 46 11 47  8 20 18
 35 29 32 33  4 14  7 18
 40 16 34 15*
 44 37 13 38 23 27  1 43
 39 22*
 12 17  6*
 25*
```

Dr.G.Heilmann
Obernhofer Str. 15
5408 Seelbach

ZSM


Zeile 1 von ZSM (1-4) CCD-Barcodes Dr.G.Heilmann


Zeile 2 von ZSM (5-10) CCD-Barcodes Dr.G.Heilmann


Zeile 3 von ZSM (11-17) CCD-Barcodes Dr.G.Heilmann


Zeile 4 von ZSM (18-24) CCD-Barcodes Dr.G.Heilmann


Zeile 5 von ZSM (25-31) CCD-Barcodes Dr.G.Heilmann


Zeile 6 von ZSM (32-42) CCD-Barcodes Dr.G.Heilmann


Zeile 7 von ZSM (43-51) CCD-Barcodes Dr.G.Heilmann


Zeile 8 von ZSM (52-60) CCD-Barcodes Dr.G.Heilmann


Zeile 9 von ZSM (61-67) CCD-Barcodes Dr.G.Heilmann


Zeile 10 von ZSM (68-74) CCD-Barcodes Dr.G.Heilmann


Zeile 11 von ZSM (75-83) CCD-Barcodes Dr.G.Heilmann


Zeile 12 von ZSM (84-90) CCD-Barcodes Dr.G.Heilmann


Zeile 13 von ZSM (91-98) CCD-Barcodes Dr.G.Heilmann


Zeile 14 von ZSM (99-106) CCD-Barcodes Dr.G.Heilmann


Zeile 15 von ZSM (107-113) CCD-Barcodes Dr.G.Heilmann


Zeile 16 von ZSM (114-123) CCD-Barcodes Dr.G.Heilmann


Zeile 17 von ZSM (124-131) CCD-Barcodes Dr.G.Heilmann


Zeile 18 von ZSM (132-140) CCD-Barcodes Dr.G.Heilmann


Zeile 19 von ZSM (141-150) CCD-Barcodes Dr.G.Heilmann


Zeile 20 von ZSM (151-160) CCD-Barcodes Dr.G.Heilmann


Zeile 21 von ZSM (161-169) CCD-Barcodes Dr.G.Heilmann


Zeile 22 von ZSM (170-178) CCD-Barcodes Dr.G.Heilmann


Zeile 23 von ZSM (179-186) CCD-Barcodes Dr.G.Heilmann


Zeile 24 von ZSM (187-192) CCD-Barcodes Dr.G.Heilmann

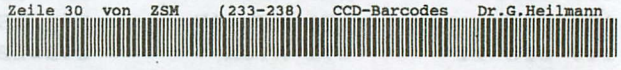

Zeile 25 von ZSM (193-200) CCD-Barcodes Dr.G.Heilmann


Zeile 26 von ZSM (201-209) CCD-Barcodes Dr.G.Heilmann


Zeile 27 von ZSM (210-216) CCD-Barcodes Dr.G.Heilmann


Zeile 28 von ZSM (217-224) CCD-Barcodes Dr.G.Heilmann


Zeile 29 von ZSM (225-232) CCD-Barcodes Dr.G.Heilmann


Zeile 30 von ZSM (233-238) CCD-Barcodes Dr.G.Heilmann


Zeile 31 von ZSM (239-245) CCD-Barcodes Dr.G.Heilmann


Zeile 32 von ZSM (246-252) CCD-Barcodes Dr.G.Heilmann


Zeile 33 von ZSM (253-258) CCD-Barcodes Dr.G.Heilmann


Zeile 34 von ZSM (259-264) CCD-Barcodes Dr.G.Heilmann


Zeile 35 von ZSM (265-273) CCD-Barcodes Dr.G.Heilmann


Zeile 36 von ZSM (274-277) CCD-Barcodes Dr.G.Heilmann

ML


Zeile 1 von ML (1-6) CCD-Barcodes Dr.G.Heilmann


Zeile 2 von ML (7-16) CCD-Barcodes Dr.G.Heilmann


Zeile 3 von ML (17-23) CCD-Barcodes Dr.G.Heilmann


Zeile 4 von ML (24-30) CCD-Barcodes Dr.G.Heilmann

Zeile 5 von ML (31-36) CCD-Barcodes Dr.G.Heilmann

Zeile 6 von ML (37-44) CCD-Barcodes Dr.G.Heilmann

Zeile 7 von ML (45-50) CCD-Barcodes Dr.G.Heilmann

Zeile 8 von ML (51-57) CCD-Barcodes Dr.G.Heilmann

Zeile 9 von ML (58-64) CCD-Barcodes Dr.G.Heilmann

Zeile 10 von ML (65-72) CCD-Barcodes Dr.G.Heilmann


Zeile 11 von ML (73-80) CCD-Barcodes Dr.G.Heilmann


Zeile 12 von ML (81-86) CCD-Barcodes Dr.G.Heilmann


Zeile 13 von ML (87-93) CCD-Barcodes Dr.G.Heilmann


Zeile 14 von ML (94-99) CCD-Barcodes Dr.G.Heilmann


Zeile 15 von ML (100-103) CCD-Barcodes Dr.G.Heilmann


Zeile 16 von ML (104-111) CCD-Barcodes Dr.G.Heilmann


Zeile 17 von ML (112-118) CCD-Barcodes Dr.G.Heilmann


Zeile 18 von ML (119-123) CCD-Barcodes Dr.G.Heilmann


Zeile 19 von ML (124-127) CCD-Barcodes Dr.G.Heilmann


Zeile 20 von ML (128-133) CCD-Barcodes Dr.G.Heilmann


Zeile 21 von ML (134-141) CCD-Barcodes Dr.G.Heilmann


Zeile 22 von ML (142-149) CCD-Barcodes Dr.G.Heilmann

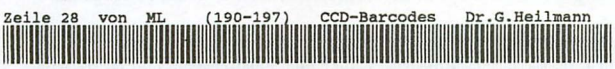

Zeile 23 von ML (150-157) CCD-Barcodes Dr.G.Heilmann


Zeile 24 von ML (158-166) CCD-Barcodes Dr.G.Heilmann

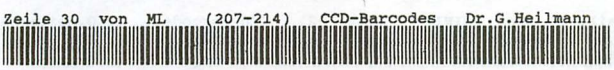

Zeile 25 von ML (167-173) CCD-Barcodes Dr.G.Heilmann


Zeile 26 von ML (174-182) CCD-Barcodes Dr.G.Heilmann


Zeile 27 von ML (183-189) CCD-Barcodes Dr.G.Heilmann


Zeile 28 von ML (190-197) CCD-Barcodes Dr.G.Heilmann


Zeile 29 von ML (198-206) CCD-Barcodes Dr.G.Heilmann


Zeile 30 von ML (207-214) CCD-Barcodes Dr.G.Heilmann


Zeile 31 von ML (215-221) CCD-Barcodes Dr.G.Heilmann


Zeile 32 von ML (222-228) CCD-Barcodes Dr.G.Heilmann


Zeile 33 von ML (229-236) CCD-Barcodes Dr.G.Heilmann

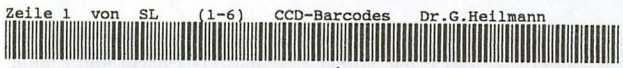

Zeile 34 von ML (237-244) CCD-Barcodes Dr.G.Heilmann


Zeile 35 von ML (245-253) CCD-Barcodes Dr.G.Heilmann


Zeile 36 von ML (254-259) CCD-Barcodes Dr.G.Heilmann


Zeile 37 von ML (260-262) CCD-Barcodes Dr.G.Heilmann

SL


Zeile 1 von SL (1-6) CCD-Barcodes Dr.G.Heilmann


Zeile 2 von SL (7-14) CCD-Barcodes Dr.G.Heilmann

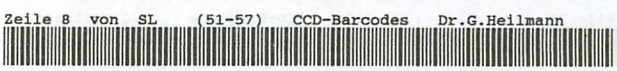

Zeile 3 von SL (15-23) CCD-Barcodes Dr.G.Heilmann

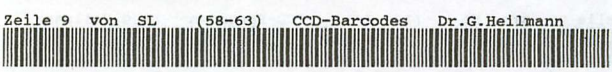

Zeile 4 von SL (24-28) CCD-Barcodes Dr.G.Heilmann


Zeile 5 von SL (29-34) CCD-Barcodes Dr.G.Heilmann


Zeile 6 von SL (35-42) CCD-Barcodes Dr.G.Heilmann


Zeile 7 von SL (43-50) CCD-Barcodes Dr.G.Heilmann


Zeile 8 von SL (51-57) CCD-Barcodes Dr.G.Heilmann


Zeile 9 von SL (58-63) CCD-Barcodes Dr.G.Heilmann


Zeile 10 von SL (64-71) CCD-Barcodes Dr.G.Heilmann


Zeile 11 von SL (72-78) CCD-Barcodes Dr.G.Heilmann


Zeile 12 von SL (79-87) CCD-Barcodes Dr.G.Heilmann


Zeile 13 von SL (88-95) CCD-Barcodes Dr.G.Heilmann

Zeile 14 von SL (96-101) CCD-Barcodes Dr.G.Heilmann

Zeile 15 von SL (102-106) CCD-Barcodes Dr.G.Heilmann

Zeile 16 von SL (107-113) CCD-Barcodes Dr.G.Heilmann

Zeile 17 von SL (114-118) CCD-Barcodes Dr.G.Heilmann

Zeile 18 von SL (119-124) CCD-Barcodes Dr.G.Heilmann

Zeile 19 von SL (125-130) CCD-Barcodes Dr.G.Heilmann

Zeile 20 von SL (131-138) CCD-Barcodes Dr.G.Heilmann

Zeile 21 von SL (139-145) CCD-Barcodes Dr.G.Heilmann

Zeile 22 von SL (146-154) CCD-Barcodes Dr.G.Heilmann

Zeile 23 von SL (155-162) CCD-Barcodes Dr.G.Heilmann

Zeile 24 von SL (163-170) CCD-Barcodes Dr.G.Heilmann

Zeile 25 von SL (171-179) CCD-Barcodes Dr.G.Heilmann

Zeile 26 von SL (180-185) CCD-Barcodes Dr.G.Heilmann

Zeile 27 von SL (186-194) CCD-Barcodes Dr.G.Heilmann

Zeile 28 von SL (195-203) CCD-Barcodes Dr.G.Heilmann

Zeile 29 von SL (204-211) CCD-Barcodes Dr.G.Heilmann

Zeile 30 von SL (212-218) CCD-Barcodes Dr.G.Heilmann

Zeile 31 von SL (219-225) CCD-Barcodes Dr.G.Heilmann

Zeile 32 von SL (226-233) CCD-Barcodes Dr.G.Heilmann

Zeile 33 von SL (234-241) CCD-Barcodes Dr.G.Heilmann

Zeile 34 von SL (242-250) CCD-Barcodes Dr.G.Heilmann

Zeile 35 von SL (251-256) CCD-Barcodes Dr.G.Heilmann

Zeile 36 von SL (257-260) CCD-Barcodes Dr.G.Heilmann

SMINIT 1. Teil

Zeile 1 von SMINIT (1-3) CCD-Barcodes Dr.G.Heilmann

Zeile 2 von SMINIT (4-12) CCD-Barcodes Dr.G.Heilmann

Zeile 3 von SMINIT (13-20) CCD-Barcodes Dr.G.Heilmann

Zeile 4 von SMINIT (21-25) CCD-Barcodes Dr.G.Heilmann

Zeile 5 von SMINIT (26-32) CCD-Barcodes Dr.G.Heilmann

Zeile 6 von SMINIT (33-39) CCD-Barcodes Dr.G.Heilmann

Zeile 7 von SMINIT (40-45) CCD-Barcodes Dr.G.Heilmann

Zeile 8 von SMINIT (46-52) CCD-Barcodes Dr.G.Heilmann

Zeile 9 von SMINIT (53-59) CCD-Barcodes Dr.G.Heilmann

Zeile 10 von SMINIT (60-65) CCD-Barcodes Dr.G.Heilmann

Zeile 11 von SMINIT (66-73) CCD-Barcodes Dr.G.Heilmann

Zeile 12 von SMINIT (74-81) CCD-Barcodes Dr.G.Heilmann

Zeile 13 von SMINIT (82-88) CCD-Barcodes Dr.G.Heilmann

Zeile 14 von SMINIT (89-96) CCD-Barcodes Dr.G.Heilmann

Zeile 15 von SMINIT (97-102) CCD-Barcodes Dr.G.Heilmann

Zeile 16 von SMINIT (103-107) CCD-Barcodes Dr.G.Heilmann

Zeile 17 von SMINIT (108-112) CCD-Barcodes Dr.G.Heilmann

Zeile 18 von SMINIT (113-121) CCD-Barcodes Dr.G.Heilmann

Zeile 19 von SMINIT (122-128) CCD-Barcodes Dr.G.Heilmann

Zeile 20 von SMINIT (129-137) CCD-Barcodes Dr.G.Heilmann

Zeile 21 von SMINIT (138-146) CCD-Barcodes Dr.G.Heilmann

Zeile 22 von SMINIT (147-152) CCD-Barcodes Dr.G.Heilmann

Zeile 23 von SMINIT (153-161) CCD-Barcodes Dr.G.Heilmann

Zeile 24 von SMINIT (162-169) CCD-Barcodes Dr.G.Heilmann

Zeile 25 von SMINIT (170-178) CCD-Barcodes Dr.G.Heilmann

Zeile 26 von SMINIT (179-185) CCD-Barcodes Dr.G.Heilmann

Zeile 27 von SMINIT (186-192) CCD-Barcodes Dr.G.Heilmann

Zeile 28 von SMINIT (193-199) CCD-Barcodes Dr.G.Heilmann

Zeile 29 von SMINIT (200-207) CCD-Barcodes Dr.G.Heilmann

Zeile 30 von SMINIT (208-216) CCD-Barcodes Dr.G.Heilmann

Fortsetzung Seite 79

Induktivität einlagiger Zylinderspulen

von Peter Jochen

HP41C, 46 Zeilen, SIZE 007

Dieses für den HF-Techniker nützliche Programm zur Dimensionierung der Induktivität einer Spule (ohne Ferritkern) fand ich in der amerikanischen Zeitschrift Electronics [1].
Es war ursprünglich für den HP25C geschrieben, wobei die geometrischen Abmessungen einer Spule (Drahtdurchmesser d, Kerndurchmesser D) in Zoll einzugeben waren. Ich habe die Formel (1) lediglich auf metrische Abmessungen umgeschrieben und das Programm für den HP41 modifiziert.

$$L = \frac{r^2 * N^2}{(9 * r + 10 * l)} \quad (1)$$

Dabei bedeutet r den Spulenradius und l die Spulenlänge in Zoll.

In der Praxis ist häufig der Spulendurchmesser D und die Drahtstärke d vorgegeben. Zur Berechnung der Induktivität L wird außerdem die Windungszahl N benötigt. Daher wird im Programm "IND" die Spulenlänge nicht mehr benötigt.
Um eine Selbstinduktion von bestimmter Größe anzufertigen, wird mit Hilfe des Programms eine Tabelle (N,L) angefertigt, um diejenige Windungszahl N zu ermitteln, die dem gesuchten Induktivitätswert am nächsten kommt. Die Windungszahl kann halbzahlig (liegender Einbau) oder ganzzahlig (stehender Einbau) sein.

Praktisch wird der vorausberechnete Wert der Induktivität nur angenähert erreicht. Denn Formel (1) gilt nur für den Fall, daß Windung neben Windung liegt.
Ferner ist zu berücksichtigen, daß eine Spule auch eine Wicklungskapazität sowie einen Widerstand besitzt. Hierdurch wird eine Spule durch ein je nach Anforderung kompliziertes R-L-C-Ersatzschaltbild modelliert.
Anders ausgedrückt: Die effektive Induktivität einer Spule ist frequenzabhängig. Tabelle 1 zeigt diese Diskrepanz zwischen der Theorie nach (1) und der Praxis.

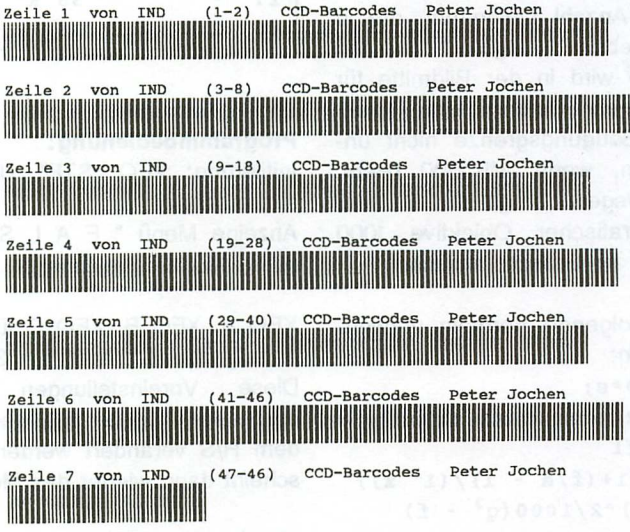
L [nHy]	f [kHz]	Rs
779	10	0,18
776	20	0,18
750	200	0,18
731	1000	0,18
715	10000	0,18

Die in **Tabelle 1** zugrunde gelegte Spule hat einen Kerndurchmesser von D = 7,5 [mm] und eine Drahtstärke von d = 0,6 [mm] CuL-Draht für eine Windungszahl von N = 12. Aus dem Programm "IND" ergibt sich hierfür eine Induktivität von: L = 0,713 µH.

Benutzerhinweise
Das Programm verlangt nach dem Starten mit R/S vom Anwender die Eingabe des Drahtdurchmessers d in [mm], sowie den Kerndurchmesser D, ebenfalls in [mm]. Dann erhält man die Induktivität L der Spule in µHenry. Bei wiederholtem Eingeben von N wird d und D so lange konstant gehalten, bis das Programm erneut gestartet wird.

Literatur
[1] Electronics, Vol. 50, No. 9, Sept. 1977, S.121.

Peter Jochen
Heilbronner Str. 240
7410 Reutlingen




```
01»LBL "IND"     13 RCL 02     25 +          37 X<>Y
02 "d/MM=?"      14 RCL 00     26 RCL 05     38 /
03 PROMPT        15 /          27 *          39 RCL 03
04 STO 01        16 2          28 STO 06     40 X^2
05 "D/MM=?"      17 /          29 10         41 *
06 PROMPT        18 STO 04     30 *          42 ARCL X
07 STO 02        19»LBL 01     31 RCL 04     43 "├ UHY"
08 RCL 01        20 "N=?"      32 9          44 PROMPT
09 25.4          21 PROMPT     33 *          45 GTO 01
10 STO 00        22 CLA        34 +          46 END
11 /             23 STO 03     35 RCL 04
12 STO 05        24 1          36 X^2
```

Schärfentiefe

für fast alle HP-Taschenrechner

von Dr. G. Heilmann

Die Kenntnis der Schärfentiefe fotografischer Objektive bei gegebener Aufnahmeentfernung (s[m]) und Blendenwahl (A) ist bei unterschiedlichen Anwendungen nützlich.

In der Regel sind die Konstruktionsmerkmale für die genaue Berechnung der kleinsten (s-[m]) und größten (s+[m]) Entfernungen für die vorgegebene Schärfentiefe gilt, nicht bekannt. Die angegebenen Formeln nach Gauß geben aber stets den Schärfetiefenbereich s+ bis s- richtig wieder. Die Werte von s- und s+ sind aber dann vor allem bei echten Teleobjektiven und Zoom-Objektiven für mit der Brennweite f vergleichbare Entfernungen nur bis auf Längen von der Größenordnung des Abstandes des Scheitels der Vorderlinse zum Scheitel der Hinterlinse des benutzten Objektives richtig.
Die erwünschte Auflösung in der Filmebene begrenzt außer dem Filmmaterial die Beugung des Lichtes an der Öffnung des Objektives: Ist l die Anzahl Linien/mm, die in der Filmebene aufgelöst werden sollen, so wird in der Bildmitte für beugungsbegrenzt korrigierte Objektive die Beugungsgrenze nicht unterschritten, wenn A*l<1500 bleibt. Hier ist wegen der großen Bildwinkel fotografischer Objektive 1000 statt 1500 als Grenze gewählt.

Es sind folgende Formeln verwendet worden:

$e = 1000*s;$

$a = (e/2)*(1+SQR(1-4f/e));$

$z = -A/fl$

$q^3 = f/(1+(f/a - 1)/(1^3 z))$

$s^3 = (q^3)^2/1000(q^3 - f)$

Die vorliegenden Programme für HP15C, HP41C/42S, HP48SX und HP71B leisten alle das Gleiche, nur fehlt dem HP15C-Programm die Meldung für die Unterschreitung der Beugungsgrenze.

HP15C

Programmbedienung:
RI: f[mm]; R0: A; R1: l[Linien/mm]
Im USER-Modus A ausführen.
s-[m] im X-Register, s+[m] im Y-Register, 0 steht für 'unendlich'.

Programm:

```
01 LBL A      19  /       37 GSB 0
02  E         20 1/X      38 RCL/2
03  3         21 RCL*I    39 X<0?
04 STO 2      22 EEX      40 CLX
05  *         23  -       41 X<>Y
06 ENTER      24 LASTX    42 GSB 0
07 1/X        25 RCL 0    43 RCL/2
08  4         26 RCL/1    44 RTN
09  *         27 RCL/I    45 LBL 0
10 RCL*I      28  +       46 EEX
11 EEX        29 EEX      47  +
12  -         30 LASTX    48 CHS
13 CHS        31  -       49 RCL*I
14 SQR        32 RDN      50 X^2
15 EEX        33  /       51 LASTX
16  +         34 X<>Y     52 RCL-I
17  *         35 R^       53  /
18  2         36  /       54 RTN
```

Anzeige: f[mm]? 50
(Eingabe oder Voreinstellung),
ENDLINE
Anzeige: Öffnung? 8
(wie vorher),
ENDLINE
Anzeige: l[Linien/mm]?:33
(wie vorher),
ENDLINE

Flag 0 wird gesetzt, löschbar mit fAUTO.

Anzeige: Entfernung[m]:
(Eingabe erforderlich)
ENDLINE
Anzeige: s- ... s+ [m]

Bei Unterschreitung der Beugungsgrenze Meldung: "unter Beugungsgrenze".
Ist Flag 0 gesetzt, dann fordert das Programm nur zur Eingabe der Entfernung auf.
Programm: *siehe nächste Seite*

HP41/42S

Programmbedienung:
Initiierung: XEQ "STF", USER-Modus erscheint;
Anzeige Menü " F A L S " (auch durch XEQ J erreichbar).

XEQ A, XEQ B, XEQ C führen zur Anzeige "F=50","A=8","L=33".
Diese Voreinstellungen können durch Neueingabe mit nachfolgendem R/S verändert werden. Es erscheint dann wieder das Menü.

Mit der eingestellten Entfernung in Metern im X-Register errechnet XEQ E die Schärfentiefe;
Anzeige: s- ... s+.
Wird die Beugungsgrenze unterschritten, dann erscheint die Meldung: "A*L > 1000".
(Das HP41-Programm mit Barcode gibt es auf gesonderten Blättern)

HP71B:

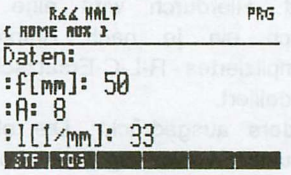
Programmbedienung:
Initiierung mit RUN STF. USER-Modus, Flag 0 ist gelöscht.

HP48SX

Programmbedienung:
STF fordert zur Eingabe auf;
Voreinstellung: f=50 mm, A=8,
L=33 Linien/mm;
der Cursor liegt auf der 8 von A,
Fortsetzung mit ENTER

Anzeige: Entfernung[m]:
Eingabe erforderlich, ENTER und Berechnung:
Anzeige: Schärfebereich[m]: s-...s+

Die Voreinstellung kann übersprungen werden, wenn Flag 1 gesetzt ist. TOG toggelt dieses Flag.



```
0010 ! STF. Schärfentiefe. Voreinstellung schützt Flag 0.
20 DESTROY P,Q,R,S,X,Y,Z @ DIM P$,Q,R$,S,X,Y,Z @ FIX 2
30 DEF KEY '#92','CFLAG 0': @ USER ON
40 IF FLAG(0) THEN GOTO 100
50 DESTROY A,F,L @ DIM A,F,L @ CFLAG 0
60 INPUT "f[mm]? :","50";F
70 INPUT "Öffnung? :","8";A
80 INPUT "l[Linien/mm]? :","33";L
90 IF A*L>1000 THEN DISP "unter Beugungsgrenze!" @ END ALL
100 SFLAG 0 @ INPUT "Entfernung[m]:";S
110 S=S*1000 @ S=S*(1+SQR(1-(4*F)/S))/2 @ Z=-A/(F*L)
120 GOSUB 150 @ P$=STR$(Q) @ GOSUB 150
130 IF Q<0 THEN R$="INF" ELSE R$=STR$(Q)
140 PRINT USING "6A,'-  ',6A,' [m]'";P$,R$ @ END ALL
150 Z=-Z @ Q=F/(1+(F/S-1)/(1+Z)) @ Q=Q^2/(1000*(Q-F)) @
    RETURN
```

```
01»LBL "STF"
02 SF 27
03 FIX 2
04 SIZE?
05 5
06 X>Y?
07 PSIZE
08 50
09 STO 00
10 8
11 STO 01
12 33
13 STO 02
14 SIGN
15 STO 03
16»LBL J
17 "F  A  L   S"
18 PROMPT
19 GTO J
20»LBL A
21 CF 22
22 RCL 00
23 "F= "
24 ARCL X
25 PROMPT
26 FS?C 22
27 STO 00
28 GTO J
29»LBL B
30 CF 22
31 RCL 01
32 "A= "
33 ARCL X
34 PROMPT
35 FS?C 22
36 STO 01
37 GTO J
38»LBL C
39 CF 22
40 RCL 02
41 "L= "
42 ARCL X
43 PROMPT
44 FS?C 22
45 STO 02
46 GTO J
47»LBL 13
48 "A*L>1000"
49 PROMPT
50 GTO 13
51»LBL E
52 FC?C 22
53 GTO J
54 CLA
55 RCL 01
56 RCL 02
57 *
58  E3
59 X<Y?
60 XEQ 13
61 RCL Z
62 *
63 RCL 00
64 X<>Y
65 ENTER^
66 1/X
67 4
68 *
69 R^
70 *
71 RCL 03
72 X<>Y
73 -
74 SQRT
75 RCL 03
76 +
77 *
78 2
79 /
80 /
81 RCL 03
82 -
83 RCL 01
84 RCL 02
85 /
86 R^
87 /
88 RCL X
89 RCL 03
90 ST+ Z
91 X<>Y
92 -
93 RDN
94 /
95 X<> Z
96 /
97 SF 04
98 XEQ 00
99 X<>Y
100»LBL 00
101 RCL 03
102 +
103 1/X
104 RCL 00
105 *
106 X^2
107 LASTX
108 RCL 00
109 -
110 /
111  E3
112 /
113 FS?C 04
114 RTN
115 ARCL X
116 "├..."
117 X<>Y
118 X<0?
119 "├INF"
120 X>0?
121 ARCL X
122 AVIEW
123 RTN
124 GTO J
125 END
```

```
      RAD HALT          PRG
{ HOME AUX }
Entfernung[m]?

1
 STF  TOG

Schärfebereich[m]:

0.92...1.09

 STF  TOG
```

```
STF
« 2 FIX
  IF 1 FC?
  THEN "Daten:" {
":f[mm]: 50
:A: 8
:l[1/mm]: 33"
{ -2 5 } V } INPUT OBJ→
'L' STO 'A' STO 'F' STO
  END
  IF A L * 1000 >
  THEN
"Beugungsgrenze
  unterschritten!"
1 DISP 1 FREEZE HALT
  END "Entfernung[m]?"
"" INPUT OBJ→ 1000 * DUP
F SWAP / 4 * 1 SWAP - √
1 + * 2 / F SWAP / 1 - F
DUP A L ROT * / 3 DUPN
NEG TF F A5 DUP
  IF 0 <
  THEN DROP "INF"
  ELSE →STR
  END 4 ROLLD TF F A5
→STR
"Schärfebereich[m]:" "
"
+ "" + "
" + SWAP +
"  " + SWAP + CLLCD 3
DISP 3 FREEZE
»

2:                  # 5572d
1:                    513.5

TF
« → z f k 'f/(1+z/(1+k))
'
»

2:                  # 16555d
1:                       66

A5
« → a f 'a^2/(1000*(a-f)
)'
»

2:                  # 54812d
1:                    69.5

TOG
« 1 DUP
  IF FS?
  THEN CF
  ELSE SF
  END
»

2:                  # 35814d
1:                       40
```

Dr.G.Heilmann
Obernhofer Str.15
5408 Seelbach

HP48SX-Toolkit

von Donelly

Für viele neu ist vielleicht die Tatsache, daß von Hewlett Packard ein Toolkit (zu Deutsch Werkzeugkasten) vertrieben wird, das die Einsatzmöglichkeiten des HP48 erheblich vergrößert.

Das Ganze ist ein Sofwarepaket, das dem Programmierer vielfältige Hilfestellungen geben kann. Als hauptsächliche Pakete werden ein Character Set Catalog, ein Menü-Label-Builder, ein Flagkatalog, ein Datenbrowser, ein Titelbrowser (Browser sind Werkzeuge zum Anschauen von Daten) und eine Toollibrary geliefert, die zusammen 74 zusätzliche Befehle für den Rechner liefert. Das Ganze ist eine 32k-Karte, läuft aber auch im RAM des Rechners ab.

Der Character Set Catalog ist eine Darstellung aller Zeichen, die der Rechner benutzen kann, d.h. die Zeichencodes von 0 bis 255. Diese werden sehr komfortabel dargestellt, nämlich in drei verschiedenen Größen. Mittels der ENTER-Taste kann dann das gewählte Zeichen in den Stack befördert werden.

Der Menü-Label-Builder ist für mich das schönste Werkzeug. Er versetzt den Anwender in die Lage, die Menüleiste individuell mit Grafiken zu gestalten, anstelle des Variablennamens oder des Programmnamens steht dann ein aussagekräftiges Bild! Inverse Darstellung ist ebenso möglich, siehe dazu die Bildbeispiele.

Natürlich wird dadurch auch nur die gewünschte Aktion ausgeführt, der Rechner wird dadurch aber einfach schöner und übersichtlicher, auch wenn nicht sehr viel Platz für die Grafiken bleibt...

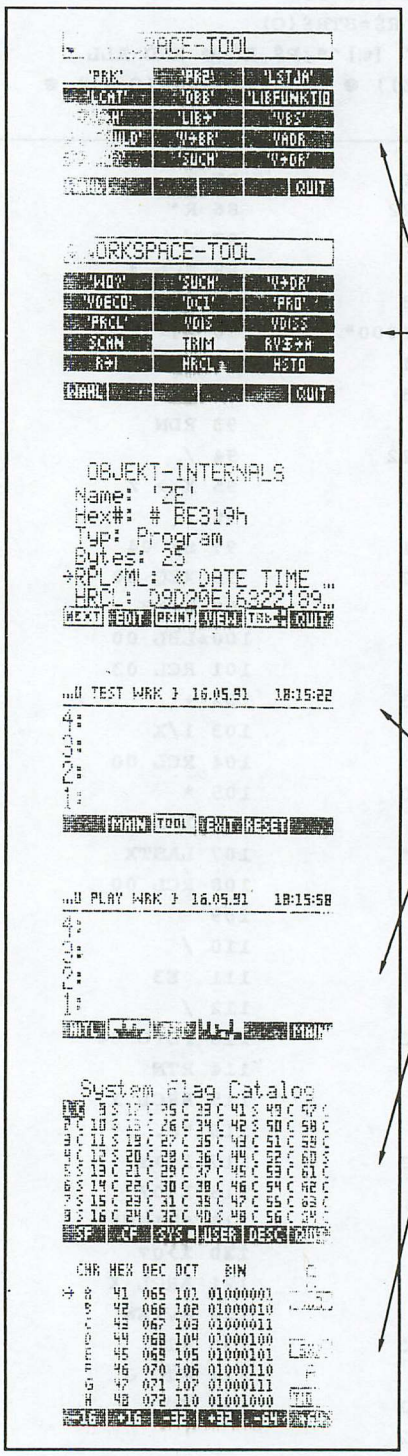
Der Flagkatalog ist eine optische Gesamtanzeige aller System- bzw. User-Flags auf einen Blick mit Anzeige ihres Zustandes. Dies ist sehr hilfreich, wenn man partout einen Fehler nicht finden kann und einen Überblick über den Status benötigt.

Gleichzeitig kann man die Userflags so definieren, daß jeweils eine Erklärung zu ihnen angezeigt werden kann, warum das betreffende Flag gesetzt ist oder nicht. Eine wirklich tolle Sache.

Der Daten und Titelbrowser gehören eigentlich zur Grundausstattung des HP48. Wer einmal damit gearbeitet hat, der kommt kaum noch ohne diese aus. Hier wird einem endlich einmal die Leistungsfähigkeit des Rechners vor Augen geführt.

Schnelles und übersichtliches Anzeigen und Editieren von auch großen Datenmengen sind endlich kein Traum mehr, sondern Wirklichkeit geworden. Man muß die beiden einfach einmal gesehen haben, um eine Ahnung der Leistungsmöglichkeiten zu bekommen. Das ist das Beste, was man für seinen treuen Freund tun kann.

In der Toollibrary sind 74 zusätzliche Befehle aufgenommen worden, die, wenn man sie erst einmal kennt, schon immer vermißt hat. Wirklich ein Hilfspaket, mit dem auf jeder Programmierstufe etwas anfangen kann, wie die beiden Browser ein abolutes Muß.



Beispiele für den Titelbrowser:
Die Pfeile im Kasten zeigen an, ob noch Folgeeinträge vorhanden sind. Das invers dargestellte Feld wird aktiviert und in den Stack geholt.
Die Menüleisten steuern den Vorgang.

Beispiel Datenbrowser:
Der Pfeil zeigt die jeweils aktivierte Zeile an, in der man mit Hilfe der Menüleiste die Werte verändern kann.

Beispiele für den Menü-Label-Builder in Verbindung mit dem Workspace:
- Hervorheben von besonderen Tasten durch inverse Darstellung
- Bilder anstelle von Befehlen
hier Schiffe versenken und Tetris für kleine Kinder in der Familie

Beispiel Flag Catalog
(hier die Systemflags)

Beispiel Character Set Catalog

Dieser Artikel ist nur eine Kurzvorstellung, da ich in meiner oben genannten Artikelserie sowohl der Assembler als auch das Toolkit eine bedeutende Rolle spielen werden und deshalb noch öfter im Gespräch sein werden.

Georg Hoppen
Hubertusring 5, 4512 Wallenhorst

Der Umsteiger

vom 41er zum 28S/48SX

von Ralf Pfeiffer

HP bietet für den 48SX die HP41CV-Emulator Karte an, damit man Programme vom 41er auch auf dem 48SX laufen lassen kann.

Viele Benutzer dieser Karte (ich habe schon mit einem gesprochen) nennen als wichtigste Nachteile:
Die Programme im 48SX laufen genauso langsam wie im 41er und ohne die Karte läuft überhaupt nichts.

Hier soll es deshalb darum gehen, wie man Programme auf dem 8er (=28S/48SX) schreibt, wenn man vorher einen 41er (41C/CV/CX) besaß, denn das neue Schreiben von Programmen stellt meiner Ansicht nach die einzige Art dar, vernünftig mit den neuen Modellen umzugehen, zumal sich der 41er fast 11 Jahre im Verkaufsprogramm gehalten hat, sollte man auch bei den Neuen langfristig denken.

Die wichtigsten *Unterschiede* zum HP41 lassen sich so beschreiben:

1. *Der Stack kann beliebig wachsen oder auch die Größe 0 haben.*
   Ruft ein 41er Programm ein anderes auf, so muß es wissen, wieviele Stackebenen durch den Unterprogrammaufruf belegt werden.
   Beim 8er kann und sollte man immer so programmieren, daß jedes Programm eine bestimmte Anzahl von Stackebenen als Eingaben verbraucht und hinterher eine bestimmte Anzahl von Stackebenen als Ergebnis hinterläßt. Ein Programm sollte sich also immer wie eine eingebaute Funktion im 8er verhalten.

2. *Programme im 8er können lokale Variable erzeugen.*
   Diese dürfen die gleichen Namen wie bereits bestehende Variable haben. Bei größerer Auswahl an gleichnamigen lokalen Variablen nimmt der 8er immer die zuletzt Erzeugte.
   Beim 41er mußte man wissen, welche Register ein Unterprogramm belegte, damit nicht Daten des Hauptprogramms überschrieben wurden. Aus diesem Grunde sollten Programme im 8er alle Eingaben und Zwischenergebnisse in lokalen und nicht in globalen Variablen speichern.

3. *Programme im 8er laufen am Anfang los und enden am Schluß.*
   Eine geregelte Programmunterbrechung in der Mitte eines Programms (im allgemeinen, weil am Ende die Unterprogramme stehen) ist beim 8er nur dann möglich, wenn der Rechner vorher von dieser Absicht in Kenntnis gesetzt wurde, z.B. durch ein HALT.
   Dabei versteckt der Rechner das unterbrochene Programm, so daß Manipulationen am Programm oder dessen Zeiger (beim 41er mit PRGM GTO.nnn o.ä.) nicht möglich sind.

   Verzweigungen innerhalb des Programms gehen nur mit Schleifenstrukturen, da in 8er-Programmen grundsätzlich keine LBL oder GTO-Befehle existieren. Wenn man also das PRPLOT-Programm vom IL-Drucker programmieren will, so muß man auf dem 8er zwei schreiben:
   Das Programm PRPLOTP für den programmierten Abruf und das Programm PRPLOT, welches nach Eingaben fragt und zum Schluß PRPLOTP als Unterprogramm aufruft.

4. *Alle Funktionen verbrauchen ihre Argumente und ersetzen sie durch Ergebnisse.*
   Beim 41er war das nur bei mathematischen Funktionen der Fall; z.B. "+" vernichtet zwei Zahlen und gibt eine zurück. Beim 8er machen das jetzt alle so, auch Vergleichsfunktionen wie "<" oder Programmfunktionen wie FOR, CF oder STO.

5. *Alle Datentypen lassen sich unter einem Namen speichern.*
   Dabei ist es egal, ob es sich dabei um Strings, Programme oder Matrizen handelt, auch die Größe spielt dabei keine Rolle.

   Es gibt keinen speziellen Datenspeicherbereich oder Programmspeicher, während man beim 41er Strings zuerst zersägen mußte, um sie in Datenspeicher zu füllen. Außerdem kann man seine Variablen geordnet ablegen (Verzeichnisse). Der 41er speichert alle Programme und Daten nebeneinander.
   Im 48SX ist es möglich, diese in Gruppen zusammenzufassen.
   Ein mathematisches Programm kann man so mit anderen gleichartigen zusammenfassen, statt es mit Datumsprogrammen zu mischen.

6. *Die Befehlszeile (COMMAND-Line) ist ein Bereich, aus dem neue Eingaben an den Stack oder Datenspeicher verteilt werden.*
   Programme kopieren die 8er nach dem Start in diesen

Bereich und führen sie aus. Unterbrochene Programme oder solche, die sich während des Ablaufs selbst löschen, existieren bis zu ihrem ordnungsgemäßen Ende in diesem Bereich weiter, auch wenn sie aus Stack und Datenspeicher für Zugriffe anderer Programme verschwanden.

7. *Softkeys* (Tasten, deren Benennung vom Programm festgelegt und verändert werden kann) *ermöglichen verbesserte Benutzerführung* (die Programmanleitung kann zuhause bleiben). Auf diese Art hat man eine Art "Maus", obwohl der Rechner kaum größer als diese ist.

Das erste Problem bei der Exploration des 8ers stellt die PRGM-Taste dar, weil sie ganz einfach fehlt.

Ein Programm ist im 8er kein besonderes Wesen mehr, welches ein Leben in der eigenen Welt (dem PRGM-Modus) fristet, sondern ein ganz normales Objekt wie z.B. eine Zahl.
Das Programm kann überall stehen, und da kommt schon der nächste Schock, auch im Stack, der nicht nur beliebig viele Ebenen haben kann, sondern auch eine geradezu unanständige Vielfalt von Objekten in beliebigen Abmessungen speichert.
Einem Programm bleiben da nur noch besondere Kennzeichen, es beginnt mit einem "«" und endet mit "»".

Hier einige Funktionen:
Schleifen laufen im 41er mit DSE und ISG und haben die Form

```
LBL 22
...
DSE 03
GTO 22
```

wobei die Schleifenparameter im Register 03 bereitstehen.

Beim 8er läuft das so:
Zuerst die Schleifenparameter (z.B. 1 und 4) bereitstellen, dann der FOR-Befehl (übernimmt die Funktion von LBL), dann eine lokale Variable (j ist der Zähler, der sonst in R03 steckt) und zum Schluß NEXT:

```
1 4 FOR j ... NEXT
```

FOR greift sich zwei Zahlen vom Stack, weiß aber nicht, was vorher passiert ist.
Wir sehen, daß 1 und 4 direkt neben FOR stehen, aber daraus sollte man nicht schließen, daß 1 und 4 zur FOR-Struktur gehören. Für FOR handelt es sich nur um zwei Zahlen, die sich gerade im Stack aufhielten. Man hätte FOR auch so füttern können:

```
0 COS 2 SQ FOR j ...
```

Auf den 8ern entfällt daher der Unterschied zwischen direkter und indirekter Schleifensteuerung!

Auch die Tests laufen etwas anders. Beim 41er testeten die Vergleichsfunktionen irgendwas, ohne eine Veränderung herbei zu führen und übersprangen ggf. den nächsten Programmschritt.

Beim 8er arbeiten Vergleiche wie z.B. "+", sie testen und verbrauchen die Objekte in Ebene 1: und 2: und hinterlassen eine 0 oder 1 als Ergebnis, weitergehende Aktionen gibt's nicht !!

Das muß jetzt nämlich eine Struktur wie IF...THEN, WHILE...REPEAT u.s.w. übernehmen. Diese besitzen gefräßige Befehle, nämlich THEN und REPEAT, die die Ebene 1: ausräumen und für ihre Entscheidung verbrauchen. Auch diesen Befehlen ist es egal, wie der Inhalt von Ebene 1: entsteht.

Beispiel:
Der Inhalt von X (Ebene 1:) und Y (Ebene 2:) soll verglichen werden.
Falls X>Y ist, dann soll die Differenz, andernfalls die Summe der beiden Zahlen gebildet werden. Auf dem 41er sieht das so aus:

```
X>Y?
CHS
+
```

auf dem 8er so:

```
IF DUP2 > THEN NEG END +
```

Der Aufruf anderer Programme erfolgt nur dadurch, daß der Name des Unterprogramms im Hauptprogramm auftaucht.
Dieser Aufruf entspricht dem XEQ-Befehl, das Unterprogramm kehrt also immer zurück, ein GTO gibt es auch hier nicht, und die Anzahl der Unterprogrammebenen ist nicht begrenzt.

Soweit die wichtigsten Unterschiede. Bei der Übernahme von Programmen mit dem 41CV-Emulator verzichtet man außerdem auf die komfortable Funktionen im 48SX wie die Menüsteuerung (es sei denn man strickt die Programme um), und ob es die synthetische Programmierung oder die Module unterstützt, sollte man seinen Händler fragen.

(Module werden natürlich nicht unterstützt, wo sollte man auch anfangen oder aufhören mit den doch im Laufe der Jahre auf dem Markt erschienenen Winzlingen; synthetische Programmierung wäre etwas schwer zu simulieren... mm)

Den 41CV-Emulator sollte man daher wohl eher als Signal an die "selten programmierenden" Anwender sehen, die den Rechner beruflich nutzen und kaum darauf verzichten können, sich andererseits aber nicht schnell umstellen können.
So gesehen ist der 41CV-Emulator ein Zeichen von Kontinuität, vor allem, wenn man sieht, daß mancher Wettbewerber neue Geräte auf den Markt wirft, aber schon beim Handbuch so versagt, daß dessen Besitz die Nutzung des Gerätes nicht unbedingt fördert.

Ralf Pfeifer
Rubensstr. 5
5000 Köln 50.

Durch einen Irrtum beim Umbruch sind die Listings auf den Seiten 73 + 74 zu finden!

Pas de deux

Teil VI

Tips & Tricks für HP28/48SX

von Ralf Pfeifer

Zunächst etwas zum Thema "Programmaufruf aus anderen Verzeichnissen" (PRISMA 91.01.32 von Ralf Schramm).

Mein Programm möchte ein Objekt (z.B. ein Programm) aus einem anderen Verzeichnis zurückrufen, z.B. das Programm DATUM im HOME PRISMA ANWEND- Verzeichnis braucht ein Formatierungsprogramm für hh,mmss Zahlen, welches im HOME EIGEN UTILS Menü steht und HHMMSS heißt.

Wie komme ich da 'ran ??
Von Hand geht es so:
Zunächst mehrfach UP ausführen (beim 48SX befindet sich der Befehl auf der Tastatur, für den 28S habe ich das Programm UP von Wickes hier abgedruckt, es ist kürzer als mein Programm UPDIR aus PRISMA 90.5-6.23) oder gleich HOME ausführen.
Dann drücke ich die NEXT (28S) oder NXT-Taste (48SX), bis die Menütaste EIGEN erscheint. Diese drücke ich dann auch noch, und suche nun mit NEXT oder NXT nach UTILS. Auch diese Taste drücke ich, und zum Schluß suche und starte ich dann mein Programm HHMMSS.

An dieser Stelle merke ich, daß ich mein Verzeichnis gewechselt habe, denn mein Programm DATUM ist nun nicht mehr aufzurufen.

Übertragen wir das nun ins Programm:
Jedes HOME, UP und NEXT/NXT können wir uns hier sparen - einfach EIGEN UTILS HHMMSS ins Programm DATUM schreiben, und das Unterprogramm läuft!
Da wir dabei das Verzeichnis wechseln, ist es für das laufende Programm egal, denn schon in Pas-de-deux 1 (PRISMA 4/90) kann man nachlesen, daß 28S und 48SX sich jedes auszuführende Programm zunächst kopieren und dann erst ausführen.

Ein Programm, welches sein eigenes Verzeichnis verläßt oder sich gar selbst löscht, läuft trotzdem bis zum Ende durch.

Was unternimmt man gegen den Wechsel des Verzeichnisses, oder wie kommt man zurück ??

Hier zunächst die Strategie für den HP28S:
Man muß sich (während man sich noch im Programm DATUM befindet) mit dem eingebauten Befehl PATH den aktuellen Verzeichnispfad holen, in unserem Beispiel legt PATH die Liste { HOME PRISMA ANWEND } in Ebene 1: des Stacks ab.
DATUM muß diese nun erst einmal in Sicherheit bringen, entweder im Stack oder in einer lokalen/globalen Variablen (wie, das ist hier egal).

Jetzt kommt der bereits beschriebene Weg:
`«...EIGEN UTILS HHMMSS...»`
ins Programm setzen.

Sobald unser DATUM Programm bereit ist, das "feindliche" Verzeichnis zu verlassen holen wir uns den gespeicherten Verzeichnispfad zurück.
Mit

```
« ... LIST→ 1 SWAP
   FOR n n ROLL EVAL
   NEXT ... »
```

kehren wir in unseren alten Verzeichnispfad (den von DATUM) zurück.

Der 48SX kann wieder einmal alles ein bißchen besser, denn bei ihm akzeptiert der RCL-Befehl nicht nur 'Namen' sondern auch Listen mit Verzeichnispfaden, in unserem Beispiel reicht es, einfach

```
{ EIGEN UTILS HHMMS }
RCL
```

zu programmieren (HOME und UPDIR können wir uns wie beim 28S sparen).

Ein EVAL hilft dann Programmen noch auf die Sprünge.
Woher weiß man sowas ?

Im Handbuch habe ich es nicht gefunden, aber gegen Aufpreis gibt es ja das "48SX Programmers Reference Manual" - Es ist wie beim Auto, die Karosserie bekommt man immer für sein Geld, nur das Lenkrad muß man manchmal extra bezahlen.

In einigen fremden Programmen habe ich schon die CASE-Struktur entdeckt, wobei aber manchmal von ökonomischem Einsatz keine Rede sein kann.
Wirklich gut kommt CASE nur, wenn zwischen THEN...END viele Programmbefehle stehen. Hat man dagegen zwischen mehreren gleichartigen Datenobjekten zu entscheiden, sollte man die Listenauswahl (s. Pas-de-deux 5 in 1/90, unter 16 ON...GOTO) vorziehen.

Beispiel:
Ein Programm bekommt die Zahlen 0...11 und soll diesen die Strings "Januar"..."Dezember" zuordnen.
Mit der CASE-Struktur sähe das dann so aus:

```
... CASE DUP 0 SAME
      THEN "JANUAR"
    END DUP 1 SAME
      ...
    END 11 SAME
```

```
    THEN "DEZEMBER"
    END
  END ...
```

Stattdessen empfehle ich folgende Listenauswahl:

```
{ "Januar" ... "Dezember" }
SWAP 1 + GET ...
```

Die Liste enthält die einzelnen Namen, unsere Eingabe läuft von 0...11, aber GET braucht Werte zwischen 1...12, weshalb ich vor GET noch schnell eine 1 addiere.

Eine andere Variante stellt das Programm OBJ→ für den 28S dar, welches den OBJ→-Befehl des 48SX simuliert. Da OBJ→ durch einzelne Befehle ersetzt werden kann, steckt man diese in eine Liste. Die Funktion TYPE prüft dann, auf welches Objekt das Programm OBJ→ einwirken soll und holt den entsprechenden Befehl aus einer Liste.

Eine kleine Hilfe für 48SX-Benutzer bietet das Programm JRPT (jährliches RPT-Intervall).
Wer sich gerne an Geburtstage freundlicher Mitmenschen oder die Todestage seiner Feinde erinnert, dem kann das TIME ALRM RPT-Menü keine Angebote machen, weil hier nur feste Wiederholungsintervalle möglich sind.
Mit JRPT geht das jetzt.

Beispiel:
W.A. Mozart wurde am 27.01.1756 geboren, und es ist uns gelungen das Programm KÖCHEL525 (wer es kann, soll es vorstellen, schließlich hat schon jemand 1981 für den 41C das Programm BACHdM für die Toccata vorgestellt) zu schreiben, welches den 48SX dazu verleitet die kleine Nachtmusik zu spielen.

Dieses vom 48SX schmerzhaft intonierte Ständchen soll uns an den Erwerb und Verzehr einiger Mozartkugeln erinnern.
Dazu wählen wir zunächst das TIME ALRM Menü, geben wie gewohnt das Datum des ersten Alarms ein (hier also 27.01.92), wählen eine beliebige Uhrzeit (in jedem Falle vor 18:30 h, weil man danach keine Schokolade mehr kriegt) und schreibt dann das Alarmprogramm

```
« JRPT KÖCHEL525
"Mozartkugeln kaufen !"
»
```

welches man mit EXEC im Alarm speichert.
Zum Abschluß noch die SET-Taste.... fertig!
(Anmerkung der Redaktion: Das Beispiel in den Listings wurde falsch übermittelt. Hier muß der Alarm ebenfalls auf 1992 gesetzt werden)

Ein Wiederholungsintervall braucht man nicht auszuwählen (es wird ohnehin gelöscht).
Wenn der Alarm fällig wird, startet er das kleine Alarmprogramm, welches zunächst zu JRPT verzweigt (JRPT entfernt die Nummer des Alarms aus Ebene 1 und setzt den Alarm für das nächste Jahr) dann KÖCHEL525 startet (für die Musik), den String "Mozartkugeln..." auf den Stack legt (nicht anzeigen lassen, der Stack speichert den String augenfällig, falls man die kleine Nachtmusik nicht hört).

JRPT funktioniert nur mit Steueralarmen und muß deshalb den Krawall der Meldealarme selbst verursachen.
Außerdem muß an der ersten Stelle des Alarmprogramms der Aufruf JRPT oder DUP JRPT (falls man die Nummer des Alarms in der Alarmliste noch braucht) stehen.
Das Programm JRPT selbst sollte im HOME Directory stehen, damit es immer vom Alarmprogramm zu finden ist.

Der Autostart hängt beim 48SX am Befehl OFF. Schaltet sich ein Programm damit aus, dann arbeitet es nach dem Einschalten *nach* dem OFF weiter.

Bei der Rechnung mit algebraischen Objekten gibt es einen interessanten Zeitfaktor:
Nehmen wir an, ein algebraisches Objekt bewohnt gerade Ebene 1: des Stacks, und wir sollen es mit einer Zahl, z.B. 3, multiplizieren, und dann durch eine weitere, z.B. 2, dividieren.
Das dauert lange, länger jedenfalls, als wenn man zunächst die Zahlen durcheinander dividiert (also 3/2 berechnet) und dann erst mit dem algebraischen Objekt multipliziert.
Zwar sind beide Operationen mathematisch gleichwertig, aber im ersten Fall muß der Rechner (egal ob 28S oder 48SX) zweimal mit dem unhandlichen algebraischen Objekt umgehen.

Wer Programme zwischen 28S und dem 48SX austauschen möchte, stellt alsbald fest, da es immer wieder kleine Inkompatibilitäten gibt. Die folgende Liste erhebt keinen Anspruch auf Vollständigkeit, und neue Funktionen die der 48SX bietet, finden hier keinen Niederschlag.

PUT und STO wirken im 28S nicht auf lokale Variablen.
Speichert man also einen Vektor oder Liste in die lokale Variable x, dann kann man das dritte Element weder mit
(neues Element) `'x(3)' STO`
noch mit `'x' 3` (neues Element) `PUT`
überschreiben.

Analog gilt das natürlich auch für Matrizen in lokalen Variablen.
Abhilfe:
Zuerst die lokale Variable auf den Stack legen (hier also x ohne ' ins Programm setzen), so daß das Objekt selbst (also Matrix, Vektor, Liste) und nicht nur sein lokaler Name bearbeitet wird.
Nach STO bzw. PUT das gesamte Objekt mit einem weiteren STO in seine lokale Variable zurückspeichern (da dieser Name nicht indiziert ist, geht das).
Merkwürdigerweise funktioniert die Umkehrung GET ganz einwandfrei.

Mit den Befehlen STO+, STO- ... SINV kann der 28S nicht auf lokale Variable einwirken.
Abhilfe:
Lokale Variable holen, die Operation (z.B. +, -, INV) im Stack ausführen und das Ergebnis mit STO zurückspeichern.

Die vom Rechner zur Verfügung gestellten Listen PAR und PPAR enthalten im 48SX jeweils ein Element mehr.

Bei PPAR steht an letzter Stelle noch die unabhängige Variable (Voreinstellung: Y), außerdem kann der 48SX statt der beiden Variablen auch eine Liste speichern, die an erster Stelle wieder die Variable (X oder Y) und dann die volle Breite der Grafik in X- bzw. Y-Richtung steht (in diesem Fall ist das LCD nur ein Fenster für die Gesamtgrafik).

Wer die Koordinaten der linken unteren Bildecke aus PPAR braucht, sollte es nicht mit

`LIST→ 7 DROPN` (48SX)
o.ä. probieren.

Sicher ist die Variante `1 GET`

Wenn der 48SX nur X oder Y holen soll, auch wenn diese in einer Liste mit dem Plotbereich liegen, verwendet

`{ } + 1 GET`

Die Liste PAR enthält beim 48SX als fünftes Element noch das Anpassungsmodell (Voreinstellung: LINFIT).
Wenn man also die X und Y-Spalte holen möchte, so nimmt man am besten

`PAR 1 GET LAST DROP 2 GET`
oder auch
`PAR LIST→ DUP2 DROPN`

dann ist das Programm zwischen 28S und 48SX austauschbar.
Da der 28S ohnehin nur die lineare Regression eingebaut hat, muß man bei der Übertragung auf den 48SX noch ein LINFIT ins Programm einbauen.

In algebraischen Objekten stellt der 48SX die MOD-Funktion zwischen die Operanden (Infix), der 28S vor diese (Präfix).

Die RND Funktion rundet beim 28S die Zahl so, wie sie angezeigt wird, im STD-Format also nie.
Der 48SX hat eine ganz neue Form der Rundung, er erwartet die Zahl oder Matrix in Ebene 2:, und in Ebene 1: eine Zahl die die Anzahl der Stellen, auf die gerundet werden soll.
Dabei entspricht 0...11 RND (48SX) dem 0...11 FIX RND (28S), -1...-11 RND (48SX) entspricht 1...11 SCI RND und 12 RND (48SX) dem einfachen RND, also der Rundung auf das aktuelle Anzeigeformat.

Die Befehle THEN, REPEAT, STEP und END (nur in DO...UNTIL) erwarten im 28S eine reelle Zahl in Ebene 1:, andernfalls gibt's tERROR !
Der 48SX erlaubt hier auch algebraische Objekte, wenn sie sich durch ein EVAL in eine reelle Zahl verwandeln.
Bei der Übersetzung für den 28S sollte man an solche algebraischen Objekte noch ein EVAL anhängen oder es in UPN umschreiben, was Zeit und Platz spart.

Die Befehle CLMF und FREEZE operieren diametral konträr, in klarem Deutsch:
Erzeugt der 28S mit DISP eine Meldung, bleibt diese auch nach Programmende in der Anzeige, wenn das Programm nicht den Befehl CLMF verwendet hat.
Der 48SX löscht dagegen immer seine Meldungen, es sei denn, FREEZE hat sie eingefroren.

Taucht in einem 28S Programm der Befehl DISP aber kein CLMF auf, so muß für den 48SX noch ein 7 FREEZE (an beliebiger Stelle) eingebaut werden.

Umgekehrt muß bei der Übertragung auf den 28S der Befehl FREEZE und die Zahl, die er in Ebene 1: erwartet, ersatzlos gestrichen werden, also kein CLMF einbauen !

HP-28S	HP-48SX
CLUSR	CLVAR
PREDV	PREDY
'PAR' 1 ROT PUT	XCOL
'PAR' 2 ROT PUT	YCOL
LAST	LASTARG
FACT	FACT, !
ABORT	0 DOERR
RCL TYPE	VTYPE
1 DISP HALT	PROMPT
/	/, RATIO
STO LAST	DUP2 STO
DO UNTIL KEY END	0 WAIT, -1 WAIT
INV ^	XROOT
PRUSR	VARS PR1

Die Funktionen TRACE und PRMD des 28S habe ich beim 48SX noch nicht entdeckt.
Die 28S-Programme REPL, TVARS, UP und OBJ→ simulieren die gleichnamigen 48SX-Funktionen.

REPL erwartet in Ebene 3: einen String/Liste, der mit dem String/Liste aus Ebene 1: überschrieben werden soll. Das Überschreiben beginnt an der Stelle, die Ebene 2: vorgibt, dabei kann sich der String/Liste aus Ebene 3 auch verlängern.
TVARS erwartet in Ebene 1: den Objekttyp (eine Zahl, wie sie TYPE ausgibt) und gibt eine Liste aus, die alle Namen enthält, die sowohl vom gewünschten Typ als auch im aktuellen USER/VARS-Menü stehen.

Auch die Flags lassen sich oft übersetzen, dabei gibt es folgende Besonderheiten:
Der 28S regelt die Druckgeschwindigkeit mit dem Flag 52 (schnell: 52 SF), der 48SX hat dafür die DELAY-Funktion.
Der 28S kennt nur den DEG (60 CF) und RAD (60 SF) Modus, beim 48SX ist das -17 CF -18 CF (DEG) und -17 SF -18 CF (RAD).
Flags:

HP28S	HP48SX
31	-55
33	-38
34	-1
35	-2
36	-3
37...42	-5...-10
43, 44	-11, -12
45	-52
46	-64
47	-37
48	-51
49, 50	-49, -50
51	-56
53...56	-45...-48
57	-20
58	-21
59	-22
61	-23
62	-24
63	-25
64	-26

Tabellen kriegt man kaum tot - auch wenn die Rechner immer leistungsfähiger werden, gibt es viel

zu viele Probleme, denen man selbst mit Spezialprogrammen nur schwer beikommt.

Ein Problem ist das Verhalten von Wasser im sog. Zweiphasengebiet.
In vielen technischen Anwendungen existieren flüssiges Wasser und Dampf nebeneinander, z.B. in der Niederdruckturbine eines Kraftwerks.
Kennt man nun zwei der folgenden Größen, wie Druck/Temperatur, Dampfgehalt (=Verhältnis von Dampfmasse zur Masse von Dampf+fl.Wasser), Enthalpie, Dichte oder Entropie, dann kann man die übrigen mit einfachen Formeln und ein paar Tabellenwerten der sog. Grenzkurve berechnen.

Mein Programm INTER erwartet, daß der Rechner irgendwo eine Tabelle als Matrix speichert, welche die Daten der Grenzkurve enthält.
Und wie in guten alten Zeiten führt das Programm dann eine Tabelleninterpolation durch, um zu einem eingegebenen Datum (z.B. dem Druck als x-Wert) den Funktionswert (z.B. die Temperatur als y-Wert) zu ermitteln. Um eine vernünftige Genauigkeit zu erhalten, sucht INTER zum eingegebenen x-Wert den y-Wert, indem jeweils die beiden nächstgrößeren und nächstkleineren x- und y-Werte aus der Tabelle herausholt und eine kubische Interpolation durchführt.

Die Anwendung geht so:
Man nehme eine Matrix oder deren 'Namen' und speichere sie in die Statistikmatrix DAT.
In der Liste PAR legt man nun mit XCOL und YCOL (48SX) bzw. COL (28S) fest, welche Spalte die x- und welche die y-Werte enthält. Dort müssen die x-Werte entweder in auf- oder absteigender Reihenfolge sortiert sein. In Ebene 1: schreibt man den x-Wert und startet das Programm INTER.

Nach Programmende enthält Ebene 1: den gesuchten y-Wert. Bei den Listings befindet sich ein Ausschnitt aus der Dampftafel für Wasser, hier erfüllen alle Spalten die verlangte Voraussetzung, der Kehrwert des spezifischen Volumens ist die Dichte, und die Worte Wasser bzw. Dampf geben an, ob sich die Daten der Spalte auf den flüssigen oder gasförmigen Grenzzustand beziehen.

Als weitere Anwendungen für INTER bieten sich z.B. die Besselfunktionen an oder Meßreihen, die man ohnehin in DAT akkumuliert, um schon aus den Meßwerten ohne Ausgleichsrechnung schnell grobe Vorhersagen für unbekannte Werte zu erhalten.
Um in der Tabelle schnell zu suchen, verwende ich einen Bisektionsalgorithmus, d.h. ich nehme den x-Wert aus der Mitte der Tabelle und vergleiche, ob der eingegebene x-Wert größer ist. Falls ja, muß ich in der oberen Hälfte der Tabelle weitersuchen und deren mittleren Wert suchen, falls nein, muß ich in der unteren Hälfte suchen.
Alles in allem findet das Bisektionsverfahren nach ungefähr ln(n)/ln(2) Schritten den richtigen Tabellenausschnitt für die Interpolation, bei 1000 Tabellenwerten kommt man so mit etwa 10 Suchschritten aus.
Falls man einen x-Wert sucht, den die Tabelle genau kennt, so liefert das Programm auch den exakten Tablellenwert ohne Interpolation.
x-Werte, die außerhalb der Tabelle liegen, führen zu einem Error. Zum einen wächst der Fehler bei einer solchen Extrapolation schnell, zum anderen fehlt einer solchen Extrapolation manchmal der Sinn, im unserem Beispiel wären negative Drücke geradezu unphysikalisch.

Bei der Berechnung sollte man sich immer über folgendes klar sein:
Die Interpolation liefert nur einen (viel) besseren Wert als eine Daumenpeilung, aber keinen exakten Wert. Das Programm liefert häufig interpolierte Zwischenwerte mit überflüssiger 12-stelliger "Genauigkeit", obwohl die Werte in der Tabelle nur mit sechs Stellen vorliegen.
Diese Tabellen fressen eine Menge Speicherplatz, aber manchmal braucht man viele verschiedene, wenn man z.B. Kühlaggregate für verschiedene Temperaturen berechnet.
Bei z.B. Kühlschrank, Gefrierschrank oder Kühlhaus benutzt man verschiedene Kühlmittel, um jedes Gerät möglichst energiesparend zu bauen.

Hier empfehle ich, die Artikel zur Datenkompression von Georg Hoppen in PRISMA 1/91 zu lesen, und die Matrizen komprimiert zu speichern, schließlich arbeitet man in der Regel nur mit einer Matrix.

Betrachtet man nämlich Wasser, so enthalten die Tabellen nur fünfstellige Werte plus einstelligem Exponenten, so daß Datenkompression hier viel bringt.
Als Unterprogramm verwende ich noch COL?, welches herausfindet, was PAR als x- und y-Spalte definiert.
Das Ergebnis legt COL? in Ebene 2: (=x-Spalte) und 1 (=y) ab, außerdem zeigt es diese und den 'Namen', welcher in DAT steht, an, um falsche Voraussetzungen bei der Berechnung zu vermeiden. Falls PAR nicht existiert, legt es COL? mit den Voreinstellungen 1 (x-Spalte) und 2 (y-Spalte) an. Falls DAT keinen Namen enthält, zeigt das Programm 'DAT' an.
Die Basis von INTER liefert die Newtoninterpolation.
Man kann diese auch einzeln von einem Programm für eine beliebige Anzahl von Datenpunkten durchführen lassen.
Dazu erwartet das Programm NEWTON (28S: EVAL in Zeile 1 und 7 durch 1 GET ersetzen) in Ebene 2: einen Vektor, der alle x-Werte (nicht notwendigerweise sortiert) enthält, und in Ebene 1: einen Vektor mit den y-Werten in der gleichen Reihenfolge. Dann kann man NEWTON starten und erhält in Ebene 1: das Interpolationspolynom als Vektor.
Auf Position eins des Vektors befindet sich der Koeffizient der höchsten Potenz von x (also x^n), an vorletzter Stelle befindet sich der Koeffizient von x und an letzter Stelle das absolute Glied. Der Vektor bringt also das Polynom in eine Form, in der es meine Programme

in Pas-de-deux 3 (PRISMA 90.5-6.22) sofort weiterverwenden können.
Es ist möglich, daß NEWTON auf der/den ersten Position/en eine Null erzeugt; daher empfiehlt es sich, als letzten Befehl in das Programm noch den Aufruf des COL-Programms aus Pas-de-deux 3 einzufügen.
Von Wernulf Beier habe ich die Idee zum Programm TRAPEZ (28S: beide EVAL durch 1 GET ersetzen). Es berechnet die Fläche eines Polygonzuges in einem Koordinatensystem.

Beispiel:
Ein Fünfeck hat die Eckpunkte (1;1) (2;3) (2;5) (-1;4) und (0;1).
Man zeichnet sich das Fünfeck in ein Koordinatensystem, geht an dessen Rand entlang und numeriert die Eckpunkte durch. Dabei ist die Richtung des Umlaufes und die Wahl des ersten Punktes egal.
In Ebene 2: erwartet TRAPEZ einen Vektor mit den x-Koordinaten, und in Ebene 1: die y-Koordinaten.

Die Eingabe gleicht also dem Programm NEWTON. Wer die Darstellung mit den runden Klammern bevorzugt, kann diese auch in einen komplexen Vektor eingeben, z.B. [(1;1) (2;3) (2;5) (-1;4) (0;1)]. Dazu muß aber als erster Befehl in TRAPEZ noch ein C→R eingefügt werden.
Auf dem 48SX kann man so die Eckpunkte auch in Polarkoordinaten eingeben.

Auch wenn ich in den bisher veröffentlichten Programmen aus den Pas-de-deux Folgen keine echten Bugs gefunden habe, möchte ich hier nocheinmal an einigen Punkten nacharbeiten.

Zu Pas-de-deux Teil 1 (3/90)
Bei den Trigonometrieprogrammen habe ich noch eine Redundanz entdeckt, die vor allem das 28S-Programm verlängert, ohne meßbar schneller zu sein. Warum ich auf die Idee kam das Programm TRI (28S und 48SX) ausgerechnet zu den Ausgaben -1, 0 oder 1 zu bewegen, um die Anzahl der Lösungen auszugeben, wei ich auch nicht mehr so genau.
Viel logischer ist es doch, eine 0 (=keine Lösung), 1 (=genau eine Lösung) oder 2 (=zwei Lösungen, die mit der größeren Fläche wird berechnet) auszugeben.
Diese Änderung erfordert allerdings auch eine Anpassung der Programme ECK3 (28S) und ∇ (48SX).

Als ich GLN programmierte, habe ich über die Polynomprogramme in Pas-de-deux 3 noch nicht nachgedacht. GLN ordnet nämlich die Koeffizienten genau andersherum an als bei den später veröffentlichten Programmen.
Um diesen Mangel zu beheben, empfehle ich die hier abgedruckte Version von GLN. Bedienung wie bisher:
In Ebene 2: steht ein Polynom als beliebiges algebraisches Objekt und in Ebene 1: der 'Name' der Variablen (meist X). Das Ergebnis ist eine Liste in Ebene 1: (welche auch symbolische Koeffizienten enthalten kann, dann aber nicht von den Pas-de-deux 3 Programmen bearbeitet wird), wobei der Koeffizient der höchsten Potenz (x^n) an Position 1 dieser Liste steht, und der vom absoluten Glied ganz hinten.

Um die Matrixinversion zu verbessern, habe ich das von HP erdachte Programm MINV ein wenig frisiert, und es in 3/90 vorgestellt. Hier möchte ich nocheinmal den gekürzten Kern von MINV angeben. Die abgedruckte 48SX-Version empfiehlt sich allerdings nur für die A-Modelle mit dem entsprechenden Bug. Für alle "gesunden" 48er und die 28er empfehle ich die vom 28S gedruckte Version, die das Newtonverfahren appliziert, schneller und bei den von mir getesteten Beispielen immer genauer war als die 48SX-A-Version.

Zu Pas-de-deux 2 (4/90)
Hier habe ich vergessen, das Programm NTAN für den 28S abzudrucken, der ja keine CASE-Struktur kennt.
Man kann CASE jedoch immer durch eine verschachtelte IF... THEN-Struktur ersetzen, so wie ich das bei der algebraischen Variante tun mußte, weil CASE in der algebraischen Syntax auch beim 48SX nicht existiert.

Der Ersatz geht so:

```
IF "Spezialfall A eingetreten"
THEN "Programmteil A ausführen"
ELSE IF "Spezialfall B eingetreten"
  THEN "Programmteil B ausführen"
  ELSE IF "Spezialfall C eingetreten"
    THEN "Programmteil C ausführen"
    ELSE ...u.s.w...

    END
  END
END.
```

Zu Pas-de-deux 3 (5-6/90)
Im Beispiel 3 zu POLYFIT hat sich eine 1 statt einer 2 ans Ende der als Lösung angegebenen Funktion geschmuggelt, aber das ebenfalls veröffentlichte Druckerprotokoll schrieb die Wahrheit.

Für den 28S habe ich meine Programme aus PRISMA 3/88 zu POLYFIT zusammengefaßt.
Die Bedienung entspricht exakt der 48SX-Version in Pas-de-deux 3.
Die Unterprogramme INDP? konnte ich ebenfalls ein wenig kürzen.

Erfreulich verlief die Optimierung von NLIN:
Die neuen Versionen sind halb so lang und doppelt so schnell wie die alten, und das bei gleicher Bedienung wie in 5-6/90 (für die 28S und 48SX-Version). Allerdings ist der Bedarf an Datenspeicherplatz etwas größer geworden.

Das Programm →EQ enthält einige überflüssige Befehle:
OVER (Zeile 1) und das g (Zeile 2) in der lokalen Definition kann man streichen.

Dem Programm TIMER empfehle ich noch die Funktion NEWOB an das END in Zeile 3 anzuhängen, so erstellt sich der Rechner zunächst eine vollständige Kopie des Objekts in Ebene 1, Zeitmessungen laufen dann (hoffentlich) etwas genauer.
Eine kleine Verbesserung ergibt sich auch bei GM (28S: XROOT durch INV ^ ersetzen) und QM (nur 48SX) bei gleicher Bedienung.
Zum mathematischen Hintergrund von KR sollte ich vielleicht noch er-

wähnen, daß der Kehrwert der Krümmung an der Stelle x einer Funktion auch als Radius des Krümmungskreises (=Krümmungsradius) bekannt ist. Fährt man mit dem Fahrrad auf der Funktion entlang und klemmt plötzlich an dieser Stelle x der Lenker, dann sagt der Kehrwert der Krümmung, welchen Radius unsere nun (unfreiwillig) gefahrene Kreisbahn hat.

Zu Pas-de-deux 4 (1/91)
Auf der Bildseite hat sich unter Punkt 1 ein kleiner Fehler eingeschlichen, über dem großen Summenzeichen (Σ) muß (wie im Text) natürlich "n=Spaltenzahl A=Zeilenzahl B" stehen.

Beim Programm CEQN sollte man eine Variable (in Ebene 1:) wählen, welche noch nicht in der symbolischen Matrix (Ebene 2:) auftaucht.
Da ich in letzter Minute noch einen Fehler im Programm NLGS korrigiert habe, stimmt dessen Beschreibung nicht mehr ganz mit dem Listing überein. Die in 1/90 abgedruckte Version funktioniert nämlich auch auf den 48SX-A-Modellen immer einwandfrei.

Leider macht das Newton-Verfahren dennoch Kummer, weil es zwecks erfolgreicher Konvergenz gute Startwerte braucht. Hat man diese gefunden, ist die Version in PRISMA 1/91 einwandfrei und schnell, die Ableitung (Jacobimatrix) berechnet das Programm ja erst in jedem dritten Schritt neu, und in der Nähe der Nullstelle werden die Abstände zwischen zwei Newtonschritten immer kleiner, weshalb sich die Ableitung auch immer weniger verändert.
Ist man von der Nullstelle noch weit entfernt, muß die Ableitung jedesmal neu berechnet werden, da sie ja den Weg zur Nullstelle angibt, und sich noch stark ändern kann. Die hier vorgestellte Version berechnet also in jedem Schritt die Ableitung neu, (kommt also hoffentlich von weiter entfernten Punkten zur Lösung) und konvergiert quadratisch.
Im temporären Menü erscheint jetzt die Funktion F→X statt →X, mit dieser neuen Funktion kann man einen beliebigen x-Vektor (in Ebene 1) in die Funktion einsetzten und eine grobe Handpeilung der Startwerte vornehmen.

Das Beispiel zeigt die Gleichungen dreier Kugeln, deren Schnittpunkt berechnet werden soll. Gibt man hier zunächst als Startwert
[1 1 1]
ein, dann konvergieren nur die Werte für Y und Z, aber X springt, was man duch Programmunterbrechnung und Neustart mit einem komplexen Vektor beheben kann.

Dazu noch ein Tip:
Im MATRIX-WRITER kann man einen reellen Vektor/Matrix nicht in einen komplexen verwandeln. Dazu wählt man besser EDIT, denn hier genügt es, ein Element in ein komplexes zu verändern, dann konvertiert der Rechner automatisch die ganze Matrix.

So mancher fragt sich, ob es nicht auch eine Regula Falsi für nichtlineare Gleichungssysteme gibt.
Doch, aber wenn man einen Rechner hat, der explizit ableiten kann ist das Newton-Verfahren besser als die Regula Falsi, obwohl es mehr Rechenaufwand verlangt.
Schon bei einer Gleichung mit einer Variablen liegt das Problem darin, daß Newton den Wert einer Ableitung (z.B. f(xi)) verwendet, der sich genau berechnen läßt.

Die Regula Falsi ersetzt diese Ableitung durch die Näherung
$f'(x_i) \quad (f(x_i)-f(x_i-1))/(x_i-x_i-1)$
aus den letzten beiden Funktionswerten.
Aber sowohl die x als auch die f-Werte nähern sich immer mehr einander an, und unterscheiden sich in den ersten Stellen kaum noch.
Theoretisch wird die Näherung der Ableitung immer genauer, aber praktisch schlagen die Rundungsfehler immer härter zu.
Das Programm JACOBI berechnet die Jacobimatrix nach der Definition im Taschenbuch der Mathematik von Bronstein-Semendjajew. Andere schlaue Bücher halten die dazu transponierte Matrix für die Jacobimatrix (so verwendet sie auch NEWTON). Was man als Anwender davon halten soll, bleibt jedem selbst überlassen.

Hier noch etwas zum Zeitsparen:
Die aufwendigste Matrixoperation ist die INV-Funktion.
Wenn man die Inverse also nur braucht, um ein lineares Gleichungssystem zu lösen, dann beschäftigen die Schritte A INV B * den Rechner etwa dreifach länger als B A / und das bei schlechterer Genauigkeit (B=Spaltenvektor, A= quadr. Matrix).

Die eingbaute Funktion RSD erlaubt nicht nur die in den Handbüchern angegebenen Operationen mit Spaltenvektoren, sondern alle Operationen mit Matrizen, die auch mit * - durchgeführt werden könnten.
Der Sinn von RSD besteht ja darin, solche Matrizenprodukte mit höherer Genauigkeit zu berechnen.

Weitere Funktionen, die auf Matrix- und Vektorobjekte (kurz MOB und VOB) einwirken können sind:
RND rundet jedes einzelne Element in MOB's und VOB's, TRNC schneidet den Nachkommateil bei jedem Element ab, DOT bildet nicht nur das Skalarprodukt von Vektoren in VOBs, sondern auch von Matrizen.

Noch einmal zur Bedeutung von Normen (RNRM, CNRM und ABS):
Nehmen wir an wir haben eine 7x7 Matrix (7, weil das auf allen 48ern funktioniert) in der Variablen A gespeichert und möchten nun wissen, welche Art von Matrixinversion die Beste ist:
Das Programm MINV für den 28S, das Programm MINV für den 48SX oder die eingebaute Funktion ??
Zweifach angewendet müßten alle drei Wege wieder A ergeben, was sie aber wegen der Rundungsfehler nicht tun. Ziehen wir dann wieder A ab, so erhalten wir die Fehler, die sich bei der Berechnung eingeschlichen haben.
Wir führen also
`A MINV MINV A -`
mit den beiden Programmen und
`A INV INV A -`
durch, und erhalten drei Matrizen mit lauter kleinen Werten. Nun ent-

scheiden wir uns für eine der drei Normen, z.B. RNRM, und wenden diese Funktion auf alle drei Matrizen an.
Alle Normen liefern als Ergebnis entweder eine Null (nur falls alle Elemente der Matrix Null waren) oder eine positive Zahl. Je kleiner diese Zahl ist, um so kleiner ist der Rundungsfehler, und um so besser war das Verfahren.

Zu Pas-de-deux 5 (1/91)
Bei Nr. 19 haben sich einige STO Befehle verkrümmelt, bei der zweiten Version sollte

```
{ A 0 B 0 C 0 D 0 }
LIST→ DROP STO STO STO STO
```

stehen.

Das Programm KEY? (nur 48SX) habe ich hier noch einmal angegeben, diesmal nimmt es die Erstfunktionen aller Tasten und wandelt sie in die aufgedruckten Alphazeichen um.

Inzwischen hatte ich noch unangenehme Begegnungen mit der Auslöschung, dem Effekt, der schnell ein paar Stellen klaut, und die Berechnung verändert.

Dazu zwei Beispiele:
10001 - 100 = 4,999875E-3
hat von 12 möglichen Stellen nur noch sieben. Mit der dritten Binomischen Formel kann man die Gleichung umformen zu
1/(10001 + 100)= 4,99987500625E-3
bei voller Genauigkeit. Auch bei
122/121 - 1 = 8,26446281E-3
verliert man gegenüber
1/121 = 8,26446280992E-3
gleich drei Stellen Genauigkeit.

Ralf Pfeifer
Rubensstr. 5
5000 Köln 50

Ankopplung an Computer?

von Markus Wiencken

"Ist er nicht schon angekoppelt?" fragen vielleicht einige.
Ich meine ganz entschieden ***"Nein!".***

Natürlich ist es möglich, mit Hilfe des Servermodus (Kermit) dem HP Befehle zu erteilen und sich automatisch das Ergebnis zurückliefern zu lassen - immerhin!
Das ist aber alles kalter Kaffee verglichen mit dem, was möglich sein sollte!

Wenn man beim IBM-PC (bei anderen Rechnern sicherlich genauso) in den Tastaturpuffer einen Befehl schreibt, wird dieser genauso ausgeführt, als ob er eingetippt worden wäre.
Wie der Inhalt in den Puffer kam, interessiert nicht, Hauptsache, er kommt da rein!

In den letzten Heften der "PRISMA" wurde immer wieder über interessante Details des HP48SX berichtet.

Ein ROM-und RAM-Layout war auch schon dabei. Die Adresse des Tastaturpuffers war auch darunter, ebenso der Bereich des Displays.

Wenn es also gelingt, sowohl in den Puffer hinein zu schreiben als auch den Speicherinhalt des Displays auszulesen, könnte man folgendes realisieren:
Über das Kabel (Sellerie-Port) werden vom Computer Befehle an den HP 48 geschickt, ebenfalls über das Kabel kommt der Inhalt der Anzeige zurück. Letzterer wird auf dem PC angezeigt. Damit könnte man den HP voll vom PC aus steuern, hätte die große Tastatur und eine große Anzeige (Monitor) zur Verfügung.

"Um Himmelswillen, ist doch alles viel zu langsam!", werden jetzt einige schreien.
"Die Übertragung von Daten ist doch einschläfernd langsam", mit Kermit versteht sich. Die Übertragung mit normalen Strings ist um etliches schneller - und bei den kleinen Übertragungsraten (9600 Baud) mehr als ausreichend sicher. Der engagierte User möge sich davon selber überzeugen.

"Die Idée ist super! Warum ist die Software noch nicht fertig?" Genau deswegen wende ich mich an die Leser der "PRISMA", um Rat einzuholen:
Wer kann mir helfen, das Problem mit dem Tastaturpuffer und dem Auslesen des Displayspeichers zu lösen?

Für sachdienliche Hinweise wäre ich dankbar, ich werde sie auch bestimmt alle veröffentlichen!

Marcus Wiencken
Riedeselstraße 64/244
6100 Darmstadt
Tel.: 06151 / 31 56 51
Fax : 06151 / 31 67 29

SYSEVAL-Adressen

für den HP48SX

00100	[DON Ofs2 Ofs1 Ofs0] Disp On, Bit Offset
00101	[Con3 Con2 Con1 Con0] Contrast
00102	[VDIG LID TRIM Con4] Disp Test (VDIG, LID, TRIM should be 0) & Contrast
00104	16 bit hardware CRC
00138	hardware timer
02248	move #100,c.a / config
028FC	Enter Machine Code (for RPL Objects)
02911	System Binary
02933	Real Number
02955	Long Real
02977	Complex Number
0299D	Long Complex
029BF	Character
029E8	Array
02A0A	Linked Array
02A2C	String
02A4E	Binary Integer
02A74	List
02A96	Directory
02AB8	Algebraic
02ADA	Unit
02AFC	Tagged
02B1E	Graphic
02B40	Library
02B62	Backup
02B88	Library Data
02D9D	Program
02DCC	Code
02E48	Global Name
02E6D	Local Name
02E92	XLIB Name
03019	MC: skip next token
0312B	End Marker
03130	RPL RETURN
0314C	Internal DEPTH → (1:System Binary)
03188	Internal DUP
031AC	Internal DUP2
031D9	Internal DUPN (N:...,1:System Binary)
03223	Internal SWAP
03244	Internal DROP
03258	Internal DROP2
0326E	Internal DROPN (1:System Binary)
03295	Internal ROT
032C2	Internal OVER
032E2	Internal PICK (N:...,1:System Binary)
03325	Internal ROLL (N:...,1:System Binary)
0339E	Internal ROLLD (N:...,1:System Binary)
03416	MC: garbage collect need 1 stack element
0341D	MC: garbage collect need C.A stack elements
0357C	MC: push A as new System Binary and continue RPL
03672	MC: restore registers, push A and continue RPL
03A81	True
03AC0	False
03AF2	Internal NOT (1:True/False)
03B06	MC: push False and continue RPL
03B1A	MC: push True and continue RPL
03B2E	if TOS-1 = TOS (object addresses are the same) → True/False
03B46	if TOS-1 = False, then DROP TOS, else DROP TOS-1
03B75	if TOS-1 = True, then DROP TOS, else DROP TOS-1
03B97	Internal SAME → True/False
03CA6	if TOS = 0 (System Binary) → True/False
03CC7	if TOS ≠ 0 (System Binary) → True/False
03CE4	if TOS-1 < TOS (System Binary) → True/False
03D19	if TOS-1 = TOS (System Binary) → True/False
03D4E	if TOS-1 ≠ TOS (System Binary) → True/False
03D83	if TOS-1 > TOS (System Binary) → True/False
03DBC	Internal + (2:System Binary,1:System Binary)
03DE0	Internal - (2:System Binary,1:System Binary)
03DEF	Internal add one (1:System Binary)
03E0E	Internal subtract one (1:System Binary)
03E2D	Internal add two (1:System Binary)
03E4E	Internal subtract two (1:System Binary)
03E6F	Internal multiply by 2 (1:System Binary)
03E8E	Internal divide by 2 (1:System Binary)
03EB1	Internal AND (2:System Binary,1:System Binary)
03EC2	Internal * (2:System Binary,1:System Binary)
03EF7	Internal / (2:System Binary,1:System Binary) → (2:rem,1:div)
03F5D	MC: pop TOS-1 and TOS (System Binary) → A.A and C.A
03F8B	push Real Number prolog <2933h>
03F95	push Complex Number prolog <2977h>
03F9F	push List prolog <2A74h>
03FA9	push Global Name prolog <2911h>
03FB3	push Program prolog <2D9Dh>
03FBD	push Algebraic prolog <2AB8h>
03FC7	push Directory prolog <2A96h>
03FD1	push Local Name prolog <2E6Dh>
03FDB	push Long Real prolog <2955h>
03FE5	push Unit prolog <2ADAh>
03FEF	<0h>
03FF9	<1h>
04003	<2h>
0400D	<3h>
04017	<4h>
04021	<5h>
0402B	<6h>
04035	<7h>
0403F	<8h>
04049	<9h>
04053	<Ah>
0405D	<Bh>
04067	<Ch>
04071	<Dh>
0407B	<Eh>
04085	<Fh>
0408F	<10h>
04099	<11h>
040A3	<12h>
040AD	<13h>
040B7	<14h>
040C1	<15h>
040CB	<16h>
040D5	<17h>
040DF	<18h>
040E9	<19h>
040F3	<1Ah>
040FD	<1Bh>
04107	<1Ch>
04111	<1Dh>
0411B	<1Eh>
04125	<1Fh>
0412F	<20h>
04139	<21h>
04143	<22h>
0414D	<23h>
04157	<24h>
04161	<25h>
0416B	<26h>

04175 <27h>
0417F <28h>
04189 <29h>
04193 <2Ah>
0419D <2Bh>
041A7 Internal OFF
04FB6 Error: Insufficient Memory
04FBB MC: Error: Insufficient Memory
0501E MC: Invoke error code in C
05023 MC: Invoke error code in A.A
05089 Internal UVAL (1:Unit)
05143 MC: Restore D,B,D1,D0 (C=D0), Clear Carry and continue RPL
05193 Internal + (2:String,1:String)
0521F Internal + (2:List,1:List)
052FA Internal + (2:List,1:Any)
05331 build composite object (N:...,2:System Binary,1:System Binary)
05445 Internal →PROGRAM (N:...,1:System Binary)
05459 Internal →LIST (N:...,1:System Binary)
0546D Internal →ALGEBRAIC (N:...,1:System Binary)
05481 Internal →UNIT (M:...,1:System Binary)
054AF explode composite object → (N:...,1:System Binary)
0554C GC & load D,B,D1,D0 (C=D0) / CC
055DF ""
055E9 { }
05622 SIZE of level 2 String → System Binary
05636 Internal SIZE (1:String) → System Binary
056B6 Internal GET; object <Nh> → N'th address in object
05944 Internal BYTES (non-ROM objects) → (2:System Binary,1:Binary Integer)
05A03 Internal B→SB (1:Binary Integer)
05B79 MC: allocate string
05BE9 name to string (1:Global Name/Local Name)
05C27 Internal R→C (2:Real Number,1:Real Number)
05D2C Internal C→R (1:Complex Number)
05E81 Deep Internal →TAG (2:Any,1:String)
05EC7 Internal OBJ→ (1:Tagged)
05EEA change prolog of list element one to Global Name
05F0F GC Need 5 Nibbles
05F2E Internal →TAG (2:Any,1:Global Name/Local Name)
05F42 RPL Garbage Collect
05F61 Internal MEM (nibbles free) → (1:System Binary)
0613E Garbage Collect & set D
06537 MC: push R0 as new System Binary
06641 MC: pop TOS (System Binary) → A.A
06657 Internal NEWOB
0670C MC: block copy
0679B MC: save D0,D1,B,D (uses C,D0), clear carry
067D2 MC: restore D,B,D1,D0 (C=D0), clear carry
06806 Let C = Space B/W RSTK & TOS
06A8E Let C.A = C.A / 5
06AD8 A=malloc(C Nibbles) ???
06AE3 A=malloc(C Nibbles)
06B00 A=malloc(B Nibbles) (C.A=free nibbles)
06E8E No Operation (continue RPL)
06E97 Place next address on the stack, do not execute
06F8E Internal EVAL (1:Any except Algebraic/List/Tagged)
06FB7 iterate loop
06FBC MC: iterate loop
06FD1 Return and execute the next token in this stream
06FD6 MC: return and execute the next token in this stream
0712A if pop TOS = True, then skip next token
0714D skip next token
07152 MC: skip next token and continue RPL
0715C skip next two tokens
07161 MC: skip next two tokens and continue RPL
0716B set return to self
07170 MC: set return to self and continue RPL
0717B MC: set return to A and continue RPL
071A2 loop
071AB exit loop
071B0 MC: exit loop
071BE MC: continue RPL
071C8 if pop TOS = False, then exit loop, else iterate loop
071CD MC: if pop TOS = False, then exit loop, else iterate loop
071E5 exit loop
071EE if pop TOS = False, then skip next two tokens and iterate
07211 MC: skip next two tokens and iterate
07221 push current loop counter as System Binary
072D7 A=orig D0, C=ptr to loop info, D0=ptr to loop counter
07334 next (internal loop)
073C3 for 0 to (TOS)-1 (1:System Binary)
073CE for 1 to (TOS)-1 (1:System Binary)
073DB for 1 to TOS (1:System Binary)
073F7 for TOS to (TOS-1)-1 (2:System Binary,1:System Binary)
074D0 store local variables (N:Any,...,1:List(of Local Names))
074E4 store local variables (M:Any,...N:Local Name,...1:System Binary)
076AE Internal DETACH from HOME directory (1:System Binary)
07709 Internal ATTACH to HOME directory (1:System Binary)
07D27 Internal STO (2:Any,1:Local Name)
08D5A recall current directory
08D92 Internal HOME
08DD4 if pop TOS = HOME directory → True/False
0CBAE Error: Nonexistent Alarm
0CBDE MC: Error: Nonexistent Alarm
0CBFA Internal TIME
0CC0E Internal DATE
0CC39 Internal DDAYS
0CC5B Internal DATE+
0CD2B Internal →DATE
0CD3F Internal CLKADJ
0CD53 Internal →TIME
0D2A3 Internal WSLOG
0D304 Internal TSTR
0DDA8 Internal ACKALL
0DDC1 Internal ACK
0DF01 'Alarms' (Global Name)
0DF28 'Alarms' (Global Name)
0E1D8 Convert internal alarm into external alarm
0E235 Recall 'Alarms' list
0E3DF Internal RCLALARM
0E402 Recall N'th Alarm (1:System Binary) → (Alarm/True,False)
0E47A 'M' (Local Name)
0E483 'N' (Local Name)
0E4A0 'M' (Local Name)
0E4AE 'N' (Local Name)
0E4C1 'M' (Local Name)
0E724 Internal DELALARM
0EAD7 Internal FINDALARM (1:Real Number)
0EB31 Internal FINDALARM (1:List)
0EB81 Internal TICKS
0F33A Internal UNIT
0F34E Internal OBJ→ (1:Unit)
0F371 Internal CONVERT
0F561 Standardize units and then strip units (levels 1,2)
0F584 Internal == (2:Real Number/Unit,1:Real Number/Unit)
0F598 Internal ≠ (2:Real Number/Unit,1:Real Number/Unit)
0F5AC Internal < (2:Real Number/Unit,1:Real Number/Unit)
0F5C0 Internal > (2:Real Number/Unit,1:Real Number/Unit)
0F5D4 Internal ≤ (2:Real Number/Unit,1:Real Number/Unit)
0F5E8 Internal ≥ (2:Real Number/Unit,1:Real Number/Unit)
0F5FC Internal ABS (1:Unit)
0F615 Internal NEG (1:Unit)
0F62E Internal SIN (1:Unit)

0F660 Internal COS (1:Unit)
0F674 Internal TAN (1:Unit)
0F6A2 Internal + (2:Real Number/Unit,1:Real Number/Unit)
0F774 Internal - (2:Real Number/Unit,1:Real Number/Unit)
0F792 Internal * (2:Real Number/Unit,1:Real Number/Unit)
0F823 Internal / (2:Real Number/Unit,1:Real Number/Unit)
0F841 Internal INV (1:Unit)
0F878 Internal ^ (2:Real Number/Unit,1:Real Number/Unit)
0F913 Internal SQ (1:Unit)
0F92C Internal sqrt (1:Unit)
0F945 Internal UBASE (1:Unit)
0FB6F Internal MAX (2:Real Number/Unit,1:Real Number/Unit)
0FB8D Internal MIN (2:Real Number/Unit,1:Real Number/Unit)
0FBAB Internal % (2:Unit,1:Real Number)
0FC3C Internal %CH (2:Real Number/Unit,1:Real Number/Unit)
0FCCD Internal %T (2:Real Number/Unit,1:Real Number/Unit)
0FCE6 Internal SIGN (1:Unit)
0FCFA Internal IP (1:Unit)
0FD0E Internal FP (1:Unit)
0FD22 Internal FLOOR (1:Unit)
0FD36 Internal CEIL (1:Unit)
0FD68 Internal RND (2:Unit,1:Real Number)
0FD8B Internal TRNC (2:Unit,1:Real Number)
10F74 Error: No Current Equation
10F79 MC: Error: No Current Equation
10FD6 Internal KILL
11036 Error: Non-Empty Directory
1103B MC: Error: Non-Empty Directory
11076 Internal CONT
112EC ?? save Last Arguments
114B3 Push @C.A as new System Binary
1158F Internal BLANK (2:System Binary,1:System Binary)
12665 recall PICT
12FB2 XFER: Save D,B,D1,D0 (uses C)
1314D Internal TEXT
13161 XFER: Restore D,B,D1,D0 (C=D0) / Clear Carry
1400E Internal ERR0
14039 Push Last Err# as System Binary
1404C Internal ERRN
14065 Internal ERRM
14088 Internal →STR (1:Any)
140AB Internal DISP (2:Any,1:Real Number)
140F1 Internal CHR (1:Real Number)
1410F Internal NUM (1:String)
14137 Internal STR→ (1:String)
1415A Internal BEEP (2:Real Number,1:Real Number)
1420A Internal > (2:String,1:String)
142A6 Internal < (2:String,1:String)
142BA Internal ≥ (2:String,1:String)
142E2 Internal ≤ (2:String,1:String)
142FB Internal FREEZE (1:Real Number)
14378 Internal HALT
1439B "halt' (Local Name)
14483 "nohalt' (Local Name)
15007 Internal DOERR (1:Real Number)
1501B Internal DOERR (1:Binary Integer)
1502F Internal DOERR (1:System Binary)
15048 Internal DOERR (1:String)
15717 Internal STEQ (1:Any)
1572B Internal RCEQ
15744 Internal RCEQ → Contents,True/False
15758 unevaluated 'EQ' (Global Name)
1576C 'EQ' (Global Name)
15781 '' (Global Name)
1592D Set last RPL token to <0h> and verify DEPTH ≥ 1
1613F Null String "" (RAM based)
166E3 Internal FIX (1:System Binary)
166EF Internal SCI (1:System Binary)
166FB Internal ENG (1:System Binary)
16707 Internal STD
16CA7 Error: Bad Argument Value
1848C Internal PATH
184E1 Internal CRDIR
18513 Internal STO (2:Any,1:Global Name)
1854F Internal PURGE (1:Global Name)
18595 Internal PGDIR (1:Global Name)
186E8 Internal TVARS (1:Real Number)
18706 Internal TVARS (1:List)
18779 Internal VARS
1884D Set last RPL Token to <0h>
18873 Internal AND (2:String,1:String)
18887 Internal OR (2:String,1:String)
1889B Internal XOR (2:String,1:String)
188AF if SIZE(TOS) = SIZE(TOS-1) (String), then NEWOB and SWAP, else Bad Argument Value
188D2 Internal NOT (1:String)
188E6 Deep internal AND (2:String,1:String)
188F5 Deep internal OR (2:String,1:String)
18904 Deep internal XOR (2:String,1:String)
18961 Deep internal NOT (1:String)
189FC Configuration code for library 002 (XLIB 2)
18A1E save last RPL token, stack size, clear @706FD.S
18A27 MC: save stack size, clear @706FD.S and continue RPL
18A51 continue RPL
18A5B Save last RPL token and verify DEPTH ≥ 3
18A68 Verify DEPTH ≥ 3
18A6D MC: verify DEPTH ≥ 3 and continue RPL
18A75 MC: verify DEPTH ≥ 3
18A80 Save last RPL token and verify DEPTH ≥ 2
18A8D Verify DEPTH ≥ 2
18A92 MC: verify DEPTH ≥ 2 and continue RPL
18A9A MC: verify DEPTH ≥ 2
18AA5 Save last RPL token and verify DEPTH ≥ 1
18AB2 Verify DEPTH ≥ 1
18AB7 MC: verify DEPTH ≥ 1 and continue RPL
18ABF MC: verify DEPTH ≥ 1
18AC6 MC: verify DEPTH ≥ C.S, expect P = 2*C.S - 1
18B6D Save last RPL token and verify DEPTH ≥ 5
18B7A Verify DEPTH ≥ 5
18B7F MC: verify DEPTH ≥ 5 and continue RPL
18B87 MC: verify DEPTH ≥ 5
18B92 Save last RPL token and verify DEPTH ≥ 4
18B9F Verify DEPTH ≥ 4
18BA4 MC: verify DEPTH ≥ 4 and continue RPL
18BAC MC: verify DEPTH ≥ 4
18C34 Save last token, check args, R→SB and verify TOS < DEPTH-1
18C4A Check args, R→SB and verify TOS < DEPTH-1
18C77 MC: save last RPL token
18C92 Error: Undefined Name
18C97 MC: Error: Undefined Name
18CA2 Error: Bad Argument Value
18CA7 MC: Error: Bad Argument Value
18CB2 Error: Bad Argument Type
18CB7 MC: Error: Bad Argument Type
18CC2 Error: Too Few Arguments
18CC7 MC: Error: Too Few Arguments
18CCE Put Error Code in C.A into A
18CD7 ABS(Real) → System Binary
18CEA Internal R→SB (1:Real Number)
18DBF Internal SB→R (1:System Binary)
18EBA Internal EVAL (1:Algebraic/List)
18ECE Save last RPL token, verify DEPTH ≥ 1 and check args.
18EDF Save last RPL token, verify DEPTH ≥ 2 and check args.
18EF0 Save last RPL token, verify DEPTH ≥ 3 and check args.

18F01 Save last RPL token, verify DEPTH ≥ 4 and check args.
18F12 Save last RPL token, verify DEPTH ≥ 5 and check args.
18F83 XFER: Bad Argument Type
18F9D Check arguments for type (don't save D0)
18FA9 Check arguments for type - XFER
18FB2 Check arguments for type
18FB7 MC: check arguments for type
194BB verify Real Array (1:Array)
1957B ASR (XLIB 2 0)
1959B RL (XLIB 2 1)
195BB RLB (XLIB 2 2)
195DB RR (XLIB 2 3)
195FB RRB (XLIB 2 4)
1961B SL (XLIB 2 5)
1963B SLB (XLIB 2 6)
1965B SR (XLIB 2 7)
1967B SRB (XLIB 2 8)
1969B R→B (XLIB 2 9)
196BB B→R (XLIB 2 10)
196DB CONVERT (XLIB 2 11)
1971B UVAL (XLIB 2 12)
1974F UNIT (XLIB 2 13)
19771 UBASE (XLIB 2 14)
197A5 UFACT (XLIB 2 15)
197C8 Internal UFACT
197F7 TIME (XLIB 2 16)
19812 DATE (XLIB 2 17)
1982D TICKS (XLIB 2 18)
19848 WSLOG (XLIB 2 19)
19863 ACKALL (XLIB 2 20)
1987E ACK (XLIB 2 21)
1989E →DATE (XLIB 2 22)
198BE →TIME (XLIB 2 23)
198DE CLKADJ (XLIB 2 24)
198FE STOALARM (XLIB 2 25)
19928 RCLALARM (XLIB 2 26)
19948 FINDALARM (XLIB 2 27)
19972 DELALARM (XLIB 2 28)
19992 TSTR (XLIB 2 29)
199B2 DDAYS (XLIB 2 30)
199D2 DATE+ (XLIB 2 31)
19A72 'ALRMDAT' (Global Name)
19B1F 'ALRMDAT' (Global Name)
19DBE 'ALRMDAT' (Global Name)
1A105 CRDIR (XLIB 2 32)
1A125 PATH (XLIB 2 33)
1A140 HOME (XLIB 2 34)
1A15B UPDIR (XLIB 2 35)
1A16F Internal UPDIR
1A194 VARS (XLIB 2 36)
1A1AF TVARS (XLIB 2 37)
1A1D9 BYTES (XLIB 2 38)
1A1FC Internal BYTES (1:Any except Global Name)
1A265 Internal BYTES (1:Global Name)
1A2BC NEWOB (XLIB 2 39)
1A2DA If TOS = ROM Object → True/False
1A303 KILL (XLIB 2 40)
1A31E OFF (XLIB 2 41)
1A339 DOERR (XLIB 2 42)
1A36D ERR0 (XLIB 2 43)
1A388 ERRN (XLIB 2 44)
1A3A3 ERRM (XLIB 2 45)
1A3BE EVAL (XLIB 2 46)
1A3FE IFTE (XLIB 2 47)
1A4A3 Internal IFTE (3:Real Number,2:Any,1:Any)
1A4CD IFT (XLIB 2 48)
1A4F0 Internal IFT (2:Real Number,1:Any)
1A513 Internal IFT (2:Symbolic,1:Any)
1A52E SYSEVAL (XLIB 2 49)
1A547 Internal SYSEVAL (1:Binary Integer)
1A584 DISP (XLIB 2 50)
1A5A4 FREEZE (XLIB 2 51)
1A5C4 BEEP (XLIB 2 52)
1A5E4 →NUM (XLIB 2 53)
1A604 LASTARG (XLIB 2 54)
1A71F WAIT (XLIB 2 55)
1A738 Internal WAIT (1:Real Number)
1A7B5 Internal WAIT (1:Real Number > 0)
1A858 CLLCD (XLIB 2 56)
1A873 KEY (XLIB 2 57)
1A8BB CONT (XLIB 2 58)
1A8D8 = (XLIB 2 59)
1A995 NEG (XLIB 2 60)
1AA1F ABS (XLIB 2 61)
1AA6E CONJ (XLIB 2 62)
1AABD pi (XLIB 2 63)
1AADF MAXR (XLIB 2 64)
1AB01 MINR (XLIB 2 65)
1AB23 e (XLIB 2 66)
1AB45 i (XLIB 2 67)
1AB67 + (XLIB 2 68)
1AC93 Internal + (2:Any,1:List)
1ACA7 Internal + (2:String,1:Any)
1ACBB Internal + (2:Any,1:String)
1ACD7 ?? + (XLIB 2 69)
1AD09 - (XLIB 2 70)
1ADEE * (XLIB 2 71)
1AF05 / (XLIB 2 72)
1B02D ^ (XLIB 2 73)
1B124 Internal ^ (2:Real Number,1:Real Number)
1B185 XROOT (XLIB 2 74)
1B1C4 ?? XROOT (XLIB 2 75)
1B278 INV (XLIB 2 76)
1B2DB ARG (XLIB 2 77)
1B30D Internal ARG (1:Real Number)
1B32A SIGN (XLIB 2 78)
1B374 sqrt (XLIB 2 79)
1B3F5 Internal sqrt (1:Real Number)
1B426 SQ (XLIB 2 80)
1B47B Internal SQ (1:Real Number)
1B48F Internal SQ (1:Complex Number)
1B4AC SIN (XLIB 2 81)
1B505 COS (XLIB 2 82)
1B55E TAN (XLIB 2 83)
1B5B7 SINH (XLIB 2 84)
1B606 COSH (XLIB 2 85)
1B655 TANH (XLIB 2 86)
1B6A4 ASIN (XLIB 2 87)
1B6EA Internal ASIN (1:Real Number)
1B72F ACOS (XLIB 2 88)
1B775 Internal ACOS (1:Real Number)
1B79C ATAN (XLIB 2 89)
1B7EB ASINH (XLIB 2 90)
1B830 ACOSH (XLIB 2 91)
1B86C Internal ACOSH (1:Real Number)
1B8A2 ATANH (XLIB 2 92)
1B8DE Internal ATANH (1:Real Number)
1B905 EXP (XLIB 2 93)
1B94F LN (XLIB 2 94)
1B995 Internal LN (1:Real Number)
1B9C6 LOG (XLIB 2 95)
1BA0C Internal LOG (1:Real Number)
1BA3D ALOG (XLIB 2 96)
1BA8C LNP1 (XLIB 2 97)
1BAC2 EXPM (XLIB 2 98)
1BB02 ! (XLIB 2 99)
1BB41 FACT (XLIB 2 100)
1BB6D IP (XLIB 2 101)
1BBA3 FP (XLIB 2 102)
1BBD9 FLOOR (XLIB 2 103)
1BC0F CEIL (XLIB 2 104)
1BC45 XPON (XLIB 2 105)
1BC71 MAX (XLIB 2 106)
1BCE3 MIN (XLIB 2 107)
1BD55 RND (XLIB 2 108)
1BDD1 TRNC (XLIB 2 109)
1BE4D MOD (XLIB 2 110)
1BE9C MANT (XLIB 2 111)
1BEC8 D→R (XLIB 2 112)
1BEF4 R→D (XLIB 2 113)
1BF1E →HMS (XLIB 2 114)
1BF3E HMS→ (XLIB 2 115)
1BF5E HMS+ (XLIB 2 116)
1BF7E HMS- (XLIB 2 117)
1BF9E RNRM (XLIB 2 118)
1BFBE CNRM (XLIB 2 119)
1BFDE DET (XLIB 2 120)
1BFFE DOT (XLIB 2 121)
1C01E CROSS (XLIB 2 122)
1C03E RSD (XLIB 2 123)
1C060 % (XLIB 2 124)
1C0D7 %T (XLIB 2 125)
1C149 %CH (XLIB 2 126)
1C1B9 RAND (XLIB 2 127)
1C1D4 RDZ (XLIB 2 128)
1C1F6 COMB (XLIB 2 129)
1C236 PERM (XLIB 2 130)
1C274 SF (XLIB 2 131)
1C28D Internal SF (1:Real Number)
1C2B0 TOS (Real) → System Binary / TOS (Real) > 0 → True/False
1C2D5 CF (XLIB 2 132)
1C2EE Internal CF (1:Real Number)
1C313 FS? (XLIB 2 133)
1C32C Internal FS? (1:Real Number)
1C331 Internal FS? (1:Real Number) → True/False
1C360 FC? (XLIB 2 134)
1C379 Internal FC? (1:Real Number)
1C399 DEG (XLIB 2 135)
1C3B4 RAD (XLIB 2 136)
1C3CF GRAD (XLIB 2 137)
1C3EA FIX (XLIB 2 138)

1C403 Internal FIX (1:Real Number)
1C41E SCI (XLIB 2 139)
1C437 Internal SCI (1:Real Number)
1C452 ENG (XLIB 2 140)
1C46B Internal ENG (1:Real Number)
1C486 STD (XLIB 2 141)
1C4A1 FS?C (XLIB 2 142)
1C4BA Internal FS?C (1:Real Number)
1C4BF Internal FS?C (1:Real Number) → True/False
1C520 FC?C (XLIB 2 143)
1C539 Internal FC?C (1:Real Number)
1C559 BIN (XLIB 2 144)
1C574 DEC (XLIB 2 145)
1C58F HEX (XLIB 2 146)
1C5AA OCT (XLIB 2 147)
1C5C5 STWS (XLIB 2 148)
1C5FE RCWS (XLIB 2 149)
1C619 RCLF (XLIB 2 150)
1C637 Internal recall System Flags
1C64E Internal recall User Flags
1C661 push Binary Integer @ D0
1C67F STOF (XLIB 2 151)
1C6A2 Internal STOF (1:List)
1C6CF Internal STOF (2:Binary Integer,1:Binary Integer)
1C6E3 Internal STOF (system) (1:Binary Integer)
1C6F7 Store user flags (1:Binary Integer)
1C731 Store system flags (1:Binary Integer)
1C783 →LIST (XLIB 2 152)
1C79E R→C (XLIB 2 153)
1C7CA RE (XLIB 2 154)
1C819 IM (XLIB 2 155)
1C85C SUB (XLIB 2 156)
1C8BB Internal SUB (3:String,2:Real Number,1:Real Number)
1C8CF Internal SUB (3:List,2:Real Number,1:Real Number)
1C8EA REPL (XLIB 2 157)
1C95A LIST→ (XLIB 2 158)
1C973 explode composite object → (N:...,1:Real Number)
1C98E C→R (XLIB 2 159)
1C9B8 SIZE (XLIB 2 160)
1CA26 Internal SIZE (1:String)
1CA3A Internal SIZE (1:Unit)
1CA4E Internal SIZE (1:Array)
1CA62 Internal SIZE (1:Graphic)
1CA85 Internal SIZE (1:Binary Integer)
1CAB4 POS (XLIB 2 161)
1CAD7 Internal POS (2:String,1:String)
1CAF0 Internal POS (2:List,1:Any)
1CB0B →STR (XLIB 2 162)
1CB26 STR→ (XLIB 2 163)
1CB46 NUM (XLIB 2 164)
1CB66 CHR (XLIB 2 165)
1CB86 TYPE (XLIB 2 166)
1CB90 Internal TYPE (1:Any)
1CDB1 Internal TYPE (1:Array)
1CDD4 Internal TYPE (1:Program)
1CE28 VTYPE (XLIB 2 167)
1CE55 Internal VTYPE (1:Global Name/Local Name)
1CE82 Internal VTYPE (1:Tagged)
1CEE3 EQ→ (XLIB 2 168)
1CF2E Internal EQ→ (1:Algebraic)
1CF7B OBJ→ (XLIB 2 169)
1CFD0 Internal OBJ→ (1:Algebraic)
1D009 →ARRY (XLIB 2 170)
1D02C Internal →ARRY (1:Real Number)
1D040 Internal →ARRY (1:List)
1D092 ARRY→ (XLIB 2 171)
1D0AB Internal ARRY→ (1:Array)
1D0DF RDM (XLIB 2 172)
1D10C Internal RDM (2:Array,1:List)
1D125 Internal RDM (2:Global Name,1:List)
1D152 Internal RDM (2:Local Name,1:List)
1D186 CON (XLIB 2 173)
1D1EA Internal CON (2:List,1:Real Number/Complex Number)
1D221 Internal CON (2:List,1:Complex Number)
1D23F Internal CON (2:Global Name,1:Real Number)
1D262 Internal CON (2:Global Name,1:Complex Number)
1D28A Internal CON (2:Local Name,1:Real Number)
1D2AD Internal CON (2:Local Name,1:Complex Number)
1D2DC IDN (XLIB 2 174)
1D313 Internal IDN (1:Real Number)
1D34A Internal IDN (1:Global Name)
1D36D Internal IDN (1:Local Name)
1D392 TRN (XLIB 2 175)
1D3BF Internal TRN (1:Global Name)
1D3E2 Internal TRN (1:Local Name)
1D407 PUT (XLIB 2 176)
1D484 Internal PUT (3:Global Name,2:Real Number/List,1:Any)
1D4DE Internal PUT (3:Array,2:Real Number/List,1:Real Number/Complex Number)
1D524 Internal PUT (3:List,2:Real Number/List,1:Any)
1D565 Internal PUT (3:Local Name,2:Real Number/List,1:Any)
1D5DF PUTI (XLIB 2 177)
1D65C Internal PUTI (3:Global Name,2:Real Number/List,1:Any)
1D6B6 Internal PUTI (3:Array,2:Real Number/List,1:Real Number/Complex Number)
1D701 Internal PUTI (3:List,2:Real Number/List,1:Any)
1D747 Internal PUTI (3:Local Name,2:Real Number/List,1:Any)
1D7C6 GET (XLIB 2 178)
1D825 Internal GET (2:Global Name/Local Name,1:Real Number/List)
1D86B Internal GET (2:Array,1:Real Number/List)
1D898 Internal GET (2:List,1:Real Number/List)
1D8C7 GETI (XLIB 2 179)
1D926 Internal GETI (2:Global Name/Local Name,1:Real Number/List)
1D96C Internal GETI (2:Array,1:Real Number/List)
1D9BC Internal GETI (2:List,1:Real Number/List)
1DD06 V→ (XLIB 2 180)
1DD29 Internal V→ (1:Complex Number)
1DD3D Internal V→ (1:Array)
1DE66 →V2 (XLIB 2 181)
1DE7F Internal →V2 (2:Real Number,1:Real Number)
1DEC2 →V3 (XLIB 2 182)
1DEDB Internal →V3 (3:Real Number,2:Real Number,1:Real Number)
1E04A INDEP (XLIB 2 183)
1E07E PMIN (XLIB 2 184)
1E09E PMAX (XLIB 2 185)
1E0BE AXES (XLIB 2 186)
1E0E8 CENTR (XLIB 2 187)
1E101 Internal CENTR (1:Real Number)
1E126 RES (XLIB 2 188)
1E150 *H (XLIB 2 189)
1E170 *W (XLIB 2 190)
1E190 DRAW (XLIB 2 191)
1E1AB AUTO (XLIB 2 192)
1E1C6 DRAX (XLIB 2 193)
1E1E1 SCALE (XLIB 2 194)
1E201 PDIM (XLIB 2 195)
1E22B DEPND (XLIB 2 196)
1E25F ERASE (XLIB 2 197)
1E27A PX→C (XLIB 2 198)
1E29A C→PX (XLIB 2 199)
1E2BA GRAPH (XLIB 2 200)
1E2D5 LABEL (XLIB 2 201)
1E2F0 PVIEW (XLIB 2 202)
1E31A PIXON (XLIB 2 203)
1E344 PIXOFF (XLIB 2 204)
1E36E PIX? (XLIB 2 205)
1E398 LINE (XLIB 2 206)
1E3C2 TLINE (XLIB 2 207)
1E3EC BOX (XLIB 2 208)
1E416 BLANK (XLIB 2 209)
1E436 PICT (XLIB 2 210)
1E456 GOR (XLIB 2 211)
1E46A Internal GOR (3:Graphic,2:List,1:Graphic)
1E488 Internal GOR (3:Graphic,2:Complex Number,1:Graphic)
1E4A6 Internal GOR (3:PICT,2:List,1:Graphic)
1E4C4 Internal GOR (3:PICT,2:Complex Number,1:Graphic)
1E4E4 GXOR (XLIB 2 212)
1E572 LCD→ (XLIB 2 213)
1E58D →LCD (XLIB 2 214)

1E5AD →GROB (XLIB 2 215)
1E5D2 ARC (XLIB 2 216)
1E606 TEXT (XLIB 2 217)
1E621 XRNG (XLIB 2 218)
1E641 YRNG (XLIB 2 219)
1E661 FUNCTION (XLIB 2 220)
1E681 CONIC (XLIB 2 221)
1E6A1 POLAR (XLIB 2 222)
1E6C1 PARAMETRIC (XLIB 2 223)
1E6E1 TRUTH (XLIB 2 224)
1E701 SCATTER (XLIB 2 225)
1E721 HISTOGRAM (XLIB 2 226)
1E741 BAR (XLIB 2 227)
1E761 SAME (XLIB 2 228)
1E783 AND (XLIB 2 229)
1E7DD Internal AND (2:Real Number,1:Real Number)
1E809 OR (XLIB 2 230)
1E863 Internal OR (2:Real Number,1:Real Number)
1E88F NOT (XLIB 2 231)
1E8D9 Internal NOT (1:Real Number)
1E8F6 XOR (XLIB 2 232)
1E946 Internal XOR (2:Real Number,1:Real Number)
1E972 == (XLIB 2 233)
1EA30 Internal == (2:Any,1:Any)
1EA44 Internal == (2:Tagged/Any,1:Tagged/Any)
1EA6C Internal == (2:Real Number,1:Complex Number)
1EA76 Internal == (2:Complex Number,1:Real Number)
1EA9D ≠ (XLIB 2 234)
1EB51 Internal ≠ (2:Any,1:Any)
1EB65 Internal ≠ (2:Tagged/Any,1:Tagged/Any)
1EB8D Internal ≠ (2:Real Number,1:Complex Number)
1EB97 Internal ≠ (2:Complex Number,1:Real Number)
1EBBE < (XLIB 2 235)
1EC40 Internal < (2:Real Number,1:Real Number)
1EC5D > (XLIB 2 236)
1ECDF Internal > (2:Real Number,1:Real Number)
1ECFC ≤ (XLIB 2 237)
1ED7E Internal ≤ (2:Real Number,1:Real Number)
1ED9B ≥ (XLIB 2 238)
1EE1D Internal ≥ (2:Real Number,1:Real Number)
1EE38 OLDPRT (XLIB 2 239)
1EE53 PR1 (XLIB 2 240)
1EE6E PRSTC (XLIB 2 241)
1EE89 PRST (XLIB 2 242)
1EEA4 CR (XLIB 2 243)
1EEBF PRVAR (XLIB 2 244)
1EEEC Internal PRVAR (1:Tagged)
1EF1E Internal PRVAR (1:List)
1EF43 DELAY (XLIB 2 245)
1EF63 PRLCD (XLIB 2 246)
1EF7E delta (complete derivative) (XLIB 2 247)
1EFD2 delta (stepwise derivative) (XLIB 2 248)
1F047 Internal DROP2 and push 0
1F0F5 Internal delta (stepwise derivative) (2:Algebraic,1:Symbolic)
1F133 RCEQ (XLIB 2 249)
1F14E STEQ (XLIB 2 250)
1F16E ROOT (XLIB 2 251)
1F1D4 integral (stack syntax) (XLIB 2 252)
1F201 Internal integral (stack syntax)
1F223 integral (algebraic syntax) (XLIB 2 253)
1F27A Internal integral (algebraic syntax)
1F2C9 Σ (XLIB 2 254)
1F354 | (stack syntax) (XLIB 2 255)
1F38B Internal | (stack syntax) (2:Symbolic,1:List)
1F3F3 | (algebraic syntax) (XLIB 2 256)
1F500 QUOTE (XLIB 2 257)
1F542 Internal QUOTE (1:Algebraic)
1F55D APPLY (stack syntax) (XLIB 2 258)
1F585 Internal APPLY (stack syntax) (2:List,1:Global Name/Local Name)
1F5C5 APPLY (algebraic syntax) (XLIB 2 259)
1F640 XLIB 2 260
1F8CF Internal STO (2:Any,1:Algebraic)
1F96F "num' (Local Name)
1F97E "fcn' (Local Name)
1F996 XLIB 2 261
1F9AE XLIB 2 262
1F9C4 →Q (XLIB 2 263)
1F9E9 →Qpi (XLIB 2 264)
1FA59 ^MATCH (XLIB 2 265)
1FA8D vMATCH (XLIB 2 266)
1FABA Internal ^MATCH (2:Real Number/Complex Number/Symbolic,1:List)
1FACE Internal vMATCH (2:Real Number/Complex Number/Symbolic,1:List)
1FAEB _ (XLIB 2 267)
1FB31 Internal _ (1:Real Number/Unit)
1FB5D RATIO (XLIB 2 268)
1FB87 DUP (XLIB 2 269)
1FBA2 DUP2 (XLIB 2 270)
1FBBD SWAP (XLIB 2 271)
1FBD8 DROP (XLIB 2 272)
1FBF3 DROP2 (XLIB 2 273)
1FC0E ROT (XLIB 2 274)
1FC29 OVER (XLIB 2 275)
1FC44 DEPTH (XLIB 2 276)
1FC64 DROPN (XLIB 2 277)
1FC7F DUPN (XLIB 2 278)
1FC9A PICK (XLIB 2 279)
1FCB5 ROLL (XLIB 2 280)
1FCD0 ROLLD (XLIB 2 281)
1FCEB CLEAR (XLIB 2 282)
1FD0B STOΣ (XLIB 2 283)
1FD2B CLΣ (XLIB 2 284)
1FD46 RCLΣ (XLIB 2 285)
1FD61 Σ+ (XLIB 2 286)
1FD8B Σ- (XLIB 2 287)
1FDA6 NΣ (XLIB 2 288)
1FDC1 CORR (XLIB 2 289)
1FDDC COV (XLIB 2 290)
1FDF7 ΣX (XLIB 2 291)
1FE12 ΣY (XLIB 2 292)
1FE2D ΣX^2 (XLIB 2 293)
1FE48 ΣY^2 (XLIB 2 294)
1FE63 ΣX*Y (XLIB 2 295)
1FE7E MAXΣ (XLIB 2 296)
1FE99 MEAN (XLIB 2 297)
1FEB4 MINΣ (XLIB 2 298)
1FECF SDEV (XLIB 2 299)
1FEEA TOT (XLIB 2 300)
1FF05 VAR (XLIB 2 301)
1FF20 LR (XLIB 2 302)
1FF7A PREDV (XLIB 2 303)
1FF9A PREDY (XLIB 2 304)
1FFBA PREDX (XLIB 2 305)
1FFDA XCOL (XLIB 2 306)
1FFFA YCOL (XLIB 2 307)
2001A UTPC (XLIB 2 308)
2003A UTPN (XLIB 2 309)
2005A UTPF (XLIB 2 310)
2007A UTPT (XLIB 2 311)
2009A COLΣ (XLIB 2 312)
200C4 SCLΣ (XLIB 2 313)
200F3 ΣLINE (XLIB 2 314)
2010E BINS (XLIB 3 315)
20133 BARPLOT (XLIB 2 316)
20167 HISTPLOT (XLIB 2 317)
2018C SCATRPLOT (XLIB 2 318)
201B1 LINFIT (XLIB 2 319)
201D6 LOGFIT (XLIB 2 320)
201FB EXPFIT (XLIB 2 321)
20220 PWRFIT (XLIB 2 322)
20234 set curve-fitting model in 'ΣPAR' (1:model)
2025E BESTFIT (XLIB 2 323)
202CE SINV (XLIB 2 324)
202F1 Internal SINV (1:Global Name)
20314 Internal SINV (1:Local Name)
2034D SNEG (XLIB 2 325)
20370 Internal SNEG (1:Global Name)
20393 Internal SNEG (1:Local Name)
203CC SCONJ (XLIB 2 326)
203EF Internal SCONJ (1:Global Name)
20412 Internal SCONJ (1:Local Name)
2044B STO+ (XLIB 2 327)
20482 Internal STO+ (2:Any,1:Global Name/Local Name)
20496 <644h>
204C3 Internal STO+ (2:Global Name/Local Name,1:Any)
20538 STO- (XLIB 2 328)
20583 Internal STO- (2:Any,1:Global Name/Local Name)
205A1 Internal STO- (2:Global Name/Local Name,1:Any)
205BF Internal STO- (2:Array,1:Global Name)
205E2 Internal STO- (2:Global Name,1:Array)
2060C STO/ (XLIB 2 329)

2066B Internal STO/ (2:Any,1:Global Name/Local Name)
20689 Internal STO/ (2:Global Name/Local Name,1:Any)
206A7 Internal STO/ (2:Global Name,1:Real Number/Complex Number)
206E8 Internal STO/ (2:Array,1:Global Name)
20729 Internal STO/ (2:Global Name,1:Array)
20753 STO* (XLIB 2 330)
207C6 Internal STO* (2:Any,1:Global Name/Local Name)
207E4 Internal STO* (2:Global Name/Local Name,1:Any)
20802 Internal STO* (2:Real Number/Complex Number,1:Global Name)
2082A Internal STO* (2:Global Name,1:Real Number/Complex Number)
2086B Internal STO* (2:Array,1:Global Name)
208AC Internal STO* (2:Global Name,1:Array)
208F4 INCR (XLIB 2 331)
20917 Internal INCR (1:Global Name)
20980 Internal INCR (1:Local Name)
209AA DECR (XLIB 2 332)
209CD Internal DECR (1:Global Name)
209EB Internal DECR (1:Local Name)
20A15 COLCT (XLIB 2 333)
20A49 EXPAN (XLIB 2 334)
20A7D RULES (XLIB 2 335)
20A93 ISOL (XLIB 2 336)
20AB3 QUAD (XLIB 2 337)
20AD3 SHOW (XLIB 2 338)
20B00 Internal SHOW (2:Symbolic,1:List)
20B20 TAYLR (XLIB 2 339)
20B40 RCL (XLIB 2 340)
20B81 Internal RCL (1:Global Name/Local Name)
20B9A Internal RCL (1:List (path/object))
20CAD Internal RCL (1:PICT)
20CCD STO (XLIB 2 341)
20D65 DEFINE (XLIB 2 342)
20D7E Internal DEFINE (1:Algebraic)
20DBF Internal DEFINE (1:Global Name/Local Name)
20EFE PURGE (XLIB 2 343)
20F35 Internal PURGE (1:List)
20F8A Internal PURGE (1:PICT)
20FAA MEM (XLIB 2 344)
20FD9 ORDER (XLIB 2 345)
20FF2 Internal ORDER (1:List)
210FC CLVAR (XLIB 2 346)
2115D TMENU (XLIB 2 347)
21196 MENU (XLIB 2 348)
211B4 'CST' (Global Name)
211E1 RCLMENU (XLIB 2 349)
211FC PVARS (XLIB 2 350)
2123A PGDIR (XLIB 2 351)
2125A ARCHIVE (XLIB 2 352)
21273 Internal ARCHIVE (1:Tagged)
2133C RESTORE (XLIB 2 353)
2137F MERGE (XLIB 3 354)
21398 Internal MERGE (1:Real Number)
213D1 FREE (XLIB 2 355)
21408 Internal FREE (2:Real Number/Global Name/Local Name,1:Real Number)
2142D LIBS (XLIB 2 356)
21448 ATTACH (XLIB 2 357)
21461 Internal ATTACH (1:Real Number)
2147C DETACH (XLIB 2 358)
21495 Internal DETACH (1:Real Number)
214A9 Internal R→SB and verify ≥ <100h> and ≠ <700h>
214F4 Internal STO (2:Any,1:Tagged)
215BF Internal STO (2:Library/Backup,1:Real Number)
21638 if TYPE(TOS) = Real Number, then do next/return, else skip next
2164C Internal SWAP and False
21660 deop level two object and True
21761 Internal RCL (1:Tagged)
217C7 Internal EVAL (1:Tagged)
217F1 Internal PURGE (1:Tagged)
21B2F Internal RESTORE (1:Backup)
21B74 Internal FREE (2:List,1:Real Number)
21C6F Internal ATTACH (1:System Binary)
21CBA Internal ATTACH to non-HOME directory (2:Directory,1:System Binary)
21CE5 Internal DETACH (1:System Binary)
21D2B Internal DETACH from non-HOME directory (2:Directory,1:System Binary)
21D54 Internal LIBS
21E75 XMIT (XLIB 2 359)
21E95 SRECV (XLIB 2 360)
21EB5 OPENIO (XLIB 2 361)
21ED5 CLOSEIO (XLIB 2 362)
21EF0 SEND (XLIB 2 363)
21F24 KGET (XLIB 2 364)
21F62 RECN (XLIB 2 365)
21F96 RECV (XLIB 2 366)
21FB6 FINISH (XLIB 2 367)
21FD1 SERVER (XLIB 2 368)
21FEC CKSM (XLIB 2 369)
2200C BAUD (XLIB 2 370)
2202C PARITY (XLIB 2 371)
2204C TRANSIO (XLIB 2 372)
2206C KERRM (XLIB 2 373)
22087 BUFLEN (XLIB 2 374)
220A2 STIME (XLIB 2 375)
220C2 SBRK (XLIB 2 376)
220DD PKT (XLIB 2 377)
224CA INPUT (XLIB 2 378)
224F4 ASN (XLIB 2 379)
22514 STOKEYS (XLIB 2 380)
22548 DELKEYS (XLIB 2 381)
22586 RCLKEYS (XLIB 2 382)
225A4 'S' (Global Name)
225BE →TAG (XLIB 2 383)
225F5 Internal →TAG (2:Any,1:String)
22618 Internal →TAG (2:Any,1:Real Number)
22633 DTAG (XLIB 2 384)
22647 Reference to hash table for library 002 (XLIB 2)
22651 Link table for library 002 (XLIB 2)
22DFE Reference to hash table for library 700 (XLIB 1792)
22E08 Link table for library 700 (XLIB 1792)
22EA3 Configuration code for library 700 (XLIB 1792)
22EC3 IF (XLIB 1792 0)
22EFA THEN (XLIB 1792 1)
22F22 Internal THEN (1:Real Number)
22F4F Internal THEN (1:Symbolic)
22FB5 ELSE (XLIB 1792 2)
22FD5 END (XLIB 1792 3)
22FEB → (XLIB 1792 4)
23033 WHILE (XLIB 1792 5)
2305D REPEAT (XLIB 1792 6)
23085 Internal REPEAT (1:Real Number)
230A3 Internal REPEAT (1:Symbolic)
230C3 DO (XLIB 1792 7)
230ED UNTIL (XLIB 1792 8)
23103 START (XLIB 1792 9)
23144 Internal START (2:Real Number,1:Real Number)
23167 Internal START (2:Real Number/Symbolic,1:Symbolic)
23180 Internal START (2:Symbolic,1:Real Number)
231A0 FOR (XLIB 1792 10)
231E1 Internal FOR (2:Real Number,1:Real Number)
23213 Internal FOR (2:Real Number/Symbolic,1:Symbolic)
2322C Internal FOR (2:Symbolic,1:Real Number)
2324C NEXT (XLIB 1792 11)
2326A Internal NEXT
23380 STEP (XLIB 1792 12)
233A8 Internal STEP (1:Symbolic)
233C1 Internal STEP (1:Real Number)
233DF IFERR (XLIB 1792 13)
23472 HALT (XLIB 1792 14)
2349C (XLIB 1792 15)
234C1 → (XLIB 1792 16)
235FE >> (XLIB 1792 17)
2361E << (XLIB 1792 18)
23639 >> (XLIB 1792 19)
23654 ' (XLIB 1792 20)
23679 ' (XLIB 1792 21)
23694 END (XLIB 1792 22)
236B9 END (XLIB 1792 23)
2371F THEN (XLIB 1792 24)
2372E "stop' (Local Name)
2373F "noname' (Local Name)
23754 { "noname' "stop' }
23768 ?? internal IF

2378D CASE (XLIB 1792 25)
237A8 THEN (XLIB 1792 26)
23813 DIR (XLIB 1792 27)
23824 PROMPT (XLIB 1792 28)
2387E "ioinprogress' (Local Name)
23908 'st' (Local Name)
23913 'ofs' (Local Name)
23920 'tok' (Local Name)
2394B 'st' (Local Name)
23956 'ofs' (Local Name)
23963 'tok' (Local Name)
24A2D 'i' (Local Name)
24A36 'j' (Local Name)
24A5D 'i' (Local Name)
24A6B 'j' (Local Name)
24B0A 'j' (Local Name)
24B1D 'i' (Local Name)
24B30 'i' (Local Name)
24BB6 'j' (Local Name)
24BD3 'i' (Local Name)
24BE1 'i' (Local Name)
25A0B "1' (Local Name)
25A16 "2' (Local Name)
25A21 "3' (Local Name)
25A3B "1' (Local Name)
25A46 "2' (Local Name)
25A51 "3' (Local Name)
272CD "ttt' (Local Name)
272DC "str' (Local Name)
272EB "ofs' (Local Name)
272FA "tok' (Local Name)
27309 "rbv' (Local Name)
27318 "idfflg' (Local Name)
2732D "tmpop' (Local Name)
27340 "tmppdat' (Local Name)
27357 "ploc' (Local Name)
27368 "bv' (Local Name)
27375 "unbound' (Local Name)
28A38 Internal _ (1:Symbolic)
29FDA Let A = Binary(TOS (Real)) / save B,D,D1,D0
2A181 XFER: load regs D,B,D1,D0 (uses C)
2A2B4 0
2A2C9 1
2A2DE 2
2A2DF 2
2A2F3 3
2A308 4
2A31D 5
2A332 6
2A347 7
2A35C 8
2A371 9
2A386 -1
2A39B -2
2A3B0 -3
2A3C5 -4
2A3D1 5
2A3DA -5
2A3EF -6
2A404 -7
2A419 -8
2A42E -9
2A443 3.14159265359
2A458 3.14159265358979 (Long Real)
2A472 9.99999999999E499
2A487 -9.99999999999E499
2A49C 1.E-499
2A4B1 -1.E-499
2A4C6 0 (Long Real)
2A4E0 1 (Long Real)
2A4FA 2 (Long Real)
2A514 3 (Long Real)
2A52E 4 (Long Real)
2A548 5 (Long Real)
2A562 .1 (Long Real)
2A57C .5 (Long Real)
2A596 10 (Long Real)
2A5B0 Internal LR→R (1:Long Real)
2A5C1 Internal R→LR (1:Real Number)
2A5D2 Internal DEG
2A5F0 Internal RAD
2A604 Internal GRAD
2A622 Internal D→R (1:Real Number)
2A655 internal R→D (1:Real Number)
2A673 Internal →HMS (1:Real Number)
2A68C Internal HMS→ (1:Real Number)
2A6A0 Internal HMS+ (2:Real Number,1:Real Number)
2A6C8 Internal HMS- (2:Real Number,1:Real Number)
2A6F5 Internal MAX (2:Real Number,1:Real Number)
2A70E Internal MIN (2:Real Number,1:Real Number)
2A738 if pop TOS < 0 (Real Number) → True/False
2A799 if pop TOS > 0 (Real Number) → True/False
2A871 Internal < (2:Real Number,1:Real Number) → True/False
2A88A Internal > (2:Real Number,1:Real Number) → True/False
2A8A0 Internal ≥ (2:Real Number,1:Real Number) → True/False
2A8B6 Internal ≤ (2:Real Number,1:Real Number) → True/False
2A8D7 Internal SIGN (1:Real Number)
2A900 Internal ABS (1:Real Number)
2A920 Internal NEG (1:Real Number)
2A930 Internal MANT (1:Real Number)
2A974 Internal + (2:Real Number,1:Real Number)
2A981 Internal - (2:Real Number,1:Real Number)
2A9BC Internal * (2:Real Number,1:Real Number)
2A9C9 Internal % (2:Real Number,1:Real Number)
2A9FE Internal / (2:Real Number,1:Real Number)
2AA0B Internal %T (2:Real Number,1:Real Number)
2AA30 Internal %CH (2:Real Number,1:Real Number)
2AAAF Internal INV (1:Real Number)
2AB2F Internal EXP (1:Real Number)
2AB42 Internal EXPM (1:Real Number)
2ABA7 Internal LNP1 (1:Real Number)
2ABBA Internal ALOG (1:Real Number)
2ABDC Internal MOD (2:Real Number,1:Real Number)
2ABEF Internal SIN (1:Real Number)
2AC40 Internal COS (1:Real Number)
2AC91 Internal TAN (1:Real Number)
2AD21 Internal ATAN (1:Real Number)
2ADAE Internal SINH (1:Real Number)
2ADDA Internal COSH (1:Real Number)
2ADED Internal TANH (1:Real Number)
2AE00 Internal ASINH (1:Real Number)
2AE39 Internal XPON (1:Real Number)
2AE62 Internal COMB (2:Real Number,1:Real Number)
2AE75 Internal PERM (2:Real Number,1:Real Number)
2AF4D Internal FP (1:Real Number)
2AF60 Internal IP (1:Real Number)
2AF73 Internal CEIL (1:Real Number)
2AF86 Internal FLOOR (1:Real Number)
2AFC2 Internal RAND
2B044 Internal RDZ (1:Real Number)
2B0C4 Internal ! (1:Real Number)
2B529 Internal RND (2:Real Number,1:Real Number)
2B53D Internal TRNC (2:Real Number,1:Real Number)
2C09F Internal UTPN (3:Real Number,2:Real Number,1:Real Number)
2C149 Internal UTPC (2:Real Number,1:Real Number)
2C174 Internal UTPF (3:Real Number,2:Real Number,1:Real Number)
2C19A Internal UTPT (2:Real Number,1:Real Number)
2C1F3 Internal STOΣ (1:Any)
2C1FD 'ΣDAT' (Global Name)
2C22F Internal CLΣ
2C293 Internal RCLΣ → Contents,True/False
2C2AC Internal RCLΣ
2C2D9 Internal Σ+ (1:Real Number)
2C32E Internal Σ+ (1:Array)
2C423 Internal Σ-
2C535 Internal NΣ
2C558 Internal MAXΣ
2C571 Internal MEAN
2C58A Internal MINΣ
2C5A3 Internal SDEV
2C5BC Internal TOT
2C5D5 Internal VAR
2C684 Internal COLΣ (2:Real Number/Array,1:Real Number)
2C6A2 store levels 1-5 into 'ΣPAR'
2C6C5 Internal XCOL (1:Real Number)
2C6DE Internal YCOL (1:Real Number)
2C738 'ΣPAR' (Global Name)

2C84B Internal CORR
2C8F5 Internal COV
2C94F Internal ΣX
2C963 Internal ΣY
2C977 Internal ΣX^2
2C99A Internal ΣY^2
2C9BD Internal ΣX*Y
2CA30 Internal LR
2CB02 Internal PREDY (1:Real Number)
2CB75 Internal PREDX (1:Real Number)
2D2E6 Error: Nonexistent ΣDAT
2D2EB MC: Error: Nonexistent ΣDAT
2D3A0 "PKNO' (Local Name)
2D3B1 "PACKET' (Local Name)
2D3C6 "RETRY' (Local Name)
2D3D9 "ERRMSG' (Local Name)
2D3EE "KP' (Local Name)
2D3FB "LNAME' (Local Name)
2D40E "OBJ' (Local Name)
2D41D "OPOS' (Local Name)
2D42E "EXCHP' (Local Name)
2D45A "KLIST' (Local Name)
2D46D "KMODE' (Local Name)
2D480 "KPTRN' (Local Name)
2D493 "KRM' (Local Name)
2D4A2 "MaxR' (Local Name)
2D816 Internal RECN (1:String/Global Name/Local Name)
2D9F5 Internal SERVER
2E5AB Internal SEND (1:Global Name/Local Name)
2E6EB Internal SEND (1:List)
2E7EF Internal KGET (1:String/Global Name/Local Name)
2E835 Internal KGET (1:List)
2E876 Internal FINISH
2E8D1 Internal PKT (2:String,1:String)
2E9D5 'IOPAR' (Global Name)
2EA59 'IOPAR' (Global Name)
2EC11 ABS(IP(Real Number)) → System Binary
2EC84 Internal BAUD (1:Real Number)
2ECCA Internal PARITY (1:Real Number)
2ED10 Internal TRANSIO (1:Real Number)
2ED4C Internal CKSM (1:Real Number)
2EDA6 Internal KERRM
2EDE1 Internal BUFLEN
2EDF5 Internal STIME (1:Real Number)
2EE18 Internal SBRK
2EE6F Internal XMIT (1:String)
2EE97 Internal SRECV (1:Real Number)
2F211 "KML' (Local Name)
2F46E "KEOF' (Local Name)
30794 Binary file header (HPHP48-B)
315C6 Internal CLOSEIO
31868 Internal CR
318A4 Internal PRSTC
318FE Internal PR1
31A25 Internal PRST
31C37 "IWrap' (Local Name)
31D56 Internal PRVAR (1:Global Name/Local Name)
31DAB Internal OLDPRT
31EE2 Internal PRLCD
31F87 'PRTPAR' (Global Name)
31FB8 'PRTPAR' (Global Name)
31FFD Internal DELAY (1:Real Number)
32F77 Internal ROOT (3 args)
34D30 " (Local Name)
34DBB "symb' (Global Name)
35CAE Internal CON (2:List,1:Real Number)
35D35 Internal IDN (1:Array)
35DEB Internal NEG (1:Array)
35E2C Internal RND (2:Array,1:Real Number)
35EA9 Internal TRNC (2:Array,1:Real Number)
35EC2 Internal RND (2:Complex Number,1:Real Number)
35F17 Internal TRNC (2:Complex Number,1:Real Number)
35F30 Internal CONJ (1:Array)
35F8F Internal RE (1:Array)
35FEE Internal IM (1:Array)
36039 Internal R→C (2:Array,1:Array)
360B6 Internal C→R (1:Array)
36115 Internal + (2:Array,1:Array)
36278 Internal - (2:Array,1:Array)
362DC Internal * (2/1:Real Number/Complex Number,1/2:Array)
363CC Internal / (2:Array,1:Real Number/Complex Number)
36435 Internal SQ (1:Array)
3643F Internal * (2:Array,1:Array)
365AC Internal RSD (3:Array,2:Array,1:Array)
366F6 Internal DOT (2:Array,1:Array)
36782 Internal CROSS (2:Array,1:Array)
368E5 Internal RNRM (1:Array)
368F9 Internal CNRM (1:Array)
3690D Internal RNRM... (1:Array)
369CB Internal ABS (1:Array)
36A2A Internal DET (1:Array)
36B0B Internal INV (1:Array)
36B60 Internal / (2:Array,1:Array)
36BF6 '#a' (Local Name)
36C01 '#b' (Local Name)
36C2F '#b' (Local Name)
36C3F '#a' (Local Name)
36CEF '#b' (Local Name)
36D18 '#b' (Local Name)
3811F Internal TRN (1:Array)
38A3E "SavedUI' (Local Name)
3A1FC Update menu display
3FACF 'SKEY' (Global Name)
3FAE8 'SKEY' (Local Name)
3FDD1 Internal RULES
40788 Null Program
4093B '.ENTER' (Global Name)
409DF '.ENTER' (Global Name)
415C9 Internal RCLMENU
41679 Internal TMENU (1:Real Number)
41A43 'UserKeys' (Global Name)
41A69 'UserKeys.CRC' (Global Name)
41AA1 Intenral STOKEYS (1:List)
41B28 Internal ASN (2:Any,1:Real Number)
41B3C Internal DELKEYS (1:List)
41B69 Internal DELKEYS (1:Real Number)
41BA5 Internal STOKEYS (1:Global Name/Local Name)
41BB9 Internal DELKEYS (1:Global Name/Local Name)
41BD7 'S' (Global Name)
41BEA 'S' (Local Name)
43395 Internal INPUT (2:String,1:String)
433CC Internal INPUT (2:String,1:List)
4353E 'ALG' (Global Name)
43555 'ALG' (Local Name)
4358A '.' (Global Name)
4359D '.' (Local Name)
435CE 'V' (Global Name)
435E1 'V' (Local Name)
47459 'X' (Global Name)
47A1A Internal XRNG (2:Real Number,1:Real Number)
47A42 Internal YRNG (2:Real Number,1:Real Number)
47A6A Internal INDEP (1:Real Number)
47A8D Internal DEPND (1:Real Number)
48D4B 'ALRMDAT' (Global Name)
491D5 Internal AUTO
4A145 'X' (Global Name)
4A16C Internal ΣLINE
4A19E 'X' (Global Name)
4A1DE 'X' (Global Name)
4A22D 'X' (Global Name)
4A25E 'X' (Global Name)
4AB1C 'X' (Global Name)
4AB2A (0,0)
4AB59 'Y' (Global Name)
4AC61 Internal CENTR (1:Complex Number)
4AE3C Internal SCALE (2:Real Number,1:Real Number)
4AF77 Internal INDEP (1:Global Name)
4AF8B Internal INDEP (1:List)
4AFB3 Internal DEPND (1:Global Name)
4AFC7 Internal DEPND (1:List)
4AFEF Internal RES (1:Real Number)
4B012 Internal RES (1:positive Real Number/Binary Integer)
4B03A Internal AXES (1:Complex Number)
4B04E Internal AXES (1:List)
4B09E Internal PMIN (1:Complex Number)
4B0C6 Internal PMAX (1:Complex Number)
4B206 Internal PDIM (2:Complex Number,1:Complex Number)

4B300 Internal PDIM (2:Binary Integer,1:Binary Integer)
4B323 Internal PDIM (2:System Binary,1:System Binary)
4B553 Internal *H (1:Real Number)
4B5AD Internal *W (1:Real Number)
4B60C Internal ERASE
4B6AC Internal DRAW
4C607 Internal DRAX
4C8F4 Internal BINS (3:Real Number,2:Real Number,1:Real Number)
4C944 "xmax' (Local Name)
4C955 "N' (Local Name)
4CF55 "EnvOK' (Local Name)
4CF68 "EXITFCN' (Local Name)
4D1AA Internal GRAPH
4D30D "EnvOK' (Local Name)
4D352 "EnvOK' (Local Name)
4D36F "EnvOK' (Local Name)
4E875 Internal LABEL
4F011 Internal PVIEW (1:Complex Number)
4F02F Internal PVIEW (1:List)
4F0AC Internal PX→C (1:List)
4F179 Internal C→PX (1:Complex Number)
4F37C Internal STO (2:Graphic,1:PICT)
4F3D1 Convert level 1 and 2 Binary Integers to System Binary
4F3EF Internal PIXON (1:Complex Number)
4F458 Internal PIXON (1:List)
4F471 Internal PIXOFF (1:Complex Number)
4F48A Internal PIXOFF (1:List)
4F4A3 Internal PIX? (1:Complex Number)
4F4BC Internal PIX? (1:List)
4F525 Internal LINE (2:List,1:List)
4F539 Internal TLINE (2:List,1:List)
4F584 Internal LINE (2:Complex Number,1:Complex Number)
4F598 Internal TLINE (2:Complex Number,1:Complex Number)
4F665 Internal BOX (2:List,1:List)
4F688 Internal BOX (2:Complex Number,1:Complex Number)
4F6A1 Internal BLANK (2:Binary Integer,1:Binary Integer)
4F6BA Internal GOR/GXOR (4:True/False,3:Graphic,2:List,1:Graphic)
4F6F6 Internal GOR/GXOR (4:True/False,3:Graphic,2:Complex Number,1:Graphic)
4F741 Internal GOR/GXOR (4:True/False,3:PICT,2:Complex Number/List,1:Graphic)
4F8D1 Internal + (2:Graphic,1:Graphic)
4F999 Internal REPL (3:Graphic,2:List,1:Graphic)
4F9F3 Internal REPL (3:Graphic,2:Complex Number,1:Graphic)
4FA2F Internal REPL (3:PICT,2:List/Complex Number,1:Graphic)
4FA7A Internal REPL (3:List,2:Real Number,1:List)
4FAF7 Internal REPL (3:String,2:Real Number,1:String)
4FB74 Internal SUB (3:Graphic,2:List,1:List)
4FBC4 Internal SUB (3:Graphic,2:Complex Number,1:Complex Number)
4FBF6 Internal SUB (3:PICT,2/1:List or Complex Number)
4FC28 Internal NEG (1:Graphic)
4FC3C Internal NEG (1:PICT)
4FC5F Internal ARC (4:Complex Number,3:Real Number,2:Real Number,1:Real Number)
4FD2C Internal ARC (4:List,3:Binary Integer,2:Real Number,1:Real Number)
4FF9D "xe' (Local Name)
4FFAA "ye' (Local Name)
4FFB7 "x' (Local Name)
4FFC2 "y' (Local Name)
4FFCD "xc' (Local Name)
4FFDA "yc' (Local Name)
4FFE7 "r2' (Local Name)
4FFF4 "left' (Local Name)
50005 "up' (Local Name)
50012 "exit' (Local Name)
503C5 XFER: Internal TEXT
503D4 Internal LCD→
50438 Internal →LCD (1:Graphic)
5046A Internal CLLCD
5048D Internal →GROB (2:Any,1:Real Number)
50578 get GROB dimensions (2:Rows,1:Columns)
50D3E "PlotEnv' (Local Name)
50FCE 'X' (Global Name)
50FE6 'Y' (Global Name)
51148 validate pop TOS is PICT (Bad Argument Type)
51288 'PPAR' (Global Name)
51300 place SCATTER on the stack, do not call
51314 place HISTOGRAM (XLIB 2 226) on the stack, do not call
51328 place BAR (XLIB 2 227) on the stack, do not call
51436 'st' (Global Name)
5179E push GROB dimensions (2:Rows,1:Columns)
5187F get PICT dimensions (2:Rows,1:Columns)
5190B "PlotEnv' (Local Name)
5196A (-1,0)
5198F Internal IM (1:Real Number)
519A3 Internal RE (1:Complex Number)
519B7 Internal IM (1:Complex Number)
51B70 Internal NEG (1:Complex Number)
51BB2 Internal CONJ (1:Complex Number)
51BD0 Internal + (2:Complex Number,1:Real Number)
51BF8 Internal + (2:Real Number,1:Complex Number)
51C16 Internal + (2:Complex Number,1:Complex Number)
51CD4 Internal - (2:Real Number,1:Complex Number)
51CE8 Internal - (2:Complex Number,1:Real Number)
51CFC Internal - (2:Complex Number,1:Real Number)
51D4C Internal * (2:Complex Number,1:Real Number)
51D60 Internal * (2:Real Number,1:Complex Number)
51D88 Internal * (2:Complex Number,1:Complex Number)
51E19 Internal / (2:Real Number,1:Complex Number)
51E64 Internal / (2:Complex Number,1:Real Number)
51EC8 Internal / (2:Complex Number,1:Complex Number)
51EFA Internal INV (1:Complex Number)
52062 Internal ABS (1:Complex Number)
52099 Internal ARG (1:Complex Number)
520CB Internal SIGN (1:Complex Number)
52107 Internal sqrt (1:Complex Number)
52193 Internal EXP (1:Complex Number)
521E3 Internal LN (1:Complex Number)
522BF Internal LOG (1:Complex Number)
52305 Internal ALOG (1:Complex Number)
52342 Internal ^ (2:Real Number,1:Complex Number)
52360 Internal ^ (2:Complex Number,1:Real Number)
52374 Internal ^ (2:Complex Number,1:Complex Number)
524AF (0,0)
524F7 (1,0)
52530 Internal SIN (1:Complex Number)
52571 Internal COS (1:Complex Number)
525B7 Internal TAN (1:Complex Number)
5262F Internal SINH (1:Complex Number)
52648 Internal COSH (1:Complex Number)
5265C Internal TANH (1:Complex Number)

52675 Internal ATAN (1:Complex Number)
5267F (0,1)
526AE (0,-1)
527EB Internal ATANH (1:Complex Number)
52804 Internal ASIN (1:Complex Number)
5281D Internal ASINH (1:Complex Number)
52836 Internal ACOSH (1:Complex Number)
52863 Internal ACOS (1:Complex Number)
53725 Set user flag (1:System Binary)
53731 Set system flag (1:System Binary)
53755 Clear user flag (1:System Binary)
53761 Clear system flag (1:System Binary)
53778 User flag set? (1:System Binary) → True/False
53784 System flag set? (1:System Binary) → True/False
53807 XFER: Error Bad Arg Val
5380E If pop TOS = True then push 1 else push 0
53842 If Last Arguments flag is clear then ???
53C37 Internal HEX
53C43 Internal BIN
53C4F Internal OCT
53C5B Internal DEC
53C96 Internal STWS (1:Real Number)
53CAA Internal STWS (1:System Binary)
53CF0 Internal RCWS
53D04 Internal AND (2:Binary Integer,1:Binary Integer)
53D15 Internal OR (2:Binary Integer,1:Binary Integer)
53D26 Internal XOR (2:Binary Integer,1:Binary Integer)
53D4E Internal NOT (1:Binary Integer)
53D5E Internal SL
53D6E Internal SLB
53D81 Internal SR
53D91 Internal SRB
53DA4 Internal RR
53DE1 Internal RRB
53E0C Internal RL
53E3B Internal RLB
53E65 Internal ASR
53EA0 Internal + (2:Binary Integer,1:Binary Integer)
53EB0 Internal - (2:Binary Integer,1:Binary Integer)
53EC3 Internal NEG (1:Binary Integer)
53ED3 Internal * (2:Binary Integer,1:Binary Integer)
53F05 Internal / (2:Binary Integer,1:Binary Integer)
53F69 XFER: Save Regs D,B,D1,D0 (uses C)
53F70 XFER: load regs D,B,D1,D0 (uses C)
54021 Let A.A = (@706C6 & #3F) + 1
54039 Internal RCWS → System Binary
540BB get contents
5429F Internal / (2:Real Number,1:Binary Integer)
542BD Internal / (2:Binary Integer,1:Real Number)
542D1 Internal * (2:Real Number,1:Binary Integer)
542EA Internal * (2:Binary Integer,1:Real Number)
542FE Internal - (2:Real Number,1:Binary Integer)
5431C Internal - (2:Binary Integer,1:Real Number)
54330 Internal + (2:Real Number,1:Binary Integer)
54349 Internal + (2:Binary Integer,1:Real Number)
5435D Internal B→R
543F9 Internal R→B
54419 push A (Real) as Binary Integer
544D9 Internal == (2:Binary Integer,1:Binary Integer)
544EC Internal ≠ (2:Binary Integer,1:Binary Integer)
54500 Internal > (2:Binary Integer,1:Binary Integer)
5452C Internal ≥ (2:Binary Integer,1:Binary Integer)
5453F Internal ≤ (2:Binary Integer,1:Binary Integer)
54552 Internal < (2:Binary Integer,1:Binary Integer)
54565 Internal IFTE (3:Symbolic,2/1:Real/Complex/Symbolic)
5456F "tcls' (Local Name)
54580 "fcls' (Local Name)
5460E "tcls' (Local Name)
54624 "fcls' (Local Name)
5465D "tcls' (Local Name)
5466E "fcls' (Local Name)
54954 Internal delta (complete derivative) (2:Symbolic,1:Symbolic)
549CC { "dvar' }
549DB "dvar' (Local Name)
54CDB Internal MINR (1.E-499)
54D12 Internal MAXR (9.99999999999E499)
54D35 Internal pi (3.14159265359)
54D58 Internal i (0,1)
54D7B Internal e (2.71828182846)
54DD0 "xSYMfcn' (Local Name)
54DE7 "xfcn' (Local Name)
54EA0 Internal RE (1:Symbolic)
54EB9 Internal IM (1:Symbolic)
54ED2 Internal NOT (1:Symbolic)
54EEB Internal NEG (1:Symbolic)
54F04 Internal ABS (1:Symbolic)
54F1D Internal CONJ (1:Symbolic)
54F36 Internal INV (1:Symbolic)
54F4F Internal ARG (1:Symbolic)
54F68 Internal SIGN (1:Symbolic)
54F81 Internal sqrt (1:Symbolic)
54F9A Internal SQ (1:Symbolic)
54FB3 Internal SIN (1:Symbolic)
54FCC Internal COS (1:Symbolic)
54FE5 Internal TAN (1:Symbolic)
54FFE Internal SINH (1:Symbolic)
55017 Internal COSH (1:Symbolic)
55030 Internal TANH (1:Symbolic)
55049 Internal ASIN (1:Symbolic)
55062 Internal ACOS (1:Symbolic)
5507B Internal ATAN (1:Symbolic)
55094 Internal ASINH (1:Symbolic)
550AD Internal ACOSH (1:Symbolic)
550C6 Internal ATANH (1:Symbolic)
550DF Internal EXP (1:Symbolic)
550F8 Internal LN (1:Symbolic)
55111 Internal LOG (1:Symbolic)
5512A Internal ALOG (1:Symbolic)
55143 Internal LNP1 (1:Symbolic)
5515C Internal EXPM (1:Symbolic)
55175 Internal ! (1:Symbolic)
5518E Internal IP (1:Symbolic)
551A7 Internal FP (1:Symbilic)
551C0 Internal FLOOR (1:Symbolic)
551D9 Internal CEIL (1:Symbolic)
551F2 Internal XPON (1:Symbolic)
5520B Internal MANT (1:Symbolic)
55224 Internal D→R (1:Symbolic)
5523D Internal R→D (1:Symbolic)
55256 Internal UBASE (1:Symbolic)
5526F Internal UVAL (1:Symbolic)
5566B "oth' (Local Name)
55783 "scl' (Local Name)
55792 "xSYMfcn' (Local Name)
557A9 "xfcn' (Local Name)
55800 "xSYMfcn' (Local Name)
55817 "xfcn' (Local Name)
55927 Internal = (2/1:all combinations)
5599A Internal AND (2:Symbolic,1:Real Number)
559B3 Internal AND (2:Real Number,1:Symbolic)
559CC Internal AND (2:Symbolic,1:Symbolic)
559E5 Internal OR (2:Symbolic,1:Real Number)
559FE Internal OR (2:Real Number,1:Symbolic)
55A17 Internal OR (2:Symbolic,1:Symbolic)
55A30 Internal XOR (2:Symbolic,1:Real Number)
55A49 Internal XOR (2:Real Number,1:Symbolic)
55A62 Internal XOR (2:Symbolic,1:Symbolic)
55A7B Internal == (2:Symbolic,1:Real Number/Complex Number/Unit)
55A94 Internal == (2:Complex Number,1:Real Number/Complex Number/Unit)
55AAD Internal == (2:Symbolic,1:Symbolic)
55AC6 Internal ≠ (2:Symbolic,1:Real Number/Complex Number/Unit)

55ADF Internal ≠ (2:Real Number/Complex Number/Unit, 1:Symbolic)
55AF8 Internal ≠ (2:Symbolic,1:Symbolic)
55B11 Internal < (2:Symbolic,1:Real Number/Unit)
55B2A Internal < (2:Real Number/Unit,1:Symbolic)
55B43 Internal < (2:Symbolic,1:Symbolic)
55B5C Internal > (2:Symbolic,1:Real Number/Unit)
55B75 Internal > (2:Real Number/Unit,1:Symbolic)
55B8E Internal > (2:Symbolic,1:Symbolic)
55BA7 Internal ≤ (2:Symbolic,1:Real Number/Unit)
55BC0 Internal ≤ (2:Real Number/Unit,1:Symbolic)
55BD9 Internal ≤ (2:Symbolic,1:Symbolic)
55BF2 Internal ≥ (2:Symbolic,1:Real Number/Unit)
55C0B Internal ≥ (2:Real Number/Unit,1:Symbolic)
55C24 Internal ≥ (2:Symbolic,1:Symbolic)
55C3D Internal % (2:Symbolic,1:Real Number/Unit)
55C56 Internal % (2:Real Number/Unit,1:Symbolic)
55C6F Internal % (2:Symbolic,1:Symbolic)
55C88 Internal %CH (2:Symbolic,1:Real Number/Unit)
55CA1 Internal %CH (2:Real Number/Unit,1:Symbolic)
55CBA Internal %CH (2:Symbolic,1:Symbolic)
55CD3 Internal %T (2:Symbolic,1:Real Number/Unit)
55CEC Internal %T (2:Real Number/Unit,1:Symbolic)
55D05 Internal %T (2:Symbolic,1:Symbolic)
55D1E Internal COMB (2:Symbolic,1:Real Number)
55D37 Internal COMB (2:Real Number,1:Symbolic)
55D50 Internal COMB (2:Symbolic,1:Symbolic)
55D69 Internal PERM (2:Symbolic,1:Real Number)
55D82 Internal PERM (2:Real Number,1:Symbolic)
55D9B Internal PERM (2:Symbolic,1:Symbolic)
55DB4 Internal RND (2:Symbolic,1:Real Number)
55DCD Internal RND (2:Real Number/Complex Number/Array/Unit,1:Symbolic)
55DE6 Internal RND (2:Symbolic,1:Symbolic)
55DFF Internal TRNC (2:Symbolic, 1:Real Number)
55E18 Internal TRNC (2:Real Number/Complex Number/Array/Unit,1:Symbolic)
55E31 Internal TRNC (2:Symbolic,1:Symbolic)
55E4A Internal MAX (2:Symbolic,1:Real Number/Unit)
55E63 Internal MAX (2:Real Number/Unit,1:Symbolic)
55E7C Internal MAX (2:Symbolic,1:Symbolic)
55E95 Internal MIN (2:Symbolic,1:Real Number/Unit)
55EAE Internal MIN (2:Real Number/Unit,1:Symbolic)
55EC7 Internal MIN (2:Symbolic,1:Symbolic)
55EE0 Internal ^ (2:Symbolic,1:Real Number/Complex Number/Unit)
55EF9 Internal ^ (2:Real Number/Complex Number/Unit,1:Symbolic)
55F12 Internal ^ (2:Symbolic,1:Symbolic)
55F2B Internal + (2:Symbolic,1:Real Number/Complex Number/Unit)
55F44 Internal + (2:Real Number/Complex Number/Unit,1:Symbolic)
55F5D Internal + (2:Symbolic,1:Symbolic)
55F76 Internal - (2:Symbolic,1:Real Number/Complex Number/Unit)
55F8F Internal - (2:Real Number/Complex Number/Unit,1:Symbolic)
55FA8 Internal - (2:Symbolic,1:Symbolic)
55FC1 Internal * (2:Symbolic,1:Real Number/Complex Number/Unit)
55FDA Internal * (2:Real Number/Complex Number/Unit,1:Symbolic)
55FF3 Internal * (2:Symbolic,1:Symbolic)
5600C Internal / (2:Symbolic,1:Real Number/Complex Number/Unit)
56025 Internal / (2:Real Number/Complex Number/Unit,1:Symbolic)
5603E Internal / (2:Symbolic,1:Symbolic)
56057 Internal MOD (2:Symbolic,1:Real Number)
56070 Internal MOD (2:Real Number,1:Symbolic)
56089 Internal MOD (2:Symbolic,1:Symbolic)
56859 'IERR' (Global Name)
56949 Internal Σ (4:Symbolic,3:Symbolic,2:Symbolic,1:Any)
56976 "sumexpr' (Local Name)
5698D "sumvar' (Local Name)
56A06 Internal Σ (4:Symbolic,3:Symbolic,2:Real Number,1:Any)
56A4C Internal Σ (4:Symbolic,3:Real Number,2:Symbolic,1:Any)
56AC9 Internal Σ (4:Symbolic,3:Real Number,2:Real Number,1:Any)
56F0B "dv' (Local Name)
5720B "nm' (Local Name)
57218 "op' (Local Name)
572A2 Internal ISOL (2:Symbolic,1:Global Name)
578E2 "ni' (Local Name)
578EF "ns' (Local Name)
5793F 'n0' (Global Name)
5795D 's0' (Global Name)
57A0C Internal EXPAN (1:Real Number/Complex Number/Symbolic)
57D90 Internal COLCT (1:Real Number/Complex Number/Symbolic)
57EF3 "*s' (Local Name)
58149 "+s' (Local Name)
58D75 Internal SHOW (2:Symbolic,1:Global Name/Local Name)
58DB6 "fl' (Local Name)
59115 "nmls' (Local Name)
591AD Internal QUAD (2:Symbolic,1:Global Name)
592C0 "c' (Local Name)
592CB "b' (Local Name)
592D6 "a' (Local Name)
59304 's1' (Global Name)
59517 "n' (Local Name)
59522 "prog' (Local Name)
595DD Internal TAYLR (3:Symbolic,2:Global Name,1:Real Number)
59646 "n' (Local Name)
59F91 Internal SIZE (1:Symbolic)
5A60F { "piflag' }
5A614 "piflag' (Local Name)
5A665 "d' (Local Name)
5A670 "r' (Local Name)
5A761 "d' (Local Name)
5A76C "R' (Local Name)
5A777 "est' (Local Name)
5A786 "X' (Local Name)
5A791 "T' (Local Name)
5AAE5 "bnds' (Local Name)
5D67D "which' (Local Name)
5D690 "op1' (Local Name)
5D69F "op2' (Local Name)
5FDC1 "ct' (Local Name)
5FDCE "pp' (Local Name)
5FDDB "ep' (Local Name)
6080B "reg' (Local Name)
6081A "sur' (Local Name)
60829 "cts' (Local Name)
60838 "sun' (Local Name)
60847 "mlg' (Local Name)
60856 "ckd' (Local Name)
60865 "prd' (Local Name)
60874 "prp' (Local Name)
60883 "rhs' (Local Name)
60BE9 "patternls' (Local Name)
60C04 "compos' (Local Name)

60C19 "varls' (Local Name)
60C4F "patternls' (Local Name)
60C6A "compos' (Local Name)
60CCA "compos' (Local Name)
60D7F "varls' (Local Name)
60E8C '&1' (Local Name)
60E97 '&2' (Local Name)
60EA2 '&3' (Local Name)
60EAD '&4' (Local Name)
60F9B drop level two object
60FBB Internal 4 ROLL
60FD8 Internal 5 ROLL
6112A drop level two and three objects
611FE Internal 3 PICK
618F7 if pop TOS = True, then DROP, do next and return, else skip next
6194B if pop TOS = True, then pop and return
61993 if pop TOS = True, then do next/return, else skip next
6199F MC XFER: return and execute the next token in this stream
619A6 XFER: skip next RPL token and continue RPL
619AD if pop TOS = True, skip next, else do next and return
619BC if pop TOS ≠ True, then skip next token
61A02 if pop TOS = True, then set carry flag
61A2C if pop TOS ≠ True, then return
61A3B if pop TOS = True, then return
61A47 XFER: RPL RETURN
61A4E continue RPL
61AD8 if pop TOS = True, then do/skip, else skip/do
61B72 if pop TOS ≠ True, then DROP
61C1C allocate nibbles (2:sized object,1:System Binary)
61D3A " (Local Name)
61F1B if pop TOS = True, then SWAP
62035 Internal DUP (?)
6205B MC: TYPE(pop TOS) = C → True/False
62063 Check Prolog → True/False
62080 Replace TOS with True
620A0 Replace TOS with False
620B9 MC: restore regs and push True
620D2 MC: restore regs and push False
62115 test TYPE(TOS) = Local Name, push True/False
62169 test TYPE(TOS) = Real Number, push True/False
6216E test TYPE(TOS) = Real Number, replace TOS with True/False
621C2 test TYPE(TOS) = Directory, replace TOS with True/False
621FC test TYPE(TOS) = Graphic, push True/False
6222B test TYPE(TOS) = Tagged, replace TOS with True/False
6223B Check for Real Array → True/False
6249E MC XFER: pop TOS (System Binary) → A.A
624B3 XFER: save regs D,B,D1,D0 (uses C)
6256A Internal add 3 (1:System Binary)
6257A Internal add 4 (1:System Binary)
6258A Internal add 5 (1:System Binary)
6259A Internal add 6 (1:System Binary)
625AA internal add 7 (1:System Binary)
625BA Internal add 8 (1:System Binary)
625CA Internal add 9 (1:System Binary)
625DA Internal add 10 (1:System Binary)
625EA Internal add 12 (1:System Binary)
625FA Internal subtract 3 (1:System Binary)
6260A Internal subtract 4 (1:System Binary)
6261A Internal subtract 5 (1:System Binary)
6262A Internal subtract 6 (1:System Binary)
62636 MC: add C.A to TOS and continue RPL (1:System Binary)
6264E Internal multiply by 10 (1:System Binary)
62674 Internal multiply by 8 (1:System Binary)
62691 Internal multiply by 6 (1:System Binary)
626AE Return and execute the sixth token in this stream
626DC Return and execute the fourth token in this stream
626E5 Return and execute the third token in this stream
626EE Return and execute the second token in this stream
626F7 Internal DUP and add two (1:System Binary)
62747 Internal SWAP and DUP
627BB Internal DUP and SIZE (1:String) → (2:String,1:System Binary)
62829 XFER: pop 2 System Binary → A.A,C.A
6289B If TOS > TOS-1 (System Binary) → True/False
62986 Return, if pop TOS = True, then do next/return, else skip next
629A1 Return, if pop TOS = True, then skip next, else do next/return
62A34 Internal RCL (1:Global Name/Local Name) → Contents,True/False
62B0B Internal DROP2 and False
62B88 Internal LIST→ (1:List) → (N:...,1:Any) (no count)
62BC4 Internal 7 ROLLD (7:...,1:Any)
62BF1 Internal *10 (1:Real Number)
62C41 Internal LIST→ (1:List) → (N:...,1:System Binary) (DUP'd count)
62C69 Internal NEWOB and SWAP
62C7D Internal ROT and DUP2 (3:Any,2:Any,1:Any)
62CA5 Internal ROT and OVER
62CB9 Internal DUP/DUP
62CCD Internal OVER and DUP
62CE1 Internal R→SB and DUP
62D31 Internal OVER and SWAP
62D45 Internal ROLL and SWAP (N:...,1:System Binary)
62D59 Internal "" and SWAP
62D81 if TOS-1 ≤ TOS (Real Number) then SWAP
62D9F if pop TOS ≠ True then SWAP
62E26 Internal add 1 and SWAP (2:Any,1:System Binary)
62E3A Internal <0h> and SWAP
62E4E Internal sub 1, <1h> and SWAP (1:System Binary)
62E67 Internal <1h> and SWAP
62E7B Internal R→SB and SWAP (1:Real Number)
62E8F Internal R→LR and SWAP (1:Real Number)
62F1B if pop TOS = True, then SWAP
62F43 if pop TOS = True, then DROP, else SWAP/DROP
62F5C if pop TOS = True, then SWAP/DROP, else DROP
62F75 Internal DROPN and DROP (1:System Binary)
62F89 Internal ROLL and DROP (N:...,1:System Binary)
62FB1 Internal DUP and ROT
62FC5 Internal DROP and ROT
62FD9 Internal subtract one and ROT (1:System Binary)
63029 Internal DROP and OVER
63079 <0h> and internal OVER
630DD Internal DUP and PICK (N:...,1:System Binary)
630F1 Internal DUP and ROLL (N:...,1:System Binary)
63119 Internal 8 ROLLD
6312D Internal 10 ROLLD
6317D if pop TOS = True, then internal UVAL
631A5 Internal subtract one and →LIST (N:...,1:System Binary)
631B9 Internal 2 →LIST
631CD Internal 3 →LIST
631E1 Internal DUP and LIST→
631F5 Internal SWAP and LIST→
632A9 Internal SWAP and R→C (2:Real Number,1:Real Num-

ber)
632BD Unevaluated No Operation (continue RPL)
632D1 →Program and EVAL (N:...,1:System Binary)
63399 if TOS-1 > TOS (System Binary), then skip next token
633B2 Internal ELSE
63411 Internal DUP and push current loop counter as System Binary
63425 Internal SWAP and push current loop counter as System Binary
63439 Internal OVER and push current loop counter as System Binary
6344D Internal SWAP and next (internal loop)
63466 Internal DROP and next (internal loop)
6347F Internal DUP and for 0 to (TOS)-1 (1:System Binary)
634F7 Internal True and False
6350B Internal False and True
6351F <0h> and False
63533 <1h> and False
635C4 Internal ≠ (2:Any,1:Any) → True/False
6365F Internal OVER and if TOS-1 < TOS (System Binary) → True/False
63673 if TOS < 3 (System Binary) → True/False
63687 Internal DUP and if TOS < 7 (System Binary) → True/False
636C8 if TOS ≠ 2 (System Binary) → True/False
636DC Internal OVER and if TOS-1 > TOS (System Binary) → True/False
636F0 if TOS > 1 (System Binary) → True/False
637B8 Internal ROT and add one (System Binary)
637F4 Internal DROP and subtract one (System Binary)
6386C Internal SWAP and DUP2
63A29 Store TOS into "dvar' (Local Name)
63A3D Store TOS into "LNAME' (Local Name)
63A6F Internal DUP and if TOS is { } → True/False
63A88 Internal DUP and <0h>
63A9C Internal DUP and <1h>
63B19 if pop TOS ≠ True, then Bad Argument Value
63B2D if TYPE(TOS) ≠ Real Number, then Bad Argument Type
63B46 if pop TOS ≠ True, then Bad Argument Type
63B5A Unevaluated * (multiply)
63B6E Unevaluated delta (stepwise derivative)
63B96 Internal SB→LR (1:System Binary)
63C04 get GROB columns
63C2C Internal SWAP and 4 ROLL
63C40 Internal DUP2 and 5 ROLL
63CFE if SAME, then do next/return, else skip next
63D3A if TOS-1 = TOS (System Binary), then skip next, else do next and return
63D67 if TOS-1 > TOS (System Binary), then do next/return, else skip next
641CC do DUP / continue RPL ML thread
64775 Internal DTAG (1:Any)
647A2 Internal DTAG level 2 object
64B12 <2Ch>
64B1C <2Dh>
64B26 <2Eh>
64B30 <2Fh>
64B3A <30h>
64B44 <31h>
64B4E <32h>
64B58 <33h>
64B62 <34h>
64B6C <35h>
64B76 <36h>
64B80 <37h>
64B8A <38h>
64B94 <39h>
64B9E <3Ah>
64BA8 <3Bh>
64BB2 <3Ch>
64BBC <3Dh>
64BC6 <3Eh>
64BD0 <3Fh>
64BDA <40h>
64BE4 <41h>
64BEE <42h>
64BF8 <43h>
64C02 <44h>
64C0C <45h>
64C16 <46h>
64C20 <4Ah>
64C2A <4Fh>
64C34 <50h>
64C3E <51h>
64C48 <52h>
64C52 <53h>
64C5C <54h>
64C66 <55h>
64C70 <56h>
64C7A <57h>
64C84 <5Bh>
64C8E <60h>
64C98 <61h>
64CA2 <62h>
64CAC <64h>
64CB6 <65h>
64CC0 <6Fh>
64CCA <70h>
64CD4 <71h>
64CDE <72h>
64CE8 <73h>
64CF2 <74h>
64CFC <75h>
64D06 <7Ah>
64D10 <80h>
64D1A <82h>
64D24 <83h>
64D2E <8Fh>
64D38 <91h>
64D42 <92h>
64D4C <9Ah>
64D56 <9Eh>
64D60 <9Fh>
64D6A <A0h>
64D74 <A1h>
64D7E <A2h>
64D88 <A5h>
64D92 <A6h>
64D9C <A7h>
64DA6 <A9h>
64DB0 <AAh>
64DBA <AEh>
64DC4 <B1h>
64DCE <BBh>
64DD8 <C0h>
64DE2 <CCh>
64DEC <D0h>
64DF6 <E1h>
64E00 <EAh>
64E0A <EEh>
64E14 <F0h>
64E1E <FDh>
64E28 <FFh>
64E32 <100h>
64E3C <102h>
64E46 <106h>
64E50 <107h>
64E5A <110h>
64E64 <111h>
64E6E <123h>
64E78 <124h>
64E82 <131h>
64E8C <132h>
64E96 <133h>
64EA0 <134h>
64EAA <135h>
64EB4 <136h>
64EBE <137h>
64EC8 <138h>
64ED2 <139h>
64EDC <13Ah>
64EE6 <13Bh>
64EF0 <13Dh>
64EFA <13Eh>
64F04 <151h>
64F0E <200h>
64F18 <205h>
64F22 <311h>
64F2C <411h>
64F36 <412h>
64F40 <444h>
64F4A <451h>
64F54 <452h>
64F5E <510h>
64F68 <511h>
64F72 <550h>
64F7C <610h>
64F86 <650h>
64F90 <700h>
64F9A <861h>

64FA4 <862h>
64FAE <865h>
64FB8 <86Eh>
64FC2 <A03h>
64FCC <A11h>
64FD6 <A12h>
64FE0 <A1Ah>
64FEA <A21h>
64FF4 <A22h>
64FFE <A2Ah>
65008 <A61h>
65012 <A62h>
6501C <A65h>
65026 <A6Eh>
65030 <AA1h>
6503A <AA2h>
65044 <AAAh>
6504E <C06h>
65058 <C07h>
65062 <C08h>
6506C <C0Ah>
65076 <C0Bh>
65080 <DFFh>
6508A <E00h>
65094 <70000h>
6509E <FFFFFh>
650A8 2.71828182846
650BD .5
650D2 -.5
650E7 10
650FC 180
65111 200
65126 360
6513B 400
69A97 "Radix' (Local Name)
69AAA "KeysOK?' (Local Name)
69AC1 "ExprLit' (Local Name)
69AD8 "BuffW' (Local Name)
69AEB "BuffH' (Local Name)
69AFE "SaveBlank' (Local Name)
69B19 "ManOp' (Local Name)
69B2C "nohalt' (Local Name)
69B41 "AppMode' (Local Name)
69B58 "NameGrob' (Local Name)
69B71 "EXITFCN' (Local Name)
69B88 "FontGauge' (Local Name)
69BA3 "LE' (Local Name)
69BB0 "LB' (Local Name)
69BBD "TE' (Local Name)
69BCA "FormEnvOK' (Local Name)
69BE5 "prow' (Local Name)
69BF6 "pcol' (Local Name)
69C07 "cursy' (Local Name)
69C1A "cursx' (Local Name)
69C2D "ttt' (Local Name)
69C3C "source' (Local Name)
69C51 "ofs' (Local Name)
69C60 "tok' (Local Name)
69C6F "rbv' (Local Name)
69C7E "idfflg' (Local Name)
69C93 "tmpop' (Local Name)
69CA6 "tmppdat' (Local Name)
69CBD "ploc' (Local Name)
69CCE "bv' (Local Name)
69CDB "unbound' (Local Name)
70200 Time Init
704EA key buffer
70551 pointer to menu display
7055B pointer to stack display
70565 pointer to PICT
7056F heap pointer
70574 saved B (return stack pointer)
70579 saved D1 (RPL stack pointer)
7057E bottom (start) of stack (grows down)
70583 local var ptr addr
70588 pointer to current loop information
7058D ?? menu keys ??
70592 begin HOME dir
70597 end HOME directory (points to 00000 prob for ATTACH)
7059C pointer to current dir
705AB pointer to alarm list
705B0 saved D0 (RPL thread pointer)
705B5 temp save A.A
705BA save arg ptrs (15 Nibbles)
705D9 ROM: ML Version Output
7065A Last RPL Token
7066E saved D (free stack space)
70673 Last Error Code
7069F stack size
706A4 Random Seed
706C5 RPL System Flags
706D5 RPL User Flags
706FD # arg ptrs @ 705BA???
706FF Save Last Err#
70713 GROB for displaying lines of stack. Enough mem for 19 char bitmap
70844 GROB for menu display: 8 by 131
70968 GROB for the rest of the screen: 56 by 131
710EC GROB for plot display: variable size
71AD8 ASCIC: Version
71AF6 ASCIC: Copyright
72000 Message table for library 000 (XLIB 0)
72281 Message table for library 00A (XLIB 10)
7232C Message table for library 00C (XLIB 12)
7260A Message table for library 00D (XLIB 13)
726A5 Message table for library 00B (XLIB 11)
72704 Message table for library 001 (XLIB 1)
72DCF Message table for library 002 (XLIB 2)
72F1E Message table for library 003 (XLIB 3)
72FE6 Message table for library 006 (XLIB 6)
736F9 Message table for library 005 (XLIB 5)
7427C Hash table for library 700 (XLIB 1792)
7448A Hash table for library 002 (XLIB 2)
7F9B5 'HOMEDIR' (Global Name)
7FED7 "A' (Local Name)

Quelle:
Derek S. Nickel
15 Maple Lane
Walnut Creek,
CA 94595-1718

Emulationskarte

für HP41C-Programme

Von Hewlett Packard ist inzwischen die Emulationskarte für HP41-Programme fertig.

Mit Hilfe dieser Karte kann man Programme, die vorher mit den im HP41 eingebauten Funktionen gelaufen sind, auf dem HP48SX emulieren, d.h. ablaufen lassen. Funktionen der Erweiterungsmodule werden nicht unterstützt.

Die einfachste Art und Weise, seine Programme vom HP41 auf den HP48 zu übertragen ist das Infrarotmodul, das es als Einsteckmodul für den HP41 gibt. Da der HP48SX ja bekanntlich über eine bidirektionale Infrarotschnittstelle verfügt, ist er in der Lage, Daten über seine Infrarotschnittstelle nicht nur zu senden, sondern diese auch zu empfangen. War diese Möglichkeit zuerst nur für den kabellosen Datenaustausch zwischen zwei HP48 gedacht, so benutzt HP jetzt diesen Weg auch für den HP41.

HP41-Programme können mit Hilfe der Emulatorkarte auch auf dem HP48 geschrieben werden. Zum Ablaufen auf dem HP48 werden diese dann dort compiliert und gestartet.

Funktionen des HP41CX, d.h. vor allem die Zeitfunktionen, sind nicht implementiert, man kann diese aber mit Funktionen des HP48 nachmachen, so zumindestens die Empfehlung von HP.

Die Funktionsumgebung der Emulatorkarte läßt sich um eigene Befehle und Funktionen ergänzen, sodaß man dringend benötigte Funktionen eines Einsteckmoduls notfalls selbst programmieren kann.

mm

Maschine&Sprache

Maschinensprache für den HP48

Teil I

von Georg Hoppen

Ich möchte mit diesem Artikel eine Reihe beginnen, die sich mit dem heiß ersehnten Thema Maschinensprache des HP48SX etwas näher auseinander setzt. Ich werde versuchen auch für Anfänger auf diesem Gebiet Verständliches zu Papier zu bringen, d.h. ich bin für Rückmeldungen bezüglich der Verständlichkeit meiner Ausführungen äußerst empfänglich.

Warum ein derartiges Projekt ?
Reicht es nicht für den normalen Anwender, sinnvolle Programme zu entwickeln oder Denkanstöße zu geben? Normalerweise ist hier ein klares Ja die Antwort, natürlich unter der Voraussetzung, daß dann mit der Masse der zu erwartenden Beiträge jede Neigung und jedes Bedürfnis abgedeckt wird.

Andererseits werden jedoch so ziemlich 70% des Rechners nicht genutzt, um nur einmal unabhängig von der reinen Geschwindigkeitssteigerung einen weiteren Vorteil zu betonen. Es können mittels der Programmierung auf Systemebene eben mehr Dinge miteinander verbunden werden.

Wie heißt es so schön:
Die Summe des Ganzen ist mehr als die Addition der Teile.

Auf diese Art und Weise kann auch der Normalanwender in den Genuß von neuen Befehlen kommen, die innerhalb eines Tools abgelegt werden können und in der Praxis als Makro zur Verfügung stehen. Wer hat nicht schon gemeckert, daß zwar interaktiv (in der Kommandozeile) Zeilen und Spalten einer Matrix gelöscht oder eingefügt werden können, dies aber innerhalb eines Programms unmöglich ist ?
Dies ist nun sehr einfach möglich, da nun die Einsprungadresse der Systemroutine zur Verfügung steht (Liste der SYSEVAL-Adressen).

Bei der Entwicklung der einzelnen Tools (Werkzeuge) werde ich großen Wert darauf legen, daß jede Hard- und Softwareumgebung berücksichtigt wird, so daß jeder in den Genuß der einzelnen Routinen kommen kann.
Wünschenswert wäre aber in jedem Fall das Programmpaket aus der Toollibrary von Donelly. Die Besitzer des Assemblers sind natürlich am besten dran, da hiermit am schnellsten und komfortabelsten Maschinencode erzeugt werden kann.

Gerade deshalb, weil kein Buch oder sonstige Anleitungen auf dem Markt sind, die dieses Projekt unterstützen könnten, bin ich auf Euer Feedback (Briefe, Telefon etc.) angewiesen, um praktikable Wünsche und Anregungen ständig mit einfließen zu lassen.

Geplant ist es, einen Datenbrowser, eine Datenbank und Entwicklungswerkzeuge zu erstellen, die kürzere Routinen für jeden Zweck zur Verfügung stellen.
Als Entwicklungsumgebung dient die vorgestellte Workspace aus PRISMA 1/91, Seite 19, die irrtümlich die Überschrift "Datenkomprimierung und Archivierung auf dem HP48SX" trug.
Grundbausteine werden in dem übergeordneten DIR gelagert und sind so für jeden Zweck verfügbar. In der Arbeitsumgebung kann nach Herzenslust getestet werden, ohne bestehende Projekte zu gefährden.

Anmerkung zum Workspace:
Mich haben verschiedene Leute angerufen, daß die vorgestellte Workspace in Teilen unverständlich war. Eine Überprüfung des zugegebenermaßen schlechten Drucks ergab, daß einige Variablen tatsächlich sehr versteckt angegeben sind.
Neben den eigentlichen Programmen ist jedoch eine Liste abgedruckt, die den Aufbau der Variablen mittels des Programms PGDIR (Ralf Pfeiffer, älterer Pas-de-deux Artikel) darstellt. Vor den einzelnen Variablen steht jeweils der Objekttyp.
Dabei sind die wichtigsten Bezeichnungen nochmal kurz erklärt:
5 bedeutet Liste, 8 Programm und 15 ist ein Sub-DIR.

Der Inhalt von GUTIL und SUTIL steht in dem Stackausdruck ziemlich gequetscht unten auf der Seite 22 im PRISMA 1/91. Dies ließ sich aus drucktechnischen Gründen offensichtlich nicht anders gestalten. Trotzdem werde ich mich bemühen, an Hand von Beispielen den grundsätzlichen Aufbau von weiteren Programmen deutlicher zu gestalten. Insofern also herzlichen Dank für die Anregung.

Der erste Teil ist bewußt kurz gehalten, um die künftige Wegrichtung nicht allzusehr einzuschränken. Der Umgang mit SYSEVAL-Befehlen muß sicherlich gelernt werden und bedeutet daher stets eine zusätzliche Belastung. Demgegenüber steht aber ein letztlich nicht überschaubarer Gewinn.
Die Freude an Neuem, der Austausch von Ideen mit Gleichgesinnten, neue Kontakte und tieferes Verständnis der dem Rechner zugrunde liegenden Operationen. Vielleicht kann ja der eine oder andere schon einmal versuchen mit Hilfe der bereits vorgestellten Einsprungadressen das eine oder andere Kurzprogramm umzustellen und das Ergebnis zu vergleichen.
Ich kann Euch versprechen, daß

die ersten Erfolge süchtig machen, es geht demnächst immer schneller und kürzer ...

Also bis zum nächsten Mal, dann werde ich die neue Speichertechnik in lokale und globale Variablen vorstellen und das eine oder andere zusätzliche Werkzeug mit anbieten, um Euch die Arbeit zu erleichtern.

Nur Mut, wenn jemandem Ideen kommen, was noch unbedingt vorgestellt werden sollte. Ich helfe gerne aus, um den Ideen zur Verwirklichung zu verhelfen. Auch ich mache sicherlich Fehler, die dann der nächste korrigiert und letztlich hat jeder ein lauffähiges Produkt, an dem alle ihre Freude haben können.

Georg Hoppen
Hubertusring 5
4512 Wallenhorst

Saturn Instruction Set

Befehlssatz des Prozessors im HP71/28/48

Im folgenden folgt eine Liste der Befehle des Mikroprozessors, der seit den Zeiten des HP71B bei Hewlett Packard als universeller Rechenknecht für die "großen" Taschenrechner verwendet wird.
Diese Liste hatte Derek S. Nickel aus den USA zusammengestellt, sie soll hier in erster Linie als Referenz abgedruckt sein, damit sich Autoren, die in Maschinensprache programmieren, für ihre Erklärungen auf diese Quelle beziehen können.
Auch für Georg Hoppen ist dies der Anfang für seine Artikelserie ***Maschine&Sprache***, die einzelnen Befehle werden in den nächsten Ausgaben dann noch im Detail erklärt, dies würde den Rahmen einer PRISMA Ausgabe doch erheblich sprengen.
Bitte habt also noch ein klein wenig Geduld, kommt Zeit, kommt Rat, kommt Erklärung...

mm

Globale Einteilungen:

```
0...    * misc operations (RTNs, modes,
          stack, status, P, logic)

1...    * data movement/loading (D0,D1,
          R0-R4)

2n      P=      n

3xn..n  LC(m)   n..n (x = m - 1)
3mc..c  LCASC   \A..A\    (m = 2*charac-
        ters - 1)
3nh..h  LCHEX   h..h (n = digits - 1)

400     RTNC
4aa     GOC     label
420     NOP3

500     RTNNC
5aa     GONC    label
6aaa    GOTO    label
6300    NOP4
64000   NOP5

7aaa    GOSUB   label
8...    * misc, tests and branching
9...    * tests w/fs

Aax     * math and assignment w/fs
Abx     * math and assignment w/fs
Bax     * math and assignment w/fs
Bbx     * math and assignment w/fs

Cx      * math and assignment w/A
Dx      * math and assignment w/A
Ex      * math and assignment w/A
Fx      * math and assignment w/A
```

Befehlsgruppe 0

```
00  RTNSXM
01  RTN
02  RTNSC
03  RTNCC
04  SETHEX
05  SETDEC
06  RSTK=C
07  C=RSTK
08  CLRST
09  C=ST
0A  ST=C
0B  CSTEX
0C  P=P+1
0D  P=P-1
0Exx     * logic operations
0F  RTI

-- field ---
 fs  A
------------
0Ea0    0EF0    A=A&B  field
0Ea1    0EF1    B=B&C  field
0Ea2    0EF2    C=C&A  field
0Ea3    0EF3    D=D&C  field
0Ea4    0EF4    B=B&A  field
0Ea5    0EF5    C=C&B  field
0Ea6    0EF6    A=A&C  field
0Ea7    0EF7    C=C&D  field

0Ea8    0EF8    A=A!B  field
```

```
0Ea9    0EF9    B=B!C  field
0EaA    0EFA    C=C!A  field
0EaB    0EFB    D=D!C  field
0EaC    0EFC    B=B!A  field
0EaD    0EFD    C=C!B  field
0EaE    0EFE    A=A!C  field
0EaF    0EFF    C=C!D  field

fs: P  WP XS X  S  M  B  W
a:  0  1  2  3  4  5  6  7
```

Befehlsgruppe 10, 11, 12

```
100 R0=A
101 R1=A
102 R2=A
103 R3=A
104 R4=A
105
106
107
108 R0=C
109 R1=C
10A R2=C
10B R3=C
10C R4=C
10D
10E
10F

110 A=R0
111 A=R1
112 A=R2
113 A=R3
114 A=R4
115
116
117
118 C=R0
119 C=R1
11A C=R2
11B C=R3
11C C=R4
11D
11E
11F

120 AR0EX
121 AR1EX
122 AR2EX
123 AR3EX
124 AR4EX
125
126
127
128 CR0EX
129 CR1EX
12A CR2EX
12B CR3EX
12C CR4EX
12D
12E
12F
```

Befehlsgruppe 13 bis 1F

```
130 D0=A
131 D1=A
132 AD0EX
133 AD1EX

134 D0=C
135 D1=C
136 CD0EX
137 CD1EX

138 D0=AS
139 D1=AS
13A AD0XS
13B AD1XS

13C D0=CS
13D D1=CS
13E CD0XS
13F CD1XS

---------- field -----------
 A   B   fs  d
-----------------------------
140 148 150a    158x    DAT0=A field
141 149 151a    159x    DAT1=A field
142 14A 152a    15Ax    A=DAT0 field
143 14B 153a    15Bx    A=DAT1 field
144 14C 154a    15Cx    DAT0=C field
145 14D 155a    15Dx    DAT1=C field
146 14E 156a    15Ex    C=DAT0 field
147 14F 157a    15Fx    C=DAT1 field

16x     D0=D0+ n
17x     D1=D1+ n
18x     D0=D0- n
19nn    D0=(2) nn
19hh    D0=HEX hh
1Annnn  D0=(4) nnnn
1Ahhhh  D0=HEX hhhh
1Bnnnnn D0=(5) nnnnn
1Bhhhhh D0=HEX hhhhh
1Cx     D1=D1- n
1Dnn    D1=(2) nn
1Dhh    D1=HEX hh
1Ennnn  D1=(4) nnnn
1Ehhhh  D1=HEX hhhh
1Fnnnnn D1=(5) nnnnn
1Fhhhhh D1=HEX hhhhh

fs: P  WP XS X  S  M  B  W
a:  0  1  2  3  4  5  6  7
```

```
x = d - 1        x = n - 1
d = x + 1        n = x + 1
```

Befehlsgruppe 80, 81

```
800  OUT=CS
801  OUT=C
802  A=IN
803  C=IN
804  UNCNFG
805  CONFIG
806  C=ID
807  SHUTDN
8080 INTON
808C PC=(A)
808F INTOFF
809  C+P+1
80A  RESET
80B  BUSCC
80Cn C=P    n
80Dn P=C    n
80E  SREQ?
80Fn CPEX   n

810 ASLC
811 BSLC
812 CSLC
813 DSLC
814 ASRC
815 BSRC
816 CSRC
817 DSRC
818
819
81A
81B
81C ASRB
81D BSRB
81E CSRB
81F DSRB
```

Befehlsgruppe 82 bis 8F, 9

```
82n no opcode, clear hardware status
    w/n as bit mask
821 XM=0
822 SB=0
824 SR=0
828 MP=0
82F CLRHST

831yy   ?XM=0
832yy   ?SB=0
834yy   ?SR=0
838yy   ?MP=0

84n     ST=0   n
85n     ST=1   n
86nyy   ?ST=0  n    ?ST#1  n
87nyy   ?ST=1  n    ?ST#0  n
88nyy   ?P#    n
89nyy   ?P=    n
8Axyy   * tests w/A (equal and not equal)
8Bxyy   * tests w/A (inequality)
8Caaaa  GOLONG label
8Daaaaa GOVLNG label
8Eaaaa  GOSUBL label
8Faaaaa GOSBVL label
9zxyy   * tests w/fs

TEST00  RTNYES
TESTyy  GOYES  label
```

Befehlsgruppe 8A, 8B, 9

```
--- field ----
  A      fs
--------------
8A0yy  9a0yy  ?A=B  field   ?B=A field
8A1yy  9a1yy  ?B=C  field   ?C=B field
8A2yy  9a2yy  ?A=C  field   ?C=A field
8A3yy  9a3yy  ?C=D  field   ?D=C field
8A4yy  9a4yy  ?A#B  field   ?B#A field
8A5yy  9a5yy  ?B#C  field   ?C#B field
8A6yy  9a6yy  ?A#C  field   ?C#A field
8A7yy  9a7yy  ?C#D  field   ?D#C field
8A8yy   9a8yy   ?A=0   field
8A9yy   9a9yy   ?B=0   field
8AAyy   9aAyy   ?C=0   field
8AByy   9aByy   ?D=0   field
8ACyy   9aCyy   ?A#0   field
8ADyy   9aDyy   ?B#0   field
8AEyy   9aEyy   ?C#0   field
8AFyy   9aFyy   ?D#0   field

8B0yy   9b0yy   ?A>B   field
8B1yy   9b1yy   ?B>C   field
8B2yy   9b2yy   ?C>A   field
8B3yy   9b3yy   ?D>C   field
8B4yy   9b4yy   ?A<B   field
8B5yy   9b5yy   ?B<C   field
8B6yy   9b6yy   ?C<A   field
8B7yy   9b7yy   ?D<C   field
8B8yy   9b8yy   ?A>=B  field
8B9yy   9b9yy   ?B>=C  field
8BAyy   9bAyy   ?C>=A  field
8BByy   9bByy   ?D>=C  field
8BCyy   9bCyy   ?A<=B  field
8BDyy   9bDyy   ?B<=C  field
8BEyy   9bEyy   ?C<=A  field
8BFyy   9bFyy   ?D<=C  field
fs: P  WP XS X  S  M  B  W
a:  0  1  2  3  4  5  6  7
b:  8  9  A  B  C  D  E  F
```

Befehlsgruppe A, C, D

```
-- field --
A   fs
-----------
```

```
C0  Aa0 A=A+B  field
C1  Aa1 B=B+C  field
C2  Aa2 C=C+A  field
C3  Aa3 D=D+C  field
C4  Aa4 A=A+A  field
C5  Aa5 B=B+B  field
C6  Aa6 C=C+C  field
C7  Aa7 D=D+D  field
C8  Aa8 B=B+A  field
C9  Aa9 C=C+B  field
CA  AaA A=A+C  field
CB  AaB C=C+D  field
CC  AaC A=A-1  field
CD  AaD B=B-1  field
CE  AaE C=C-1  field
CF  AaF D=D-1  field

D0  Ab0 A=0    field
D1  Ab1 B=0    field
D2  Ab2 C=0    field
D3  Ab3 D=0    field
D4  Ab4 A=B    field
D5  Ab5 B=C    field
D6  Ab6 C=A    field
D7  Ab7 D=C    field
D8  Ab8 B=A    field
D9  Ab9 C=B    field
DA  AbA A=C    field
DB  AbB C=D    field
DC  AbC ABEX   field
DD  AbD BCEX   field
DE  AbE ACEX   field
DF  AbF CDEX   field
```

```
fs: P  WP XS X  S  M  B  W
a:  0  1  2  3  4  5  6  7
b:  8  9  A  B  C  D  E  F
```

Befehlsgruppe B, E, F

```
-- field ---
A   fs
------------
E0  Ba0 A=A+B  field
E1  Ba1 B=B+C  field
E2  Ba2 C=C+A  field
E3  Ba3 D=D+C  field
E4  Ba4 A=A+1  field
E5  Ba5 B=B+1  field
E6  Ba6 C=C+1  field
E7  Ba7 D=D+1  field
E8  Ba8 B=B-A  field
E9  Ba9 C=C-B  field
EA  BaA A=A-C  field
EB  BaB C=C-D  field
EC  BaC A=B-A  field
ED  BaD B=C-B  field
EE  BaE C=A-C  field
EF  BaF D=C-D  field
```

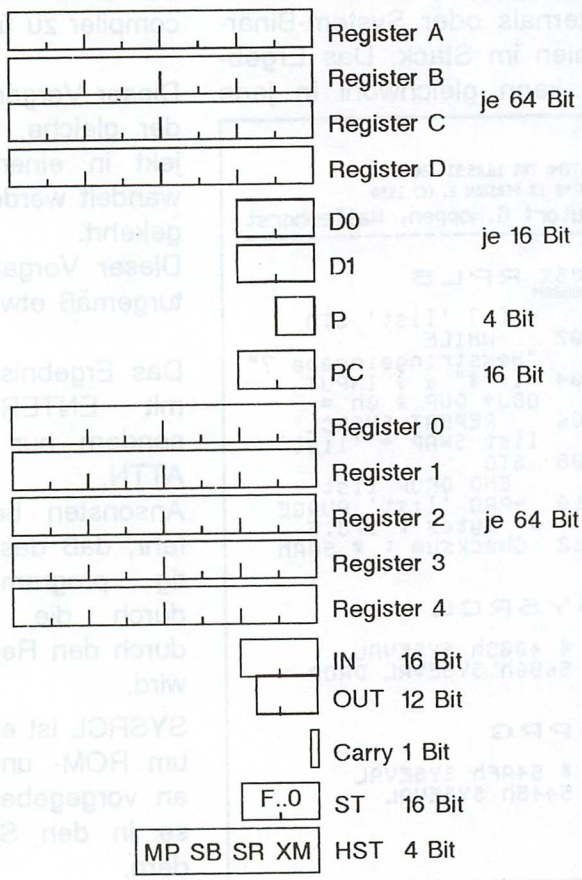
```
F0  Bb0 ASL    field
F1  Bb1 BSL    field
F2  Bb2 CSL    field
F3  Bb3 DSL    field
F4  Bb4 ASR    field
F5  Bb5 BSR    field
F6  Bb6 CSR    field
F7  Bb7 DSR    field
F8  Bb8 A=-A   field
F9  Bb9 B=-B   field
FA  BbA C=-C   field
FB  BbB D=-D   field
FC  BbC A=-A-1 field
FD  BbD B=-B-1 field
FE  BbE C=-C-1 field
FF  BbF D=-D-1 field
```

```
fs: P  WP XS X  S  M  B  W
a:  0  1  2  3  4  5  6  7
b:  8  9  A  B  C  D  E  F
```

Speicherpaltzanforderung

```
00..0  BSS     expr
nn..n  CON(m)  expr
nn..n  REL(m)  expr
zy..a  NIBASC  'chars'
zy..a  NIBASC  \chars\
hh..h  NIBHEX  h..hh
```

Registersatz des Saturn-Prozessors



HP48 RPL-Builder
oder Systembaukasten
von Georg Hoppen

Als Grundidee diente hier das Sammeln der verschieden Objekte in der richtigen Reihenfolge innerhalb einer Liste. Diese wird dann aufgelöst (interner Befehl OBJ→) und dann zu zu einem Programmobjekt umgeformt.

Als Besonderheit sei erwähnt, daß hier dann die Begrenzungszeichen "«" bzw. "»" fehlen, damit wenigsten diese überflüssigen 10 Byte gespart werden können und das Objekt im Stack nicht mehr aus Versehen editiert werden kann.
Es stehen dann nur noch Externals oder System-Binärzahlen im Stack. Das Ergebnis kann gleichwohl in jede Variable gespeichert werden und versieht dann seinen vorgesehenen Zweck.

```
DATUM: THU 16.05.91  20:26:03
HP 48 SX VERSION E.(O 1990
Autor: G.Hoppen, Wallenhorst
------------------------

48SX RPLB
PROGRAM
    « { } 'list' STO
002   WHILE
   "Hexstringeingabe ?"
004  { "#" α } INPUT
   OBJ→ DUP # 0h ≠
006    REPEAT SYSRCL
   list SWAP + 'list'
008  STO
     END DROP list
010  →PRG 'list' PURGE
   »  Bytes : 173.5
012  Checksum : # 5AAh

SYSRCL

« # 4003h SYSEVAL
# 56B6h SYSEVAL DROP
»

→PRG

« # 54AFh SYSEVAL
# 5445h SYSEVAL
»
```

Wenn jemand trotzdem das Ergebnis mit VISIT betrachten will, hier noch ein kurzer Hinweis:
Trotz eines scheinbar geringen Byteumfangs der Objekte kann es manches Mal sehr lange dauern, bis das Objekt tatsächlich angezeigt wird. Es kann ja z.B. einer Bytemenge von 40-50 Byte auch die gleiche Anzahl von Externals zugeordnet sein.
Diese versucht der Rechner erst mit einem internen Decompiler zu übersetzen.

Dieser Vorgang ist im Prinzip der gleiche, als ob ein Objekt in einen String umgewandelt werden soll und umgekehrt.
Dieser Vorgang benötigt naturgemäß etwas Zeit.

Das Ergebnis läßt sich nicht mit ENTER abschließen, sondern nur mit der Taste ATTN.
Ansonsten besteht die Gefahr, daß das eigentlich richtig programmierte Objekt durch die Decompilierung durch den Rechner verändert wird.

SYSRCL ist eine Kurzroutine, um ROM- und RAM-Objekte an vorgegebener Hex-Adresse in den Stack zu befördern.

→PRG ist der interne Befehl erweitert auf eine vorherige Listenauflösung. Irgendwo müssen nämlich die Objekte gesammelt werden, um dann ein Programm bilden zu können. Es wäre auch über den Stack gegangen, so aber kann man auch einmal zwischendurch etwas anderes im Stack machen.

Wer normale Stringaufrufe für Prompt und Input-Befehle ebenfalls in die neue Programmierung mit einbinden möchte, der muß das Programm entsprechend modifizieren, d.h. z.B. die RPL-Schleife selbständig ablaufen lassen. Bei der Eingabe von #0h wird dann nicht sofort in ein Programm umgewandelt, sondern alternativ normaler USERCODE erzeugt und in die Liste abgespeichert.
Dabei ist aber sicher zu stellen, daß die Reihenfolge der Objekte wie in der RPL-Schleife abgespeichert wird, d.h. gemäß der Reihenfolge Liste SWAP +.

Dadurch wird sichergestellt, daß immer an das Ende der Liste gespeichert wird.

Natürlich ließe sich auf diese Weise auch eine Liste mit USERCODE einfügen.

Georg Hoppen
Hubertusring 5
4512 Wallenhorst

Formeln und Gleichungsgruppen

von Michael August

Liebe Clubmitglieder,
da ich mich bisher noch nicht aktiv am Clubleben bzw. an der Gestaltung von PRISMA beteiligt habe, möchte ich mich zuerst einmal vorstellen.
Mein Name ist Michael August, Jahrgang '59. Von Beruf bin ich Dipl. Ing. Fahrzeugbau und zur Zeit als Konstrukteur bei einem Nutzfahrzeug-Aufbauhersteller tätig.
Mein Einstieg als HP-Anwender erfolgte auf Grund der Qualitätsunterschiede zu japanischen Rechnern über den HP-11C; darauf kam der HP41CX und jetzt schließlich der HP48SX.
Da ich mit der Programmierung des HP-48SX ziemliche Schwierigkeiten hatte, beim 41'er war's leichter gewesen , suchte ich nach einer einfachen Möglichkeit Formeln und Gleichungsgruppen zu lösen.
Formeln lassen sich gut mit Hilfe der eingebauten Funktion SOLVE lösen. Durch den Kauf der Gleichungslöserkarte ist es mir nun auch möglich, Gleichungsgruppen nach einem ähnlichem Schema auszuwerten. Vorteil der SOLVE-Funktion bzw. des Gleichungslösers ist die einfache Aufstellung eines Gleichungssystems.
Ein großer Nachteil ist die geringe Ausführungsgeschwindigkeit. Wenn mir jemand Tips zur Geschwindigkeitssteigerung geben könnte wäre ich sehr erfreut.
Da es unter Euch sicherlich Leute gibt, die mit der Elektronik des Rechners besser vertraut sind als ich, wären ich und bestimmt viele andere Mitglieder dankbar, wenn es gelingen würde den HP48SX mit einem Akku auszurüsten, um die häufigen Batteriewechsel auch im Hinblick auf den Umweltschutz einzusparen.
Nun aber zurück zum eigentlichen Thema.
Ich hoffe, daß der eine oder andere mit den folgenden Programmen etwas anfangen kann.
Die Menütasten des VAR-Menüs sind mit den Formeln bzw. den Gleichungsgruppen belegt. Um die Ausführung mit nur einem Tastendruck zu starten, waren zwei kleine Hilfsprogramme erforderlich, die im Hauptverzeichnis HOME abgelegt sind (↓GLE, ↓FOR). Das Programm ↓FOR dient der Ausführung einer Formel, das Programm ↓GLE der Ausführung einer Gleichungsgruppe, was nur mit der Gleichungslöserkarte möglich ist. Beide Programme definieren die ON-Taste um, damit nach Ablauf der Berechnungen ein festgelegter Rechnerzustand hergestellt und die erzeugten Variablen zwecks Erhaltung der Speicherkapazität gelöscht werden. Zu diesem Zweck wird eine Liste der für die Formeln bzw. Gleichungsgruppen benötigten Variablen in 'VARIA' abgelegt.
'VARIA' wird beim ersten Ausführen erzeugt. Nach Beendigung werden 'VARIA', die Definition der ON-Taste und der USER-Modus wieder gelöscht.
Im Programm ↓GLE tauchen die Befehle R↓ und R↑ auf, die einigen von Euch bestimmt vom 41'er her bekannt sind. Ich persönlich kann darauf nicht verzichten.

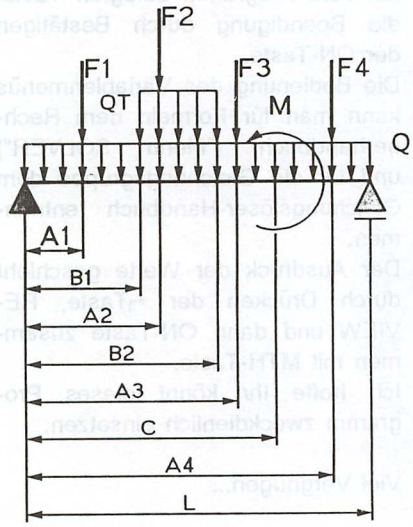
Die Bereitstellung einer Formel erfolgt nach folgendem Schema:

```
« { 'Gleichung' { Liste
der Variablennamen } }
↓FOR »
```

Der Aufbau einer Gleichungsgruppe erfolgt folgendermaßen:

```
« { 'Gleichung 1' 'Glei-
chung 2' usw. } "Benen-
nung" { Liste der Varia-
blennamen } ↓GLE »
```

Beispiel für eine Formel ist das Programm MBX, mit dem es möglich ist das Biegemoment MX für einen einfach gelagerten Träger mit den gezeigten Belastungen zu berechnen, bzw. mit der PLOT-Funktion darzustellen. Die Formel ist eine Erweiterung der Gleichung "Simple Moment Nr. 1.5" aus der Gleichungsbibliothek.



Träger

L	Stützlänge des Trägers
Q	Gleichstreckenlast
A1 - A4	Abstand der Punktlasten
F1 - F4	Punktlasten
B1	Abstand zum Beginn der Gleichstreckenteillast
B2	Abstand zum Ende der Gleichstreckenteillast
QT	Gleichstreckenteillast
C	Abstand zum Moment
M	Moment
X	Abstand zum Biegemoment (Schnittstelle)
MX	Biegemoment an Stelle X

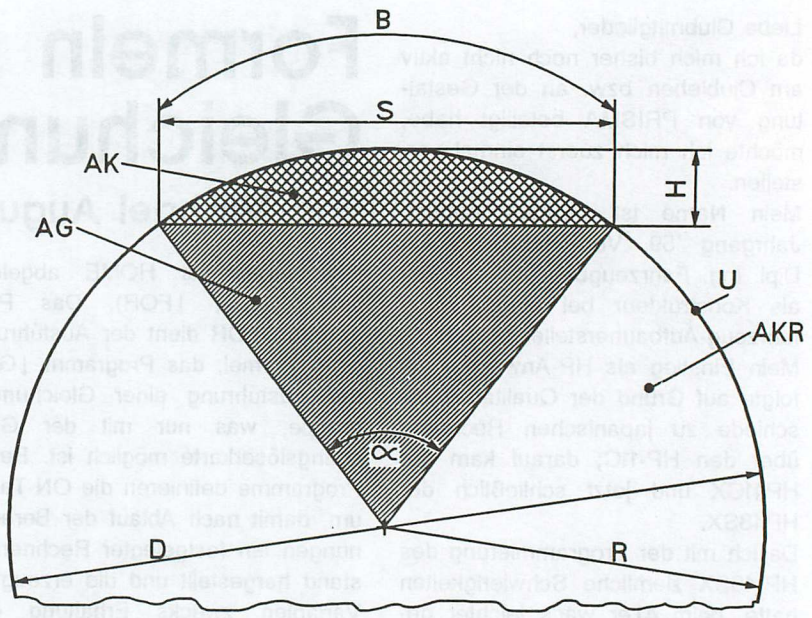
Achtung, da die Formel relativ lang ist, dauert es eine Weile, bis das Variablenmenü bereitgestellt wird.
Beispiel für eine Gleichungsgruppe ist das Programm KREIS, mit dem Daten eines Kreises, Kreisausschnitts und Kreisabschnitte berechnet werden können.

D	Kreisdurchmesser
R	Kreisradius
B	Bogenlänge Kreiausschnitt bzw. Kreisabschnitt
S	Sehnenlänge Kreisabschnitt
a	Winkel Kreisausschnitt bzw. Kreisabschnitt

H	Höhe Kreisabschnitt
AKR	Kreisfläche
U	Kreisumfang
AG	Fläche Kreisausschnitt
AK	Fläche Kreisabschnitt

Der Start erfolgt durch Drücken der mit dem Programm belegten Taste, die Beendigung durch Bestätigen der ON-Taste.
Die Bedienung des Variablenmenüs kann man für Formeln dem Rechnerhandbuch ("Menu SOLVER") und für die Gleichungsgruppe dem Gleichungslöser-Handbuch entnehmen.
Der Ausdruck der Werte geschieht durch Drücken der ↰Taste, REVIEW und dann ON-Taste zusammen mit MTH-Taste.
Ich hoffe Ihr könnt dieses Programm zweckdienlich einsetzen.

Viel Vergnügen...



Kreis

↓GLE

```
HP-48SX
MON 18.03.91  18:35:56

↓GLE
« R↓ R↓ STEQ MINIT R↑ R↑
DUP 'VARIA' STO
  « 1 VARIA SIZE
    FOR I VARIA I GET
DUP
      IF TYPE 6 SAME
      THEN PURGE
      ELSE DROP
      END
    NEXT 91 DELKEYS
'VARIA' PURGE -62 CF 0
MENU
  » 91 ASN -62 SF MITM
MSOLVR
»

# 44658d
227
```

↓FOR

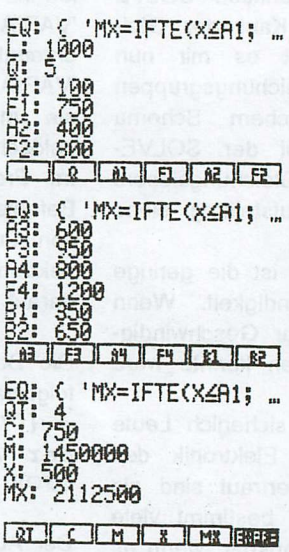
```
HP-48SX
MON 18.03.91  18:35:21

↓FOR
« DUP 2 GET 'VARIA' STO
  « 1 VARIA SIZE
    FOR I VARIA I GET
DUP
      IF TYPE 6 SAME
      THEN PURGE
      ELSE DROP
      END
    NEXT 91 DELKEYS
'VARIA' PURGE -62 CF 0
MENU
  » 91 ASN -62 SF STEQ
30 MENU
»

# 54983d
206,5
```

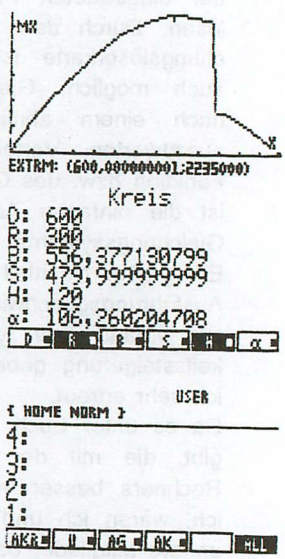
R↓ und R↑

```
HP-48SX
MON 18.03.91  18:36:45

R↓
« DEPTH ROLLD
»

# 44430d
21,5

HP-48SX
MON 18.03.91  18:37:10

R↑
« DEPTH ROLL
»

# 52170d
21,5
```

MBX

```
HP-48SX
MON 18.03.91  18:34:07

MBX
« { 'MX=IFTE(X≤A1;F1*(L-
A1)*X/L;F1*A1*(L-X)/L)+
IFTE(X≤A2;F2*(L-A2)*X/L;
F2*A2*(L-X)/L)+IFTE(X≤A3
;F3*(L-A3)*X/L;F3*A3*(L-
X)/L)+IFTE(X≤A4;F4*(L-A4
)*X/L;F4*A4*(L-X)/L)+
IFTE(X≤C;M*X/L;-M*(L-X)/
L)+QT*X*(B2-B1+(B1^2-B2^
2)/2/L)-IFTE(X≥B1 AND X≤
B2;QT/2*(X-B1)^2;0)-IFTE
(X≥B2;QT*(X*(B2-B1)+(B1^
2-B2^2)/2);0)+Q*X/2*(L-X
)' { L Q A1 F1 A2 F2 A3
F3 A4 F4 B1 B2 QT C M X
MX } ) ↓FOR
»

# 34536d
860
```





KREIS

```
HP-48SX
MON 18.03.91  18:33:40

KREIS
« { 'D=2*R' 'AKR=π*R^2'
'U=2*π*R' 'AG=π/360*R^2*
α' 'AK=R^2/2*(π/180*α-
SIN(α))' 'B=π/180*α*R' '
S=2*R*SIN(α/2)' 'H=R*(1-
COS(α/2))' } "Kreis" { D
R B S H α AKR U AG AK }
↓GLE
»

# 13469d
395,5
```

Michael August
Weidornweg 6a
2056 Glinde

Maschinensprache auf dem HP28S

1. Teil: Grundlagen

von Peter Röhl

1. Aufbau der Programme

Die HP28S-Programme setzen sich aus einer Liste von Adressen zusammen. Jedem der HP28S-Befehle ist eine feste ROM-Adresse zugeordnet.
(Das Handbuch zum Rechner unterscheidet zwischen Befehlen, Funktionen und analytischen Funktionen. Hier werden diese drei Typen der Einfachheit halber jedoch einfach 'Befehl' genannt.)
Beispiel: ADDIERE

```
«
  + 1 DISP
»
```

Die Befehle dieses Programms untereinander geschrieben und rechts daneben ihre Adressen angefügt ergibt folgendes Bild:

Befehl	Adresse	Wert im Speicher (RAM/ROM)
	#2C67h	76C20
«	#E9D0h	0D9E0
+	#8EEEh	EEE80
1	#112C9h	9C211
DISP	#A5A6h	6A5A0
»	#E9E6h	6E9E0
	#2F90h	09F20

Tabelle 1

Dem Befehl "«" ist also die Adresse #E9D0h zugeordnet. Entsprechend ist "+" die Adresse #8EEEh zugeordnet.
Also steht in der Adresse #8EEEh die Information, daß gegebenenfalls eine Addition auszuführen ist.
Anstelle von "+" könnte also auch #8EEEh SYSEVAL ausgeführt werden.

In der Tabelle fällt auf:

a.) Die dritte Spalte. Erklärung folgt weiter unten.

b.) Vor dem "«" scheint noch etwas zu stehen, das unsichtbar ist. Es handelt sich dabei um die Markierung für den Programmanfang, sozusagen um ein BEGIN.
(In vielen Sprachen markiert BEGIN den Anfang einer eine Einheit bildende Anzahl von Befehlen, z.B. in Pascal.)
Entsprechend befindet sich am Ende ein ebenfalls unsichtbares END, das diese Einheit (nämlich das Programm) abschließt.

Dieses BEGIN und das END halten also die einzelnen Befehle zusammen, machen aus ihnen eine Einheit, nämlich das Programm.
Zwischen BEGIN und END sind die Adressen der einzelnen Befehle aufgelistet. Deshalb kann man hier auch von einer Liste sprechen.
In der Tat ist eine Liste ganz ähnlich aufgebaut, nämlich wie ein Programm. Würde man die Adresse #2C67h in #2A96h ändern, so verwandelt sich dadurch das Programm in eine Liste und man kann mit den Befehlen aus dem LISTMenü die einzelnen Objekte z.B. aus der Liste holen!
Die Adresse #2C67h definiert also den Anfang eines Programms.
Weil ein Programm ein Objekt ist, wird diese Adresse als objektdefinierende Adresse eines Programms bezeichnet. Unterscheiden sich verschiedene Objekttypen nicht in ihrem Aufbau, sondern nur durch ihre objektdefinierende Adresse, so kann man einfach durch die Änderung der Adressangabe im Objekt das Objekt von einem Objekttyp in den anderen konvertieren.

Beispiel:
Das Programm P→L wandelt die Adresse #2C67h in #2A96h. Gibt man also obiges Programm ein und führt P→L aus, so steht eine Liste im Stack.

```
1: { « + 1 DISP » }
```

(dies ist eine Liste mit 5 Elementen)
siehe hierzu Tabelle 2
4 GET liefert `1: DISP`

Beachte:
#2F90 (END) schließt nicht nur Programme, sondern auch Listen ab und ist dann als "}" sichtbar.
Nebenbei: Auch algebraische Ausdrücke werden damit abgeschlossen.
Hätte man hingegen einfach 1 →STR anstelle von P→L eingegeben, so wäre folgendes Objekt entstanden:

Befehl	Adresse	Wert im Speicher (RAM/ROM)
{	#2C96h	69C20
«	#E9D0h	0D9E0
+	#8EEEh	EEE80
1	#112C9h	9C211
DISP	#A5A6h	6A5A0
»	#E9E6h	6E9E0
}	#2F90h	09F20

Tabelle 2

Befehl	Adresse	Wert im Speicher (RAM/ROM)
{	#2C96h	69C20
	#2C67	76C20
«	#E9D0h	0D9E0
+	#8EEEh	EEE80
1	#112C9h	9C211
DISP	#A5A6h	6A5A0
»	#E9E6h	6E9E0
	#2F90h	09F20
}	#2F90h	09F20

Tabelle 3

Jetzt erkennt man auch den Sinn von BEGIN und END in einem Programm:
BEGIN und END kapseln das Programm, d.h. sie halten die einzelnen Befehle (besser: Objekte) zusammen. Diese Liste enthält dann auch nur ein Objekt, nämlich das Programm.
Die Programmzeichen "«" und "»" braucht nur der Editor zur Erkennung, daß ein Programm eingege-

ben wurde. Für die Funktionsfähigkeit der Programme sind sie bedeutungslos.
Nun lassen sich in einem Programm eine ganze Menge verschiedener Objekttypen einbinden. Deshalb nun eine Auflistung all derer, die der HP28S kennt.

2. Objektdefinierende Adressen

Adresse	Objekttyp	Art	Kommentar
#2911h	SHORT INTEGER	1	11920zyxwv mit #vwxyz in Hex
#2933h	REAL	1	normale Fließkommazahlen
#2955h	LONG REAL	1	16 Stellen Mantisse
#2977h	COMPLEX	1	normale Fließkommazahlen
#299Dh	LONG COMPLEX	1	16 Stellen Mantisse
#29BFh	BYTE	1	
#29E1h	ARRAY, not used	2	warum, das wissen die Götter
#2A0Ah	ARRAY	2	
#2A2Ch	ARRAY, not used	2	warum, das wissen die Götter
#2A4Eh	STRING	3	
#2A70h	INTEGER	3	Binary Integer
#2A96h	LIST	4	
#2AB8h	RAM/ROM PAIR	?	für Directories
#2ADAh	ALGEBRAIC	4	algebraische Ausdrücke
#2C67h	RPL-PROGRAM	4	
#2C96h	INLINE-MCODE	3	
#2D12h	GLOBAL NAME	5	
#2D37h	LOCAL NAME	5	
#2D5Ch	ROM POINTER	?	einzelne Befehle
#2F90h	END		

Tabelle 4

Art

1 Der objektdefinierenden Adresse folgt direkt der Wert.
Eine Längenangabe des Objektes ist unnötig, da die Länge des jeweiligen Objektes fest durch den Objekttyp vorgegeben ist.
Beispiel: 1192080000 ist der SHORT INTEGER mit dem Wert 8;
339200000000000000080 ist eine reelle Zahl mit dem Wert 8.

2 Arrays sind wie folgt aufgebaut:
ARRAY NIBBLES TYP DIM ZAHL1 ZAHL2 ...
mit *ARRAY* = A0A20
NIBBLES = fünfstelliger Wert, der die Gesamtlänge des Objekt minus 5 in Nibbles angibt. (Jede Hexadezimalziffer belegt im Speicher 4 Bits. 4 Bits = 1 Nibble = 1 Digit).
Wie immer ist der Wert mit vertauschter Ziffernfolge angegeben. Siehe dazu weiter unten.
TYP = entweder 33920, dann enthält die Matrix nur reelle Zahlen oder 77920, dann enthält die Matrix nur komplexe Zahlen
DIM = *1. Möglichkeit:* 10000 zyxwv ist ein Vektor mit #vwxyzh Zahlen
2. Möglichkeit: 20000 zyxwv utsrq ist eine Matrix mit #vwxyzh Zeilen und #qrstuh Spalten. (Das h hinter #vwxyz und #qrstu soll zeigen, daß es hexadezimale Werte sind.)
ZAHL1, ZAHL2 sind die Matrixelemente ohne die TYPAngabe, also zum Beispiel 0000000000000010000000000 0000020 = 1 2

3 hinter der objektdefinierenden Adresse steht eine fünfstellige Zahl, deren Wert sich aus der Anzahl der Nibbles des Objektinhalts plus 5 zusammensetzt.
Beispiel:
"HALLO"= E4A20F00008414C4C4F4
oder etwas übersichtlicher:
E4A20 es ist ein String!
F0000 er ist #Fh = 15 Nibbles lang:

84	das H (ASCII #48h)
14	das A (ASCII #41h)
C4	das L (ASCII #4Ch)
C4	das L (ASCII #4Ch)
F4	das O (ASCII #4Fh)

HALLO sind 5 Zeichen = 5 Bytes = 10 Nibbles. Dazu kommen 5 für die Längenangabe macht 15 = #Fh. Dieses F fünfstellig schreiben: #0000Fh.
Und schließlich die Reihenfolge der Ziffern umdrehen ergibt die Nibblefolge, wie sie im Speicher steht: F0000

4 Objekttypen, die mit dem oben erwähnten END abschließen und über keine Längenangabe verfügen.
Beispiele siehe oben.

5 Namen; diese sind so ähnlich wie Objekte der Art 3 aufgebaut. Sie besitzen jedoch nur ein Byte zur Längenangabe und dessen Wert setzt sich durch die Anzahl der Bytes (!) des Namens zusammen.
Beispiel:
'ADDIERE'=
21D207014444494542554
oder

21D20	das Objekt ist ein Name
70	der Name ist 7 Bytes lang
14	A
44	D
44	D
94	I
54	E
25	R
54	E

3. Die Speichertechnik

Es ist bereits in Tabelle 1 gezeigt worden, daß der HP28 Daten in umgekehrter Reihenfolge in den Speicher schreibt (und auch liest). Der Grund hierfür liegt im Arbeitsvorgang des 4 Bit schmalen Adress/Datenbusses. Bei jedem Datenzugriff der CPU auf den Speicher können also nur 4 Bits auf einmal gelesen (oder geschrieben) werden. Man stelle sich ein-

mal vor, man hätte einen Stapel Bücher auf dem Tisch liegen und möchte diesen Stapel in kürzest möglicher Zeit in eine Schublade räumen, könne dabei aber nur ein Buch zur Zeit nehmen. Dann wird man das oberste Buch zuerst in die Schublade legen, dann das nächste u.s.w..
Das zuletzt eingeräumte Buch war im Stapel auf dem Tisch das unterste und ist jetzt in der Schublade das oberste.
Die Reihenfolge der Bücher wurde also umgekehrt. Hat also der Rechner auf seinem Arbeitsplatz (einem der CPU-Register) die Nibbles in richtiger Reihenfolge, so werden sie durch den stückweisen Schreibvorgang in das RAM umgedreht. Konsequenter Weise müssen sie in ebenfalls umgekehrter Reihenfolge gelesen werden.
Am einfachsten werden Speicherbereiche ausgelesen, indem man den gewünschten Speicherbereich in ein CPU-Register (satte 64 Bit) einliest, und dessen Inhalt dann in einen String schreibt.
(Dies macht das Programm PEEK.)
Diesen String kann man dann auf den Stack legen und mit den Befehlen des STRING-Menüs bearbeiten.

Beispiel: Mit
```
« → x « 64 STWS HEX x
DUP SUB NUM R→B » »
```
mit x = 1,2,...,8
kann der Wert des x-ten Bytes im String gelesen werden.

Er wird als hexadezimaler Wert, also als Byte ausgegeben. Dabei taucht aber eine unschöne Eigenart des HP auf:
Der Rechner vertauscht die Nibbles in einem Byte!
D.h. der Buchstabe A (= ASCII-Zeichen 65 = #41h) wird nicht als Nibblekombination 41 (= vier eins), sondern als 14 (= eins vier) abgespeichert!
Überhaupt werden alle sinngemäß zusammen gehörenden Codes in umgekehrter Reihenfolge in den Speicher geschrieben (und natürlich auch wieder in umgekehrter Reihenfolge gelesen). Diese Codes sind
- Adressen (immer 5 Nibbles)
- reelle Zahlen
- Längenangaben
- Binary Integer
- Short Integer
- was ich sonst noch vergessen habe, kurz alles, was der HP auf einmal in seine CPU-Register laden kann und sinngemäß eine Einheit bildet. (Eine Zahl wie z.B. Pi ist eine Einheit und nicht zwei oder mehr).

3.14159265359 ist die Zahl Pi.
3.14159 265359 sind dagegen zwei Zahlen. Ein String ist in diesem Sinne keine Einheit, weil er länger als 64 Bit sein kann und somit in kein CPU-Register mehr paßt. Aber die Bytes in einem String bilden jedes für sich eine Einheit. In ihnen werden deshalb die Nibbles beim Abspeichern in das RAM umgedreht. Eine Ausnahme bilden die Maschinenbefehle. Sie werden freundlicherweise nicht umgedreht.

Weiter unten steht das Programm
69C20 Adresse, Nibblereihenfolge ist verdreht
F0000 Längenangabe, Nibblereihenfolge ist verdreht
142 Maschinenbefehl, Nibblereihenfolge ist nicht verdreht
164 Maschinenbefehl, Nibblereihenfolge ist nicht verdreht
808C Maschinenbefehl, Nibblereihenfolge ist nicht verdreht

Der Befehl mit dem Code 142 hat also tatsächlich die Folge eins-vier-zwei und nicht etwa 241. Daß sie nicht umgedreht werden liegt daran, daß ihre Befehlscodes in Maschinenbefehlstabellen bereits umgedreht werden. Es gibt Maschinenbefehle, die mit Argumenten versehen werden können. Diese Argumente müssen dann natürlich umgedreht werden.

Beispiel: CPU-Register C soll mit dem Wert #120FAh geladen werden. (D.h. C=#120FAh)
Der "Lade C mit"-Befehl hat folgendes Aussehen:
3xn...n Lade C mit dem Wert n...n (wobei der Wert eine Länge von x+1 Nibbles hat) beginnend an der Position, die durch den Wert in Register P bestimmt wird. Ist der Wert in P gleich 0, was meistens der Fall ist, dann wird das erste Nibble des Wertes in das Nibble 0 des Registers C, das 2. Nibble des Wertes in das Nibble 1 des Registers C, u.s.w. geschrieben. In diesem Beispiel lautet der Befehlscode also 34AF021.

Doch zurück zum Programm PEEK. Man müßte jedesmal die gelesenen Nibbles in den Bytes umdrehen, und dazu kommt dann noch, daß auch die Nibbles einer Einheit umgedreht sind.

Um diesem Horror nicht ausgesetzt sein zu müssen, gibt es die Programme COD und DECOD, die die Umkehrung der Nibbles in den Bytes vornehmen.
Ihre Funktion läßt sich leicht erkennen, da sie relativ kurz sind. Nehmen wir einmal an, wir hätten einen 8 Zeichen langen String im Stack, der einen Speicherauszug ab einer bestimmten Adresse darstellt. Dieser enthält einige Zeichen, die hier nur schlecht dargestellt werden können. Deshalb stehen hier nur die ASCII-Codes der 8 Bytes:

dezimal	hexadezimal
0	00
144	90
53	35
101	65
146	92
21	15
20	14
3	03

Tabelle 5

oder die hexadezimalen Ziffern einfach hintereinander geschrieben:
0090356592151403 (String1)
Na, was ist das?
PI! Dreht man nämlich die Nibbles in den Bytes, so wird daraus:
0009535629514130 (String2)
Wenn man jetzt die Kette umdreht (jede relle Zahl ist eine Einheit!), erhält man:
0314159265359000 (String3)
Ist das erste Nibble darin eine 9, so ist die Zahl negativ.
Dementsprechend ist -Pi:

9314159265359000 (String4)
Die verbleibenden 3 Nullen stellen den Exponenten dar. Er wird wie die Mantisse dezimal dargestellt. Fließkommazahlen werden intern immer im SCI 12-Format dargestellt, d.h. immer mit Exponent und einer Stelle vor dem Komma.

Der Exponent berechnet sich wie folgt:
→Exponent = Exponentenwert,
falls 0 ≤ Exponentenwert < 500
→Exponent = 1000-Exponentenwert,
falls 500 ≤ Exponentenwert ≤ 999

Das Programm DECOD liefert nun, falls der 8 Zeichen lange String vorher im Stack stand, den String2:

`1:"0009535629514130"`(String5)

(Beachte 16 Bytes lang und gut lesbar)
COD ist die Umkehrung von DECOD. Diese Programme können gar nicht auch noch diese Zeichenketten umdrehen, da sie nicht wissen können, wo eine Eiheit beginnt bzw. endet, es sei denn man macht aus ihnen Assembler und Disassembler. Das geht zwar, kostet aber untragbar viel Speicher.
Da vor einem Objektinhalt immer auch noch die objektdefinierende Adresse stehen muß, damit der Rechner auch weiß, um welches Objekt es sich handelt, sieht das vollständige Objekt, "die Zahl Pi", wie folgt aus:

`"339200009535629514130"`
(String6)

Die Nibbles in den Bytes sind dabei vertauscht dargestellt, also so, wie DECOD einen String darstellt. Das ist auch gut so, denn wer will sich schon mit der nervigen Nibblevertauscherei beschäftigen?
Das übernehmen die Programme COD und DECOD.
Zu bemerken ist auch, daß in diesem Aufsatz nur der String1 so dargestellt ist, wie PEEK die Nibbles in den String schreibt. Ansonsten wurde hier alles so dargestellt, wie es das Programm DECOD liefert bzw. COD benötigt. Einfach, weil es leichter lesbar ist und man so nichts mehr mit dem Vertauschen der Nibbles zu tun hat.
Will man eine bestimmte Nibblefolge in den Rechner schreiben und daraus dann z.B. obige Zahl generieren, so gibt man einfach String6 ein und führt COD aus. Als Ergebnis wird ein String zurückgegeben, der die Nibbles in exakt der richtigen Reihenfolge enthält. Man muß dem Rechner nur noch klar machen, daß in dem String die Information einer Zahl steht. Wie man das macht, steht weiter unten.

Jetzt müßte auch die dritte Spalte in Tabelle 1 klar sein.

Auffallend ist jedoch, daß der String6 nun 15 Zeichen lang ist.
Wenn nun immer 2 Nibbles vertauscht werden müssen, dann würde zwangsläufig eines übrig bleiben, denn nur bei einer geraden Anzahl können immer 2 restlos vertauscht werden. Damit jedoch das Programm COD die Nibbles im Byte vertauschen kann, wird einem String mit ungerader Länge eine Null voran gesetzt, womit die Stringlänge gerade wird.

Beispiel:
`1: "5142434"`

COD ausführen ergibt

`1: "PABC"`

Beachte:
P ASCII(#50h)
A ASCII(#41h)
B ASCII(#42h)
C ASCII(#43h)

DECOD ausführen erbibt dann erwartungsgemäß

`1: "05142434"`

Erneutes Ausführen von COD ergibt natürlich wieder

`1: "PABC"`

wobei diesmal COD keine weitere Null voran gesetzt hat, weil die Länge ja gerade ist.

Eine weitere Besonderheit:
Im Programm ADDIERE steht nun nicht 339200000000000000010, sondern 9C211. Das liegt daran, daß im ROM des HP28 bereits alle ganzen Zahlen von -9 bis 9 mit der objektdefinierenden Adresse 33920 vorhanden sind.
Das heißt in der Adresse #112C9h befindet sich der Code 339200000000000000010. Es spart einfach viel Speicherplatz, wenn die am häufigsten benutzten Zahlen einmal gespeichert werden und dann nur noch deren Adresse benutzt wird.

4. Auswertung von Objekten

Wurde bisher nur der Aufbau von Objekten beschrieben, so wird hier gezeigt, wie der HP die Objekte auswertet. Dazu wieder ein Beispiel:

```
ADD45
«
  4 5 ADDIERE
»
```

Als Adressenliste geschrieben:

Codes	Bedeutung
76C20	BEGIN
0D9E0	«
80311	4
D1311	5
21D20 70	} dies ist der Name ...
14 44 44 94 54 25 54	} ... ADDIERE !
6E9E0	»
09F20	END

Tabelle 6

Im HP stehen natürlich nur die Codes aus der linken Spalte und zwar in der Form:

`76C200D9E080311D131121D2 07014444494542554 6E9E009F20` (String7)

Um nun die Abarbeitung eines Programms erklären zu können, ist es notwendig, kurz die CPU-Register und ihre Funktion darzustellen.

4.1 Die CPU

Sie besteht aus
a.) den 64 Bit-Registern A,B,C,D
A: allgemeine Anwendung und setzen des Programcounters mittels des Befehles PC=(A)
B: enthält Pointer auf den RPL-Returnstack

C: Hauptrechenregister
D: enthält den Zähler für die Anzahl der noch freien Plätze auf dem Stack; wenn er = 0, dann wird die Garbage Collection ausgeführt.

b.) den 64 Bit-Registern R0, R1, R2, R3, R4
Sie dienen dazu, gegebenenfalls die Inhalte der Register A und C aufzunehmen. Sie sind also eine Art Zwischenablage. Damit ist es nicht nötig, Zwischenergebnisse immer gleich in das RAM auszulagern, nur um die Register A und C wieder frei zu bekommen.
Achtung: Die Nibbles 0 bis 4 von R4 werden als Interruptspeicher verwendet.

c.) einem Systembus-Register S (64 Bit)

d.) einem OUT-Register (12 Bit) für Datenausgaben (eventuell ist hier der Piezo angeschlossen)

e.) einem IN-Register (16 Bit) (eventuell war hier beim HP71B der Barcodeleser angeschlossen)

f.) dem Programcounter PC (20 Bit)

g.) dem Maschinen-Returnstack (8 Register zu je 20 Bit).
Es handelt sich hierbei um einen automatischen Stack, d.h. der Programmierer muß sich nicht um die Datenschieberei im Stack kümmern, der gültige Wert steht immer oben auf.

h.) einem Field-Pointerregister P (4 Bit)
Vor allem genutzt zur Selektierung von Nibbles bei der Datenmanipulation in CPU Registern

i.) einem Statusregister ST (16 Bit)

j.) einem Harware-Statusregister ST (4 Bit)

k.) einem Carryregister (1 Bit) enthält das Carryflag

l.) zwei Data-Pointing-Registern D0, D1 (je 20 Bit)

4.1.1 Der Threadpointer

D0 ist der sogenannte Threadpointer, d.h. er zeigt auf das als nächstes abzuarbeitende Objekt im RPL-Programm.
Ein RPL-Programm kennt keine GOTOs oder Interrupts, weshalb man einen (roten?) 'Faden' (den "Thread") durch ein Programm legen könnte. Auf diesem Faden sind dann die einzelnen Befehle aufgereiht. Wegen des Fehlens von GOTOs und Interrupts hat dieser Faden keine Unterbrechungen oder Knoten und es gibt nur einen einzigen Faden. Weiter unten im Text erfahrt Ihr mehr dazu.

4.1.2 Der User-Stackpointer

D1 ist der Stackpointer.
Er zeigt auf eine Adressenliste, den Stack. Die einzelnen Stackelemente, die der Benutzer im Stack sieht, stehen also nicht selber im Stack, sondern nur die Adressen, ab der sie im Speicher beginnen.

Beispiel:
Im Display steht

```
3:          "Hallo"
2:                1
1:    « + 1 DISP »
```

Annahme:
"Hallo" beginnt in der Adresse #CE509h
1 steht im ROM ab Adresse #112C9h
Das Programm beginnt bei Adresse #C8A9Ah

Dann sieht der Stack in Wirklichkeit so aus:

```
3:          905EC
2:          9C211
1:          A9A8C
```

(So wird er vom HP natürlich nicht angezeigt, denn wer kann damit schon etwas anfangen. Der HP guckt in den Adressen nach, was dort steht und zeigt das dann an.)

Weitere Annahme:
Der Stack beginnt ab Adresse #C1476h
Dann enthält das CPU-Register D1 den Wert #C1476h.
(Der Wert steht richtig herum in dem CPU-Register. Wenn man sich jedoch mittels eines Programms den Wert in's RAM schreiben läßt, so wird er natürlich umgedreht, wie zuvor beschrieben.)

Der Befehl SWAP vertauscht folglich nur die Adressen in Ebene 1: und 2:. Also hat der Stack nach einem SWAP folgendes Aussehen:

```
3:          905EC
2:          A9A8C
1:          9C211
```

DROP macht einfach folgendes

`D1=D1+5` Stack beginnt nun bei #C147Bh (= #C1476h + 5)
`D=D+1` es ist jetzt ein Platz im Stack frei geworden.

(Das sind übrigens Maschinenbefehle.)

4.2 Auswertung eines Programms

Doch jetzt zurück zur Auswertung von Programmen. Der Einfachheit halber sei hier angenommen, daß das Programm schon bis zur 4 ausgeführt ist. Die 4 sei also noch nicht ausgeführt. Die letzten Befehle im Objekt "«" sind

```
A=DAT0 A
D0=D0+5
PC=(A)
```

oder etwas ähnliches. Diese Befehlssequenz ist so eine Art *Standard Programmbeendigungs Sequenz* und wird massenhaft verwendet.

`A=DAT0 A` heißt:
"hole Wert indirekt D0 und schreibe ihn in das Adressfeld von A"

Erklärung: D0 enthält eine Adresse. Das DAT0 sagt der CPU, daß nicht der Wert aus D0 geholt werden soll, sondern der Wert, der in der Adresse steht, die in D0 vorgegeben ist.
Nun muß der CPU noch mitgeteilt werden, wieviele Bits aus dieser Adresse gelesen werden sollen.
Dies geschieht hier durch die An-

gabe des A hinter DAT0. A steht für Adressfeld und das sind immer 20 Bit, weil der HP mit 20 Bit Adressen hantiert.
(Übrigens:
2^20 in Nibbles = 1 Mega Nibbles = 512k Bytes = 512 kB maximaler Adressbereich)
Das Adressfeld des CPU-Registers A sind die Nibbles 0 bis 4. Dorthin schreibt der Rechner also den Inhalt der Adresse, die in D0 vorgegeben wurde.

D0=D0+5 erhöht den Wert in D0 um 5.

PC=(A) setzt den Programcounter (PC) auf die Adresse, die in der Adresse steht, auf die der Inhalt des Adressfeldes des CPU-Registers A zeigt.
Also: Der Wert im Adressfeld von Register A ist eine Adresse. In dieser Adresse steht wiederum eine Adresse. Und genau in diese Adresse wird der PC gesetzt.

Im HP läuft es nun folgender Maßen ab:
D0 zeigt auf das nächste Objekt im Faden. Wärend also "«" ausgeführt wird, zeigt D0 auf die 4.
Mit A=DAT0 A wird nun die objektdefinierende Adresse der 4 nach A geschafft (natürlich in's Adressfeld, das ist jetzt ja klar). In A steht also #11308h.
Als nächstes muß der Threadpointer D0 auf das übernächste Objekt im Faden (die 5) gesetzt werden. Denn wenn gleich mit PC=(A) in das nächste Objekt (die 4, denn der Rechner befindet sich nach wie vor im Objekt "«") gewechselt wird, dann muß D0 wieder auf das nächste Objekt im Faden zeigen.
PC=(A) setzt also den PC in die Adresse des Objektes 4, in der steht, was zu tun ist, wenn das Objekt ausgeführt werden soll.

Diese Anweisung beinhaltet folgende Schritte:

1. Testen, ob auf dem Stack noch ein Platz frei ist.
   Wenn nein, dann die Garbage Collection aufrufen. Wenn diese keinen Speicher mehr frei machen kann, dann wird "Low Memory" angezeigt.
2. Wenn Platz frei ist, dann D1 um 5 und D um 1 verringern.
3. In indirekt D1 die Adresse schreiben, in der das Objekt (die 4) steht.
   "Indirekt D1" heißt wieder:
   In D1 steht eine Adresse und in die soll geschrieben werden und nicht in D1.

Am Ende des Anweisungsteils dieses Objekts steht wieder die Sequenz

```
A=DAT0 A D0=D0+5 PC=(A)
```

oder etwas ähnliches,

womit in das nächsten Objekt (die 5) gewechselt wird.

Und mit der 5 passiert wieder genau das gleiche. Aber dann trifft PC=(A) auf den Namen ADDIERE.

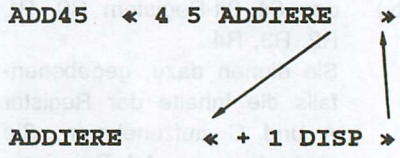
Hierbei fallen zwei Punkte auf:

1. D0 zeigt nicht auf das nächste Objekt (das "»"), sondern auf die Längenangabe des Namens (also auf den Code 70). Deswegen muß als erstes D0 korrigiert werden, d.h. D0 muß um 16 erhöht werden. 16 deshalb, weil nicht nur die 7 Bytes des Namens, sondern auch noch das Byte der Längenangabe übersprungen werden muß. Also muß D0 um 8 Bytes = 16 Nibbles erhöht werden. Diese Korrektur wird im Ausführungsteil des Objektes Namen durchgeführt.
2. Diesmal soll nichts auf den Stack gelegt werden, sondern ein Programm wurde aufgerufen. Wird es nicht gefunden, so wird die Adresse des Namens auf den Stack gelegt. Dies macht natürlich auch der Ausführungsteil des Objektes "Namen".

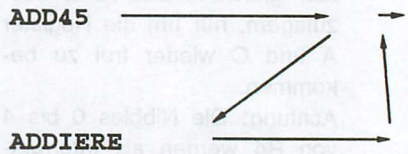
Findet der Rechner es aber, so wird es ausgeführt.
Das erste Objekt (oder auch der erste Befehl, ganz wie man will) in dem Programm ist das BEGIN. In dessen Ausführungsteil werden 2 wesentliche Dinge erledigt.

1. Der Threadpointer (also der Inhalt von D0) muß unbedingt gesichert werden. Denn der Aufruf des Programms ADDIERE legt den Faden eben in das Programm ADDIERE.

Zur leichteren Verständlichkeit folgendes Bild:



Der Faden hat also folgenden Verlauf:



Mit dem Verlegen des Fadens verläuft dieser in einem anderen Adressbereich. Deswegen muß der Wert von D0 abgespeichert werden. Dies macht der Ausführungsteil in BEGIN. Er legt den Wert von D0 im RPL-Returnstack ab, dessen Anfangsadresse im CPU-Register B steht.
Er ist genauso wie der User-Stack aufgebaut (also der, auf den D1 zeigt). Es wird also genauso getestet, ob noch ein Platz vorhanden ist, ... siehe oben.

2. Es muß in D0 ein neuer Wert installiert werden. Hier zeigt der Wert auf "«" von ADDIERE, denn dieses Objekt ist dasjenige, welches auf das BEGIN im Programm ADDIERE folgt.

Selbstverständlich können von ADDIERE aus wieder weitere Programme aufgerufen werden. Und sie werden es auch, denn die meisten Programme im ROM sind von gleicher Struktur wie die hier gezeigten. "+" und "DISP" gehören dazu. Allerdings sparen sie die "«" und "»", denn die braucht sowieso nur der Editor zur Erkennung, daß der Benutzer ein Programm eingeben möchte. Zum Anderen werden im ROM die Programme nicht mit Namen, sondern nur über ihre Adresse aufgerufen. Aber das ge-

schieht hier bei "+" und "DISP" ja auch. (+ steht nicht als 21D2010B2 im Speicher, sondern als EEE80. EEE80 ist die Adresse #8EEEh. Somit ist #8EEEh SYSEVAL nichts anderes als die Ausführung von +.)

Wenn nun das Ende von ADDIERE erreicht wird, muß der Faden wieder zurück nach ADD45 gelegt werden. Das END macht genau das. Es holt den Threadpointer vom RPL-Return-Stack zurück, (führt dabei natürlich ein DROP aus), und speichert ihn in D0. Dann kommt wieder die bekannte Endsequenz

```
A=DAT0 A
D0=D0+5
PC=(A)
```

oder etwas ähnliches,

und das war es dann schon mit ADDIERE.

4.2.1 IF-THEN-ELSE Aufbau

Zum Schluß noch eine kurze Erläuterung, wie IF-THEN-ELSE-END Schleifen aufgebaut sind:

```
2E3E0   IF
XXXXX
8F3E0   THEN
XXXXX
954E0   ELSE   } diese beiden
XXXXX          } sind optional
974E0   END
```

Für die XXXXX kann entweder
- die Adresse eines Objektes stehen, also z.B. EEE80 (dies entspricht dem "+")

oder - ein ganzes Objekt, z.B.

```
76C20
.....      diese .....
               (Punkte)
  .        stehen
  .        für
.....      mehrere
.....      Objekte
09F20
```

(Dieses hier stellt eine Befehlsfolge dar, z.B. DUP 5 == in der Folge IF DUP 5 == THEN DROP END.)

Genau genommen gehört der entweder-Punkt auch hier in den oder-Punkt, wiel eine Adresse eben auch ein Objekt ist. Wichtig ist, daß zwischen den Befehlen IF, THEN, ELSE, END immer genau ein Objekt stehen muß!

Das Programm

```
« IF THEN ELSE END »
```

hat demnach folgendes Aussehen:

```
76C20      BEGIN
0D9E0      «
  2E3E0      IF
76C20        BEGIN
    09F20      END
  8F3E0      THEN
    76C20      BEGIN
    09F20      END
  954E0      ELSE
    76C20      BEGIN
    09F20      END
  974E0      END
6E9E0      »
09F20      END
```

Wenn man die BEGIN END heraus nimmt, dann stürzt der Rechner ab, eben weil genau ein Objekt zwischen den Befehlen IF, THEN, ELSE, END stehen muß!

4.2.2 Absolut adressierte Maschinenprogramme

Schließlich bleibt noch die Klasse der absolut adressierten Maschinenprogramme übrig.
Eine Adresse im Speicher gibt an, an welcher Adresse das Maschinenprogramm beginnt.
Die Länge ist nirgends vermerkt und eine definitive Endsequenz fehlt, weshalb so ein Programm vom HP nicht als Objekt behandelt werden kann.

Zudem darf ein solches Programm nicht einfach im Speicher verschoben werden, sondern darf nur an einer Adresse gehalten werden, weshalb diese Programmform ungeeignet für die Abspeicherung im RAM ist, denn ein einziger STO-Befehl kann bereits einen großen Bereich des Speicherinhalts verschieben und so dazu führen, daß der Programcounter auf eine Adresse gesetzt wird, an der jetzt gar kein Maschinenprogramm mehr steht. (siehe Tabelle 7)

Durch Verschieben des Programms innerhalb des Speichers steht der Programmcode an anderer Adresse. Der Programmcode wird dadurch aber nicht geändert. Startet man dann das verschobene Programm, so wird der PC nach wie vor in die Adresse #CE987h gesetzt, obwohl dort gar nicht mehr der Befehl A=DAT0 A steht, sondern irgend etwas anderes. Im ROM wird diese Technik allerdings sehr viel verwendet, weil sie sehr sparsam mit dem Speicherplatz umgeht.

```
#CE982   789EC      dies heißt: in Adresse #CE987h beginnt das Programm
#CE987   142        A=DAT0 A
#CE98A   34F3E8B    C=#B8E3Fh
  .        .          .
  .        .          .
                          ↑ dies sind die Mnemonics
                    ↑ dies ist der Programmcode
↑ dies sind die Adressen, an denen die Befehle stehen (Die Punkte sollen weitere Zeilen darstellen, ist ja wohl logisch, oder?)
```

Tabelle 7

4.2.3 Der INLINE-MCODE

Wesentlich besser ist es, den INLINE-MCODE zu verwenden:

Beispiel:

```
69C20   objektdefinierende Adresse
        des INLINE-MCODEs
F0000   Länge des Programms (in
        Nibbles) + 5
142     A=DAT0 A
164     D0=D0+5
808C    PC=(A)
```

Den INLINE-MCODE kann man dann nämlich 'in line' in Objektlisten (also z.B. in Programmen, denn

diese sind, wie oben erwähnt, Objektlisten) verwenden, weil er ein echtes Objekt ist. Dargestellt wird dies im Display und auf dem Papierausdruck als System Object, weil der Rechner keine andere Darstellungsform dafür hat.

Oben stehendes Programm stellt das kürzest mögliche INLINE-MCODE-Programm dar. Es besteht nur aus einem INLINE-MCODE-Kopf (69C20 XXXXX) und der Standardausstiegsroutine (142 164 808C).

Dieser Programmtyp ist nicht an feste Adressen gebunden und ist deshalb im gesamten Speicher lauffähig und kann folglich beliebig umher geschoben werden. Durch seine Längenangabe besitzt der INLINE-MCODE eine genau bestimmte Länge. Damit findet der HP auch das Ende dieses Programmtyps, weshalb er ihn als Objekt behandeln kann (und es auch tut).

5. Erzeugen beliebiger Objekte

Man kann sich sicher nun schon denken, wie man dem Rechner klar macht, daß in String6 eine Zahl steht, und daß er diese auf den Stack legen soll.

String6 liegt wie folgt im Speicher:

```
E4A20B100003392000095356
29514130
```

(Es steht eine zusätzliche 0 vor der objektdefinierenden Adresse der Zahl (33920), weil die Anzahl der Nibbles in einem String gerade sein muß. Denn jedes Zeichen in einem String belegt genau ein Byte (= 2 Nibbles). Folglich ist die Längenangabe B1000. Dies entspricht #1Bh = 27. (3392000095356295141300 sind 22 Nibbles. Dazu kommen die 5 Nibbles für die Längenangabe. Macht zusammen 22 + 5 = 27.)

5.1 Die Adresse eines Objekts

Kennt man die Adresse, ab der dieses Objekt beginnt, (d.h. wo das E dieser Nibblekette steht) so kann man den String6 mit #xxxxxh SYSEVAL ausführen (#xxxxxh ist die Adresse).
Damit wird aber nur der String auf den Stack gelegt. Wenn aber die Adresse um 10 erhöht wird, und man dann SYSEVAL ausführt, so trifft der HP auf die objektdefinierende Adresse einer Zahl. Wie oben erwähnt, ist in der objektdefinierende Adresse einer Zahl eine Routine vorhanden, die testet, ob noch Platz auf dem Stack ist, und falls ja, die Zahl dann auf den Stack legt.
Folglich legt der HP die Zahl auf den Stack, wo sie uns zur freien Verfügung steht! Noch könnte die Zahl allerdings eventuell durch eine Löschoperation (PURGE oder etwas ähnliches) zerstört werden, denn sie ist immer noch Teil des String6. Und nur ihre Adresse liegt auf dem Stack, wie bereits beschrieben.

Wenn man aber gleich nach dem SYSEVAL die Zahl mit STO abspeichert, dann erzeugt der Befehl STO eine Kopie der Zahl und speichert diese dann ab. Damit ist dann die Zahl völlig eigenständig und von String6 getrennt.

5.1.1 Auffinden der Adressen

Bleibt nur die Frage, wie man die Adresse von String6 findet. Dazu gibt es 2 Möglichkeiten.

Die erste:
Wie bereits erwähnt, zeigt CPU-Register D1 auf den USER-Stack, genauer auf die Adresse in Ebene 1 des Stacks.
Liegt String6 in Ebene 1, so zeigt D1 also auf die Adresse des Strings. Wenn man jetzt eine Routine hätte, die diese Adresse liest und sie in einen Binary Integer schreibt, so müßte man nur noch diesen in den Stack legen. Wenn dann noch 10 dazu addiert wird, erhält man die Adresse der Zahl (die im String kodiert ist).
Mit SYSEVAL kann dann die Zahl auf den Stack gebracht werden. Die Sache hat nur einen Haken:
Scheinbar gibt es im ROM keine Routine, die so etwas tut. Und selbst schreiben kann man die Routine nicht, denn dazu müßte man ein Maschinenprogramm in den Rechner bringen. Deshalb jetzt

Die zweite: Das RAM beginnt bei Adresse #C0000h und endet bei #CFFFFh. Befindet man sich im HOME-Menu, so wird ein Objekt beim Abspeichern mittels STO in die höchsten Adressen abgelegt:

```
2:"3392000095356295141300"
1:                 'String6'
```

Wenn dann HOME SWAP COD SWAP STO ausgeführt wird, sieht der Speicher also wie folgt aus:

```
# CFFE0h: E  ← hier beginnt
             die objektdefinie-
             rende Adresse
             des Strings
# CFFE1h: 4
# CFFE2h: A
# CFFE3h: 2
# CFFE4h: 0
# CFFE5h: B  ← hier beginnt
             die Längenangabe
             des Strings
# CFFE6h: 1
# CFFE7h: 0
# CFFE8h: 0
# CFFE9h: 0
# CFFEAh: 0
# CFFEBh: 3  ← hier beginnt
             die objektdefinie-
             rende Adresse der
             Zahl
# CFFECh: 3
# CFFEDh: 9
# CFFEEh: 2
# CFFFFh: 0
# CFFF0h: 0  ← hier beginnt
             der Exponent der
             Zahl
# CFFF1h: 0
# CFFF2h: 0
# CFFF3h: 9  ← hier beginnt
             die Mantisse der
             Zahl
# CFFF4h: 5
# CFFF5h: 3
# CFFF6h: 5
# CFFF7h: 6
# CFFF8h: 2
# CFFF9h: 9
# CFFFAh: 5
# CFFFBh: 1
# CFFFCh: 4
# CFFFDh: 1
# CFFFEh: 3
# CFFFFh: 0
```

Tippt man dann also #CFFEBh SYSEVAL ein, so wird die Zahl auf den Stack gelegt! So einfach ist das! (Man muß es nur wissen.)

(Nebenbei:
#0h : Anfangsadresse vom ROM
#3FFFFh: Endadresse vom ROM

Kurioserweise befindet sich das RAM auch auf den Adressen #D0000h bis #DFFFFh.
Wahrscheinlich hat HP einfach einen Adresspin nicht mitverkabelt. #Ch (= #1100b) und #Dh (= #1101b) unterscheiden sich nämlich nur in einem Bit.)

Damit oben beschriebener Vorgang auch wirklich klar ist, hier noch einmal "mit der Hand am Arm" und "ganz zu Fuß":

5.2 Schrittweise Erzeugung eines Objektes

Die Nibblefolge
339200009535629514130 (Folge1)

soll in den Speicher. Es sind 21 Bytes, also eine ungerade Länge. Sie muß aber gerade sein, damit jeweils 2 Nibbles in einem Byte zusammen gefaßt werden können. Deshalb wird der Kette eine 0 voran gesetzt:

0339200009535629514130 (Folge2)

Jetzt je 2 Nibbles in einem Byte zusammen fassen:

03 39 20 00 09 53 56 29 51 41 30

Dann die Nibbles in jedem einzelnen Byte drehen:

30 93 02 00 90 35 65 92 15 14 03

Jetzt # davor und h dahinter:

```
#30h #93h #02h #00h #90h
#35h #65h #92h #15h #14h
#03h
```

Diese Integer stellen die Werte für die Bytes dar, die den String bilden.
Folglich:

```
#30h  B->R CHR
#93h  B->R CHR +
#02h  B->R CHR +
#00h  B->R CHR +
#90h  B->R CHR +
#35h  B->R CHR +
#65h  B->R CHR +
#92h  B->R CHR +
#15h  B->R CHR +
#14h  B->R CHR +
#03h  B->R CHR +
```

In Ebene 1: steht jetzt ein String, der exakt die richtige Nibblekombination hat. Dieser String muß noch in die höchsten Adressen des RAMs (dem sogenannten Top of RAM) gebracht werden.
Dazu wird wieder HOME 'String6' STO ausgeführt.

Besser ist es aber,

```
HOME 'String6' DUP PURGE
STO
```

auszuführen, denn dann ist sichergestellt, daß sich nicht bereits an anderer Stelle im HOME-Menu ein Eintrag 'String6' befindet, der dann überschrieben werden würde, und zwar mit der Konsequenz, daß der String nicht im Top of RAM steht, sondern da, wo vorher der Eintrag 'String6' stand.

Jetzt sieht der Speicher wieder aus wie in Tabelle 7.

Also wieder

```
#CFFEBh SYSEVAL
```

ausführen, und als Ergebnis steht die Zahl im Stack.

5.2.1 Adressberechnung

Die Adresse läßt sich natürlich auch berechnen. Folge1 sind 21 Nibbles. Diese 21 zieht man von #D0000h ab.

```
#D0000h - 21 = #CFFEBh
```

Obwohl die Folge2 in den Speicher gebracht wurde, darf man sie nicht zur Berechnung der Adresse heranziehen, da sie mit einer 0 beginnt, die aber ja nur voran gesetzt wurde, damit sich die Bytes erzeugen lasssen.

6. Probleme...

Die in Kapitel 5.2 beschriebene Methode zur Erzeugung von Objekten funktioniert zwar, ist jedoch sehr umständlich und deshalb fehlerträchtig. Deshalb wurden im Laufe der Zeit einige Programme geschrieben, die das Erzeugen auch noch so skurriler Objekte zum Kinderspiel macht. Doch der Reihe nach.

6.1 ... und deren Lösung, Teil 1

*) Bereits erwähnt wurden die Programme COD und DECOD. Sie liegen jeweils in 2 Versionen vor.

1. als gewöhnliche RPL-Programme kurz, einfach einzugeben und sehr langsam
2. als Maschinenprogramme ebenfalls kurz, nicht ganz so einfach einzugeben und sehr schnell (etwa 300 bis 400 mal schneller als die RPL-Versionen)

*) ESNL (Erase SPACEs and NEWLINEs) ist ein Programm, das in einem String alle NEWLINEs (= ASCII-Zeichen 10) und SPACEs (= ASCII- Zeichen 32) entfernt.
Schreibt man einen sehr langen String, so kann es die Eingabe des Strings erleichtern und die Übersichtlichkeit erhöhen, wenn man die einzelnen Zeichen im String durch SPACEs und NEWLINEs trennt.

Bsp:
Folge1 war
339200009535629514130

Will man die Zahl erzeugen, dann muß man den String

```
1: "339200009535629514130"
```
(String8)

eingeben. Übersichtlicher wäre folgende Eingabe:

```
1:"33920 000 953562951413 0"
```
(String9)

also alles sauber nach Funktionalität geordnet.

Bevor man String9 mit COD in einen codierten String wandelt, müssen die SPACEs und NEWLINEs aus dem String entfernt werden, denn sie sollen ja nicht mitcodiert werden. Dazu einfach ESNL ausführen. Wem das Beispiel zu

banal ist, der kann anstelle von String8 natürlich auch String7 verwenden.

*) P→L wurde auch schon erwähnt. Es wandelt ein Programm in eine Liste. L→P ist die Umkehrung von P→L. Zu beachten ist bei beiden, daß sie keine Kopie eines Objektes erzeugen, sondern direkt den Objekttyp ändern, das heißt sie ändern die objektdefinierende Adresse des Objektes in Ebene 1. Wurde dieses Objekt per RCL in den Stack gebracht, so ändern die Programme nicht nur das Objekt im Stack, sondern auch das abgespeicherte, weil im Stack die Adresse dieses Objekts steht.

*) ASO heißt "Address of Stack Object". Sie wurde bereits in Kapitel 5.1.1 unter "Die erste" beschrieben. Liegt im Stack beispielsweise die Zahl 1, so ersetzt ASO diese durch das Ergebnis #112C9h, also der Adresse, in der das Objekt zu finden ist.

OSA ist die Umkehrung von ASO. Sie ersetzt also den Binary Integer in Ebene 1 durch einen Pointer, der durch den Wert des Binary Integers vorgegeben ist.

Ein Beispiel:

```
1:      #112C9h
```

Der Binary Integer sei in der Adresse #C1E1Ah abgespeichert. Dann steht im Stack in Ebene 1 in Wirklichkeit der Wert (Pointer) A1E1C.

```
1: A1E1C ---> #C1E1Ah:
07A20510009C21100000000000
```

Dann nimmt OSA den Wert aus der Adresse #C1E1Ah+10 (dort steht 9C211) und schreibt ihn in Ebene 1 hinein.
Mit OSA wird auch SYSEVAL völlig überflüssig. OSA führt ein Objekt im Gegensatz zu SYSVEAL nicht aus und infolge dessen stürzt der Rechner auch nicht ab, wenn man mal die falsche Adresse eingetippt hat. (Das Objekt wird erst dann ausgeführt, wenn EVAL gestartet wird.) Deshalb sollte OSA das erste Maschinenprogramm sein, das in den HP eingegeben wird. Es gibt doch eine Möglichkeit, bei der der HP durch OSA abstürzt:
Wenn der Pointer in Ebene 1 auf einen leeren algebraischen Ausdruck (ADA20 09F20) gesetzt wird. Diesen kann der HP nämlich nicht anzeigen, weil er ganz verzweifelt nach Variablen, mathematischen Funktionen und Zahlen sucht und keine finden kann. Man sollte diese Objekte deshalb niemals erzeugen.

*) Viele interne Routinen verwenden Short Integer. Es gibt 2 Systemroutinen für die Konvertierung zwischen einem Short Integer und einer reellen Zahl. R→SI wandelt eine reelle Zahl in einen Short Integer. SI→R wandelt einen Short Integer in eine reelle Zahl.

*) PEEK wurde auch schon erwähnt. #xxxxxh PEEK liefert als Ergebnis einen 8 Byte langen String, der einen Speicherauszug ab Adresse #xxxxxh darstellt. Um den String lesbar zu machen, ruft man DECOD auf. Ein POKE habe ich bisher nicht benötigt (ist auch viel zu gefährlich).

*) Weil der Aufruf der Sequenz

```
« 64 STWS HOME ESNL DUP
COD 'A' DUP PURGE STO
#D0000h SWAP SIZE - OSA
»
```

mir immer noch nicht schnell und elegant genug war, schrieb ich das Programm Q→O (= Quelltext in Objekt wandeln). (Oben stehende Sequenz macht aus einem String das in ihm codierte Objekt. Es ist prinzipiell die gleiche Befehlsabfolge wie in Kapitel 5.2. Man gibt aber einfach einen String ein und startet die Sequenz. Also beispielsweise den String6 aus Kapitel 3.

Q→O kommt ohne die Speicherei im HOME-Menu aus - die Adresse, in der das Objekt steht, wir mit ASO bestimmt.

Die Sache mit PEEK DECOD war mir auch zu unbequem. Deshalb habe ich das Programm O→Q geschrieben, das ein Objekt in den dazu gehörenden Quelltext wandelt. Einfach das Objekt in Ebene 1 legen und O→Q ausführen. Das Ergebnis ist der String.
*) Aber auch das war mir noch nicht bequem genug. Denn wenn ein Maschinenprogramm in ein Schleifenkonstrukt eingesetzt werden soll, dann muß man umständlich das Programm mit P→L in eine Liste wandeln, dann die Position der Schleife bestimmen u.s.w. (Genaueres dazu steht in der Dokumentation zu P→L.) Abhilfe schafft das Programm DP→CP.

Weiter kann ein Programm, das System Objekts enthält, nicht einfach editiert werden. Deshalb gibt es auch zum Programm DP→CP eine Umkehrung: CP→DP. Genaueres siehe in der Dokumentation dazu.

Die hier erwähnten Programme befinden sich im zweiten Teil diese Artikels. Es handelt sich dabei um Papierausdrucke vom HP28S.

!!!!!!!!!!!!!!! Wichtig !!!!!!!!!!!!!!!!!
Für alle hier angegebenen Programme muß 64 STWS eingestellt sein, denn die Programme testen die Wordsize nicht
!!

6.2 ... und deren Lösung, Teil 2

Zur Verdeutlichung der Funktionsweise der Maschinenprogramme wird jetzt das kurze Maschinenprogramm OSA beschrieben. (Funktion von OSA siehe oben). Damit das ganze interessanter wird, werden 2 Versionen besprochen.

6.2.1 MaschinenprogrammOSA alte Version, im zweiten Teil zu finden

```
« B->R  System Object
System Object »
```

(Die Klammern '«' und '»' werden bei der Version, die sich auf den Kopien befindet, nicht mit erzeugt. Wie erwähnt haben sie nichts mit der Funktionsfähigkeit der Programme zu tun. In diesem Text sind sie nur angegeben, damit man sieht, daß es sich um ein Programm handelt.)

Der erste Befehl hat vor allem die Aufgabe zu testen, ob in Ebene 1 des Stacks auch wirklich ein Binary

Integer steht.

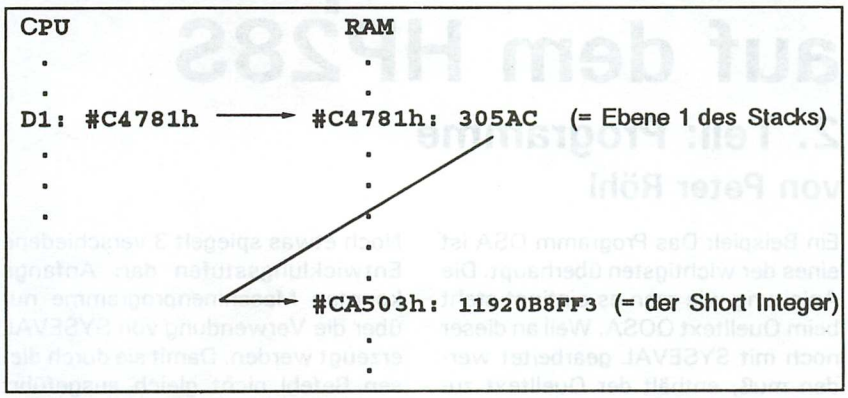
Das erste System Object ist die Adresse #C53Bh bzw. der Code B35C0. (Anstelle dieses System Objects könnte also auch #C53Bh SYSEVAL stehen, jedoch wäre das eine sehr unschöne Lösung (langsam, lang und völlig stillos).) In dieser Adresse steht die Routine R→SI (Beschreibung siehe oben).

Das zweite System Object ist das Maschinenprogramm.

Nach dem der Befehl B→R und das erste System Object ausgeführt wurden, steht in Ebene 1: also ein Short Integer mit dem Wert, den der Binary Integer hatte.

Angenommen der Stack befindet sich im Speicher ab Adresse #C4781h, der Short Integer beginnt im Speicher an der Adresse #CA503h und der Wert des Binary Integers ist #3FF8Bh.

Dies kann man folgendermaßen graphisch darstellen:



Das Maschinenprogramm :

```
69C20     INLINE-MCODE
12000     LEN (=LÄNGE)
143       A=DAT1 A
133       AD1EX
174       D1=D1+5
147       C=DAT1 A
133       AD1EX
145       DAT1=C A
142       A=DAT0 A
164       D0=D0+5
808C      PC=(A)
```

Funktionsbeschreibung des Maschinenprogramms:

```
69C20     INLINE-MCODE
```

```
12000     LEN (=LÄNGE)
```

ist der übliche Header (=Programmkopf)

```
143       A=DAT1 A
```

Dieser Befehl liest den Inhalt der Adresse, die in D1 spezifiziert ist, in das Adressfeld des CPU-Registers A. D1 zeigt auf den Stackanfang (= #C4781h). Dort steht die Adresse des Elements, das in Ebene 1: steht. Hier also die Adresse, wo der Short Integer beginnt: #CA503h Also steht im Adressfeld des CPU-Registers A nach diesem Befehl der Wert (bzw. die Adressse) #CA503h.

```
133       AD1EX
```

Vertauscht den Inhalt von CPU-Register D1 mit dem Inhalt des Adressfeldes von CPU-Register A.
Hier hat dies den Zweck, daß der Inhalt von D1 in A gesichert wird und D1 gleichzeitig mit einem neuen Pointer geladen wird. Da vor diesem Befehl in A die Adresse des Short Integers stand, zeigt D1 nach diesem Befehl nicht mehr auf den Stack, sondern auf den Short Integer bzw. auf dessen objektdefinierende Adresse.

```
174       D1=D1+5
```

Erhöht den Wert in D1 um 5. Damit zeigt der Pointer in D1 nicht mehr auf die objektdefinierende Adresse des Short Integers, sondern auf seinen Wert (= #3FF8Bh bzw. B8FF3).

```
147       C=DAT1 A
```

Kopiert den Wert, der in der Adresse steht, die durch D1 angegeben ist in das Adressfeld des CPU-Registers C.
Oder kurz: D1 zeigt auf den Wert des Short Integers. Mit diesem Befehl wird dieser Wert in das Adressfeld des Registers C geladen.

```
133       AD1EX
```

Wie oben; der Sinn ist hier jedoch das Zurückschreiben des Stackpointers. Wenn man das nicht tut stürzt der Rechner ab, weil er den Stack dann nicht mehr findet.

```
145       DAT1=C A
```

Die Umkehrung des Befehls
`C=DAT1 A`
Der Inhalt des Adressfeldes von C wird in die Adresse geschrieben, die in D1 angegeben ist. D1 zeigt hier wieder auf den Stack. Im Stack stehen die Adressen, wo die einzelnen Stackobjekte gespeichert sind. Also ersetzt dieser Befehl im Stack die Adresse des Short Integers durch die, die durch den Wert des Binary Integers vorgegeben wurde, hier #3FF8Bh bzw. B8FF3. Weil im Stack die Adresse des Short Integers somit verschwunden ist, vergißt der HP den Short Integer. Er steht zwar noch im Speicher, fällt aber der nächsten Garbage Collection zum Opfer.

```
142       A=DAT0 A   siehe oben
164       D0=D0+5
808C      PC=(A)
```

6.2.2 MaschinenprogrammOSA
neue Version, nicht auf den Kopien angegeben

Man muß allerdings nicht den Weg über den Short Integer gehen.
Schließlich ist es auch möglich, den Wert mit dem Maschinenprogramm direkt aus dem Binary Integer zu lesen. Dieser hat aber einen anderen Aufbau als ein Short Integer.
Hier beide im Vergleich:

Short Integer 11920B8FF3

Binary Integer
07A2051000B8FF300000000000

Der Binary Integer besitzt zusätzlich eine 5-stellige Längangabe (51000). Diese und die objektdefinierende Adresse muß im Maschinenpro-

gramm übersprungen werden. Dazu wird einfach der Befehl

```
174    D1=D1+5
```

in

```
179    D1=D1+10
```

geändert. Außerdem wird das erste System Object durch den Befehl R→B ersetzt.

Übrigens:
wenn man OSA tatsächlich mit dem Binary Integer #3FF8Bh in Ebene 1: ausführt, dann wird der String "Version 2BB" als Ergebnis in den Stack gebracht.

Wichtig:
Im Buch CUSTOMIZE YOUR HP28 von W.A.C. Mier-Jedrzejowicz befindet sich ein grober Fehler. Im Folgenden ist der Sachverhalt korrekt wieder gegeben.
Sprungweitenberechnung bei relativen Sprüngen:

1.) für alle *relativen (!) GOTO's:*

6hhh	GOTO
4hh	GOC
5hh	GONC
8Chhhh	GOLONG
yy	GOYES

Die Sprungweitenangabe (kurz: Swa) hh, hhh, hhhh, yy muß immer mit in die Sprungweite (kurz: Sw) einbezogen werden bei

a.) Sprüngen zu höheren Adressen: die Adresse, in der die Swa beginnt plus die Swa ergibt die Zieladresse.

b.) Sprüngen zu niedrigeren Adressen: von der Swa muß 1 abgezogen werden. Dieses Resultat muß dann invertiert werden (auf Bitebene: 1 wird 0 und 0 wird 1). Dies so erhaltene sogenannte B-Komplement wird von der Adresse, in der die Swa beginnt, abgezogen. Das Resultat ist die Zieladresse.

Ist das ganz rechts im Code der Swa stehende Nibble ≤ 7, so ist a.) zu verwenden, sonst b.)

2.) für alle *relativen (!) GOSUBs:*

7hhh	GOSUB
BEhhhh	GOSUBL

Die Swa darf nicht mit in die Sw einbezogen werden!

a.) und b.) wie oben, jedoch Adresse des auf die Swa folgenden Befehls.
Aus dem eben erwähnten Buch müssen noch die Seiten 101-102 sowie 177-201 als Informationsquelle hinzugezogen werden !

Peter Röhl
Osterfeuerbergstraße 70
2800 Bremen 1

Maschinensprache auf dem HP28S

2. Teil: Programme

von Peter Röhl

In diesem Teil finden sich fast ausschließlich Maschinenprogramme, die den Umgang mit der Maschinensprache sehr erleichtern. Einmal ausprobiert und es wird klar, warum ich nicht mehr auf sie verzichten möchte. Es ist jedoch unbedingt nötig, sich vorher das im ersten Teil angesprochene Buch durchzulesen, weil man sonst nicht versteht, was diese Programme machen.

Um den Rechner möglichst schnell mit den wichtigsten Programmen für die Erzeugung von Maschinenprogrammen auszustatten, habe ich eine Art Anleitung geschrieben, nach der man am besten stur verfährt. Sie ist mit "Erzeugung diverser Utilities für die Maschinenprogrammierung" überschrieben. Auf den folgenden Seiten finden sich dann (leider verstreut, sorry) Erklärungen für die Funktionsweisen der Programme.

Ein Beispiel: Das Programm OSA ist eines der wichtigsten überhaupt. Die Anleitung wie man es eintippt steht beim Quelltext QOSA. Weil an dieser noch mit SYSEVAL gearbeitet werden muß, enthält der Quelltext zusätzlich die Codes für eine Listendefinition (69A20 ...09F20, siehe Teil 1, da steht der Grund).

Später wiederholt sich mehrmals derselbe Quelltext, allerdings aus dem lauffähigen Programm zurückgewonnen und mit der Hand nachbearbeitet oder als dokumentierter Quelltext zu OSA mit Funktionsbeschreibung oder als Demonstrationsobjekt.

Dieses Chaos tut mir zwar leid, es war aber nicht zu verhindern, weil die meisten Ausdrucke nach und nach entstanden sind und teilweise mit wichtigen Kommentaren versehen wurden.

Noch etwas spiegelt 3 verschiedene Entwicklungsstufen dar: Anfangs konnten Maschinenprogramme nur über die Verwendung von SYSEVAL erzeugt werden. Damit sie durch diesen Befehl nicht gleich ausgeführt werden, sind sie in eine Liste verpackt.

Später als ich die Programme ASO, OSA, L→P, P→L, Q→O geschrieben hatte, war dieser Aufwand nicht mehr nötig, weil ich mit diesem Programmen einzelne Befehle aus einem Programm herausholen und ersetzen konnte. Ein Beispiel hierfür ist DECOD.

Man tippt DECOD ein und läßt das System Objekt dabei weg. Dann wird QDECOD eingegeben und mit Q→O in das System Objekt gewandelt. Jetzt müssen wir nur noch irgendwie dieses System Objekt in das Programm DECOD einsetzen.

Wie man das macht steht am Anfang der Programmlistings.

Kurz und ohne Erklärung:

```
2: «DUP NUM DROP ... »
1: {System Objekt}

SWAP P→L DUP SIZE 1- TRENN
ROT SWAP + + L→P)
```

Dann war mir auch dieser Weg zu umständlich. Deswegen schrieb ich einen Art Mini-Inline-Assembler und Mini-Inline-Disassembler. Sie heißen CP→DP und DP→CP. Bei ihnen steht der Assemblerquelltext in einem String im normalen Programm.

Legt man dieses Programm auf den Stack und führt DP→CP aus, so wird das lauffähige Programm erzeugt.

Beispiele finden sich in den Listings.

Hinweise:

1. Manche Programme stehen ohne « und » da. Das ist vollkommen egal! Ich habe sie weggelassen, weil sie zusammen 5 Bytes kosten. Wen das stört, der kann sie ja selbst hinzufügen.

2. Bei einigen Quelltexten (z.B. QDECOD) steht noch die Definition einer Liste drin. Im lauffähigen Programm muß sie natürlich entfernt sein, sonst wird nicht der Listeninhalt, sondern die Liste ausgeführt. Im obigen Beispiel haben die beiden Plus diese Liste aufgelöst
`({A} {B} {C} ++ => {A B C}).`

Ich glaube aber, daß man sich durch dieses Chaos wühlen kann. Wer trotzdem Fragen hat, soll mir einfach schreiben.

Kurze Namenserklärung einiger Programme:

O→Q : Objekt in Quelltext wandeln
Beispiel:
`5 O→Q => "33920...050"`
also Umkehrung von Q→O.

Q→O : Quelltext in Objekt wandeln
Beispiel:
`"339200000000000000050" Q→O => 5.`

PEEK : Speicherauszug (8 Bytes) ab Adresse #XYZh in den Stack als String legen.

DECOD : Maschinenversion des Programms aus dem Buch (300 bis 400 mal schneller).

SI→R : Short Integer in Realzahl wandeln.

R→SI : Realzahl in Integer Wandeln.

UNDO : UNDO auf dem Tastenfeld.

SPEED : siehe PRISMA, beachte: Alle meine Programme dürfen überall im Speicher stehen. Sie sind an kein Directory gebunden!

SLOW : 0 SPEED.

FAST : 15 SPEED.

TYP : legt die objektdefinierende Adresse eines Objektes als Binary Integer in den Stack.

NOP : erzeugt ein leeres Menüfeld!

PNOP : löscht ein leeres Menüfeld.

GVARS : (Get variables) holt alle Variablennamen aus einem Programm oder einem algebraischen Ausdruck und legt sie in einer Liste auf den Stack; auch wenn nur der Name eines Programms oder eines algebraischen Ausdrucks angegeben wird. Beispiel:

```
« x y * » GVARS => {x y}
```
oder
```
 'A+B' 'C' STO 'x+y''A' STO
'C' GVARS  => {x y B}.
```

SMF : set Message Flag (Gegenteil von CLMF).

ZEIG : Objekt vom Stack nehmen und in Zeile 1 invers anzeigen.

MDISP : (Menu Display) zeigt momentanes Menu an. Beispiel:
```
CLLCD MDISP =>
```
nur das Menu ist zu sehen.

OFF : Schaltet den Rechner aus.

EDIT : EDIT von der Tastatur.

BOFF : Busy-Indicator OFF.

CONT : Continue vom Tastenfeld.

O→S:Objekt in String verpacken
Beispiel: `"A" O→S => ""A""`

SHIFT: SHIFT von der Tastatur

CMEN: gibt das Custommenu in den Stack (also Gegenteil von Menu)

NB→S: wandelt Namen in Strings:
`'A' NB→S => "A"`

So das war's. Bleibt nur noch zu sagen, daß diese Programme so nur auf dem HP28S laufen. Für andere Rechner (HP28C, HP48SX) müssen die objektdefinierenden Adressen geändert werden. Bei meinem Rechner sind mit diesen Programmen noch keine Abstürze aufgetreten (Ausnahme siehe Listings).

Erzeugung diverser Utilities für die Maschinenprogrammierung

1.) folgende Programme in das HOME-MENU eintippen:

```
ESNL
« -12 SWAP 1 2
  START SWAP 22 +
SWAP
    WHILE DUP 3 PICK
CHR POS
    REPEAT LAST DUP2
1 - 1 SWAP SUB ROT
ROT 1 + OVER SIZE
SUB +
    END
  NEXT SWAP DROP
»

COD
«
  IF DUP SIZE 2 MOD
  THEN "0" SWAP +
  END "" SWAP
  WHILE DUP SIZE
  REPEAT DUP 3 OVER
SIZE SUB "#" ROT 1 2
SUB + RCLF 8 STWS
HEX SWAP STR→ RR RR
RR RR B→R CHR SWAP
STOF ROT SWAP + SWAP
  END DROP
»
```

und folgenden String:

```
QOSA
"
69A20 76C20 BBCB0
B35C0 69C20 12000
143 133 174 147 133
145 142 164 808C
09F20 09F20"
```

Folgendes Programm schafft aus dem String in QOSA das Programm OSA:

```
MAKE
« HEX 64 STWS QOSA
ESNL DUP SIZE
# D0000h SWAP - SWAP
COD HOME 'ABC' DUP
PURGE STO
»
```

MAKE ausführen.

Jetzt muß # CFFBCh im Stack stehen, sonst ist etwas falsch.

Wenn # CFFBCh im Stack steht, wird SYSEVAL ausgeführt.

Sofort wird folgende Liste in den Stack gelegt:

```
{ B→R System Object System
Object }
```

`LIST→ DROP` führt zu

B→R System Object System Object

Dies ist das Programm OSA. Also 'OSA' STO und das erste M-Code-Programm ist 'im Kasten'.

Ab jetzt braucht man SYSEVAL nicht mehr, OSA ist viel besser.

Nun QASO und MAKE2 eintippen:

```
QASO
"
69A20 76C20
11920 00000
01670
69C20 03000
133 131 100 147 174 143
133 174 145 110 133 142
164 808C
A2670 016C0 79CB0
09F20 09F20"

MAKE2
« HEX 64 STWS QASO
ESNL DUP SIZE
# D0000h SWAP - SWAP
COD HOME 'ABC' DUP
PURGE STO OSA
»
```

MAKE2 ausführen.

Es liegt nun folgende Liste im Stack:

```
{ System Object SWAP System
Object DROP System Object
R→B }
```

LIST→ DROP 'ASO' STO und fertig ist ASO.

Weil es ziemlich umständlich ist, Programme so zu erstellen, muß ein weiteres Programm her, das diese Aufgabe übernimmt. Q→O ist dieses Programm.

```
Q→O
« ESNL DUP COD ASO
SWAP SIZE 2 MOD + 10
+ OSA
»
```

MAKE2 und MAKE sind nun überflüssig.

Q→O erledigt deren Aufgabe. Ein weiterer Vorteil ist, daß man sich nicht mehr im HOME-MENU befinden muß, und es wird im ohne STO gearbeitet, was chic und schnell ist.

Beispiel: Die Folge

```
QASO Q→O
```

erzeugt die Liste, die wir nach MAKE2 erhalten hatten - so einfach ist das.

Jetzt QL→P eingeben:

```
QL→P
"
69A20 69C20 73000
143 133 100 143
3469A20 8A6C0 3476C20
145 110 133 142 164
808C 09F20"
```

Die Folge

```
QL→P Q→O LIST→ DROP 'L→P'
STO
```

erzeugt das Programm L→P.

Das gleiche macht man mit QP→L:

```
QP→L
"
69A20 69C20 73000
143 133 100 143
3476C20 8A6C0 3469A20
145 110 133 142 164
808C 09F20"
```

Die Folge

```
QP→L Q→O LIST→ DROP 'P→L'-
STO
```

erzeugt das Programm P→L.

2.) Erzeugen von nicht normalen Programmen.

SPUT, TASTE, PEEK, DECOD und OTYPE sind Programme, die System Objects enthalten.

Diese erzeugt man wie folgt:

Beispiel: SPUT

Zunächst folgendes Programm eingeben:

```
« 3 DUPN R→C DROP NUM DROP A
B »
```

Mit P→L das Programm in eine Liste umwandeln.

DUP 'A' POS liefert die Position von A in der Liste: 8.

#C1DDh OSA PUT ersetzt das A durch die Adresse #C1DDh (OSA wandelt den BinaryInteger #C1DDh in die Adresse #C1DDh, die als System Object angezeigt wird, da unter dieser Adresse eine Routine steht, für die der HP keinen Namen hat).

DUP 'B' POS liefert 9.

Jetzt den Quelltext des 2. System Objects eingeben, dieses mit Q→O umwandeln und mit LIST→ DROP frei setzen. Mit PUT wird nun das B durch dieses Maschinenprogramm ersetzt.

Mit L→P wird die Liste in ein Programm umgesetzt.

'SPUT' STO fertig.

Nun kann es aber vorkommen, daß sich die Platzhalter (oben waren es A und B) in einem eigenständigen Objekt befinden. Dann kann mit PUT und GET nicht direkt auf die Platzhalter zugegriffen werden und POS findet sie nicht.

Dann muß mit GET erst dieses eigenständige Objekt aus der Liste geholt und dann mit PUT, GET, POS, L→P und P→L eingegriffen werden.

Beispiel:

« IF 4 5 + THEN END »

Ersetze + durch - .

P→L DUP (+) LIST→ DROP POS ergibt 0. + wird also nicht gefunden.

DUP 3 GET liefert 4 5 +

Dies ist ein Programm, weshalb man es mit P→L in eine Liste wandelt. Jetzt funktioniert

DUP (+) LIST→ DROP POS

und liefert 3.

(-) LIST→ DROP PUT

ersetzt + durch - . Jetzt muß die Liste wieder in ein Programm zurück gewandelt werden und an die 3. Stelle in obige Liste gebracht weren:

L→P 3 SWAP PUT.

Anschließendes L→P macht auch hier wieder aus der Liste ein lauffähiges Programm.

Natürlich kann ein eigenständiges Objekt wieder in einem eigenständigen Objekt stehen, das kann wieder in einem eigenständigen Objekt stehen usw. Dann muß man natürlich sooft GET ausführen, bis man bei dem Platzhalter angelangt ist.

COD (in der Maschinenversion) und POKE stehen ohne Programmklammern im Ausdruck. Das ist aber egal.

Weil mir das Eintippen aber immer noch zu mühsam war, habe ich die Programme CP→DP und DP→CP geschrieben, die die Eingabe auch noch so verschachtelter Objekte zum Kinderspiel machen.

Näheres kann der zugehörigen Dokumentation entnommen werden.

Namenerklärungen:

P→L: Programm in Liste umwandeln

L→P: Liste in Programm umwandeln

ASO: adress of stack object

OSA: Umkehrung von ASO
COD: wandelt HEX-String in Byte-String
ESNL: erase spaces and newlines
Q→O: Quelltext in Objekt wandeln

Die wichtigsten Routinen

Es handelt sich um eine Sammlung von Routinen, Strings und Programmen, die den Umgang mit Maschinenprogrammierung und anderen Manipulationen der Software auf dem HP28S stark vereinfachen. Gewöhnliche (d.h. ohne Maschinenprogrammierung funktionierende Programme) sind nur soweit ausgedruckt, wie sie für die Schaffung der Maschinenprogramme unbedingt notwendig sind (Beispiel ESNL).

Die einzelnen Programme und Strings sind nicht sinnvoll sortiert ausgedruckt, sondern in der Reihenfolge, wie ich sie in den Rechner eingegeben habe.

Der Grund dafür ist, daß ich so immer weiß, in welcher Reihenfolge ich die Programme am geschicktesten eingeben kann.

Viele (fast alle) dieser Programme enthalten System Objects. Für alles, wofür der Rechner keinen Namen hat, schreibt er System Object. Deshalb würde beim Ausdrucken dieser Programme nicht deren wirklicher Inhalt auf dem Papier erscheinen sondern "System Object", was keinerlei Rückschlüsse auf die jeweilige Funktion zuläßt.

Aus diesem Grund wurden alle Programme vor dem Print mit Hilfe des Programms CP→DP "entschlüsselt". Das Programm DP→CP "verschlüsselt" das in Ebene 1 stehende Programm wieder und macht es so ausführbar.

Details dazu siehe Dokumentation für CP→DP und DP→CP.

```
'ESNL'
« -12 SWAP 1 2
  START SWAP 22 +
SWAP
    WHILE DUP 3 PICK
CHR POS
    REPEAT LAST DUP2
1 - 1 SWAP SUB ROT
ROT 1 + OVER SIZE
SUB +
    END
  NEXT SWAP DROP
»

127 BYTES
```

```
'QOSA'
"69A20 76C20 BBCB0
 B35C0 69C20 12000
 143 133 174 147 133
 145 142 164 808C
 09F20 09F20"

98.5 BYTES

'QASO'
"69A20 76C20
 11920 00000
 01670
 69C20 03000
 133 131 100 147 174 143
 133 174 145 110 133 142
 164 808C
 A2670 016C0 79CB0
 09F20 09F20"

143.5 BYTES

'QL→P'
"69A20 69C20 73000
 143 133 100 143
 3469A20 8A6C0 3476C20
 145 110 133 142 164
 808C 09F20"

100.5 BYTES

'COD'
« DUP NUM DROP DUP
SIZE DUP LN DROP 2
MOD
  IF
  THEN "0" SWAP +
  END SUBCODE
"69C201C0000504143133101
1741471331 31C210A1361081
3016417415361713193 9EAF0
31649EAA16F5031039EE6062
50B6A671031149EE606E3031
73B6A15801601 1A1331318B2
8A11034F7000CA1301111311
42164808C110130111131142
164808C"
"Bad Source" 1 DISP
1400 .075 BEEP UNDO
KEYN DROP KILL 1
OVER SIZE 2 / SUB
»

248.5 BYTES

'OSA'
B→R SUBCODE "B35C0"
SUBCODE
"69C2012000143133174147 1
331451421648 08C"

36.5 BYTES

'ASO'
SUBCODE "119200B40C"
SWAP SUBCODE
"69C2003000133131100147 1
7414313317414511013314 21
64808C"
DROP SUBCODE "016C0"
R→B

54 BYTES

'Q→O'
« ESNL DUP COD ASO
SWAP SIZE 2 MOD + 10
+ OSA
»

72.5 BYTES
```

```
'C↔D'
«
  WHILE DUP SIZE
  REPEAT DUP 1 GET
    IF DUP
      IF 2 FS?
      THEN 'SUBCODE'
      ELSE
        IF DUP TYPE
2 ==
        THEN DROP 1
        ELSE →STR
        END
"System Object"
      END SAME
    THEN
      IF 2 FS?
      THEN SWAP 2
OVER SIZE
        IF DUP 2 ≥
        THEN SUB DUP
1 GET
          IF DUP
TYPE 2 ==
          THEN Q→O
ROT DROP
          ELSE SWAP
+ SWAP
          END
        ELSE DROP2
SWAP
        END
      ELSE 'SUBCODE'
SWAP O→Q 2 →LIST
      END P&
    ELSE
      IF DUP TYPE 5
==
      THEN { } SWAP
C↔D DROP 1 →LIST P&
      ELSE COPY P→L
        IF DUP TYPE
5 ==
        THEN { }
SWAP C↔D DROP COPY
L→P P&
        ELSE
          IF 2 FC?
          THEN
            IF { 2 7
8 9 } OVER TYPE POS
NOT
            THEN
              IF DUP
1 →LIST DUP →STR
IFERR STR→
THEN
END ≠
              THEN
O→Q 'SUBCODE' SWAP 2
→LIST
              END
            END
          END P&
        END
      END
    END
  END
»

647.5 BYTES

'P$'
«
  IF DUP TYPE 5 ==
  THEN { } SWAP C↔D
DROP
  END
»

61.5 BYTES

'P&'
« ROT SWAP + SWAP 2
OVER SIZE SUB
»

36.5 BYTES
```

```
'DP→CP'
« 2 SF 1 CF
  IF DUP TYPE 8 ==
  THEN COPY P→L { }
SWAP C↔D DROP COPY
L→P
  ELSE P$
  END
»
```

110.5 BYTES

```
'L→P'
SUBCODE
"69C207300014313310014334
69A208A6C03476C20145110
133142164808C"
```

37.5 BYTES

```
'P→L'
SUBCODE
"69C207300014313310014334
76C208A6C03469A20145110
133142164808C"
```

37.5 BYTES

```
'COPY'
« 1 →LIST LIST→ DROP
»
```

28.5 BYTES

```
'CP→DP'
« 2 CF 1 SF DUP COPY
  IF DUP TYPE 8 ==
  THEN P→L
    IF DUP TYPE 5 ≠
    THEN SWAP 0→Q
'SUBCODE' SWAP 2
→LIST
    ELSE { } SWAP
C↔D DROP
    END L→P
  ELSE P$
  END SWAP DROP 1 CF
»
```

182.5 BYTES

```
'O→Q'
« DUP ASD
  IF # 40000h 1 FS?
≠ ≥
  THEN DUP ~~'SUBCODE'~~
'A' "" DUP CRDIR EVAL
STO MEM CLUSR MEM -
BACK '" PURGE NEG
10.5 + 2 ≠ ""
    DO DUP SIZE 4
PICK ASD + PEEK
DECOD +
    UNTIL DUP2 SIZE
<
    END ROT DROP 1
ROT SUB
  ELSE ASD ASD PEEK
DECOD 11 15 SUB
  END
»
```

276 BYTES

```
'BACK'
« PATH DUP SIZE 1 -
DUP SUB LIST→ DROP
EVAL
»
```

43.5 BYTES

```
'OSIZE'
« RCL LAST '" DUP
CRDIR EVAL STO MEM
CLUSR MEM - BACK '"
PURGE NEG 16 +
»
```

89 BYTES

```
'PEEK'
« B→R R→B SUBCODE
"69C20C200014313216914613
6156713214216E15CF189808
C"
DROP "DUMMYSTR" >>
```

63.5 BYTES

```
'DECOD'
« DUP NUM DROP DUP
SIZE LN DROP DUP +
SUBCODE
"69C20E90000504201431331
0117414334500000EEC6C6C6F
617413313610813613 0CA133
3400000DA15B017034900008
BAB03470000CA3403000CA15
8116113613413313188 6CB11
9137118136142164808C"
»
```

123.5 BYTES

```
'KEYN'
« SUBCODE "668F1"
SUBCODE "CC791" ASD
B→R 27636 -
  IF
  THEN SI→R
  ELSE 0
  END
»
```

70.5 BYTES

```
'SI→R'
« SUBCODE "016C0"
»

'R→SI'
« SUBCODE "B35C0"
»

'UNDO'
« SUBCODE "A43E3"
»

'KK'
« PATH → P
  « HALT P L→P EVAL
  »
»
```

47 BYTES

```
'SPEED'
« R→B SUBCODE
"69C207300014313317920 15
F0133174E71321B00FFF15C0
132142164808C"
»
```

52 BYTES.

```
'SLOW'
« SUBCODE
"69C20520002030713 21B00F
FF15C0132142164808C"
»
```

39.5 BYTES

```
'FAST'
« SUBCODE
"69C20520002030F1321B00F
FF15C0132142164808C"
»
```

39.5 BYTES

```
'RPL'
« { P→L L→P ASD DSA
SI→R R→SI PEEK DECOD
COPY COD ESNL OSIZE
BACK 0→Q Q→0 CP→DP
DP→CP } MENU
»
```

149.5 BYTES

```
'URCL'
« VARS 1 OVER SIZE
  FOR A DUP A GET
    IFERR RCL LAST
ROT
    THEN DROP
    END
  NEXT DROP
»
```

72.5 BYTES

```
'TRENN'
« DUP2 1 SWAP SUB
ROT ROT 1 + OVER
SIZE SUB
»
```

47 BYTES

```
'ALG→P'
« SUBCODE
"69C207300014313310014334
ADA208A6C03476C20145110
133142164808C"
P→L {
  « } SWAP + {
  » } + L→P
»
```

90 BYTES

```
'TYP'
« DUP DROP SUBCODE
"69C20020001431330614313
71411421648 08C"
ASD
»
```

47.5 BYTES

```
'NOP'
« { SUBCODE "C2422"
SUBCODE "197E1" }
LIST→ DROP SUBCODE
"DB460"
»
```

35 BYTES

```
'PNOP'
« { SUBCODE "197E1"
} LIST→ DROP PURGE
»
```

33.5 BYTES

```
'CONT'
« SUBCODE "92622"
»

'O→S'
« SUBCODE "4B962"
»
```

```
'SHIFT'
« SUBCODE "734F3"
»

'CMEN'
« 0 SUBCODE
"69C2073000342D00C136142
1363497C818A6B03498040DA
141142164808C"
»

51 BYTES


'FF'
« 23 MENU # 25AFCh
SYSEVAL " DIR: "
PATH 2 OVER SIZE SUB
DUP SIZE 1 - DUP SUB
LIST→ DROP DUP EVAL
SUBCODE "61FC3" +
SUBCODE "58A81"
SUBCODE "A5832"
SUBCODE "AAA81"
»

106 BYTES


'CTEMP'
« M→SI CMEN 2 →LIST
→ M
  « 1 OVER SIZE
    FOR A A DUP2 GET
{ SUBCODE "72650" }
OVER NB→S + SWAP + {
CONT } + L→P PUT
    NEXT MENU MDISP
HALT M LIST→ DROP
MENU SI→M CLMF
  »
»

162.5 BYTES

'NB→S'
« SUBCODE "61FC3"
»

'M→SI'
« SUBCODE "21EF1"
»

'SI→M'
« SUBCODE "99002"
»

'GVARS'
« SUBCODE "BB971"
»

'SMF'
« SUBCODE "58A81"
»

'ZEIG'
« →STR SMF SUBCODE
"A5832"
»

'MDISP'
« SUBCODE "5DE81"
»

'OFF'
« SUBCODE "85E81"
»

'EDIT'
« SUBCODE "D9FB1"
»

'BOFF'
« SUBCODE "668F1"
»
```

SPEK zeichnet ein Linienspektrum zu einer in EQ gegeben Gleichung in Programmform. Der Linienabstand wird durch RES bestimmt. In EQ wird als letzter Befehl SPEK eingesetzt.

Ein Beispiel:

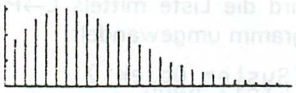
'2^X/FACT(X)' soll als Linienspektrum dargestellt werden.

Dazu wird das Programm

```
« X 2 OVER ^ SWAP FACT / SPEK
»
```

in EQ gespeichert und dann DRAW ausgeführt.

Das Ergebnis:



```
PPAR
{ (0,0) (6.8,2.2) X
5 (0,0) }

SPEK
« DUP PPAR LIST→ 3
DROPN EVAL 4 ROLLD
SWAP - IM DUP2 / 32
* ABS CEIL ROT ROT
DROP OVER / 0 ROT 1
-
  FOR A DUP2 A * R→C
PIXEL
  NEXT DROP2
»

123 BYTES

PEEK
« B→R R→B
ILMC { 69C20 C2000
143 132 169 146 136
1567 132 142 16E 15CF
189 808C } DROP
"DUMMYSTR"
»
63.5 BYTES


DECOD
« DUP NUM DROP DUP
SIZE LN DROP DUP +
System Object
»
123.5 BYTES

Das System Objekt ist:
QDECOD
"
69A20 69C20 E9000
05 04 20 143 133 101
174 143 3450000 EE
C6 C6 C6 F6 174 133 136
108 136 130 CA 133
3400000 DA 15B0 170
3490000 8BAB0 3470000
CA 3403000 CA 1581 161
136 134 133 131 8B6CB
119 137 118 136
142 164 808C 09F20"


COD
DUP NUM DROP DUP
SIZE DUP LN DROP 2
MOD
IF
THEN "0" SWAP +
END System Object
"Bad Source" 1 DISP
1400 .075 BEEP UNDO
KEYN DROP KILL 1
OVER SIZE 2 / SUB

243.5 BYTES

Das System Objekt:

QCOD
"
69A20
69C20 1C000
05 04 143 133 101 174
147 133 131 C2 10A 136
108 130 164 174 1536
171 3193 9EAF0 3164
9EAA1 6F50 3103 9EE60
6250 B6A 6710
3114 9EE60 6E30 3173
B6A 1580 160 11A 133
131 8B28A 110 34F7000
CA 130 111 131 142 164
808C 110 130 111 131
142 164 808C
09F20"
```

OTYPE gibt im Klartext den Objekttyp eines in Ebene 1 stehenden Objektes in den Stack. Es funktioniert anscheinend immer.

```
OTYPE
« DUP ASO SWAP
System Object ASO
DOP {
  «
    IF DUP2 5 - ==
    THEN "Mcode"
    ELSE "???"
    END
  » "Short Integer"
"Real" "Long Real"
"Complex"
"Long Complex"
"Byte" "Array 1"
"Array" "Array 2"
"String" "Integer"
"List" "Directory"
"Algebraic"
"RPL-Program"
"In-Line-Mcode"
"Global Name"
"Lokal Name"
"ROM-Pointer" "End"
} { # 2911h # 2933h
# 2955h # 2977h
# 299Dh # 29BFh
# 29E1h # 2A0Ah
# 2A2Ch # 2A4Eh
# 2A70h # 2A96h
# 2AB8h # 2ADAh
# 2C67h # 2C96h
# 2D12h # 2D37h
# 2D5Ch # 2F90h }
ROT POS 1 + GET EVAL
ROT ROT DROP2
»

666 BYTES
```

Das System Objekt wird mit folgendem Quelltext erzeugt.

```
69A20
69C20 02000
143 133 D6 143 137
141 142 164 808C
09F20
```

TYP: `« DUP DROP SystemObject ASO »`

```
L→P

69A20       LISTE
69C20       INLINE-MCODE
73000       LEN
143         A=DAT1 A
133         AD1EX
100         R0=A
143         A=DAT1 A
3469A20     C=#02A96   ← ①
8A6 C0      ?A≠C A
            GOYES#0C
3476C20     C=#02C67   ← ②
145         DAT1=C A
110         A=R0
133         AD1EX
142         A=DAT0 A
164         D0=D0+5
808C        PC=(A)
09F20       ENDE LISTE
```

L→P wandelt eine in Ebene 1 stehende Liste in ein Programm, indem 69A20 durch 76C20 ersetzt wird. Wenn in Ebene 1 keine Liste steht passiert nichts.

Programm P→L: Oben Zeile 1 mit Zeile 2 vertauschen

```
OSA

69A20       LISTE
76C20       RPL-PRMG
BBCB0       B→R
B35C0       R→SI
69C20       INLINE-MCODE
12000       LEN
143         A=DAT1 A
133         AD1EX
174         D1=D1+5
147         C=DAT1 A
133         AD1EX
145         DAT1=C A
142         A=DAT0 A
164         D0=D0+5
808C        PC=(A)
09F20       ENDE RPL-PRGM
09F20       ENDE LISTE
```

OSA ist die Umkehrung von ASO.

OSA nimmt einen Binary Integer (Format (also HEX, DEC, OCT, BIN) ist egal, da BinaryIntegers intern immer als HEX-Kombinationen gespeichert werden) vom Stack und ersetzt ihn (besser gesagt die Adresse im Stack) durch den Wert des Binary Integers. Ist dessen Wert ≥ #FFFFFh, so wird er zu #FFFFFh transformiert (dies macht R→SI).

Es steht somit nach der Ausführung von OSA die durch den Binary Integer spezifizierte Adresse im Stack.

Wie man mit ASO und L→P NOP und PNOP wesentlich verkürzen kann:

```
LNOP 1 →LIST
« LIST→ DROP » +
ergibt

1: « «System Objekt»
     LIST→ DROP »
```

Der Abschnitt #64BDh SYSEVAL wird ersetzt durch die Adresse DB460. Dies geschieht so:

Da nur die Adresse gebraucht wird reicht folgendes:

```
#64BDh OSA 1 →LIST
ergibt

2: « «Sytem Objekt»
     LIST→ DROP »
1: «System Objekt».
```

Jetzt werden die 2 Listen mit + zusammen gefügt:

```
1: « «System Objekt»
     LIST→ DROP
   System Objekt»
```

Nun wird die Liste mittels L→P in einProgramm umgewandelt:

```
1: «System Objekt»
     LIST→ DROP
     System Objekt

dann 'NOP' STO .
```

Für PNOP:

```
LNOP 2 2 SUB 1 →LIST
« LIST→ DROP PURGE » +
ergibt

1: « « » LIST→ DROP
   PURGE »
```

Auch diese Liste wird mittels L→P in ein Programm umgewandelt:

```
1: « » LIST→ DROP PURGE

dann 'PNOP' STO .


NOP
« System Object »
LIST→ DROP
System Object

30 BYTES

PNOP
« » LIST→ DROP PURGE

29.5 BYTES
```

Mit P→L und OSA können leicht System Objects in Programme gebracht werden.

Beispiel:

```
1: « SWAP MDISP + »
```

Hier soll MDISP durch die Adresse #18AAAh ersetzt werden.

```
P→L 3
#18AAAh OSA PUT L→P

ergibt

1: « SWAP System Object
     + »
```

Zu P→L und L→P

Wird auf ein im Stack stehendes Objekt P→L angewendet, so stellt jeder Befehl ein Objekt dar.

Beispiel:

```
1: « 4 « + » DROP 'A'
   STO »

P→L ergibt

1: « « 4 « + » DROP 'A'
   STO » »

LIST→


10: «
9: 4
8: 7: « +
»
6: DROP
5: '
4: 'A'
3: '
2: STO
1: »
```

Das Objekt in Ebene 8 hat die Adresse #E862h und ist das NOEVAL, das der Rechner vor Programme setzt, die in einem Programm stehen. Das NOEVAL verursacht den seltsamen Ausdruck (es wurde PRST verwendet).

Die Objekte in Ebene 3 und 5 dürfen nicht voneinander getrennt werden, sonst folgt MEMORY LOST (LIST→ stellt die absolute Ausnahme dar, sollte aber auch vermieden werden, da bereits `4 ROL DROP 9 →LIST` zu MEMORY LOST führt). Die einfachen Anführungszeichen können durch `«...{A} LIST→ DROP... »` vermieden werden. Die Liste {A} ist ein Objekt (eine andere Möglichkeit: Den NOEVAL vor das A ohne einfache Anführungszeichen setzen).
Es darf jedoch aus obiger Liste der Name mittels `7 GET` gewonnen werden. Auch darf `7 'HALLO' PUT` ausgeführt werden. Jedoch würde, wie erwähnt, `1 6 SUB` (sowie `8 10 SUB`) zu MEMORY LOST führen, weil die Hochkommatas dadurch getrennt werden würden.
Generell sollte die Syntax nicht verletzt werden.
Es ist jedoch möglich, Schleifenbefehle zu trennen und/oder zu ersetzen. Beispiel:

```
1:  « START 1 STEP »
P→L 2 2 SUB
1:  { START }
« START NEXT » P→L 3 3 SUB +
1: { START NEXT }
L→P
1:   START NEXT
```

NOEVAL kann auch wie folgt verwendet werden:

```
#E862h OSA  'NOEVAL' STO
          ↑
```

hier liegt das Objekt, das unter der Adresse #E862h gespeichert ist im Stack. Es ist unsichtbar (!), weshalb man meinen könnte, der Stack sei leer.

Verwendung von NOEVAL in Programmen:

Beispiel:

```
7'A' STO
« ... NOEVAL A » EVAL
```

legt 'A' in den Stack, obwohl A bekannt ist!

Beachte: `« #E862h SYSEVAL » 'NOEVAL' STO` funktioniert nicht im obigen Beispiel! Es muß immer der obige Weg mit OSA gewählt werden.

`KEYN`
`« #1F866h SYSEVAL #197CCh SYSEVAL ASO B→R 27636 - IF THEN SI→R ELSE 0 END »` gibt für alle Tasten eine tastenspezifische reelle Zahl in den Stack. Das Programm wartet etwa 10 Minuten auf einen Tastendruck, dann schaltet sich der Rechner automatisch ab (Auto-Power-Off).

`M→SI`
`« #1FE12h SYSEVAL »` gibt durch momentanes Menu spezifizierten SI (=Short Integer) in den Stack.

`SI→M`
`« #20099h SYSEVAL »` nimmt SI vom Stack und schaltet das mit dem SI spezifierte Menu ein.

```
ZVARS
« RCEQ M→SI CMEN ROT
{ MEHR ENDE } MENU
DUP 1 CLLCD DISP
GVARS 1 OVER SIZE
  FOR A DUP A GET
DUP →STR 2 OVER SIZE
1 - SUB "= " + SWAP
   IFERR RCL
   THEN DROP
"undefined"
   END →STR + 3
DISP MDISP
    DO KEYN
      IF DUP 74 ≠
      THEN
          IF 80 ==
          THEN DUP
SIZE 'A' STO 1
          ELSE 377 .07
BEEP 0
          END
       END
    UNTIL
    END
  NEXT DROP MENU
SI→M CLMF
»

335.5 BYTES

M→SI
« # 1FE12h SYSEVAL
»
34 BYTES

SI→M
« # 20099h SYSEVAL
»
34 BYTES

GVARS
« # 179BBh SYSEVAL
»
34 BYTES

KEYN
« # 1F866h SYSEVAL
# 197CCh SYSEVAL ASO
B→R 27636 -
  IF
  THEN SI→R
  ELSE 0
  END
»
96.5 BYTES

SI→R
« # C610h SYSEVAL
»
34 BYTES

R→SI
« # C53Bh SYSEVAL
»
34 BYTES

Q→D
« ESNL DUP COD ASO
SWAP SIZE 2 MOD + 10
+ OSA
»

72.5 BYTES

COD
« RCLF 8 STWS HEX ""
"0" 4 ROLL + DUP
SIZE 2 MOD 1 + OVER
SIZE SUB
  WHILE DUP SIZE
  REPEAT DUP 3 OVER
SIZE SUB "#" ROT 1 2
SUB + STR→ RR RR RR
RR B→R CHR ROT SWAP
+ SWAP
  END DROP SWAP STOF
»

149.5 BYTES

ASO

69A20          LISTE
76C20          RPL-PRGM
11920 00000    SI#0
01670          SWAP
69C20          INLINEMCODE
03000          LEN
133            AD1EX
131            D1=A
100            R0=A
147            C=DAT1 A
174            D1=D1+5
143            A=DAT1 A
133            AD1EX
174            D1=D1+5
145            DAT1=C A
110            A=R0
133            AD1EX
142            A=DAT0 A
164            D0=D0+5
808C           PC=(A)
A2670          DROP
016C0          SI→R
79CB0          R→B
09F20          ENDE PRGM
09F20          ENDE LISTE
```

Diese Version ist optimal, sie kann nicht mehr verbessert werden. Ein Versuch, bei dem SI#0 durch #0h (07A20 A0000 0000) und der zweite 174 (D1 = D1 + 5) durch 179 (D1 = D1 + #A) ersetzt wurden und SI→R sowie R→B gestrichen wurden, scheiterte, weil OTYPE nicht mehr richtig arbeitete, obwohl ASO interaktiv betrieben in Ordnung war! Grund: Die oben stehende Version liefert einen Integer, dessen Wert im Stack definiert ist und seine Länge ist 64 Bit und nicht 20 Bit wie oben!

```
QP→L
"
69A20 69C20 73000
143 133 100 143
3476C20 8A6C0 3469A20
145 110 133 142 164
808C 09F20"
```

nach ESNL: 70 Zeichen

```
QL→P
"
69A20 69C20 73000
143 133 100 143
3469A20 8A6C0 3476C20
145 110 133 142 164
808C 09F20"
```

nach ESNL: 70 Zeichen

```
QASO
"
69A20 76C20
11920 00000
01670
69C20 03000
133 131 100 147 174 143
133 174 145 110 133 142
164 808C
A2670 016C0 79CB0
09F20 09F20"
```

nach ESNL: 103 Zeichen

```
QOSA
"
69A20 76C20 BBCB0
B35C0 69C20 12000
143 133 174 147 133
145 142 164 808C
09F20 09F20"
```

nach ESNL: 68 Zeichen

QALG→UPN:

wie QL→P, jedoch 3469A20 durch 34ADA20 ersetzen.

```
'X*F' ALG→UPN
      ----------->  X F *
       « X F * »
```

(Die Umkehrung UPN→ALG ist verboten wegen hoher Absturzgefahr).

"Nette Version" von ALG→UPN:

```
« System Object P→L {«} SWAP
+ {»} + L→P »
```

das System Object ist das mit QALG→UPN erzeugte Programm.

Neue Möglichkeit, Quelltexte in die Zielobjekte umzuwandeln:

```
1: (Quelltext)
   ↓ESNL DUP COD HOME
   'A' STO #D0000h
   SWAP SIZE-
1: (Binary Integer)
        ↑
   Adresse, wo Zielobjekt beginnt
```

jetzt nicht SYSEVAL, sondern OSA! Dies führt das Zielobjekt nicht aus, sondern zeigt es lediglich im Stack an. Es entspricht also einem RCL, bei dem kein Name, sondern eine Adresse spezifiert wird.

Folgendes Programm arbeitet wie oben angegeben, jedoch entfällt die Notwendigkeit der Speicherung des kodierten Strings im TOP OF RAM.

```
1: (Quelltext)
   ↓Q→O
1: (Zielobjekt)
```

```
Q→O (Quelltext in Objekt)
« ESNL DUP COD ASO
  SWAP SIZE 2 MOD +
  10 + OSA »
```

72,5 Bytes
(es dauert wegen ESNL und COD etwas).

Beispiel:

```
1: "69A20 9C211 09F20"
   ↓Q→O
1: {1}
```

Damit Q→O funktioniert, muß der Quelltext mit einer einen bestimmten Objektyp definierenden Adresse beginnen. Hier im Beispiel ist dies eine Liste (69A20); im Beispiel ist also der Quelltext "9C211" (nur die 1) nicht zulässig.

```
SPUT
« 3 DUPN R→C DROP
NUM DROP
System Object
System Object
»

101 BYTES

1. System Object:
# 0C1DDh

2. System Object:

69A20
69C20 D7000
7E50 100 7750 101
147 137 174 143 137
3450000 EA 81C 119
8B292 CE F2 81E 81E 81E
143 CA 34A0000 CA 118
133 14D 133 142 164
808C 147 137 174 143
137 174 E7 01
09F20

Funktion:

3:          (Zielstring)
2:(n-tes Byte=Position)
1:          (ASCII-Code)
```

SPUT setzt das durch die reelle Zahl in 1 spezifizierte Zeichen an die durch die reelle Zahl in 2 spezifizierte Stelle in den Zielstring.

Dabei wird das vorher an dieser Position befindliche Zeichen überschrieben.

Ist die Zahl in 2 größer als die Stringlänge oder ≤ 0, so bleibt der String unverändert.

```
SGET
« DUP2 CHR DROP NUM
DROP 983040 + R→SI
System Object
System Object
»

135.5 BYTES

Die 2 System Objects:

69A20 69C20 E9000
147 137 174 143 137 174
F0 81C 81C 81C 100
147 137 174 143
137 3450000 EA 118
8B244 CE CE 143 CA
34A0000 CA 133 14F 133
1C4 143 133 174 145 133
143 174 141 E7 142 164
808C
3468311 145 164 142 164
808C 016C0
09F20

Funktion:

2:          (Quellstring)
1:(n-tes Byte=Position)
```

zu langsam!

SGET gibt den ASCII-Code des an der durch die reelle Zahl in 1 spezifizierten Stelle stehenden Zeichens aus dem Quellstring in den Stack.

Ist die Zahl in 1 größer als die Stringlänge oder ≤ 0, so wird -1 in den Stack gelegt.

Schneller: `« DUP SUB NUM »`

```
POKE
B→R R→B OVER SIZE 8
- NOT 3 PICK TYPE 2
- NOT * LN DROP
System Object

90.5 BYTES

Das Systemobject:

69A20 69C20 54000
143 133 179 147 133 174
E7 143 133 100 179 1537
137 1517 110 133 174 E7
142 164 808C 09F20
```

Abstürze, die durch die Behandlung spezieller Objekte hervorgerufen werden:

1.) leerer algebraischer Ausdruck:

Könnte er angezeigt werden, so würde er so aussehen: ''

```
Code:  ADA20 09F20
          ↑     ↑
      objekt-   End
    definierte
  ende Adresse
   des algebr.
    Ausdrucks
```

Der Rechner stürzt nicht ab, wenn a.)

`« "ADA20 09F20" Q→O DROP »`

oder b.)

`« "ADA20 09F20" Q→O EVAL »`

ausgeführt wird, da in beiden Fällen der leere algebraische Ausdruck nicht angezeigt werden muß. Nach b.) ist der leere algebraische Ausdruck verschwunden, weshalb hier kein DROP nötig ist.

c.)

`« "ADA20 09F20" Q→O 1 →LIST DROP »` führt aus dem gleichen Grund nicht zum Absturz.

Wird jedoch d.)

`« "ADA20 09F20" Q→O →STR DROP »` ausgeführt, so stürzt der Rechner ab.

Auch wenn nach a.), b.) oder c.) LAST von Hand ausgeführt wird stürzt der Rechner ab, weil der leere algebraische Ausdruck dann in den Stack gelegt wird und der Rechner dann versucht, diesen anzuzeigen.

Kurz: Leerer algebraischer Ausdruck führt zum Absturz, wenn er entweder angezeigt oder in einen String gewandelt werden soll.

2.) Apostroph:

Namen können in einem Programm

mit Apostrophen versehen werden um deren direkte Auswertung zu verhindern:

```
« 'A' »
```

wird so ein Programm mittels P→L in eine Liste gewandelt ({« 'A' »}) und 2 GET ausgeführt, so liegt ein Apostroph im Stack.

Der Rechner stürzt nicht ab, wenn

a.) `« 2 GET →STR »`
oder
b.) `« 2 GET DROP »`
oder
c.) `« 2 GET TYPE »`
ausgeführt wird, weil in diesen Fällen nicht nach dem 2. Apostroph gesucht werden muß.
Wird jedoch
d.) `« 2 GET 1 →LIST »`
ausgeführt, so stürzt der Rechner ab, weil `1 →LIST` eine vollständige Kopie des Objekts in die Liste legt. Vollständige Kopie heißt aber, daß der Rechner nach dem 2. Apostroph suchen muß. Ebenso stürzt der Rechner ab, wenn nach a.), b.) oder c.) LAST ausgeführt wird, weil er wieder nach dem 2. Apostroph sucht, es aber nicht finden kann.

```
CD→DP
« 2 CF 1 SF DUP COPY
  IF DUP TYPE 8 ==
  THEN P→L
    IF DUP TYPE 5 ≠
    THEN SWAP 0→Q 'SUBCODE' SWAP 2 →LIST
    ELSE { } SWAP C←→D DROP
    END L→P
  ELSE P$
  END SWAP DROP 1 CF
»
```

182,5 Bytes

```
DP→CP
« 2 SF 1 CF
  IF DUP TYPE 8 ==
  THEN COPY P→L { } SWAP C←→D DROP
                                COPY L→P
  ELSE P$
  END
»
```

110,5 Bytes

```
P$
« IF DUP TYPE 5 ==
  THEN { } SWAP C←→D DROP
  END
»
```

61,5 Bytes

```
P&
« ROT SWAP + SWAP 2 OVER SIZE SUB »
```

36,5 Bytes

```
C←→D
« WHILE DUP SIZE
  REPEAT DUP 1 GET
    IF DUP
      IF 2 FS?
      THEN 'SUBCODE'
      ELSE
        IF DUP TYPE 2 ==
        THEN DROP 1
        ELSE →STR
        END "System Object"
      END SAME
    THEN
      IF 2 FS?
      THEN SWAP 2 OVER SIZE
        IF DUP 2 ≥
        THEN SUB DUP 1 GET
          IF DUP TYPE 2 ==
          THEN Q→0 ROT DROP
          ELSE SWAP + SWAP
          END
        ELSE DROP2 SWAP
        END
      ELSE 'SUBCODE' SWAP 0→Q 2 →LIST
      END P&
```

Flag 2 gesetzt: Programm in Ebene 1 chiffrieren
Flag 2 gesetzt: Programm in Ebene 1 dechiffrieren

Strings nicht testen, ob sie "System Object" enthalten

alle anderen Objekte testen, ob sie ein System Object sind

diese Schleife testet, ob Syntax (nach Schlüsselwort SUBCODE muß ein String folgen) auch eingehalten wurde folgt hinter SUBCODE überhaupt
ja irgend ein Objekt?
teste ob dies auch ein String ist
ja, also chiffriere diesen String
nein, also nicht chiffrieren und Schlüsselwort SUBCODE nicht unterdrücken
nein, also behandle das Schlüsselwort wie jeden anderen Befehl

System Object decodieren

```
  ELSE
    IF DUP TYPE 5 ==                          ist Objekt eine Liste?
    THEN { } SWAP C←→D DROP 1 →LIST P&        ja: Listenelemente untersuchen
    ELSE COPY P→L                             oder ist Objekt ein Programm?
      IF DUP TYPE 5 ==
      THEN { } SWAP C←→D DROP COPY L→P P&      JA: Programmelemente untersuchen
      ELSE
        IF 2 FC?
        THEN
          IF {2,7,8,9} OVER TYPE POS NOT      wenn Objekt nicht String, nicht Programm, nicht
          THEN                                lokaler Name, nicht algebr. Ausdruck ist,dann
            IF DUP 1 →LIST DUP →STR           falls nicht normalisiertes Objekt
              IFERR STR→
              THEN
              END ≠
            THEN 0→Q 'SUBCODE' SWAP 2 →LIST   dann dechiffrieren
            END
          END
        END P&
      END
    END
  END
END »
```

645 + 2,5 Bytes

Wie aus dem Programm ersichtlich werden algebraische Ausdrücke nichr dekodiert, auch wenn sie System Objects enthalten.

Dies wäre auch sehr aufwendig, da die Absturzgefahr dabei sehr hoch ist (leere algebraische Ausdrücke führen fast immer zu Abstürzen und gewöhnliche Befehle oder Strings dürfen in algebraischen Ausdrücken nicht vorkommen).

Wird in dem zu chiffrierenden Programm gegen die Syntax verstoßen, daß hinter dem Schlüsselwort SUBCODE ein String stehen muß, so wird das Schlüsselwort nicht unterdrückt (siehe unten) und das darauf folgende Objekt nicht chiffriert und nicht unterdrückt.

Das Programm wird also in diesen Fällen an dieser Stelle nicht geändert, ganz entgegen zum

normalen Fall:

auf das Schlüsselwort SUBCODE muß ein String folgen; durch das Programm DP→CP wird dann das in Ebene 1 liegende Objekt (zum Beispiel ein Programm oder eine Liste) das Schlüsselwort und String enthält durchsucht. Wenn Schlüsselwort und String gefunden wurden, dann wird das Schlüsselwort entfernt und der String kodiert. Der String wird dann durch das kodierte Objekt ersetzt.

```
0→Q

« DUP ASO
  IF #40000h 1 FS? * ≥
  THEN DUP 'A' 'o' DUP
    CRDIR EVAL STO MEM
    CLUSR MEMBACK 'o'
    PURGE NEG 10,5 + 2 * ""
    DO DUP SIZE 4 PICK ASO
    + PEEK DECOD +
    UNTIL DUP2 SIZE <
    END ROT DROP 1 ROT SUB
  ELSE ASO ASO PEEK DECOD
  11 15 SUB
  END
»
```

276 Bytes

Beispiel:

```
1: 7 0→Q
   ----->
1: "33920 0000000000000070"
```

falls Flag 1 gelöscht

falls aber Flag 1 gesetzt ist:

```
1: 7 0→Q
     ----->
1: "74311"
```

(Adresse der Zahl 7, "falsch" herum).

Diese Unterscheidung findet nur bei Objekten statt, die im ROM gespeichert sind! (D.H. bei RAM-Objekten ist Flag 1 egal).

Grund: Der HP speichert ROM-Objekte im RAM nur als Zeiger ab, RAM-Objekte können jedoch auch vollständig kopiert werden (sie sind dann also mehrfach im RAM vorhanden, ein ROM-Objekt steht jedoch nur einmal im Rechner). Diese Besonderheit mußte für das Programm CP→DP eingeführt werden.

Der Aufruf von ASO in der Schleife ist nötig, da die Position des Objekts im RAM stehts neu bestimmt werden muß, da die Garbage Collection dieses Objekt unter Umständen im RAM verschiebt.

Mit DP→CP ist es sehr einfach, Maschinencode oder Systemroutinen in Programme einzubinden.

Beispiel: Es sollen einmal alle Meldungen, die der HP28S besitzt, angezeigt werden. Dazu muß folgendes Programm eingetippt werden:

```
ALLE

« 1 2818 FOR A A
     SUBCODE "B35C0"
     SUBCODE "31730"
```

```
    A 1 DISP
    IF DUP 2 DISP "" ≠
    THEN .2 WAIT
    END
  NEXT
»
```

Dieses Programm wird nun auf den Stack gelegt und DP→CP ausgeführt.

Nach etwa 3-4 Sekunden ist das Programm fertig, und man sieht, daß die Zeilen

```
SUBCODE "B35C0"
```

und

```
SUBCODE "31730"
```

durch System Object System Object ersetzt wurden!

Durch das Programm DP→CP wurde die Information, die in dem String hinter dem Schlüsselwort SUBCODE stand in das Programm ALLE eingebunden und das Schlüsselwort SUBCODE entfernt.

Schlüsselwort heißt, daß das Programm DP→CP nach diesem Shlüsselwort sucht.

Hat es das gefunden, so entfernt es dieses und übersetzt den String (der direkt auf dieses Schlüsselwort folgen muß!).

Übersetzen heißt hier, der ASCII-String wird in einen Binärstring umgewandelt und nur der Inhalt dieses Binärstrings in die Befehlssequenz des Programms (ALLE) eingefügt.

Das so übersetzte Programm kann nun abgespeichert werden und ist lauffähig.

Wenn man es startet, dann sieht man in Zeile 1 die jeweilige Meldungsnummer und in Zeile 2 die dazugehörige Meldung. Es sind aber nicht alle 2818 Nummern und Meldungen belegt.

Wenn nun das Programm verändert werden soll, so kann es nicht einfach editiert werden, da die System Objects durch den Editor zerstört werden.

Deshalb wandelt man das Programm mit CP→DP um. Dadurch werden die System Objects in die Strings umgewandelt. Damit diese auch eindeutig als Subcodes erkennbar sind, wird ihnen das Schlüsselwort SUBCODE vorangestellt.

Maschinencode läßt sich natürlich genauso einfach einbinden:

```
OSA

« B→R
  SUBCODE "B35C0"
  SUBCODE "69C20 12000
           143 133 174 147
           133 145 142 164
           808C" »
```

DP→CP liefert dann `« B→R System Object System Object »`

Achtung: Enthält das Programm einfache Anführungszeichen, so dürfen die Programme nicht mit SST durchgefahren werden! (z.B. `« 'A' »`) sonst MEMORY LOST, siehe P→L.

Peter Röhl
Osterfeuerbergstraße 70
2800 Bremen 1

Assemblermodul für den HP48SX

von Georg Hoppen

Es war einmal ... halt, beginnen nicht so die Märchen aus der guten "alten Zeit" ?
Die Zeit, als uns noch die SWAPS und DROPS umschwirrten und jedermann wußte, was einen erwarten konnte, sofern er die entsprechende Taste auf dem HP48er gedrückt hatte. Und heute ??

WARNUNG ! WARNUNG !
NICHT WEITERLESEN !
WARNUNG ! WARNUNG !

DAS LESEN DER FOLGENDEN ZEILEN MACHT SÜCHTIG; ES LÄßT DEN VORHERIGEN UMGANG MIT DEM 48er FAD UND TROSTLOS ERSCHEINEN - MIT EINEM WORT, WER NOCH LÄNGER SEINE RUHE BEWAHREN MÖCHTE, DER SOLLTE SPÄTESTENS HIER AUFHÖREN ZU LESEN...

Es war Ende März, als mich eine kleine, unschuldig aussehende Karte der Firma W&W erreichte.

"ASSEM 48" stand auf der Rückseite, ein (zu) kurz geratenes Loseblattwerk "Bedienungsanleitung" (es verdient nur den zweiten Teile der Bezeichnung), das war's dann auch schon. Inzwischen soll jedoch ein vollständig überarbeitetes Handbuch mit ausgeliefert werden, so der verantwortliche Programmierer.

Neugierig geworden steckte ich das Ding in den erwartungsvoll blickenden Port meines 48ers. Und nichts tat sich. W&W hatte leider einen Prototyp versandt, der noch gravierende Fehler beinhaltete. Fragende Blicke in die Bedienungsanleitung gaben zumindestens den Namen des Autors preis:
Raymond Hellstern - natürlich, er hatte doch schon einmal etwas im PRISMA über Maschinensprache veröffentlicht (PEEK-Funktion im Heft 2/90). Also kurzerhand die Telefonnummer besorgt und siehe da, ohne großen Aufwand bekam ich endlich eine lauffähige Testversion.

Und jetzt ging es so richtig los. Ein heller Stern begann über dem HP-Himmel zu leuchten.

ASSEMBLER = Übersetzungsprogramm zur Umwandlung einer maschinenorientierten Programmiersprache in die spezielle Maschinen-

sprache; so drückte sich zumindestens der Duden aus.

Na schön, aber was bedeutet das? Zunächst ist damit schlichtweg ausgedrückt, daß man auf der Betriebssystemebene des Rechners diesen dazu bewegen kann, "Pföt-chen zu geben", was heißen soll, daß man so ziemlich alles mit seinem Rechner anstellen kann.

Und nicht nur anstellen, sondern auch schneller. In der Regel ist der Tuningfaktor 3 - 500 oder größer erreichbar. Je nach Anwendung bringen z.B. Schleifenoperationen den größten Vorteil, da sich hier kleine Zeitvorteile zu großen Summen hochaddieren. Ebenso ist der Speicherbedarf ca. 30-50% geringer und man kommt an Dinge heran, die für den normalen Benutzer unzugänglich sind (z.B. das Betriebssystem-ROM).

Wie funktioniert das Ganze nun ?
Der Assembler arbeitet in zwei Phasen einen zeilenorientierten String ab, der folgenden Aufbau haben muß:

```
"<Label> <Mnemonic> <Parameter> <*Kommentar>"
```

oder

```
"<Label> <Makro> <*Kommentar>"
```

Hierbei ist jedes Argument <> durch ein Leerzeichen von dem nächsten zu trennen, um dem Assembler den richtigen Aufbau zu bieten. In zwei Schritten wird dann das Objekt umgewandelt. Im ersten wird die syntaktische Korrektheit geprüft, d.h. ob die Worte, die da stehen, auch an der richtigen Stelle stehen und bekannt sind. Im zweiten Schritt erfolgt dann die Umsetzung in den Hexwert, der den eigentlichen Maschinenbefehl darstellt.

Die Mnemonics (frei übersetzt: Gedächtnisstützen) sind ähnliche Anweisungen wie der USER-Code, also anstelle von SWAP und DROP steht hier z.B. SETHEX, GOTO oder RESET und GOVLNG zum Einsatz. Es handelt sich hier um einen maschinenorientierten Befehlssatz, der unter anderem im Buch von Mier-Jedrzejowicz beschrieben wird. ("Customize Your HP28", siehe PRISMA 1/91 Seite 23)

Der fast vollständige Befehlssatz wird in der Bedienungsanleitung des Moduls mit aufgeführt, leider nur viel zu kurz, so daß ein Anfänger in der Maschinensprache hoffnungslos allein gelassen ist.
(Siehe auch meinen Artikel Maschine&Sprache, TOOL, Teil 1, Eine Einführung in das Betriebssystem des HP48SX)

Das Ergebnis der sehr schnellen Übersetzung des Assemblerprogramms (natürlich wird auch hier reiner Machinencode und RPL angewandt) ist ein unschuldig aussehender Begriff "Code" im Stack bzw. in der Variablen COD.$ (dieses Objekt kann jetzt in jede Variable weggespeichert werden und so innerhalb eines Programms per Aufruf seine gewünschte Wirkung entfalten). Ein einfaches DUP und BYTES fördert jedoch eine überraschende Größenangabe zutage. Dies zeigt deutlich, daß der Begriff tatsächlich das ist, wofür er sich ausgibt, nämlich eine Anweisung für den Rechner, etwas bestimmtes zu tun.

Damit ist die Leistung der Assemblerkarte aber noch keineswegs erschöpft. Neben der eigentlichen Übersetzungsmöglichkeit wird dem Anwender noch ein umfangreiches TOOL-Paket zur Verfügung gestellt. Einige Befehle, die wir von Texteditoren kennen wie Zeilen einfügen, zeilenorientiertes Löschen, schnelle Stringbehandlung (Umdrehen, Suchen, Umwandeln usw.), Grafikbefehle und Displaybefehle stellen ein zusätzliches Hilfsmittel dar, so daß die Programme übersichtlicher, kürzer und schneller gestaltet werden können.

Zwar beziehen sich viele Befehle auf den Bereich der PEEKR und POKER, aber auch der normale Anwender findet überraschend viele Selbstverständlichkeiten, die es leider in der Grundausstattung des Rechners noch nicht gibt.

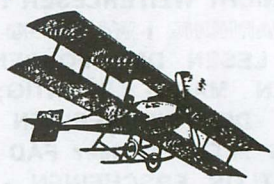
Die Bedienung des Moduls bedarf am Anfang einer gewissen Eingewöhnung, da das Eingehen auf die Maschinenbefehle bei syntaktischen Ungereimtheiten sehr schnell mit der beliebten Meldung "Memory Lost" belohnt werden kann.

Hier ist also auf jeden Fall die Zusammenarbeit mit einem externen Rechner oder eine zweite Speichererweiterungskarte anzuraten, um wiederholtes Eintippen seines Speicherinhaltes zu vermeiden.

Insgesamt läßt sich eigentlich nur Positives über das Assembler-Modul berichten. Eine Ausnahme bildet der Preis, der mir mit 500.- DM doch etwas hoch gegriffen erscheint. Vielleicht sollte die einfache betriebswirtschaftliche Weisheit (mehr Umsatz bringt auch mehr Gewinn) ein wenig zum Nachdenken anregen.

Da der Assembler selbst so vielfältig einsetzbar ist und eine völlig neue Welt im Rechner erschließt, werde ich eine eigene Reihe im PRISMA vorstellen. Der vorläufige Arbeitstitel ist Maschine&Sprache, TOOL. Hier werde ich Anwendungsbeispiele entwickeln, das Betriebssystem mit so ziemlich allen ROM-Adressen und Subroutinen vorstellen und so nebenbei ein zur Zeit fehlendes Werk für Maschinensprache für den HP48 für die Maschinensprache und RPL versuchen vorzustellen.

Hier hoffe ich auf ein reges Interesse und ein Feedback, so daß einmal der Spaß an der weiteren umfangreichen Arbeit erhalten bleibt und auch vielleicht besondere Neigungen und Sachgebiete berücksichtigt werden können.



Georg Hoppen
Hubertusring 5
DW4512 Wallenhorst

DAMPF

```
[[0,01    6,9828   0,0010001   129,20    29,34 2514,4   0,1060   8,9767 ]
 [0,02   17,513    0,0010012    67,01    73,46 2533,6   0,2607   8,7246 ]
 [0,03   24,100    0,0010027    45,67   101,00 2545,6   0,3544   8,5785 ]
 [0,04   28,983    0,0010040    34,80   121,41 2554,5   0,4225   8,4755 ]
 [0,05   32,898    0,0010052    28,19   137,77 2561,6   0,4763   8,3960 ]
 [0,06   36,183    0,0010064    23,74   151,50 2567,5   0,5209   8,3312 ]
 [0,08   41,534    0,0010084    18,10   173,86 2577,1   0,5925   8,2296 ]]
```

Druck	Temperatur	Wasser	Dampf	Wasser	Dampf	Wasser	Dampf
in bar	in °C	spez. Volumen in	m³/kg	Enthalpie in kJ/kg		Entropie in kJ/(kg·K)	

```
.PFEIFER  SUN 21.04.91  13:52:47 UHR
48SX COL?
     « 'ΣDAT' DUP
002    IF VTYPE 6 SAME
       THEN RCL
004    END 2 DISP 0
     PREDY DROP ΣPAR
006  LIST→ DUP2 DROPN
     DUP2 "Xcol: " ROT +
008  "    Ycol: " + SWAP
     + 1 DISP
010  »  Bytes : 115
     Checksum : # 1365h

.PFEIFER  SUN 21.04.91  13:52:58 UHR
48SX ΣINTER
     « COL? → s x y
002    « 1 NΣ RCLΣ ( 1 x
     ) GET RCLΣ ( NΣ x )
004  GET
         IF DUP2 ≥
006      THEN SWAP ROT 4
     ROLLD
008      END
         IF OVER s >
010  OVER s < OR
         THEN DROP2
012  DROP2 1539 DOERR
         END 4 ROLL 4
014  ROLL
         DO DUP2 + 2 /
016  IP SWAP OVER x 2
     →LIST RCLΣ SWAP GET
018  DUP 5 ROLLD
           IF s ≥
020        THEN 4
           ELSE ROT 6
022        END ROLL
     DROP2
024      UNTIL DUP2 -
     ABS 1 ∠
026      END MAX ROT ROT
     DROP2 2 - 1 MAX NΣ
028  3 - MIN DUP 3 +
         FOR j RCLΣ ( j
030  x ) GET RCLΣ ( j y
     ) GET
032      NEXT 8 ROLL 7
     ROLL 6 ROLL 5 ROLL
034  4 →ARRY 'x' STO 4
     →ARRY 0 1 1 3
036      FOR j DUP 4
     PICK 1 GET * ROT +
038  SWAP s x j GET - *
     ROT 1 4 j -
040          FOR k k DUP2
     GET LASTARG 1 + GET
042  - x k GET LASTARG j
     + GET - / PUT
044          NEXT ROT ROT
         NEXT ROT 1 GET
046    » * +
     »  Bytes : 548
048  Checksum : # 25B2h

1 XCOL   2 YCOL
'DAMPF' 'ΣDAT' STO

Xcol: 1   Ycol: 2
'DAMPF'
4:
3:
2:                  1
1:                  2

   0,043 ΣINTER
1:  30,243761
```

```
28S COL?
     « 'ΣDAT' DUP RCL
002  TYPE 6
       IF SAME
004    THEN RCL
       END 2 DISP 0 PREDY
006  DROP ΣPAR LIST→ DUP2
     DROPN DUP2 "Xcol "
008  ROT →STR +
     "    Ycol " + SWAP
010  →STR + 1 DISP
     »  Bytes : 120,5
012  Checksum : # ED25h

28S ΣINTER
     « RCLΣ EVAL SIZE 1
002  GET COL? → s n x y
       « 1 n RCLΣ ( 1 x )
004  GET RCLΣ ( n x ) GET
     DUP2 -
006      IF ≥
         THEN SWAP ROT 4
008  ROLLD
         END OVER s >
010  OVER s <
         IF OR
012      THEN DROP2 DROP2
     "Nicht in Tabelle" 1
014  DISP ABORT
         END 4 ROLL 4
016  ROLL
         DO DUP2 + 2 / IP
018  SWAP OVER x 2 →LIST
     RCLΣ SWAP GET DUP 5
020  ROLLD s
           IF ≥
022        THEN 4
           ELSE ROT 6
024        END ROLL DROP2
       UNTIL DUP2 - ABS
026  1 ∠
       END MAX ROT ROT
028  DROP2 2 - 1 MAX n 3
     - MIN DUP 3 +
030      FOR j RCLΣ ( j x
     ) GET RCLΣ ( j y )
032  GET
         NEXT 8 ROLL 7
034  ROLL 6 ROLL 5 ROLL 4
     →ARRY 'x' STO 4
036  →ARRY 0 1 1 3
         FOR j DUP 4 PICK
038  1 GET * ROT + SWAP s
     x j GET - * ROT 1 4
040  j -
           FOR k k DUP2
042  GET LAST 1 + GET - x
     k GET LAST j + GET -
044  / PUT
           NEXT ROT ROT
046      NEXT ROT 1 GET
       » * + CLMF
048  »  Bytes : 589
     Checksum : # A6F1h

.PFEIFER  SUN 21.04.91  18:34:32 UHR
48SX MINV
     « DUP IDN OVER /
002  LASTARG 3 PICK RSD
     ROT / +
004  »  Bytes : 37,5
     Checksum : # C3C0h

28S MINV
     « INV LAST OVER IDN
002  LAST ROT RSD OVER *
     +
004  »  Bytes : 35
     Checksum : # 3BEDh
```

```
.PFEIFER  SUN 21.04.91  15:17:12 UHR
48SX NEWTON
     « DUP2 SIZE EVAL →
002  x n
       « SWAP 0 CON DUP
004  n 1 PUT 1 n 1 -
         FOR j DUP 4
006  PICK 1 GET * ROT +
     OVER ARRY→ EVAL
008  ROLL LASTARG →ARRY
     ROT x j GET * - ROT
010  1 n j -
           FOR k k DUP2
012  GET LASTARG 1 + GET
     - x k GET LASTARG j
014  + GET - / PUT
           NEXT ROT ROT
016      NEXT ROT 1 GET
       » * +
018  »  Bytes : 225,5
     Checksum : # EC67h

Eine Funktion hat Null-
stellen bei -4 -1 1 5
und den Wert 20 bei x=0,
Wie lautet das Polynom ?

2:     [ -1 1 -4 5 0 ]
1:     [ 0 0 0 0 20 ]
   NEWTON
1:  [ 1 -1 -21 1 20 ]

f(x)=X^4-X^3-21X^2+X+20

.PFEIFER  SUN 21.04.91  15:40:09 UHR
48SX QM
     « ΣPAR 1 GET 1 2
002  RCLΣ SIZE 2 GET 1 +
       FOR j XCOL ΣX^2
004  NΣ / √ j
       NEXT 1 - →ARRY
006  SWAP XCOL
     »  Bytes : 81,5
008  Checksum : # 65A6h

.PFEIFER  SUN 21.04.91  15:40:18 UHR
48SX GM
     « 1 RCLΣ SIZE 2 GET
002    FOR a 1 1 NΣ
         FOR b RCLΣ b a
004  2 →LIST GET 0 MAX *
       NEXT NΣ XROOT
006    NEXT RCLΣ SIZE 2
     GET →ARRY
008  »  Bytes : 93,
     Checksum : # 9A5Eh

.PFEIFER  SUN 21.04.91  15:41:05 UHR
48SX KEY?
     « -1 WAIT IP 10 MOD
002  LASTARG / IP 1 - 6
     * +
004  "ABCDEFGHIJKLMNOPQR
     STUVWX YZ ← 789/
006  456*  123-  0. +"
     SWAP DUP SUB
008  »  Bytes : 116
     Checksum : # C4FAh

28S REPL
     « ROT 1 4 ROLL 1 -
002  SUB LAST + 4 PICK
     SIZE + OVER SIZE SUB
004  ROT SWAP + +
     »  Bytes : 60
006  Checksum : # 5CF1h
```

```
PFEIFER  SUN 21.04.91  17:59:03 UHR
48SX NLGS
    «
002   IF OVER DUP EVAL
    LASTARG SIZE →LIST
004   ≠
    THEN
006   "Variable löschen"
    DOERR
008   END 1 CF OVER { }
    SWAP DUP2 SIZE → x
010   j v f n
      « LIST→ 1
012     FOR a a 1 →LIST
    'x' APPLY 2 →LIST
014   'j' STO+ -1
        STEP { } 1 n
016     START SWAP EQ→
    - SWAP n 1
018       FOR b OVER v
    b GET a COLCT j |
020         IF DUP TYPE
    9 SAME
022         THEN { EVAL
    } +
024         END SWAP +
    -1
026       STEP SWAP j |
    { EVAL } + 'f' STO+
028     NEXT n DUP 2
    →LIST 1 →LIST + 'j'
030   STO { { "→NXT"
        « 'x' STO 0
032   CONT
        » } { "X→F"
034     « 'x' STO f
    EVAL n →ARRY
036     » } { "|| F ||"
        « f EVAL n
038   →ARRY RNRM
        » } { "DET J"
040     « j EVAL →ARRY
    DET ABS
042     » } CONJ {
    "EXIT"
044     « 1 CONT
        » } } TMENU
046     DO CLLCD x
    DEPTH 'v' STO
048       DO DUP n 1 2
    →LIST RDM 1 DISP
050       IFERR 'x' f
    EVAL n →ARRY j EVAL
052   →ARRY / STO- 1 CF 0
          THEN DEPTH
054   v - DROPN 1
            IF 1 FS?C
056         THEN
    "ERROR !
058   Neues [ x ] wählen"
    1 DISP 1 FREEZE
060           ELSE SF 0
    x DUP RNRM DUP 2
062   IFTE 1 % CON 'x'
    STO+
064         END
        END
066   UNTIL x -9
    RND ROT OVER SAME
068   ROT KEY DROPN
    LASTARG OR OR
070     END IM ABS x
    DUP RE IFTE DUP 'x'
072   STO
      UNTIL HALT
074   END 0 MENU
    »
076 »  Bytes : 868
    Checksum : # A217h

5:      'X^2+Y^2+Z^2=8'
4:    'X^2+SQ(Y-2)+Z^2=64'
3:    'X^2+Y^2+SQ(Z+1)=27'
2:          { X Y Z }
1:      [ (0;15) (-10;0)
          (10;0)]
   NLGS
1:  [ (0;15,5563491862)
      (-13;0)
      (9;0) ]

PFEIFER  SUN 21.04.91  18:03:08 UHR
48SX JRPT
    « RCLALARM LASTARG
002   DELALARM LIST→ ROLL
    .000001 + 4 ROLLD
004   TYPE 4 →LIST
    STOALARM DROP
006 »  Bytes : 53
    Checksum : # 7765h

TIME ALRM
27,011991        )DATE
16               )TIME
« JRPT KOCHEL525
"Mozart Geburtstag" »
          EXEC SET

28S ECK3
  « { ZYKL A B C WA WB
002 WC 3 } LIST→
    START 0 SWAP STO
004   NEXT STEQ 24 MENU
  HALT 23 MENU TRI 1
006   IF -
    THEN LAST 1 SAME
008 "2 Dreiecke"
  "Geht nicht" IFTE 1
010 DISP
    END
012 »  Bytes : 170
    Checksum : # FB74h

28S TRI
    « DEPTH → n
002   «
        IFERR 0 1 7
004       START WA NOT
    DUP WB AND WC
006         IF AND
            THEN -1 ACOS
008 WB - WC - 'WA' STO
          END A AND
010 DUP B AND C
            IF AND
012         THEN B SQ C
    SQ + A SQ - 2 / B /
014 C / ACOS 'WA' STO
          END B WB AND
016 DUP C WC AND OR ROT
            IF AND
018         THEN
              IF OR
020           THEN B WB
              ELSE C WC
022           END SIN A
    * OVER / LAST < ROT
024 A < AND SWAP ASIN
    'WA' STO DUP
026         END DROP WA
    A NOT AND DUP B WB
028 AND
            IF AND
030         THEN WA SIN
    B * WB SIN / 'A' STO
032         END B C AND
            IF AND
034         THEN B SQ C
    SQ + 2 B * C * WA
036 COS * - √ 'A' STO
          END ZYKL
038     NEXT ZYKL ZYKL
      THEN DEPTH n -
040 DROPN -1
      ELSE RCLF 9 SCI
042 A B + WC 2 / SIN *
    →STR WA WB - 2 / COS
044 C * →STR ≠ SWAP
      IF STOF
046     THEN DROP -1
      END
048   END
    » 1 +
050 »  Bytes : 680
    Checksum : # 91ABh

PFEIFER  SUN 21.04.91  19:13:47 UHR
48SX NLIN
    « RCLΣ DUP TRN STOΣ
002 Σ- RCLΣ SWAP * RCLΣ
    ROT STOΣ DUP TRN *
004 SWAP OVER / LASTARG
    3 PICK RSD ROT / +
006 DUP 'L' STO SIZE
    LIST→ 0 SWAP ROT
008   FOR j L j GET j 1
    →LIST 'X' APPLY * +
010   NEXT
    »  Bytes : 144,5
012 Checksum : # E0C0h

28S NLIN
  « RCLΣ DUP TRN STOΣ
002 Σ- RCLΣ SWAP * RCLΣ
  ROT STOΣ DUP TRN *
004 SWAP OVER / LAST 3
  PICK RSD ROT / + DUP
006 'L' STO SIZE LIST→ 0
  SWAP ROT
008   FOR j L j GET
    "'X(" j →STR + STR→
010 * +
    NEXT
012 »  Bytes : 143
    Checksum : # 8486h

PFEIFER  SUN 21.04.91  19:15:30 UHR
48SX INDP?
    « PPAR 3
002   IFERR GET
      THEN DROP2 'X'
004   END { } +1 GET
    »  Bytes : 59,5
006 Checksum : # 5551h

INDP?          (kein PPAR)
1: 'X'

28S INDP?
    « PPAR 3
002   IFERR GET
      THEN DROP2 'X'
004   END
    »  Bytes : 47
006 Checksum : # 7781h

PFEIFER  SUN 21.04.91  19:19:39 UHR
48SX ▒
    « { ZYKL A B C α β
002 γ 3 } DUP LIST→
    START 0 SWAP STO
004   NEXT STEQ → 1
      « 30 MENU
006 "Nach Eingabe CONT"
  PROMPT { } 2 7
008     FOR n 1 n GET
          IF DUP RCL
010       THEN DROP
          ELSE +
012       END
        NEXT
014   » 0 MENU TRI
    CASE DUP NOT
016     THEN DROP2
  "Geht nicht" DOERR
018     END 2 SAME
        THEN 1 FREEZE
020 "2 Dreiecke" 1 DISP
      END
022   END 1
    DO GETI RCL
024 LASTARG →TAG ROT
  ROT
026   UNTIL -64 FS?
    END DROP2
028 »  Bytes : 309,5
  Checksum : # D358h

PFEIFER  SUN 21.04.91  19:20:19 UHR
48SX TRI
  « DEPTH → n
002   «
        IFERR 0 1 9
004       START
            CASE α NOT
006   β AND γ AND
              THEN -1
008   ACOS β - γ -
            END A NOT
010   α OR
            THEN α
012         END B C
    AND
014         THEN B SQ
    C SQ + A SQ - 2 B *
016 C * / ACOS
            END B β
018 AND DUP C γ AND OR
            THEN
020           IF OR
              THEN B
022 β
              ELSE C
024 γ
            END SIN
026 A * OVER / LASTARG
  < ROT A < AND SWAP
028 ASIN
            END DROP
030 α
          END 'α' STO
032       CASE 'A' α
    NOT A OR
034         THEN DROP
            END B β
036 AND
          THEN α
038 SIN B * β SIN /
    SWAP STO
040       END B C
    AND
042       THEN B SQ
    C SQ + 2 B * C * α
044 COS * - √ SWAP STO
          END DROP
046     END ZYKL
      NEXT
048   THEN DEPTH n -
  DROPN -1
050   ELSE
        IF A B + γ 2
052 / SIN * -9 RND α β
    - 2 / COS C * -9
054 RND ≠ LASTARG AND
    NOT OR
056       THEN DROP -1
        END
058   END
    » 1 +
060 »  Bytes : 635
    Checksum : # 6C44h

28S UP
    « PATH DUP SIZE 1 -
002 1 MAX GET EVAL
    »  Bytes : 32,5
004 Checksum : # 5AA9h

28S TVARS
    « → t
002   « VARS LIST→ { } 1
        IF ROT
004     THEN LAST
          START OVER RCL
006 TYPE t
            IF SAME
008         THEN +
            ELSE SWAP
010 DROP
          END
012       NEXT
        ELSE DROP
014     END
    »
016 »  Bytes : 99
    Checksum : # 7C01h

28S NTAN
    « → x n a
002   « n 2
        IF %
004     THEN a x * TAN n
    1 - ^ n 1 - a * / x
006 n 2 - a NTAN -
        ELSE n 1
008       IF SAME
          THEN a x * COS
010 ABS LN a / NEG
        ELSE x
012     END
      END
014   »
    »  Bytes : 191,5
016 Checksum : # 561Ch

RAD  0,33  4  2  NTAN
1: 1,98605651805E-2

PFEIFER  SUN 21.04.91  16:09:29 UHR
48SX TRAPEZ
    « DUP ARRY→ EVAL
002   ROLL LASTARG →ARRY
    - SWAP DUP ARRY→
004   EVAL ROLL LASTARG
    →ARRY + DOT 2 /
006 »  Bytes : 55
    Checksum : # 5E44h

Beispiel im Text:
1:      [ 1 2 2 -1 0 ]
2:        [ 1 3 5 4 1 ]
   TRAPEZ
1: -8

PFEIFER  SUN 21.04.91  20:15:11 UHR
48SX GLN
    « -1 → v n
002   «
        WHILE DUP 0
004   SAME NOT
        REPEAT v DUP2 {
006   0 } + 1 'n' INCR !
      / COLCT ROT ROT a
008     END n 1 +
      »
010   WHILE DUP2 SWAP 0
    SAME AND
012   REPEAT SWAP NOT -
      END 1 + DUP 1 +
014 ROLLD 2 LASTARG
      FOR n n ROLL
016   NEXT →LIST
    »  Bytes : 179
018 Checksum : # F221h

28S POLYFIT
  « 1 + NΣ MIN DUP 1 2
002 →LIST DUP 0 CON DUP
  DUP TRN * ROT COL? →
004 b x y
    « 1 NΣ
006   FOR a RCLΣ a x 2
  →LIST GET 4 PICK 1
008 OVER
      START 1 - ^
010 LAST
      NEXT DROP2 b
012 →ARRY ROT OVER RCLΣ
    a y 2 →LIST GET * +
014 ROT ROT DUP TRN * +
      NEXT
016   » SWAP OVER / LAST
    3 PICK RSD ROT / +
018 SWAP 1 →LIST RDM DUP
    'L' STO ARRY→ LIST→
020 - INDP? 0 ROT 0
    FOR n n 3 + ROLL 3
022 PICK n ^ * + -1
    STEP CLMF SWAP
024 DROP
    »  Bytes : 312
026 Checksum : # 60Fh

28S OBJ→
    « { C→R STR→ ARRY→
002 ARRY→ LIST→ } OVER
    TYPE GET EVAL
004 »  Bytes : 37,5
    Checksum : # DB63h
```

Ralf Pfeifer (116) Rubensstr. 5 5000 Köln 50

Byte Loader in M-Code

von Klaus Hupperts

Wie die Überschrift schon sagt, handelt es sich bei diesem Programm um die MCODE-Version des Byteladers von Bill Wickes und Keith Jarret.
Der eigentliche Kern dieses Programms stammt von Stephane Barizien aus dem PPC-T. Nimmt man die Zeilen 93 bis 97, fügt davor A<>C ALL; R=0 ein und hängt am Schluß R=3; NCGO 2337 (PUTPC) an, so hat man schon eine lauffähige Byteladerversion, welche ganze 11 Bytes lang ist. Die LB-Version aus dem Wickes hat dagegen stolze 441 Bytes.

Bei meinem Bytlader ist es mir darum gegangen, die Programmierung per Anzeige mitverfolgen zu können. Der Aufwand ist zugegebenermaßen beträchtlich gegenüber der 11-Byte-Vesion des französischen Clubmitgieds des PPC-T, dafür kann man aber einige Dinge, die im Wickes doch recht mühsam beschrieben sind, nun über die Anzeige mitverfolgen.
Das Programm ist außerdem auch etwas sicherer als die synthetische Version. Ein MEMORY LOST ist praktisch ausgeschlossen, wenn man nicht gerade ein Text-15-Byte ohne folgenden Text vor das .END. plaziert und dann ein PACK ausführt oder mittels waghalsiger END-Programmierung Kamikaze-Pilot spielt.
Das schlimmste, was im Normalfall passieren kann, ist, daß ein falscher Programmschritt angezeigt wird, da im LASTX-Register Werte für die Anzeigesteuerung abgelegt werden. Wer ganz sicher gehen will, der löscht dieses Register vor der Benutzung von BYTE.

Bedienung:

Als erstes positioniert man den Rechner auf die Programmzeile, vor die man einfügen möchte. Dann wird BYTE aufgerufen und die drei Prompts mit einer dezimalen Byteeingabe beantwortet. Die neue eventuell noch unvollständige Zeile steht nun in der Anzeige, egal, ob man sich im PRGM- oder RUN-Modus befindet.

Programmgliederung:

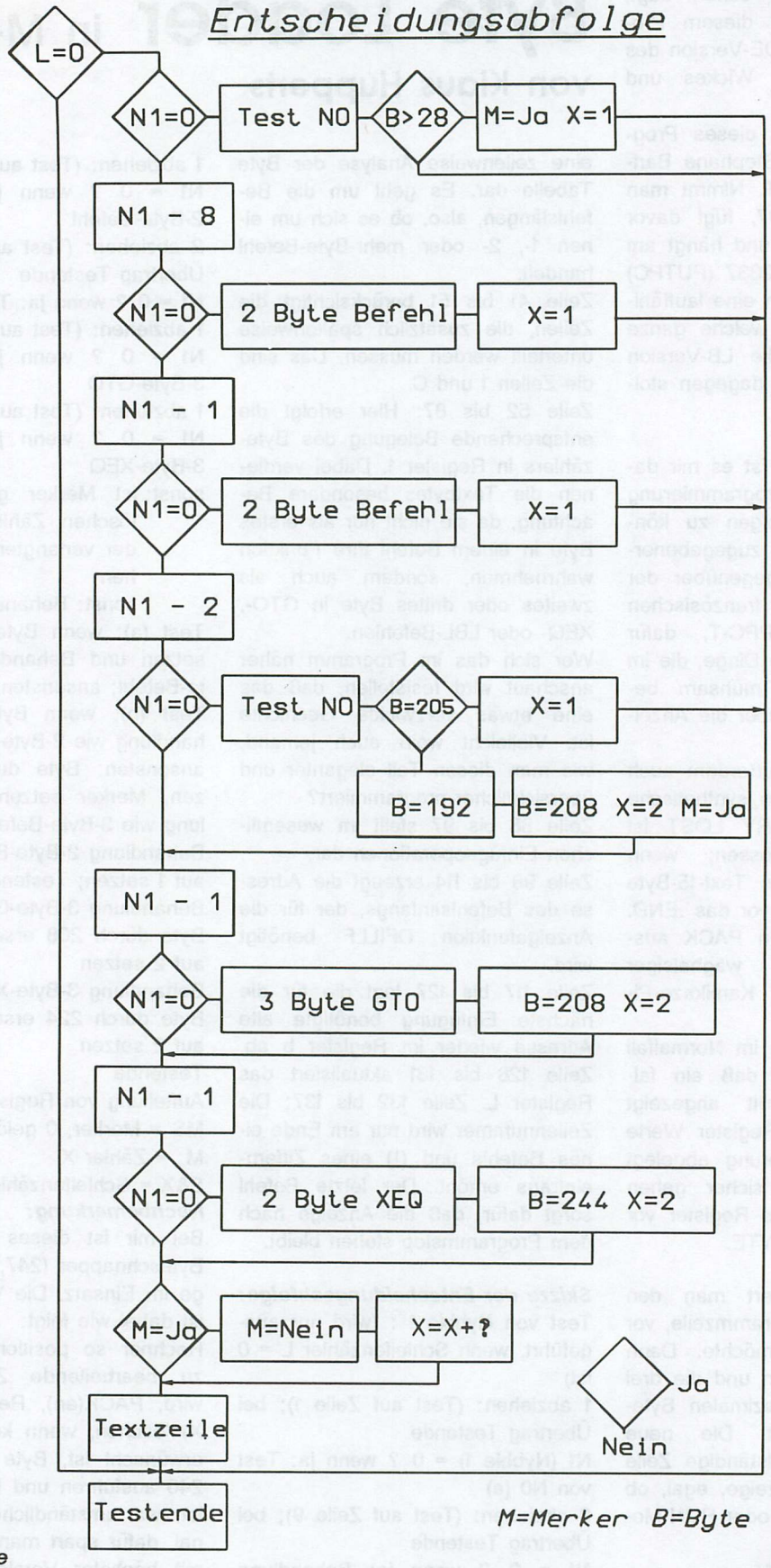
Der erste Block bis Zeile 40 stellt eine zeilenweise Analyse der Byte Tabelle dar. Es geht um die Befehlslängen, also, ob es sich um einen 1-, 2- oder mehr-Byte-Befehl handelt.
Zeile 41 bis 51 berücksichtigt die Zeilen, die zusätzlich spaltenweise unterteilt werden müssen. Das sind die Zeilen 1 und C.
Zeile 52 bis 87: Hier erfolgt die entsprechende Belegung des Bytezählers in Register 1. Dabei verdienen die Textbytes besondere Beachtung, da sie nicht nur als erstes Byte in einem Befehl ihre Funktion wahrnehmen, sondern auch als zweites oder drittes Byte in GTO-, XEQ- oder LBL-Befehlen.
Wer sich das im Programm näher anschaut wird feststellen, daß das eine etwas verzwickte Gechichte ist. Vielleicht weiß auch jemand, wie man diesen Teil eleganter und übersichtlicher programmiert?
Zeile 88 bis 97 stellt im wesentlichen Einfügeoperationen dar.
Zeile 98 bis 114 erzeugt die Adresse des Befehlsanfangs, der für die Anzeigefunktion DFILLF benötigt wird.
Zeile 117 bis 127 legt die für die nächste Einfügung benötigte alte Adresse wieder im Register b ab.
Zeile 128 bis 131 aktualisiert das Register L. Zeile 132 bis 137: Die Zeilennummer wird nur am Ende eines Befehls und (!) eines Ziffereintrags erhöht. Der letzte Befehl sorgt dafür, daß die Anzeige nach dem Programmstop stehen bleibt.

Skizze der Entscheidungsabfolge:

Test von Nybble 1 : (wird nur ausgeführt, wenn Schleifenzähler L = 0 ist)
1 abziehen: (Test auf Zeile 1); bei Übertrag Testende
N1 (Nybble 1) = 0 ? wenn ja: Test von N0 (a)
8 abziehen: (Test auf Zeile 9); bei Übertrag Testende
N1 = 0 ? wenn ja: Behandlung 2-Byte-Befehl
1 abziehen: (Test auf Zeile A)
N1 = 0 ? wenn ja: Behandlung 2-Byte-Befehl
2 abziehen: (Test auf Zeile C); bei Übertrag Testende
N1 = 0 ? wenn ja: Test von N0 (b)
1 abziehen: (Test auf Zeile D)
N1 = 0 ? wenn ja: Behandlung 3-Byte-GTO
1 abziehen: (Test auf Zeile E)
N1 = 0 ? wenn ja: Behandlung 3-Byte-XEQ
sonst: ist Merker gesetzt, Merker löschen, Zähler X um Anzahl der verlangten Zeichen erhöhen
sonst: Behandlung Textzeile
Test (a): wenn Byte > 28: Merker setzen und Behandlung wie 2-Byte-Befehl; ansonsten Testende
Test (b): wenn Byte > 205: Behandlung wie 2-Byte-Befehl
ansonsten: Byte durch 192 ersetzen, Merker setzen und Behandlung wie 3-Byte-Befehl
Behandlung 2-Byte-Befehl: Zähler X auf 1 setzen; Testende
Behandlung 3-Byte-GTO:
Byte durch 208 ersetzen, Zähler X auf 2 setzen
Behandlung 3-Byte-XEQ:
Byte durch 224 ersetzen, Zähler X auf 2 setzen
Testende
Aufteilung von Register 4(L) :
MS = Merker, 0 gelöscht, 1 gesetzt
M = Zähler X
S&X = Schleifenzähler L

Nachbemerkung:

Bei mir ist dieses Programm als Byteschnapper (247,240) schon lange im Einsatz. Die Vorgehensweise ist dabei wie folgt:
Rechner so positionieren, daß die zu bearbeitende Zeile angezeigt wird, PACK(en), Register L (LAST X) löschen, wenn korrekte Anzeige erwünscht ist, Byte 247 und Byte 240 ausführen und PACK(en). Dies ist zwar umständlicher als das Orginal, dafür spart man sich aber eine mit höchster Vorsicht zu generierende Tastaturzuweisung.



Programm siehe nächste Seite

Klaus Hupperts
Nievelsteiner-Str. 30
4050 Mönchengladbach

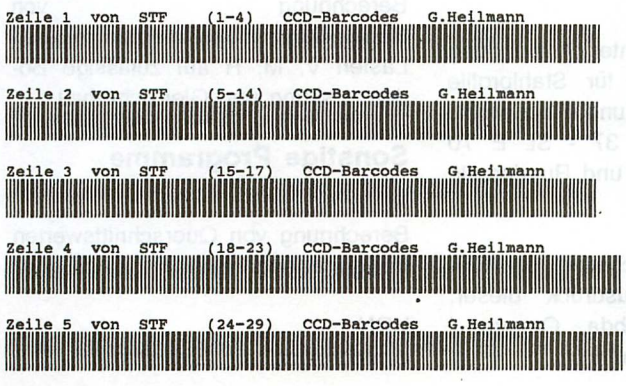
```
0001  B43B  085  "E"
0002  B34C  014  "T"
0003  B34D  109  "Y"          Erwartet als Eingabe eine dreistellige
0004  B34E  102  "B"          Dezimalzahl
0005  B34F  000  NOP          Funktion ist nicht programmierbar
0006  B350  086  B=A S&X      Byte nach CPU-B retten
0007  B351  138  READ 4(L)    hier stehen Schleifenzähler
0008  B352  2FA  ?C#0 M       BST-Zähler
0009  B353  1FF  JC B392 +3F  wenn ein anderes als das erste Byte eines
                               Befehls eingegeben wurde, dann Sprung
0010  B354  31C  R= 1         Pointer auf den 'Zeilenindex' der Bytetabelle
                               setzen
0011  B355  1A2  A=A-1  R     stammt das Byte aus der ersten Reihe, also ein
0012  B356  1E7  JC B392 +3C   Einbytebefehl, dann Sprung
0013  B357  342  ?A#0  R      stammt das Byte aus der zweiten Reihe
0014  B358  0D8  JNC B373 +1B wenn ja, dann Test auf die letzten drei Spalten
0015  B359  210  LD>R 8        alle Zeilen bis einschließlich Zeile 8
0016  B35A  3DC  R=R+1         überspringen
0017  B35B  1C2  A=A-C  R     stammt das Byte aus eine der übersprungenen
0018  B35C  1B7  JC B392 +36   Zeilen, damm Sprung
0019  B35D  342  ?A#0  R      stammt das Byte aus Zeile 9, dann Sprung
0020  B35E  133  JNC B384 +26 zur Behandlung von 2-Byte-Befehlen
0021  B35F  1A2  A=A-1  R     stammt das Byte aus Zeile A, dann Sprung
0022  B360  342  ?A#0  R      zur Behandlung von 2-Byte-Befehlen
0023  B361  11B  JNC B384 +23
0024  B362  090  LD>R 2       Vorbereitung Zeile B überspringen
0025  B363  3DC  R=R+1
0026  B364  1C2  A=A-C  R     stammt das Byte aus Zeile B, also ein
0027  B365  16F  JC B392 +2D   Einbytebefehl, dann Sprung
0028  B366  342  ?A#0  R      stammt das Byte aus Zeile C, dann Test
0029  B367  09B  JNC B37A +13 auf globale Marke
0030  B368  1A2  A=A-1  R     stammt das Byte aus Zeile D, dann Sprung
0031  B369  342  ?A#0  R      zur Behandlung von 3-Byte-Befehlen:
0032  B36A  0E3  JNC B386 +1C 3-Byte-Gto's
0033  B36B  1A2  A=A-1  R     stammt das Byte aus Zeile E, dann Sprung
0034  B36C  342  ?A#0  R      zur Behandlung von 3-Byte-Befehlen:
0035  B36D  10B  JNC B38E +21 3-Byte-XEQ's
0036  B36E  1A2  A=A-1  R     handelt es sich um ein Textbyte, dann
0037  B36F  138  READ 4(L)    wird bei gesetztem Merker zur normalen
0038  B370  2FE  ?C#0 MS      Behandlung einer Textzeile verzweigt
0039  B371  10F  JC B392 +21  wenn das nicht der Fall ist, wird
0040  B372  173  JNC B3A0 +2E der Status des Merkers aktualisiert
0041  B373  130  LDI S&X      Beginn des Tests auf die letzten drei
0042  B374  00D               Spalten von Zeile 1
0043  B375  306  ?A<C S&X     wenn Byte <= 28,
0044  B376  0E7  JC B392 +1C  dann Sprung
0045  B377  138  READ 4(L)    ist das Byte > 28, dann wird der Merker
0046  B378  23E  C=C+1 MS     gesetzt, und das GTO, XEQ oder W wie ein
0047  B379  093  JNC B38B +12 2-Byte-Befehl behandelt
0048  B37A  130  LDI S&X      Beginn des Tests auf globale Marken
0049  B37B  00E
0050  B37C  306  ?A<C S&X     wenn Byte >= 206, dann erfolgt Behandlung
0051  B37D  03B  JNC B384 +07 wie 2-Byte-Befehl
0052  B37E  130  LDI S&X      repräsentiert das Byte eine globale Marke,
0053  B37F  0C0               wird es kathegorisch durch Byte 192
0054  B380  0E6  C<>B S&X     ersetzt
```

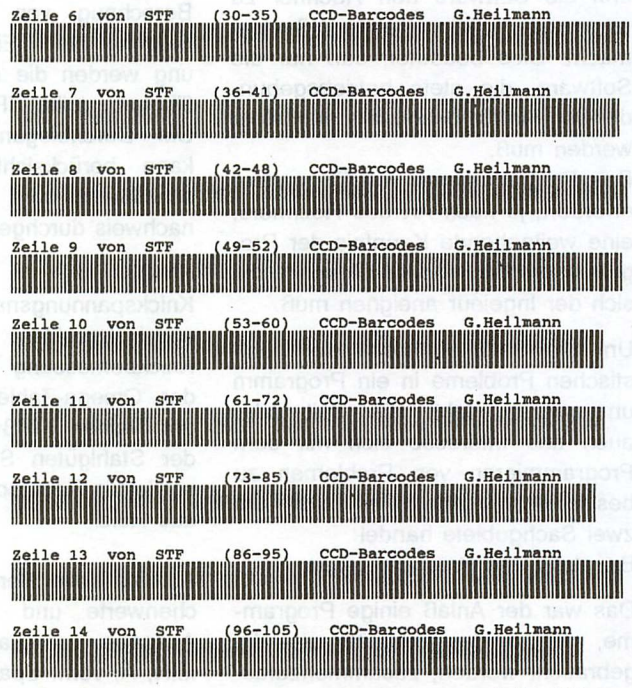
```
0055  B381  138   READ 4(L)          dann wird der Merker gesetzt, um das evtl.
0056  B382  23E   C=C+1 MS           folgende Textbyte an dritter Position richtig
0057  B383  -Ø3B  JNC B38A +Ø7       zu behandeln; Sprung zur Behandl. 3-Byte-Befehl
0058  B384  138-  READ 4(L)          Behandlung von 2-Byte-Befehlen
0059  B385  Ø33  ┌JNC B38B +Ø6       Sprung um Bytezähler auf 1 zu setzen
0060  B386  13Ø  LDI S&X             Beginn Behandlung 3-Byte-GTO, jedes Byte aus
0061  B387  ØDØ                     Zeile D wird kathegorisch durch Byte 2Ø8
0062  B388  ØE6   C<>B S&X           ersetzt
0063  B389  138-  READ 4(L)          dann wird der Bytezähler auf 2 gesetzt
0064  B38A  226   C=C+1 S&X
0065  B38B  226   C=C+1 S&X
0066  B38C  128   WRIT 4(L)
0067  B38D  Ø2B  -JNC B392 +Ø5       dann Sprung
0068  B38E  13Ø  LDI S&X             Beginn Behandlung 3-Byte-XEQ, jedes Bytes aus
0069  B38F  ØEØ                     Zeile E wird kathegorisch durch Byte 224
0070  B39Ø  ØE6   C<>B S&X           ersetzt
0071  B391  3C3-  JNC B389 -Ø8       dann wird der Bytezähler auf 2 gesetzt
0072  B392  138   READ 4(L)
0073  B393  2FE   ?C≠Ø MS            ist der Merker nicht gesetzt, wird der nächste
0074  B394  Ø73  -JNC B3A2 +ØE       Test ausgelassen
0075  B395  Ø66   A<>B S&X           ansonsten wird das vollständige Byte wieder nach
0076  B396  Ø86   B=A S&X            CPU-A kopiert,
0077  B397  13Ø  LDI S&X
0078  B398  ØFØ                     240 vom vorausgesetzten Textbyte abgezogen, um
0079  B399  1C6   A=A-C S&X          die Zeichenzahl zu ermitteln, und falls es sich
0080  B39A  Ø47  -JC B3A2 +Ø8        doch nicht um ein Textbyte handelte, Test verlassen
0081  B39B  138   READ 4(L)
0082  B39C  ØA6-  A<>C S&X           neue Bytezahl im Steuerregister ablegen
0083  B39D  Ø5E   C≠Ø MS             und Merker löschen
0084  B39E  128   WRIT 4(L)          aus dem letzten Test kommend, darf der BST-Zähler
0085  B39F  Ø1B  -JNC B3A2 +Ø3       nicht mehr abfragt werden
0086  B3AØ  2FA   ?C≠Ø M             war der Merker nicht gesetzt und das Byte ein
0087  B3A1  3DB-  JNC B39C -Ø5       Textbyte, dann erfolgt ein Sprung zur Textbyte-
                                     behandlung, wenn es das erste Byte im Befehl ist
0088  B3A2  138  -READ 4(L)          hier werden BST- und Bytezähler verschoben,
0089  B3A3  Ø3C   RCR 3              sodaß der BST-Zähler im Exponenten liegt
0090  B3A4  128   WRIT 4(L)
0091  B3A5  ØE6   C<>B S&X           Anfang der eigentlichen Einfügungsroutine
0092  B3A6  39C   R= Ø               das Byte wird nach CPU-C kopiert und der Pointer
                                     auf das nullte Nybble gesetzt, um es als
0093  B3A7  Ø58   G=C  R;+          Anfangsbedingung für die Betriebssystemroutinen
0094  B3A8  141                     nach CPU-G zu speichern
0095  B3A9  ØA4   ?NCXQ 295Ø (GETPC) holt den Programmzeiger und dient als
                                     Vorbereitung für die Programmierung
0096  B3AA  399                     das in CPU-G stehende Byte wird vor der Zeile
0097  B3AB  ØA4   ?NCXQ 29E6 (INSBYT) eingefügt, die im Prgm-Modus zu sehen wäre
0098  B3AC  138   READ 4(L)          der Exponent vom Statusreg. L dient als
0099  B3AD  ØA6   A<>C S&X           Schleifenzähler für folgendes BST
0100  B3AE  1A6   A=A-1 S&X          abweisende Schleife, deshalb Dekrement vorher
0101  B3AF  Ø77  -JC B3BD +ØE        Sprung, wenn Exponent von L Null war
0102  B3BØ  Ø1C-  R= 3               Position der Bytezahl im Programmzähler
0103  B3B1  19Ø  LD R 6              Vergleichsziffer, falls Übertrag nötig wird
0104  B3B2  3DC   R=R+1              Pointer wieder auf Byteposition
0105  B3B3  ØA2   A<>C  R            und Vergleichsziffer nach CPU-A
0106  B3B4  338   READ 12(b)         Programmzähler nach CPU-C
0107  B3B5  222   C=C+1  R           Bytezahl erhöhen
```

```
0108  B3B6  3Ø2   ?A<C  R          Test ob Übertrag nötig ist
0109  B3B7  Ø1B  -JNC B3BA +Ø3     wenn nicht: Sprung
0110  B3B8  Ø42   C=Ø  R          sonst: Bytezahl auf Null setzen
0111  B3B9  226   C=C+1 S&X        und Registerzahl um eins erhöhen
0112  B3BA  328  -WRIT 12(b)       Programmzeiger wieder zurückschreiben
0113  B3BB  1A6   A=A-1 S&X        BST-Zähler  dekrementieren
0114  B3BC  3A3-  JNC B3BØ -ØC     wenn kein Übertrag, dann Sprung zum Schleifenanf.
0115  B3BD  18D  ──┐               Anzeige der laut Programmzeiger aktuellen Zeile
0116  B3BE  Ø14  ?NCXQ Ø563        (DFILLF)
0117  B3BF  138   READ 4(L)        dient wieder als Schleifenzähler, diesmal
0118  B3CØ  ØAE   A<>C ALL         für SST
0119  B3C1  338   READ 12(b)       Programmzeiger
0120  B3C2  Ø1C ┌─>R= 3            Position der Bytezahl im Prgm-Zeiger
0121  B3C3  262 │  C=C-1  R       Bytezahl erniedrigen
0122  B3C4  Ø1B │ -JNC B3C7 +Ø3   wenn kein Übertrag nötig, dann Sprung
0123  B3C5  19Ø │  LD R 6         sonst: 6 einfügen
0124  B3C6  266 │  C=C-1 S&X      und Registerzahl um eins erniedrigen
0125  B3C7  1A6 -┘ -A=A-1 S&X     SST-Zähler dekrementieren
0126  B3C8  3D3  -JNC B3C2 -Ø6    wenn kein Übertrag, dann Sprung zum Schleifenanf.
0127  B3C9  328   WRIT 12(b)      Programmzeiger auf alte Position vor
                                  Funktionsaufruf stellen
0128  B3CA  138   READ 4(L)       SST-BST-Zähler wieder nach M und
0129  B3CB  1BC   RCR 11          Bytezähler nach S&X; dann SST-BST-Zähler
0130  B3CC  23A   C=C+1 M         inkrementieren, damit, wenn nötig, ein Mehr-
0131  B3CD  266   C=C-1 S&X       bytebefehl immer von Anfang an angezeigt wird
                                  und die Bytezahl dekrementieren
0132  B3CE  Ø2B  -JNC B3D3 +Ø5    wird dabei ein Übertrag erzeugt, wird die
0133  B3CF  3F8   READ 15(e)      Zeilennummer um eins erhöht, und Status-
0134  B3DØ  226   C=C+1 S&X       register L gelöscht
0135  B3D1  3E8   WRIT 15(e)
0136  B3D2  Ø4E   C=Ø ALL         ansonsten werden beide Zähler auf dem neuen
0137  B3D3  128  -WRIT 4(L)       Stand nach L zurückgespeichert
0138  B3D4  2Ø1                   zum Schluß wird das Messageflag gesetzt,
0139  B3D5  ØØE   ?NCGO Ø38Ø      sodaß die Meldung nach Programmstop stehen bleibt
```

SMINIT 2. Teil

STF





Fortsetzung Seite 88

Statik Modul HE 1.0

von Josef Herten

Modulvorgeschichte

Obwohl der Taschenrechner HP-41 schon lange auf dem Markt ist, hat er immer noch gegenüber dem Groß-Computern durch seine Kompaktheit und seinem Permanentspeicher bedeutende Vorzüge. Hinzu kommt, daß dieser Klein-Computer mit vier Klein-Batterien von fremden Spannungsquellen unabhängig ist und sofort und überall betrieben werden kann.
Durch die Benutzung des CCD-Moduls, das speziell für den HP-41 entwickelt wurde, wird das Betriebssystem um etwa 100 Funktionen erweitert. Dazu kommt eine Vielzahl von Betriebssystemerweiterungen, sowie auch der Ausdruck von Kleinbuchstaben. Infolge dieser Verbesserungen ist ein sehr leistungsfähiges, preisgünstiges und programmierbares Tisch- und Taschencomputersystem entstanden, das den konstruktiv tätigen Ingenieur von der reinen Rechenarbeit am Schreibtisch und auch auf der Baustelle entlastet.

Wie heute auch bekannt ist, daß erst die Software den Rechner zu einem gut funktionierendem System macht. Dies bedeutet, daß nur die Software, die stets betriebsgebunden ist, hierfür besonders ausgelegt werden muß.
Das Programmieren von Computern erfordert, je nach Art des Rechners, eine weitgehende Kenntnis der Programmiersprache oder Eigenart, die sich der Ingeieur aneignen muß.

Um die Sachkenntnisse der statistischen Probleme in ein Programm umzusetzen, fehlt die Zeit oder auch das Interesse sich mit dem Programmieren von Problemen zu beschäftigen, zumal es sich um zwei Sachgebiete handel:
Baustatik und Programmieren.

Das war der Anlaß einige Programme, die in der Baustatik ständig gebraucht werden, zusammenzufassen und in ein Modul für den HP-41 oder HP-41CY zu konzipieren. Das Statikmodul HE 1.0 umfasst elf Programme.

Statik

EZFTR:
Ein- und Zweifeldträger mit beliebiger Belastung (Einzellast, Gleichlast, Dreieckslast und Momente) und Lagerungsbedingung. Berechnung der Auflager und Schnittkräfte sowie Ausdruck der Schnittkräfte als Plott.

MFTR:
Mehrfeldträger mit 2 bis 8 Feldern mit beliebiger Belastung und Lagerungsbedingung, auch Gelenke als Zwischenbedingung können angegeben werden.
Desweiteren Stapelbearbeitung von verschiedenen Lastfällen. Berechnung der Auflager und Schnittkräfte (Querkräfte, Momente, Biegelinie und Biegewinkel) sowie Ausdruck der Schnittkräfte als Plott.

Stahl- und Holzbau

STPRO:
Berechnug von Stahlprofilen auf Biegung. Nach Eingabe der Belastung werden die erforderlichen Profilhöhen (eihe IP-IPBv) errechnet. Die Durchbiegung eines Trägers kann berücksichtigt werden. Anschließend wird ein Spannungsnachweis durchgeführt.

KNICK:
Knickspannungsnachweis von Stahl- und Holzprofilen.
Knickbemessung unter Anwendung der Omega-Zahlen für Stahlprofile der Reihen I-IPBv und Rundprofile der Stahlgüten St. 37 - St. E 70 sowie für Rechteck und Rundprofile aus Holz.

Automatische Berechnung der Flächenwerte und Ausdruck dieser. Ausgabe von Lambda, Omega, i MIN, A, vorh. Spannung.

IP:
Statistische Werte von Stahlprofilen. Näherungsweise Berechnung der Profilwerte der Reihe I-IPBv.

TRAGL:
Berechnung von Stahlprofilen auf Knicken mit Biegung nach dem Traglastenverfahren.

Stahlbetonbau

STBT:
Bemessung von Stahlbetonträgern für einachsige Biegung mit oder ohne Druckkraft nach dem KH-Verfahren. Anwendung auf alle Beton- und Stahlgüten. Ausgabe des KH-Wertes sowie der erforderlichen Bewährung as1, as2 und von eb, es.

Grundbau

WSM:
Nachweis von Winkelstützmauern auf zulässige Bodenpressung, Standsicherheit und Gleitsicherheit bei automatischer Optimierung des Sporns.
Geneigtes, steigendes oder fallendes Gelände möglich. Ausgabe aller erforderlichen Nachweise sowie von Schnittkräften in der Stützmauer.

SGM:
Nachweis von Schwergewichtsmauern auf Bodenpressung, Standsicherheit und Gleitsicherheit. Ausgabe aller erforderlichen Nachweise.

REFU:
Berechnung von Rechteckeinzelfundamenten mit den Lasten V, M, H auf zulässige Bodenpressung und Gleitsicherheit.

Sonstige Programme

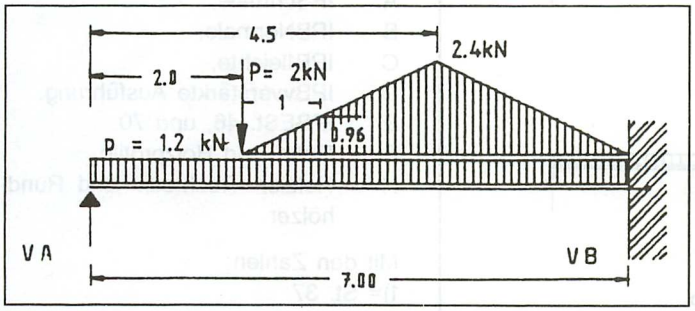
POLY:
Berechnung von Querschnittswerten polygonal begrenzter Flächen.

URN:
Umrechnung von Einheiten.

1. Programm

EZFTR:
Ein- und Zweifeldträger
Mit diesem Programm können folgende Träger berrechnet werden:

Einfeldträger I----------I
AA Träger auf zwei Stützen
AE Träger links freies Auflager, rechts eingespannt
EA Träger rechts freies Auflager, links eingespannt
EE Träger an beiden Seiten eingespannt
EF Träger mit Kragarm rechts
FE Träger mit Kragarm links.



Zweifeldträger I----------I-----I
Der Zweifeldträger, der ebenfalls im Programm EZFTR enthalten ist, hat folgende Trägerformen:
AAA Zweifeldträger mit drei freien Auflagern
AAF Zweifeldträger bedeutet Einfeldträger mit Kragarm
EAF Zweifeldträger links eingespannt ein freies Auflager in der Mitte mit Auskragung rechts
EAE Zweifeldträger links und rechts eingespannt mit einer Mittellast
EAA Zweifeldträger links eingespannt, rechts zwei freie Auflager
AAE Zweifeldträger links zwei freie Auflager, rechts eingespannt.

Nach Aufruf des Programmes, erfolgen in Dialogform die Eingaben wie Anzahl der Lasten, Dimensionen, und die Lasteingaben, wie Momente, Gleich- Strecken- Dreiecks- und Einzellasten, entsprechend der Lastanzahl.
Intern werden die Lastvektoren berechnet, und es ertönt nach Beendigung ein akustisches Signal. Es bedeutet Eingabe des Trägersystems wie oben, sodann werden die Lagerkräfte VA, VB, für den Einfeld- oder VA, X1, VB für den Zweifeldträger ausgedruckt.

Anschließend können die Schnittkräfte an jeder beliebiger Stelle des Trägers abgefragt werden, wie z.B. Querkräfte, Momente, Biegung und Biegewinkel.

Die Grundlage für diese Berechnung sowie der nachfolgenden Mehrfeldträger erfolgt nach dem erweiterten Reduktionsverfahren und Übertragungsmatrizen für Stabwerke nach Prof. Dr. Ing. Nikola S. Dimitrov.

2. Programm

MFTR:
Mehrfeldträger
Mit diesem Programm können viele Trägerformen mit beliebigen Belastungen und Felder berechnet werden.
Bei dem Programm EZFTR tritt z.B. beim Zweifeldträger an der Mittelstütze in der Querkraft ein unbekannter Sprung Q-links und Q-rechts auf. Zur Ermittlung dieser Unbekannten werden Gleichungen aufgestellt, die als Unterprogramm je nach Trägerform aufgerufen werden.

Bei den Trägern mit mehr als einer Mittelstütze ist eine Lösung wie beim Zweifeldträger nicht mehr sinnvoll.
Hier kommt die Matrizenmethode zur Anwendung, die gegenüber dem Zweifeldträger etwas mehr Rechenzeit erfordert.

Bei diesem System, zwischen Anfang und Ende des Trägers werden Zwischenauflager angeordnet, die wieder mit großen Buchstaben gekennzeichnet sind und eingegeben werden.
So bedeutet A ein gelenkiges Auflager, G ein Gelenk, H eine Stabführung und Q ein Querkraftgelenk.

Nachdem die Trägerform gewählt ist wird im Programm die Systemmatrix, die unabhängig von der Belastung ist, aufgestellt.
Das bedeutet, daß für jeden weiteren Lastfall nur ein neuer Lastvektor bestimmt wird und mit der inversen Systemmatrix multipliziert wird. Diese Möglichkeit wird zur Ermittlung der ungünstigsten Laststellungen genutzt.

Bevor die Berechnung der Lastvektoren erfolgt erscheint im Display "Lastfälle (?)".
Ist die Anzahl mit R/S eingetastet, so erfolgt die Eingabe der Lasten für jeden Lastfall. Bei einem oder mehreren Lastfällen werden intern entsprechende Register zur Belegung bereitgestellt.

(siehe hierzu Bild nächste Seite: Trägersystem A A A)

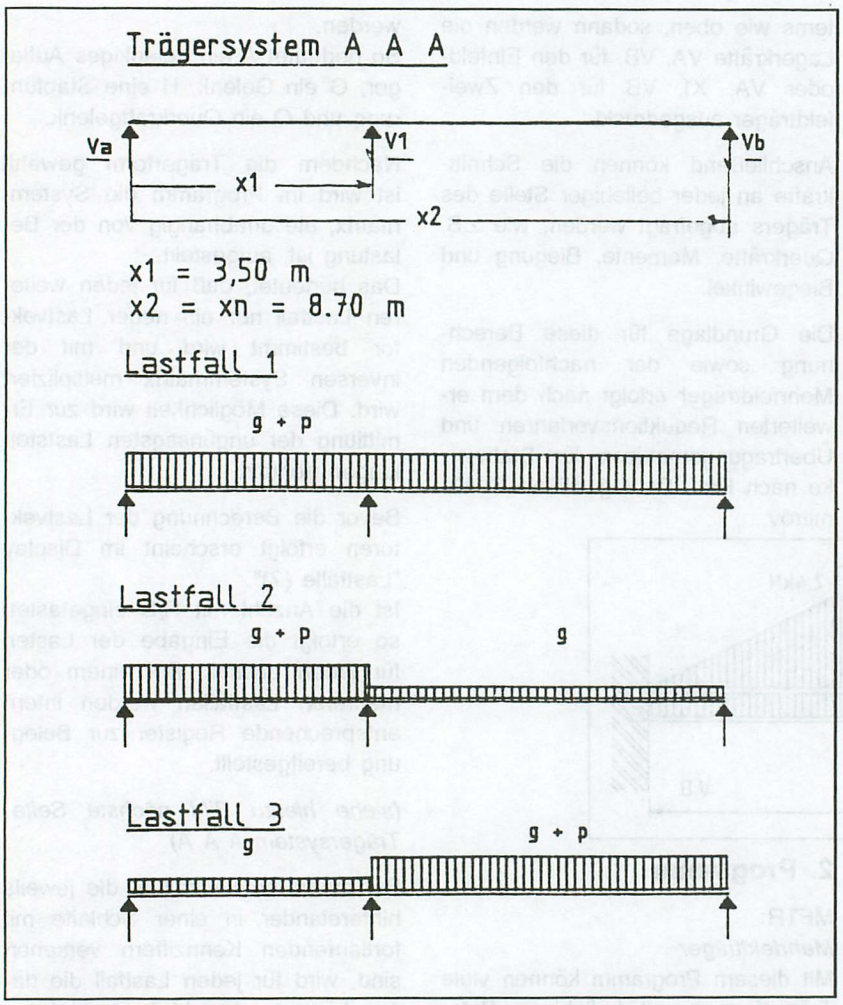
Bei mehreren Lastfällen, die jeweils hintereiander in einer Schleife mit fortlaufenden Kennziffern versehen sind, wird für jeden Lastfall die dazugehörende Anzahl Lasten eingegeben. Diese Ergebnisse werden innerhalb des Computers gespeichert und stehen zur Berechnung der Schnittkräfte zur Verfügung.

Im Display erscheint "Schnitt - Kr.".
Damit ist der Eingabevorgang abgeschlossen und die Ermittlung der Schnittkräfte, oder Plotten, kann für die eingegebenen Lastfälle erfolgen.

In einer Schleife wird jeweils der Lastfall angezeigt und die dazu gehörenden Auflagerkräfte ausgedruckt.

Anschließend erfolgt die Berechnung der Schnittkräfte. Sind alle Lastfälle erfaßt, so ist der Programmiervorgang abgeschlossen. Bei Bedarf kann das Plott-Programm aktiviert werden.

Mit diesem Programm können für alle Lastfälle die Querkraft-, Momente-, Biegelinie-, oder Biegewinkel beschrieben werden.



3. Programm

STPRO:
Stahlprofilträger
Sind z.B. aus einer statischen Berechnung die Schnittkräfte bekannt, wie Biegemoment, Querkraft und IE- fache Durchbiegung, so kann mit diesem Programm das gesuchte Stabprofil bzw. Trägertype ermittelt werden.
Nach erfolgter Eingabe erscheint im Display "Biegung J/N". Wenn bereits aus einer Berechnung eine JE-fache Biegung bekannt ist, so wird die J-Taste bestätigt und der Wert eingetastet. Ist kein Wert vorhanden, so wird die N-Taste gedrückt, um intern das Trägheits- und Widerstandsmoment zu berechnen.

Es können fünf Profiltypen wahlweise eingegeben werden:
A = IPB, B = IPB, C = IPB1,
D = IPB v, E = IPE
Durch Bestätigung der entsprechenden Taste erfolgt der Ausdruck des gewählten Profil mit Profil Höhe in cm. Anschließend werden die statistischen Werte ausgedruckt. Es können danach auch sonstige Profiltypen eingetastet werden, um sie miteinander zu vergleichen.

4. Programm

KNICK:
Knickstäbe Stahl und Holz
Das Omega Verfahren nach DIN 4114, wird fär die Berechnung von Stahl- und Holzstützen in diesem Programm angewendet.
Der besondere Wert dieses Verfahrens, liegt darin, da es die verstandesmäßig schwierigen Berechnungsgrundlagen der Knickbeanspruchung in einer für die praktischen Anwendung geeigneten, vereinfachten Form enthält.

Es wird desshalb die vorhandene Druckkraft um den Omegawert erweitert, sodaß der Querschnitt, wie unter Zugbeanspruchung bemessen werden kann.

$$\delta = w * N / F$$

Dieses Verfahren findet bei der Berechnung von verschiedenen Materialien, wie Holz und Stahl Verwendung.
Für die Ermittlung der w Zahlen, gelten zwei Bereiche, die von der Grenzschlankheit abhängig sind. Ab diesem Abschnitt aufwärts folgen die Werte einem Potenzgesetz.
Für kleinere Schlankheiten erfolgt die Berechnung durch ein Polynom.

Die Eingabe der Materialien erfolgt mit Buchstaben, so bedeutet:
A IPSchmale,
B IPBNormale,
C IPB1leichte,
D IPBvverstärkte Ausführung.
E IPBESt. 46, und 70.
F Rohr- und Holzprofile
H Holzier- Rechteck- und Rundhölzer.

Mit den Zahlen:
1)= St. 37
2)= St. 52
3)= St. E46
4)= St. E70
kann die gewünschte Stahlsorte durch Bestätigen der entsprechenden Taste eingegeben und auch angezeigt werden.

Nach dieser Materialwahl muß noch die Stabhöhe sowie die Profilhöhe des gewünschten Stahlträgers eingetastet werden, bei Holz der Querschnitt oder Durchmesser, wie auch für die Stahl-Hohlprofile der Querschnitt oder Durchmesser, und natürlich die Wandstärke.
Aus diesen Werten werden im Programm i MIN und der Querschnitt des Stabes berechnet.
Nach Eintasten der jeweilig zulässigen Spannung wird die Traglast angezeigt bzw. ausgedruckt.
Als Abschluß kann dann noch die erforderliche Last eingegeben werden, woraufhin die Stahlspannung angezeigt bzw. ausgedruckt wird.

5. Programm

TRAGL:
Traglastverfahren
Berechnung von Holz- und Stahlstützen nach dem Tragwerksverfah-

ren, nach einem Forschungsbericht der Universität Stuttgart, (Dipl. Ing. H. F. Hoyer, Modell für Biegung und Knicken), welches hier verwendet wird.

Eingaben:
Normalkraft, Moment, Knickhöhe, Lastsicherheit, Anfang- und zulässige Ausmitte, sowie das E-modul und die Fließspannung, die vom Material abhängt (Holz, Stahl).
Nachdem die vorerst geschätzten Werte, wie Querschnitt, Widerstands- und Trägheitsmoment eingegeben sind, wird intern die erforderliche Querschnittfläche berechnet und angezeigt.

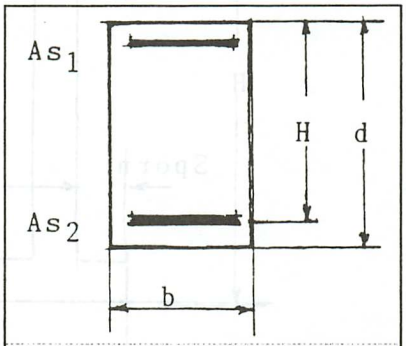
Die Berechnung erfolgt nach Gleichungen, die Kurvenscharen entwikkeln, welche zu einem Bemessungsmodell führen. Daraus lassen sich für einen Lastfall und mit einem gewählten Material drei Interaktionslinien festlegen, woraus sich drei Lastfälle unterscheiden.

1.Fall Der Querschnitt ist gleichermaßen von Spannung und Verformung ausgenutzt (kleinste Fläche).
2.Fall Die Verformung ist maßgebend.
3.Fall Die Spannung ist bestimmend.

In dem Programm wird der maßgebende Fall berücksichtigt und der benötigte Wert angezeigt.
Sodann kann ein entsprechendes Profil eingegeben werden. Diese Eingabe wird intern im Rechner überprüft und gegebenenfalls neu berechnet, um den exakten Wert nochmals anzuzeigen.
Bei der quadratischen Holzstütze wird als Fläche eine 1 eingegeben und für Wx und Ix nur noch 6 1/x bzw. 12 1/12.

6. Programm

STBT:
Stahlbetonbemessung
Bemessung von Querschnitten aus Stahlbeton für einachsige Biegung mit oder ohne Zug- oder Druckkraft erfolgt allgemein nach Tabellen z.B. im Beton Kalender oder sonstigen Nachschlagewerken.
Nach der Eingabe aller erforderlichen Werte wird der Kh-Wert berechnet und angezeigt. Nach DIN 1045 wird der zugehörige Dehnungszustand gefunden. Damit ein einheitlicher Sicherheitsfaktor von 1,75 verwendet werden kann unterscheidet man ohne Druckbewehrung, wie in den Tabellen drei Dekkungsbereiche. Die Dehnung werden beim Ausdruck der Bewehrung mit angezeigt.



Aus Stahlbetonbau Bemessung, Konstruktion, Ausführung von Dipl. Ing. Gottfried C.O. Lohmeyer.

7. Programm

WSM:
Winkelstützmauer
Die Berechnung der Erddruckwerte beruht auf Reibung, deren theoretische Grundlagen von Coulomb entwickelt wurden.
Für die analytische Ermittlung gelten die Formeln von Müller-Breslau und Krey-Ehrenberg, die hier zur Anwendung kommen.

Nach DIN 1055 soll die aus ständigen Lasten resultierende Kraft, die die Sohlfläche im Kern schneidet (b 1/3), keine klaffende Fuge erzeugen. Aus der Gesamtheit der resultierenden Kraft darf jedoch in begrenztem Umfang eu=in ein Klaffen der Sohlfuge zugelassen werden.
Diese Bedingungen werden berücksichtigt und führen zur Optimierung der Konstruktion.
Hat die Hinterfüllung eine steigende Ebene oder fallende Böschung, so ergeben sich drei unterschiedliche Lastfälle, die bei der Berechnung ebenfalls berücksichtigt werden können.
(siehe dazu Bild am Anfang der nächsten Seite)

8. Programm

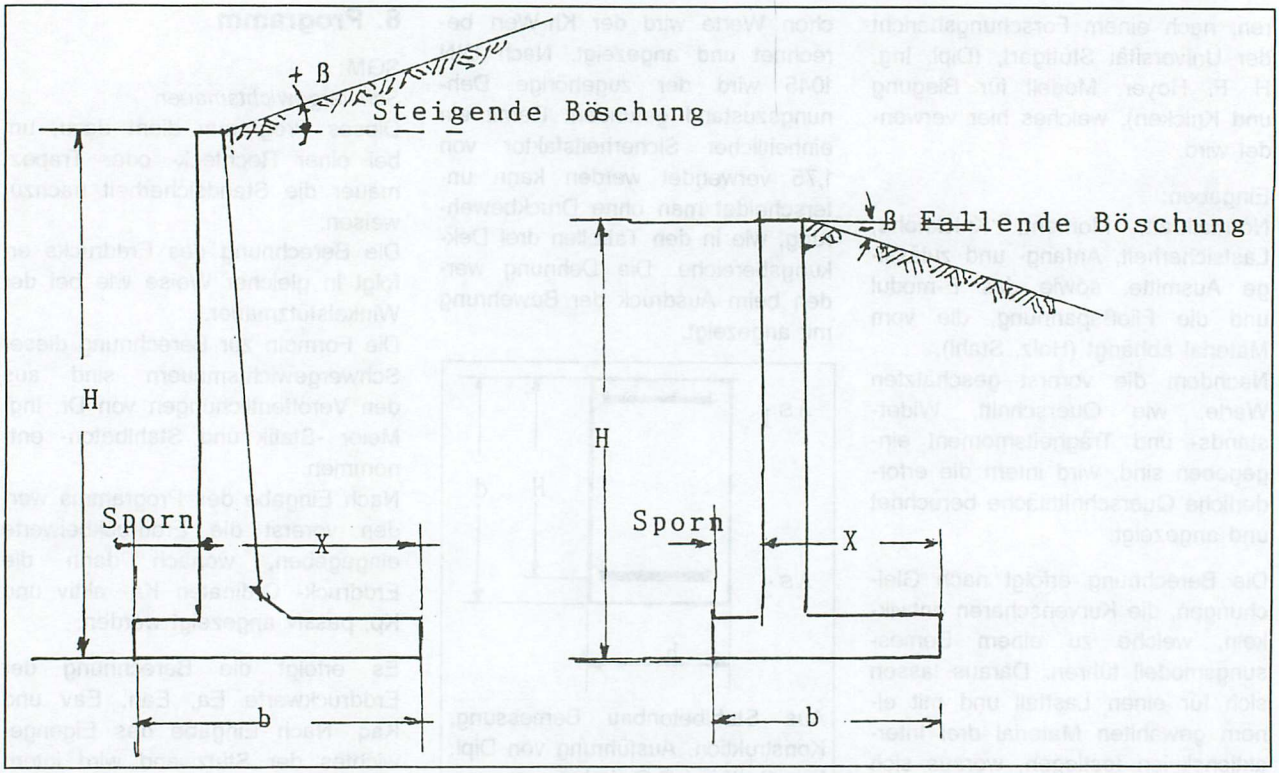
SGM:
Schwergewichtsmauer
Dieses Programm dient dazu, um bei einer Rechteck- oder Trapezmauer die Standsicherheit nachzuweisen.
Die Berechnung des Erddrucks erfolgt in gleicher Weise wie bei der Winkelstützmauer.
Die Formeln zur Berechnung dieser Schwergewichtsmauern sind aus den Veröffentlichungen von Dr. Ing. Meier -Statik und Stahlbeton- entnommen.
Nach Eingabe des Programms werden vorerst die Erddruckbeiwerte eingegeben, wonach dann die Erddruck- Ordinaten Ka- aktiv und Kp, passiv angezeigt werden.

Es erfolgt die Berechnung der Erddruckwerte Ea, Eah, Eav und Kaq. Nach Eingabe des Eigengewichtes der Stützwand wird intern eine Verzweigung eingeschaltet, wobei durch Wahl von J/N, mit J eine Rechteckmauer und mit N eine Trapezmauer berechnet wird.
Nachdem mit Berücksichtigung der DIN 1055, die Fundamentabmessung, Bodenpressung, sowie die Stand- und Gleitsicherheit berechnet sind, werden diese ausgedruckt bzw. angezeigt.

9. Programm

POLY:
Polygonal begrenzte Fläche
Nach dem Verfahren von H. Fleßner können Tragwerksquerschnitte, die polygonal begrenzt sind, in der Praxis hinreichend genau berechnet werden (siehe Beton Kalender 1965).

Handelt es sich z.B. um einen geschlossenen Körper mit n=5 Eckpunkten, so erfolgt die Eingabe ab n=0, der an einer beliebigen Stelle sein kann. Für jeden Eckpunkt gibt es zwei Ordinaten zur Begrenzung, Yi und Zi.
So werden nacheinander alle Eckpunktordinaten eingegeben, wobei berücksichtigt werden muß, daß der Eckpunkt n-5 identisch mit n-0 ist und nicht nochmals eingegeben wird.



Durch Betätigen der R/S Taste werden dann der Flächeninhalt A, die statischen Momente, Sy und Sz, sowie die Trägheitsmomente Iy und Iz und das Zentrifugalmoment Iyz ausgegeben, bzw. gedruckt.
Die Schwerpunktslagen ys und zs, sowie die Hauptträgheitsmomente I ETA und I KSI weren zusammen mit dem Achsenverdrehungswinkel ausgedruckt bzw. angezeigt

10. Programm

REFU:
Rechteckige Fundamente
Berechnung der Bodenpressungen bei rechteckigen Fundamenten nach Beton Kalender 1958 II von Dr. Ing. Fuchssteiner.
Im allgemeinen Fall greifen an einem Einzelfundament außer einer senkrechten Last F meistens noch ein Horizontalschub und Moment auf. Es ist desshalb empfehlenswert, eine exzentrische Anordnung des Fundamentes vorzunehmen.

Die Bodenpressungen und die Gleitsicherheit müssen für jeden möglichen Lastfall nachgewiesen werden. Im allgemeinen sind zwei Lastfälle bestimmend für die größten Bodenpressungen bei Fuge 1 und Fuge 2. Dieser Umstand macht den Entwurf von Fundamenten etwas schwierig, wenn man die zulässige Bodenpressungen ausnutzen will, zumal man im voraus nicht weiß, ob für den Spannungsnachweis Gl.(2) oder Gl.(4) maßgebend ist. Auch kann die zulässige Bodenpressung nach DIN 1054 erst endgültig festgelegt werden, wenn die Fundamentabmessungen bekannt sind.
Da für das Entwerfen von Einzelfundamenten keine Entwurfsformeln zur Verfügung stehen wird, wie folgt, ein Rechenvorgang gewählt.

Sind bereits die Fundamentabmessungen grob geschätzt worden, so kann durch geringe Änderungen der Abmessungen eine günstigere Ausnutzung der Bodenpressung erreicht werden.
Stellt man beim groben Berechnen fest, daß für die Bodenpressungen die Spannung nicht ganz ausgenutzt oder auch überschritten werden, so tritt an der Bodenfuge 1 die größte Pressung durch F1. H1. und M1 auf, und für Bodenfuge 2, wenn der ungünstigste Lastfall durch F2. H2. und M2. auftritt.

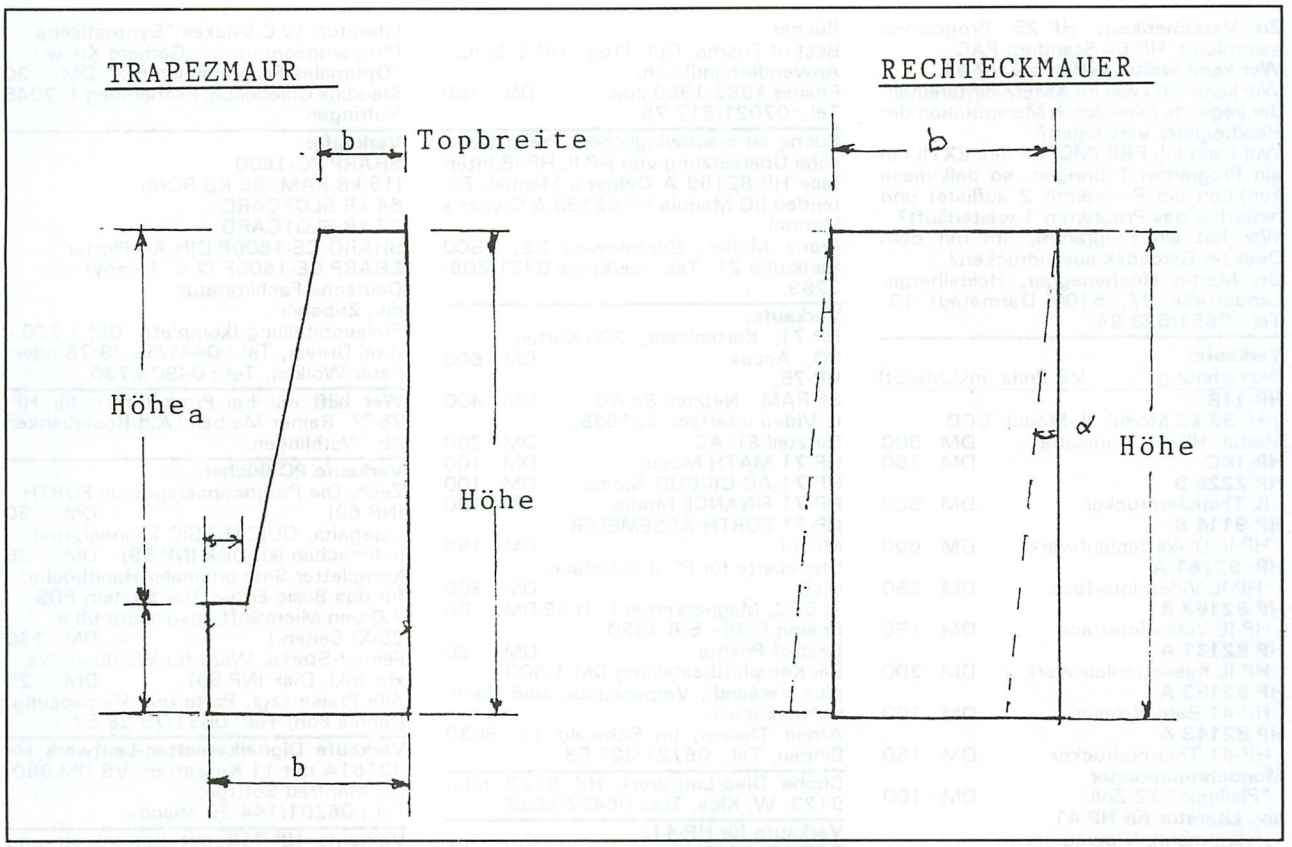
Vor allem aber haben sich die Größenordnung der Abmessungen bereits bestätigt. Mithin wäre also nur eine Verbesserung der Maße d und a erforderlich.

In diesem Programm sind drei Unterprogramme entwickelt worden, die sich je nach Lastfall und Lage der Resultierenden Inhaltlich voneiander unterscheiden. Diese Unterprogramme A - B - oder C können durch die entsprechende Taste aufgerufen und aktiviert werden.

A *Die Bodenfuge klafft nicht.*
Nach der Eingabe der Lasten und Abmessungen wird der gewünschte Fall aufgerufen. Nach kurzer Rechenzeit erfolgt der Ausdruck, und die unbekannten Bodenlänge und die Außermittigkeit a zwischen G-P wird angegeben. Sie werden bei der Berechnung der Bodenpressung verwendet (e= ≤ d/6).

B *Die Bodenfuge klafft.*
Es kann Klaffen zugelassen werden. Dieses darf jedoch höchstens bis zum Schwerpunkt der Sohlfläche reichen (d/3 ≥ e ≥ d/6).

C *Die Lage der Resultierenden ist magebend.*
In diesem Fall kann die Bodenpressung nicht ausgenutzt werden. Es werden die



Bedingungen nach DIN 1055 2. Standsicherheit von Flächengründungen berücksichtigt.
Es wird dann e1=d/3, e2=-d/3.

11. Programm

URN:
Umrechnung und Dimensionierung
Hierzu sind in der Anleitung keine weiteren Erläuterungen zu finden gewesen.

Empfohlene Literatur:
Berechnungen in der Tragwerkslehre mit programmierbaren Taschenrechnern.
Lineare Stabtragwerke von Prof. Dr. Ing. Nikola S. Dimitrov / Dipl. Ing. Hans Florian Hoyer / Dipl. Ing Bernd Raff.
Erschienen im R. Oldenbourg Verlag München Wien 1980.

Finite Last- Element Methode und ihre Programmierung von Dipl. Ing. Hans Florian Hoyer.
Eine Zusammenfassung der Operatorenstatik nach Mikusinski-Dimitrov und des erweiterten Reduktionsverfahrens wird demnächst im Werner-Verlag erscheinen.

Das soeben beschriebene Modul liegt bislang lediglich als Programmsammlung vor, da W&W bislang nicht zu einem Einbrennen derselben bereit war. Wahrscheinlich erscheint ihnen der mögliche Kundenkreis als zu klein, um die Software zu vermarkten.
Aus diesem Grund sucht Hr. Herten nach einer anderen Möglichkeit des Einbrennens. Es soll in England eine Firma ZENGRANGE geben, die dazu in der Lage sein soll.
Wer kennt diese Firma bzw. deren Adresse, damit ein Kontakt hergestellt werden kann. Bitte wendet Euch direkt an Hr. Herten, er ist für eine baldige Information sehr empfänglich!

Dipl. Ing. B. D. B.
Josef Herten
Scheibenstraße 33
DW4000 Düsseldorf 30
0211 / 49 82 080

Michael Krocker

HP48 Insights

Part I: Principles and Programming
von William C. Wickes
First Edition, February 1991
ISBN 0-9625258-3-9
erschienen bei: Larken Publications
4517 NW Queens Avenue
Corvallis, Oregon 97330, USA
Inhalt:
Es ist eine Überarbeitung des Buches HP28 Insights des gleichen Autors, es wird in zwei Teilen erscheinen.
Der 380 Seiten umfassende erste Teil konzentriert sich auf die grundlegenden Operationen des Rechners, sowie auf die Mittel, diesen zu programmieren.
Besondere Aufmerksamkeit findet die Objektspeicherung, die Display-Verwaltung und die benutzerspezifische Anpassung mit Tastenbelegungen, Menüsteuerungen und Modi.
In über 100 Programmierbeispielen werden Programmiertechniken, lokale Variable, Rekursion und die Benutzung von Listen und Arrays veranschaulicht.
Der zweite Teil von HP48 Insights wird sich mit den HP48 eigenen Funktionen wie das Zeichnen von Grafiken, symbolischen Lösungen von Gleichungen und der Einheitenverwaltung befassen. (soweit der Text des Covers)

Marco Lauer
Schloßstraße 4, DW6752 Winnweiler

Zu Verschenken: HP-25 Programmsammlung, HP-64 Standard PAC
Wer kann weiter helfen bei HP-41:
Wie kann ich zwei im XMEM hintereinander liegende Files durch Manipulation der Headregister vereinigen?
Wie kann ich PRP (MCPRP des EXTIL) in ein Programm 1 bringen, so daß diese Funktion ein Programm 2 auflistet und hinterher das Programm 1 weiterläuft?
Wer hat ein Programm, um mit dem DeskJet Barcodes auszudrucken?
Dr. Martin Hochenegger, Heidelberger Landstraße 97, 6100 Darmstadt 13, Tel.: 0651/523 94

Verkaufe:

Bezeichnung	VB-Preis (inkl.MwSt)
HP-71B	
inkl. 32 kB Modul, IL-Modul, CCD-Modul, HP-41 Translator	DM 800
HP-18C	DM 150
HP 2225 B	
- IL ThinkJetdrucker	DM 500
HP 9114 B	
- HP-IL Diskettenlaufwerk	DM 600
HP 92261 A	
- HP-IL Video Interface	DM 250
HP 82163 B	
- HP-IL Video Interface	DM 150
HP 82161 A	
- HP-IL Kassettenlaufwerk	DM 300
HP 82153 A	
- HP-41 Barcodeleser	DM 100
HP 82143 A	
- HP-41 Thermodrucker	DM 150
Monochrommonitor	
- "Phillipps" 12 Zoll	DM 100
div. Literatur für HP-41	
- Heldermann, Vieweg etc.; Liste auf Anfrage, pro Buch	DM 20

Die Mehrwertsteuer ist ausweisbar.
Carsten Antelmann, Raunheim, Tel.: 06142/218 90 (ab 19.00 Uhr)

HP-71B
mit Zubehör soll verkauft werden:

HP-71b	DM 350
2 x 4k Memory-Module	DM 50
1 x HP-71 Math-Modul	DM 80
1 x HP-71 Forth-Assembler-Modul	DM 80
1 x HP-71 IL-Modul	DM 120

Bitte telefonische oder schriftliche Anfragen an N. Maier, Am Wingert 11, 6350 Bad Nauheim (Beim wählen nicht verzweifeln ..: Tel.: 06032/357 06

SM 194 **19" Großmonitor für ATARI-Mega-ST** inkl. Grafikkarte (Sockel für 68881): DM 1950.-
MEGA-ST2 DM 950.-; beides 1 Jahr alt.
Martin Meyer, Kelkheimer Straße 20, 6232 Bad Soden, Tel.: 06196/87-4255 tags, 06196/231 50 abends.

Verkaufe

HP-71B	DM 500
Kartenleser mit Karten	DM 180
IL-Modul mit 6 Kabeln	DM 250
2 4k-Module à	DM 40
Forth-Modul	DM 140
Texteditor-Modul	DM 100
IL-Kassettenlaufwerk mit Kassetten und Netzteil	DM 400
IL-Thermodrucker	DM 300
Grabau GR7B	DM 600
Zenith-Monitor	DM 100
Star NL 10 Drucker	DM 150

Komplettpreis DM 2.500
alles mit Handbüchern und z.T. mit Verpackung.
Karsten Elles, Am Thekbusch 51, 5620 Velbert 1, Tel.: 02051/661 48.

Verkaufe für HP-41:

Magnetkartenleser mit 250 Magnetkarten	DM 220
Barcodeleser	DM 120

Bücher:
Best of Prisma, Opt. Prog., HP-IL-Sch., Anwenderhandbuch,
Prisma 1982-1990 zus. DM 190
Tel.: 07021/817 76

Suche zu erschwinglichem Preis: Deutsche Übersetzung von HP-IL/HP-IB Interface HP 82169 A Owner's Manual, Extended I/O Module HP 82183 A Owner's Manual.
Franz Müller, Buchenweg 32, 7500 Karlsruhe 21, Tel.: werktags 0721/608-3289.

Verkaufe:

HP-71, Kartenleser, 200 Karten, I/O, Accus	DM 600
HP-75, 8k RAM, Netzteil 8v AC	DM 400
IL Video Interface 82163B, Netzteil 8V AC	DM 200
HP-71 MATH Modul	DM 100
HP-71 AC CIRCUIT Modul	DM 100
HP-71 FINANCE Modul	DM 100
HP-71 FORTH ASSEMBLER Modul	DM 150
Steckkarte für PC IL Interface, Disk	DM 300
TI 58C, Magnetkarten f. TI 59	DM 50
Prisma 6/85 - 5/6 1990, Best of Prisma	DM 20

Bei Komplettbestellung DM 1.600 plus Versand-, Verpackungs- und Nachnahmekosten.
Armin Thesen, Im Schwalg 14, 6530 Bingen, Tel.: 06721/327 53

Suche Disc-Laufwerk HP 9122 oder 9123. W. Klos, Tel.: 06427/8032

Verkaufe für HP-41:

CCD Modul Version A	DM 90
IR Modul (82242 A)	DM 90
HP MATH I Module für HP-41CX	DM 30

Literatur: W.C.Wickes "Synthetische Programmierung ...", Gerhard Kruse "Optimales Programmieren .." DM 30
Klaudius Chlebosch, Eschenweg 1, 7045 Nufringen

Verkaufe:
SHARP PC-1600
(16 kB RAM, 96 KB ROM)
64 kB SLOTCARD
32 kB SLOTCARD
SHARO CE-1600P DIN-A4-Plotter
SHARP CE-1600F (2,5" Floppy)
Deutsche Fachliteratur
div. Zubehör
Preisvorstellung (komplett): DM 1.200.-
Axel Dreyer, Tel.: 0441/59 29 75 oder Peter Wolken, Tel.: 04903/730.

Wer hilft mir bei Programmen für HP-75C? Rainer Marben, A.d.Röstebänken 8b, Wathlingen.

Verkaufe PC-Bücher:
Zech, Die Programmiersprache FORTH (NP 69) DM 30
Lategahn, QUICKBASIC kennenl. und beherschen ikl. Disk (NP 59) DM 25
Kompletter Satz originaler Handbücher für das Basic Entwickler System PDS 7.0 von MicroSoft, insgesamt über 2000 Seiten ! DM 130
Felmet-Starke, Word für Windows Praxis, inkl. Disk (NP 59) DM 25
Alle Preise zzgl. Porto und Verpackung.
Dennis Föh, Tel.: 0551/79 28 57

Verkaufe Digitalkassetten-Laufwerk HP 82161A mit 11 Kassetten, VB DM 350.
Dr. Manfred Söfftge,
Tel.: 06201/144 20 abends.

Verkaufe HP-71B mit Kartenleser und Mathemodul für DM 500.-
Ralf Wanser, Artilleriestraße 25, 8000 München 19.

Impressum

Titel:
PRISMA

Herausgeber:
CCD -
Computerclub Deutschland e.V.
Postfach 11 04 11
6000 Frankfurt am Main 1

Verantwortlicher Readakteur:
Alf-Norman Tietze (ant)

Redaktion:
Michael Krocker (mik)
Martin Meyer (mm)
Dieter Wolf (dw)

Herstellung:
CCD e.V.

Manuskripte:
Manuskripte werden gerne von der Redaktion angenommen. Honorare werden in der Regel nicht gezahlt. Die Zustimmung des Verfassers zum Abdruck wird vorausgesetzt. Für alle Veröffentlichungen wird weder durch den Verein noch durch seine Mitglieder eine irgendwie geartete Garantie übernommen.

Druck und Weiterverarbeitung:
Reha Werkstatt Rödelheim
Biedenkopfer Weg 40a
6000 Frankfurt am Main 90

Anzeigenpreisliste:
Es gilt unsere Anzeigenpreisliste 3 vom Juni 1987.

Erscheinungsweise:
PRISMA erscheint jeden 2. Monat.

Auflage:
2500

Bezug:
PRISMA wird allen Mitgliedern des CCD ohne Anforderung übersandt. Ein Anspruch auf eine Mindestzahl von Ausgaben besteht nicht. Der Bezugspreis ist im Mitgliedsbeitrag enthalten.

BEST OF PRISMA
Schutzgebühr: 30,- DM

Nachsendedienst PRISMA
Schutzgebühr: 5,- DM pro Heft für Jahrgänge 1982-86
10,- DM pro Heft für Jahrgänge ab 1987

Inhaltsverzeichnis PRISMA
Schutzgebühr: 3,- DM in Briefmarken

Programmbiliothek HP-71
Die bislang in PRISMA erschienenen Programme können durch Einsenden eines geeigneten Datenträgers (3,5" Diskette, Digitalkassette oder Magnetkarte) und eines SAFU angefordert werden.

MS-DOS Inhaltsverzeichnis
Kann durch das Einsenden einer formatierten 360 kB oder 1,2 MB 5,25"-Diskette oder einer formatierten 720 kB oder 1,44 MB 3,5"-Diskette und einem SAFU angefordert werden.

ATARI Inhaltsverzeichnis
Kann durch das Einsenden einer 3,5"-Diskette + SAFU bei Werner Müller angefordert werden.

UPLE
Das UPLE-Verzeichnis mit der Kurzbeschreibung der einzelnen Programme sowie den Bezugsbedingungen kann gegen Einsenden von DM 10,- in Briefmarken angefordert werden.

Programme aus BEST OF PRISMA

a) Eine Kopie der Programme von BEST OF PRISMA auf Kassette erfordert das Beilegen einer Leerkassette und eines SAFU.
b) Für Barcodes von BEST OF PRISMA-Programmen gibt es folgendes Verfahren:
Schickt eine Liste mit den Namen und der Seitenangabe (der Barcodeseiten) an die Clubadresse, pro Barcodeseite legt bitte 40 Pf., plus 2,40 DM für das Verschicken, in Briefmarken bei. Die Liste der verfügbaren Programme ist in Heft 3/88 auf der Seite 35 abgedruckt, sie kann gegen einen SAFU angefordert werden.

Der Bezug sämtlicher Clubleistungen erfolgt über die Clubadresse, soweit dies nicht anders angegeben ist, oder telefonisch bei Dieter Wolf:

(069) 76 59 12

Die eventuell anfallenden Unkostenbeiträge können als Verrechnungsscheck beigelegt werden, Bargeld ist aus Sicherheitsgründen nicht zu empfehlen; ist dies nicht der Fall, so wird Rechnung gestellt, dies macht die Sache natürlich nicht unbedingt einfacher, bzw. schneller.
Formvorschriften für Schreiben an die Clubadresse gibt es keine; das Schreiben kann durchaus handschriftlich verfasst sein, ein normaler Sterblicher sollte es noch lesen können. Vor allem den Absender und die Mitgliedsnummer deutlich schreiben !
(SAFU = Selbst Adressierter Freiumschlag)

CLUBADRESSEN

1. Vorsitzender
Gerhard Link (3107),
Postfach 1615, 6090 Rüsselsheim,
✆ (06142) 81 51 0, Fax: (06142) 81 57 9, GEO1:G.LINK

2. Vorsitzender
Alf-Norman Tietze (1909),
Sossenheimer Mühlgasse 10,
6000 Frankfurt 80, ✆ (069) 34 62 40, GEO1:A.N TIETZE

Schatzmeister
Dieter Wolf (1734),
Pützerstraße 29, 6000 Frankfurt 90,
✆ (069) 76 59 12, GEO1:D.WOLF

1. Beisitzer
Norbert Resch (2739),
Tsingtauerstraße 69, 8000 München 82

2. Beisitzer
Werner Dworak (607),
Allewind 51, 7900 Ulm,
✆ (07304) 32 74, GEO1:W.DWORAK

MS-DOS Service / Beirat
Alexander Wolf (3303),
Pützerstraße 29, 6000 Frankfurt 90,
✆ (069) 76 59 12

ATARI Service / Beirat
Dr. Werner Müller (1865),
Schallstraße 6, 5000 Köln 41,
✆ (0221) 40 23 55, MBK1:W.MUELLER

Regionalgruppe Berlin
Jörg Warmuth (79), Wartburgstraße 17, 1000 Berlin 62

Regionalgruppe Hamburg
Alfred Czaya (2225), An der Bahn 1, 2061 Sülfeld,
✆ (040) 43 36 68 (Mo.-Do. abends)
Horst Ziegler (1361), Schüslerweg 18b, 2100 Hamburg 90,
✆ (040) 79 05 67 2

Regionalgruppe Karlsruhe / Beirat
Stefan Schwall (1695), Rappenwörthstraße 42,
7500 Karlsruhe 21, ✆ (0721) 57 67 56, GEO1:S.SCHWALL

Regionalgruppe Rheinland/Ruhrgebiet
Jochen Haas (2874), Roßstraße 27, 5000 Köln 30,
✆ (0221) 51 98 70

Regionalgruppe München / Beirat
Victor Lecoq (2246), Seumestraße 8, 8000 München 70,
✆ (089) 78 93 79

Regionalgruppe Rhein-Main
Andreas Eschmann (2289), Lahnstraße 2, 6906 Raunheim,
✆ (06142) 46 64 2

Beirat
Manfred Hammer (2742), Oranienstraße 42, 6200 Wiesbaden

Beirat
Peter Kemmerling (2466), Danziger Straße 17, 4030 Ratingen

Beirat
Martin Meyer (1000), Kelkheimer Straße 20, 6232 Bad Soden 1

E-Technik
Werner Meschede (2670), Sorpestr. 4, 5788 Siedlingshausen

Grabau GR7 Interface
Holger von Stillfried (2641), Am Langdiek 13, 2000 Hamburg 61

Hardware 41
Winfried Maschke (413), Ursulakloster 4, 5000 Köln 1,
✆ (0221) 13 12 97

HP-71 Assembler (LEX-Files)
Matthias Rabe (2062), Teichsheide 13, 4800 Bielefeld,
GEO1:M.RABE

Mathematik
Andreas Wolpers (349), Steinstraße 15, 7500 Karlsruhe

Naturwissenschaften
Thor Gehrmann (3423), Hobeuken 18, 4322 Spockhövel 2,
✆ (02339) 39 63

Programmbibliothek HP-71
Henry Schimmer (786), Homburger Landstr. 63,
6000 Frankfurt 50

"Clubadresse"
CCD e.V., Postf. 11 04 11, 6000 Frankfurt 1, ✆ (069) 76 59 12

Postvertriebsstück
Gebühr bezahlt

D 2856 F

CCD - Computerclub Deutschland
Postfach 11 04 11
D-6000 Frankfurt am Main 1

CCD
ISSN 0176-8735
PRISMA

März - Juni 1991

STF

Zeile 15 von STF (106-115) CCD-Barcodes G.Heilmann

Zeile 16 von STF (116-120) CCD-Barcodes G.Heilmann

Zeile 17 von STF (121-125) CCD-Barcodes G.Heilmann