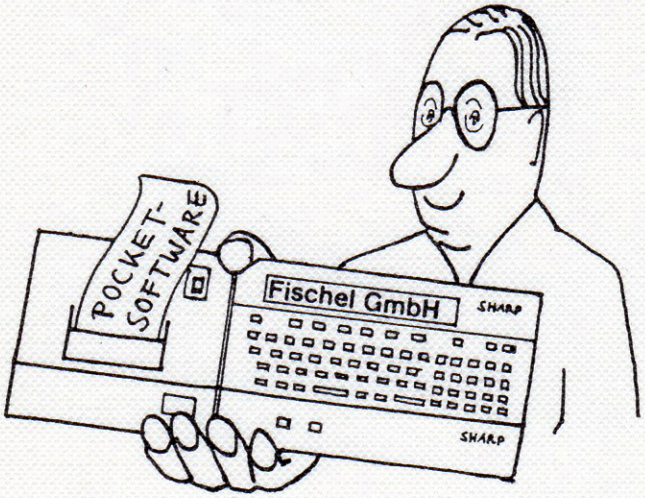
PC-1500

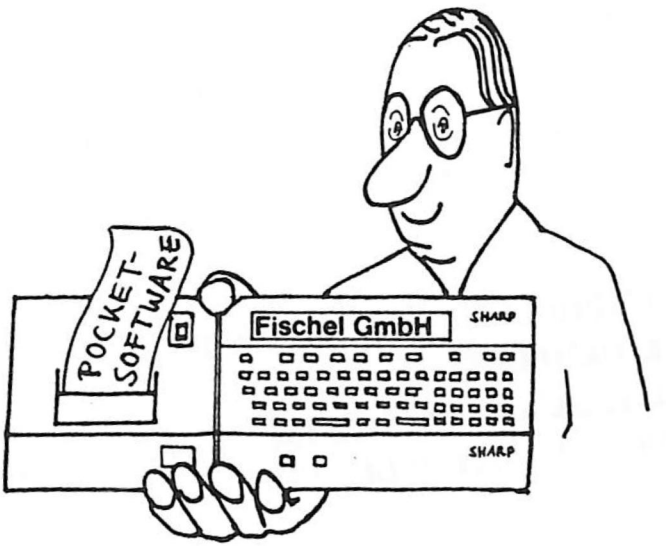
TASCHENCOMPUTER PC-1500 PROGRAMMIER- UND PROGRAMMHANDBUCH



FISCHEL GmbH

PC-1500

TASCHENCOMPUTER PC-1500 PROGRAMMIER- UND PROGRAMMHANDBUCH



FISCHEL GmbH

Fischel GmbH (Hrsg.):
PC-1500 Programmier- und Programmhandbuch.
Berlin, 1984.
ISBN 3-924327-00-9

SHARP - COMPUTER
FISCHEL GMBH
KAISER - FRIEDRICH - STR. 54 A
1000 BERLIN 12

Tel. 030/ 323 60 29

Layout, Cartoons: Johann Hartl
Druck: Offsetdruckerei Gerhard Weinert
Friedrichstr. 224, 1000 Berlin 61

Inhaltsübersicht

Basicprogrammierung

GRUNDGEDANKEN:

Das Rechnen ohne Programmunterstützung ist beim PC-1500 oft sehr umständlich, wenn es über die vier Grundrechenarten hinausgeht. Wollen Sie z.B. die Wurzel aus einer Zahl ziehen,die bereits auf der Anzeige steht,so ist dies mit einigem Tastendrücken verbunden. Die Taste [DEF] und der BASIC-Befehl AREAD schaffen jedoch wesentliche Vereinfachungen.
Haben Sie die folgende Programmzeile im BASIC-Speicher Ihres Taschencomputers,so ziehen Sie Wurzeln wie auf einem herkömmlichen Taschenrechner.

```
1ØØ "A"AREAD A:A=SQR A:PRINT A:END
```

Beispiel:

15 = ?

Tastenbedienfolge: [1] [5] [DEF] [A]

Auf der Anzeige erscheint das Ergebnis.Es bleibt bis zum nächsten CLEAR-Befehl bzw. bis zur Neuverwendung der Variablen A in der Variablen A gespeichert.
Die Taste A hat also in diesem Fall als Zweitbelegung (über [DEF]) die Funktion des Wurzelziehens.So können alle Tasten,die über [DEF] ein Programm aufrufen können,(Siehe Bedienungsanleitung Seite 63) mit einer Taschenrechnerfunktion belegt werden.
Da nun jedoch die Möglichkeit genommen wird andere Programme über DEF aufzurufen,die mit den entsprechenden Tastenbuchstaben markiert sind,hier eine nahezu gleichwertige Alternative.
Sie bezeichnen Ihre Programme nicht nur mit einem,sondern mit zwei oder mehr Buchstaben und setzen diese Bezeichnungen ebenfalls in Anführungsstrichen an die Programmanfänge.Haben Sie die folgende Programmzeile im BASIC-Speicher,so können Sie z.B. das mit den Buchstaben FNA bezeichnete Programm so aufrufen:

Tastenbedienfolge: [F] [N] [A] [DEF] [=]

```
1Ø2 "="AREAD A$:CLS :GOTO A$
```

Über die [DEF] - und [=] -Taste sowie einer Bezeichnung,die in der Anzeige steht,ist mit der Zeile 1Ø2 ein Sprung in alle markierten Programme möglich.

ZUM LISTING:

Die folgenden Taschenrechnerfunktionen lassen sich in jeder beliebigen Programmzeile unterbringen.Daher stehen am Anfang der Zeilen im Listing keine Programmzeilennummern.
Auch die Markierungsbuchstaben und die Verarbeitungs- und Ergebnisvariablen können beliebig getauscht werden und sind daher im Listing nur Vollständigkeitshalber mit angegeben.

Beispiel:

```
"A"AREAD A:A=SIN A:PRINT A:END
```

Wird getauscht zu:

```
"J"AREAD J:J=SIN J:PRINT J:END
```

Sie können daher die Taschenrechnerfunktionen,die Sie für Ihren Anwendungsbereich für Sinnvoll halten,auf die Buchstaben der beiden unteren Tastaturzeilen des PC-1500 verteilen.Die Gleichheit zwischen Markierungsbuchstabe und Verarbeitungs- bzw. Ergebnisvariable hat den bereits weiter oben beschriebenen Vorteil,daß das Ergebnis unter dem Buchstaben weiter mit hoher Genauigkeit zur Verfügung steht.

In der Funktionsbeschreibung hinter den Programmzeilen haben die kleinen Buchstaben folgende Bedeutung:

z -- Ergebnis(erscheint auf der Anzeige)
x -- Wert aus der Anzeige
y -- Zusätzlicher Wert,der über INPUT "Y=";... erfragt wird.
n -- Konstante

LISTING:

Trigonometrische Funktionen:

```
"A"AREAD A:A=SIN A:PRINT A:END                          z=sin x
"S"AREAD S:S=COS S:PRINT S:END                          z=cos x
"D"AREAD D:D=ASN D:PRINT D:END                          z=arcsin x
"F"AREAD F:F=ACS F:PRINT F:END                          z=arccos x
"G"AREAD G:G=TAN G:PRINT G:END                          z=tan x
"H"AREAD H:H=1/TAN H:PRINT H:END                        z=cot x
"J"AREAD J:J=ATN J:PRINT J:END                          z=arctan x
"K"AREAD K:K=1/ATN K:PRINT K:END                        z=arccot x
```

Mathematische Funktionen:

```
"L"AREAD L:L=1/L:PRINT L:END                            z=1/x
"Z"AREAD Z:Z=Z 2:PRINT Z:END                            z=x^2
"X"AREAD X:X=SQR X:PRINT X:END                          z=√x
"C"AREAD C:INPUT "Y=";Q:C=C Q:PRINT C:END               z=x^y
"V"AREAD V:INPUT "Y=";Q:V=V (1/Q):PRINT V:END           z=y√x
"B"AREAD B:W=1:FOR Q=1TO B:W=W*Q:NEXT Q:
   B=W:PRINT B:END                                      z=x!
```

Logarithmische Funktionen:

```
"A"AREAD A:A=LOG A:PRINT A:END                          z=lg x
"S"AREAD S:S=10^S:PRINT S:END                           z=10^x
"D"AREAD D:D=LN D:PRINT D:END                           z=ln x
"F"AREAD F:F=EXP F:PRINT F:END                          z=e^x
"G"AREAD G:INPUT"Y=";Q:G=LOG G/LOG Q:PRINT Q:E          z=log_y x
"H"AREAD H:INPUT"Y=";Q:H=Q^H:PRINT H:END                z=y^x
```

Hyperbelfunktionen:

```
"Z"AREAD Z:Z=(EXP Z-EXP -Z)/2:PRINT Z:END               z=sinh x
"X"AREAD X:X=LN (X+√(X^2+1)):PRINT X:END                z=arsinh x
"C"AREAD C:C=(EXP C+EXP -C)/2:PRINT C:END               z=cosh x
"V"AREAD V:V=LN (V+√(V^2-1)):PRINT V:END                z=arcosh x
"B"AREAD B:B=(EXP B-EXP -B)/(EXP B+EXP -B):
   PRINT B:END                                          z=tanh x
"N"AREAD N:N=(LN ((1+N)/(1-N)))/2:PRINT N:END           z=artanh x
"M"AREAD M:M=(EXP M+EXP -M)/(EXP M-EXP -M):
   PRINT M:END                                          z=coth x
"L"AREAD L:L=(LN ((L+1)/(L-1)))/2:PRINT L:END           z=arcoth x
```

■ Winkeltransformation:

```
"A"AREAD A:A=SIN A:DEGREE:A=ASN A:PRINT A:END      z=x
"S"AREAD S:S=SIN S:RADIAN:S=ASN S:PRINT S:END      z=x
"D"AREAD D:D=SIN D:GRAD :D=ASN D:PRINT D:END       z°=x
```

■ Transformation von Polar- und Rechteckkoordinaten:

```
"F"AREAD F:INPUT"jY=";Q:W=√(F*F+Q*Q):E=(Q=Ø)
   +SGN Q:Q=ACS (F/W)*E:PRINT W:PRINT Q:END      w⌊q=x+jy
"G"AREAD G:INPUT"PHI=";Q:W=G*COS Q:Q=G*SIN Q:
   PRINT W:PRINT Q:END                           w+jq=x⌊PHI
```

■ Mittelwertbildung:

Arithmetisches Mittel: $\bar{A}=1/w \sum_{i=1}^{w} a_i$

```
"Z"AREAD Z:Q=Q+Z:W=W+1:Z=Q/W:PRINT Z:END
```

Geometrisches Mittel: $G=\sqrt[w]{a_1*a_2*\ldots*a_w}$

```
"C"AREAD C:E=E*C:W=W+1:C=E (1/W):PRINT C:END
```

Harmonisches Mittel: $H=\dfrac{W}{\sum_{i=1}^{-W} 1/a_i}$

```
"V"AREAD V:Q=Q(1/W):W=W+1:V=W/Q:PRINT V:END
```

Rücksetzen der Laufvariablen bei Rechenbeginn:

```
"X"W=Ø:Q=Ø:E=1:END
```

■ Zwischenspeicher:

```
"D"AREAD D:END                                    Memory
"A"AREAD Q:A=A+Q:PRINT A:END                      Memory +
"S"AREAD Q:S=S-Q:PRINT S:END                      Memory ---
```

■ Nachkommastellen,Runden:

```
"F"AREAD F:F=INT (F*1Ø^n)/1Ø^n:PRINT F:END
"G"AREAD G:G=INT (G*1Ø^n+Ø.5)/1Ø^n:PRINT G:END  Runden
```

n - Zahl der Nachkommastellen


AN DIESER STELLE
MÖCHTE ICH SCHON
EINMAL VORSORGLICH
AUF MEINE U(H)RHEBER-
RECHTE HINWEISEN!

Zufallszahlen

Beobachtet man die Ziehung der Lottozahlen, so kann man berechtigten Zweifel an die wirkliche Zufälligkeit der gezogenen Zahlen anmelden. (Dieser Zweifel ist beim Mittwochlotto besonders berechtigt.) Jedermann weiß, daß man gleichschwere Gegenstände (Kugeln) nicht herstellen kann. Die Kugeln haben alle ein unterschiedliches Gewicht. Auch wenn der Gewichtsunterschied noch so winzig klein sein sollte, die Schwerkraft ist mit von der Partie, und die Wahrscheinlichkeitsrechnung bringt es bei großer Anzahl von Ziehungen an den Tag.
Mit der Erzeugung von Zufallszahlen hat man sich auch in der Mathematik beschäftigt. Es wurden Tabellen von zufälligen Zahlen entworfen und auch wieder verworfen. Es wird behauptet, daß es wirkliche Zufallszahlen niemals geben wird. Wenn man aber Zahlen vorliegen hat, in denen man keine Gesetzmäßigkeiten erkennen kann, so werden diese als Zufallszahlen anerkannt und benutzt.
Kurz nachdem die ersten elektronischen Rechenmaschinen entwickelt waren (1945), kam der Wunsch auf, auch mit dem Computer Zufallszahlen zu erzeugen. Ein geeigneter Algorithmus mußte gefunden werden. J. v. Neumann war der erste der einen Vorschlag hierzu machte.
Es war die Quadratmittenmethode. Man geht aus von einer beliebigen, jedoch geeignet großen Zahl, quadriert diese und nimmt den mittleren Teil davon. Dieses wäre dann die erste Zufallszahl.
Diese Zahl quadriert man erneut und nimmt wieder den mittleren Teil davon. Auf diese Weise entstehen sogenannte Zufallszahlen, die aus den vorhergehenden gewonnen werden.
Wie sich jedoch schnell herausstellte, war dieser Algorithmus sehr schlecht. Taucht nämlich eine Null in der Zufallszahl auf, so wird man diese in den darauf folgenden Zahlen nicht mehr los.
Es sind also keine guten Zufallszahlen. Inzwischen sind andere, bessere Algorithmen entwickelt worden. Sämtliche Algorithmen für Zufallszahlen haben eines gemein. Die neue Zufallszahl ist gekoppelt an die vorherigen Zahlen, nur sieht man diese Koppelung nicht so schnell. Auch der Sharp-1500 kann Zufallszahlen erzeugen. Der Algorithmus hierzu ist nirgends angegeben.
Ob der Sharp-1500 einen guten oder schlechten Algorithmus für Zufallszahlen verwendet, können wir vorerst nicht entscheiden. Hierzu existieren eine Reihe von Test-Algorithmen.

```
10 19 32 39 7 6

PC-1500   "AUS"
PC-1500   "EIN"

10 19 32 39 7 6

PC-1500   "AUS"
PC-1500   "EIN"

10 19 32 39 7 6


 8:I=0:X=49
12:I=I+1
13:IF I>6GOTO 18
14:LPRINT RND X;
16:GOTO 12
18:TEXT
```

Daß auch der Sharp-1500 seine Zufallszahlen aus den vorhergehenden gewinnt, möge durch nebenstehendes kleine Programm "6 aus 49" belegt sein. (Die existierende Anfangszahl im Rechner ist offenbar nicht zu beeinflussen.)
Diese Art von Wiederholung schränkt die Güte der Zufallszahlen nicht ein. Innerhalb einer Kette von Zufallszahlen dürfen jedoch keine Wiederholungen auftreten. Wir wollen uns nun einer Aufgabe zuwenden, die typisch für die Anwendung des Zufallgenerator ist.

Aufg.: Zwei Menschen A und B wollen sich täglich zwischen 12:00 Uhr und 13:00 Uhr zu einem Erfahrungsaustausch treffen. Beide nehmen diese Vereinbarung nicht sehr wichtig, zumal sie sich an den darauf folgenden Tagen sicherlich einmal treffen werden. Jeder beschließt auf den anderen 15 Min. zu warten.
Wie groß ist die Wahrscheinlichkeit, daß beide sich treffen?
Es gibt zwei Möglichkeiten diese Aufgabe zu lösen.
a) Durch theoretische Überlegungen b) Durch ein Experiment

Wir wollen im weiteren den Fall b) verfolgen.
Man beobachtet die beiden Menschen A und B an möglichst vielen Tagen. Da dieses jedoch sehr zeitraubend ist, wollen wir ein Programm darüber schreiben und das Eintreffen der beiden durch Zufallszahlen simulieren.

Es folgen die grundsätzlichen Überlegungen zum Programm:

A und B können unabhängig voneinander zu unendlich (überabzählbar) vielen Zeitpunkten eintreffen. Dieses ist nicht computergerecht. Wir wollen deshalb nur das minutenweise Eintreffen unterscheiden. (D. h. es wird vom kontinuierlichen zum diskreten Fall übergegangen.) Den dadurch entstehenden "geringen" Fehler nehmen wir inkauf.
Wenn T1 die Eintreffzeit von A und T2 die Eintreffzeit von B bezeichnet, so ist es zu einem Treffen gekommen, wenn $ABS(T1-T2) \leq 15$ gilt.

Es bezeichne N die Zahl der Tage, an denen Beobachtungen stattfinden sollen. J bezeichne die Anzahl der Tage, an denen es zu einer Begegnung der beiden gekommen ist. W=J/N ist die gesuchte Wahrscheinlichkeit. Hat ein Treffen stattgefunden, setzen wir T=1. Andernfalls Ø.

Nr.	A	B	T
*1	*3	59	*0
*2	19	57	*0
*3	33	24	*1
*4	*1	21	*0
*5	55	*5	*0
*6	45	12	*0
*7	20	29	*1
*8	*6	*8	*1
*9	55	46	*1
10	18	36	*0
11	38	17	*0
12	22	24	*1
13	54	33	*0
14	33	37	*1
15	*9	11	*1
16	*7	41	*0
17	41	29	*1
18	53	*3	*0
19	54	27	*0
20	11	11	*1
21	*2	44	*0
22	43	24	*0
23	10	29	*0
24	51	18	*0
25	52	42	*1
26	*3	*6	*1
27	*6	*9	*1
28	17	29	*1
29	48	11	*0
30	11	*6	*1
31	15	31	*0
32	52	49	*1
33	33	36	*1
34	47	59	*1
35	25	29	*1
36	48	10	*0
37	30	16	*1
38	60	56	*1
39	19	14	*1
40	11	12	*1
41	21	52	*0
42	44	46	*1
43	28	34	*1
44	55	53	*1
45	*1	17	*0
46	21	50	*0
47	53	15	*0
48	38	34	*1
49	51	33	*0
50	31	47	*0
51	49	31	*0
52	36	48	*1
53	15	31	*0
54	45	*9	*0
55	11	58	*0
56	13	46	*0
57	27	*5	*0
58	55	50	*1
59	*7	33	*0
60	28	34	*1

W= 0.48

```
 8:"A":INPUT N
10:LPRINT "␣␣Nr."
   ;"␣␣␣A";"␣␣␣B"
   ;"␣␣␣T"
11:LF +1
12:I=0:J=0
14:X=60:Y=60
16:T1=RND X:T2=
   RND Y
18:I=I+1
20:IF I>=N+1GOTO
   34
22:IF ABS (T1-T2)
   <=15GOTO 27
23:USING "***"
24:LPRINT "␣";I;"
   ␣␣";T1;"␣";T2;
   "␣";Ø
25:USING
26:GOTO 16
27:USING "***"
28:LPRINT "␣";I;"
   ␣␣";T1;"␣";T2;
   "␣";1
29:USING
31:J=J+1
32:GOTO 16
34:LF +1
35:USING "###.##"
36:LPRINT "␣␣W=";
   J/N
38:USING

STATUS 1
                310
```

Anmerkung:

a) Die theoretisch errechnete Wahrscheinlichkeit für den diskreten Fall (60 min.) beträgt W=0.45

b) Die Wahrscheinlichkeit für den kontinuierlichen Fall beträgt W=0.4375

LEN, STR$, VAL, LEFT$, MID$, RIGHT$

Werden die natürl. Zahlen 1,2,3,4,... tabellarisch (untereinander) aufgeschrieben und rechts daneben die zugehörigen Quersummen, so stellt man eine Regelmäßigkeit fest.
Die Quersummen der Zahlen laufen von 1 bis 9 und wiederholen sich in dieser Reihenfolge ständig. Diesen Umstand wollen wir ausnutzen, um die Zufallszahlen des PC-1500 auf ihre "Güte" (Zufälligkeit) zu prüfen.

N	Q
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	1
11	2
12	3
13	4
14	5
15	6
16	7
17	8
18	9
19	1
20	2
21	3
.	.
.	.
.	.

Wir ordnen den Zufallszahlen ihre Quersummen zu, und haben dadurch nur Zahlen zwischen 1 und 9 zu betrachten. Für Mathematiker bzw. mathem. Versierte wird der Hinweis "modulo 9" hinzugefügt.
Es werden zwei beliebige, jedoch computergerechte, natürl. Zahlen X und Y ausgewählt. Durch den Rechner lassen wir eine Kette von Zufallszahlen zwischen 1 und 9*X ausdrucken. Die Länge der Kette sollte aus "demokratischen Gründen" 9*Y sein. Gleichzeitig lassen wir durch den Rechner zu jeder Zufallszahl die Quersumme errechnen und rechts daneben ausdrucken.
Eine Prüfmöglichkeit der Zufallszahlen wäre nun, daß man die Anzahl der Zufallszahlen mit jeweilig fester Quersumme ermittelt und durch die Gesamtzahl der ausgedruckten Zufallszahlen (hier 9*Y) dividiert. Dieses Verhältnis müßte bei größerer Zahl Y ungefähr 1/9 betragen. Diese Vorgehensweise erfordert eine wirklich große Anzahl von Zufallszahlen (also großes Y), um verläßliche Ergebnisse zu erzielen. Wir wollen deshalb die beschriebene Prüfung modifizieren, indem wir nur einen geeigneten Mittelwert betrachten.
Hierzu die folgenden Bezeichnungen:
A1=Anzahl der Zufallszahlen mit Quersumme 1
A2=Anzahl der Zufallszahlen mit Quersumme 2
⋮
A9=Anzahl der Zufallszahlen mit Quersumme 9
M0=(1+2+...+9)/9 =5 ist der ideale Mittelwert
M1=(A1*1+A2*2+...+A9*9)/9*Y ist der reale Mittelwert
Wenn die Zufallszahlen des Rechners von großer Güte sind, so sollte bei großem Y, der Wert M1 möglichst nahe bei M0=5 liegen.
Dieses soll überprüft werden.

Sicherlich gibt es mehrere verschiedene Methoden um die Quersumme einer Zahl auszurechnen. Wir wollen jedoch die BASIC-Befehle LEN, STR$, VAL, LEFT$, MID$ und RIGHT$ anwenden und schlagen deshalb den hier schrittweise beschriebenen Weg ein.

Es soll die Quersumme der dreistelligen Zahl N=152 durch den Computer ausgerechnet werden.

1. Idee: Es liegen die Befehle LEFT$, MID$ und RIGHT$ vor. Schreiben wir, unter Beachtung wie die Befehle anzuwenden sind, ein kurzes Programm.

```
10:N=152
12:L$=LEFT$ (N,1):M$=MID$ (N,2,1):R$=RIGHT$ (N,1)
14:Q$=L$+M$+R$
15:PRINT Q$
```

Nach Betätigung von RUN-ENTER steht im Display: ERROR 17 IN 12. Die Befehle LEFT$, MID$ und RIGHT$ lassen sich nur auf Text-Variable anwenden, und N ist numerische Variable.

2. Idee: Wir wandeln die numerische Variable N, durch den Befehl STR$, in eine Text-Variable um. Korrigieren wir obiges Programm.

```
10:N=152
11:S$=STR$ N
12:L$=LEFT$ (S$,1):M$=MID$ (S$,2,1):R$=RIGHT$ (S$,1)
14:Q$=L$+M$+R$
15:PRINT Q$
```

Nach Betätigung von RUN-ENTER steht im Display: 152.
Wir haben also nichts erreicht. Der Rechner hat die Einzeltexte 1,2 und 3 wieder zusammengefügt.

3. Idee: Wir wandeln die einzelnen Text-Variablen L$, M$ und R$, durch den Befehl VAL wieder um in numerische Variablen und addieren danach. Korrigieren wir obiges Programm erneut.

```
10:N=152
11:S$=STR$ N
12:L$=LEFT$ (S$,1):M$=MID$ (S$,2,1):R$=RIGHT$ (S$,1)
13:L=VAL L$:M=VAL M$:R=VAL R$
14:Q=L+M+R
15:PRINT Q
```

Nach Betätigung von RUN-ENTER steht im Display endlich das richtige Ergebnis: 8.

Der hier beschriebene Vorgang stellt die Grundidee des Programms auf der nächsten Seite dar. Die Abfrage, ob die Zahl N einziffrig, zweiziffrig, dreiziffrig oder gar vierziffrig ist, erfolgt durch den Befehl LEN, über den Umweg Text-Variable.

X=13:Y=6

I	N	Q
1	57	3
2	2	2
3	38	2
4	43	7
5	46	1
6	3	3
7	54	9
8	68	5
9	24	6
10	78	6
11	32	5
12	32	5
13	16	7
14	6	6
15	12	3
16	39	3
17	74	2
18	54	9
19	51	6
20	98	8
21	11	2
22	13	4
23	63	9
24	40	4
25	100	1
26	57	3
27	22	4
28	20	2
29	101	2
30	97	7
31	106	7
32	77	5
33	113	5
34	15	6
35	89	8
36	53	8
37	46	1
38	114	6
39	32	5
40	29	2
41	63	9
42	42	6
43	8	8
44	47	2
45	24	6
46	84	3
47	52	7
48	11	2
49	8	8
50	61	7
51	109	1
52	36	9
53	105	6
54	70	7

4 9 6
3 6 9 7
5 5
270 54
M1= 5.00

X=78:Y=6

I	N	Q
1	493	7
2	95	5
3	58	4
4	630	9
5	440	8
6	285	6
7	234	9
8	460	1
9	44	8
10	295	7
11	457	7
12	673	7
13	15	6
14	332	8
15	595	1
16	339	6
17	72	9
18	252	9
19	178	7
20	574	7
21	561	3
22	261	9
23	369	9
24	49	4
25	403	7
26	124	7
27	31	4
28	693	9
29	474	6
30	357	6
31	473	5
32	332	8
33	605	2
34	571	4
35	494	8
36	118	1
37	598	5
38	220	4
39	125	8
40	46	1
41	356	5
42	458	9
43	508	4
44	444	3
45	382	4
46	345	3
47	206	8
48	525	3
49	123	6
50	20	2
51	448	7
52	455	5
53	629	8
54	410	5

4 2 4
7 6 6 9
8 8
313 54
M1= 5.79

X=783:Y=6

I	N	Q
1	5437	1
2	5249	2
3	911	2
4	6856	7
5	2633	5
6	4165	7
7	4181	5
8	4546	1
9	5894	8
10	1668	3
11	3115	1
12	1157	5
13	5470	7
14	5994	9
15	3957	6
16	6434	8
17	7030	1
18	6654	3
19	5036	5
20	3065	5
21	20	2
22	454	4
23	3377	2
24	144	9
25	3305	2
26	5537	2
27	502	7
28	4489	7
29	4581	9
30	6694	7
31	5958	9
32	3119	5
33	1255	4
34	672	6
35	1357	7
36	3005	8
37	5679	9
38	3756	3
39	1802	2
40	6190	7
41	1415	2
42	4353	6
43	1444	4
44	5006	2
45	2374	7
46	5266	1
47	1307	2
48	1865	2
49	594	9
50	6595	7
51	3696	6
52	442	1
53	3118	4
54	1222	7

6 11 3
4 6 4 11
3 6
262 54
M1= 4.85

```
  2:REM QUERSUMMEN
    TEST FUER ZUFA
    LLSZAHLEN
  4:X=13:Y=6
  6:USING "#####"
  7:LPRINT "     I"
    ;"     N";"
    Q"
  8:LF +1
  9:P=9%X:E=9%Y:I=
    0
 10:A1=0:A2=0:A3=0
    :A4=0:A5=0:A6=
    0:A7=0:A8=0:A9
    =0
 12:N=RND P
 13:I=I+1
 14:IF I=E+1GOTO 1
    00
 22:S$=STR$ N
 28:W=LEN S$
 31:IF W=1LET Q=N:
    GOTO 67
 32:IF W=2GOTO 40
 33:IF W=3GOTO 50
 34:IF W=4GOTO 60
 40:L$=LEFT$ (S$,1
    ):R$=RIGHT$ (S
    $,1)
 41:L=VAL L$:R=VAL
    R$
 42:Q=L+R
 43:S$=STR$ Q
 44:W=LEN S$
 45:IF W=2GOTO 40
 46:GOTO 67
 50:L$=LEFT$ (S$,1
    ):M$=MID$ (S$,
    2,1):R$=RIGHT$
    (S$,1)
 51:L=VAL L$:M=VAL
    M$:R=VAL R$
 52:Q=L+M+R
 53:S$=STR$ Q
 54:W=LEN S$
 55:IF W=2GOTO 40
 56:GOTO 67
 60:L$=LEFT$ (S$,1
    ):M1$=MID$ (S$
    ,2,1):M2$=MID$
    (S$,3,1):R$=
    RIGHT$ (S$,1)
 61:L=VAL L$:M1=
    VAL M1$:M2=VAL
    M2$:R=VAL R$
 62:Q=L+M1+M2+R
 63:S$=STR$ Q
 64:W=LEN S$
 65:IF W=2GOTO 40
 67:LPRINT I;N;Q
```

```
 69:REM  ZAELWERK
 70:IF Q=1GOTO 80
 71:IF Q=2GOTO 82
 72:IF Q=3GOTO 84
 73:IF Q=4GOTO 86
 74:IF Q=5GOTO 88
 75:IF Q=6GOTO 90
 76:IF Q=7GOTO 92
 77:IF Q=8GOTO 94
 78:IF Q=9GOTO 96
 80:A1=A1+1
 81:GOTO 12
 82:A2=A2+1
 83:GOTO 12
 84:A3=A3+1
 85:GOTO 12
 86:A4=A4+1
 87:GOTO 12
 88:A5=A5+1
 89:GOTO 12
 90:A6=A6+1
 91:GOTO 12
 92:A7=A7+1
 93:GOTO 12
 94:A8=A8+1
 95:GOTO 12
 96:A9=A9+1
 97:GOTO 12
 99:REM BILANZ
100:LF +1
104:A=A1%1+A2%2+A3
    %3+A4%4+A5%5+A
    6%6+A7%7+A8%8+
    A9%9
106:LPRINT A1;A2;A
    3;A4;A5;A6;A7;
    A8;A9
107:LPRINT A;I-1
110:USING
112:USING "###.##"
114:M1=A/(I-1)
116:LPRINT "   M1=
    ";M1
118:USING

STATUS 1
                1116
```

Kommentar:

1) Die Eingabe der Daten X, Y geschieht in Zeile 4.

2) Alternativ kann Zeile 4 ersetzt werden durch:

```
4:"A":INPUT "X="
  ;X;"Y=";Y
```

(Manch einer zieht die Eingabe über INPUT vor.)

3) Die Zeilen 106 und 107 könnten zur Kürzung des Programms weggelassen werden. Diese beiden Zeilen bewirken lediglich das Ausdrucken, wie oft Zufallszahlen mit fester Quersumme beim Test vorgekommen sind. Ferner wird das Zwischenergebnis A und die Laufzahl I vermindert um 1 ausgedruckt. In diesem Sinne bedeuten die Zahlen beim rechten Test auf der Vorseite ganz unten

```
   6   11   3
 4   6   4  11
   3   6
 262   54
```

6 Zufallszahlen mit Quersumme 1
11 Zufallszahlen mit Quersumme 2
u.s.w.
Die Zahl 262 ist A.

Das Display des PC-1500

In der Sharp-Zeitung Heft 8, 83 war unter dem Titel "Kino" ein interessantes PC-1500 Programm vorgestellt worden. Ein Männchen läuft über das Display und schiebt Buchstaben. Dem Autor dieses Programms, dessen Namen nicht genannt wurde, muß bescheinigt werden, daß er ein Könner auf dem Gebiete des Displays ist.
Hierdurch angeregt, soll ein Programm entworfen werden, in welchem wir uns in jenen Befehlen üben, die das Display betreffen.
Es sind die Befehle GCURSOR und GPRINT.
Damit dieses Üben nicht ohne Motivation ist, wollen wir uns eine Aufgabe stellen. Dieser Aufgabe werden, zum besseren Verständnis, folgende allgemeine Bemerkungen vorangeschickt:

"Operations-Research" ist, genau wie "Informatik" eine Neudisziplin an den Universitäten. Die deutsche Übersetzung dieses Fachgebiets wäre "Unternehmensforschung". Entstanden ist Unternehmensforschung während des 2. Weltkriegs und hat militärischen Ursprung.
Aus dem Bereich "Operations-Research" wollen wir das Warteschlangenproblem aufgreifen.

Geht man zur Bank, Post oder ähnlichen öffentlichen Einrichtungen, so ist dieses oftmals mit unangenehmen Wartezeiten verbunden.
Es haben sich Warteschlangen gebildet und man stellt sich hinten an oder geht unverrichteter Dinge wieder weg.
Dieses hat die Unternehmerseite nicht gerne, denn es gibt Geschäftsverluste und verärgerte Kundschaft. Es wird deshalb Reservepersonal in Bereitschaft gehalten, um bei größerem Andrang mehr Schalter öffnen zu können.
Andererseits kann es auch vorkommen, daß kaum Kundschaft da ist, und das Personal sitzt tatenlos herum. Dieses hat die Unternehmerseite wegen der unnötigen Personalkosten auch nicht gerne.
Die Geschäftsleitung wird einen Kompromiß anstreben. Hierzu werden Beobachtungen angestellt, in welchem mittleren Zeitabstand die Kundschaft den Schalterraum betritt und wie schnell die mittlere Abfertigungszeit am Schalter ist.
Der Unternehmer wird daraus Schlüsse ziehen, wieviel Personal er bereit halten muß, um gewisse Unregelmäßigkeiten im Publikumverkehr auszugleichen. (Daß hierbei die unterschiedlichen Tageszeiten auch eine Rolle spielen versteht sich von selbst.)

Aufg.: Die Entwicklung von Warteschlangen in einer öffentlichen Institution soll im Display des PC-1500 simuliert werden. Durchschnittlich betritt alle T1 Sek. ein Kunde den Schalterraum. Die Durchschnittliche Abfertigungszeit beträgt T2 Sek. Es sei T1 kleiner-gleich T2 (in Zeichen T1≤T2), denn sonst bildet sich keine Warteschlange.

Nun sind die mathematischen Gegebenheiten hierzu kompliziert, so daß wir zu recht drastischen Vereinfachungen gezwungen sind.

a) Ein Schalter ist geöffnet.
Alle T1 Sek. schließt sich ein Kunde an die Schlange an. Pro Sek. wird die Schlange somit um 1/T1 Kunden länger. Alle T2 Sek. wird ein Kunde am Schalter abgefertigt. Pro Sek. wird die Warteschlange um 1/T2 Kunden kürzer. Die resultierende wirkliche Verlängerung der Schlange beträgt somit 1/T1-1/T2 = (T2-T1)/T1*T2.
Wie lange dauert es nun, bis die Schlange um einen Kunden länger geworden ist?
Diese Zeit ist S1 = T1*T2/(T2-T1).
Angenommen, die Schlange wird immer länger. Warten in der Schlange mehr als A Kunden, so wird ein zweiter Schalter geöffnet. Nun laufen, wie der Leser aus eigener Erfahrung bestätigen wird, ca. die Hälfte der Kunden über zum zweiten Schalter. (Hierbei entstehen oft Ungerechtigkeiten. Leute aus dem hinteren Teil der Schlange stehen plötzlich ganz vorne am neu geöffneten Schalter.)

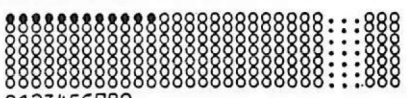
b) Zwei Schalter sind geöffnet.
Alle 2*T1 Sek. schließt sich ein Kunde an jeweils eine Schlange an. Pro Sek. wird jede der beiden Schlangen um 1/2*T1 Kunden länger. Alle T2 Sek. wird an jedem der beiden Schalter jeweils ein Kunde abgefertigt. Pro Sek. wird jede der beiden Schlangen um 1/T2 Kunden kürzer. Die resultierende wirkliche Verlängerung (Verkürzung) der Schlangen beträgt somit 1/2*T1-1/T2 = (T2-2*T1)/2*T1*T2.
Wie lange dauert es nun, bis jede der beiden Schlangen um einen Kunden länger (kürzer) geworden ist?
Diese Zeit ist S2 = 2*T1*T2/(T2-2*T1).
Ist S2 negativ, so werden beide Schlangen immer kürzer. Sind beide Schlangen auf die Länge null abgebaut, so soll der zweite Schalter wieder schließen.

Gilt T2=2*T1, so bleiben beide Schlangen der Länge nach konstant. Ist hingegen S2 posetiv, so werden beide Schlangen immer länger. Warten in beiden Schlangen jeweils mehr als A Kunden, so wird ein dritter Schalter geöffnet, und die Kunden verteilen sich gleichmäßig auf alle drei Schalter.

c) Drei Schalter sind geöffnet.

Alle 3*T1 Sek. schließt sich ein Kunde an jeweils eine Schlange an. Pro Sek. wird jede der drei Schlangen um 1/3*T1 Kunden länger. Alle T2 Sek. wird an jedem der drei Schalter jeweils ein Kunde abgefertigt. Pro Sek. wird jede der drei Schlangen um 1/T2 Kunden kürzer. Die resultierende wirkliche Verlängerung (Verkürzung) der Schlangen beträgt somit 1/3*T1-1/T2 = (T2-3*T1)/3*T1*T2.
Wie lange dauert es nun, bis jede der drei Schlangen um einen Kunden länger (kürzer) geworden ist?
Diese Zeit ist S3 = 3*T1*T2/(T2-3*T1).
Ist S3 negativ, so werden alle drei Schlangen immer kürzer.
Sind alle drei Schlangen auf die Länge null abgebaut, so soll der dritte Schalter wieder schließen.
Gilt T2=3*T1, so bleiben alle drei Schlangen der Länge nach konstant.
Ist hingegen S3 posetiv, so werden die drei Schlangen immer länger.
Warten in allen drei Schlangen jeweils mehr als A Kunden, so bleibt es dem Leser überlassen einen vierten Schalter in seinem Programm zu berücksichtigen. Wir legen hier das Programm für maximal drei geöffnete Schalter aus.

Einige Bemerkungen zur Darstellung der Warteschlangen im Display

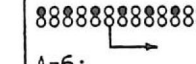
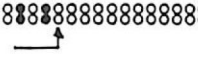
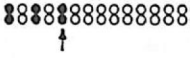
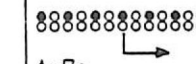
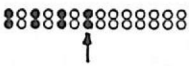
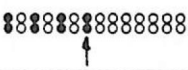


Nebenstehendes Bild soll das Display darstellen. Es ist hierbei ein Schalter geöffnet und zwölf Kunden warten davor.

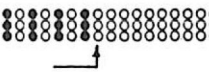
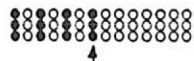
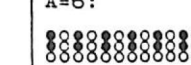
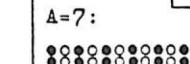
Die Erfahrung mit dem Display, bezüglich dieses Programms, hat gezeigt, daß das Auge die waagerechten Leuchtpunkte schlecht trennt. Das Programm wurde (im Nachhinein) so abgefaßt, daß nur jeder zweite waagerechte Punkt im Display angesteuert wird. Nebenstehendes Bild zeigt zwölf wartende Kunden vor einem geöffneten Schalter. (Dieses tritt ein bei der Wahl von A=11 Kunden im Programm.)

Da jeweils die "Übergänge" (ein zusätzlicher Schalter öffnet) größere Schwierigkeiten bereiteten, sollen diese etwas genauer dargelegt werden.

Vor einem geöffneten Schalter stehen mehr als A Kunden und ein zweiter Schalter wird geöffnet. Dieser Vorgang wird für die Fälle A=5, A=6 und A=7 dargelegt.

Löschen (Zeilen 35-56)	Aufbau (Zeilen 65-86)	Weiter (Zeilen 95-128)
Löschen ab A	74:IF K>A GOTO 1ØØ	Weiter ab L=2*INT(A/2) Anz. der Kunden =INT(A/2)+1
A=5: 		
A=6: 		
A=7: 		

Vor zwei geöffneten Schaltern stehen jeweils A Kunden und ein dritter Schalter wird geöffnet. Dieser Vorgang wird ebenfalls für die Fälle A=5, A=6 und A=7 dargelegt.

Löschen (Zei. 135-156)	Aufbau (Zei. 165-186)	Weiter (Zei. 195-228)
B=2*A-INT((2*A+1)/3) Löschen ab B+1	174:IF N>B GOTO 2ØØ	Weiter ab P=2*INT(2*A/3) Anz. der Kunden =INT(2*A/3)+1
A=5: 		
A=6: 		
A=7: 		

```
  2:REM  WARTESCHL
    ANGE
  3:T1=1:T2=50:A=5
    0:CLS
  4:IF T2=T1GOTO 2
    30
  5:IF T2=2*T1GOTO
    231
  6:IF T2=3*T1GOTO
    232
  7:REM  1 SCHALTE
    R AUF
  8:S1=T1*T2/(T2-T
    1):I=-2
 10:I=I+2
 14:IF I>2*AGOTO 3
    6
 20:GCURSOR I:
    GPRINT 1;
 24:WAIT S1
 28:GOTO 10
 35:REM  LOESCHEN
    AB A
 36:J=A-1
 40:J=J+1
 44:IF J>2*AGOTO 6
    6
 48:GCURSOR J:
    GPRINT 0;
 52:WAIT 0
 56:GOTO 40
 65:REM AUFBAU VON
    0-A
 66:K=-2
 70:K=K+2
 74:IF K>AGOTO 100
 78:GCURSOR K:
    GPRINT 3;
 82:WAIT 0
 86:GOTO 70
 95:REM 2 SCHALTER
     AUF
100:S2=2*T1*T2/(T2
    -2*T1):L=2*INT
    (A/2):U=0
104:L=L+2*U
108:IF L=-2GOTO 8
112:IF L>2*AGOTO 1
    36
116:GCURSOR L:
    GPRINT 3;0;0;
120:U=SGN (T2-2*T1
    )
124:WAIT ABS (S2)
128:GOTO 104
```

```
135:REM  LOESCHEN
    AB B+1 WOBEI B
    =2*A-INT((2*A+
    1)/3)
136:M=2*A-INT ((2*
    A+1)/3)
140:M=M+1
144:IF M>2*AGOTO 1
    66
148:GCURSOR M:
    GPRINT 0;
152:WAIT 0
156:GOTO 140
165:REM  AUFBAU VO
    N 0-B WOBEI B=
    2*A-INT((2*A+1
    )/3)
166:N=-2:B=2*A-INT
    ((2*A+1)/3)
170:N=N+2
174:IF N>BGOTO 200
178:GCURSOR N:
    GPRINT 7;
182:WAIT 0
186:GOTO 170
195:REM 3 SCHALTER
     AUF
200:S3=3*T1*T2/(T2
    -3*T1):P=2*INT
    (2*A/3):U=0
204:P=P+2*U
208:IF P=-2GOTO 8
216:GCURSOR P:
    GPRINT 7;0;0;
220:U=SGN (T2-3*T1
    )
224:WAIT ABS (S3)
228:GOTO 204
230:PRINT "1 SCHL.
     KON. 1 KUND."
231:PRINT "2 SCHL.
     KON. JEW.";
    INT (A/2)+1;"K
    UND."
232:PRINT "3 SCHL.
     KON. JEW.";
    INT (2*A/3)+1;
    "KUND."

STATUS 1
                1018
```

Kommentar:

1) Die Eingabe der Daten T1, T2 u. A geschieht in Zeile 3. Drücke im Run-Mode RUN-ENTER u. noch einmal ENTER.

2) Alternativ kann Zeile 3 ersetzt werden durch:

```
3:"S":INPUT "T1=
  ";T1,"T2=";T2,
  "A=";A:CLS
```

Drücke im Run-Mode DEF-S u. dann ENTER.

3) Zeile 231 lautet ausgeschrieben:

```
PRINT "2 Schlangen
konstant jeweils";
INT(A/2)+1;"Kunden"
```

4) Dieses Programm wurde mit einem PC-1500 Version A01 geschrieben.

Literaturnachweis:

Kütting, Herbert: Didaktik der Wahrscheinlichkeitsrechnung. Studienbücher Mathematik, Herder-Verlag, Freiburg/Brsg.

Churchman, C. West und Russel J. Ackoff, E. Leonard Arnoff: Operations Research. Oldenbourg-Verlag, München/ Wien.

PEEK und POKE

Weitere Möglichkeiten von PEEK und POKE (Teil 1)

Längere Zeit ärgerte ich mich darüber, daß beim PC-1500 manche Zeichen nicht direkt über die Tastatur aufgerufen werden können. Hierbei handelt es sich um die CHR-Codes &27, &5C, &7B, &7D, &7E, &7F. Zum einen ließen sich die genannten Zeichen ab und zu bei einer Displayanzeige verwenden, zum anderen könnten sie u.a. in Textprogrammen als 'Steuerbytes' eingesetzt werden (wie z.B. in 'TEXT PLUS', siehe Info).

Mittels den beiden Befehlen PEEK und POKE lassen sich die genannten Codes problemlos auf die Reservetasten 1-6 der Ebene I bringen. Im RUN oder PRO-MODE wird nach Löschen des Reservespeichers eingegeben:

```
POKE PEEK&7863*256+&56, 1, &27, 2, &5C, 3, &7B, 4, &7D, 5, &7E, 6, &7F  ENTER
```

Nach diesem Befehl ist die erste Reserveebene wie folgt belegt:

```
F1: CHR$ &27, F2: CHR$ &5C, F3: CHR$ &7B
F4: CHR$ &7D, F5: CHR$ &7E, F6: CHR$ &7F
```

Die genannten Codes können jetzt direkt über diese Funktionstasten aufgerufen werden.

Ein weiteres Ärgernis stellte für mich die Tatsache dar, daß das CE-150 nur ca. 10 cm. Papier einziehen kann. Auch dieses Ärgernis läßt sich mittels dem Befehl POKE leicht beseitigen.

Um das Papier auf der Y-Koordinate 'richtig' zu bearbeiten verwendet der PC-1500 einen systeminternen Koordinatenzähler, der in den beiden Adressen &79E4 und &79E5 untergebracht ist. Die Zählung dieses Abschnittzählers gestatten nur ein Einziehen über -LF 24 in CSIZE 2. Durch den Befehl POKE kann dieser Zähler zurückgesetzt werden. Hierbei verfährt man wie folgt:

```
POKE &79E4, 0, 0
```

Sofern mit diesem Befehl gearbeitet wird, ist darauf zu achten, daß er erst dann eingesetzt wird, nachdem 10 cm. Papier eingezogen wurden.
2 Beispiele:

```
10:TEXT :LF 40:LF -20:POKE &79E4, 00:LF -20
20:GRAPH :GLCURSOR (0,-1000):GLCURSOR (0,-500):POKE &79E4, 0, 0: GLCURSOR (0, 0)
```

Mittels PEEK und POKE können auch Programme die mit MERGE hinzuge-

Laden werden programmgesteuert gelöscht werden. Hierbei wird es notwendig, daß die Pointer für Programmende (enthalten in den Adressen &7867, &7868) und 'Mergebeginn' (enthalten in den Adressen &7869, &786A) umgesetzt werden.

Als kleine Tücke erweist sich hierbei allerdings, daß jeweils nach Pro.1 und Pro.2 ein &FF als Kennung für das Programmende steht. Somit würde bei der direkten Übertragung des Mergepointers auf den Programmendpointer quasi ein Byte 'verschenkt'. Den Inhalt des Low-Byte vom Mergepointer um eins zu verringern wäre zu riskant, da dieser Inhalt u.U. 0 sein kann!

Würde allerdings nur der Programmendpointer versetzt werden, wäre das Pro.1 nach dem Löschen von Pro. 2 nicht editierfähig. Aus diesem Grund muß auch der Mergepointer versetzt werden, sodaß er auf den Programmstart zeigt, also denselben Inhalt bekommt wie der Programmstartpointer (enthalten in den Adressen &7865, &7866).

Wenn Sie das mit MERGE hinzugeladene Programm um nachstehende 2 Zeilen erweitern, wird es bei Ansprache dieser Zeilen gelöscht, Pro. 1 ist dann auch wieder voll editierfähig.

```
10:E=PEEK &7869*256+PEEK &786A:E=E-1:D=INT (E/256):E=E-D*256:POKE &7867,D,E
20:POKE &7869,PEEK &7865,PEEK &7866:END
```

(Geschrieben mit 'TEXT PLUS')

Weitere Möglichkeiten von PEEK und POKE (Teil 2)

Einige PC-1500 Anwender würden gerne nicht nur Maschinenprogramme, sondern auch BASIC Programme vor Veränderungen im PRO-Mode schützen und sie jederzeit aufrufbar halten, sodaß Bandoperationen vermieden werden können.

Heute möchte ich anhand eines kleinen Beispiels aufzeigen, daß das Schützen von BASIC-Programmen mittels den Befehlen PEEK und POKE ohne weiteres zu bewerkstelligen ist.

Zunächst benötigt man eine kleine Modifikation der im letzten Teil vorgestellten Routine zum Löschen von mit MERGE hinzugeladenen Programmen:

```
10:"M"E=PEEK &7869*256+PEEK &786A:E=E-1:D=INT (E/256):E=E-D*256
20:POKE (STATUS 2-STATUS 1-2),D,E:POKE &7867,D,E:POKE &7869,PEEK &7865,PEEK
   &7866
```

Dieses kleine Programm setzt also auf die Adressen STA.2-STA.1-2 und STA.2-STA.1-1 dieselben Zahlen wie auf den Programmendpointer! Dies wird benötigt, um das geschützte Programm jederzeit wieder 'verstecken' zu können.

Als Beispielprogramm habe ich mir ein kleines Programm ausgedacht, das es erlaubt, mittels der ↓ Taste, von Hand beliebig viel Papier einzuziehen, quasi als Gegenstück zu der ↑ Taste.

```
10:' 'TEXT :A$='':A$=INKEY$ :A=ASC A$
20:IF A=&APOKE &79E4,0,0:LF -1
30:IF A=&FPOKE &7865,PEEK (STATUS 2-STATUS 1-2),PEEK (STATUS 2-STATUS 1-1):
   END
40:GOTO 10
```

In Zeile 30 wird also der Programmstartpointer so umgesetzt, daß er auf die erste ansprechbare Speicherzelle nach diesem Programm zeigt, was von dem Programm 'M' (für jedes Programm) errechnet wird.

Weitere Möglichkeiten von PEEK und POKE (Teil 3; 'KEEP')

Ab und zu passiert es auch dem vorsichtigsten Programmierer, daß er versehentlich ein Programm mit NEW löscht. Im Regelfall ist dies weniger schlimm, man hat ja das Programm auf Cassette gespeichert. Beim Einlesen passiert dann zu allem Überfluß ab und zu ein weiteres Malheur: ERROR 44

Solche Szenen können selbst alten Routiniers an die Nerven gehen. Eine Abhilfe bietet hier zwar der Tool 1 der Fa. Tramsoft. Allerdings hat der Tool auch zwei Nachteile: Zum einen ist er nicht billig, zum anderen verliert der PC-1500 sein Taschenformat.

In diesem Teil möchte ich das Programm 'KEEP' vorstellen. Es ermöglicht es, ein versehentlich mit NEW gelöschtes Programm wieder sichtbar zu machen bzw. nach ERROR 44 den noch gelesenen Teil editieren zu können. 'KEEP' steht in BASIC und belegt 179 Bytes.

```
1:'K'C=STATUS 2-STATUS 1:A=PEEK (C-2)*256+PEEK (C-1):POKE A,0
2:IF PEEK (A+2+PEEK (A+2))<>13POKE A,&FF:GOTO 4
3:A=A+3+PEEK (A+2):IF PEEK A<>&FFTHEN 2
4:B=INT (A/256):A=A-B*256:POKE &7867,B,A:POKE &7865,PEEK (C-2),PEEK (C-1):
  CLEAR: END
```

Nachdem das Programm eingegeben wurde, wird es auf Cassette gespeichert. Im Anschluß daran wird 'KEEP' mit NEW gelöscht.

Um 'KEEP' zu implementieren wird wieder die in Teil 2 vorgestellte Modifikation des Programms 'M' benötigt. Die einzelnen Schritte sind:

- NEW STATUS 2+2 ENTER
- CLOAD 'KEEP'
- MERGE 'M'
- Starten des Programms 'M' mit DEF 'M'
- PEEK &7865 ENTER, angezeigte Zahl = >Zahl 1<
- PEEK &7866 ENTER, angezeigte Zahl = >Zahl 2<
- Belegung einer Reservetaste: F1:POKE &7865,>Zahl 1<,>Zahl 2<@
- NEW STATUS 2-1 ENTER

'KEEP' ist jetzt vor Änderungen im PRO-MODE geschützt. Der Aufruf geschieht über die Schritte: F1, DEF 'K'.

Nach einem versehentlichen NEW oder nach ERROR 44 wird 'KEEP' wie beschrieben aufgerufen. Für ein BASIC-Programm ist 'KEEP' sehr schnell. Um ca.10KB zu 'retten' benötigt es ca. 15 sec.. Nachdem das 'BUSY'-Symbol verschwunden ist, kann das gerettete Programm voll editiert werden, 'KEEP' bleibt 'versteckt'.

Die Arbeitsweise von 'KEEP':

Zeile 1: Berechnet erstes Byte des zu rettenden Programms und poked hierauf 0, da an dieser Stelle nach NEW ein &FF steht. (Sollte die erste Zeilennummer >255 sein, ändert sich diese Nummer, alles andere bleibt gleich.

Zeile 2: Prüft ob eine Zeile mit 13 abschließt. Wenn nicht, (ERROR 44)! wird auf das erste Byte dieser Zeile ein &FF (Programmende) gepoked.

Zeile 3: Errechnet Anfangsbyte von Programmzeilen und sucht Programmende (&FF).

Zeile 4: Setzt Programmendpointer so, daß gerettetes Programm sichtbar ist; versteckt 'KEEP'.

Selbstverständlich kann 'KEEP' durch NEW 0 gelöscht werden.

Weitere Möglichkeiten von PEEK und POKE (Teil 4; 'LISTPRO 1')

Manch einem PC-1500 Anwender ist das Listing seines Computers ein Dorn im Auge. Und wegen des Listings allein lohnt sich die Anschaffung eines externen Druckers kaum. Mittels dem Befehl PEEK läßt sich ein sehr komfortables Listingsprogramm erstellen, das ich in diesem und dem folgenden Teil vorstellen möchte. In diesem Teil soll die 'Grundversion' von 'LIST PRO' vorgestellt werden, im folgenden Teil wird dann die Ausbauversion vorgestellt, die ein strukturiertes Listing erzeugt.

Das Programm soll zwei Ansprüchen genügen:

- Um auch bei kleineren Speicherversionen eingesetzt werden zu können, soll es weniger als 1 KB Speicherplatz belegen.
- Die Arbeitszeit soll noch annehmbar sein.

Eine Schwierigkeit hierbei ist, daß die BASIC-words an verschiedenen Speicherstellen codiert werden. Bei der Version PU=0 (angeschlossenes CE-150, kein CE-158) stehen die Druckerbefehle im Bereich &B055-&B0E6, die Cassettenbefehle im Bereich &B855-B881, die restlichen BASIC-words werden sowohl bei der Version PU=0, als auch bei der Version PU=1 im Bereich &C055-C34B codiert. Danach stehen noch die Anzeigen:

- NEW0? :CHECK
- BREAK IN
- ERROR IN

Allerdings sind die Befehle in den genannten Bereichen nicht nach ihren Tokens geordnet. D.h. würde man sie jedesmal mit PEEK suchen, wäre die Arbeitszeit des Listingsprogramms kaum erträglich. Aus diesem Grund habe ich eine Tabelle mit folgendem Aufbau angelegt:

Die Tokens sind geordnet: Zunächst stehen die 7 Tokens der Form &E6II, dann der Token &E79A (RMT), dann die 20 Tokens der Form &F0II, zuletzt diejenigen der Form &F1II.

Aus diesem Grund ist es ausreichend, nur den 2.Teil der Tokens in die Tabelle zu schreiben. Die zweite Zahl jeder Tabellenzeile beinhaltet die Länge des jeweiligen BASIC-words, in der dritten und vierten Zahl jeder Zeile ist die jeweilige Anfangsadresse im ROM dieses BASIC-words angegeben. Die Tabelle belegt insgesamt 416 Bytes und kann an jeder beliebigen Stelle im Speicher stehen.

'LISTPRO 1' wird genauso implementiert wie 'KEEP' (siehe Teil 3). 'LISTPRO 1' belegt 788 Bytes.

Die Arbeitsweise von 'LISTPRO 1':

Zeilen 20-140: Unterprogramme zum Ausdruck, Zeile 80 manipuliert den Y-Koordinatenzähler (siehe Teil 1)

Zeile 160: Hier wird die Tabelle eingelesen; ab der Adresse &7650 !
D.h. die Tabelle liegt im Bereich der Standardvariablen E$-Z$ und auf einem Teil der LCD-Anzeige. Aus diesem Grund entsteht beim Einlesen ein eigentümliches Punktemuster auf dem Display, das erhalten bleibt, bis 'LISTPRO 1' seine Arbeit beendet hat. Selbstverständlich kann die Tabelle auf jeder beliebigen Stelle im Rechner stehen. In diesem Fall wird das Einlesen überflüssig. Die Variable T in Zeile 180 muß in diesem Fall so geändert werden, daß sie die Startadresse der Tabelle enthält.

Zeilen 180-300 lesen die jeweiligen CHR-Codes aus dem Speicher.

Zeilen 320-440 suchen die BASIC-words anhand der Tabelle und weisen sie der Variablen B$ zu.

Zeile 460 beendet und versteckt 'LISTPRO 1'

Achtung:'LISTPRO 1' ist nur bei der Version PV=0 (Kein CE-158) Lauffähig. Sofern mit einem zweiten Zeichensatz gearbeitet wird, darf der größte CHR-Code nicht größer als &E5 (=229) sein, da größere Codes als BASIC-Tokens interpretiert werden (vgl. Zeile 240)

Der Aufruf von 'LISTPRO 1' erfolgt über die 2 Schritte: F1 (belegt, wie in Teil 3 beschrieben), DEF 'L'. Es meldet sich mit der Frage 'TABELLE ?' ENTER bewirkt das Einlesen der Tabelle.

```
 20:GLCURSOR (0,-980):LINE -(220,-980):GLCURSOR (0,-1000):SORGN :X=202:Y=0:
    GLCURSOR (X,Y):RETURN
 40:IF X>2GOSUB 80:RETURN
 60:GOSUB 20:RETURN
 80:X=X-20:Y=0:GLCURSOR (X,-500):POKE &79E4,0,0:GLCURSOR (X,0):RETURN
100:IF Y-L*12)-960THEN 140
120:GOSUB 40:Y=-72:GLCURSOR (X,Y)
140:LPRINT B$:Y=Y-L*12:B$="":RETURN
160:"L"GRAPH :ROTATE 1:CSIZE 2:PRINT "TABELLE ?":CLOAD M"TABELLE";&7650
180:C=STATUS 2-STATUS 1:A=PEEK (C-2)*256+PEEK (C-1):T=&7650:X=202:Y=0:SORGN :
    GLCURSOR (X,Y)
```

```
200:B=PEEK A*256+PEEK (A+1):B$=STR$ B+":":IF B>65297THEN 460
220:L=LEN B$:IF L<6LET B$=" "+B$:GOTO 220
240:GOSUB 100:FOR S=A+3TO A+1+PEEK (A+2):P=PEEK S:IF P>&E5THEN 280
260:B$=CHR$ P:L=1:GOSUB 100:GOTO 300
280:GOSUB 320:GOSUB 100
300:NEXT S:GOSUB 40:GLCURSOR (X,Y):A=A+3+PEEK (A+2):GOTO 200
320:IF P=&E6LET U=T:GOTO 400
340:IF P=&E7LET U=T+28:GOTO 400
360:IF P=&F0LET U=T+32:GOTO 400
380:U=T+112
400:W=PEEK (S+1):IF W<>PEEK ULET U=U+4:GOTO 400
420:O=PEEK (U+2)*256+PEEK (U+3):FOR R=1TO PEEK (U+1):B$=B$+CHR$ PEEK O:O=O+1:
    NEXT R
440:B$=B$+" ":L=LEN B$:S=S+1:RETURN
460:GOSUB 20:TEXT :LF 2:POKE &7865,PEEK (C-2),PEEK (C-1):CLEAR :END
```

TABELLE

```
5000: 80 05 B0 5F
5004: 81 05 B0 69
5008: 82 08 B0 73
500C: 83 07 B0 80
5010: 84 05 B0 C6
5014: 85 06 B0 BB
5018: 86 04 B0 E1
501C: A9 03 B8 7D
5020: B4 06 C0 AA
5024: B5 05 C3 29
5028: B8 03 C0 BF
502C: B9 05 B8 5F
5030: BF 05 B8 73
5034: 90 04 C1 86
5038: 91 05 C1 62
503C: 93 07 C1 4D
5040: 95 05 B8 69
5044: 97 05 C2 09
5048: 9F 06 C1 42
504C: B2 05 B8 55
5050: B5 05 B0 55
5054: B6 02 B0 8C
5058: B7 04 B0 93
505C: B8 05 B0 9C
5060: B9 06 B0 A6
5064: BA 05 B0 B1
5068: BB 03 B0 D0
506C: BC 04 B0 D8
5070: 50 03 C0 5F
5074: 51 02 C1 F2
5078: 58 03 C1 C1
507C: 5B 04 C3 06
5080: 5C 06 C1 7B
5084: 5D 02 C2 13
5088: 60 03 C0 87
508C: 61 04 C2 D8
5090: 62 03 C3 3E
5094: 63 04 C0 CF
5098: 64 03 C1 A6
509C: 65 03 C0 F4
50A0: 66 03 C0 FC
50A4: 67 06 C2 E1
50A8: 68 05 C2 40
50AC: 6B 03 C2 C0
50B0: 6D 03 C1 DB
50B4: 6E 05 C2 1A
50B8: 6F 04 C2 24
50BC: 70 03 C0 67
50C0: 71 03 C1 73
50C4: 72 06 C2 99
50C8: 73 03 C0 77
50CC: 74 03 C0 7F
50D0: 75 03 C0 6F
50D4: 76 02 C1 97
50D8: 77 03 C1 8F
50DC: 78 03 C1 15
50E0: 79 03 C2 D0
50E4: 7A 05 C1 AE
50E8: 7B 04 C1 C9
50EC: 7C 03 C2 86
50F0: 7D 03 C2 C8
50F4: 7E 03 C0 C7
50F8: 7F 03 C2 FE
50FC: 80 05 C0 55
5100: 81 04 C0 8F
5104: 82 04 C0 98
5108: 83 04 C0 A1
510C: 86 04 C1 59
5110: 87 05 C0 B5
5114: 8A 04 C0 D8
5118: 8B 03 C0 E1
511C: 8C 06 C0 E9
5120: 8D 04 C1 04
5124: 8E 03 C1 0D
5128: 92 04 C1 2F
512C: 94 05 C1 38
5130: 96 02 C1 6C
5134: 98 03 C1 9E
5138: 99 06 C2 66
513C: 9A 04 C1 D2
5140: 9B 03 C1 E3
5144: 9C 02 C1 EB
5148: 9D 03 C1 F9
514C: 9E 03 C2 01
5150: A0 05 C2 2D
5154: A1 04 C2 37
5158: A2 05 C2 4A
515C: A4 03 C2 5E
5160: A5 03 C1 27
5164: A6 04 C2 71
5168: A7 07 C2 7A
516C: A8 06 C2 8E
5170: AA 06 C2 A4
5174: AB 03 C2 AF
5178: AC 04 C2 B7
517C: AD 04 C2 EC
5180: AE 04 C2 F5
5184: AF 04 C3 0F
5188: B0 05 C3 18
518C: B1 02 C3 22
5190: B3 04 C3 46
5194: B4 05 C1 1D
5198: B5 04 C1 B8
519C: B6 06 C3 33
```

"LISTRO 2"

```
**: Beginnt neuen Papierstreifen :**
  20:GLCURSOR (0,-980):LINE -(220,-980):GLCURSOR (0,-1000):SORGN :X=202:Y=0:
     GLCURSOR (X,Y):RETURN
**: Übergibt nach 40 wenn Kommentar geschrieben :**
  30:"1"GOSUB 40:GOTO "2"
**: Übergibt auf 80 bzw. 20 :**
  40:
    IF X>2GOSUB 80:RETURN
  60:GOSUB 20:RETURN
**: Zieht Papier ein (manipuliert Y-Koordinatenzähler :**
  80:X=X-20:Y=0:GLCURSOR (X,-500):POKE &79E4,0,0:GLCURSOR (X,0):RETURN
**: Gsf. Übergabe nach 40, Druck eines Strings der Form B$ :**
 100:
    IF Y-L*12>-960THEN 140
 120:GOSUB 40:Y=-72:GLCURSOR (X,Y)
 140:LPRINT B$:Y=Y-L*12:B$="":RETURN
**: Aufruf, Eingabe Zeilenbereich :**
 150:"L":PRINT "BEREICH":INPUT H,I:ON ERROR GOTO "2"
**: Tabelle ?, A=Anfangsbyte des Programms, I=I-Koord.,Y=Y-Koord. :**
 160:GRAPH :ROTATE 1:CSIZE 2:PRINT "TABELLE ?":CLOAD M"TABELLE";&7650
 180:C=STATUS 2-STATUS 1:A=PEEK (C-2)*256+PEEK (C-1):T=&7650:X=202:Y=0:SORGN :
     GLCURSOR (X,Y)
**: Sucht zu listenden Bereich, B=Zeilennummer :**
 200:B=PEEK A*256+PEEK (A+1):B$=STR$ B+":":
    IF B<HLET A=A+3+PEEK (A+2):GOTO 200
 205:
    IF B>ITHEN 460
**: Übergibt auf Programm 'KOM' :**
 210:GOTO "KOM"
**: Zeilennummer = B$, 'Formatieren' :**
 220:"2"L=LEN B$:
    IF L<6LET B$=" "+B$:GOTO "2"
**: Übergabe auf 100, ggf. Übergabe auf 200 :**
 240:GOSUB 100:
  FOR S=A+3TO A+1+PEEK (A+2):P=PEEK S:
    IF P>&E5THEN 280
**: CHR's aus Speicher lesen, Übergabe auf 100 ggf. 320 :**
 260:B$=CHR$ P:L=1:GOSUB 100:GOTO 300
 280:GOSUB 320:GOSUB 100
**: Nächste Zeilennummer, Übergabe auf 200 :**
 300:
  NEXT S:GOSUB 40:GLCURSOR (X,Y):A=A+3+PEEK (A+2):GOTO 200
**: Token: &E6II,&E7II,&F0II,&F1II ?, Übergabe auf 400 :**
 320:
    IF P=&E6LET U=T:GOTO 400
 340:
    IF P=&E7LET U=T+28:GOTO 400
 360:
    IF P=&F0LET U=T+32:GOTO 400
 380:U=T+112
**: Sucht Standort des Basic words anhand der Tabelle :**
 400:W=PEEK (S+1):
    IF W<>PEEK ULET U=U+4:GOTO 400
**: B$=>Basic word<, Prüfen ob FOR, NEXT, IF :**
 420:O=PEEK (U+2)*256+PEEK (U+3):
```

```
   FOR R=1TO PEEK (U+1):B$=B$+CHR$ PEEK 0:0=0+1:
   NEXT R
  440:B$=B$+" ":L=LEN B$:S=S+1:
     IF W=&A5OR W=&9AOR W=&96GOSUB 40:GOTO 450
  445:RETURN
**: Wenn FOR, NEXT oder IF dann Übergabe auf 40 :**
  450:
     IF W=&96LET Y=-60:GOTO 458
  455:Y=-36
  458:GLCURSOR (X,Y):RETURN
**: Beendet und versteckt das Programm :**
  460:GOSUB 20:TEXT :LF 2:POKE &7865,PEEK (C-2),PEEK (C-1):CLEAR :END
```

Kommentar zu 'LISTPRO 2'

```
 20:LPRINT "**: Beginnt neuen Papierstreifen :**":RETURN
 30:LPRINT "**: Übergibt nach 40 wenn Kommentar geschrieben :**":RETURN
 40:LPRINT "**: Übergibt auf 80 bzw. 20 :**":RETURN
 80:LPRINT "**: Zieht Papier ein (manipuliert Y-Koordinatenzähler :**":
    RETURN
100:LPRINT "**: Gsf. Übergabe nach 40, Druck eines Strings der Form B$ :**":
    RETURN
150:LPRINT "**: Aufruf, Eingabe Zeilenbereich :**":RETURN
160:LPRINT "**: Tabelle ?, A=Anfangsbyte des Programms, I=I-Koord., Y=Y-Koord.
    :**":RETURN
200:LPRINT "**: Sucht zu listenden Bereich, B=Zeilennummer :**":RETURN
210:LPRINT "**: Übergibt auf Programm 'KOM' :**":RETURN
220:LPRINT "**: Zeilennummer = B$, 'Formatieren' :**":RETURN
240:LPRINT "**: Übergabe auf 100, gsf. Übergabe auf 280 :**":RETURN
260:LPRINT "**: CHR's aus Speicher lesen, Übergabe auf 100 gsf. 320 :**":
    RETURN
300:LPRINT "**: Nächste Zeilennummer, Übergabe auf 200 :**":RETURN
320:LPRINT "**: Token: &E6II,&E7II,&F0II,&F1II ?, Übergabe auf 400 :**":
    RETURN
400:LPRINT "**: Sucht Standort des Basic words anhand der Tabelle :**":
    RETURN
420:LPRINT "**: B$=>Basic word<, Prüfen ob FOR, NEXT, IF :**":RETURN
450:LPRINT "**: Wenn FOR, NEXT oder IF dann Übergabe auf 40 :**":RETURN
460:LPRINT "**: Beendet und versteckt das Programm :**":RETURN
500:"KOM":GOSUB 8:GOTO "1"
```

Weitere Möglichkeiten von PEEK und POKE (Teil 5; 'LISTPRO 2')

Zwar konnte durch 'LISTPRO 1' die Lesbarkeit von Programmen schon verbessert werden, Programmstrukturen bleiben aber auch mit 'LISTPRO 1' im Verborgenen. U.a. bei längeren Programmen wäre es auch schön, wenn aus dem Listing hervorginge, mit welcher Aufgabe die einzelnen Programmteile vertraut sind. Durch REM können solche Überschriften zwar in das Listing gebracht werden, dies wirkt sich allerdings nachteilig auf die Geschwindigkeit des Programms aus. 'LISTPRO 2' ermöglicht es, beliebige Programmzeilen mit Überschriften zu versehen. Nachdem das Listing erzeugt ist, können diese Überschriften durch das Betätigen nur einer Funktionstaste gelöscht werden.
Ein kommentiertes Listing von 'LISTPRO 2' finden Sie auf der nachfolgenden Seite. Darunter ist als Beispiel das Kommentarprogramm zu 'LISTPRO 2' abgebildet. Das Kommentarprogramm hat für jedes zu listende Programm dieselbe allgemeine Form:
- Zeilennummer derjenigen Zeile über der der Kommentar stehen soll
- LPRINT '>Kommentar<':RETURN
Die Zeichen '**:' haben keine Bedeutung, sie wurden von mir aus Übersichtsgründen eingesetzt.

Nach den Kommentarzeilen muß eine Zeile mit der Form:
'KOM':GOSUB 8:GOTO '1' stehen.

Das Kommentarprogramm wird mit MERGE zu dem zu listenden Programm hinzugeladen, erst dann wird 'LISTPRO 2', wie in Teil 4 beschrieben aufgerufen.

Es meldet sich mit der Frage 'BEREICH'. Hier wird ein ENTER durchgeführt. Danach erscheint ein ?. Eingabe : Zeile ab der gelistet werden soll. Nach ENTER erscheint wieder ein ?: Eingabe der Zeile bis zu der einschließlich gelistet werden soll ENTER. Danach erscheint die Anzeige 'TABELLE ?'. ENTER führt zum Einlesen der Tabelle.

Von 'LISTPRO 2' werden FOR NEXT Schleifen abgesetzt, desgleichen IF-Bedingungen. Wird kein Kommentar gewünscht, müssen die Zeilen 30 und 210 aus dem Programm genommen werden. In dem zu listenden Programm darf der Markenname 'KOM' nicht verwendet werden.
Da das Kommentarprogramm mit MERGE hinzugeladen wird, kann es mit der in Teil 1 vorgestellten Routine gelöscht werden. Man stellt der Routine einen Markennamen voran, mit GOTO 'Name' wird dann das Kommentarprogramm gelöscht. Dieser Befehl kann selbstverständlich auf eine Reservetaste gelegt werden. 'LISTPRO 2' kann mit NEW 8 gelöscht werden.

Weitere Möglichkeiten von PEEK und POKE (Teil 6)

Sofern mit Maschinensprache programmiert wird, ist es oft so, daß kleine Maschinenroutinen von einem BASIC-Programm gesteuert werden. Sofern so gearbeitet wird, ist es allerdings sehr mühselig, bis alle Programme im Rechner sind. Zunächst muß der Speicherplatz für die Maschinenprogramme reserviert werden, erst dann werden die Maschinenprogramme eingegeben, danach erst das BASIC-Steuerprogramm.

Einfacher wäre es, wenn das bzw. die Maschinenprogramme direkt in das Steuerprogramm eingebunden wären, und alles zusammen auf einmal in den PC-1500 geladen würde.

Auch ein solches Programm kann mit den Befehlen PEEK und POKE erstellt werden.

Als Beispiel nehme ich ein Maschinenprogramm, das einen Speicherbereich nach 'oben' verschiebt. Dieses kleine Transferprogramm soll in ein Steuerprogramm eingebunden werden, das ein abgewandeltes DELETE ermöglicht. Wurde zu einem Programm ein weiteres Programm mit MERGE hinzugeladen, soll das mit MERGE hinzugeladenen Programm erhalten bleiben, Programm 1 soll gelöscht werden.

Das Programm 'N' führt dieses Löschen durch. Zunächst müssen in der ersten Programmzeilen soviel Zeichen gesetzt werden, wie von dem Maschinenprogramm belegt werden. Nun kann in einem 2. Programm das Maschinenprogramm niedergeschrieben werden, welches mit POKE in das Programm 'N' übertragen wird. Im Programm 'U' ist dies der Verschiebungsteil, nicht enthalten sind die Ladebefehle für I, Y und U-Register. Das Programm 'U' wird mit MERGE zum Programm 'N' hinzugeladen und mit DEF U gestartet. Danach ist in Zeile 10 von 'N' ab '?'Nr. 12 das Transferprogramm niedergelegt. 'U' hat sich selbst gelöscht. Das Programm 'N' kann jetzt ganz normal gesaved werden, das Maschinenprogramm wird mit auf Cassette gespeichert. Im Anschluß daran wird 'N' genauso wie 'KEEP' implementiert (Mit dem Hilfsprogramm 'M').

Soll jetzt ein Programm gelöscht werden, das mit MERGE hinzugeladene jedoch erhalten bleiben wird 'N' durch die Schritte F1, DEF N aufgerufen. Prog. 1 wird gelöscht, Prog. 2 bleibt erhalten. Danach kann zu dem erhaltenen Programm selbstverständlich wieder ein Programm mit MERGE hinzugeladen werden. Wird jetzt 'N' aufgerufen wird wiederum das erste Programm im Speicher gelöscht.

```
10:??????????????
   ??????
20:A=INT (B/256):
   B=B-A*256:
   RETURN
30:"N"C=STATUS 2-
   STATUS 1-2:I=
   PEEK C*256+
   PEEK (C+1):D=
   PEEK &7869*256
   +PEEK &786A
40:E=PEEK &7867*2
   56+PEEK &7868:
   F=E-D
50:G=STATUS 2-
   STATUS 1+3:B=D
   :GOSUB 20:POKE
   G, &48, A, &4A, B:
   H=G+4
60:B=I:GOSUB 20:
   POKE H, &58, A, &
   5A, B:H=H+4
70:B=F:GOSUB 20:
   POKE H, &68, A, &
   6A, B:CALL (
   STATUS 2-
   STATUS 1+3)
80:POKE &7869,
   PEEK C, PEEK (C
   +1):B=I+F:
   GOSUB 20:POKE
   &7865, PEEK C,
   PEEK (C+1), A, B
   :END

STATUS 1
                325
```

```
10:"U"Y=STATUS 2-
   STATUS 1:Y=Y+1
   5
20:POKE Y, &F5, &88
   , &03, &FD, &62, &
   93, &07, &9A
30:E=PEEK &7869*2
   56+PEEK &786A:
   E=E-1:D=INT (E
   /256):E=E-D*25
   6:POKE &7867, D
   , E
40:POKE &7869,
   PEEK &7865,
   PEEK &7866

STATUS 1
                155
```

Programm "V" (bindet Mapro Teil 2 in "N" ein)

```
****: Y=STARTADRESSE TEIL 2 VON MAPRO :****
   10:"U"Y=STATUS 2-STATUS 1:Y=Y+15
****: POKED TEIL 2 VON MAPRO (VERSCH.↑) AUF Y FF. :****
   20:POKE Y, &F5, &88, &03, &FD, &62, &93, &07, &9A
****: LÖSCHT 'U' :****
   30:E=PEEK &7869*256+PEEK &786A:E=E-1:D=INT (E/256):E=E-D*256:POKE &7867, D, E
   40:POKE &7869, PEEK &7865, PEEK &7866
```

Programm "N" (Löscht Prog.1, wenn Prog.2 mit MERGE geladen wurde)

```
****: FREIER PLATZ FÜR MAPRO (20 BYTES)~ :****
   10:??????????????????
****: INTEGERZAHL -- 2 BYTEZAHL (A=HIGH, B=LOW BYTE) :****
   20:A=INT (B/256):B=B-A*256:RETURN
****: I=1. BYTE PROG.1, D=1. BYTE PROG.2 (MERGE) :****
   30:"N"C=STATUS 2-STATUS 1-2:I=PEEK C*256+PEEK (C+1):D=PEEK &7869*256+PEEK &7
      86A
****: E=LETZTES BYTE PROG.2, F=LÄNGE PROG.2 :****
   40:E=PEEK &7867*256+PEEK &7868:F=E-D
****: SPEIST I-REG. MIT D :****
   50:G=STATUS 2-STATUS 1+3:B=D:GOSUB 20:POKE G, &48, A, &4A, B:H=G+4
****: SPEIST Y-REG. MIT I :****
   60:B=I:GOSUB 20:POKE H, &58, A, &5A, B:H=H+4
****: SPEIST U-REG. MIT F/RUFT MAPRO AUF :****
   70:B=F:GOSUB 20:POKE H, &68, A, &6A, B:CALL (STATUS 2-STATUS 1+3)
****: SETZT START-MERGE-UND ENDPOINTER UM :****
   80:POKE &7869, PEEK C, PEEK (C+1):B=I+F:GOSUB 20:POKE &7865, PEEK C, PEEK (C+1),
      A, B:END
```

Weitere Möglichkeiten von PEEK und POKE (Teil 7; 2.Reservespeicher)

Sehr erfreulich ist, daß sich beim PC-1500 18 Tasten frei definieren lassen. Manch ein Anwender hat allerdings mehrere Programme im Rechner, die durch di Reservetasten gesteuert werden. In diesem Fall reichen die 18 Tasten evt. nicht aus. Mittels den beiden Befehlen PEEK und POKE kann ein zweiter Reservebereich angelegt werden, der unabhängig von dem Standardreservebereich ist. Man verfährt folgendermaßen:

- NEW 0 ENTER (dies ist wichtig, da der 2. Reservebereich unmittelbar nach dem ersten stehen soll und vor Überschreiben durch BASIC-Programme geschützt werden muß.
- NEW STATUS 2+256 ENTER (=Schützen von Bereich 2)
- PEEK &7863 ENTER: Anzeige = >Zahl 1< (notieren). Zahl 1=High Byte der Startadresse des Standardreservespeichers (Low Byte ist immer &00)
- >Zahl 2<=>Zahl 1<+1 (Zahl 2 = High Byte von Bereich 2)

Jetzt wird die Reservetaste des Standardbereichs F1 wie folgt belegt:
- F1:POKE &7863,>Zahl 2<0 (Durch diese Taste kann in Bereich 2 umgeschaltet werden) Nachdem dies geschehen ist, wird in Bereich 2 die Taste F1 belegt:
- F1:POKE &7863,>Zahl 1<0 (Mit dieser Taste wird von Bereich 2 in den Standardreservebereich zurückgeschaltet.

Bereich 1 und Bereich 2 sind unabhängig voneinander. D.h. wird einer der Bereiche mit NEW im Reservemode gelöscht, bleibt der andere Bereich erhalten.

Durch den Befehl NEW 0 bei aktivem Standardreservebereich kann der selbst angelegte 2. Bereich im PRO-Mode gelöscht werden.

Weitere Möglichkeiten von PEEK und POKE (Teil 8; 'Speicherbereiche')

Auf dem Markt werden für den PC-1500 diverse Speicherausbaumöglichkeiten (max. 28 KB BASIC-Speicher) angeboten. Hierdurch wird es dem Anwender möglich, mehrere Programme im Rechner zu haben. Der PC-1500 bietet diese Möglichkeit durch den Befehl MERGE. Werden mehrere Programme mit diesem Befehl in den Rechner geladen, muß der Anwender allerdings gegen einige Schwierigkeiten kämpfen:

- Es dürfen niemals dieselben Variablennamen benutzt werden.
- CLEAR in nur einem Programm löscht sämtliche Variablen (die man oftmals noch brauchen würde).
- NEW löscht den gesamten Speicher (man will aber nur 1 Programm löschen).

Diese und noch manch andere Schwierigkeiten können ganz schön an die Nerven gehen. Auch hier kann durch die beiden Befehle PEEK und POKE Abhilfe geschaffen werden. Durch das Programm 'Bereiche' wird es möglich, den Speicher des PC-1500 in max. 10 voneinander unabhängige Speicher aufzuteilen ! D.h.:

- In jedem Bereich können dieselben Variablen definiert werden. So kann z.B in Bereich 1 und in Bereich 5 das Variablenfeld A(*) über DIM angelegt und mit verschiedenen Variablen belegt werden. Es kommt nicht zu ERROR 5
- NEW löscht nur das Programm in dem gerade aktiven Bereich. Die Programme in den anderen Bereichen bleiben mitsamt den definierten Variablen erhalten. Es werden lediglich die Standardvariablen gelöscht. Daher ist es zu empfehlen, die Standardvariablen als 'Laufvariablen' zu benutzen.
- Auch CLEAR löscht nur die Standardvariablen und die dimensionierten Variablen des aktiven Bereichs. Alle anderen bleiben erhalten.
- Jeder Bereich ist voll editierbar. Die Statusfunktionen geben Auskunft über den noch zu Verfügung stehenden Speicherplatz in dem aktiven Bereich.

Somit hat man quasi bis zu 10 Computer in einem !

Bevor das Programm 'Bereiche' gestartet wird, muß die 'fast schon übliche Reservetaste F1' belegt werden:

```
PEEK &7865 = >Zahl 1<; PEEK &7866 = >Zahl 2<
F1:POKE &7865,>Zahl 1<,>Zahl 2<@
```

Das Programm Bereiche wird allerdings nicht durch NEW STA.2-1 versteckt. Es wird auch nicht mit dem Hilfsprogramm 'M' bearbeitet.

Das Programm 'Bereiche' belegt 849 Bytes und verfügt über 3 Modi:

- Definieren der Bereiche (DEF D)
- Umschalten zwischen den Bereichen (DEF B)
- Löschen der angelegten Bereiche (DEF L)

Definieren der Bereiche

Nach DEF D meldet sich der Rechner mit der Frage:

- BEREICHE ?. Hier wird die Anzahl der zu definierenden Bereiche eingegeben. Diese Zahl darf max. 10 sein. Ist die eingegebene Zahl >10, wird sie nicht akzeptiert. (Wird dieser Programmteil aufgerufen wenn schon Bereiche angelegt sind, erscheint das >-Symbol). Als nächste Anzeige erfolgt:

- NOCH X SEITEN. Eine 'Seite' umfaßt 256 Bytes. Es steht zur Definition der einzelnen Bereiche die angezeigte Anzahl an Speicherseiten zur Verfügung. Die nächste Abfrage erfolgt nach ENTER:

- BER. X SEITEN ? Hier wird eingegeben, wieviel Speicherplatz der Bereich X umfassen soll. Der Speicherplatz ist: Eingegeben Zahl*256-1. In diesem Bereich wird das Programm mit den Variablen untergebracht, d.h. der Speicherplatz, der von den Variablen belegt wird, muß mit eingegeben werden. Wird eine Seitenzahl eingegeben, die größer als die zur Verfügung stehende Seitenzahl ist, wird sie nicht akzeptiert.
  Nach der Eingabe wird ENTER durchgeführt.

Die eben beschriebene Anzeige und Abfrage erfolgt so oft, wie bei der Abfrage 'BEREICHE ?' eingegeben wurde. Nachdem alle Bereiche definiert sind, wird automatisch auf Bereich Nr. 1 umgeschaltet.

Umschalten zwischen den Bereichen

Bevor die Bereiche mit Programmen belegt werden, müssen sie mit NEW initialisiert werden. Hierbei muß jeder Bereich angewählt werden.

Bereich 1 ist schon angewählt und kann im PRO-Mode durch NEW initialisiert werden. Das Umschalten auf die anderen Bereiche geschieht denkbar einfach:

Betätigen der Funktionstaste F1, danach DEF B. Der Rechner meldet sich mit der Abfrage:

BEREICH NR ?'. Eingabe des gewünschten Bereichs mit anschließendem ENTER.

Die Umschaltzeit beträgt ca. 1 sec. Es ist empfehlenswert, zunächst alle Bereiche zu initialisieren und erst dann zu belegen. Es kann beliebig oft der Bereich gewechselt werden. Sofern die Variablen eines Bereichs nicht gelöscht wurden, bleiben sie erhalten und stehen somit bei neuerlicher Ansprache des betreffenden Bereichs zur Verfügung.

Wird eine Bereichszahl eingegeben die nicht existiert, wird sie nicht akzeptiert.

Achtung: Es darf erst angewählt werden, wenn schon definiert wurde, da ansonsten ein 'Speichersalat' entsteht, der sich im schwersten Fall nur durch ALL Reset beseitigen läßt.

Löschen der angelegten Bereiche

Durch die beiden Befehle: F1, DEF L lassen sich die angelegten Bereiche löschen (nur so). Werden diese Befehle ausgeführt, wenn noch nicht definiert wurde, geschieht gar nichts, es erscheint das >-Symbol.

Durch diese Befehle werden auch die Variablen und das Programm 'Bereiche' gelöscht.

Ausschalten des PC-1500

Beim Ausschalten des Rechners muß darauf geachtet werden, daß vor dem Abschalten unbedingt der letzte Bereich angewählt wird. Außerdem müssen beim Definieren die Bereiche so angelegt werden, daß alle zur Verfügung stehenden Speicherseiten ausgenutzt werden. Dies gilt auch für den Fall, daß Sie den PC-1500 ausgehen lassen wollen. Auch hier muß der letzte Speicherbereich aktiviert sein.

Übrigens: Wenn Sie den Druckerreset leid sind, können Sie eine Funktionstaste mit dem Befehl CALL &E33F@ belegen. Durch diese Taste kann der Rechner auch abgeschaltet werden. Beim Anschalten erfolgt kein Druckerreset. Es erscheint das BUSY-Symbol. Betätigen einer beliebigen Taste (außer SML, DEF, SHIFT) läßt das BUSY-Symbol verschwinden, der Rechner ist dann betriebsbereit.

Wichtige Hinweise:

Nach dem Wechseln des Bereichs werden die Standardvariablen B und C auf 0 gesetzt.
Programme die in selbst definierten Speicherbereichen stehen, können durch 'versteckte' BASIC-Programme (wie z.B das vorgestellte 'KEEP') nicht bearbeitet werden.
In den einzelnen Bereichen darf nicht mit MERGE gearbeitet werden.
In den einzelnen Bereichen darf nicht mit dem Befehl NEW X gearbeitet werden. Selbstverständlich können die Bereiche auch für Maschinenprogramme genutzt werden. Nach NEW gibt STATUS 2 die Startadresse des jeweiligen Bereichs an.

Programmerläuterungen:

Es ist sehr wichtig, daß in Zeile 1 genau 75* das '?' steht. Hier werden die für die einzelnen Bereiche notwendigen Systemadressen gespeichert (pro Bereich 7 Bytes.) Abgelegt sind dort die Inhalte der Adressen:

- &7865, &7866 (Programmstartpointer), dieser Inhalt kommt auch auf die Adressen &7869, &786A (Mergepointer), daher ist ein Arbeiten mit MERGE nicht möglich.
- &7867, &7868 (Programmendpointer). Wird bei Ansprache eines neuen Bereichs übernommen, dadurch ist das Editieren in den einzelnen Bereichen möglich.
- &7864 (High-Byte der Startadresse des Variablenspeichers +1. Low-Byte = 0). Daher müssen die Bereiche in 'Seiten' zu 256 Bytes angelegt werden. (Anmerkung: PEEK &7864*256 entspricht STA.3)
- &7899, &789A (Endadresse Variablenspeicher). Wird auch immer neu übernommen. Dadurch bleiben definierte Variablen erhalten.

Auf '?'-Nr.1 ist die Anzahl der angelegten Bereiche niedergelegt. Auf '?'-Nr.2 ist der gerade aktivierte Bereich vermerkt. Auf '?'-Nr.3-72 liegen (wie beschrieben) die notwendigen Inhalte für die Systemadressen. Auf den letzten 3 '?' ist der 'Normalzustand' der Adressen &7864, &7899 und &789A gespeichert. (Daher ist ein CLEAR vor dem Anlegen der Bereiche zu empfehlen).

Solange mit einzelnen Bereichen gearbeitet wird, darf auf gar keinen Fall an der Zeile 1 etwas geändert werden !!! Ansonsten wäre sehr schnell ein ALL Reset fällig.

Selbstverständlich kann statt '?' ein anderes Platzhaltersymbol gewählt werden. In diesem Fall muß dessen ASC II Code an Stelle der 63 in Zeile 5 stehen.

```
 1:??????????????
   ??????????????
   ??????????????
   ??????????????
   ??????????????
   ?????
 2:H=INT (L/256):
   L=L-H*256:
   RETURN
 3:C=STATUS 2-
   STATUS 1+4:
   RETURN
 4:C=C+1+((PEEK C
   -1)*7):RETURN
 5:"D"GOSUB 3:IF
   PEEK C<>63CALL
   &CA58
 6:A=STATUS 2+MEM
   -INT (MEM /256
   )*256:B=(
   STATUS 3-A)/25
   6
 7:POKE (C+71),
   PEEK &7864,
   PEEK &7899,
   PEEK &789A
 8:INPUT "BEREICH
   E ? ";D:IF D>1
   0THEN 8
 9:POKE (C-1),D:C
   =C+1:FOR X=1TO
   D:WAIT :PRINT
   "NOCH";B;" SEI
   TEN":WAIT 0:
   PRINT "BER.";X
   ;" SEITEN ";
10:INPUT E:IF B-E
   <0THEN 10
11:CLS
12:L=A:GOSUB 2:
   POKE C,H,L,H,L
   :C=C+4:A=A+E*2
   56:B=B-E:L=A:
   GOSUB 2:POKE C
   ,H,H,0:C=C+3:
   NEXT X
13:CLEAR :GOSUB 3
   :B=1:GOTO 16
14:"B"GOSUB 3:
   GOSUB 4:C=C+2:
   POKE C,PEEK &7
   867,PEEK &7868
   ,PEEK &7864,
   PEEK &7899,
   PEEK &789A:
   GOSUB 3
15:INPUT "BEREICH
    NR ? ";B:IF B
   >PEEK (C-1)
   THEN 15
16:POKE C,B:GOSUB
   4:POKE &7867,
   PEEK (C+2),
   PEEK (C+3),
   PEEK C,PEEK (C
   +1):POKE &7864
   ,PEEK (C+4)
17:POKE &7899,
   PEEK (C+5),
   PEEK (C+6):
   POKE &7865,
   PEEK C,PEEK (C
   +1):B=0:C=0:
   END
18:"L"GOSUB 3:IF
   PEEK (C+73)<>0
   CALL &CA58
19:C=C+71:POKE &7
   864,PEEK C:
   POKE &7899,
   PEEK (C+1),
   PEEK (C+2)
20:A=PEEK &7865:B
   =PEEK &7866:
   POKE &7867,A,B
   ,A,B:POKE (A*2
   56+B),&FF:
   CLEAR :END

STATUS 1
                849
```

```
****: PLATZ FUR SYSTEMADRESSEN DER BEREICHE (75*'?') :****
   1:??????????????????????????????????????????????????????????????????????????????????
     ??
****: INTEGERZAHL -- 2 BYTEZAHL (H=HIGH, L=LOW-BYTE) :****
   2:H=INT (L/256):L=L-H*256:RETURN
****: C=4. BYTE ZLE. 1 = 2.'?' (=AKTUELLER BEREICH) :****
   3:C=STATUS 2-STATUS 1+4:RETURN
****: C=1. SYSTEMADRESSE DES NEU GEWAHLTEN BEREICHS :****
   4:C=C+1+((PEEK C-1)*7):RETURN
****: 'DEFINIEREN', SCHON DEFINIERT ?--'RUCKKEHR INS BASIC' :****
   5:"D"GOSUB 3:
     IF PEEK C<>63CALL &CA58
****: A=ANFANGSADRESSE 1. SEITE, B=FREIE SEITEN :****
   6:A=STATUS 2+MEM -INT (MEM /256)*256:B=(STATUS 3-A)/256
****: STA.3 'NORMAL' AUF '?'NR.73-75 :****
   7:POKE (C+71),PEEK &7864,PEEK &7899,PEEK &789A
****: BEREICHE DEFINIEREN NOTWENDIGE ADRESSEN AUF '?' :****
   8:INPUT "BEREICHE ? ";D:
     IF D>10THEN 8
   9:POKE (C-1),D:C=C+1:
  FOR X=1TO D:WAIT :PRINT "NOCH";B;" SEITEN":WAIT 0:PRINT "BER.";X;" SEITEN ";
  10:INPUT E:
     IF B-E<0THEN 10
  11:CLS
  12:L=A:GOSUB 2:POKE C,H,L,H,L:C=C+4:A=A+E*256:B=B-E:L=A:GOSUB 2:POKE C,H,H,2
     :C=C+3:
  NEXT X
****: BEREICH NR. 1 ANWAHLEN/UBERGABE AUF 16 :****
  13:CLEAR :GOSUB 3:B=1:GOTO 16
```

```
****: ADRESSEN BISLANG AKTIVER BEREICH AUF '?' :****
   14:"B"GOSUB 3:GOSUB 4:C=C+2:POKE C,PEEK &7867,PEEK &7868,PEEK &7864,PEEK &78
      99,PEEK &789A:GOSUB 3
****: NEUEN BEREICH ANWAHLEN/POINTER UMSETZEN :****
   15:INPUT "BEREICH NR ? ";B:
     IF B>PEEK (C-1)THEN 15
   16:POKE C,B:GOSUB 4:POKE &7867,PEEK (C+2),PEEK (C+3),PEEK C,PEEK (C+1):POKE
      &7864,PEEK (C+4)
   17:POKE &7899,PEEK (C+5),PEEK (C+6):POKE &7865,PEEK C,PEEK (C+1):B=0:C=0:
      END
****: LÖSCHEN, NICHT DEFINIERT ? - 'RÜCKKEHR INS BASIC' :****
   18:"L"GOSUB 3:
     IF PEEK (C+73)<>0CALL &CA58
****: 'ALTEN"STA.3 HERSTELLEN (UGL. 7) :****
   19:C=C+71:POKE &7864,PEEK C:POKE &7899,PEEK (C+1),PEEK (C+2)
****: POINTER UMSETZEN -- ALLES GELÖSCHT : ****
   20:A=PEEK &7865:B=PEEK &7866:POKE &7867,A,B,A,B:POKE (A*256+B),&FF:CLEAR :
      END
```

Sofern mit diesem Befehl gearbeitet wird, ist darauf zu achten, daß er erst dann eingesetzt wird, nachdem 10 cm. Papier eingezogen wurden.
2 Beispiele:

```
10:TEXT :LF 40:LF -20:POKE &79E4,0:LF -20
20:GRAPH :GLCURSOR (0,-1000):GLCURSOR (0,-500):POKE &79E4,0,0: GLCURSOR (0,0)
```

Mittels PEEK und POKE können auch Programme die mit MERGE hinzuzuladen werden programmgesteuert gelöscht werden. Hierbei wird es notwendig, daß die Pointer für Programmende (enthalten in den Adressen &7867, &7868) und 'Mergebeginn' (enthalten in den Adressen &7869, &786A) umgesetzt werden.

Als kleine Tücke erweist sich hierbei allerdings, daß jeweils nach Pro.1 und Pro.2 ein &FF als Kennung für das Programmende steht. Somit würde bei der direkten Übertragung des Mergepointers auf den Programmendpointer quasi ein Byte 'verschenkt'. Den Inhalt des Low-Byte vom Mergepointer um eins zu verringern wäre zu riskant, da dieser Inhalt u.U. 0 sein kann!

Würde allerdings nur der Programmendpointer versetzt werden, wäre das Pro.1 nach dem Löschen von Pro. 2 nicht editierfähig. Aus diesem Grund muß auch der Mergepointer versetzt werden, sodaß er auf den Programmstart zeigt, also denselben Inhalt bekommt wie der Programmstartpointer (enthalten in den Adressen &7865, &7866).
Wenn Sie das mit MERGE hinzugeladene Programm um nachstehende 2 Zeilen erweitern, wird es bei Ansprache dieser Zeilen gelöscht, Pro. 1 ist dann auch wieder voll editierfähig.

```
10:E=PEEK &7869*256+PEEK &786A:E=E-1:D=INT (E/256):E=E-D*256:POKE &7867,D,E
20:POKE &7869,PEEK &7865,PEEK &7866:END
```

Weitere Möglichkeiten von PEEK und POKE (Teil 9; 'SPLIT')

U.a. bei der blockmodularen Programmierung ist es manchmal von Vorteil, ein Programm so zu 'splitten', als ob der zweite Teil hinzugeladen wäre. Wird ein Programm so bearbeitet, wäre es mit den MERGE-routinen von Teil 1 und 6 der Serie möglich, größere Programmteile zu Löschen.

Um ein Programm zu 'splitten' werden drei Schritte notwendig:

1.) Auffinden des Bytes ab dem gesplittet werden soll;
2.) Blockverschiebung ab diesem Byte um ein Byte nach 'unten';
3.) Umsetzen des Mergepointers auf das gefundene Byte+1.

Wie in Teil 6 wird wieder das Verschiebeprogramm als Maschinenprogramm direkt in 'SPLIT' eingebunden. Hierfür verantwortlich ist das kleine Hilfsprogramm 'H-SPLIT'. Das Implementieren von 'SPLIT' geschieht entsprechend wie bei 'KEEP', mit der Ausnahme, daß statt des Programms 'M' das Programm 'H-SPLIT' verwendet wird (Start: DEF H)

Um jetzt ein Programm zu splitten, wird 'SPLIT' mit den Schritten F1, DEF S aufgerufen. Es meldet sich mit der Abfrage:

SPLIT NACH ZEILE ?

Eingabe der Zeile, nach der gesplittet werden soll mit anschließendem ENTER. Danach wird das Programmsplitting so durchgeführt, daß der Bereich der nach der angegebenen Zeile steht so interpretiert wird, als ob er mit MERGE hinzugeladen worden wäre.

Wird eine Zeilennummer eingegeben die nicht existiert, kommt es nach ggf. langer Zeit zu ERROR 19. Hier die CL-Taste drücken. Danach muß der Befehl:

POKE &7865,C,D ENTER

durchgeführt werden. Dann kann 'SPLIT' neu aufgerufen werden.

"SPLIT"

```
****: FREIER PLATZ FUR MAPRO (VERSCHIEBUNG ↓) :****
   10:??????????????????????????
****: INTEGERZAHL == 2 BYTE-ZAHL (H=HIGH,L=LOW BYTE) :****
   20:H=INT (L/256):L=L-H*256:RETURN
```

```
****: AUFRUF/B=START MAPRO/ANFANG-ENDPOINTER HAUPTPRO. :****
   30:"S"A=STATUS 2-STATUS 1:B=A+3:C=PEEK (A-2):D=PEEK (A-1):A=C*256+D:E=PEEK &
      7867*256+PEEK &7868
****: EINGABE/A=1. BYTE NACH EINGEGEBENER ZEILE :****
   40:INPUT "SPLIT NACH ZEILE ? ";S
   50:
     IF PEEK A*256+PEEK (A+1)<=SLET A=A+3+PEEK (A+2):GOTO 50
****: 'SPEIST' X UND Y-REGISTER :****
   60:L=E:GOSUB 20:POKE B,&48,H,&4A,L:L=E+1:GOSUB 20:POKE (B+4),&58,H,&5A,L
****: 'SPEIST' U-REGISTER/RUFT MAPRO AUF/SETZT MERGEPOINTER :****
   70:L=E-A:GOSUB 20:POKE (B+8),&68,H,&6A,L:CALL B:L=E+1:GOSUB 20:POKE &7867,H,
      L:POKE A,&FF:L=A+1:GOSUB 20
****: SETZT ENDPOINTER UM 1 HOCH/VERSTECKT SPLIT :****
   80:POKE &7869,H,L:POKE &7865,C,D:END
```

STATUS 1: 373

"H-SPLIT"

```
****: A=STARTBYTE VON SPLIT :****
   10:"H"A=STATUS 2-STATUS 1
****: AB A+15 VERSCH.+/FREIER RAUM FÜR LADEBEF.(60-70 IN SPLIT) :****
   20:POKE (A+15),&F5,&46,&46,&56,&56,&88,&07,&FD,&62,&93,&0B,&9A
****: VERSTECKT 'H-SPLIT' :****
   30:B=PEEK &7869*256+PEEK &786A:B=B-1:C=INT (B/256):B=B-C*256
   40:POKE (A-2),C,B:POKE &7867,C,B,PEEK &7865,PEEK &7866:END
```

STATUS 1: 176

Weitere Möglichkeiten von PEEK und POKE (Teil 10; 'KEEP 2')

Das in Teil 3 vorgestellte 'KEEP' konnte zwar ein Programm retten, es versagte seinen Dienst allerdings, wenn mehrere Programme mit MERGE geladen waren, bzw. wenn es bei MERGE zu einem ERROR 44 kam. Das 'KEEP 2' rettet an Programmen alles 'was es zu retten gibt'.

'KEEP 2' wird genauso wie 'KEEP' implementiert. Auch der Aufruf erfolgt durch die beiden Schritte: F1, DEF K.

Nach dem Aufruf meldet es sich mit der Abfrage:

NEW=1, ERROR 44=2

Bei dieser Abfrage wird die gewünschte Betriebsart eingegeben und mit ENTER abgeschlossen. Wurde 1 eingegeben, erfolgt die Abfrage:

PROGRAMME (ANZAHL) ?

Hier wird die zu rettende Anzahl an Programmen eingegeben. Der Mergepointer zeigt nach dem Retten wie üblich auf das letzte Programm. Wird bei dieser Abfrage 1 eingegeben, wird der Mergepointer nicht gesetzt. Wurde die Betriebsart 'ERROR 44' gewählt, ist die nächste Abfrage:

MERGE, JA=1, NEIN=2 ?

1 ENTER wird eingegeben, wenn ERROR 44 bei MERGE auftrat, 2 wenn der Fehler nicht bei MERGE auftrat. Bei der Eingabe 1 wird der Mergepointer so umgesetzt, daß er auf das Programm zeigt, bei dem der Fehler auftrat, d.h. schon im Rechner befindliche Programme verlieren wie bei MERGE üblich ihre Editierfähigkeit.

Dieser Teil war der vorläufig letzte Teil der Serie. Selbstverständlich konnten nicht alle Aspekte von PEEK und POKE genannt werden. Ich hoffe aber dennoch, daß die Serie (die voraussichtlich in Bälde mit einem 2-pass 'seven-Level' Macroassembler in der PC-1500 Zeitung endgültig abgeschlossen wird) Ihnen einige Anregungen geben konnte.

"KEEP 2"

```
****: SETZT ENDPOINTER UM (LETZTES PROGRAMMBYTE) :****
   5:L=A:GOSUB 50:POKE &7867,H,L:RETURN
****: A=ERSTES BYTE DES ERSTEN PROGRAMMS :****
  10:A=PEEK (C-2)*256+PEEK (C-1):POKE A,0:RETURN
****: A=PROGRAMMENDE BZW. ENDE NACH ERROR 44 :****
  20:
    IF PEEK (A+2+PEEK (A+2))<>&DPOKE A,&FF:RETURN
  30:A=A+3+PEEK (A+2):
    IF PEEK A<>&FFTHEN 20
  40:RETURN
****: INTEGERZAHL == 2 BYTEZAHL (H=HIGH,L=LOW BYTE) :****
  50:H=INT (L/256):L=L-H*256:RETURN
****: AUFRUF/C-2=SPEICHER FÜR 1. PROGRAMMBYTE/NEW, ERROR 44 ? :****
  60:"K"Z=0:C=STATUS 2-STATUS 1:INPUT "NEW=1, ERROR 44=2 ? ";D:
    IF D<1OR D>2THEN 60
****: ERROR 44 ? ÜBERGABE AUF 120 :****
  70:
    IF D=2THEN 120
****: PROGRAMMZAHL (P)/1 PROG.? ANSPRACHE 20,ÜBERGABE AUF 110 :****
  80:INPUT "PROGRAMME (ANZAHL) ? ";P:GOSUB 10:
    IF P=1GOSUB 20:GOTO 110
****: LETZTES PROGRAMM ? MERGEPOINTER UMSETZEN :****
  90:Z=Z+1:
    IF Z=PLET L=A:GOSUB 50:POKE &7869,H,L
****: PROGRAMM RETTEN (ANSPRACHE 20) :****
  95:GOSUB 20
```

```
****: NACHSTES PROGRAMM RETTEN/ENDBYTE UM 1 VERRINGERN :****
  100:
     IF Z<=PLET A=A+1:GOTO 90
  105:A=A-1
****: ANSPRACHE 5/UBERGABE AUF 160 :****
  110:GOSUB 5:GOTO 160
****: ERROR 44 BEI MERGE ? :****
  120:INPUT "MERGE, JA=1, NEIN=2 ? ";M:
     IF M<1OR M>2THEN 120
****: NEIN: ANSPRACHE 10/UBERGABE AUF 150 :****
  130:
     IF M=2GOSUB 10:GOTO 150
****: JA: A=ENDADRESSE+1/MERGEPOINTER SETZEN :****
  140:A=(PEEK &7867*256+PEEK &7868)+1:POKE A,0:L=A:GOSUB 50:POKE &7869,H,L
****: ANSPRACHE 20/ANSPRACHE 5 :****
  150:GOSUB 20:GOSUB 5
****: KEEP 2 VERSTECKEN (STARTPOINTER UMSETZEN) :****
  160:POKE &7865,PEEK (C-2),PEEK (C-1):END
```

STATUS A:544

"BEREICHE 2"

1. Vorbemerkung:

Beim Durchsehen des Listings von 'BEREICHE 2' fallen Ihnen sicher die vielen '?' in den Zeilen 1-5 auf. Auf diese Zeilen werden einerseits vom Programm selbst 'Informationsbytes' bezüglich der Pointerinhalte gesetzt (Zle.1 und teilweise Zle.5), andererseits werden von dem Hilfsprogramm 'MPB2' (=Maschinenprogramme für 'BEREICHE 2') in Zle.2-Zle.5 Maschinenprogramme direkt einsebunden. Nachdem dies geschehen ist, wird das gesamte Programm auf Cassette gespeichert; (Sofern Sie eine Programmcassette bestellt haben, ist dies selbstverständlich schon geschehen) und kann als Gesamtprogramm (also mitsamt den Maschinenprogrammen) mit dem Befehl CLOAD in den Rechner geladen werden. Sie brauchen sich nun in keiner Weise um die Maschinenprogramme kümmern, sie werden automatisch gesteuert. Die Maschinenprogramme sind voll relokatibel, d.h. 'BEREICHE 2' kann bei jeder Speicherversion ohne Änderungen gestartet werden.

Sollten Sie mit einem 2. Zeichensatz oder einem 2. Reservespeicher arbeiten, ist es empfehlenswert, diesen vor dem Laden von 'BEREICHE 2' durch NEW X zu sichern.

2. Einbinden der Maschinenprogramme

Sofern Sie 'BEREICHE 2' nicht auf Cassette besitzen, müssen vor dem Aufruf die Maschinenprogramme eingebunden werden. Hierfür wird das Programm 'MPB2' mit MERGE zu dem Programm 'BEREICHE 2' hinzugeladen und mit DEF M gestartet. Nachdem das BUSY-Symbol verschwunden ist, hat sich 'MPB2' selbst gelöscht und 'BEREICHE 2' ist 'fertig'. Es ist zu empfehlen, es auch auf Cassette zu speichern.

3. Belegung einer Reservetaste

Vor Aufruf des Programms 'BEREICHE 2' muß eine Reservetaste wie folgt belegt werden:

- Im RUN oder PRO-Mode wird eingegeben:

- PEEK &7865 ENTER Anzeige: >Zahl 1< (notieren)
- PEEK &7866 ENTER Anzeige: >Zahl 2< (notieren)
- Belegung der Reservetaste:
  F1:POKE &7865,>Zahl 1<,>Zahl 2<@

4. Bedienungsanleitung für 'BEREICHE 2'

4.1. Erstaufruf (Definieren der Bereiche)

Der Erstaufruf von 'BEREICHE 2' erfolgt durch DEF B. Der PC-1500 meldet sich mit der Frage:

ANZAHL ?

Hier wird die Anzahl der gewünschten Bereiche eingegeben. (Es werden nur Zahlen zwischen 2 und 10 akzeptiert.) Die nächste Anzeige ist:

NOCH X SEITEN

Eine 'Seite' umfaßt 256 Bytes. Zur Definition der einzelnen Bereiche steht die angezeigte Anzahl an Speicherseiten zur Verfügung. Nach dieser Anzeige ist ein ENTER durchzuführen. Daraufhin meldet sich der PC-1500 mit der Frage:

BEREICH X SEITEN ?

Hier wird eingegeben, wieviel Speicherplatz der Bereich X umfassen soll.

Der Speicherplatz eines Bereichs berechnet sich nach der Formel:

Eingegebene Zahl*256-1

In einem Bereich wird ein Programm mitsamt Variablen untergebracht, d.h. der von den Variablen belegte Speicherplatz ist zu berücksichtigen.

Wird eine Zahl eingegeben, die größer als die noch freie Anzahl der Seiten ist, wird sie nicht angenommen. Nach der Eingabe ist ein ENTER durchzuführen.

Während der Eingabe wird von dem Plotter eine kleine Tabelle mit der Form:

B.1: xx (&7865)
B.2: xx (&7865)
...

ausgegeben. Auf die Bedeutung dieser Tabelle wird später eingegangen.

Die oben beschriebene Abfrage wiederholt sich so oft, bis die gewünschte Anzahl der Bereiche definiert ist. Beim letzten zu definierenden Bereich erfolgt keine Abfrage, da bei diesem Bereich automatisch die restlichen Speicherseiten verwendet werden.

4.2. Initialisieren bzw. Umschalten der Bereiche

Nachdem alle Bereiche definiert wurden, schaltet der PC-1500 automatisch in Bereich Nr.1. Bevor die Bereiche mit Programmen belegt werden, müssen sie mit NEW initialisiert werden, da sonst ein 'Speichersalat' entsteht, der sich im schlimmsten Fall nur durch ein ALL Reset beseitigen läßt. Es ist anzuraten, daß die Initialisierung der Bereiche unmittelbar nach der Definition durchgeführt wird. Bereich 1 wird also mit NEW im PRO-Mode initialisiert. Danach geschieht dies mit allen anderen Bereichen. Das Anwählen der Bereiche geschieht wie folgt:

- Betätigen der Funktionstaste F1
- DEF B. Nach einem Signalton meldet sich der Rechner mit der Anzeige:

  BEFEHL ?

- Bei dieser Anzeige wird eingegeben:

WBNR (=Wähl Bereich Nummer). Bei dieser Eingabe verschwindet die Anzeige 'BEFEHL ?' Die Eingabe 'WBNR' muß mit DEF (SPACE) abgeschlossen werden. Daraufhin meldet sich der PC-1500 mit der Frage:

- BEREICH NR ? Eingabe der gewünschten Nummer mit anschließendem ENTER führt zum Umschalten. Auf diese Weise können die Bereiche zum Initialisieren und auch später, wenn sie mit Programmen belegt sind, angesprochen werden.
  Anmerkung: Wird eine Nummer eingegeben, die nicht als Bereich definiert wurde, wird sie nicht angenommen.

Die einzelnen Bereiche sind vollständig voneinander unabhängig, d.h. In jedem Bereich kann mit denselben Zeilennummern, Marknamen und Variablennamen gearbeitet werden.

Selbstverständlich kann in jedem Bereich der Befehl CSAVE verwendet werden. Die Statusfunktionen geben Auskunft über den Speicherplatz des jeweiligen Bereichs. Es kann nicht zu einer Überladung der Bereiche kommen, da die entsprechenden Standardfehlermeldungen (z.B. ERROR 10) immer für den jeweils aktiven Bereich gelten.

Achtung: Von 'BEREICHE 2' werden die Standardvariablen A-Z, A$ und B$ auf 0 bzw. Leerstring gesetzt. Diese Variablen können trotzdem in den Bereichen verwendet werden, sie werden allerdings bei Aufruf von 'BEREICHE 2' gelöscht. Daher ist es zu empfehlen, sie als 'Laufvariablen' zu verwenden. Alle anderen Variablen werden von 'BEREICHE 2' nicht angetastet, d.h. sie bleiben auch erhalten, wenn in einem anderen Bereich mit den Befehlen RUN, CLEAR oder NEW gearbeitet wird. Diese Befehle haben nur in dem jeweils aktiven Bereich ihre bekannte Wirkung.

4.3. Die weiteren Funktionen von 'BEREICHE 2'

Um die weiteren Funktionen von 'BEREICHE 2' aufzurufen, verfährt man entsprechend wie beim Anwählen der Bereiche, d.h. zunächst wird das Programm 'BEREICHE 2' über die Schritte F1, DEF B aufgerufen. Bei der Anzeige 'BEFEHL ?' sind folgende Eingaben möglich:

- PTRANS DEF (SPACE)
- UTRANS DEF (SPACE)
- POUT DEF (SPACE)
- KEEP DEF (SPACE)
- LINK DEF (SPACE)
- BNEW DEF (SPACE)
- AUS DEF (SPACE)

Sollte ein 'Befehl' eingegeben werden, der nicht existiert bzw. Sie vertippen sich bei der Eingabe, kommt es zu der Standardfehlermeldung:

ERROR 11 IN 37

In diesem Fall ist die CL-Taste zu betätigen, die entsprechende Eingabe kann wiederholt werden. In diesem Fall kommt es zu einer Blockierung der ↑ und ↓-Taste nach Anwählen eines Bereichs. Diese Blockierung wird wie folgt aufgehoben:

- LIST ENTER
- Nachdem eine Zeile auf dem Display ist, wird die ➡ bzw. ⬅ Taste mit anschließendem ENTER betätigt. Danach ist die Blockierung aufgehoben.

4.3.1. PTRANS DEF (SPACE)

Der Programmteil PTRANS (=Programmtransfer) erlaubt es, ein Programm eines beliebigen Bereichs in einen anderen Bereich zu kopieren. Da der eigentliche Transfer von einem Maschinenprogramm vorgenommen wird, ist die hierfür benötigte Zeit sehr gering. Nach Aufruf meldet sich der Rechner mit den Abfragen:

- BEREICH NR ? Eingabe des Bereichs, aus dem das Programm kopiert werden soll. (Das Programm bleibt auch in diesem Bereich erhalten) Diese Eingabe muß mit ENTER abgeschlossen werden.
- NACH BEREICH ? Eingabe des Bereichs, in den das Programm kopiert werden soll ENTER

Werden bei diesen Abfragen Bereichnummern angegeben die nicht existieren, werden sie nicht angenommen, die Eingabe kann wiederholt werden.

Sofern alle Eingaben ordnungsgemäß erfolgten, wird der Transfer durchgeführt. Nach Beendigung des Transfers meldet sich der PC-1500 wieder mit der Anzeige 'BEFEHL ?'

4.3.1.1. Fehlermeldungen bei PTRANS

PTRANS verfügt über 3 zusätzliche Fehlermeldungen, die mit einem Signalton eingeleitet werden:

- ??? ERROR 1
  Grund: In dem zuerst eingegebenen Bereich existiert kein Programm.

Was tun ? ENTER führt dazu, daß in den als erstgenannten Bereich geschaltet wird. Danach kann das Programm 'BEREICHE 2' neu aufgerufen werden.

- ??? ERROR 2
  Grund: In dem als zweiten angegebenen Bereich existiert schon ein Programm.
  Was tun ? ENTER führt dazu, daß in den zuerst genannten Bereich umgeschaltet wird.

- ??? ERROR 3
  Grund: Das zu kopierende Programm paßt nicht in den zweitgenannten Bereich, da dieser entweder zu klein ist, oder die dort evt. vorhandenen Variablen den Bereich zu stark auslasten.
  Was tun ? Auch hier wird nach ENTER in den zuerst genannten Bereich geschaltet.

4.3.2. UTRANS DEF (SPACE)

Mit dem Programmteil UTRANS (=Variablentransfer), lassen sich die Variablen eines Bereichs in einen anderen kopieren. Die Vorgehensweise ist dieselbe wie bei dem Teil PTRANS. 'BEREICHE 2' verwendet zum Variablentransfer dasselbe Unterprogramm wie PTRANS. Aus diesem Grund ist die Verarbeitungszeit identisch. Nach Durchführung des Transfers erfolgt wiederum die Anzeige 'BEFEHL ?'.

Werden mit diesem Befehl DIM-Variablen transferiert, dürfen sie im 'neuen' Bereich nicht dimensioniert werden, da der PC-1500 die DIM-Anweisung mit dem Transfer quasi mit übernimmt.

4.3.2.1. Fehlermeldungen bei UTRANS

Prinzipiell sind die Fehlermeldungen hier dieselben, wie in Kap. 4.3.1.1. beschrieben. Lediglich die Nummern sind verschieden.

```
??? ERROR 1  ==  ??? ERROR 4
??? ERROR 2  ==  ??? ERROR 5
??? ERROR 3  ==  ??? ERROR 6
```

Sollte einer der genannten Fehler auftreten, wird genau wie in 4.3.1.1. beschrieben verfahren.

4.3.3. POUT DEF (SPACE)

POUT (=Programm out) löscht in einem angebbaren Bereich das Programm. Die Variablen (außer den Standardvariablen A-Z, A$ und B$; s.o.) bleiben erhalten. Nach dem Aufruf erfolgt die Abfrage:

- BEREICH NR ? Eingabe desjenigen Bereichs, in dem gelöscht werden soll. Die Eingabe wird wiederum mit ENTER abgeschlossen. Auch hier werden Nummern die nicht existieren nicht angenommen. Wurde der Bereich richtig eingegeben, wird das Programm gelöscht. Nachdem das Programm gelöscht ist (ca. 2. sec.), meldet sich der Rechner wieder mit der Anzeige: 'BEFEHL ?'

4.3.4. KEEP DEF (SPACE)

Auch der Programmteil KEEP kann für jeden beliebigen Bereich angewählt werden. Die Abfrage ist dieselbe wie bei 'POUT'. Für das Retten der Programme nach NEW bzw. ERROR 44 benötigt der PC-1500 etwas länger. Um ca. 5 Kb zu retten benötigt KEEP ungefähr 15 sec. Nach ERROR 44 kann allerdings nur der noch gelesene Programmteil gerettet werden. Selbstverständlich sind die geretteten Programme voll editierbar. KEEP kann auch dann eingesetzt werden, wenn ein Bereich mit POUT gelöscht wurde.

Anmerkung: War die erste Zeilennummer des zu rettenden Programms >255, wird diese von KEEP geändert. Selbstverständlich kann diese geänderte Zeilennummer wieder editiert werden.

4.3.5. LINK DEF (SPACE)

Mit LINK können beliebig viele Bereiche zusammengeschlossen werden. Nach dem Aufruf meldet sich der Rechner mit den Fragen:

- BEREICH NR ? Eingabe des ersten Bereichs ab dem zusammengeschlossen werden soll. Auch hier werden Nummern die nicht existieren nicht akzeptiert.
- MIT BEREICH ? Eingabe der Bereichsnummer bis zu der einschließlich zusammengeschlossen werden soll. Es ist darauf zu achten, daß die eingegebene Zahl größer als die zuerst eingegebene Zahl ist.

Werden mehrere Bereiche mit diesem Befehl zusammengeschlossen, erfolgt der Variablenzugriff auf die Variablen des zuletzt eingegebenen Bereichs. Alle anderen bleiben - auch wenn sie denselben Namen haben - unangetastet.

Es besteht die Möglichkeit, zwischen allen zusammengebundenen Bereichen zu springen. Dies ist sowohl mit dem Befehl GOTO als auch mit den Befehlen GOSUB und RETURN möglich. Hierbei verfährt man folgendermaßen:

Die Sprungziele müssen mit Markennamen adressiert werden, sofern von einem Bereich in den anderen gesprungen werden soll. Sprünge innerhalb eines Bereichs werden wie gewöhnlich geschrieben (also auch mit Zeilennummern). Beim Sprung zwischen den Bereichen muß vor dem GOTO, GOSUB bzw. RETURN der Programmstartpointer umgesetzt werden. Soll z.B. von Bereich 5 nach Bereich 1 gesprungen werden, ist vor dem Sprungbefehl der Befehl:

```
POKE &7865,xx
```

zu setzen. xx ist hierbei dezimal die Zahl, die in der Übersichtsliste nach B.1 ausgegeben wurde. Allgemein gilt, daß beim Springen zwischen den Bereichen immer der Startpointer so umgesetzt wird, daß er auf den jeweiligen Bereich zeigt, in den gesprungen wird. Es handelt sich immer um die dezimal angegebene Zahl xx, die nach der jeweiligen Bereichnummer auf der Liste angegeben ist. Der genannte Befehl muß auch immer vor RETURN stehen. Trotzdem ist es möglich, ein Unterprogramm eines Bereichs aus verschiedenen Bereichen anzuspringen. In diesem Fall setzt man vor das RETURN den allgemeinen Befehl:

```
POKE &7865,X
```

Der Variablen X wird dann vor Ansprache des Unterprogramms der entsprechende Inhalt zugewiesen. (Diejenige Zahl, die nach der Bereichnummer steht, aus der das Unterprogramm angesprochen wird.)

Die Standardvariablen können in diesem Fall benutzt werden, da sie erst nach Aufruf von 'BEREICHE 2' zurückgesetzt werden

Die Sprungziele zwischen den Bereichen dürfen nur als Markennamen geschrieben werden. In den einzelnen Bereichen können aber durchaus dieselben Markennamen verwendet werden, der Sprung erfolgt immer in denjenigen Bereich, der durch den Befehl POKE &7865,xx angewählt wird.

Das Low-Byte des Programmstartpointers muß nicht umgesetzt werden, da der Inhalt bei allen Bereichen 0 ist.

Beim Arbeiten mit LINK ist auf folgendes zu achten:

- Der Programmstart muß in dem Bereich erfolgen, der als erster Bereich angegeben wurde.
- Sprünge sind nur in den Bereichen möglich, die zwischen dem zuerst und dem zuletzt genannten Bereich liegen. D.h. wurden die Bereiche 2 und 5 zusammengeschaltet, dürfen die Sprünge nur in den Bereichen 2-5 geschehen, nicht etwa nach Bereich 1 oder 6.
- Solange die Bereiche zusammengeschlossen sind, ist die MODE-Taste blockiert, da die Bereiche während des Zusammenschluß' nicht editiert werden dürfen. Sollte dennoch ein Fehler auftreten, bzw. das Programm editiert werden, ist dies möglich. In diesem Fall ist das Programm 'BEREICHE 2' neu aufzurufen. Beim Aufruf wir der Zusammenschluß aufgehoben, die einzelnen Bereiche können wie gewohnt angesprochen und editiert werden. Danach können sie wieder zusammengeschlossen werden.

4.3.5.1. Fehlermeldung bei LINK

Die Fehlermeldung, die bei LINK auftreten kann, ist:

??? ERROR 7

Grund: Bei der Abfrage MIT BEREICH ? wurde ein Zahl eingegeben, die größer als die erstgenannte Zahl ist, bzw. nicht als Bereich definiert ist.
Was tun ? ENTER, der PC-1500 schaltet in den zuerst angegebenen Bereich.

4.3.6. BNEW DEF (SPACE)

Der Befehl BNEW (=ALLE Bereiche NEW), dient dazu, alle angelegten Bereiche zu Löschen. Das Programm 'BEREICHE 2' bleibt erhalten. Nachdem die Bereiche gelöscht sind, ertönt nach ca. 10 sec. der Signalton.

Nun kann entweder das Programm 'BEREICHE 2' aufgerufen werden (DEF B) oder das Programm 'BEREICHE 2' wird durch den Befehl NEW ENTER im PRO-Mode gelöscht. Der Speicher ist dann wieder wie gewöhnlich voll zugänglich. Wird das Programm nach BNEW mit DEF B aufgerufen, wird dieser Aufruf als Erstaufruf interpretiert, d.h. Der Rechner meldet sich mit der Abnfrage: ANZAHL ? Die Speicheraufteilung kann also neu definiert werden.

4.3.7. AUS DEF (SPACE)

Mit diesem Befehl wird der Rechner ausgeschaltet. Nach dem Wiedereinschalten erscheint das BUSY-Symbol, es erfolgt kein Druckerreset. Bei aktivem BUSY-Symbol muß die RCL-Taste betätigt werden. Alle anderen Tastenbetätigungen (außer DEF, SHIFT, SML hier geschieht nichts), quittiert der PC-1500 damit, daß er sich ausschaltet. Sie können also sicher sein, daß Ihr Rechner nicht von Unbefugten benutzt wird. Nachdem die RCL-Taste betätigt wurde, erscheint die Abfrage 'BEFEHL ?'.

Achtung: Es ist zu empfehlen, den Rechner nur auf diese Weise auszuschalten, da in anderen Fällen die Bereiche ignoriert werden könnten. Sollten Sie dennoch aus Versehen einmal mit OFF abschalten, bzw. der Rechner 'schläft' nach den üblichen 7 min. ein, kann es passieren, daß er sich nach dem Wiedereinschalten mit der Anzeige:

NEW0?: CHECK

meldet. Selbstverständlich wird in diesem Fall das NEW 0 nicht durchgeführt. Man verfährt wie folgt. Vor dem Aufruf von 'BEREICHE 2' wird nach Betätigen der CL-Taste eingegeben:

- PEEK &7865 ENTER
- Jetzt vergleicht man die angezeigte Zahl mit der Liste. Sollte die angezeigte Zahl identisch mit der letzten Zahl auf der Liste sein, kann 'BEREICHE 2' getrost wieder aufgerufen werden. Sollte die angezeigte Zahl allerdings nicht die, die zuletzt in der Liste aufgeführten sein, muß vor dem Aufruf von 'BEREICHE 2' eingegeben werden:
- POKE &7864, >die Zahl, die in der Liste nach der angezeigten aufgefürt ist< ENTER.

Nach diesem Befehl kann 'BEREICHE 2' wieder aufgerufen werden. Sollte sich der Rechner nach versehentlichem 'falschen' Ausschalten nicht mit NEW0? :CHECK melden, sind trotzdem die eben beschriebenen Schritte durchzuführen, da sonst die Bereiche von dem Rechner zerstört werden.

5. Wichtige Hinweise

- Selbstverständlich können Sie in dem Programm Änderungen vornehmen und weitere Befehle einfügen. Dem kommentierten Listing kann die Aufgabe der einzelnen Unterprogramme entnommen werden. Es ist aber darauf zu achten, daß in den Zeilen 1-5 keine Änderungen vorgenommen werden, da hier die Maschinenunterprogramme eingebunden sind. Eine Änderung dieser Zeilen könnte fatale Folgen haben.
- Innerhalb den einzelnen Bereichen kann nicht mit dem Befehl MERGE gearbeitet werden, da für den Mergepointer keine 'Informationsbytes' vorgesehen sind.
- Innerhalb den einzelnen Bereichen darf der Befehl NEW X zum Hochsetzen der Programmstartpointer nicht verwendet werden. Selbstverständlich können in die einzelnen Bereiche auch Maschinenprogramme gelesen werden. Die Startadresse eines Bereichs ist die in der Liste hinter der Bereichnummer angegebene Zahl*256. Die Endadresse ist die nachfolgende Zahl*256-1. (=STATUS 3-1 des jeweiligen Bereichs). Beim letzten Bereich kann die Endadresse also über den Befehl STATUS 3 berechnet werden.
- Die von mir angebotene 'VIP' bzw. 'EDITOR'Software kann nicht direkt bei aktiver Speicherunterteilung eingesetzt werden. Auf Wunsch können die entsprechenden Programme selbstverständlich auf 'BEREICHE 2' eingerichtet werden. Es müssen einige kleine Änderungen vorgenommen werden, die selbst der absolute Neuling binnen ca. 10 min. durchführen kann. Sollten Sie ein entsprechendes Programm auf 'BEREICHE 2' anpassen wollen, rufen Sie mich bitte an. Ich übersende Ihnen dann eine kostenlose Anleitung zum Ändern.

6. Einige Einsatzmöglichkeiten von 'BEREICHE 2'

- Zunächst macht 'BEREICHE 2' den Befehl MERGE überflüssig.
- Vor allem in Simulationsmodellen sind die Transferbefehle von großem Wert. Einmal eingegebene Daten können mit verschiedenen Programmen bearbeitet werden, ohne daß verschiedene Variablennamen verwendet werden.
- Auch bei der Programmierung ergeben sich Vorteile:
  So können beispielsweise in einem Bereich die Unterprogramme liegen, in anderen Bereichen verschiedene Programme zur verschiedenen Koordination der Unterprogramme.
- Sollen mehrere Programme im Rechner bleiben (z.B im Außendienst), ist dies mit 'BEREICHE 2' ohne weiteres möglich. Nachdem die Bereiche definiert sind, bleiben diese auch ohne CE-150 erhalten, sofern sich im PC-1500 Batterien befinden.

Die hier aufgeführten Beispiele stellen selbstverständlich nur einen kleinen Teil der Möglichkeiten dar. Ich hoffe, daß das Programm 'BEREICHE 2' dazu beiträgt, Ihren PC-1500 optimal zu nutzen.

INFO: 'TASTATUR 2' FÜR 2. ZEICHENSATZ

Ein Handicap bei der Verwendung eines 2.Zeichensatzes war bislang immer die Tatsache, daß Groß- bzw. Kleinbuchstaben, Graphikzeichen etc. nur umständlich über die SHIFT -Funktion oder gar über den Befehl PRINT bzw. LPRINT CHR$ I aufgerufen mußten. Dieses Ärgernis wird durch das kleine BASIC-Programm 'TASTATUR 2' ein für allemal beseitigt. Es bietet dem Anwender die Möglichkeit, für jeden beliebigen 2. Zeichensatz eine 2. Tastaturbelegung frei zu definieren! Selbstverständlich müssen hier nicht alle Tasten mit einer neuen Belegung versehen werden, diejenigen, die nicht neu definiert werden, erhalten in der 2. Belegung automatisch dieselbe Funktion wie in der ersten Belegung.

Die Bedienung des Programms 'TASTATUR 2' ist denkbar einfach. Während der Definition einer 2. Tastaturbelegung befindet sich auf dem Display die Anzeige:
ALT=? NEU=?
Eingabe ALT: Bisherige Tastaturbelegung
Eingabe NEU: frei definierbare 2. Tastaturbelegung ('ALT' wird in der 2. Belegung durch 'NEU' ersetzt).

Es ist möglich, hier sowohl die ASC II-Zeichen als auch CHR-Codes (ohne Kennung!!) einzugeben, was es ermöglicht, auch die MODE - Taste etc. neu zu belegen. Außerdem können so auch Zeichen auf die Tastatur gebracht werden, die bislang nicht direkt aufgerufen werden konnten (z.B. CHR$ 92 oder CHR$ 127).

Während der Neudefinition wird vom Programm 'TASTATUR 2' ein 285 Bytes langes Hexcodeprogramm erzeugt, das es gestattet, von der bislang benutzten Tastaturbelegung auf die frei definierte umzuschalten. Selbstverständlich kann auch wieder von der frei definierten 2. Tastaturbelegung auf die erste Tastaturbelegung des 2. Zeichensatzes umgeschaltet werden. Da diese Umschaltung von einem Maschinenprogramm gesteuert wird, beträgt die hierfür benötigte Zeit weniger als 1 sec!! Über das von 'TASTATUR 2' erzeugte Maschinenprogramm wird es möglich, sowohl programmgesteuert als auch von Hand (über eine Funktionstaste) diese Umschaltung durchzuführen.

Von 'TASTATUR 2' wird jedoch nicht nur das Maschinenprogramm erzeugt, sondern gleichzeitig ein Übersichtsliste
Tastaturbelegung 1 - Tastaturbelegung 2
ausgedruckt. Desweiteren werden die Startadressen der Maschinenprogramme I (CALL I = Tastatur 1) und Y (CALL Y = Tastatur 2) ausgedruckt.

Nach der Erzeugung des Maschinenprogramms kann das Programm 'TASTATUR 2' aus dem Speicher gelöscht werden, somit werden also vom Maschinenprogramm nur 285 Bytes belegt.

Informationen zur Speicherbelegung

STATUS 0 (= MEM)
Zahl der nicht durch Basic-Programme belegten Bytes

STATUS 1
Zahl der durch Basic-Programme belegten Bytes

STATUS 2
Adresse des ersten auf Basic-Programme folgenden Bytes

STATUS 3
Adresse des ersten Bytes des Datenbereiches (Felder u. Variable)

STATUS 3 - STATUS 2
Zahl der nicht durch Programme oder Daten belegten, also tatsächlich verfügbaren Bytes

256 * PEEK &7864 - STATUS 3
Zahl der durch Daten belegten Bytes

256 * PEEK &7865 + PEEK &7866
Erste Adresse des für Basic-Programme nutzbaren oder bereits durch selbige genutzten Speicherbereiches

256 * (PEEK &7865 - PEEK &7863) + PEEK &7866 - &C5
Zahl der Bytes, um die die Startadresse des Basic--Programmspeichers durch NEW &XXXX heraufgesetzt wurde

Getrenntes Löschen der Standardvariablen und selbst definierter Variablen

Mit den beiden nachstehenden Befehlsfolgen ist die Möglichkeit gegeben, den Standardvariablenspeicher und den Programmspeicherbereich, welcher von Feldern und selbst definierten Variablen belegt wird, unabhängig von einander zu löschen.

a) Alleiniges Löschen der Standardvariablen

1. POKE &78AE, PEEK &7899, PEEK &789A
2. CLEAR
3. POKE &7899, PEEK &78AE, PEEK &78AF

b) Alleiniges Löschen selbst definierter Variablen

```
POKE &7899, PEEK &7864, 0
```

Allgemeines zur Maschinensprache

Allgemeines zu Maschinensprache

Zahlensysteme:

Die CPU des PC-1500 arbeitet in Maschinensprache und benutzt darum bei Zahlenwerten die duale Schreibweise, da 8 Datenleitungen (Bit) zur Verfügung stehen. Diese Schreibweise ist für den normalen Menschen sehr uneffektiv, da für Zahlen viel mehr Stellen gebraucht werden als im Dezimalsystem. Daher hat der Mensch ein Zahlensystem eingeführt, daß ein Vermittler zwischen Dual- und Dezimalsystem sein soll, das Hexadezimalsystem. Im Hexadezimalsystem gibt es 16 Ziffer (Dual 2 / Dezimal 10), die an jede Stelle gesetzt werden können. Diese 16 Ziffern sind: 0-9 A-F (je 4 Bit eine Hex-Zahl). Um Hex-Zahlen zu kennzeichnen schreibt man ein '$' oder '&' davor, ein 'H' dahinter oder schreibt die Zahl 16 als Index hinter die Hex-Zahl. Man rechnet die Zahlen um, wie bei allen anderen Zahlensystemen:

```
&A75B =
 B(11) * 16^0
+5     * 16^1
+7     * 16^2
+A(10) * 16^3
ergibt  46458
```

Bei dem PC-1500 kann man die Umrechnung Hex nach Dez mit dem '&'-Zeichen vor der Hex-Zahl vornehmen (max. 4 Hex-Ziffern). Die Dez-nach-Hex-Umrechnung geschieht mit einem kleinen Programm:

Programm:Zahl aus H nach HE$ umwandeln: (0<=H<=255)

```
10:H$="0123456789ABCDEF"
20:HE$=MID$ (H$,H/16+1,1)+MID$ (H$,(HAND 15)+1,1)
```

Programm:...für beliebig viele Stellen:

```
10:IF H=0LET HE$="0":RETURN
20:I=0:HE$=""
30:IF H>=16LET H=H/16:I=I+1:GOTO 30
40:FOR J=0TO I
50:HE$=HE$+MID$("0123456789ABCDEF",H+1,1):H=(H-INT H)*16
60:NEXT J:RETURN
```

Ein Hexmonitor, den man für Maschinensprache verwendet, arbeitet, wie der Name schon sagt, mit Hex-Zahlen. Es gibt aber verschiedene Hexmonitore, die auch mit Dez-Zahlen oder anderen Werten arbeiten können. Beim PC-1500 rechnet man mit 8 oder 16 Bit, d.h. mit 2- oder 4-stelligen Hex-Zahlen. Eine 2stellige Hex-Zahl representatiert 8 Bit und man nennt sie ein Byte. In diesem Byte werden die 8 Bit numeriert. Man nennt das höhswertige Bit MSBit oder auch Bit7 und das niederwertigste Bit LSBit oder Bit0. 16-Bit-Werte unterteilt man in 2 Byte, d.h. ein höherwertiges und ein niederwertiges, MSByte und LSByte. Die Bit dieses Wertes kann auch durchnumeriert werden von 15 bis 0. Wird ein solcher 16-Bit-Wert in den Speicher des PC-1500 ablegen, so stellt man fest, daß eine Speicherstelle nur 8 Bit aufnehmen kann. Darum legt man das MSByte in der bestimmten Adresse ab und das LSByte in der darauf folgenden Speicherstelle. Wird deshalb von einem 16-Bit-Wert gesprochen, so wird dessen 1.Speicherstelle nur angegeben.

Ein negativer Wert wird durch sein Zweierkomplement dargestellt. Dieser wird durch Invertierung aller Bits (Einerkomplement) und incrementieren (weiterzählen um 1) errechnet.

Ein Byte:

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	
(MSBit)							(LSBit)	
128	64	32	16	8	4	2	1	(dezimal)
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	(Potenzen)
&80	&40	&20	&10	&8	&4	&2	&1	(hex)

Was ist ein Maschinen-Sprache-Programm ?

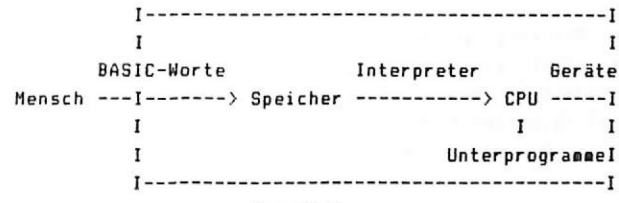
Sie besitzen einen PC-1500. Mit der Bedienungsanleitung lernten Sie seine Programmierung in BASIC. Aber der PC-1500 kann noch mehr:

- Der PC-1500 versteht zusätzliche BASIC-Befehle
- Der PC-1500 ist in Maschinensprache programmierbar

In Artikeln wird immer wieder darauf hingewiesen. Vielen Benutzern fehlt aber eine grundlegende Information. Mit diesem Artikel sollen Sie den ersten Schritt zur Maschinensprache machen.

BASIC ist eine leicht zu erlernende (fast) normierte Programmiersprache. Daher laufen auf dem PC-1500 auch Programme, die ursprünglich für andere Computer-Typen entwickelt wurden.

Das Herz des Computers ist sein Microprozessor. Darin ist die CPU seine Zentraleinheit, die die einzelnen Befehle ausführt. Die CPU "versteht" unsere BASIC-Befehle nicht direkt. Daher ist im Rechner ein BASIC-Interpreter installiert. Dieser verzweigt während der Programmabarbeitung zu im Rechner gespeicherten Unterprogrammen, die in einem für die CPU verständlichen M(aschinen)-Code geschrieben sind. Beim Programmieren in BASIC geben wir Buchstabenfolgen ein, die der Computer zum Teil als Befehle erkennt. Wir wissen auch, welche Funktionen der Computer bei der Befehlsausführung durchführt, denn der Computer hat eine interne Befehlstabelle, in der enthalten ist, was bei einem BASIC-Befehl auszuführen ist.



BASIC-Programme laufen also relativ langsam, da sie bei der Ausführung in Maschinensprache "übersetzt" werden. Manche Probleme können wir aber auch in Maschinensprache programmieren und den M-Code direkt eingeben. Dadurch verkürzt sich dei Ausführungszeit erheblich.

Mit POKE können M-Codes in freie Speicherbereiche geschrieben werden. Will man das M-Programm verstehen benötigt man weitere Informationen. Ohne Zusatzliteratur und Software versteht man nur wilde Zahlenfolgen.

Für Maschinensprache gilt ähnliches wie für BASIC. Auch hier muß die CPU bei bestimmten Zahlencodes bestimmte vorher absehbare Funktionen durchführen. Es exestiert auch in der CPU eine Befehlstabelle für alle M-Codes, in denen die nötigen Schaltungsfolgen enthalten sind. Um nicht immer mit Zahlencodes arbeiten zu müssen, hat man Mnemonics eingeführt. Das sind Befehlsworte, vergleichbar den Befehlen in BASIC. Wie jeder Computer ein anderes BASIC besitzt, so hat jede CPU eine andere Mnemonic, obwohl man versucht, diese Systeme in beiden Fällen einander anzugleichen.

Werkzeuge für Maschinensprache:

Wenn diese Maschinensprache so kompliziert ist, dann gibt es Werkzeuge, die die Arbeit mit ihr erleichtern. Den freien Eingriff in den Speicher, ohne PEEK oder POKE, verbunden mit Funktionen zum Starten von M-Programmen bietet ein HEXMONITOR. Die übersetzung von Mnemonics, die leichter zu bearbeiten sind, in M-Programmen mit Fehlerkontrolle ermöglicht ein ASSEMBLER. Die Arbeit der Rückübersetzung der codierten Befehle in verständliche Mnemonics nimmt uns ein DISASSEMBLER ab. Ein DEBUGGER oder TRACER kann M-Programme testen bzw. kontrolliert ablaufen lassen, wie man BASIC-Programme mit dem Befehl TRACE untersuchen kann.

Neue BASIC-Befehle:

Der Speicher des Rechners besteht aus einzelnen Speicherstellen. Jede Speicherstelle hat eine Adresse vergleichbar einer Hausnummer. In jeder Zelle ist eine Zahl 0 - 255 (&00 - &FF) gespeichert. Man kann diese Werte mit folgenden Befehlen manipulieren:

1. PEEK(#) Adresse : Der Inhalt der angegebenen Speicherstelle wird ausgelsen und kann weiterverwendet werden (A=PEEK x) Wird das '#'-Zeichen mit angegeben, so wird auf die zweite Seite des Speichers zugegriffen, da der PC-1500 zwei Speicherseiten mit je 64K Adressraum besitzt. Die zweite Seite ist von I/O-Bausteinen (Drucker/Kassette/Tastatur/...) belegt.
2. POKE(#) Adresse,Wert1,Wert2,... : Ab der angegebenen Speicherstelle werden die Zahlenwerte (0 - 255) abgelegt. Für das '#'-Zeichen siehe auch 1.
3. CALL Adresse,Var : Das Unterprogramm in Maschinensprache beginnend bei der angegebenen Adresse wird aufgerufen.
   - Ist eine numerische Variable angefügt, so wird der enthaltene Wert (-32766 bis 32766) dem X-Register der CPU übergeben vor Start des Unterprogrammes.
   - Ist eine String-Variable angefügt, so wird im X-Register die Startadresse und im A-Register die maximale Länge des Strings angegeben.
   - Ist nach Rücksprung aus dem Programm das C-Flag der CPU gesetzt, so wird der Inhalt, gespeichert wie die Anfangswerte, in die Variablen zurückgenommen. Ein Aufruf in der 2. Seite ist nicht möglich.
4. CSAVE M("<Name>";)Startadresse,Endadresse : Der angegebene Speicherinhalt wird wie bei CSAVE auf Kassette gespeichert.
5. CLOAD M("<Name>";) (Adresse) : Der auf Band gespeicherte Speicherbereich wird ab der angegebenen Adresse eingelesen und

gespeichert. Wird keine Adresse angegeben, so wird die beim Abspeichern verwendete Start-Adresse verwendet.

6. NEW Adresse : Der Anfang des BASIC-Bereiches wird auf den angegebenen Wert gesetzt. Es müssen mindestens 197 Byte für die Reserve-Tasten-Belegung bis zum Anfang des RAM-Speichers verbleiben.
7. OPN ("<Gerätename>") : Setzt interne Zeiger für dieses Gerät, so daß Befehle mit gleichem Namen auf anderen Geräten auch laufen (CE-158)

Speichertypen:

Es gibt im PC-1500 drei Speichertypen:

ROM: Read Only Memory.
Dieser Speicherbereich ist "fest verdrahtet". Hier kann der Benutzer keine Änderung vornehmen. In diesem Bereich liegen feste Programme in Maschinensprache. Diese Programme bilden den Grundstock zum System und treten auch als BASIC-Interpreter auf.

RAM: Random Access Memory
Dieser Speicherbereich ist im freien Zugriff und kann vom Benutzer verändert werden. BASIC-Programm-Texte, sowie System-Speicherplätze liegen in einem solchen Speicher.

IO-Ports: Input / Output
Dieser Speicherbereich ist direkt auf mit Leitungen verschaltet, die aus dem Computer führen, wie der Tastatur, dem Drucker oder ähnlichem. Wenn man diese Speicherbereiche manipuliert, so kann man Ausgänge des PC-1500 schalten. !VORSICHT! Wenn man etwas falsches in diese Speicherbereiche schreibt, so können, speziell beim Drucker, mechanische Teile zerstört werden !!!

Erforschung des PC-1500:

Die Speicheraufteilung ist abhängig von dem eingesetzten Erweiterungsmodul. Sie ist einer Memory-Map zu entnehmen:

PC-1500 ohne Erweiterung: &4000-&47FF
PC-1500 mit CE-151 : &4000-&57FF
PC-1500 mit CE-155 : &3800-&5FFF

```
PC-1500 mit CE-159      :&2000-&47FF
PC-1500 mit CE-160      :&0000-&1FFF und &4000-&47FF
PC-1500 mit CE-161      :&0000-&47FF
```

Die für den Anfänger wichtigsten Adressen stehen in:(Adresse=256*PEEK (1.Wert)+PEEK (2.Wert))

&7865/&7866 BASIC-Startpointer. Zeigt auf Speicherzelle, ab der BASIC-Programmzeilen abgespeichert und auf dem Display angezeigt werden können.

&7867/&7868 Zeigt auf letztes Byte des BASIC-Programms. Dieses Byte steht hinter der letzten Programm-Zeile und ist ein &FF.

&7869/&786A Zeigt auf Anfang des zuletzt geladenen Programms. Es sind nur die BASIC-Zeilen editierbar, die ab dieser Adresse gespeichert sind.

Mit diesen Kenntnissen und den Befehlen PEEK / POKE kann das Innere des Computers erforscht werden. Obwohl die Befehle mit dezimalen Zahlen arbeiten, ist es nützlich, das Hexadezimalsystem anzuwenden. Denn dann kann man die Adressen direkt aus dem High- und Lowbyte ablesen.

Eingabe von Maschinensprache-Programmen:

Maschinensprache-Programme, die man in Zeitschiften o.ä. findet, können in drei verschiedenen Formen vorliegen:

1. Das Programm liegt als Hex-Dump, erkennbar an den vielen zweizifferigen Hex-Zahlen, vor. Es kann dann über einen Hexmonitor Byte (ein Byte ist eine zweizifferige Hex-Zahl, deren Ziffern 0-9 und A-F umfassen) für Byte nacheinander eingegeben werden. Man kann auch ein BASIC-Programm schreiben, welches diese fehleranfällige Arbeit abnimmt:

```
10 DATA ..,..,..,..,..,..
20 DATA ..,..,..,..,..,..,-1

30 A=(Startadresse):RESTORE
40 READ B:IF B=-1END
50 POKE A,B:A=A+1:GOTO 40
```

Sie geben in den DATA-Zeilen die Bytes ein: DATA &XX,&YY,&ZZ,...,-1 und geben als letzten Wert in der letzten DATA-Zeile den Wert -1 ein, damit das Programm das Ende der Liste erkennt. Sie starten das

Programm dann mit RUN, nachdem Sie die Startadresse des Programmes eingegeben haben in Zeile 30. Diese Startadresse sollte in einem günstigen Bereich liegen. z.B. kann man STATUS 2 eingeben, die Endadresse des BASIC-Programmes, oder auch &7650, dem Bereich, in dem die Variablen E$-O$ liegen (Dieser Bereich geht nur bis &76FF und darf nicht überschritten werden). Nachdem Sie das Programm nun so leicht in den Speicher übertragen haben, können Sie es entsprechend der Programmanleitung des Maschinenprogrammes starten.

2. Das Programm liegt als Assembler-Quell-Code vor, was man an den vielen Namen und den Mnemonics der Maschinensprache erkennen kann.(z.B. LD A,Anzahl) Falls Sie keinen Assembler haben, müßten Sie dieses Programm von Hand in Maschinensprache übersetzen und dann wie bei 1. eingeben. Haben Sie jedoch einen Assembler, dann geben Sie das Programm entsprechend der Konventionen Ihres Assemblers ein und übersetzen es anschließend in einen günstigen Speicherbereich.

3. Das Programm liegt als disassembliertes Listing vor, was man an den Mnemonics aber fehlenden Namen erkennen kann.(z.B. LD A,55) Hier können Sie wie bei 2. verfahren.

Allgemein muß man sagen, daß man Programme, die als Assembler Quell-Code vorliegen am leichtesten weiterverwenden (umformen) kann. Sie müssen auch die Schreibweise Ihres Assemblers beachten, denn es gibt verschiedene Mnemonics für die gleichen Befehle, da es verschiedene Systeme gibt.(z.B. LDD (X),A / SDE X). Die fertigen Programme können auf Band gesichert werden mit 'CSAVE M"<Name>";Startadresse,Endadresse'. Laden kann man diese Programme wieder mit 'CLOAD M' an die Stelle, von wo man sie abgespeichert hat, oder mit 'CLOAD M Startadresse' an eine gewünschte neue Adresse. Man muß beachten, daß es viele Maschinen-Programme gibt, die nur in einem Bereich lauffähig sind, also nicht in einen anderen Speicherbereich geladen werden dürfen.

Memory-Map

(XXXX/.. bedeutet, daß diese und die nächste Speicherstelle zusammen den angegebenen Wert ergeben, wobei in XXXX das MSByte steht.)

Adresse	Name	Bedeutung
7000-71FF		Gleich dem Bereich &7400-&75FF
7200-73FF		Gleich dem Bereich &7400-&75FF

7400-744D Anzeige 1 1. und 3. Viertel der Anzeige, die in 4 Blöcke eingeteilt ist, deren interner Aufbau kompliziert ist. Der Speicherbereich ist so aufgeteilt, daß die untere (LSB) Hälfte der Byte das 1.Viertel, die andere Hälfte das 3.Viertel der Anzeige ergibt. Dabei bilden die geraden Adressen die (in der Anzeige) obere Hälfte, die ungeraden die andere. Es ergibt sich:

Aufbau der Punktmatrixanzeige:

```
unten              oben
----------------------
I  I  I  I  I  I  I  I  Spaltennummer 0/Spaltenadresse &7400
----------------------
 2  1  0   3  2  1  0   Bit-Nr. der betreffenden Byte
 &7401       &7400

                        Bit-Nr. der betrffenden Byte
 6  5  4   7  6  5  4   Spaltennummer 78/Spaltenadresse &7600
 &7601       &7600
```

(Von den ungeraden Adressen bleibt MSBit unbenutzt) Die Spaltennummer ist gleich dem BASIC-Kommando GCURSOR, die Spaltenadresse gleich der geraden Speicheradresse. Zur Unterscheidung des 1. und 3.Viertels wird im 3.Viertel der Speicherbereich &7600-&764C verwendet, der gleich diesem Bereich ist.

744E Status 1 Anzeige-Segmente DEF I II III SMALL ht SHFT BUSY Diese Symbole können auch in Programmen gesetzt werden, so daß

Sie die Reserve-Ebenen oder das SMALL-Sysmbol umschalten können. Diese Symbole in der Anzeige werden von versch. Programmen ausgewertet (SIN,COS,TAN,Small bei Input...) Sie müssen dabei in die Speicherstelle den Wert einschreiben, der sich aus der Addition der Zahlenwerte für die einzelnen Symbole ergibt. DEF= 128,I= 64,II= 32,III= 16,SMALL= 8,ht= 4,SHFT= 2,BUSY= 1

744F Status 2 Anzeige-Segmente RUN PRO RESERVE RAD G DE
Diese Speicherstelle ist wie Status 1 zu gebrauchen. RUN= 64,PRO= 32,RESERVE= 16,RAD= 4,G= 2,DE= 1

7450-745F E$ Standartvariable E$ Dieser Bereich kann vom Benutzer sehr gut für kleine Maschinenprogramme genutzt werden, solange er den Bereich nicht überschreitet. Außerdem wird dieser Bereich bei CLEAR,NEW oder Wertzuweisungen geändert. Gleiches gilt für die Speicher F$ bis O$, Q$ bis Z$, sowie A bis Z.

7460-746F F$
7470-747F G$
7480-748F H$
7490-749F I$
74A0-74AF J$
74B0-74BF K$
74C0-74CF L$
74D0-74DF M$
74E0-74EF N$
74F0-74FF O$

7500-754D Anzeige 2 2. und 4. Viertel der Anzeige Die Benutzung ist gleich dem Bereich Anzeige 1. Zur Kennzeichnung des 4.Viertel wird der Bereich &7700-&774C verwendet.

7550-755F P$
7560-756F Q$
7570-757F R$
7580-758F S$
7590-759F T$
75A0-75AF U$
75B0-75BF V$
75C0-75CF W$
75D0-75DF X$
75E0-75EF Y$
75F0-75FF Z$

7600-77FF Gleich dem Bereich &7400-&75FF

7800-784F CPU-Stack Dieser Bereich wird normalerweise von der CPU als Stapel gebraucht, ist daher nicht vor Datenverlust geschützt, auch darf man in diesen Bereich nichts einschreiben.

785B/.. Eingabe-Adr. Wenn <Eingabe-Flag> gesetzt ist, wird beim Aufruf des Unterprogrammes &E243 (Standard-Zeichen-Eingabe, wird vom PC-1500 z.B. bei INPUT oder bei der Eingabe eines Befehls im RUN-Mode ('>') genutzt, so das praktisch alle Eingaben über dieses Programm laufen.) zu der in dieser Speicherstelle abgelegten Adresse verzweigt. LSBit Dieses Wertes wird für das PV-Banking verwendet, so daß, wenn LSBit gesetzt ist, PV=1 gesetzt wird vor Aufruf des Programmes. Mit dieser Speicherstelle kann man z.B. eine Eingabe-Routine schaffen, die ein Auto-Repeat auf alle Tasten legt. Mit dem folgenden Programm wird eine Taste abgefragt und anschließend ein Bit gesetzt, daß dann ein Auto-Repeat ermöglicht: CALL (&E24A) : OR (&7B0E),&40 : RET Dieses Programm kann man mit POKE ADR,&BE,&E2,&4A,&EB,&7B,&E,&40,&9A (ADR ist eine geeignete freie Stelle im Speicher) und POKE &785B,ADR/256,ADR AND 255 und POKE &79D4,&55 einschalten. Jetzt gehen alle Eingaben zu diesem Programm und es haben fast alle Tasten Auto-Repeat.(&E24A wird aufgerufen, da es das gleiche wie &E243 ist, nur ohne der Umlenkung zur eigenen Eingabe.) Eingabe-Flag soll zuletzt gesetzt werden, da sonst die Eingabe schon umgelenkt wird, obwohl die restlichen Werte noch gar nicht stimmen und es so zu einem Aufhänger kommt.

785D Zs-Flag Aktivierung 2. Zeichensatz: 0=Anzeige+Drucker / &80=Anzeige / &FF=kein 2.Zeichensatz. Der Inhalt dieser Speicherstelle bleibt bei ON nicht erhalten. Mit dem 2.Zeichensatz kann man die Zeichen CHR$(&80)-CHR$(&FF) auf Anzeige und Drucker selbst definieren. Außerdem ist eine neue Tastaturbelegung möglich. Bei der Tastaturbelegung wird in der Anzeige das ht-Symbol gesetzt. Sollte dieses Zeichen erscheinen, ohne daß eine Tabelle für die 2.Tastaturbelegung angelegt ist, so kommen beim Drücken der Tasten nur falsche Zeichen. Man muß zweimal SMALL drücken, um dieses Zeichen zu löschen.

785E Sh MSByte der Startadressen der Tabellen für den 2.Zeichensatz. In diesen Tabellen ist abgelegt, wie die neuen Zeichen aussehen. Die Beschreibung des Aufbaus dieser Tabellen würde den Rahmen dieses Buches sprengen.

7860 Reserve neu MSByte der Startadresse des Reservespeichers Wenn dieses Byte nicht &FF ist, so wird der Inhalt aus &7863 ignoriert und stattdessen dieser Wert eingesetzt.

7861/.. Prganf neu Startadresse des BASIC-Programmspeichers Wenn diese Byte nicht &FF enthalten, so wird der Inhalt aus &7865/.. ignoriert und stattdessen dieser Wert eingesetzt.

7863 Reserve In dieser Speicherstelle steht das MSByte des Speicheranfangs.(Wird nicht mit NEW xx festgelegt.) Der Reservespeicher beginnt bei &100x<Reserve>+8.

7864 Varstart MSByte der Startadresse des Variablenspeichers+1, d.h. die höchste Benutzer-RAM-Adresse+1. Der Variablen Bereich wird von oben nach unten aufgebaut, darum heißt dieser Wert Variablen-Start. Dieser Wert wird nach ON wieder

gesetzt. Als LSByte wird immer 00 angenommen, so ergibt sich ein Wert, der 1 größer ist, als die höchste benutzbare Stelle. Bei CLEAR wird dieser Wert auch für das Ende des Variablenspeichers eingesetzt.

7865/.. Prganf Startadresse des BASIC-Programmes. Dieser Wert kann verändert werden mit NEW &xxyy oder auch per POKE &7865,&xx,&yy, wobei xx=MSByte und yy=LSByte des Wertes. Bei NEW wird dieser Wert als Startadresse angenommen und die anderen Werte danach ausgerichtet.

7867/.. Prgend Endadresse des BASIC-Programmes. Dieser Wert zeigt auf die Speicherstelle, in der &FF steht als Endkennzeichen eines Programmes. Dieser Wert kann verändert werden mit POKE &7867,&xx,&yy, wobei xx=MSByte und yy=LSByte der neuen Endadresse. Man kann damit Programmzeilen am Ende abschneiden (wenn man daß &FF wieder setzt) oder Teile dazunehmen. Dieser Wert und der Wert <Prganf> wird bei CSAVE als Adressen verwendet, wenn er also 1 Byte falsch steht (!), so kann es sein, daß ein Programm nicht mehr richtig läuft.

7869/.. Prgmrg Startadresse des letzten geMERGEten Programmes. Diese Speicherstelle wird als 1. Speicherstelle verwendet beim Editieren des Programmes. Man kann so z.B. nach MERGE auch das erste Programm wieder editieren, indem man POKE &7869,PEEK &7865,PEEK &7866 sagt, und so den Zeiger auf den Anfang des Programmspeichers legt. Will man ein bestimmtes Programm editieren, so sagt man: RUN "<Label>" BREAK POKE &7869,PEEK &789E,PEEK &799F <ENTER> Jetzt ist die Startadresse des betreffenden Programms als Such-Adresse eingestellt.

786B RMT/BEEP Remote (MSBit) und Beep (LSBit) an(=0) / aus(=1)

7871 Wait y/n Wait=0, Wait 0=3, Wait x=2

7872/.. Wait Counter Enthält den angegebenen WAIT-Wert aus BASIC.

7874 Cursor enable Wenn Wert in <Spaltennummer> bei Ausgabe auf der Anzeige verwendet werden soll, dann >0, sonst wird die Anzeige gelöscht und Spaltennummer 0 für die Ausgabe verwendet.

7875 Spaltennummer Enthält den nächsten zu verwendenen GCURSOR-Wert (oder CURSOR mal 6) für Ausgabe auf der Anzeige, wenn <Cursor enable> gesetzt ist.

787C Blink Flag Zustand des Cursors: aus=1 / an=&81 / nur Strich=0

787D Blink Char. ASCII-Wert des gerade blinkenden Zeichens (Cursor)

787E/.. Blink Cursor Spaltenadresse, wo der Cursor blinkt. Ab dieser Adresse wird abwechselnd ein Kästchen und das Zeichen aus <Blink Char> geschrieben.

788B Inputbuf.-Pt. Zeiger auf Input-Buffer

788D Trace TRON=1, TROFF=0

788E Trace Condit. Zustand, wenn Trace aktiv

788F Outp.Buff.Poi.Zeiger auf Ausgabe-Buffer

7890 For Pointer Stack-Pointer für For-Next-Schleifen

7891 Gosub Pointer Stack-Pointer für Gosub-Routinen

7894 Strng.Buf.Poi.Zeiger auf String-Buffer

7895 Using F/F USING-Format:

Bit7:'^',Bit6:'*',Bit5:'+',Bit4:',' Wenn die entsprechenden Bits gesetzt sind, wird das Format bei der Ausgabe berücksichtigt.

7896 Using M Länge des ganzahligen Wertes von USING mit Vorzeichenstelle

---

7897 Using & Länge des nichtnumerischen Wertes von USING (String).

7898 Using m Länge des Nachkommawertes von USING (mit Punkt)

7899/.. Variabl.Poin. Zeiger auf Endadresse des Variablen-Speichers, der von oben (&7864) nach hier aufgestockt ist. Wenn ein neues Feld dazukommt, so wird von hier in Richtung des

BASIC-Programm-Endes ein Feld reserviert und dieses in Richtung von BASIC-Programm nach hier aufgebaut.(STATUS 3)

789B Ern ERROR-Nummer

789C/.. Current Line Aktuelle Zeilennummer Wenn der Wert gleich null wird, stoppt der Interpreter das laufende Programm.

789E/.. Current Top Startadresse des aktuellen Programms Sind mehrere Programme mit MERGE zusammengeladen, so steht hier die Speicheradresse des Anfangs des laufenden Programmes. Die einzelnen Programmteile sind durch &FF voneinander getrennt.

78A0/.. Previous Adr. Speicheradresse des vorhergehenden Befehls

---

78A2/.. Previous Line Vorhergehende Zeilennummer

78A4/.. Previous Top Startadresse des vorhergehenden Programms

---

78A6/.. SEARCH Adres. Speicheradresse der bei dem Unterprogramm SEARCH im ROM (Wird bei GOTO,GOSUB,THEN,RESTORE verwendet) gefundenen Zeile.

78A8/.. SEARCH Line Zeilennummer die bei SEARCH gefunden wurde

---

78AA/.. SEARCH Top Startadresse des Programms, der die bei SEARCH gefundene Zeile enthält.(MERGE)

78AC/.. Break Address Speicheradresse, wo BREAK auftrat

78AE/.. Break Line Zeilennummer bei Break

78B0/.. Break Top Startadresse des laufenden Programms bei Break

---

78B2/.. Error Address Speicheradresse, wo ein Error auftrat

78B4/.. Error Line Zeilennummer, wo Error auftrat

78B6/.. Error Top Startadresse des Programms, wo Error auftrat (MERGE)

78B8/.. On Error Addr.Enthält ON ERROR GOTO-Zeilen-Adresse

78BA/.. On Error Line Enthält ON ERROR GOTO-Wert

78BC/.. On Error Top Startadresse des Programms, wohin ON ERROR GOTO verzweigt

---

78BE/.. Data Pointer Zeigt auf das nächste bei READ zu lesende Data-Element

---

```
78C0-78CF A$            Standartvariablen
78D0-78DF B$
78E0-78EF C$
78F0-78FF D$
7900-7907 A
7908-790F B
7910-7917 C
7918-791F D
7920-7927 E
7928-792F F
7930-7937 G
7938-793F H
7940-7947 I
7948-794F J
7950-7957 K
7958-795F L
7960-7967 M
7968-796F N
7970-7977 O
7978-797F P
7980-7987 Q
7988-798F R
7990-7997 S
7998-799F T
79A0-79A7 U
79A8-79AF V
79B0-79B7 W
79B8-79BF X
79C0-79C7 Y
79C8-79CF Z

----------------------------------------

79D0      Pv Banking    LSBit:Pv-Banking-Selektion

----------------------------------------

79D1                        Opn              Dv
Gerät:&60=LCD/&5C=CMT/&58=MGP/&C4=LPRT/&C0=COM

----------------------------------------

79D3      Modulation     Wenn=&55, bei CSAVE oder PRINT # keine
Modulation
```

79D4 Eingabe-Flag Wenn=&55, dann verzweigt das Unterprogramm &E243 (Standart-Eingabe eines Zeichen) zu der in <Eingabe-Adr> angegebenen Adresse

---

79DA User IRQ Wenn=&55, dann Verzweigung bei Interrupt durch Break-Taste in Maschinenprogramm, wenn I=1, zu der in <Usr-Irq-Adr> angegebenen Speicheradresse. Setzt man in seinem M-Programm das I-Flag der CPU und im PC-1500-Port das richtige Bit für die Break-Taste, so wird durch die Break-Taste ein Interrupt erzeugt, der in der Interrupt-Verarbeitungsroutine eine Verzweigung erzeugt.

79DB/.. User-IRQ-Adr Speicheradresse des Programms für User-Interr-Verarbeitung. LSBit gibt an, ob PV-Banking durchgeführt werden soll. Es wird also bei allen ungeraden Adresse PV=1 gesetzt.

79E0/.. Usr Counter X Zähler für X-Koordinate des Zeichenstiftes

---

79E2/.. Usr Counter Y Zähler für Y-Koordinate des Zeichenstiftes

---

79E4/.. Scisso.Coun.Y Y-Abschnittszähler

79E6 Abs.Pos.X Zeichenstift-Position horizontal Wenn Sie diesen Zähler verändern, wird der Zeichenstift nicht mehr am Rand gestoppt, da das Koordinatensystem verschoben wird.

79E7/.. Scisso.Coun.X X-Abschnittszähler

79EA Line Type Enthält den gewählten Linientyp bei LINE oder RLINE.

79EB Dot Line Coun. Punktzähler bei gestrichelten Linien

79EC Pen up/down Flag, ob Zeichenstift angehoben oder abgesetzt ist beim Zeichnen gestrichelter Linien

---

79ED X Motor hold Counter

79EE Port C Aktuelle Motorphase

79EF Y Motor hold Counter

79F0 Graph/Text Graph=&FF/Text=0-Mode

79F1 Power Stromversorgung von CE-150 ausreichend =0 / nicht =&FF

79F2 Rotate Enthält ROTATE-Wert

79F3 Color Enthält COLOR-Wert

79F4 Csize Enthält CSIZE-Wert

79FF Lock Lock=0/Unlock=&FF

7A00-7A07 Ar-X Arithmetikregister X Diese Arithmetikregister werden als Zwischenspeicher bei Rechenoperationen verwendet. In jedem Register kann eine Zahl mit Exponent und Vorzeichen abgelegt werden.
7A08-7A0F Ar-Z Arithmetikregister Z
7A10-7A17 Ar-Y Arithmetikregister Y
7A18-7A1F Ar-U Arithmetikregister U
7A20-7A27 Ar-V Arithmetikregister V
7A28-7A2F Ar-W Arithmetikregister W
7A30-7A37 Ar-S Arithmetikregister S

7A38-7AFF Basic-Stack For-Next- / Gosub- / Data- / Function-Stack

---

7B00-7B07 Rnd-Reg Enthält Zufallszahl

7B0A-7B0C Aut-Pow-Of.ct Timer für automatisches Auto-Power-Off

---

7B0E Autorepeat Bit 6:Autorepeat ein=1/aus=0 für letztes Zeichen

---

7B10-7B5F String-Buf Buffer für Stringverarbeitung
7B60-7BAF Output-Buf Ausgabe-Zwischenspeicher
7BB0-7BFF Input-Buf Eingabe-Zwischenspeicher

7C00-7FFF Gleich dem Bereich &7800-&7BFF

Weitere wichtige Speicherplätze:

Wenn die Speicherstelle &0000 den Wert &55 enthält und als ROM geschaltet ist (CE-161), so gilt folgende Belegung:

0001 MSByte der Reserve-Speicher-Adresse
0002/..BASIC-Startadresse
0004 Gibt Größe des Speichers an: 1K = &4 / 2K = &8 / 4K = &10 / 8K = &20 / 16K = &40
0007 Wenn vertrauliches Programm, dann =0, sonst &FF

Diese Werte werden nach dem Einschalten mit ON geprüft und evtl. angenommen als neue Startadressen. Diese Speicherstellen werden bei Programm-Modulen verwendet, so daß das BASIC-Programm des Benutzers unversehrt bleibt und nach dem Herausziehen wieder verfügbar ist.

Aufbau des Reserve-Speicher-Bereiches:

Die Startadresse bildet sich aus <Reserve> &7863 für das MSByte und &08 als LSByte.

XX08-XX21 Text für 1. Ebene, der beim Umschalten erscheint
XX22-XX3B Text für 2. Ebene, der beim Umschalten erscheint
XX3C-XX55 Text für 3. Ebene, der beim Umschalten erscheint
XX56-XXC4 Reserve-Tasten-Belegung

Die Reserve-Tasten-Belegung ist folgend aufgeschlüsselt:

Die Tasten-Belegungen werden in Reihenfolge der Eingabe abgelegt, wobei bei Neueingaben die alte Belegung gelöscht und die neue angehängt wird. Eine Tasten-Belegung beginnt mit dem Tasten-Code und enthält dann den Text, der dieser Taste zugeordnet wird. Abschluß der Liste bildet ein Byte mit 0 als Inhalt.

Tasten-Code:

Taste	F1	F2	F3	F4	F5	F6
Ebene I	&1	&2	&3	&4	&5	&6
Ebene II	&11	&12	&13	&14	&15	&16
Ebene III	&9	&A	&B	&C	&D	&E

Aufbau des Programmspeichers:

Die Startadresse bildet sich aus <Prganf> &7865 als MSByte und &7866 als LSByte. Ab dieser Adresse ist das BASIC-Programm abgelegt:

Aufbau einer Programmzeile:

XX YY ZZ BASIC-Programm-Text bestehend aus Token und ASCII-Zeichen
0D

XX ist das MSByte der Zeilennummer (0<= XX <= 254)
YY ist das LSByte der Zeilennummer
Die Zeilennummer darf nicht 0 sein, da das System dies nicht zuläßt.

ZZ ist die Länge des BASIC-Programm-Textes mit dem Abschluß-Zeichen.
0D ist der ASCII-Wert für <ENTER> bzw. <RETURN> und bildet das Endzeichen der Zeile.

Aufbau eines BASIC-Programmes:

Ein BASIC-Programm besteht aus einer Aneinander-Kettung von Programmzeilen. Als Abschluß eines Programmes dient ein Byte mit dem Inhalt &FF nach der letzten Zeile. Die Zeilennummern werden bei der Eingabe der Größe nach geordnet. Solche Programme können aneinander gefügt und separat gestartet werden (siehe auch Bedienungsanleitung). Wenn das System beim Durchsuchen des Speichers nach einer Programmzeile auf ein Byte mit dem Inhalt &FF trifft, so wird überprüft, ob die momentane Speicheradresse noch kleiner als der Wert aus <Prgend> ist. Ist dies nicht der Fall wird mit der Suche abgebrochen. Falls doch, so wird das Byte übersprungen. Diese Marken werden bei Sprüngen in andere Programm-Module mit GOTO "Label" berücksichtigt und als <Current Top> abgelegt, wenn die gefundene Zeile nach einem solchen Byte gefunden wird.

Aufbau des Variablenspeichers:

Die Startadresse bildet sich aus <Variabl.Point> (STATUS 3) &7899 als MSByte und &789A als LSByte, da der Speicher von oben nach unten aufgestockt wird. Ab dieser Adresse sind Variablen abgelegt in folgendem Format:

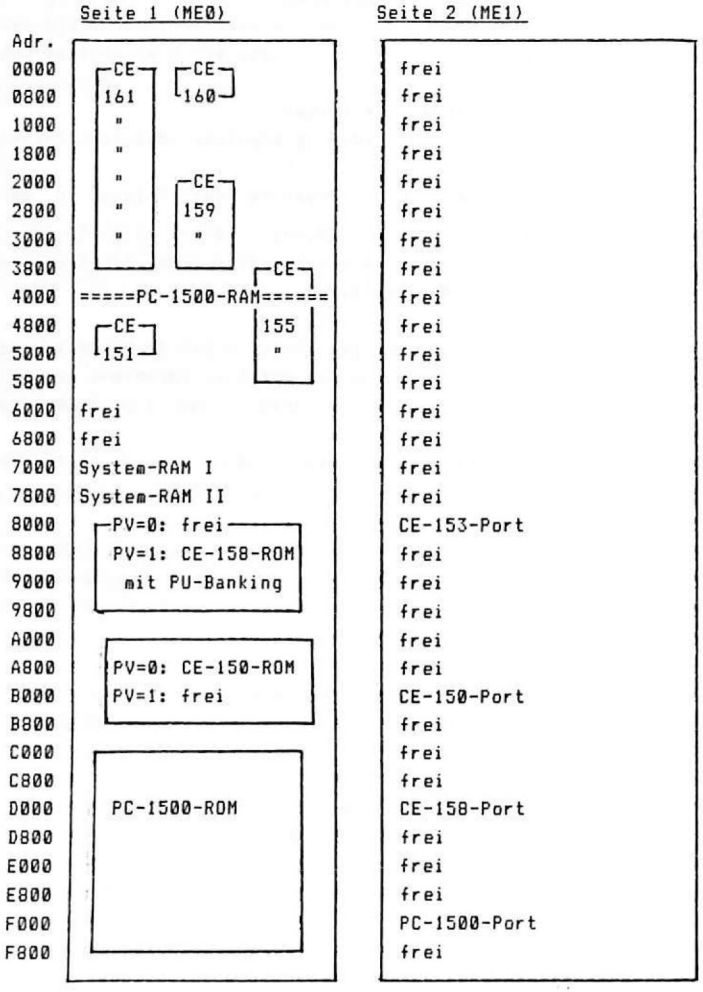
Adresse enthält 1.Buchstaben der Variable/Feldes
Adresse+1 enthält 2.Buchstaben der Variable/Feldes, mit: MSBit:1=Feld/0=Variable Bit5:1=String/0=numerisch
Adresse+2/.. enthält die Länge der Felddaten+3
Adresse+4 enthält zweiten Index des Feldes. Wird ein Feld nur eindimensional definiert, so wird er 1. Index mit 0 angenommen und der angegebene Index hier abgelegt.
Adresse+5 enthält ersten Indes des Feldes.
Adresse+6 enthält die Länge jedes Elementes. Bei numerischen Elementen wird hier 88 eingetragen.
Adresse+7 hier beginnen die Elemente des Feldes in der Reihenfolge: (0,0) (0,1) (0,2) .. (0,n-1) (0,n) (1,0) (1,1) .. (m,n-1) (m,n) wobei m der erste und n der zweite Index ist.Einfache Variablen werden mit Index 0,0 eingetragen.

Man kann aus einem dimensionierten Feld (DIM A$(0)*80) eine einfache Variable machen, indem man das MSBit in Adresse+1 löscht. Diese Variable hat nun die dimensionierte Länge (auch über 16 Zeichen).

Solche Variablen/Feld-Dimensionen sind aneinander gereiht und reichen bis <Varstart>-1.

Speicheraufteilung des PC-1500:

Angegeben ist jeweils die erste Adresse eines 2K-Blocks



ROM-Programme

Wichtige Unterprogramme aus dem ROM des SHARP PC-1500:

Aufbau der Liste:

Gerätename: Plotter / Display / PC-1500-System / Tastatur

Aufruf:&XXXX gibt hexadezimal den Einsprungspunkt an (CALL &XXXX). (&XX) gibt den Einsprungpunkt über die letzte Seite an (CALL (&XX)). Es gibt Unterprogramme, die einen oder mehrere Parameter verlangen, die folgendermaßen angegeben werden:

```
CALL (&XX)
P1          ;P1 ist ein Byte
```

Die entsprechenden Unterprogramme springen beim Rücksprung hinter die Parameter (hier nur einer).

Bei Unterprogrammen im Drucker sind zwei Adressen angegeben, denn es gibt zwei Versionen des Drucker-ROMs. Die erste Zahl bezieht sich auf die erste Version, die zweite auf die andere Version des Druckers. Sie können die Version Ihres Drucker feststellen, indem Sie nachsehen, welcher Wert in der Speicherstelle &A800 steht. Bei &44 haben Sie die erste Version, bei &BE die zweite.

Register: 8-Bit: A Acku XH,XL,YH,YL,UH,UL 8-Bit Register der CPU
16-Bit: X,Y,U 16-Bit Register der CPU
Flags: Es gibt die Flags Carry,Zero,Halfcarry,Interrupt,Overflow der CPU

Speicherplätze: Werden von Programmen bestimmte Speicherplätze bearbeitet, so sind diese angegeben. Es gibt folgende Speicherplätze (besonders bei Anzeige), die hier gesondert genannt werden sollen:
-Spaltenpointer (&7875): Dieser Speicher gibt die nächste gültige Spaltennummer an (0-155). In Verbindung hierzu gibt es die Spaltenadresse. Diese Adresse zeigt in den Speicherbereich der LCD-Anzeige und dient dem Ansprechen einzelner Spalten. Der Adressbereich dieses Wertes kann &7400-&774C annehmen, mit kleinen Unterbrechungen. Bei Druckerroutinen wird der Bereich ab &7B7F zur übergabe von Parametern genutzt. Bei diesen Routinen sollte X auf diesen Bereich zeigen.

Tast. &E243 X,U,A geändert
Wartet, bis über die Tastatur ein Zeichen eingegebe wird. Dieses Zeichen wird in A abgelegt. Wenn Break gedrückt wurde, ist das C-Flag der CPU gesetzt. Diese Routine achtet auch auf eine eigene Eingaberoutine und greift auch Eingabe-Flag zu, um eine solche Umlenkung zu erkennen. Diese Umlenkung darf nicht Y verändern.

Tast. (&A6) &E451 keine Register geändert
Prüft, ob die Taste BREAK gedrückt ist. Wenn Ja, dann ist Z=0, sonst =1. Dieses Programm greift auf die Speicherstelle &F00B der 2.Speicherseite des PC-1500 zu. In dieser Speicherstelle steht in Bit 1, ob die BREAK-Taste gedrückt war.

Disp. &EE22 A,X geändert
Berechnet aus der Spaltennummer, die in A übergeben wird die Spaltenadresse und legt diese in X ab.

Disp. (&8C) &EE1F A,X geändert
Berechnet aus der Spaltennummer, die im Spaltenpointer (&7875) übergeben wird, die Spaltenadresse und legt diese in X ab.

Disp. (&8E) &EDB1 A geändert
Inkrementiert Spaltenpointer (&7875), solange dieser kleiner 156 ist, sonst wird das CPU-Flag C=1 gesetzt. Das Ergebnis wird wieder im Spaltenpointer abgelegt, wo er für weitere Ausgaben zur Verfügung steht. (Der Wert steht auch noch in A)

Disp. (&F2) &EE71 A,U geändert
Löscht LCD-Anzeige (CLS). Diese Routine setzt den Spaltenpointer (&7875) nicht auf 0.

Disp. (&88) &EDF6 A,X,UH geändert
In A wird ein Wert übergeben, der codiert ist wie beim GPRINT X-Kommando in BASIC. Dieser Wert wird in die Spalte auf der Anzeige gesetzt, deren Adresse (!) in X übergeben wird. Nach der Anzeige der Grafik-Spalte wird X mit der Adresse der nächsten Spalte geladen. Wird der rechte Rand der Anzeige überschritten wird nicht automatisch am linken Rand wieder angefangen.

Disp. &EDEF A,X,UH geändert
Gleich dem Programm (&88), nur wird die Spaltenadresse aus der Spaltennummer in &7875 errechnet. Nach der Rückkehr steht die nächste Spaltenadresse in X.

Disp. (&92) &ED00 A,X,U geändert
Ausgabe von A (Anzahl steht in A) Buchstaben auf der Anzeige. Diese Buchstaben stehen im Speicher ab der Adresse, die in U übergeben wird. Die Ausgabe der Zeichen beginnt ab der Spalte, deren Nummer im Spaltenpointer (&7875) angegeben ist. Können nicht alle Zeichen auf der Anzeige untergebracht werden, so steht in A die Anzahl der verbleibenden Zeichen.

Disp. (&8A) &ED5B A,X,U geändert
Es wird ein Zeichen auf dem Display ab der Spaltenadresse in X ausgegeben. In X steht nach der Rückkehr die Adresse für das nächste Zeichen. Eine Überschreitung des rechten Randes wird nicht berücksichtigt.

Disp. &EE48 A,Y geändert
Berechnet für den Buchstaben, dessen ASCII-Wert in A übergeben wird die Anfangsadresse seiner Tabelle, in der fünf GPRINT-Werte stehen, die nacheinander das Zeichen ergeben. Diese Adresse wird in Y abgelegt.

Plot. ?/&A306 A geändert
Schaltet Papiervorschub-Taste aus. Dieses Programm sollte vor anderen Plotter-Programmen aufgerufen werden.

Plot. &A747/&A769 A geändert
Schaltet Motor aus. Dieses Programm sollte nach anderen Plotter-Programmen aufgerufen werden !!!!

Plot. &A9CB/&A9F1 A,U geändert
Druckstift wird an den Anfang der nächsten Zeile gefahren.X muß vor Aufruf des Programmes auf einen Bereich zeigen, der von X-10 bis X+1 zerstört werden kann (&7A22).

Plot. &A4F7/&A519 A,UH,XL geändert
Wählt Farbe des Zeichenstiftes, die in UL steht. (COLOR)

Plot. &A8B7/&A8DD A,X,Y,U geändert
X muß auf einen Speicherbereich zeigen, in dem die Bewegung des Zeichenstiftes angegeben ist als 16 Bit-Zahl:
(X+0)/(X+2) ergibt den relativen Wert in X-Richtung,
(X+1)/(X+3) ergibt den relativen Wert in Y-Richtung,
die der Zeichenstift bewegt wird. Negative Werte werden als Zweierkomplement angegeben.

Plot. &AABD/&AAE3 A geändert
Der Zeichenstift wird entsprechend dem Inhalt der Speicherstelle &79E9 angehoben (=&00) oder abgesetzt auf das Papier (=&FF).

Plot. ?/&B4F4 A,X,Y,UH geändert
Ausgabe von UL Zeichen auf dem Plotter ab der momentanen Stiftposition. Am Zeilenende wird automatisch an den Anfang der nächsten Zeile gefahren. Die Zeichen müssen im Speicher ab der Stelle stehen, deren Adresse in X übergeben wird. Steht in der

Zeichenfolge ein <Return CHR$(13)> oder ein <Nul CHR$(0)>, so wird die Ausgabe abgebrochen.

Plot. &A75B/&A781 A,X,Y geändert
Es wird ein Zeichen ausgegeben, das in der Speicherstelle steht, deren Adresse in X abgelegt ist. Am Zeilenende wird kein Zeilenvorschub ausgeführt, sondern es wird weitergeschrieben. Es wird der Bereich von X-10 bis X+1 zerstört.

Plot. &A9DE/&AA04 A,X,Y,U geändert
Das Papier wird um eine Anzahl von Zeilen eingezogen bzw. ausgerollt. Die Anzahl der Zeilen wird im Speicher als 16 Bit-Wert abgelegt, wobei eine negative Anzahl als Zweierkomplement anzugeben ist. Die Adresse dieses Speicherplatzes wird in X übergeben.

Rec. &BF23 A,U geändert
Die Remote-Leitungen der Recorder 0 und 1 werden in Abhängigkeit von A geschaltet:
A=&03: Recorder 0 ein
A=&05: Recorder 0 aus
A=&09: Recorder 1 ein
A=&11: Recorder 1 aus

1500 &E66F A geändert
Es wird ein Ton generiert, dessen Tonhöhe in UL und dessen Länge+&100 in X übergeben wird.

1500 &E669 A,X,UL geändert
Es wird ein Standart-Ton generiert (BEEP 1).

1500 &E88C A,U geändert
Es wird eine Zeitverzögerung um U*15.625 ms durchgeführt. Mit BREAK kann der Vorgang abgebrochen werden.

1500 &E33F A,X,U geändert
Der PC-1500 wird ausgeschaltet wie bei Auto-Power-Off. Der Computer kann mit der Taste ON wieder angeschaltet werden ohne Druckerinitialisierung und Anzeige-Löschen. Das Programm verzweigt dann nach &E243, der Standart-Zeicheneingabe, von wo der Rücksprung in Ihr Programm erfolgt. Der Wert der Taste ist dann in A abgelegt.

1500 &CA58
Sprung in den BASIC-Interpreter.

1500 (&12) &DF93 A,X geändert
Die Startadresse des BASIC-Programm-Bereiches wird in X geladen.

1500 (&14) &DFFA A,X,U geändert
Die Startadresse des BASIC-Programm-Bereiches wird in X und die Länge des dort abgelegten Programmes in U abgelegt.

1500 (&84) &EF00
Schaltet blinkenden Cursor aus.

1500 (&CA)P1 A,U geändert
Der Wert aus X wird in die Speicherstellen &78XX (MSByte) und &78XX+1 (LSByet) abgelegt. XX wird dabei von dem Parameter festgelegt, der direkt dem Aufrufkommando folgt.

1500 (&CC)P1 A,U,X geändert
Der Wert aus den Speicherstellen &78XX (MSByte) und &78XX+1 (LSByte) wird in X abgelegt. XX wird dabei von dem Parameter festgelegt, der direkt dem Aufrufkommando folgt.

1500 (&50) &DA71 X,Y geändert
Es wird U mit Y multipliziert. Ist das Ergebniss größer als ein Wert, der mit 16 Bit dargestellt werden kann, so wird das C-Flag der CPU gesetzt, sonst wird das Ergebnis in X und Y abgelegt.

1500 (&38) &CE9F X,UH,A geändert
Die Reservespeicher-Startadresse wird in X abgelegt.

1500 (&F4)P1,P2 U,A geändert
Es wird der 16-Bit-Wert aus der Speicherstelle &XXYY (MSByte) und &XXYY+1 (LSByte) in U geladen. XX wird durch P1, YY durch P2 festgelegt, die direkt dem Aufrufkommado folgen müssen.

1500 (&F6)P1,P2 A,U geändert
Es wird der 16-Bit-Wert aus U in die Speicherstellen &XXYY (MSByte) und &XXYY+1 (LSByte) abgelegt. XX wird durch P1, YY durch P2 festgelegt, die direkt dem Aufrufkommando folgen müssen.

1500 (&16) &DFF5 A,U geändert
Es wird die Differenz zwischen X und dem Ende des BASIC-Programmes gebildet und in U abgelegt, solange X kleiner dem Ende des BASIC-Programmes ist. Ist z.B. X gleich der BASIC-Programm-Anfangsadresse, so ist der erhaltene Wert gleich STATUS 1-2.

1500 (&10)P1 A,X,Y,U geändert
In Abhängigkeit von P1 werden folgende Operationen ausgeführt:
P1=&00:Der Wert aus U wird in AR-X im BCD-Format abgelegt.
P1=&40:Der Wert aus U wird ab der durch Y festgelegten Adresse in ASCII als Dezimalzahl abgelegt. Y zeigt nach dem Rücksprung auf auf die nächste freie Stelle nach der abgespeicherten Zahl.
P1=&80:Der Wert aus U wird in AR-X als numerischer Wert abgelegt, dabei wird das Vorzeichen (Bit 15) beachtet. Ist es =1, so ist die

Zahl negativ, d.h. das Zweierkomplement (!) der positiven Zahl.
P1=&E0:Wie bei P1=&40, nur ist ein führendes Leerzeichen vorhanden, und die maximale Anzahl von Ziffern ist 2.

1500 (&B2) &B897
Es wird getestet, ob U = &FFFF ist. Wenn ja, dann wird Z-Flag der CPU gesetzt.

Token

Liste aller bisher bekannten Token für den PC-1500:

Angegeben ist jeweils das Befehlswort, der interne Token, soweit bekannt, und das Gerät, das diesen Token enthält. Diese Liste ist nach den internen Codes sortiert.

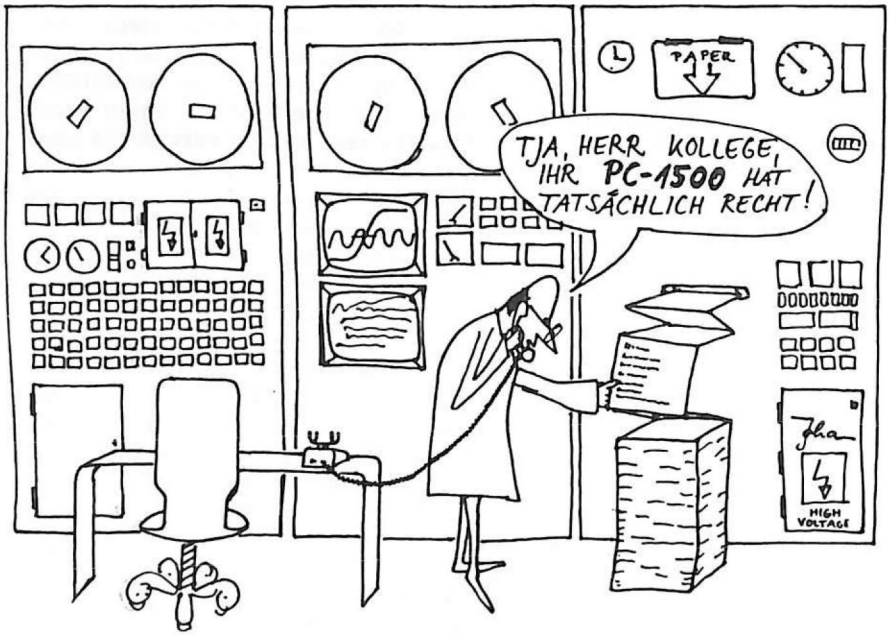
CLR	?	ABM-0	CSIZE	E680	CE-150	LEN	F164	PC-1500
BELL	?	ABM-0	GRAPH	E681	CE-150	DEG	F165	PC-1500
BCD$	?	ABM-0	GLCURSORE	682	CE-150	DMS	F166	PC-1500
FAC	?	ABM-0	LCURSOR	E683	CE-150	STATUS	F167	PC-1500
PGM	?	TOOL3	SORGN	E684	CE-150	POINT	F168	PC-1500
VLIST	?	TOOL3	ROTATE	E685	CE-150	SQR	F16B	PC-1500
ERN	?	TOOL3	TEXT	E686	CE-150	NOT	F16D	PC-1500
MOVE	?	ABM-0	RMT	E7A9	CE-150	PEEK#	F16E	PC-1500
REDIM	?	TOOL3	DEV$	E857	CE-158	PEEK	F16F	PC-1500
LINK	?	TOOL1	COM$	E858	CE-158	ABS	F170	PC-1500
SPLIT	?	TOOL1	INSTAT	E859	CE-158	INT	F171	PC-1500
FRAC	?	ABM-0	RINKEY$	E85A	CE-158	RIGHT$	F172	PC-1500
PPOKE	?	ABM-0	OUTSTAT	E880	CE-158	ASN	F173	PC-1500
COFF	?	ABM-0	SETCOM	E882	CE-158	ACS	F174	PC-1500
WRITE	?	ABM-0	TERMINALE	883	CE-158	ATN	F175	PC-1500
GETKEY	?	ABM-0	DTE	E884	CE-158	LN	F176	PC-1500
PURGE	?	TOOL3	TRANSMITE	885	CE-158	LOG	F177	PC-1500
VKEEP	?	TOOL3	SETDEV	E886	CE-158	EXP	F178	PC-1500
INSTR	?	TOOL3	ERN	F052	CE-158	SGN	F179	PC-1500
CLR	?	TOOL3	ERL	F053	CE-158	LEFT$	F17A	PC-1500
RESUME	?	TOOL3	SPACE$	F061	CE-158	MID$	F17B	PC-1500

FRE	?	TOOL3	DEC	F070	TOOL3	RND	F17C PC-1500
PCALL	?	ABM-0	HEX$	F071	TOOL3	SIN	F17D PC-1500
SWAP	?	TOOL3	CURSOR	F084	PC-1500	COS	F17E PC-1500
PTR$	?	ABM-0	USING	F085	PC-1500	TAN	F17F PC-1500
PSIZE	?	TOOL3	CLS	F088	PC-1500	AREAD	F180 PC-1500
FRC	?	TOOL3	CLOAD	F089	CE-150	ARUN	F181 PC-1500
STA$	?	ABM-0	MERGE	F08F	CE-150	BEEP	F182 PC-1500
PPEEK	?	ABM-0	LIST	F090	CE-150	CONT	F183 PC-1500
ERL	?	TOOL3	INPUT	F091	PC-1500	GRAD	F186 PC-1500
LBYTE	?	ABM-0	GCURSOR	F093	PC-1500	CLEAR	F187 PC-1500
BDEC	?	ABM-0	CSAVE	F095	CE-150	CALL	F18A PC-1500
HLIST	?	ABM-0	PRINT	F097	PC-1500	DIM	F18B PC-1500
HHEX$	?	ABM-0	GPRINT	F09F	PC-1500	DEGREE	F18C PC-1500
STRING$	?	TOOL3	FEED	F0B0	CE-158	DATA	F18D PC-1500
HBYTE	?	ABM-0	CONSOLE	F0B1	CE-158	END	F18E PC-1500
HTAN	?	RWEBASIC	CHAIN	F0B2	CE-150	GOTO	F192 PC-1500
WHILE	?	RWEBASIC	ZONE	F0B4	CE-158	GOSUB	F194 PC-1500
ELSE	?	RWEBASIC	COLOR	F0B5	CE-150	IF	F196 PC-1500
ANGLE	?	RWEBASIC	LF	F0B6	CE-150	LET	F198 PC-1500
IF#	?	RWEBASIC	LINE	F0B7	CE-150	RETURN	F199 PC-1500
SUBCLR	?	RWEBASIC	LLIST	F0B8	CE-150	NEXT	F19A PC-1500
GSB	?	RWEBASIC	LPRINT	F0B9	CE-150	NEW	F19B PC-1500
HACS	?	RWEBASIC	RLINE	F0BA	CE-150	ON	F19C PC-1500
CASE	?	RWEBASIC	TAB	F0BB	CE-150	OPN	F19D PC-1500
FAC	?	RWEBASIC	TEST	F0BC	CE-150	OFF	F19E PC-1500
ROUND	?	RWEBASIC	APPEND	F0C0	TOOL1	POKE#	F1A0 PC-1500
HSIN	?	RWEBASIC	CHANGE	F0C1	TOOL1	POKE	F1A1 PC-1500
SUB	?	RWEBASIC	DELETE	F0C2	TOOL1	PAUSE	F1A2 PC-1500
FN	?	RWEBASIC	ERASE	F0C3	TOOL1	RUN	F1A4 PC-1500
ENDIF	?	RWEBASIC	FIND	F0C4	TOOL1	FOR	F1A5 PC-1500
HCOS	?	RWEBASIC	KEEP	F0C5	TOOL1	READ	F1A6 PC-1500
DO	?	RWEBASIC	PLIST	F0C6	TOOL1	RESTORE	F1A7 PC-1500
SUBEND	?	RWEBASIC	PROGRAM	F0C7	TOOL1	RANDOM	F1A8 PC-1500
SELECT	?	RWEBASIC	RENUMBER	F0C8	TOOL1	RADIAN	F1AA PC-1500
HATN	?	RWEBASIC	PLAST	F0C9	TOOL1	REM	F1AB PC-1500
ENDSELE.	?	RWEBASIC	AND	F150	PC-1500	STOP	F1AC PC-1500
DEFFN	?	RWEBASIC	OR	F151	PC-1500	STEP	F1AD PC-1500
INTEGRAL	?	RWEBASIC	MEM	F158	PC-1500	THEN	F1AE PC-1500
LOOP	?	RWEBASIC	TIME	F15B	PC-1500	TRON	F1AF PC-1500
HASN	?	RWEBASIC	INKEY$	F15C	PC-1500	TROFF	F1B0 PC-1500
EXIT	?	RWEBASIC	PI	F15D	PC-1500	TO	F1B1 PC-1500
FLOAD	E180	TOOL2	ASC	F160	PC-1500	WAIT	F1B3 PC-1500
FSAVE	E181	TOOL2	STR$	F161	PC-1500	ERROR	F1B4 PC-1500
FCHAIN	E182	TOOL2	VAL	F162	PC-1500	LOCK	F1B5 PC-1500
VERIFY	E183	TOOL2	CHR$	F163	PC-1500	UNLOCK	F1B6 PC-1500

Diese Liste ist nach den Befehlswörtern sortiert.

ABS	F170	PC-1500	FRE	?	TOOL3	POKE	F1A1	PC-1500
ACS	F174	PC-1500	FSAVE	E181	TOOL2	POKE#	F1A0	PC-1500
AND	F150	PC-1500	GCURSOR	F093	PC-1500	PPEEK	?	ABM-0
ANGLE	?	RWEBASIC	GETKEY	?	ABM-0	PPOKE	?	ABM-0
APPEND	F0C0	TOOL1	GLCURSOR	E682	CE-150	PRINT	F097	PC-1500
AREAD	F180	PC-1500	GOSUB	F194	PC-1500	PROGRAM	F0C7	TOOL1
ARUN	F181	PC-1500	GOTO	F192	PC-1500	PSIZE	?	TOOL3
ASC	F160	PC-1500	GPRINT	F09F	PC-1500	PTR$	?	ABM-0
ASN	F173	PC-1500	GRAD	F186	PC-1500	PURGE	?	TOOL3
ATN	F175	PC-1500	GRAPH	E681	CE-150	RADIAN	F1AA	PC-1500
BCD$	?	ABM-0	GSB	?	RWEBASIC	RANDOM	F1A8	PC-1500
BDEC	?	ABM-0	HACS	?	RWEBASIC	READ	F1A6	PC-1500
BEEP	F182	PC-1500	HASN	?	RWEBASIC	REDIM	?	TOOL3
BELL	?	ABM-0	HATN	?	RWEBASIC	REM	F1AB	PC-1500
CALL	F18A	PC-1500	HBYTE	?	ABM-0	RENUMBER	F0C8	TOOL1
CASE	?	RWEBASIC	HCOS	?	RWEBASIC	RESTORE	F1A7	PC-1500
CHAIN	F0B2	CE-150	HEX$	F071	TOOL3	RESUME	?	TOOL3
CHANGE	F0C1	TOOL1	HHEX$	?	ABM-0	RETURN	F199	PC-1500
CHR$	F163	PC-1500	HLIST	?	ABM-0	RIGHT$	F172	PC-1500
CLEAR	F187	PC-1500	HSIN	?	RWEBASIC	RINKEY$	E85A	CE-158
CLOAD	F089	CE-150	HTAN	?	RWEBASIC	RLINE	F0BA	CE-150
CLR	?	ABM-0	IF	F196	PC-1500	RMT	E7A9	CE-150
CLR	?	TOOL3	IF#	?	RWEBASIC	RND	F17C	PC-1500
CLS	F088	PC-1500	INKEY$	F15C	PC-1500	ROTATE	E685	CE-150
COFF	?	ABM-0	INPUT	F091	PC-1500	ROUND	?	RWEBASIC
COLOR	F0B5	CE-150	INSTAT	E859	CE-158	RUN	F1A4	PC-1500
COM$	E858	CE-158	INSTR	?	TOOL3	SELECT	?	RWEBASIC
CONSOLE	F0B1	CE-158	INT	F171	PC-1500	SETCOM	E882	CE-158
CONT	F183	PC-1500	INTEGRAL	?	RWEBASIC	SETDEV	E886	CE-158
COS	F17E	PC-1500	KEEP	F0C5	TOOL1	SGN	F179	PC-1500
CSAVE	F095	CE-150	LBYTE	?	ABM-0	SIN	F17D	PC-1500
CSIZE	E680	CE-150	LCURSOR	E683	CE-150	SORGN	E684	CE-150
CURSOR	F084	PC-1500	LEFT$	F17A	PC-1500	SPACE$	F061	CE-158
DATA	F18D	PC-1500	LEN	F164	PC-1500	SPLIT	?	TOOL1
DEC	F070	TOOL3	LET	F198	PC-1500	SQR	F16B	PC-1500
DEFFN	?	RWEBASIC	LF	F0B6	CE-150	STA$	?	ABM-0
DEG	F165	PC-1500	LINE	F0B7	CE-150	STATUS	F167	PC-1500
DEGREE	F18C	PC-1500	LINK	?	TOOL1	STEP	F1AD	PC-1500
DELETE	F0C2	TOOL1	LIST	F090	CE-150	STOP	F1AC	PC-1500
DEV$	E857	CE-158	LLIST	F0B8	CE-150	STR$	F161	PC-1500

DIM	F18B	PC-1500	LN	F176	PC-1500	STRING$	?	TOOL3
DMS	F166	PC-1500	LOCK	F1B5	PC-1500	SUB	?	RWEBASIC
DO	?	RWEBASIC	LOG	F177	PC-1500	SUBCLR	?	RWEBASIC
DTE	E884	CE-158	LOOP	?	RWEBASIC	SUBEND	?	RWEBASIC
ELSE	?	RWEBASIC	LPRINT	F0B9	CE-150	SWAP	?	TOOL3
END	F18E	PC-1500	MEM	F158	PC-1500	TAB	F0BB	CE-150
ENDIF	?	RWEBASIC	MERGE	F08F	CE-150	TAN	F17F	PC-1500
ENDSELE.	?	RWEBASIC	MID$	F17B	PC-1500	TERMINALE	883	CE-158
ERASE	F0C3	TOOL1	MOVE	?	ABM-0	TEST	F0BC	CE-150
ERL	?	TOOL3	NEW	F19B	PC-1500	TEXT	E686	CE-150
ERL	F053	CE-158	NEXT	F19A	PC-1500	THEN	F1AE	PC-1500
ERN	?	TOOL3	NOT	F16D	PC-1500	TIME	F15B	PC-1500
ERN	F052	CE-158	OFF	F19E	PC-1500	TO	F1B1	PC-1500
ERROR	F1B4	PC-1500	ON	F19C	PC-1500	TRANSMITE	885	CE-158
EXIT	?	RWEBASIC	OPN	F19D	PC-1500	TROFF	F1B0	PC-1500
EXP	F178	PC-1500	OR	F151	PC-1500	TRON	F1AF	PC-1500
FAC	?	ABM-0	OUTSTAT	E880	CE-158	UNLOCK	F1B6	PC-1500
FAC	?	RWEBASIC	PAUSE	F1A2	PC-1500	USING	F085	PC-1500
FCHAIN	E182	TOOL2	PCALL	?	ABM-0	VAL	F162	PC-1500
FEED	F0B0	CE-158	PEEK	F16F	PC-1500	VERIFY	E183	TOOL2
FIND	F0C4	TOOL1	PEEK#	F16E	PC-1500	VKEEP	?	TOOL3
FLOAD	E180	TOOL2	PGM	?	TOOL3	VLIST	?	TOOL3
FN	?	RWEBASIC	PI	F15D	PC-1500	WAIT	F1B3	PC-1500
FOR	F1A5	PC-1500	PLAST	F0C9	TOOL1	WHILE	?	RWEBASIC
FRAC	?	ABM-0	PLIST	F0C6	TOOL1	WRITE	?	ABM-0
FRC	?	TOOL3	POINT	F168	PC-1500	ZONE	F0B4	CE-158


PAPER
TJA, HERR KOLLEGE, IHR PC-1500 HAT TATSÄCHLICH RECHT!
HIGH VOLTAGE

Programmieren in Maschinensprache

Allgemeines zur Serie "Programmieren in Maschinensprache"

Die in dieser Serie vorgestellten Programme erheben keinen Anspruch auf Vollkommenheit, sondern sollen lediglich als Übungsobjekt zur Erlernung der Struktur der Maschinensprache dienen. Es könnten wesentlich mehr Programmierkniffe verwendet werden, was die Arbeit aber nicht verständlicher macht.

Ich möchte an dieser Stelle auf weitere Literatur verweisen. Für weitere Anwendungen gibt es das Systemhandbuch von RVS, erhältlich bei: RVS Datentechnik, Postfach 55, 8055 Hallbergmoos. Dieses Buch enthält eine Aufstellung der Maschinensprache-Befehle und eine kleine Auflistung von Unterprogrammen im ROM. Die dabei verwendete Syntax habe ich in meiner Serie auch verwendet.
In einem weiteren System-Handbuch (von Holtkötter) wird die Hardware des PC-1500 dargelegt. Es ist erhältlich bei: Fischel GmbH, Kaiser-Friedrich-Str. 54 a 1000 Berlin 12.
Ein Buch, in dem nur Unterprogramme aus dem ROM stehen, gibt es bei:
H-G. Schlieker, Weichselstr. 5, 2800 Bremen 21. Von hier gibt es auch ein Buch, in dem das gesamte ROM des PC-1500 mit Drucker CE-150 disassembliert aufgelistet und kommentiert ist.
Zum allgemeinen Erlernen der Maschinensprache kann man das Buch "Maschinensprache" für den C-64 von Data Becker, Merowingerstr. 30, 4000 Düsseldorf, empfehlen.

Einführung

Uns steht im PC-1500 eine 8-Bit-CPU zur Verfügung. Das bedeutet, daß wir mit 8-Bit-Werten arbeiten. Mit 8 Bit kann man 256 verschiedene Bit-Kombinationen erzeugen, die ich als Dezimalzahlen (0- 255) oder als Hexadezimalzahlen (&00 - &FF) darstellen kann.

Ein 8-Bit-Wert, wir nennen es jetzt Byte, ist folgend aufgebaut:

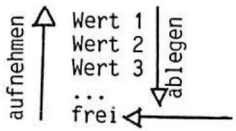
$B_7\ B_6\ B_5\ B_4\ B_3\ B_2\ B_1\ B_0$ B_7 = MSB = höchstwertiges Bit, da es als 128 zählt.
B_2 z.B. zählt nur als 4.

In der CPU des PC-1500 stehen uns mehrere solcher 8-Bit-Register zur Verfügung:
A Akkumulator, oder allgemeines Rechenwerk.
XH, XL, YH, YL, UH, UL : Diese letzten 6 Register werden paarweise zu 16-Bit-Registern zusammengefaßt, so daß man mit einem Doppelbyte 65 536 Kombinationen erzeugen kann, genug, um jeden Speicherplatz (64 K = 64 x 1024 = 65 536) mit einer Nummer anzusprechen, zu adressieren. Man nennt diese neuen Register X, Y oder U, und weil sie einen Speicher direkt ansprechen können, auch Pointer-Register.

Zusätzlich haben wir noch andere Register, die feste Aufgaben haben:

P (16 Bit) Programmzähler, er zeigt auf die Speicherstelle, in der der nächste auszuführende Befehl steht.

S (16 Bit) Stapelzeiger, er zeigt auf die erste frei werdende Stelle unseres Stapels. Dieser Stapel kann irgendwo in den Speicher gelegt werden und ist eine hilfreiche Einheit, denn man kann auf ihn Daten ablegen und in umgekehrter Reihenfolge wieder aufnehmen:



T (9 Bit) Timer, dieses Register wird mit jedem Takt weitergezählt.
F (5 Bit) Flag-Register, in diesem Register werden Zustände angezeigt, die nach Operationen auftraten.

Z Zero-Flag	Das Ergebnis einer Operation ist Null.
C Carry-Flag	Bei einer arithmetischen Operation werden zur vollständigen Darstellung des Ergebnisses mehr als die verfügbaren 8 Bit benötigt.
V Overflow-Flag	Nach einer arithmetischen Operation mit Vorzeichen-behafteten 8-Bit-Werten verläßt das Ergebnis den Bereich -127 bis 127. Vorzeichen-behaftete 8-Bit-Werte sind Werte, bei denen B_7 das Vorzeichen angibt (1 = negativ, 0= positiv). Diese Werte werden in einem besonderen Format geschrieben. Dieses Format soll später besprochen werden.
H Half-Carry-Flag	Nach einer arithmetischen Operation liegt ein Übertrag vom unteren Halbbyte (Nibble) (B_0 - B_3) auf das obere vor.
I Interrupt-Flag	Mit diesem Flag kann man der CPU angeben, ob es auf Unterbrechungen (Interrupts) von Uhr oder anderem reagieren soll oder nicht.

Es wird im folgenden der Syntax der Firma RVS verwendet.

Wie löse ich nun Probleme ?

1. Problem: Ich will die Anzeige mit einem Strich füllen.
In BASIC: CLS:FOR I=1TO 156:GPRINT 8;:NEXT I
In Maschinensprache können wir das Problem ähnlich lösen:
Wir beginnen mti dem Löschen der Anzeige:

Für das Löschen der Anzeige muß man ein aufwendiges Programm schreiben. Im ROM des PC-1500 steht fest an einer Stelle schon solch ein Programm. Wenn wir dieses Programm benutzen wollen, so rufen wir es auf mit CALL .. (ähnlich GOSUB). Damit am Ende des Unterprogrammes im ROM ein Rücksprung in unser Programm erfolgt, hat es den Befehl RTN (ähnlich RETURN).

Wir können nun folgendes schreiben:
CALL &EE71.
Dieser Befehl bewirkt einen Sprung zu dem Programm, welches ab &EE71 beginnt und wieder zurückkehrt mit RTN.
Diesen Befehl kann man ändern: Im ROM des PC-1500 im Bereich &FF00 bis &FFFF stehen 128 Adressen (je 2 Byte erst high-Byte, dann low-Byte). Diese Adressen zeigen auf häufig benötigte Programme im ROM, so wie z.B. die Anzeige zu löschen. In &FFF2 und &FFF3 steht diese Adresse. Wir können nun auch schreiben CALL (&F2). Der Microprocessor nimmt nun nicht wie früher die Adresse &EE71 direkt in das P-Register, sondern holt sich den Wert aus &FFF2 und &FFF3 , welche er, &FFF2 zuerst, in das P-Register übernimmt, wo die Adresse zum nächsten Befehl steht.

Als nächste Einheit haben wir eine Schleife. Solch eine Schleife kann man aufbauen, indem man den Zahlenwert 156 in ein Register nimmt, und solange 1 subtrahiert, bis 0 im Register steht. Also:

LD UL,156 (Anweisung)	Lade direkt die Zahl 156 in das Register UL
DEC UL	subtrahieren 1 von UL
JR NZ,-x	springe, wenn das Ergebnis aus 'DEC UL' nicht Null ist (Z=0), x Schritte zurück (Vorzeichen) (Es wird auf die (Anweisung) gesprungen.)

Das Programm nimmt nun die Zahl 156 in ein Register (UL). Danach beginnt die Schleife, in der die "Anweisung" durchgeführt wird und anschließend das Register (UL) um 1 vermindert wird. Nach der letzten Operation, einer arithmetischen Operation (DFC) wird auch das Z(ero)-Flag gesetzt. Solange in unserem Register UL aber noch ein Wert steht, wird Z auf 0 gesetzt, denn das Ergebnis ist nicht Null. In der letzten Anweisung nun wird ein bedingter Sprung ausgeführt, d.h. wenn die Bedingung erfüllt ist (NZ wahr, Z nicht gesetzt), wird ein Sprung um x Schritte (x wird später ausgerechnet) zurück ausgeführt. Dieser Sprung würde auf die 'Anweisung' führen, damit die Schleife geschlossen ist. Wenn 'Bedingung' nicht erfüllt ist, d.h. in unserem Fall ist im UL-Register aus 1 Null geworden, so ist das Z-Flag gesetzt und der Sprung wird nicht ausgeführt. Das ist das Ende der Schleife. Diese Schleife läßt sich etwas kürzer lösen, indem man einen anderen Befehl verwendet für DEC UL, JR NZ,-x , nämlich DIC ,-x. Dieser Befehl bewirkt, daß im UL-Register 1 subtrahiert wird und anschließend überprüft wird, ob im UL-Register jetzt 255 (&FF) steht, d.h. ob von 0 auf 255 gewechselt wurde. Wurde Null noch nicht unterschritten, so wird der Sprung um x Schritte nach vorn ausgeführt. Unser Programm also:

```
LD UL,155   (155, da nicht mehr von 156 bis 1, sondern von 155 bis 0 gezählt
            wird)
(Anweisung)
DJC ,-x
```

Unsere 'Anweisung' besteht aus dem Befehl 'GPRINT 8;' . Für diesen Befehl gibt es wieder ein Unterprogramm, denn zur Ansteuerung der Anzeige gehört viel Arbeit. Dieses Unterprogramm verlangt aber von uns bestimmte Angaben. Nämlich:
Wo soll ich welches Zeichen zeichnen.
Das Wo teilen wir dem Programm mit, indem wir im X-Register die 'Spaltenadresse' haben. Diese Adresse ist, wie alles bei der Arbeit mit der Anzeige, ein kompliziertes Gebilde. Es genügt, wenn wir wissen, daß die Adresse für die erste Spalte auf der Anzeige &7400 ist. Die weiteren Adressen brauchen wir in diesem Programm nicht zu berechnen, da das von uns angesprungene Unterprogramm diese selber ausrechnet und uns im X-Reg. wieder zur Verfügung stellt. Wir müssen also vor Beginn der Schleife die erste Adresse aufbauen, denn die weiteren werden für uns errechnet. Wir sagen:

```
LD XH,%74     ;Lade &74 in das XH-Register, der oberen Hälfte des X-Reg.
LD XL,&00     ;
```

Damit haben wir die erste Adresse im X-Reg. In der Schleife ist die Wo-Frage gelöst, bleibt nur noch die welches-Frage. Das UP (Unterprogramm) verlangt von uns das anzuzeigende Zeichen (GPRINT) im Akkumulator, dem Hauptrechenregister. Wir sagen daher:

```
LD A,8        ;Lade 8 in A
CALL &EDF6 oder CALL (&88)
```

Nach diesem Unterprogramm ist der Akku und das UH-Reg. zerstört. Da wir diese Reg. nicht weiter brauchen, können wir diese Tatsache vernachlässigen.

Wir haben jetzt alle Befehle unseres Programms erklärt und hoffentlich verstanden. Wir können jetzt das Programm schreiben:

Muemonics	übersetzt in CPU-Code	Hex-Listing
LD XH,&74		&48 &74
LD XL,&00		&4A %00
CALL (&F2)		&F2
LD UL,155		&6A 155
LD A,8		&B5 8
CALL (&88)		&CD &88
DJC ,-6		&88 6
RET		&9A

Jetzt können wir erst unser x bei DJC berechnen. Wir sehen, daß die CPU 6 Schritte zurückgehen muß, um von RET (&9A) auf LD A,8 (&B5) zu kommen, denn unser P(rogrammcounter) steht nach Lesen des Befehles DJC und der Zahl 6 auf RET. Nun muß er 6 Schritte zurückspringen (wenn Bedingung erfüllt ist).

Wir können unser 14 Byte langes Programm nun in einen Speicherplatz POKE'n und mit CALL aufrufen:
Wir wählen den Speicherraum &7650 bis &765C. In diesem Speicherraum liegt die Variable E$. Wir sagen also:
POKE &7650,&48,&74,&4A,0,&F2,&6A,155,&B5,8,&CD,&88,&88,6,&9A.

Jetzt können wir unser Programm mit 'CALL &7650' starten und wir werden das Ergebnis sehen.

```
10:CALL &7650       Aufruf aus BASIC mit Warteschleife in 20.
20:GOTO 20
```

2. Problem: Laufschrift

Wir wollen versuchen, eine Laufschrift auf dem PC-1500 zu erzeugen - in Maschinensprache.
In BASIC sähe es folgend aus:

```
10:WAIT 0
20:FOR I=155TO 0STEP -1
30:CLS :GCURSOR I:PRINT (TEXT):NEXT I
```

Wieder können wir unser Programm strukturieren:
Fangen wir mit der Schleife an:

```
  LD UL,155
┌▷Anweisung
└ DJC ,-x    (Sprung zur Anweisung)
```

Die 'Anweisung'besteht aus der Ausgabe des Textes auf der Anzeige an der Stelle, die durch den 8-Bit-Register UL angegeben wird (Zählvariable). Zu Beginn der Anweisung brauchen wir ein 'CLS', welches wir in Maschinensprache durch 'CALL (&F2)' (indirekt) erwirken. Diese Routine zerstört aber das U-Register (16-Bit), darum sichern wir dieses Register erst auf den Stapel mit den Befehlen:

```
  PUSH U      (das ganze 16-Bit-Register)
  CALL (&F2) (Anzeige löschen)
  POP U
```

Mit dem Befehl 'POP' laden wir das Register wieder vom Stapel. Das ist sehr wichtig, da der Stapel von vielen Routinen benutzt wird, und so immer wieder hergestellt werden muß. So ist unser Programm auch nur eine Routine, die den Stapel so zurücklassen muß, wie er am Anfang war !!! Jetzt haben wir die Anzeige gelöscht und müssen nur (!) noch den Text beschreiben.

Den Befehl GCURSOR ersetzen wir durch die Anweisung, welche den Anfang des zu schreibenden Textes in eine Speicherstelle setzt, die Spaltenpointer genannt wird und die Nummer (0-155) der Spalte enthalten muß.

```
  LD A,UL      (Lade den Inhalt vom UL-Register in den Akku)
  LD (&7875),A  (Lade den Inhalt des Akkus in die Speicherstelle &7875)
```

Den Befehl 'PRINT' ersetzen wir ebenfalls durch ein Unterprogramm, welches aber die Startadresse des Textes und dessen Länge braucht, d.h. die Adresse, ab der der Text gespeichert ist. Diese Adresse werden wir schon beim Aufruf des Programmes im X-Reg (16-Bit) und die Länge in A vorfinden. Wir werden das

X-Reg in das Y-Reg übertragen, damit uns der Wert nicht verloren geht, denn das Y-Reg ist vom Unterprogramm noch nicht gebraucht. Ebenso werden wir uns den Wert aus A sichern, nämlich in das UH-Register. Also müssen wir vor der Schleife folgendes einsetzen:

```
LD Y,X      (Übertrage den Wert aus X in Y)
LD UH,A     ( "         "   "    "  A in UH)
```

Wir haben also unsere Werte, die wir brauchen, in den Registern Y und UH. Das Unterprogramm, welches wir aufrufen, braucht die Startadresse des Textes im U-Reg und die Länge in A. Wir müssen also erst das U-Reg sichern, um die Werte nicht zu zerstören. Damit sichern wir gleichzeitig die Schleifenvariable (UL) und die Länge (UH) des Textes. Anschließend übertragen wir das UH-Reg mit der Länge des Textes in den Akku (A). Jetzt können wir den Inhalt aus Y nach U speichern, was jedoch nicht einfach geht, wir müssen einen Umweg über das X-Reg nehmen:

```
PUSH U      (sichere U auf den Stapel)
LD A,UH     (Lade die Textlänge in den Akku)
LD X,Y      (Lade die Startadresse in das X-Reg)
LD U,X      ( "    "   "           in das U-Reg)
```

Wir haben jetzt alle Werte aufgebaut und könne die Routine aufrufen:

```
CALL (&92)
```

Jetzt müssen wir unbedingt den 16 Bit-Wert wieder vom Stapel holen, den wir dort abgelegt haben:

```
POP U       (Hole den 16-Bit-Wert vom Stapel und bringe ihn in U)
```

Wir haben jetzt das Ende der Anweisung erreicht und können das 'NEXT' programmieren mit dem Befehl DJC (siehe Anfang).

Wie erhalte ich die Startadresse und die Länge des Textes bei Aufruf des Programmes ? Bei unserem ersten Problem haben wir das Programm mit CALL (Adresse) aufgerufen. Jetzt rufen wir das Programm mit CALL (Adresse),(Variable) auf, wobei die Variable eine String-Variable ist.

Am Ende unseres Programmes wollen wir noch eine kleine Zeitverzögerung einbauen, damit man den Text lesen kann. Wir haben im UL-Reg zuletzt 255 stehen, da wir von 0 subtrahiert haben. Also laden wir in das UH-Reg 0, damit der Wert 255 im U-Reg (16-Bit) steht. Anschließend rufen wir eine Routine mit CALL (&AC) auf, die eine Verzögerung von U x 1/64 Sekunden erzeugt. Am Ende des Programmes setzen wir das Carry-Flag zurück, und setzen den Befehl RET. Das Carry-Flag setzen wir zurück, damit der Interpreter (BASIC) nicht den Text ab der Startadresse X mit der Länge A in die String-Variablen aus dem Aufruf überträgt, da diese Werte jetzt nicht mehr stimmen.

Fertig sieht unser Programm so aus:

Muemonics	übersetzt in Maschinensprache	Hex-Listing
LD Y,X	Startadresse String sichern	&FD &5A
LD UH,A	Länge String sichern	&28
LD UL,155	Lade Zählwert	&6A 155
PUSH U ←┐	Rette Register	&FD &A8
CALL (&F2)	Lösche Anzeige	&F2
POP U	Hole Register	&FD &2A
LD A,UL	Lade GCURSOR-Wert	&24
LD (&7875),A	Cursor setzen	&AE &78 &75
PUSH U	Rette Register	&FD &A8
LD A,UH	Hole Länge des Strings	&A4
LD X,Y	Startadresse des Strings	&FD &18

(Fortsetzung nächste Seite)

```
LD U,X      nach U übertragen                          &FD &6A
CALL (&92)  Text ausgeben                              &CD &92
            Anmerkung: Versuchen Sie doch nach jeder
            Anzeige des Textes eine kleine Zeitschleife
            einzubauen, damit man den Text besser lesen
            kann. Lösung:
            LD UH,0  :LD UL,10 : CALL (&AC)
            DJC-Wert auf -28 setzen (!)
POP U       Register wiederholen                       &FD &2A
DJC -22 ——  Schleifensprung und GCURSOR-Wert verringern &88 22
LD UH,Ø     U-Register für Zeitschleife vorbereiten    &68 Ø
CALL (&AC)  Warteschleife                              &CD &AC
RCF         Ende                                       &F9
RET                                                    &9A
```

Dieses Programm kann man in den Speicher POKE'n (ab &7750). Danach setzt man einen Text in eine String-Variable, z.B. A$="PC-1500 SHARP" (13 Zeichen) und ruft das Programm auf mit CALL &7750,A$.

Speicher-Test-Programm

Man liest immer wieder von Speicher-Test-Programmen in den verschiedensten Versionen, und ich nahm mir vor, ein solches zu schreiben.
Es ergaben sich folgende Anforderungen:
1) Voll rolokatibel, d.h. an jeder Speicherstelle lauffähig (außer ROM)
2) frei wählbarer zu untersuchender Bereich
3) 1./2. Seite (64K) des Speichers müssen untersucht werden können
4) nach dem Test sollen alle Speicherinhalte noch vorhanden sein
5) zwei verschiedene Tests sollen möglich sein:
   Test 1: In alle Speicherstellen wird &FF geschrieben und überprüft, in alle Speicherstellen wird %00 geschrieben und überprüft.
   Test 2: In alle Speicherstellen werden alle Kombinationen von 0 bis 255 geschrieben und überprüft.

Programmierung:

Ich begann mit der Schleife, welche den Speicherbereich durcharbeitet:

```
Anfang (Schleife)   Test
                    Zähler (X) erhöhen
Einsprung           Zähler größer als max. Grenze (y) ?
                    wenn nein, weiter      Anfang
```

Den Einsprung wählte ich extra im Ende der Schleife, damit keine Speicherstelle überprüft wird, die schon gar nicht mehr geprüft werden darf (Sicherheit !).
Der Aufruf meines gesamten Programmes sollte mit einem HEXMONITOR erfolgen, so daß man die Start- (X) und Endadresse (Y) des zu prüfenden Speicherbereiches besser eingeben kann. Außerdem steht in UH die Test-Art:
1≙schnell (Test 1)
0≙langsam (Test 2)

Sollte das Programm einen Fehler gefunden haben, so kehrt es zurück mit der gefundenen Adresse+1 in X (damit man wieder besser starten kann, ohne den Wert (x) neu einzugeben), dem in die Speicherstelle eingeschriebenen Wert in UL und dem alten Wert der Speicherstelle in A. Y ist unverändert.

Ich habe jetzt meine Test-Routine so aufgebaut, daß sie möglichst kurz wird:
Test-Schleife:

```
Label  Muemonics

1 x    LD A,(X)   ;Hole alten Wert
       POP A      ;Sichere alten Wert
       LD UL,255  ;Lade ersten zu testenden Wert
2      LD A,UL    ;Lade zu testenden Wert in den Akku
  x    LD (X),A   ;Lade Wert in die zu testende Zeile
  x    LD A,(X)   ;Hole Wert zurück
       CP A,UL    ;Vergleiche gefundenen Wert mit Original
       JR Z, 3    ;Verzweige, wenn beide Werte gleich, zum Ende der Schleife 3
       POP A      ;Hole alten Wert
  x    LD (X),A   ;Bringe ihn ingetestete Zelle
       INC X      ;Erhöhe Zeiger auf nächste Zelle
       RET        ;Ende, da Fehler gefunden
3      LD A,UH    ;Hole Test-Art
       CP A,1     ;Test-Art 1?
       JR NZ, 4   ;Verzweige, wenn Test-Art 2    4
       LD UH,&81  ;Hole Test-Art 1-Marke + Marke (&80)
       LD UL,1    ;Schreibe Test-Wert 1 in UL, damit er bei DJC 0 wird
4      DJC        ;Generiert nächsten Test-Wert in UL und springt zu 2
       LD A,UH    ;Lade Test-Art-Marke in Akku
       OR A,1     ;Lösche eventuell vorhandene Marke
       LD UH,A    ;Lade Test-Art-Marke zurück
       POP A      ;Hole alten Wert
  x    LD (X),A   ;Schreibe ihn in die getestete Zelle zurück
       INC X      ;Erhöhe Zeiger auf nächste Stelle
```

Label:
1 ≙ Anfang der Schleife

Dadurch, daß zwischen 3 und 4 der Wert 1 in UL geladen wird, erhält UL nach DJC den Wert 0 (Test-Art 1). Beim nächsten Durchlauf steht nicht mehr die Marke 1 in UH, sondern &81, damit nicht nocheinmal 1 in UL geladen wird.

Die Schleife wird bei Test-Art 1 zweimal durchlaufen und bei Test-Art 2 256mal durchlaufen, daher kommt der Begriff schnell und langsam, denn 16K langsam testen kostet erheblich Zeit.

An den durch x gekennzeichneten Stellen kann man den Zugriff auf die 2.Speicherseite definieren mit LDH A,(X) bzw. LDH (X),A.

Benutzt man einen Macro-Assembler, kann man sehr gut diese gekennzeichneten Stellen durch MACROs ersetzen, die man an einer Stelle definiert und so leicht auf die zweite Seite umschalten kann.

Man kann sich nun zwei Versionen erstellen, eine für die 1. Seite, eine für die 2. Seite. Man kann auch eine Version für die erste Seite generieren, bei der an der x-Stellen ein NOP eingefügt ist, welches man gegebenfalls in &FD umwandelt für die zweite Seite, so daß das Programm fertig übersetzt auf Kassette, zusammen mit dem HEXMONITOR in die 2K RAM des PC-1500 passen, damit bei Total-Ausfällen von Speichern noch Fehler gesucht werden können.

Wir müssen nur noch beide Teile zusammenfassen, um das fertige Programm zu erhalten:

```
START JR Einsprung
      Test-Schleife
Einsprung
      RET
```

Im MACROASSEMBLER kann man nun Makros einfügen und IF-ENDIF-Makros verwenden. Das fertige Programm habe ich als HEX-LISTING und ASSEMBLER-LISTING unten angefügt.

```
PROGRAM-CODE:
4000: 8E 23 05 FD
4004: C8 6A FF 24
4008: 0E 05 26 8B
400C: 05 FD 8A 0E
4010: 44 9A A4 B7
4014: 01 89 04 68
4018: 81 6A 01 88
401C: 16 FD 8A 0E
4020: 44 A4 B9 01
4024: 28 84 96 91
4028: 27 89 04 14
402C: 06 93 2D 9A
```

```
;SPEICHER-TEST
=!LDA! LD A,(X)
=!LDX! LD (X),A
START &5000,&6000,&4000,&5000
;---------------------------
CORE  JR LOOPE
LOOP  !LDA!:PUSH A:LD UL,255
LOPTST LD A,UL:!LDX!:!LDA!
      CP A,UL
      IF#NZ:POP A:!LDX!:INC X:RET:ENDIF
      LD A,UH:CP A,1
      IF#Z:LD UH,&81:LD UL,1:ENDIF
      DJC LOPTST
;
      POP A:!LDX!:INC X:LD A,UH:AND A,1:LD UH,A
LOOPE LD A,XH:CP A,YH:JR NC,LOOP
      IF#Z:LD A,YL:CP A,XL:JR C,LOOP:ENDIF
      RET
END
```

Erweitertes Speicher-Test-Programm

Hier möchte ich eine Erweiterung des bereits ausgeführten Speichertestes vorstellen. Diese Version prüft einen weiteren Fehler, der oft bei Eigenentwicklungen auftritt. Dieser Fehler äußert sich, indem zwei Speicherbereiche angesprochen werden bei einer Adresse. Dieser Fehler liegt an der Decodierung. Deshalb sind die Adressen, die doppelt angesprochen werden,im Abstand von &800 (2K), weil die ICs meist 2K speichern.

Diesen zweiten Teil des Programms füge ich an das erste Programm, darum müssen die Anfangswerte (X,Y) gesichert werden, damit sie nach dem ersten Test noch verfügbar sind. Dis geschieht durch ein 'PUSH Y:PUSH X' (Reihenfolge !) vor dem relativen Sprung am Programmstart. Wird im ersten Test bereits ein Fehler festgestellt, so müssen diese Register wieder vom Stack geholt werden (!). Jetzt erkennt man, warum die Register in dieser Reihenfolge abgelegt wurden. Man kann mit 'POP Y:POP Y' die Werte vom Stack holen, wobei das originale Y-Reg überschrieben, danach aber wieder geladen wird. Die Werte nach einem Rücksprung aus dem ersten Teil bleiben deshalb wie zuerst beschrieben. Wird der erste Teil erfolgreich durchlaufen, so muß der zweite Teil die abgelegten Werte wieder vom Stack lesen mit 'POP X:POP Y.' Danach beginnt der zweite Teil, der dem ersten Teil sehr ähnlich ist.

Es erfolgt ein Sprung zum Ende der Schleife, an dem auf Ende geprüft wird. Zuerst jedoch wird der Wert aus X-Reg in U-Reg übertragen für den späteren

Test. Jetzt wird zum X-Reg &800 addiert mit 'LD A,8:RCF:ADC A,XH:LD XH,A. Jetzt wird überprüft wie bei der ersten Schleife, allerdings steht XH bereits in A. Ist das Ende noch nicht erreicht, so wird zum Anfang der ersten Schleife verzweigt. Hier wird in die Speicherstelle, auf die U-Reg zeigt, 0 und in die Speicherstelle von X-Reg 255. (X-Reg ist 2K weiter wie U-Reg !). Würde jetzt ein Decodierfehler vorliegen, so überschreibt die 2. Speicherung die erste. Darum werden jetzt die Speicherinhalte wieder ausgelesen und auf ihren Wert überprüft. Stimmt dieser Wert nicht, so wird U-Reg auf 0 gesetzt und es erfolgt ein Aussprung aus dem Programm mit der Adresse in X-Reg, die die um 2K tiefere überschrieben hat.

```
;SPEICHER-TEST ?
=!LDA! LD A,(X)
=!LDAU! LD A,(U)
=!LDX! LD (X),A
=!LDU! LD (U),A
START &5000,&6000,&4000,&5000
;----------------------------
CORE  PUSH Y:PUSH X:JR LOOPE
LOOP  !LDA!:PUSH A:LD UL,255
LOPTST LD A,UL:!LDX!:!LDA!
      CP A,UL
      IF#NZ:POP A:POP Y:POP Y:!LDX!:INC X
      ELSE#:LD A,UH:CP A,1
         IF#Z:LD U,&8101:ENDIF
         DJC LOPTST
;
         POP A:!LDX!:INC X:LD A,UH:AND A,1:LD UH,A
LOOPE    LD A,XH:CP A,YH:JR NC,LOOP
         IF#Z:LD A,YL:CP A,XL:JR C,LOOP:ENDIF
;TEST TEIL 2
TEST2    POP X:POP Y:JR LOOP2E
LOOP2    LD A,0:!LDU!:LD A,255:!LDX!:!LDAU!
         IF#NZ:LD X,0
         ELSE#:!LDA!:INC A
            IF#NZ:LD U,0
            ELSE#
LOOP2E         LD U,X:RCF:LD A,8:ADC A,XH:LD XH,A:CP A,YH
               JR NC,LOOP2
               IF#Z:LD A,YL:CP A,XL
                  JR C,LOOP2
TEST2E         ENDIF
            ENDIF
         ENDIF
      ENDIF
      RET
END
```

```
PROGRAM-CODE:
4000: FD 98 FD 88
4004: 8E 28 05 FD
4008: C8 6A FF 24
400C: 0E 05 26 8B
4010: 0A FD 8A FD
4014: 1A FD 1A 0E
4018: 44 8E 4C A4
401C: F2 F1 B5 04
4020: 68 81 6A 01
4024: 88 1B FD 8A
4028: 0E 44 A4 B9
402C: 01 28 84 96
4030: 91 2C 89 04
4034: 14 06 93 32
4038: FD 0A FD 1A
403C: 8E 19 B5 00
4040: 2E B5 FF 0E
4044: 25 8B 06 48
4048: 00 4A 00 8E
404C: 1A 05 DD 8B
4050: 0C 68 00 6A
4054: 00 8E 10 FD
4058: 6A F9 B5 08
405C: 82 08 96 91
4060: 23 89 04 14
4064: 06 93 29 9A
```

Teilerzerlegung

Problem: Es soll eine Zahl in Teiler zerlegt werden. Diese Teile sollen aus einer vorher feststehenden Menge sein. Es soll sich folgendes Bild ergeben: Zahl=x_1 mal y_1 / Zahl=x_2 mal y_2 / ...

Dieses Problem wurde von einem Leser der PC-1500-Zeitung bereits mit einem Such-Verfahren in BASIC gelöst. Nur wird diese Routine oft benötigt und sie ist sehr langsam. Deshalb entwickelte ich ein Maschinensprache-Programm, welches das Problem durch Probieren löst. Es werden alle Teiler mit (fast) allen multipliziert und dann auf Richtigkeit geprüft. Als großer Vorteil sei erwähnt, daß alle Zahlen ganze Zahlen sind und daß sie kleiner 65 536 sind, also im 16-Bit-Format darstellbar sind.

Ich gliederte das Programm in 4 Teile:
- Initialisierung
- Eingabe der möglichen Teiler
- Suche der Teiler (1)
- Suche nach weiteren Teilern (2)

Die Übergabe der Werte kann mit dem CALL erfolgen, gefolgt von einer Variablen.

Initialisierung:

Die Initialisierung ist nötig, da das Programm eine Liste aller möglichen Teiler aufbauen muß. Bei der Initialisierung werden nun die Pointer für diese Liste gesetzt. Die Liste wird der Einfachheit halber direkt hinter das BASIC-Programm gelegt. Die Folge von Befehlen, die mir diesen Wert in ein 16-Bit-Register der CPU lädt, habe ich im Assembler als MAKRO definiert:

```
=!LDXEND!LD A,(PRGEND):LD XH,A:LD A,(PRGEND+1):LD XL,A      / PRGEND=&7867
```

wird vorher gesetzt.
Dieses Makro ergibt übersetzt: Lade Pointer auf Programm-Ende in X

```
LD A,7867
LD XH,A
LD A,7868
LD XL,A
```

(wir werden diese Folge noch mehrmals verwenden).

Den gleichen Makro habe ich auch für das Y-Register definiert. Diesen letzten Makro verwende ich zuerst, um den Pointer in das Y-Reg zu laden. Damit wäre die Initialisierung fast abgeschlossen, denn ich muß diesen Pointer noch im Speicher sichern, damit nach dem Rücksprung in BASIC die Zahl nicht verloren geht. Zum Ablegen dieses Pointers verwende ich den Speicherbereich &77F0-&77FF, in dem nachher noch mehr Werte abgelegt werden. Ich stellte mir folgende Adressen auf:

```
ZWX=&77F0       (Zwischenspeicher, reserviert für das X-Reg)
ZWY=ZWX+2       ( "                "          "   "  Y-Reg)
ZWU=ZWY+2       ( "                "          "   "  U-Reg)
ZWWERT=ZWU+2    ( "                "          "  den zu untersuchenden
                                                 Wert)
```

Dazu auch folgende MAKRO-BIBLIOTHEK:

```
=!SVEX!LD A,XH:LD (ZWX),A:LD A,XL:LD (ZWX+1),A
```

gleiches mit Y und U und dem 'Wert':

```
=!SVEWERT!LD A,XH:LD (ZWWERT),A:LD A,XL:LD (ZWWERT+1),A
```

Der Wert wird in X behandelt, da er hier vom BASIC übergeben wird.

Zum Wiederholen dieser Werte:

```
=!GETX!LD A,(ZWX):LD XH,A:LD A,(ZWX+1):LD XL,A
```

so auch mit Y, U, Wert

Nach dem Anlegen dieser Makros und der Adressen kann man jetzt richtig anfangen (mit ASSEMBLER):

```
INIT !LDYEND!:!SVEY!:RCF:RET    Der Pointer-Wert wird aus der Adresse 7867
                                geladen und in ZWY abgelegt. Danach wird das Carry-Flag ge-
                                löscht und es erfolgt ein Rücksprung in BASIC, ohne den In-
                                halt aus dem X-Reg zu übernehmen (Carry).
```

Eingabe der möglichen Teiler:

Dieses Programm kann auf die Makros aus der Initialisierung zurückgreifen, d.h. ich muß nicht alles noch einmal eingeben, denn Makros sind keine Unterprogramme, sondern nur Befehlsfolgen, die man unter einem Namen ablegt.

```
STORE !GETY!      Der Pointer auf die Tabelle wird geladen. Er zeigt auf das
      LD A,XH     erste freie Element (bei der Initialisierung festgelegt).
      LDI (Y),A   Das H-Byte des einzuspeichernden Wertes aus X wird übernom-
                  men und abgelegt, gleichzeitig wird der Pointer erhöht.
      LD A,XL
      LDI (y),A   Gleich dem H-Byte. Jetzt steht die Adresse des nächsten
      !SVEY!      freien Tabellenplatzes wieder in Y und wird wieder abgelegt.
      RCF
      RET
```

Diese Routine legt also eine Tabelle im Speicher an, beginnend bei der in INIT festgelegten Adresse, die in ZWY steht und mit !GETY! geladen bzw. mit !SVEY! wieder abgelegt wird.
Wir wollen vereinbaren, daß das BASIC-Programm zuletzt eine Ø einspeichert mit diesem Programm, womit das Ende der Tabelle angezeigt ist durch zwei aufeinander folgende Byte mit 0 als Inhalt.

Bei dem letzten Teil, der Suche nach Teilern, verwende ich ein Unterprogramm aus dem ROM, welches mir das Y-Reg mit dem U-Reg multipliziert und das Ergebnis in X-Reg und Y-Reg übergibt. Das Ergebnis darf maximal 16 Bit lang sein. Wird diese Länge überschritten, ist nach der Rückkehr das Carry-Flag gesetzt. Die Anfangswerte in Y-Reg und U-Reg dürfen auch nur maximal 16 Bit lang sein (größer geht auch nicht !). Der Aufruf ist: CALL (&50) und als Makro definiert.

Suche nach Teilern:

Diesen Teil gestalte ich einfach in 2 FOR-NEXT-Schleifen.

Zuerst sichere ich den im X-Reg übergebenen gesuchten Wert mit !SVEWERT!. Dann beginne ich die erste Schleife:

```
!LDXEND!.       Mit diesem Befehl setze ich den 1.Zähler in X.
                Die zweite Schleife starte ich entsprechend:
!SVEX!:LD Y,X.  Ich sichere den alten X-Reg und lade diesen Wert nach Y-Reg,
                meinem 2. Zähler. Ich beginne also nicht immer von vorn,
                weil ich nicht alle mit allen Zahlen multiplizieren brauche,
                sondern nur die Hälfte.
```

Jetzt beginnt der eigentliche Teil der Schleife, in dem die beiden Zahlen, auf die die Schleifenzähler zeigen, miteinander multipliziert werden und verglichen wird, ob es schon das Ergebnis ist. Das geschieht folgend:

```
!SVEY!          Das Y-Reg sichern
LDI A,(Y):LD XH,A:LD  A,(Y):LD XL,A  lädt die Zahl, auf die Y-Reg (For 2)
                zeigt, in X-Reg
                Wenn der letzte Wert 0 war, ist Z gesetzt.
  JR NZ,Weiter        Sprung, wenn noch nicht Ende der Tabelle erreicht.
  LD A,XH:JR Z,NEXT1  Sprung, wenn Ende der Tabelle (0)
 WEITER LD Y,X:!GETX!   Geladene Zahl nach Y-Reg, und X-Reg wieder restau-
                rieren (wurde mit SVEX gerettet).
  LDI A,(X):LD UH,A:LD A,(X):LD UL,A:!SVEU!          Zahl laden, auf die X-
                Reg zeigt (FOR1) nach U-Reg und absichern.
  !MUL!:JR C,NEXT2    Multiplizieren, und wenn Ergebnis zu groß, dann nächs-
                ten Wert nehmen.
  !GETWERT!:LD A,YH:CP A,XH:JR NZ,NEXT2            Gesuchte Zahl nach X-
  LD A,YL:CP A,XL     Reg laden und mit Y-Reg (Ergebnis) vergleichen. Wenn
  JR NZ,NEXT2         nicht gleich, dann nächsten Wert.
  !GETU!:LD X,U:SCF:RET       Gefundenen Teiler wieder laden und zurück zu
                BASIC
 NEXT2 !GETY!:INC Y:INC Y:JR TEST     Der Schleifenzähler der 2.Schleife
                wird erhöht. Sprung zum Anfang der Schleife.
NEXT1 !GETX!:INC X:INC X:INC X:LDD A,(X):JR NZ,FOR2
                Zähler der 1. Schleife erhöhen und 1.Hälfte
                des Wertes laden. Wenn nicht Null, dann FOR2
                eröffnen.
  INC X:INC X:LDD A,(X):DEC X:JR NZ,FOR2     ebenso 2.Hälfte.
  ENDE LD X,0:SCF:RET         Ende der Suche, 0 übergeben in BASIC
```

```
;TEILER-SUCHE
START &5000,&6000,&4000,&5000
ZWX=&77F0:ZWY=ZWX+2:ZWU=ZWY+2:ZWWERT=ZWU+2:PRGEND=&7867
=!LDXEND!LD A,(PRGEND):LD XH,A:LD A,(PRGEND+1):LD XL,A:INC X
=!LDYEND!LD A,(PRGEND):LD YH,A:LD A,(PRGEND+1):LD YL,A:INC Y
=!SVEU!LD A,UH:LD (ZWU),A:LD A,UL:LD (ZWU+1),A
=!SVEY!LD A,YH:LD (ZWY),A:LD A,YL:LD (ZWY+1),A
=!SVEX!LD A,XH:LD (ZWX),A:LD A,XL:LD (ZWX+1),A
=!SVEWERT!LD A,XH:LD (ZWWERT),A:LD A,XL:LD (ZWWERT+1),A
=!GETX!LD A,(ZWX):LD XH,A:LD A,(ZWX+1):LD XL,A
=!GETY!LD A,(ZWY):LD YH,A:LD A,(ZWY+1):LD YL,A
=!GETU!LD A,(ZWU):LD UH,A:LD A,(ZWU+1):LD UL,A
=!GETWERT!LD A,(ZWWERT):LD XH,A:LD A,(ZWWERT+1):LD XL,A
      JR INIT
      JR STORE
      JR SUCH
      JR NEXT2
INIT  !LDYEND! !SVEY! RET
STORE !GETY! LD A,XH:LDI (Y),A:LD A,XL:LDI (Y),A:!SVEY! RCF:RET
SUCH  !SVEWERT!
FOR1  !LDXEND!
FOR2  !SVEX! LD Y,X
```

```
TEST   !SVEY! LDI A,(Y):LD XH,A:LD A,(Y):LD XL,A:JR NZ,WEITER
       LD A,XH:JR Z,NEXT1
WEITER LD Y,X:!GETX! LDI A,(X):LD UH,A:LD A,(X):LD UL,A:!SVEU!
       CALL (&50):JR C,NEXT2
       !GETWERT!:LD A,YH:CP A,XH:JR NZ,NEXT2
       LD A,YL:CP A,XL:JR NZ,NEXT2
       !GETU!:LD X,U:SCF:RET

NEXT2  !GETY! INC Y:INC Y:JR TEST
NEXT1  !GETX! LD A,3:ADD X,A:LDD A,(X):JR NZ,FOR2
       INC X:INC X:LDD A,(X):DEC X:JR NZ,FOR2
ENDE   LD X,0:SCF:RET
END
```

PROGRAM-CODE:

```
4000: 8E 06 8E 16   4050: AE 22 F3 55   40A0: F0 08 A5 22
4004: 8E 20 8E 80   4054: 08 15 0A 89   40A4: F1 0A B5 03
4008: A5 28 67 18   4058: 03 84 8B 42   40A8: FD 5A 47 99
400C: A5 28 68 1A   405C: FD 50 A5 22   40AC: 6C 44 44 47
4010: 54 94 AE 22   4060: F0 08 A5 22   40B0: 46 99 22 48
4014: F2 14 AE 22   4064: F1 0A 45 28   40B4: 00 4A 00 FB
4018: F3 9A A5 22   4068: 05 2A A4 AE   40B8: 8A
401C: F2 18 A5 22   406C: 22 F4 24 AE
4020: F3 1A 84 54   4070: 22 F5 CD 50
4024: 04 51 94 AE   4074: 83 1C A5 22
4028: 22 F2 14 AE   4078: F6 08 A5 22
402C: 22 F3 F9 9A   407C: F2 0A 94 86
4030: 84 AE 22 F6   4080: 89 10 14 9E
4034: 04 AE 22 F7   4084: 89 0C A5 22
4038: A5 28 67 08   4088: F4 28 A5 22
403C: A5 28 68 0A   408C: F5 29 FD 29
4040: 44 84 AE 22   4090: FB 9A A5 22
4044: F0 04 AE 22   4094: F2 18 A5 22
4048: F1 FD 5A 94   4098: F3 1A 54 54
404C: AE 22 F2 14   409C: 9E 53 A5 22
```

Unter Verwendung von Struktur-Direktiven:

```
;TEILER-SUCHE
START &5000,&6000,&4000,&5000
ZWX=&77F0:ZWY=ZWX+2:ZWU=ZWY+2:ZWWERT=ZWU+2:PRGEND=&7B67
=!LDXEND!LD A,(PRGEND):LD XH,A:LD A,(PRGEND+1):LD XL,A:INC X
=!LDYEND!LD A,(PRGEND):LD YH,A:LD A,(PRGEND+1):LD YL,A:INC Y
=!SVEU!LD A,UH:LD (ZWU),A:LD A,UL:LD (ZWU+1),A
=!SVEY!LD A,YH:LD (ZWY),A:LD A,YL:LD (ZWY+1),A
=!SVEX!LD A,XH:LD (ZWX),A:LD A,XL:LD (ZWX+1),A
=!SVEWERT!LD A,XH:LD (ZWWERT),A:LD A,XL:LD (ZWWERT+1),A
=!GETX!LD A,(ZWX):LD XH,A:LD A,(ZWX+1):LD XL,A
=!GETY!LD A,(ZWY):LD YH,A:LD A,(ZWY+1):LD YL,A
=!GETU!LD A,(ZWU):LD UH,A:LD A,(ZWU+1):LD UL,A
=!GETWERT!LD A,(ZWWERT):LD XH,A:LD A,(ZWWERT+1):LD XL,A
      JR INIT
      JR STORE
      JR SUCH
      JR NEXT2
```

```
INIT   !LDYEND! !SVEY! RET
STORE  !GETY! LD A,XH:LDI (Y),A:LD A,XL:LDI (Y),A:!SVEY! RCF:RET
SUCH   !SVEWERT!
FOR1   !LDXEND!
FOR2   BEGIN:!SVEX! LD Y,X
TEST   |  !SVEY! LDI A,(Y):LD XH,A:LD A,(Y):LD XL,A
       |  IF#Z:LD A,XH:ENDIF
WEITER |  IF#NZ:LD Y,X:!GETX! LDI A,(X):LD UH,A:LD A,(X):LD UL,A:!SVEU!
       |  |     CALL (&50)
       |  |     IF#NC:!GETWERT!:LD A,YH:CP A,XH
       |  |     |  IF#Z:LD A,YL:CP A,XL
       |  |     |  |  IF#Z:!GETU!:LD X,U:SCF:RET:ENDIF
       |  |     |  ENDIF
       |  |     ENDIF
NEXT2  |  |     !GETY! INC Y:INC Y:JR TEST
       |  |
       |  ENDIF
NEXT1  |  !GETX! LD A,3:ADD X,A:LDD A,(X)
       |  IF#Z:INC X:INC X:LDD A,(X):DEC X:ENDIF
       UNTIL#Z
ENDE   LD X,0:SCF:RET
END
```

```
1000 CALL &4000:RESTORE
1010 READ A:CALL &4002,A:IF A>0THEN 1010
1020 INPUT X:LPRINT X:Y=X:CALL &4004,Y:IF Y=0THEN 1020
1030 LPRINT Y:Y=X:CALL &4006,Y:IF Y>0THEN 1030
1040 LPRINT "---------":GOTO 1020
2000 DATA 2,4,6,8,10,12,14,20,200,60,157,6547,0
```

GPRINT ohne Löschen des Untergrundes

Hier stelle ich ein Programm vor, das einen GPRINT ohne Löschen des Untergrundes auf der Anzeige ermöglicht. In BASIC wäre der entsprechende Befehl GPRINT POINT x OR y (y bei aktueller Spalte x zeichnen).

Die Maschinensprache-Lösung soll aber einen Zeichen-String im GPRINT-Format bringen, z.B. "08087F7F00007F7F0808".
Die GPRINT-Routine, die im ROM steht, löscht den Untergrund. Das Löschen des Untergrundes wird in dem Programm erzwungen. Wenn wir ein Programm schreiben, das auch einen GPRINT ausführt und das Löschen unterdrückt, so ist ein Großteil des Problems bereits gelöst.

Die ROM-Routine:

```
    LD UH,A          ;Sichern des Punktmusters
    AND A,&F         ;obere Anzeigenhälfte extrahieren
    CP XH,&76        ;muß eine andere Aufteilung erfolgen?
┌   JR C,+16         ;Verzweigung, wenn 3. oder 4. Hälfte angesprochen ist
|   AND (X),&FD      ;Untergrund löschen
|   OR A,(X)         ;Einzeichnen des neuen Bildes
```

```
        LDI (X),A       ;Abspeichern des Bildes und weiterzählen des Pointers X
        LD A,UH         ;Original-Muster holen
        SWP             ;Byte umdrehen (Bit 7-4 mit Bit 3-0 austauschen)
        AND A,&F        ;Der gleiche Vorgang wiederholt sich
        AND (X),&F0:OR A,(X):LDI (X),A
        CP XL,&4E       ;Ende eines Viertels erreicht?
        JR C,+1         ;Ja
        RET             ;Rücksprung mit nächstem Pointer-Wert in X
        LD XL,0         ;Pointer für nächstes Viertel setzen
        INC XH          ; "
        RET             ;Rücksprung mit gesetztem Pointer für nächstes Viertel
        SWP             ;Jetzt muß die Reihenfolge gewechselt werden
        AND (X),&F      ;Hintergrund löschen
        OR A,(X)        ;Neues Bild erzeugen
        LDI (X),A       ;und abspeichern
        LD A,UH         ;Original-Byte holen
        AND A,&F0       ;
        AND (X),&F      ;Hintergrund löschen
        JR -&18         ;Sonst wie vor
```

Nun konstruieren wir um dieses Programm, aus dem wir das Löschen streichen, unser Programm:
Nach dem Aufruf unseres Programmes durch CALL X,String-Variable steht in A die Länge der Variablen (evtl. ist weniger Text in der Variablen). Darum sagen wir erst LD UL,A:DEC UL um den Wert (-1) in unseren Zähler zu laden. Jetzt beginnen wir mit unserer Schleife:

```
LOOP LDI A,(X)  ;Buchstabe holen (in X steht der Anfang des Textes)
     JR Z,ENDE  ;Wenn kein Buchstabe vorhanden ist, Ende
     !HEXBIN!   ;Diese Routine verwandelt ein Hex-Zeichen (Buchstabe) in
                 einen binären Wert (4 Bit)
     SWP:LD UH,A:LDI A,(X)   ;Wert in obere Hälfte übertragen und abspei-
                 chern. Neuen Buchstaben holen.
     JR Z,ENDE  ;Kein Zeichen mehr
     !HEXBIN!:RCF:ADC A,UH   ;Buchstabe in binären Wert wandeln und zum alten
                 Wert addieren
     LD YL,A    ;Punktmuster sichern
     PUSH X     ;Zeiger auf ASCII-Text sichern
     CALL (&8C) ;Die Spaltenadresse nach X laden
     LD A,YL    ;GRPINT-Routine ohne Löschen und ohne Laden von A nach UH,
                 stattdessen wird A später aus YL geladen. Rücksprung ist
                 JR,NEXT
NEXT POP X      ;Text-Pointer wieder laden
     CALL (58E) ;Spaltenpointer weiterzählen (&7875), wenn möglich
     JR C,ENDE  ;Spaltenpointer war schon 155
     DEC UL     ;Zähler decrementieren für das erste Zeichen
     JR NC,ENDE ;Wenn der Text schon zu Ende war, aufhören
     DJC,LOOP   ;Wieder decrementieren und überprüfen, sonst weiter
ENDE RCF:RET    ;C-Flag löschen, damit nach dem Rücksprung die Variable er-
                 halten bleibt.
```

Das Programm wird mit GCURSOR x (das darf entfallen wie bei GPRINT) CALL Startadresse,String-Variable aufgerufen, z.B.:
A$="0102030405060708":CALL X,A$

Das verwendete MAKRO "HEXBIN" sieht so aus:
BIT A,&10 (Buchstabe?) JR NZ,+3 (nein) RCF:ADC A,9 (aus A AND &F 10 machen) AND A,&F (reinen Zahlenwert bilden)

```
;GPRINT
START &5000,&5800,&4000,&5000
=!HEXBIN! BIT A,&10:IF#Z:RCF:ADC A,9:ENDIF:AND A,&F
GPRINT  LD UL,A:DEC UL
LOOP    LDI A,(X):JR Z,ENDE
        !HEXBIN!
        SWP:LD UH,A:LDI A,(X):JR Z,ENDE
        !HEXBIN!:RCF:ADC A,UH:LD YL,A
        PUSH X:CALL (&8C)
;GPRINT
        LD A,YL:AND A,&F:CP XH,&76:JR C,BLOCK2
        OR A,(X):LDI (X),A:LD A,YL:SWP:AND A,&F
GP      OR A,(X):LDI (X),A:CP XL,&4E:JR NC,NEXT
        LD XL,0:INC XH:JR NEXT
BLOCK2  SWP:OR A,(X):LDI (X),A:LD A,YL:AND A,&F0:JR GP
NEXT    POP X:CALL (&8E):JR C,ENDE
        DEC UL:JR NC,ENDE
        DJC LOOP
ENDE    RCF:RET
END
```

```
PROGRAM-CODE:
4000: 2A 62 45 8B
4004: 46 BF 10 89
4008: 03 F9 B3 09
400C: B9 0F F1 28
4010: 45 8B 3C BF
4014: 10 89 03 F9
4018: B3 09 B9 0F
401C: F9 A2 1A FD
4020: 88 CD 8C 14
4024: B9 0F 4C 76
4028: 83 12 0B 41
402C: 14 F1 B9 0F
4030: 0B 41 4E 4E
4034: 81 0E 4A 00
4038: FD 40 8E 08
403C: F1 0B 41 14
4040: B9 F0 9E 14
4044: FD 0A CD 8E
4048: 83 05 62 81
404C: 02 88 4D F9
4050: 9A
```

```
A$="01020408102040":CLS :WAIT 0:CURSOR 10:PRINT "TEST":CURSOR 10
FOR I=1TO 3:CALL &4000,A$:NEXT I
FOR I=1TO 1000:NEXT I
```

Problem: Inverse-Print

In einer der letzen Ausgaben der PC-1500-Zeitung (1983) wurde ein Inverse-Print vorgestellt, das mit REM-Anweisungen arbeitet. Warum so kompliziert?

Hier meine einfachere Lösung:

```
ANFANG LD UL,A:PUSH X:DEC UL  ;Zähler setzen und Text-Anfang sichern
       LD A,255:CALL &EDEF    ;Ausgabe eines senkrechten Striches
       CALL (&8E):JR C,ENDE   ;an der aktuellen Cursor-Position
LOOP   POP X:LDI A,(X)        ;Textpointer wiederholen und Zeichen holen
       PUSH X:JR Z,ENDE       ;Textpointer wieder sichern, wenn kein Zeichen
                               da war, Ende
       CALL &EE48             ;das ROM-Programm errechnet die Startadresse
                               des GPRINT-Musters für den Buchstaben in A
```

```
        PUSH U:LD UL,4          ;alten Zähler sichern und neuen setzen
LOOP 2  LDI A,(Y):XOR A,&FF     ;GPRINT-Muster holen und invertieren
        CALL &EDEF:CALL (&8E)   ;Muster ausgeben und Matrixpointer incrementie-
                                 ren
        DJC LOOP2               ;solange bis 5 Muster übertragen sind.
        LD A,255:CALL &EDEF:CALL (&8E)    ;Zwischenraum ausgeben
        POP U:DJC LOOP          ;alten Zähler wiederholen und weitermachen
ENDE    CALL (&8E)              ;Weiterzählen
ENDE    POP X:RCF:RET           ;Ende
```

Hier werden zwei neue Unterprogramme verwendet:

&EDEF Ausgabe des Punktmusters in A an der Stelle, die durch den Matrixpointer &7875 festgelegt ist

(&8E) Der Matrixpointer &7875 wird weitergezählt, solange er kleiner als 155 ist, sonst ist C gesetzt.

Wenn bei diesem Programm das Ende der Anzeige erreicht wird, stoppt es nicht, sondern schreibt weiter.

Weiter wurde das Unterprogramm &EE48 verwendet, welches aus dem ASCII-Code im Akku die Startadresse des Zeichens in der GPRINT-Tabelle errechnet. In dieser Tabelle sind für jeden Buchstaben 5 GRPINT-Code abgelegt.

```
FF CODE AM-CODE:
4000: 2A FD 88 62
4004: B5 FF BE ED
4008: EF CD 8E 83
400C: 23 FD 0A 45
4010: FD 88 8B 1C
4014: BE EE 48 FD
4018: A8 6A 01 55
401C: BD FF BE ED
4020: EF CD 8E 88
4024: 0A B5 FF BE
4028: ED EF CD 8E
402C: FD 2A 88 23
4030: CD 8E FD 0A
4034: F9 9A
```

```
;INVERSE-PRINT
START &5000,&6000,&4000,&5000
ANFANG LD UL,A:PUSH X:DEC UL:LD A,255:CALL &EDEF:CALL (&8E):JR C,ENDE
LOOP   POP X:LDI A,(X):PUSH X:JR Z,ENDE
       CALL &EE48:PUSH U:LD UL,4
LOOP2  LDI A,(Y):XOR A,255:CALL &EDEF:CALL (&8E):DJC LOOP2
       LD A,255:CALL &EDEF:CALL (&8E):POP U:DJC LOOP
ENDE   CALL (&8E):POP X:RCF:RET
END
CLS :WAIT 0
FOR I=1TO 10:PRINT "-";:A$="*/+":CALL &4000,A$:NEXT I
```

Der Hex-Monitor zum PC-1500

Um etwas mehr Klarheit in die Problematik zu bringen, soll erst einmal der Begriff des Monitors geklärt werden. Ein Monitor ist ein Programm, daß grundlegende Funktionen für den Computer zur Verfügung stellt. Der Computer macht ja ohne ein Programm überhaupt nichts, deswegen wird also ein Monitor programmiert, der für die Abfrage und Decodierung der Tastatur (welche Taste wurde gedrückt, welchen ASCII-Code hat diese?), für die Ansteuerung der Anzeige (Sichtbarmachen von ASCII-Zeichen in der Anzeige), für die Aktivierung des Tongenerators (Töne bestimmter Frequenz und Länge erzeugen), für die Steuerung der eingebauten Uhr (stellen und lesen), für die Handhabung externer Massenspeicher (meist Kassettenrecorder; Daten aufzeichnen, einlesen, auf richtige Aufzeichnung überprüfen) usw. zuständig ist. Je nach Gerät wird der Monitor mehr oder weniger Aufgaben dieser Art erfüllen.

Es sollte jetzt auch klar sein, daß jeder Computer über einen Monitor verfügt. Dieser kann sowohl separat oder auch in ein größeres Programm, meist eine Programmiersprache, integriert vorliegen. Bei den meisten Computern liegt der Monitor in einem Festspeicher (ROM), damit er immer verfügbar ist. Ein Monitor hat meist eine Länge von einigen 1000 Bytes. Die Programmierung eines Monitors ist eine relativ schwierige Angelegenheit, da hier auf nichts zurückgegriffen werden kann, sondern wirklich auch jede kleinste Operation erst geschaffen werden muß. Dabei sind sehr genaue Kenntnisse der Hardware des Computers nötig. Die Programmierung von auf den Monitor zurückgreifender Software ist dann betreffend grundlegender Funktionen nicht mehr so schwierig, weil diese Funktionen als Unterprogramme aus dem Monitor aufgerufen werden können.

Die Bezeichnung Hex-Monitor weist nun darauf hin, daß der Monitor zusätzlich über die Eigenschaft verfügt, hexadezimale Zahlen (Zahlen in einem Zahlensystem mit der Basis 16, diese Zahlen werden vom Mikroprozessor direkt verstanden) ein- und auszugeben. Damit wird also die Möglichkeit geschaffen, mit bestimmten Befehlen mit dem Monitor zu kommunizieren. Dies sind meist Befehle zum Auslisten von Speicherbereichen, Ändern von Speicherstellen, Abspeichern und lesen von Speicherbereichen auf Kassette und Anspringen

von bestimmten Speicherstellen, um ein Programmteil zu starten. Mit diesen Befehlen ist es möglich, Maschinenprogramme zu schreiben, abzuspeichern und auszuführen, allerdings auf höchst umständliche und primitive Weise. Aus diesem Grund wird bei vielen Monitoren auf die hexadezimalen Ein- und Ausgabemöglichkeiten gar nicht hingewiesen, viele Monitore sind in Wirklichkeit Hex-Monitore. Wegen dieser primitiven Möglichkeiten hat man ja auch höhere Programmiersprachen und Hilfsprogramme zur Erzeugung von Maschinenprogrammen, wie z. B. Assembler, geschaffen.

Nun zum Hex-Monitor des PC-1500 . Hier muß erst einmal festgestellt werden, daß es sich gar nicht um einen Hex-Monitor handelt, sondern allenfalls um ein Programm zur hexadezimalen Ein- und Ausgabe. Denn der oben erläuterte Monitor ist beim PC-1500 in Festspeichern ja schon vorhanden, die einzelnen Funktionen brauchen nur noch aufgerufen werden. Man könnte allenfalls behaupten, der sogenannte "Hex-Monitor" erweitert den im PC-1500 vorhandenen Monitor zum "Hex"-Monitor. Indiz dafür ist auch die für einen Monitor viel zu kleine Länge von einigen 100 Bytes (s. o). Das Programmieren dieses "Hex-Monitors" ist dann auch keine große Leistung mehr, die Grundlagen sind in Form des Monitors schon vorhanden, und dem Entfaltungsrahmen des Programmierers sind durch die Aufgabenstellung und die begrenzte Hardware des PC-1500 (winzige Ein-Zeilen-Anzeige für 24 Zeichen) starke Grenzen gesetzt.

Die Schwierigkeiten liegen auf einem ganz anderen Gebiet. Sharp hat nämlich weder den Befehlssatz des im PC-1500 verwendeten Mikroprozessors noch den Aufbau des Monitors der öffentlichkeit zugänglich gemacht, wahrscheinlich aus gutem Grund. Um nun den "Hex-Monitor" in Maschinensprache programmieren zu können, mußten diese Informationen erst einmal gewonnen werden, dieser Vorgang wird im Computer-Jargon mit "Knacken" bezeichnet, bildlich gesprochen bricht man den Computer auf, legt seine Innereien bloß und analysiert deren Aufbau und Funktion. Der erste Schritt ist also, den Befehlssatz des Mikroprozessors zu ermitteln. Dies ist ohne irgendwelche Informationen enorm schwierig. Die Befehle eines solchen Mikroprozessors bestenen ja aus ein bis vier Bytes, jedes Byte kann 256 verschiedene Zustände annehmen. Rein rechnerisch ergeben sich damit 4.294.967.296 verschie-

dene Möglichkeiten. Selbst wenn man einen Großteil dieser Kombinationen von vornherein ausschließen kann, dauert die Analyse und Entschlüsselung viele Monate.

Als nächster Schritt muß dann mit Hilfe des nun bekannten Befehlssatzes des Mikroprozessors die im Gerät vorhandene Software analysiert werden. Da diese ca. 16000 Bytes umfaßt, ist auch diese Aufgabe nicht einfach, vor allem, da keine Hilfsmittel wie Disassembler zur Verfügung stehen, und wird sicher Monate in Anspruch nehmen. Erst jetzt aber, nachdem sämtliche Informationen über den Mikroprozessor und die im Gerät vorhandene Software zur Verfügung stehen, kann mit der Programmierung in Maschinensprache begonnen werden, die nun keine großen Probleme mehr aufwirft und praktisch von jedem versierten Programmierer in kurzer Zeit durchgeführt werden kann, ohne daß sich die Ergebnisse wesentlich unterscheiden würden.

Zum Schluß noch ein Wort zum Kopieren von Software. Jeder Computerbesitzer hat mit seinem Gerät ja schon die vollständige Ausrüstung zur Herstellung qualitativ dem Original in nichts nachstehenden Kopien. Dies ergibt sich einfach daraus, daß die Daten nicht wie beim überspielen von Musik direkt von z. B. einem Recorder auf einen anderen kopiert werden, sondern erst von einem externen Speicher in den Hauptspeicher geladen werden und dann wieder auf einen externen Datenträger geschrieben werden. Es kann also kein Qualitätsverlust eintreten. Die Befehle zum Einlesen und Aufzeichnen sind in der Software des Computers schon vorhanden, so daß man eigentlich niemanden übel nehmen kann, wenn er diese Befehle auch anwendet. Dazu kommt, daß die meisten Softwarehersteller Anleitungen zum Kopieren ihrer Software auch noch mitliefern, die narrensicher abgefaßt sind, so daß auch der letzte mitbekommt, wie Software kopiert wird. Zeigt sich einmal ein Programm hartnäckig, weil der Hersteller kleine Tricks angewendet hat, um daß Kopieren zu Erschweren, so sind für jeden Computer Kopierprogramme erhältlich, die auch mit jeder Manipulation fertig werden. Diese Programme werden z. T. beim Computer- oder Softwarekauf mitgeliefert. Ihre Handhabung ist so einfach, daß selbst völlig unkundige damit fertigwerden: Die ProgrammKassette einlegen, eine Taste drücken, abwarten, eine leere Kassette einlegen, eine Taste drücken, abwarten, fertig. Das Kopieren von Disketten ist noch einfacher.

Vielfach läßt sich auch gar nicht mehr feststellen, wer eigentlich der Urheber eines Programms ist. Die Grenzen zwischen kopierter Software und modifizierter Software oder Eigenentwicklungen sind völlig fließend. Es ist heute ja auch Gang und Gebe, Software aus dem Ausland, meist Japan oder USA, zu importieren, auf deutsche Geräte anzupassen und dann zu behaupten, der Importeur wäre der Urheber. Der Hersteller ist Tausende von Kilometern entfernt und bekommt davon gar nichts mit. Copyright-Hinweise verschwinden meist sehr schnell aus Programmen. Auch Tauschbörsen tragen ihren Teil dazu bei.

Software läßt sich ohne zusätzlichen Hardwareaufwand nicht gegen Kopieren schützen. Der Wettlauf zwischen Schützern und Kopierern ist bisher immer von den Kopierern gewonnen worden. Die Schuld dafür kann nicht unbedingt den Kopierern in die Schuhe geschoben werden. Solange Software ohne jede Schwierigkeit, und sei es durch überall frei erhältliche Kopierprogramme, mit jedem Gerät problemlos kopiert werden kann, und solange Hersteller die Anleitungen dazu auch noch mitliefern und sich dann wundern, wenn Anwender ihre völlig überteuerten Produkte dann auch tatsächlich kopieren, und solange der Computermarkt weiter so explodiert wie heute, solange wird kopiert, und auch wenn einige Hersteller versuchen, mit Hilfe von Gerichten Exempel zu statuieren, wird sich kaum jemand am Kopieren hindern lassen. Das wäre auch nicht im Sinne der Computerindustrie, die darauf angewiesen ist, daß Software für ihre Geräte im Umlauf ist, denn ohne Software läuft nun mal nichts, auch wenn man nicht weiß, wer der Urheber ist!

Zu der Einlassung der Kläger, daß sog. "Trace-Funktion" in dem Hexmonitor eine Besonderheit darstellt

Die Entwicklung des von den Klägern als "Tracer-Funktion" bezeichneten Programmteiles innerhalb ihres Gesamtprogrammes "Hex-Monitor" ist lediglich die Anwendung von im System durch den Hersteller implementierten Befehlen und den durch die Befehle angesprochenen Befehlsroutinen.

Das System enthält die zwanghafte Möglichkeit, den Ablauf sowohl des reinen Betriebsprogrammes, welches der Hersteller implementiert hat, als auch der Anwenderprogramme anzuhalten und den Systemzustand im Moment des Anhaltens in vom Hersteller bestimmten Bereichen festzuhalten.

Zum Zwecke des Anhaltens wird ein ganz bestimmter Speicherteil (eine Speicherzelle) mit einem"Inhalt" als Signal (engl. Flag = Flagge) versehen.

Dieses Flag bewirkt, daß nach Ausführung der Anweisung "unterbrechen" (engl. interrupt) die Daten, die für die Weiterführung des Programmes nach dem Aufheben der Unterbrechung notwendig sind, an vom Hersteller bestimmter Stelle gespeichert werden.

Wenn der Inhalt der "Flagregister-Zelle" gelöscht wird, wird das Programm unter Verwendung der o.a. gespeicherten Daten weitergeführt.

Diese "Anhalte- und Fortführungsbefehle" sind feste, ständig wiederkehrende Routinen, die in den vom Hersteller vorgegebenen Befehlsfolgen für das System z.B. immer dann Anwendung finden, wenn Daten über Drucker, Anzeigeeinrichtungen oder Leitungen an andere Systeme ausgegeben werden. Auch unter anderen Bedingungen, zum Beispiel, wenn das System vom Anwender oder anderen Computereinheiten Daten anfordern muß, ist es notwendig die Befehlsabfolge solange anzuhalten, bis die geforderten Bedingungen erfüllt sind.

Die Befehle lauten "FD,81" (in hexadezimaler Schreibweise) - (dezimal = 253,129) = Interruptenable-Flag setzen und "FD,BE" (in hex.) - (dez. = 253,190) = Interruptenable-Flag löschen. (Zur Erklärung: ein Computer kann nur Zahlenwerte verarbeiten. Die sog. Befehle sind tatsächlich nur Positionen einer Zahlentabelle. Die durch die Zahlenwerte bestimmte Position enthält weiterführende Zahlenwerte, die wiederum Befehle symbolisieren, sodaß durch das Benennen einer "Befehlsposition" tatsächlich immer eine Folge von durch den Hersteller vorgegebenen Befehlen, sog. Routinen ausgelöst werden.) Die Kläger verwenden für die wenig einprägsamen Zahlenkombinationen wie üblich Abkürzungen des englischen Sprachgebrauches. Für "Flag setzen" = EI, für "Flag löschen = DI (enableinterrupt bezw. disenableinterrupt). Beweis siehe das "Systemhandbuch" der Kläger.

Die Kläger verwenden diese Befehle in ihrem Dienstprogramm "Hex-Monitor" natürlich auch (Beweis siehe in ihrem Systemhandbuch - Beschreibung bei "Run-Mode - Funktion: F)

Die nach dem jeweiligen Anhalten der Programmausführung mögliche Darstellung der Systemzustände oder des Inhaltes einzelner Speicherstellen richtet sich nach den Bedürfnissen der Anwender.

Im Idealfall würde für jeden Anwender seine spezifische Darstellung möglich sein.

Dies würde jedoch erfordern, daß für jeden Anwender ein besonderes nur für ihn erstelltes Dienstprogramm unter Berücksichtigung seiner speziellen Bedürfnisse und Geräteausstattungen erstellt werden müßte.

In der Praxis hat sich jedoch herausgestellt, daß für den Anwender zumeist bei Verzicht ganz spezieller Darstellungen ein bestimmtes Maß an Informationen (die Erfahrungen vieler Anwender ergaben dieses Maß) ausreichend ist.

Das Programm der Kläger bietet lediglich einige von fast unendlich vielen Darstellungsmöglichkeiten, wie die Kläger selbst durch Verweis auf andere "Hex-Monitore" zugeben. Die Kläger geben auch in ihren eigenen Einlassungen zu, daß sie bei der Auswahl der Möglichkeiten auf bestimmte Geräteausstattungen der Anwender Rücksicht genommen haben. Für Anwender mit z.B. besserer Speicherausstattung - sprich mehr Speicherplatz ist der "Hex-Monitor durchaus nicht befriedigend, da sie bedingt durch ihre bessere Speicherausstattung komfortablere Darstellungen benötigen.

Die Kläger bezeichnen in ihrem Werbematerial ihren "Hexmonitor" als " ein leistungsfähiges Werkzeug " sprich "Dienstprogramm" und geben an unterschiedlichen Stellen zu, daß sie Systemteile für ihr Programm verwenden, ja sie warnen sogar wegen dieser Verwendung vor Fehlern, die bei Anwendung ihres Programmes entstehen können. Beweis siehe zwei Ablichtungen aus ihrem Werbematerial.

Peripherie- Beschreibungen und Hardware- Erweiterungen

CE-153 Softwareboard

Laut Handbuch des CE-153 Softwareboardes wird zum Swb. eine Kassette geliefert mit dem Systemprogramm und zwei Beispielprogrammen. Diese Kassette ist am besten abspielbar auf dem CE-152-Recorder (!). Für das Systemprogramm müssen 1066 Byte gesichert werden im BASIC-Bereich. Das Systemprogramm dient der Abfrage der Tastatur des Softwareboardes (140 Tasten).Das normale Program wird unter Berücksichtigung des 2.Zeichensatzes aufgerufen und nimmt Eingaben von der Tastatur des PC-1500 entgegen (--> Z$(0)) und eine Taste vom Softwareboard als 4stelligen Code in Z$(Enter kann für Taste gedrückt werden).

Die Tasten des Softwareboardes sind also nicht programmierbar, sonderr der Benutzer erhält immer den Code der gedrückten Tasten (Zeile/Spalte). Die Beispielprogramme auf Kassette sind "Sales Management" und "Word Master". Beim Ersten handelt es sich um ein kleines Kalkulationsprogramm (nicht Tabellenkalkulation). Das Zweite ist ein Programm, welches das Lernen von engl. Wörtern erleichtern soll. Es sind fest Wörter für die Tastatur vorgesehen, die man ausgeben kann, oder bei denen man das eigene Wissen überprüfen kann. Im Anhang des Handbuches sind drei weitere Programme aufgeführt. "Zeichenprogramm" erstellt mit Hilfe des Softwareboardes als Menuebrett eine Zeichnung, wobei feste Figuren abgerufen werden können. "Sternenposition" zeichnet über dem Softwareboardmenue ausgewählte Sternbilder an eingegebenen Tagen und Orten. "Lagerhaltung" ist ein einfaches Lagerverwaltungsprogramm (ähnlich aus dem PC-1500-Manual).
Das Softwareboard ist also nicht ganz als 2.Tastatur mit belegten Tasten zu gebrauchen, kann aber Programgestützt eingesetzt werden. Das Softwareboard wird über ein Kabel mit dem PC-1500 verbunden und mit Tastatur-Schablonen beschriftet.


HIERMIT BESTELLE ICH DAS INTERFACE CE-158, DAMIT ICH ENDLICH MEINEN MATRIXDRUCKER DORT AN DEN PC-1500 ANSCHLIESSEN KANN!

Parallel-Serielles Interface CE-158

Das CE-158 stellt eine Verbindung zwischen dem PC-1500 und der datenverarbeitenden Umwelt her. Es enthält zwei genormte Schnittstellen (Centronics 8 bit parallel + RS-232 (V24) seriell). Die parallele Schnittstelle kann nur zur Ausgabe verwendet werden (Drucker o.ä.), während die serielle Schnittstelle zur Daten- bzw. Programmein- und Ausgabe benutzt werden kann.

1. RS-232-Schnittstelle

   Diese kann in zwei Betriebsarten benutzt werden.

   a) Terminalmode

      Nach Eingabe der Anweisung DTE bzw. TERMINAL kann der PC-1500 ein Terminal bzw. Datenendgerät ersetzen. Über verschiedene Menues wird die Betriebsart festgelegt. Der Unterschied zwischen DTE und TERMINAL besteht darin, daß bei TERMINAL die alten Übertragungsparameter erhalten bleiben, außerdem unterscheiden sich die default-Werte der anderen Parameter.
      Über die Menues, die durch die ↑-und ↓-Tasten angesteuert werden, sind die folgenden Anwahlen möglich.

      Terminal:

      Ent - Start des Terminalbetriebes
      Aut - Start mit automatischer Aussendung des SIGN ON-Telegrammes (autom.Logon)
      Quit - Rückkehr zum Basic-Modus

      Setup:

      Aut - Festlegung des SIGN ON-Telegrammes
      Fnc - Belegung der Funktions(Soft)-Tasten für Ebene 1
      Com - Festlegung der Übertragungsparameter: Puffergröße, Baud-Rate (Übertragungsgeschwindigkeit 50, 100, 110, 200, 300, 600, 1200, 2400 bits/sec = baud), Wortlänge (5-8 bit), Parity (odd, even, none), Stop-Bits (1, 1.5, 2)

Operate:

Nrm - Normalmodus

A/P - Page-Modus , Übertragungsstop nach 512 bytes

A/L - Line-Modus , Übertragungsstop am Zeilenende

Protocol:

Xon/Xoff - Übertragungsstop durch Sendung von XOFF(DC3)--CTRL5, weiter bei XON(DC1)-CTRLQ bzw. DTR--Leitung

Echo - Eingegebene Zeichen werden im Display automatisch angezeigt

Bei Betätigung von SHIFT ↑ erscheint das folgende Menue.

Output:

Ext - Einschalten des Druckers (Parallelport)

TRC - Trace, d.h. Übertragung der empfangenen Daten über den Parallelport

DSP - Clean Text-Modus, wort- oder zeichenweises Rollen der Anzeige

ETX - Einschalten der ETX-Sendung bei Drücken der CL-Taste

Diese Menues können durch die ↑-und ↓-Tasten zyklisch angewählt werden. Aus dem Terminal-Menue kann man nun den Terminal-Betrieb starten. Je nach Speicherausbau des PC-1500 steht ein unterschiedlich großer Empfangspuffer zur Verfügung. Zum Interface wird eine entsprechende Tastatur-Schablone geliefert. Nach Stoppen des Empfangs mittels XOFF-Taste kann der Display-Inhalt mit den Cursor-Tasten auf dem Empfangspuffer verschoben werden (Fensterfunktion).

b) Basic-Mode

Sämtliche Funktionen der RS-232-Schnittstelle können über Basic angesprochen werden. Hierzu dienen die schon bekannten Anweisungen zur Ein- und Ausgabe (PRINT, INPUT, LPRINT, CLOAD, CSAVE, MERGE, PRINT#, INPUT#), die über eine spezielle Steueranweisung auf den seriellen Port umgeleitet werden. Neu hinzu kommen die

folgenden Funktionen:

SETCOM - Übertragungsparameter festlegen
COM$ - Übertragungsparameter
SETDEV - Umleitung der I/O-Funktionen auf den seriellen Port
DEV$ - Zuordnung von I/O-Anweisungen zum seriellen Port
OUTSTAT - RTS-, DTR-Signal festlegen
INSTAT - Eingabe der Schnittstellensignale
INPUT$ - Eingabe ohne Umsetzung von Ausdrücken (bei alphanumerischen Variablen wird z.B. SIN 30 nicht als Sinus aufgefaßt)
INPUT% - Einlesen einer Zeichenfeldvariablen
CSAVEa - Programm im ASCII-Code senden
CSAVEr - Reserve-Bereich übertragen
CLOADa, CLOADr } entsprechende Ladeinstruktionen
MERGEa - ASCII-Programmdaten hinzuladen
TRANSMIT
BREAK n - n = 1-255; Für eine gewisse Zeit (n/64 sec) Leerstellen übertragen
RINKEY$ - entsprechend INKEY$ für serielle Schnittstelle
ERN - Fehlercode
ERL - Fehlerzeile, nützlich bei ON ERROR GOTO
SPACE$n - Zeichenkette mit n Leerstellen
FEED o.
FEED n - Ende-Zeichen übertragen
CONSOLEn - Zeilenbreite festlegen
CONSOLE n,x,y - Zeilenbreite und Ende-Code festlegen:
bzw. n,x x,y = 0 : CR ; x,y = 1 : LF
ZONE n - Festlegung der Spaltenbreite bei Benutzung von Kommata innerhalb LPRINT

2. Die Parallelschnittstelle

Durch OPN"LPRT" werden alle LPRINT, LLIST, FEED und CONSOLE-Kommandos über das Parallelport (z.B. Drucker mit Centronics-Schnittstelle) ausgeführt.
OPN macht diese Zuordnung rückgängig.

3. Hinweise zum Anschluß

a) RS-232-Schnittstelle

Bei dieser Schnittstelle handelt es sich um eine voll modemfähige Ausführung, d.h. neben den Datenleitungen TD und RD werden auch Steuersignale bedient. Möchte man ein Terminal o.ä. anschließen, welches nur eine eingeschränkte Schnittstelle (drei Leitungen - RD, TD und GND) besitzt, so müssen auf der CE-158 Seite folgende Signale gebrückt werden:

RTS - CTS (4 ↔ 5)
DSR - DTR (6 ↔ 20)

Nach Verbinden von Punkt 3 ↔ 2, 2 ↔ 3, 7 ↔ 7 kann nun der PC-1500 mit einem anderen Datenendgerät kommunizieren.

b) Parallelschnittstelle

Neben den acht Datenleitungen müssen auf jeden Fall BUSY und STROBE verdrahtet werden. Alle Datenleitungen sollten als "twisted Pair", d.h. mit einer GND--Leitung verdrillt, ausgeführt werden

Programmvektoren bei Verwendung des CE-161

(vergl. Heft 8 / Dez. 1983, Seite 13)

Nach SHARP-Systemhandbuch, Seite 122, werden aus den ersten 8 Speicherstellen eines ROM bei Bedarf Informationen für das Betriebssystem entnommen. Steht in der Adr. 0000H ein 55H, so werden beim Einschalten die drei folgenden Bytes (ROM-Top = Lage des Reservespeichers, BASIC-Top H und BASIC-Top L) in die Systemadressen 7860H, 7861H und 7862H übertragen; sonst steht hier FFH. Nach NEW Ø steht dann in 7865/66H als BASIC-Beginn 4000H und nicht 40C5, weil der Reservespeicher sich - definiert durch das ROM-Top-Byte - weiter vorn befindet! Steht in den Adressen 7861/62H nicht 55H, so werden diese beiden Bytes als BASIC-Top interpretiert und die Werte in 7865/66H ignoriert. Dies alles gilt im Prinzip auch für den CE-161, wenn er als ROM geschaltet ist.

Ist der in der Adresse 0000H des als ROM geschalteten CE-161 aber nicht 55H, dann enthalten die Systemadressen 7860-62H FFH und 7863/64H die Werte 40H und 48H, d.h. der als ROM geschaltete CE-161 wird für den programmierbaren BASIC-Bereich ignoriert. BASIC-Top nach NEW Ø ist dann 40C5H.

Aus alle dem folgt: Richtig arbeitet der CE-161 nur, wenn er entweder als RAM geschaltet ist, oder wenn man sich im ROM-Betrieb streng an die Bedienungsanleitung hält (d.h. u.a. Füllen aller nicht benutzten Speicherstellen mit FFH)!

Will man jedoch wechselweise den CE-161 einmal programmierbar machen (RAM), ihn dann aber beim Programmtest wieder zerstörungssicher schalten (ROM), so empfiehlt es sich, die ersten 4 Bytes des CE-161 wie folgt zu belegen:

0000H: 55H
0001H: ØØH

0002H: BASIC-Top H
0003H: BASIC-Top L

Damit bleiben dann ROM-Top (≙ Reservebereich) und BASIC-Top beim Umschalten unverändert.

Zerstörung des Speicherinhalts des CE 161 durch fehlerhaftes(?) PC-1500-Betriebssystem:

Vorgehen bei der Programmierung des CE 161 für den Nur-Lese-Betrieb nach Bedienungsanleitung CE 161 Seite 66, Abschnitt (3):

1) Einstellschalter in Lösestllg.
2) Computer ausschalten,
   CE 161 anschließen
3) NEW Ø eintasten
   (Damit liegt der BASIC-Beginn
   bei &ØØC5)
4) ... SOFT-Taste (was ist das?)
5) Das (nebenstehende) BASIC-
   Programm eingeben, insbesondere
   auch den "Tippfehler" in Z. 70!

Original-Pgm:

```
10:"A":WAIT 100
20:PRINT "VERSUCH
   S-PGM"
30:PRINT "FUER AD
   R &00FF"
40 BEEP 3
50:PRINT "ADRESSE
    &00FF OK."
60:U=220
70:I=0:REM FEHLER
80:R=U/I
```

Zerstörtes Pgm:

```
10:"A":WAIT 100
20:PRINT "VERSUCH
   S-PGM"
30:PRINT "FUER AD
   R &00FF"
40 q 3
50:PRINT "ADRESSE
    &00FF OK."
60:U=220
70:I=0:REM FEHLER
80:R=U/I
```

Bevor man schließlich entsprechend Punkt 11) das Programm sichert, probiert man es aus, indem man es mit DEF A startet:
Es werden zunächst alle Print-Zeilen ordnungsgemäß angezeigt, also auch "ADRESSE &ØØFF OK.". Sodann bleibt das Programm erwartungsgemäß stehen mit der Meldung "ERROR 38 IN 80" (Division durch Null). Weiter nach Bedienungsanleitung PC-1500, Seite 36, Abschnitt 14.7: Taste [↑] drücken, um sich die fehlerbehaftete Zeile 80 anzeigen zu lassen. Fehlermeldung mit [CL] löschen (das Korrigieren des I=Ø kann man sich hier sparen). Neustart mit DEF A. Jetzt bleibt das Programm bereits in Zeile 40 hängen mit der Meldung "ERROR 21 in 40". Das Beepen und die Anzeige "ADRESSE &ØØFF OK." unterbleiben. Das Betätigen der Taste [↑] zeigt:
Das Basic-Kommando BEEP in Zeile 40 ist zerstört.

URSACHE:

In Adresse &ØØFF des RAM werden durch das Betätigen der Taste [↑] im RUN-Modus nach ERROR-Meldung oder nach BREAK Bit 7 und Bit 1 auf Null gesetzt!

C E - 1 6 5

In der,in der letzten Ausgabe der PC-1500-Zeitung veröffentlichten, SHARP-Preisliste tauchten unter anderem das Programmiergerät für CE-160 (CE-165) und das 8-K--ROM-Modul CE-160 auf. Obwohl diese Geräte für den privaten PC-1500-Anwender, aufgrund der in der Liste genannten Unverbindl. Preisempf. (CE-160: 398,-DM ; CE-165: 3290,-DM), kaum in Betracht kommen werden, soll im Folgenden kurz erläutert werden, was es mit ihnen auf sich hat.
In Verbindung mit dem CE-165 lassen sich die CE-160 mit Programmen beschreiben, so daß man Softwaremodule erhält, die sich in das Modulfach des PC-1500 einstecken und nicht ohne weiteres löschen bzw. verändern lassen. Zu diesem Zweck besitzt das CE-165 sechzehn Stecksockel zur Aufnahme von CE-160-Modulen und einen Anschluß, der die Verbindung zum PC-1500 darstellt. Der anzuschliessende PC-1500 muß mit dem 8-K-RAM-Modul CE-155 und wahlweise mit dem Softwareboard CE-153, falls dieses vom Programm benötigt wird, verwendet werden. Auf diese Art lassen sich bis zu sechzehn Module gleichzeitig mit dem im PC-1500 und CE-155 befindlichen Programm beschreiben. Die Programmlänge kann bis zu 7800 Bytes, mit Softwareboard bis zu 6733 Bytes, betragen. Durch eine Verify-Funktion ist die Kontrolle des korrekten Überspielens des Programms möglich.

Der PC-1500 wird zum PC-1500 A !

Sie können nicht "nur" Programme schreiben, sondern auch mit Werkzeug (Schraubenzieher, Lötkolben usw.) umgehen ...? Dann machen Sie doch aus Ihrem PC-1500 einen mit einem "A" !

Sie brauchen dazu nicht viel, nur ein RAM 6116LP-3 o.ä., ca. einen Meter sehr dünnen Draht (0.1..0.3 mm Durchmesser und möglichst lötbarer Isolation), sowie den Schaltplan des Gerätes (und ein bischen Mut?!?). Sollten Sie keinen Schaltplan besitzen, richten Sie sich genau nach der Skizze.

Worum geht es ?

Der Bereich &7000 - &7FFF (System-RAM) umfaßt 2 kBytes. Durch Verwendung der RAM-Bausteine TC 5514 (1k·4 bit) ist jedoch nur der untere Bereich &7000 - &7BFF voll dekodiert. Durch Ersatz dieser IC's durch ein RAM 6116 wird der gesamte Bereich dekodiert und Ihnen stehen somit 1kByte für Maschinen-Programme im Bereich &7C01 - &7FFF zur freien Verfügung.

Was ist zu tun ?

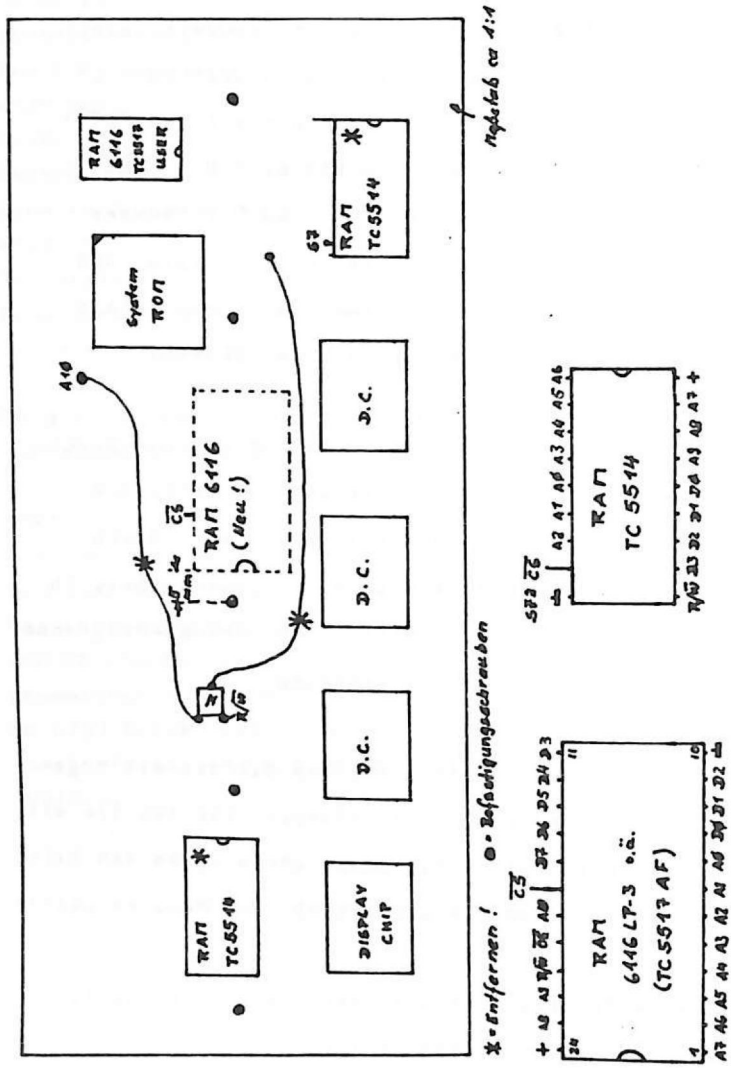
Nehmen Sie die Skizze zur Hand und entfernen Sie die mit * gekennteichneten Bauteile und Draht-Verbindungen. Kneifen Sie mit einem (scharfen) Seitenschneider die Beine Ihres neuen RAM 6116 in einem Abstand von ca. 2 mm vom Gehäuse ab. Nur Mut, es passiert schon nichts !

Kleben Sie mit einem Tropfen Kontaktkleber das so behandelte IC an die in der Skizze bezeichnete Stelle.

Jetzt gehts ans Anschließen! Hierbei ist mit größter Sorgfalt vorzugehen!!

Übersichts-Skizze und Anschlüsse

PC-1500 und RAM 6116 bzw. TC 5514



Die gekürzten Anschlüsse des neuen RAM's sind mit denen des eingebauten Flat-Pak 6116 (oder TC 5517) zu verbinden, d.h. Sie löten Ihren Draht einfach auf die Beine drauf. Mit einer Ausnahme: Die Leitung $\overline{CS}$ ist mit S7 zu verbinden!

Sind alle Anschlüsse richtig beschaltet und die Batterien wieder eingesetzt, so muß sich der Rechner mit dem üblichen NEWØ:CHECK melden. Nach Initialisierung testen Sie die Funktion des neuen Speicherbereichs mit POKE&7CØ1,&55 und PEEK&7CØ1 = 85 dezimal, sowie POKE&7CØ1,&AA und PEEK&7CØ1 = 170 dezimal.
Hat es funktioniert? Dann fixieren Sie die Drähte mit Kleber und bauen den Rechner zusammen.
Ein kleines Prüfprogramm testet den gesamten neuen Speicherbereich komplett:

```
10 FOR I=&CØ1 TO &FFF: POKE(&7ØØØ+I),&55: IF PEEK(&7ØØØ+I)=&55
                                                    NEXT I
15 PRINT (&7ØØØ+I),PEEK (&7ØØØ+I)
```

Dasselbe Programm läßt man anschließend mit dem Wert &AA (anstatt &55) ablaufen. Wird nach Beendigung auf dem Display 32767 85 bzw. 170 angezeigt, haben Sie gut gearbeitet, d.h. alles läuft fehlerfrei!

Warum &55 bzw. &AA?

Vielleicht wundern Sie sich über die Prüfwerte. Die Erklärung ist einfach: &55 bedeutet binär 01010101 ; &AA aber 10101010 ! Innerhalb eins Bytes ist damit jedes Bit entgegengesetzt dem seiner Nachbarschaft.So können nicht ansprechbare Speicherzellen oder Kurzschlüsse zuverlässig erkannt werden. Anhand des ange-

zeigten Wertes kann die Adresse sowie das Datenbit ermittelt werden, falls ein Fehler vorliegt!

Hat alles geklappt?

Dann besitzen Sie eine wichtige Option des PC-1500A, d.h. Sie verfügen einen Maschinensprach-Bereich von 1 kByte im Adreßraum &7CØ1 - &7FFF.

Abhängig von der System-ROM-Version kann die Adresse &7CØØ mit &ØD überschrieben werden und sollte deshalb nicht benutzt werden!!

EIN PC-1500 IM ENDSPURT !!

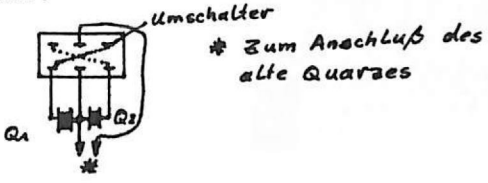
Recht lange Verarbeitungszeiten bei komplexen Rechenoperationen, logischen Vergleichen und FOR...NEXT oder GOSUB Anweisungen, sowie langsames, gemütliches Drucken ist man beim PC-1500 gewöhnt. Aber es geht auch schneller !!

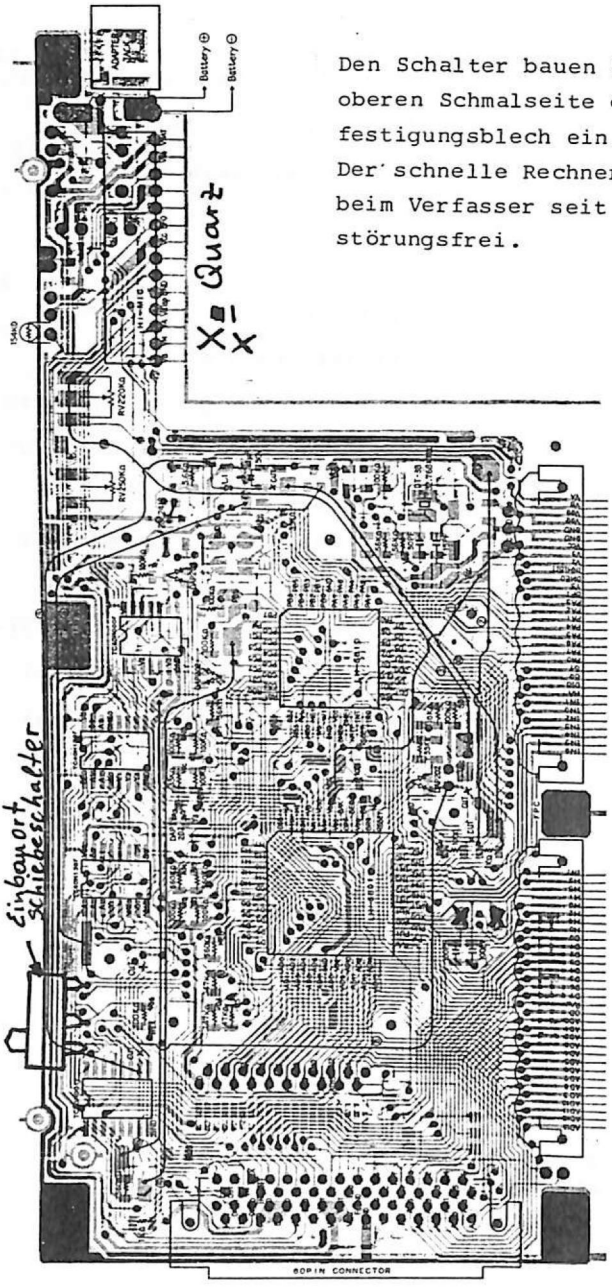
"Tunen" Sie Ihren Rechner!

Der Systemtakt beträgt 1.3 MHz, abgeleitet vom standartmäßig eingebauten Quarz 2.6 MHz. Lt. Datenblatt kann die CPU aber durchaus mit einem 2 MHz-Takt (aus einem 4 MHz Quarz) arbeiten. Eingehende Untersuchungen ergaben, daß für die peripheren Bausteine die kürzere Dekodierungszeit völlig ausreichend ist. Dieser "Speed-up" macht sich vor allem bei Plotter bemerkbar; Sie werden Staunen, wie schnell das Listing Ihres Programms auf Papier erscheint! Das Speichern auf Cassette erfolgt natürlich auch mit höherer Frequenz. Allein deshalb (um kompatibel zu "langsameren Brüdern" zu bleiben) sollte die Quarzfrequenz umschaltbar gemacht werden.

Was ist zu tun?

Sie besorgen sich einen 4 MHz-Quarz und einen Umschalter (Umschalter mit 2 Ebenen), natürlich in Miniatur-Ausführung. Entfernen Sie den alten Quarz,er befindet sich auf der Rückseite der Bodenprintplatte (ein grüner Knubbel mit 2 Beinen). Verschalten sie den Schalter und die beiden Quarze folgendermaßen:





Den Schalter bauen Sie am besten an der oberen Schmalseite des Rechners am Befestigungsblech ein.
Der schnelle Rechner + Peripherie läuft beim Verfasser seit ca. 1 Jahr völlig störungsfrei.

-33-

CPU: LH 5801

8 Bit, CMOS
Taktfrequenz: 2,6 MHz
intern: 1,3 MHz
80 Befehle
6 Register
64 KByte Direkt-
adressierung
8 Bit Port
3 Interruptmöglich-
keiten
Timerfunktion

verkauft:

Fischel Betriebstechnischliches
Berechnungs- und Programmierbüro GmbH
Kaiser-Friedrich-Straße 84a
1000 Berlin 12 · Tel. 323 60 29

aus dem Service Manual PC-1500 mit "Ersatzteil-liste"


DU HORNOCHSE! PYTHAGORAS HAT NOCH OHNE PC-1500 SEINEN SATZ BEWIESEN!
KULTURFILM AG
REGIE

Programmlistings

HARDCOPY, Übertragen des Display-Inhaltes auf den Plotter

Dieses Unterprogramm stellt das Display samt Inhalt nahezu maßstabsgetreu auf dem CE-150-Plotter dar. Es kann mittels des MERGE-Befehles zu beliebigen Programmen hinzugeladen werden. Von diesen Programmen wird es dann mit GOSUB "Z" aufgerufen und es plottet dim aktuelle Anzeige mit Anzeige der gewählten Reserve-Ebene und des Winkelmodes. Die gewünschte Farbe ist vor dem Aufruf von "HARDCOPY" festzulegen. Die ersten beiden Zeilen enthalten eine Abfrageroutine, die es ermöglicht, ohne einen Plot sofort in das Hauptprogramm zurück zukehren. Soll ein Plot ausgeführt werden, so ist die "J"-Taste zudrücken. Bei einem Drücken einer beliebigen anderen Taste findet der Rücksprung in das Hauptprogramm statt.
Soll das Programm direkt von der Tastatur abgerufen werden, muß der Start auf jeden Fall mit DEF Z erfolgen, so daß die Anzeige erhalten bleibt. Zusätzlich sind die folgenden Änderungen durch zuführen: Streichen der ersten beiden Zeilen, Setzen von Label "Z" in 65271 und Ersetzen des RETURN-Befehles in Zeile 65279 durch die END-Anweisung.

```
65269 "Z"PAUSE :W$=INKEY$ :IF W$=""GOTO 65269
65270 IF W$<>"J"RETURN
65271 GRAPH :LINE (40,-300)-(40,0)-(90,0)-(90,-300):ROTATE 1:CSIZE 1
65272 K=PEEK &764FAND 7:W$="RAD":J=0:IF K<>4LET W$="G"+W$:J=1:IF K=3LET W$="DEG"
:J=3
65273 GLCURSOR (78,-8):LPRINT "BUSY":GLCURSOR (78,J-258):LPRINT W$:GLCURSOR (78,
-315):LPRINT "RUN":GLCURSOR (0,-4):SORGN
65274 W$="":FOR K=0TO 5-LN (PEEK &764EAND 112):W$=W$+"I":NEXT K
65275 FOR K=0TO 155:A=POINT K:IF A=0GOTO 65279
65276 FOR J=6TO 0STEP -1:IF A<2^JGOTO 65278
65277 A=A-2^J:L=7-J:LINE (L*4+41,-K*4)-(L*4+43,-3-K*4),,,B
65278 NEXT J
65279 NEXT K:LINE (40,-295)-(40,-624)-(90,-624)-(90,-295):GLCURSOR (78,-545):LPR
INT W$:TEXT :LF 7:RETURN
```

Sortierprogramm für Zeichenketten und Zahlen

Dieses Programm sortiert Zeichenketten in alphabetischer Reihenfolge. Maximal zulässig sind 256 Elemente. Diese Beschränkung rührt daher, daß sich beim PC-1500 Arrays nur mit max. 256 Elementen definieren lassen. Das Programm wird mit RUN"SORT" gestartet. Es fragt nach der geschätzten Anzahl der zusortierenden Elemente. Diese Zahl darf höher sein als die Zahl der tatsächlichen Elemente, denn wird bei der Frage nach einem Element nur die ENTER-Taste betätigt, so wird das Einlesen abgebrochen und der Sortiervorgang beginnt. Sind die Elemente in richtiger Reihenfolge, erfolgt der Ausdruck der sortierten Liste auf dem CE-150. Die Druckroutine ist in Zeile 60 geschrieben. Wird hier der Befehl LPRINT in ein einfaches PRINT umgewandelt, so läßt sich das Programm auch ohne den Drucker betreiben. Sollen anstatt von Zeichenketten numerische Ausdrücke sortiert werden, muß im Programm das Array A$ in ein numerisches umgewandelt werden (z.B. B(N)).

```
    5 "SORT"CLEAR :CLS :WAIT 0:INPUT "Elemente (max.256)?";N:N=N-1:DIM A(99),A$(
N)
   10 FOR I=0TO N:CLS :PRINT STR$ (I+1);". ";:INPUT A$(I):NEXT I:A(1)=N
   15 L=A(X):Y=A(X+1):X=X-2
   20 K=L:J=Y:D=(L+Y)/2:A$=A$(D)
   25 IF A$(K)<A$LET K=K+1:GOTO 40
   30 IF A$(J)>A$LET J=J-1:GOTO 40
   35 IF K<=JLET B$=A$(K):A$(K)=A$(J):A$(J)=B$:K=K+1:J=J-1
   40 IF K<=JGOTO 25
   45 IF K<YLET X=X+2:A(X)=K:A(X+1)=Y
   50 Y=J:IF L<YGOTO 20
   55 IF X<>-2GOTO 15
   60 LPRINT "Sortierte Folge :":LPRINT "----------------":FOR I=0TO N:LPRINT A
$(I):NEXT I:END
```

Funktionseingabe durch INPUT

Dieses Unterprogramm erspart bei Programmen, die mit vorgegebenen Funktionen rechnen sollen, ein Umschalten zwischen RUN- und PRO-Mode, falls die Funktion geändert werden soll.
Bei Aufruf durch DEF A wird nach der einzugebenden Funktion y=f(x) gefragt. Als Eingabewerte können die Ziffern 0 bis 9 und sämtliche mathematischen Funktionen (s.Bedienungsanleitung) des PC-1500 verwendet werden. Weiterhin ist der Großbuchstabe "X" als Eingabewert erlaubt. Sollen weitere Buchstaben oder Zeichen verwendet werden können, so kann die Abfrage in Zeile 30 mit dem entsprechenden ASCII-Code ergänzt werden. Bei der Eingabe dürfen beliebig viele Leerzeichen benutzt werden, sofern diese nicht ein Basic-Schlüsselwort teilen. Die Gesamtlänge der Eingabe darf 70 Zeichen nicht überschreiten.
Das Programm schreibt Einzelzeichen sofort, Basic-Wörter nach Übersetzung in die entsprechenden Token, in Zeile 5. Bei der Umsetzung der Basic-Wörter finden die beiden DATA-Zeilen Verwendung. In ihnen befinden sich die verwendbaren Basic-Begriffe mit dem jeweils zweiten Byte ihres Tokens als Zeichenketten abgespeichert. Lediglich bei den Funktionen SIN, COS und TAN können die ASCII-Zeichen,die dem zweiten Byte des Token entsprechen würden, nicht direkt eingegeben werden. So muß beim Eingeben des Listings darauf geachtet werden, daß nach Zeile 35 die im Listing aufgeführten POKE-Befehle Anwendung finden.
Die Reihenfolge, der in den DATA-Zeilen stehenden Begriffe, sollte möglichst eingehalten werden, da ansonsten Fehler im Programmablauf auftreten könnten. Sollte eine Zeile nicht auf Anhieb einzugeben sein, evtl. zu lang, so muß erst ein Teil der Zeile eingegeben werden und nach einem ENTER-Befehl der Rest 'angehängt' werden.

Soll die eingegebene Funktion vom Hauptprogramm aus gelesen werden, so muß in letzterem ein GOSUB "FX" stehen.

Das gesamte Unterprogramm kann sowohl in ein Programm eingebaut werden, als auch zu einem bereits vorhandenen Programm mittels des MERGE-Befehles dazugeladen werden. Im letzteren Fall ist in Zeile 60 ein RETURN zu ergänzen. Auf jeden Fall ist darauf zu achten, daß sich die Zeilen 5 und 10 im selben Programmblock wie die restlichen Zeilen befinden und daß sie die ersten beiden Zeilen dieses Blockes darstellen !!!

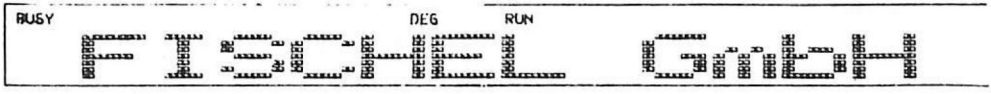
```
   5 "FX"REM y=f(x).....65.beliebige.Zeichen.!.....
.....
   10 RETURN
   15 "A"GRAPH :DIM FF$(0)*70:P=STATUS 3+1:POKE P,PEEK P-&80:P=PEEK &789E*256+PE
EK &789F+7:POKE P,89,61:P=P+2
   20 CLS :FF$="":INPUT "f(x)= ";FF$:CLS
   25 FOR I=1TO LEN FF$:A=ASC MID$ (FF$,I,1):IF A=32GOTO 60
   30 IF (A)39AND A<>44AND A<58)OR A=88OR A=91OR A=93OR A=94GOTO 55
   35 DATA "ABSp","INTq","EXPx","ATNu","ACSt","ASNs","LOGw","SIN ","COS ","TAN "

POKE STATUS 2- 4,&7F
POKE STATUS 2-11,&7E
POKE STATUS 2-18,&7D

   40 DATA "SGNy","LNv","SQR√","PIπ","EE"
   45 RESTORE 35:FOR J=1TO 15:READ A$:L=LEN A$-1:IF LEFT$ (A$,L)<>MID$ (FF$,I,L)
NEXT J:BEEP 3:PAUSE "? ? ?":GOTO 20
   50 A=ASC RIGHT$ (A$,1):I=I+L-1:IF J<13POKE P,&F1,A:P=P+2:GOTO 60
   55 POKE P,A:P=P+1
   60 NEXT I:POKE P,58,&F1,&AB
```

INTEGRATION

Zusammen mit dem Programm FUNKTIONSEINGABE DURCH INPUT bilden die Zeilen 65-105 ein vollstaendiges Integrationsprogramm. Die Berechnung erfolgt nach der Rechteckmethode. Eine gesonderte Erklaerung der Zeilen 65-105 eruebrigt sich, da sie sich weitgehend waehrend des Programmablaufs selbst erklaeren sollten.

```
 65:CLEAR :WAIT 0:
    INPUT "Untergr
    enze:";XU,"Obe
    rgrenze:";XO,"
    Schrittweite:"
    ;DX
 70:L=INT ((XO-XU)
    /DX):DX=(XO-XU
    )/L:X=XU:FOR I
    =0TO L-1:X=XU+
    I*DX:GOSUB 5
 75:PRINT USING "#
    ######.####";X
    :I1=I1+Y:NEXT
    I:USING
 80:I1=I1*DX:X=XU:
    GOSUB 5:YU=Y:X
    =XO:GOSUB 5:YO
    =Y:I2=I1-YU*DX
    +YO*DX
 85:M=INT LOG DX:I
    =INT ((I1+I2)/
    (2*10^M)+.5)*1
    0^M
 90:BEEP 3:PRINT X
    U;" ";:GPRINT
    "40403E0101";:
    WAIT :PRINT XO
    ;" =";I
 95:INPUT "Andere
    Grenzen (J/_)
    ? ";A$:IF
    LEFT$ (A$,1)="
    J"GOTO 65
100:A$="":INPUT "N
    eue Funktion (
    J/_) ? ";A$:IF
    LEFT$ (A$,1)="
    J"GOTO 15
105:END
```

Beispiel zu "HARDCOPY" (vgl. S. 147)



CREF, Erstellung einer Referenzliste

Dieses Programm erstellt eine Liste, aus der ersichtlich ist, welche Programmzeilen des bearbeitenden Programmes von wo aus angesprungen werden. Dabei werden sowohl die Sprungbefehle GOTO, GOSUB, RESTORE, THEN und ON...GOTO..., als auch Labels (z.B."A"), welche allerdings als Zeilennummern in der Liste auftauchen, berücksichtigt.
Das Programm wird mit dem MERGE-Befehl zu dem zu bearbeitenden Programm geladen und mit RUN"CREF" gestartet. Zur korrekten Ausführung des Programmes ist ein ausreichender Speicherplatz notwendig, da "CREF" nach dem Start zunächst eine interne Liste hinter den Programmblöcken aufbaut. Für jeden Sprung im zubearbeitenden Programm werden 4 Bytes zusätzlichen Speicherplatzes benötigt. Vor Programmstart sollte also abgeschätzt werden, ob der zur Verfügung stehende Platz ausreicht. Sollte dies nicht der Fall sein, so tritt bei der Bearbeitung öfter die Zeilennummer 65535 auf. Verändert sich dann auch die Anzeige, muß das Programm abgebrochen werden und der Programmspeicher mit NEW gelöscht werden. Das zubearbeitende Programm sollte also vor der Anwendung von "CREF" schon auf Cassette gespeichert sein.
Bei fehlerfreiem Programmablauf löscht sich "CREF" selbst, so daß nach Erstellung der Liste normal weitergearbeitet werden kann.
Als "Fehler" werden von "CREF" genannt: Sprungbefehle zur Zeile Ø (z.B. ON ERROR GOTO Ø), Sprungbefehle hinter denen weitere Berechnungen stehen (aufgeführt als Sprung nach 65535) und Sprungbefehle, die auf nicht existente Zeilen verweisen.

```
 1 "CREF"CLS :WAIT 0:CURSOR 6:TEXT :CSIZE 1:C=255:D=256:A=STATUS 2-STATUS 1:P
OKE 28750,0,0:K=STATUS 2:LF 3:I=A
 2 E=PEEK &7869*D+PEEK &786A-1:LPRINT " Zeile/ von":PRINT "* * Cref * *":USIN
G "######"
 3 FOR J=I+3TO I+PEEK (I+2)-1:IF PEEK J<>241GOTO 13
 4 J=J+1:P=PEEK J:IF P<>146AND P<>148AND P<>167AND P<>174GOTO 13
```

```
 5 A$="":G=0
 6 J=J+1:P=PEEK J:G=GOR (P>229):IF P<>13AND P<>44AND P<>58LET A$=A$+CHR$ P:GO
TO 6
 7 IF A$=""GOTO 12
 8 IF ASC A$=34GOTO 33
 9 IF VAL A$=OOR G=1LPRINT A$;"  ";PEEK I*D+PEEK (I+1):GOTO 12
10 F=VAL A$
11 B=INT (F/D):POKE K,B,F-D*B,PEEK I,PEEK (I+1):K=K+4
12 IF P=44GOTO 5
13 IF P=171LET J=E
14 NEXT J:GOSUB 40:IF FGOTO 3
15 K=K-4:I=A
16 G=PEEK I:H=PEEK (I+1):B=0:FOR J=STATUS 2TO KSTEP 4:IF PEEK J<>GOR PEEK (J+
1)<>HGOTO 20
17 IF B=0LPRINT G*D+H;:B=1
18 IF B>=6LPRINT :TAB 6:B=1
19 LPRINT PEEK (J+2)*D+PEEK (J+3);:B=B+1:POKE J,0,0,0,0
20 NEXT J:IF BLPRINT
21 GOSUB 40:IF FGOTO 16
22 B=0:FOR J=STATUS 2TO KSTEP 4:G=PEEK J:H=PEEK (J+1):IF G=0AND H=0GOTO 25
23 IF B=0LPRINT "Fehler:":LPRINT "von Zeile   nach Zeile":B=1
24 LPRINT PEEK (J+2)*D+PEEK (J+3);:TAB 11:LPRINT G*D+H
25 NEXT J:I=A:B=0
26 J=I+3:IF PEEK J<>34GOTO 31
27 IF B=0LPRINT "Label:":B=1
28 LPRINT PEEK I*D+PEEK (I+1);:TAB 10:LPRINT CHR$ 34;
29 J=J+1:P=PEEK J:IF P<>34AND P<>13LPRINT CHR$ P;:GOTO 29
30 LPRINT CHR$ 34
31 GOSUB 40:IF FGOTO 26
32 POKE 30823,E/D,EAND C,PEEK 30821,PEEK 30822:POKE 28750,65,67:LF 3:END
33 U=I:I=A:IF ASC RIGHT$ (A$,1)<>34LET A$=A$+CHR$ 34
34 IF PEEK (I+3)<>34GOTO 37
35 W=I+2+LEN A$:FOR V=I+3TO W:IF ASC MID$ (A$,V-I-2,1)<>PEEK VLET V=W:NEXT V:
GOTO 37
36 NEXT V:F=PEEK I*D+PEEK (I+1):GOTO 39
37 GOSUB 40:IF FGOTO 34
38 F=65535
39 I=U:GOTO 11
40 I=I+3+PEEK (I+2):F=(I<E)AND (PEEK I<>C):RETURN
```

PC-1500-Programm RETTE

Dieses Prgm. kann Prgm.e wiederherstellen, die durch NEW, Kassettenfehler oder Maschinensprache zerstört sind.

Laden Sie dazu das zerstörte Prgm. auf Band:

1.Fall Das Prgm. ist nicht mehr von der Kassette zu lesen. Hier können Sie den Teil noch retten, den Sie noch einlesen können.

2. Fall: Das Prgm. ist durch Neu gelöscht. Hier müssen Sie erst mit dem Befehl CSAVE M"TEXT";STATUS2/STATUS3" das noch vorhandene Prgm. auf Band sichern.

3. Fall: Das Prgm. ist durch Maschinensprache zerstört. Hier können Sie Das Prgm. mit CSAVE auf Band sichern.

Jetzt laden Sie das Prgm. Rette in den Rechner und laden das zerstörte Prgm. dazu mit Merge/CLOADM"TEXT";STATUS2(für 2.Fall). Sie starten das Prgm mit RUN"RETTE. Es erscheint ein Text in der Anzeige:

xxxxx yyy:_____ XXXXX=Zeilennummer YYY=Cursor-Position

Der Rechner faßt Ihr zerstörtes Prgm. als langen Text auf, den Sie jetzt wieder neu einteilen müssen.Dazu stehen Ihnen folgende Befehle zur Verfügung:

E Hier Ende der aktuellen Prgm.-Zeile setzen

Z Neue Zeilennummer festlegen.

S Hier Schluß. Rette verschwindet aus dem Computer.(Nur nach E).Die angezeigte Zeile wird nicht mehr mit übernommen.

C Cursor-Position eingeben

T Cursor springt auf Ende der Prgm.-Zeile.(geht nur, wenn Zeiger noch richtig)

P Ausgabe des ASCII-Wertes des aktuellen Zeichens unter dem Cursor.

SHIFT Eingabe eines ASCII-Zeichens, welches auf die Cursor-Position kommt.

▶Eine Stelle rechts

◀Eine Stelle links

Prgm.-Zeilen sind so aufgebaut, daß am Ende ein ENTER steht, welches als Leerzeichen dargestellt wird und auf welches der Cursor stehen muß, wenn Sie E eingeben, da sonst die nächste Zeilennummer falsch wird. Besteht das zerstörte Prgm. aus zwei mit Merge verbundenen Teilen, so erscheint einmal in der Anzeige am Zeilenende (nach ENTER) ein Kästchen, das den ASCII-Wert 255 hat. Das ist eine Marke, und die müssen Sie mit in die Zeile aufnehmen, weil sonst die nachfolgenden Zeilennummern falsch erscheinen. Dieses Zeichen können Sie später wenn das Prgm. fertig ist von Hand wieder löschen. Nachdem Sie das reparierte Prgm. auf Band gesichert haben müssen Sie deb Rechner mit NEW 0 wieder neu initialisieren.Befehle erscheinen nicht als Buchstaben, sondern als 2 Zeichen(Byte).

```
1:REM BBBBBBBBB
2:CLS :G=ASC
  INKEY$ :IF G
  BEEP 1,5,2:
  GOTO 13
3:GOTO 7
4:"RETTE"C=255:D
  =256:A=PEEK &7
  869*D+PEEK &78
  6A:BEEP ON :A=
  A+STATUS 1*(A=
  STATUS 2-
  STATUS 1):Y=A:
  WAIT 0
5:B=STATUS 2-
  STATUS 1+5:
  POKE B,&FD,&6A
  ,&B5,20,&BE,&E
  D,0,&9A
6:X=1:WAIT 0:Z=
  PEEK Y*D+PEEK
  (Y+1)
7:CLS :PRINT
  USING "######"
  ;Z;USING "####
  ";X;":";:I=Y+2
  +X:F=I-(X-4)*(
  X<4)-3:CALL B,
  F
8:T=(10+X)*(X<4)
  +14*(X>3):S=0
9:S=1-S:F=0:
  CURSOR T:PRINT
  CHR$ (127*(S=1
  )+PEEK (I)*(S=
  0))
10:F=F+1:C=ASC
  INKEY$ :IF F=5
  GOTO 9
11:IF G=0GOTO 10
12:F=0:CURSOR T:
  PRINT CHR$
  PEEK I
13:IF G=9CLS :
  INPUT "Nummer:
  ";Z:POKE Y,Z/D
  ,Z-PEEK Y*D:
  GOTO 6
14:IF G=6POKE I,
  13:POKE Y+2,X:
  Y=I+1.GOTO 6
15:IF G=1CLS :
  INPUT "ASCII-W
  ert:";F:F=ABS
  INT F:IF F<0
  POKE I,F:GOTO
  7
16:IF G=12AND X<C
  LET X=X+1:GOTO
  2
17:IF G=8AND X>1
  LET X=X-1:GOTO
  2
18:IF G=8POKE Y,
  C:POKE &7865,A
  /D,AAND C,Y/D,
  YAND C,A/D,A
  AND C:END
19:IF G=10AND
  PEEK Y=CLET Y=
  Y+1:GOTO 6
20:IF G=8WAIT :
  PRINT PEEK Y:
  WAIT 0
21:IF G=8LET X=
  PEEK (Y+2):
  GOTO 7
22:IF G=6CLS :
  INPUT "Cursor
  ";F.F=INT ABS
  F.IF F>0AND F<
  OLET X=F:GOTO
  7
23:GOTO 7
```

Fischel Betriebswirtschaftlicher
Beratungs- und Programmierdienst GmbH
Kaiser-Friedrich-Straße 54a
1000 Berlin 12 - Tel. 323 60 29

Der PC-1500 verfügt in Verbindung mit dem Plotter CE 150 über eine komfortable und leistungsstarke Graphikausgabe. Trotzdem stellt sich in der Praxis oft die Frage nach dem Verlauf einer Kurve oder Funktion, ohne daß man jedesmal ein spezielles Plotprogramm erstellen möchte. Es ergibt sich also die Forderung nach einem Programm mit dem möglichst jede beliebige Funktion graphisch dargestellt werden kann.
Das vorliegende Plotprogramm ermöglicht es, einerseits mit Ausnutzung der Voreinstellung nur durch Eingabe der Funktionsvorschrift eine Funktion darzustellen, andererseits können Koordinatensystem (cartesisch, polar), Schrittweite, automatische Schrittweitenanpassung, Ausschnittvergrößerungen, Bildgröße, Achsenskalierung und Farben frei gewählt werden. Man kann beliebig viele Kurven in ein Bild legen. Die Eingabeparameter können zum Schluß in übersichtlicher Form ausgelistet werden. Durch eine besondere Art der Fehlerbehandlung erfolgt kein Abbruch des Programms an nicht definierten Stellen der Funktion (z.B. bei f(x) = 1/x und x = 0). Ein besonderer Vorteil dieses Programms besteht in der automatischen Schrittweitenanpassung. Sie ist hauptsächlich beim erstmaligen Plotten von Funktionen und bei Unstetigkeitsstellen zu empfehlen, da die Schrittweite dx für das Plotten an jeder Stelle optimal gewählt wird. Bei wenig gekrümmten Teilen ergibt sich ein großes dx, während bei starken Krümmungen und Knicken dx rechtzeitig verringert wird. Man erhält also in jedem Fall eine einwandfreie Kurve, ohne sich um Schrittweiten kümmern zu müssen.

EINGABEDATEN

Die darzustellende Funktion wird in Zeile 1 in der Form y = f(x) eingetragen, z.B.

```
1: Y = X^3
```

Dann wird das Programm mit Def A gestartet. Beantwortet man alle Fragen nur mit ENTER, so ergibt sich folgendes Bild:

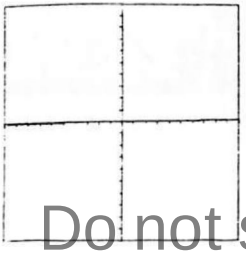


Bild 3

Fischel Betriebswirtschaftlicher
Beratungs- und Programmierdienst GmbH
Kaiser-Friedrich-Straße 54a
1000 Berlin 12 - Tel. 323 60 29

Die graphische Darstellung kann jedoch über die Eingabedaten vielfältig geändert werden.
Die erste Abfrage "Wertebereich x" fordert die Werte für x_{min}, x_{max} und delta x nach Bild 4. In der nächsten Abfrage "Wertebereich y" werden die entsprechenden Werte für y_{min}, y_{max} und delta y verlangt.

"Hy" ist die Höhe in mm des y-Wertebereichs auf der Zeichnung. Wird ein Rahmen gewünscht, gibt man bei der Abfrage "Rahmen" eine 1, sonst eine 0 ein.

Die Frage "Farbe" kann mit 0,1,2 oder 3 (schwarz, blau, grün, rot) beantwortet werden. Für die Eingabe bei "dx" gibt es drei grundsätzlich verschiedene Möglichkeiten:

1.) dx = Zahlenwert größer 0
Es wird mit der konstanten Schrittweite des Zahlenwertes geplottet.

2.) dx = 0
Die Schrittweite ist variabel und wird dem Funktionsverlauf automatisch optimal angepaßt.

3.) dx = Zahlenwert kleiner 0
Die Darstellung der Funktion erfolgt in Polarkoordinaten mit der konstanten Schrittweite des Zahlenwertes in Grad. Das Plotten in Polarkoordinaten wird erst beim Betätigen einer beliebigen Taste unterbrochen!

Will man mehrere Kurven darstellen, so schreibt man die neue Funktion in Zeile 1 und startet mit Def N. Es werden darauf erneut "Farbe" und "dx" verlangt. Ein Wechsel des Koordinatensystems (cartesisch, polar) ist also jederzeit möglich.

Def L listet die Werte für x_{min}, y_{min}, x_{max}, y_{max}, delta x, delta y, Maßstab in x-Richtung (1/mm), Maßstab in y-Richtung (1/mm) sowie die Funktionsvorschrift in übersichtlicher Form aus.

Bild 5 zeigt für die oben vorgestellte Funktion einen Ausschnitt, in dem zusätzlich die 1. und 2. Ableitung zu sehen sind. Die Eingabedaten sind der Tabelle unter der Zeichnung zu entnehmen.

Fischel Betriebswirtschaftlicher
Beratungs- und Programmierdienst GmbH
Kaiser-Friedrich-Straße 54 a
1000 Berlin 12 - Tel. 323 60 29

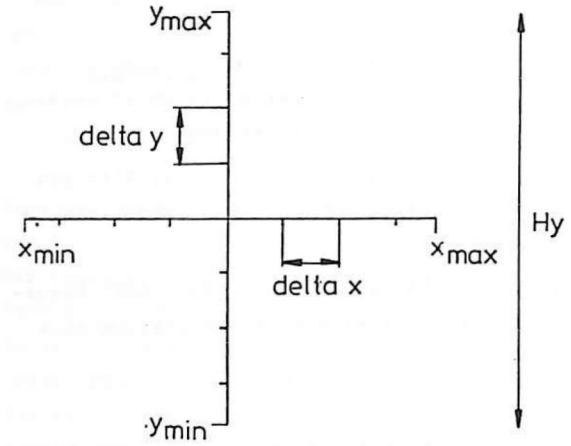


Bild 4

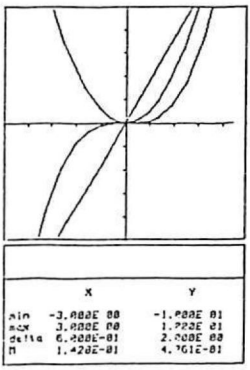
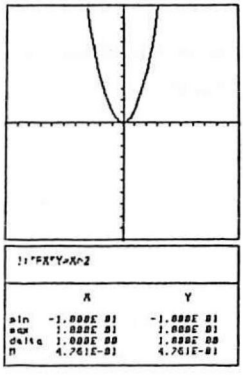
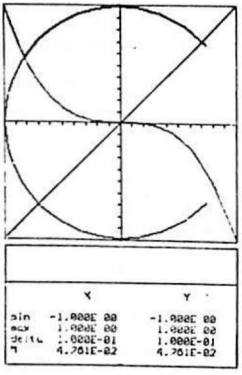
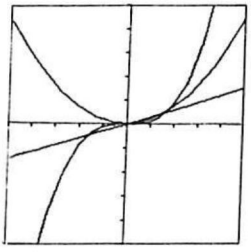


Bild 5

Die folgenden Bilder zeigen Beispiele verschiedener Funktionen und Kurven. Alle Funktionen im cartesischen Koordinatensystem sind mit variabler Schrittweite geplottet.







Fischel Betriebswirtschaftlicher
Beratungs- und Programmierdienst GmbH
Kaiser-Friedrich-Straße 54a
1000 Berlin 12 - Tel. 323 60 29

```
  1:Y=_____
  2:RETURN
 10:"A"CLEAR :GRAPH
 20:P=1:N=42:XI=-10:XA=10:D=1:YI=-10:YA=10:E=1:A=1
100:INPUT "Wertebereich X :";XI,XA,D
120:INPUT "Wertebereich Y :";YI,YA,E
130:INPUT "Hy ";N,"Rahmen ";A
135:MX=210/(XA-XI):MY=5*N/(YA-YI):U=MY/MX:H=D
140:COLOR 0:GLCURSOR (-XI*MX,-YA*MY):SORGN
170:X=XI:B=3+YI*MY:IF SGN YI<>SGN YALINE (XA*MX,0)-(XI*MX,0):B=0
180:GLCURSOR (X*MX,B):RLINE -(0,-3):IF X<XALET X=X+D:GOTO 180
190:Y=YI:B=3+XI*MX:IF SGN XI<>SGN XAAND XI<>0LINE (0,YA*MY)-(0,YI*MY):B=0
200:GLCURSOR (B,Y*MY):RLINE -(-3,0):IF Y<YALET Y=Y+E:GOTO 200
205:IF ALINE (MX*XI,MY*YI)-(MX*XA,MY*YA),0,0,B
206:"N"INPUT "Farbe = ";L,"dx = ";K
207:ON ERROR GOTO 1000
208:WAIT 0:COLOR L
209:IF K<0GOTO 500
210:D=(XA-XI)/80:O=2*ATN 1E99:X=XI
220:COLOR L:IF K<>0LET D=K
240:IF X>XAGOTO 270
245:W=Y:GOSUB 1:IF Y>YAOR Y<YIGOTO "EXT"
247:IF X=XIGLCURSOR (X*MX,Y*MY)
250:LINE -(X*MX,Y*MY)
255:IF K=0GOSUB "DX"
260:X=X+D:GOTO 240
270:N=X:X=XA:W=Y:GOSUB 1:IF Y<YIOR Y>YALET X=N:GOTO 600
275:LINE -(X*MX,Y*MY)
280:GLCURSOR (XI*MX,YI*MY)
300:END
400:"L"SORGN :LINE (0,-110)-(210,-5),,0,B:LINE (0,-35)-(210,-35)
404:GLCURSOR (0,-10):TEXT :CSIZE 1:LF 4
408:LPRINT TAB 12;"X";TAB 27;"Y":LF 1
410:RESTORE :USING "##.###^":FOR I=1TO 4:READ A$,A,B:LPRINT TAB 1;A$;TAB
    7;A;TAB 22;B:NEXT I:LF 3
420:DATA "min",XI,YI,"max",XA,YA,"delta",H,E,"M",5/MX,5/MY
425:LF -13:LLIST 1:LF 6
430:END
500:F=0:D=ATN 1E99*-K/90
510:X=F:GOSUB 1
520:X=Y*COS F:Y=Y*SIN F
530:IF PLINE -(X*MX,Y*MY):GOTO 550
540:GLCURSOR (X*MX,Y*MY)
550:A$=INKEY$ :IF A$=""LET F=F+D:GOTO 510
560:GOTO 280
```

Fischel Betriebswirtschaftlicher
Beratungs- und Programmierdienst GmbH
Kaiser-Friedrich-Straße 54a
1000 Berlin 12 - Tel. 323 60 29


```
600:"EXT"R=YA:S=YI:IF Y<YILET R=YI:S=YA
605:PRINT "Y > Yg !"
610:IF X>XILINE ((X-D)*MX,W*MY)-(MX*((R-W)*D/(Y-W)+X-D),R*MY)
620:W=Y:N=X:X=X+D
621:IF X<XAGOSUB 1:IF K=0GOSUB "DX"
622:IF X>XAGOTO 280
623:IF Y<YIOR Y>YAGOTO 620
625:IF SGN R<>SGN WLET R=S
627:IF KLET D=K
630:CLS :LINE (MX*((X-N)*(W-R)/(W-Y)+N),R*MY)-(X*MX,Y*MY)
650:J=1:GOTO 255
900:"DX"Z=X:U=Y
910:IF D*MX<=.5LET D=.5/MX:X=Z:Y=U:RETURN
915:X=Z+D:GOSUB 1:A=Y
920:X=Z+D+D:GOSUB 1:B=Y
930:C=ATN (U*(A-U)/D):O=ATN (U*(B-A)/D)
940:G=ABS (O-C)/Q
945:IF G>.25LET D=D/10:GOTO 910
950:D=D*.0225*EXP (3.8*(1.08-G))
960:X=Z:Y=U:RETURN
1000:C=PEEK &789B:IF C>36AND C<42GOTO 620
1010:WAIT :BEEP 3:PRINT "Fehler Nr";C:END
```

Sehr geehrter Herr

bei dem Programm "Funktionsplot" muß die gewünschte Funktion in die 1. Programmzeile eingetragen werden, z.B.:

1: "FX"Y = X ^ 2

für eine Parabel. Vor der Funktionsvorschrift darf aber kein REM stehen, da sonst die folgenden Befehle nicht vom Rechner abgearbeitet werden.

Wenn Sie die obige Zeile eintragen und nach dem Programmstart alle Eingaben einfach mit ENTER beantworten, bekommen Sie den beiliegenden Ausdruck. Ich hoffe, daß das Programm damit zu Ihrer Zufriedenheit läuft. Falls noch Fragen auftreten, wenden Sie sich bitte direkt an mich.

Mit freundlichen Grüßen

H. Siepmann

FISCHEL

BETRIEBSWIRTSCHAFTLICHER BERATUNGS- UND PROGRAMMIERDIENST GMBH

Die Unternehmensberatung

für SHARP-Computer

alle Preise incl. 7% MWST:

49,-DM

39,-DM

49,-DM

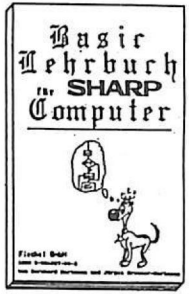
49,-DM

39,-DM

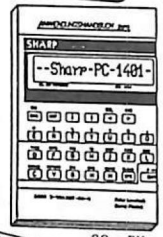
User-Club Deutschland

DURCH INFORMATION VORN

29,-DM



BASIC-Lehrbuch für Sharp Computer
160 Seiten: Preis: 49,-DM



39,-DM

ALLE VERGANGENEN HEFTE SIND NOCH LIEFERBAR !!
Bitte siehe Bestellschein für ein Abonnement. 6,-DM pro Heft

49,-DM

ISBN 3-924327-00-9