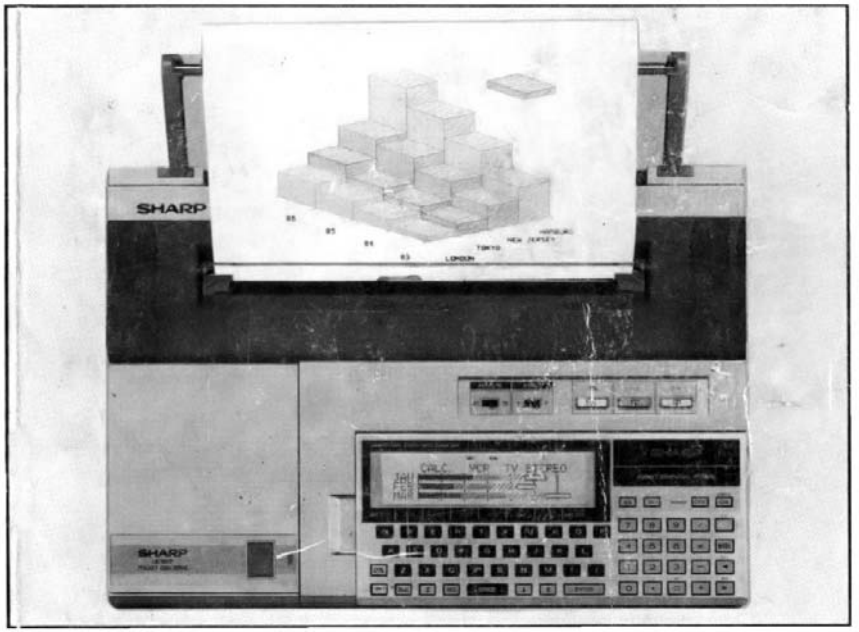
Maschinensprachehandbuch zum SHARP PC-1600 Taschencomputer



ISBN 3-89374-001-5

Thomas Jeger

Fischel GmbH

PC-1600 Maschinensprachehandbuch

Herausgeber: Fischel GmbH
Kaiser-Friedrich-Str. 54a

D-1000 Berlin 12
Deutschland

L?3, &hj2.. (uk3/) 001101 &93 LD 32,24 HL aL, DE

PC-1600 Maschinensprache Handbuch

LD A,&52 INC A RST 00 LD (&2731),HL RET INC B

Inhaltsverzeichnis

Grundlagen:

Befehlserklärungen:

PC-1600 intern:

IOCS:

Anhang:

Sharp Microcomputer .....
..... Fischel GmbH
Kaiser-Friedrich-Str. 54 a
D- 1000 Berlin 12 .....
Tel . 030 / 323 60 29
Mo -Fr 10 - 18. 00, Sa - 14 h

VORWORT

Dieses Buch ist für alle Freunde der Maschinensprache gedacht; egal ob Anfänger oder bereits Fortgeschrittener. Von Grund auf kann hier Sinn und Zweck, Schwierigkeiten und Vorteile der Programmierunq in Maschinensprache erlernt werden. Kenntnisse der Materie, werden keine vorausgesetzt. Im ersten Teil lernen Sie das Programmieren des Z80 Prozessors, währenddem der zweite Tei1 spezifisch auf Eigenheiten des PC-1600 abgestimmt ist.
Ebenfalls in diesem Buch ist ein Assembler-Programm zur Eingabe Ihrer Maschinensprache-Pprogramme in Basic abgedruckt. Dies ist ein wichtiges Hilfsmittel für das Studium und die Erstellung von Maschinensprachprogrammen.

Muntelier

Thomas Jeger

Maschinensprache (Nachfolgend im ganzen Buch "ML" für "Maschine Language" genannt ist die Sprache, die der Prozessor versteht. Der Prozessor hat, Register zur Verfügung, die entweder als Zeiger (auf bestimmte Adressenoder als Datenträger aufzufassen sind.
Ein Vergleich mit einer anderen Sprache liegt auf der Hand. Nehmen wir als Vergleichsmitteilung die Programmiersprache BASIC da diese im PC-1600 “eingebaut” ist:
Die Variablen (also z.B. A(10), D$)stehen für die Register. Aber auch Befehlswörter wie “GOTO”, “END”, etc. sind in der Maschinensprache enthalten.
Maschinensprache besteht im Prinzip nur aus Zahlen, die der Prozessor als Befehlswörter oder Daten interpretiert. Eine solche Programmierung ist unübersichtlich und führt deshalb schnell zu Fehlern, die nicht ohne weiteres zu korrigiern sind. Deshalb bedienen wir uns der Assemblersprache, die weitaus übersichtlicher ist. In der Assemblersprache
steht für jenen Befehl ein Wort, (ähnlich wie
Basic). So bedeutet "RET" das Beenden eines ML- Programmes. In der ML steht hierfür nur der hexadezimale Code "&C9". Würde man in eigentlicher Maschinensprache programmieren, könnte schnell ein kleiner Tippfehler (z.b. ”&C8'', "&D9'', etc.) passieren. Da der Code ”&C8” für etwas ganz anderes steht (z.B. für einen Sprung-, oder Ladebefehl),kann das Programm sehr leicht abstürzen.
Aus diesem Grunde ist in diesem Buch einAssemblerprogramm abgedruckt (in BASIC), *welches* Ihre Assemblerprogramme in ML-Programme
übersetzt.

Es gibt zwei Hauptgründe, wieso ein Interesse bestehehen könnte, in Maschinensprache zu Programmieren: Die erhöhte Arbeitsgeschwindigkeit und der niederigere Speicherplatzbedarf gegenüber irgend einer anderen Programmiersprache.
Da man in Maschinensprache auf Prozessorebene programmiert, braucht es weder Interpreter (wie bei den meisten BASIC-Versionen; so auch PC1600) noch Compiler (z.B. PASCAL) . Bei. einem Interpreter wird das Programm Schritt für Schritt aufgeteilt und ausgeführt; dies beansprucht enorm Zeit. Beim Compiler wird zwar in eine maschinennahe Sprache übersetzt (grosse Geschwindigkeit) , man muss aber extrem viel Speicherplatz besitzen. Bei der Programmierung in ML entfallen sowohl Geschwindigkeitseinbussen wie auch Speicherplatzverschwendung. Ein weiterer Vorteil der ML ist, dass man Zugriff auf die Systemebene hat, d.h. man kann Sachen machen, die in BASIC schlicht unmöglich wären (Z.B. Interruptprogramme).

Natürlich gibt es auch Nachteile mit ML: Der Aufwand um nur ein kurzes Programm zum Laufen zu bringen, ist unwahrscheinlich gross. Ein anderer Nachteil ist der komplizierte Umgang mit Zahlen.
Es gibt durchaus Anwendungen (z.B. kleinere mathematische Probleme wie quadratische Gleichungen) bei denen sich der Einsatz der ML nicht lohnt. Die Entscheidung, welche Program-miersprache eingesetzt wird, liegt beim Pro-grammierer. Es sollte auch erwägt werden, mehrere Programmiersprachen (z.B. BASIC und ML) in einem Programm einzusetzen. So kann ein Spielprogramm an Geschwindigkeit gewinnen, wenn z.B. die Ausgabe-routine auf das Display in ML abgefasst ist.

Binäres Zahlensystem:

Im Alltag rechnen wir im Dezimalsystem. Dies kommt wahrscheinlich davon, weil wir 10 Finger haben. Wir rechnen also zur Basis 10, d.h. wir können eine beliebige Zahl wiefolgt schreiben:

z.B. 392: $2*10^0 + 9*10^1 + 3*10^2$
$2 + 90 + 300 = 392$

Da aber der Computer nur 2 Zustände unterscheiden kann ("ein"/"aus"), rechnet er zur Basis 2. Dieses Zahlensystem nennt man Binärsystem. Auch hierzu ein Beispiel.

z.B. 10110 : $0*2^0 + 1*2^1 + 1*2^2 + 0*2^3 + 1*2^4$
dez. : $0 + 2 + 4 + 0 + 16$
$= 22$

Im Anhang am Schluss des Buches finden Sie eine Umrechnugstabelle für die Zahlen 0-255 ins binäre und hexadezimale System.

Dieses System hat aber den Nachteil, dass man bereits bei kleinen Zahlen, einen grossen, unübersichtlichen Strang von "1" und "0" bekommt. Dazu kommt noch, dass dieser relativ kompliziert umzurechnen ist.

Aus diesem Grund hat man das hexadezimalsystem

eingeführt.. Es unterscheidet statt nur 2 Zustände gleich deren 16. Um solche Zahlen nun aufschreiben zu können, "entlieh" man dem dezimalen Zahlensystem die Zahlen 0-9 und fügte die Buchstaben A-F dazu. Die dezimale Zahl 12 z.B. schreibt man hexadezimal also "C".

Umsetzungstabelle:

dez.	hex.	dez.	hex.	dez.	hex.	dez.	hex.
0	&0	4	&4	8	&8	12	&C
1	&1	5	&5	9	&9	13	&D
2	&2	6	&6	10	&A	14	&E
3	&3	7	&7	11	&B	15	&F

Zur besseren Unterscheidung zwischen dezimalen und hexadezimalen Zahlen, führen wir hier das Zeichen "&" ein, welches jeder hexadezimalen Zahl voransteht. Die Wahl fiel auf dieses Zeichen, da der PC-1600 bei der Eingabe einer hexadezimaler Zahl in dieser Form, diese ins dezimale Zahlensystem umrechnet.

Auch hier ein Zahlenbeispiel:

```
 z.B. &E3A:        A         3          E
      dez.:      10*160 +  3*161 + 14*162 = 3642
```

Eine binäre Zahl ins hexadezimale System umzurechnen, bereitet uns weniger Mühe: Wir splitten die binäre Zahl in 4er Päckchen auf und rechnen diese einzeln um.
Beispiel:
z.B. 1110110101:

```
                    11  1011 0101
                 &  3     B    5
   =&3B5
   = 949 (dez.)
```

In der Assemblersprache braucht man vorwiegend das Hexadezimalesystem. Das binäre Zahlensystem braucht man insbesondere bei der Erstellung von Display-Grafiken und Bitmanipulationen. Das dezimale Zahlensystem kommt in der ML in der Regel nicht vor.

Bevor wir uns auf die verschiedenen Gruppen von Befehlen und Adressierungsarten stürzen, müssen noch diverse Begriffe erklärt werden.
Wie in BASIC oder Pascal gibt es in ML auch Variablen, nur werden sie hier Register und Flags genannt. Wir unterscheiden folgende Register:

A, B, C, D, E, H, L	sind 8-Bit-Register
IX, IY, BC, DE, HL	sind 16-Bit-Register

Mit 8-Bit-Registern kann man Grössen von (dez.) 0-255 darstellen. Mit 16-Bit-Registern Grössen von 0-65535 (&0 - &FFFF).
Die Register BC, DE und HL sind zwar 16-BitRegister, setzen sich aber aus den 8-BitRegistern B&C, D&E sowie H&L zusammen.
Der Z80-Prozessor stellt die 8-Bit-Register doppelt zur Verfügung, d.h. mit einem Befehl kann man die aktuelle Belegung mit einer anderen (im internen RAM des Prozessors) austauschen. Somit hat man quasi das doppelte der Anzahl 8-Bit-Register zur Verfügung.
Nun kennt der Z8O-Prozessor noch diverse Flags. Flags sind im Prinzip 1-Bit-Register, da sie nur die Zustände 1 und 0 unterscheiden. Diese Flags braucht der Prozessor um z.B. einen Überlauf eines Registers anzuzeigen. Hier alle Flags und ihre Bedeutungen:

S	Vorzeichenflag (1, wenn negativ)
Z	Nullflag (1, wenn Resultat =0)
AC	Hilfsübertragflag (auch H genannt)
P	Paritätflag (1, wenn überlauf auftrat)
N	Stibtraktionsflag
CY	Übertrag-Flag (Carryflag, auch C genannt)

8-Bit-Ladebefehle:

LD A,	A.B.C.D.E.H.L.xx.(xxyy).(BC).(DE).(HL).(IX+zz).(IY+zz)
LD B, LD C, LD D, LD E, LD H, LD L,	A.B.C.D.E.H.L.xx.(HL).(IX+zz).(IY+zz)
LD (xxyy), LD (BC), LD (DE), LD I, LD R,	A
LD (HL), LD (IX+zz), LD (IY+zz),	A.B.C.D.E.H.L.xx

Flags ändern sich keine, ausser bei den Befehlen LD A,I und LD A,R werden S und Z entsprechend gesetzt, AC wird rückgesetzt und P enthält Inhalt vom Interrupt-Flip-Flop 2. N wird rückgesetzt und CY bleibt unverändert.

16-Bit-Ladebefehle:

LD BC, LD DE, LD HL, LD IX, LD IY,	xxyy.(xxyy)

```
LD SP,        - HL.IX.IY.xxyy.(xxyy)

LD (xxyy),    - BC.DE.HL.SP.IX.IY

PUSH          - AF.BC.DE.HL.IX.IY
POP           -
```

Flags werden keine verändert.

Austauschbefehle:

```
EX (SP),      - HL.IX.IY
EX AF, AF'
EX DE, HL
EXX
```

Flags werden keine verändert.

Blockbefehle:

```
LDI
LDD

Flags:  AC : 0
        P  : 0, falls Inhalt von BC=0
        N  : 0
---------------------------------------
LDIR
LDDR

Flags:   AC : 0
         P  : 0
         B  : 0
```

Suchbefehle:

```
CPI
CPD
CPIR
CPDR
```

Flags: S : 1, wenn Resultat neg. ist
Z : 1, wenn Resultat 0 ist
AC : 0, wenn Borgen von Bit 4 nötig war
P : 0, wenn BC = 0 ist
N : 1

8-Bit Arithmetik- und Logikbefehle:

Bei allen Befehlen dieser Gruppe gilt:
Flags: S : 1, wenn Resultat neg. ist
Z : 1, wenn Resultat 0 ist

ADD A, - A B.C.D.E.H.L.xx.(HL).(IX+zz).
ADC A, - (IY+zz)

Flags: AC : 1, wenn Übertrag von Bit 3
P : 1, wenn Überlauf auftrat
N : 0
CY : 1, wenn Übertrag von Bit 7

SUB - A.B.C.D.E.H.L.xx.(HL).(IX+zz).
SBC A, - (IY+ZZ)
CP -

Flags: AC : 0, wenn Bit 4 geborgt
P : 1, wenn Überlauf auftrat
N : 1
CY : 1, wenn Borgen nötig war

AND - A.B.C.D.E.H.L.xx.(HL).(IX+zz).
OR - (IY+zz)
XOR -

Flags: AC: 0
P : 1, wenn Parität gerade
N : 0
CY: 0

INC - A.B.C.D.E.H.L.xx.(HL).(IX+zz).
- (IY+zz)

Flags: AC : 1, wenn Übertrag von Bit 3
P : 1, wenn Überlauf auftrat
N : 0

DEC - A.B.C.D.E.H.L.xx.(HL).(IX+zz).
- (IY+zz)

Flags: AC : 0, wenn Bit 4 geborgt
P : 1, wenn Überlauf auftrat
N : 0

DAA

Flags: AC 1: wenn Übertrag Bit 3 und N = 0
0, wenn Bit 4 geborgt und N = 1
P 1: wenn Parität gerade

Flagbefehle:

SCF

Flags: AC : 0
N : 0
CY : 1

CCF

Flags: AC : Alter Wert vom Übertragflag
N : 0
CY : 1, wenn Übertragflag vorher = 0

CPL

Flags: AC : 1
N : 1

NEG

Flags: S : 1, wenn Resultat neg.
Z : 1, wenn Resultat 0
AC : 0, wenn Bit 4 geborgt
P : 1, wenn Überlauf auftrat
N : 1
CY : 1, wenn geborgt

16-Bit-Arithmetikbefehle:

ADD HL, - BC.DE.HL.SP
ADD IX, - BC.DE.SP.IX
ADD IY - BC.DE.SP.IY

Flags: AC : 1, wenn Übertrag von Bit 11
N : 0
CY : 1, wenn Übertrag von Bit 15

ADC HL, - BC.DE.HL.SP

Flags: S : 1, wenn Resultat neg.
Z : 0, wenn Resultat 0
AC : 1, wenn Übertrag von Bit 11
P : 1, wenn Überlauf auftrat
N : 0
CY : 1, wenn Übertrag von Bit 15

SBC HL, - BC.DE.HL.SP

Flags: S : 1, wenn Resultat neg.
Z : 0, wenn Resultat 0
AC : 0, wenn Bit 12 geborgt
P : 1, wenn Überlauf auftrat
N : 1
CY : 1, wenn geborgt

INC - BC.DE.HL.SP.IX.IY
DEC -

Flags werden keine verändert

Rotations-/Verschiebebefehle:

```
RLC          - A.B.C.D.E.H.L.(HL).(IX+zz).
RL           - (IY+zz)
SLA          -

Flags:  S  : 1, wenn Resultat neg.
        Z  : 1, wenn Resultat 0
        AC : 0
        P  : 1, wenn Parität gerade
        N  : 0
        CY : Bit 7 des ursprünglichen Inhalts
```

```
RRC          - A.B.C.D.E.H.L.(HL).(IX+zz).
RR           - (IY+zz)
SRA          -
SRL          -

Flags:  S  :  1,wenn Resultat neg.
        Z  :  1,wenn Resultat 0
        AC :  0
        P  :  1, wenn Parität gerade
        N  :  0
        CY :  Bit 0 des ursprünglichen Inhalts
```

```
RLCA
RLA

Flags:  AC : 0
        N  : 0
        CY : Bit 7 des ursprünglichen Inhalts
```

```
RRCA
RRA

Flags:  AC : 0
        N  : 0
        CY : Bit 0 des ursprünglichen Inhalts
```

RLD
RRD

Flags: S : 1, wenn A neg. wurde
Z : 1, wenn A 0 wurde
AC : 0
P : 1, wenn Parität gerade
N : 0

Bit-Manipulationsbefehle:

Bit 0, - A.B.C.D.E.H.L.(HL).(IX+zz).
Bit 1, - (IY+zz)
Bit 2, -
Bit 3, -
Bit 4, -
Bit 5, -
Bit 6, -
Bit 7, -

Flags: S : unbekannt
Z : 1, wenn entspr. Bit = 0
AC : 1
P : unbekannt
N : 0

SET 0, - A.B.C.D.E.H.L.(HL).(IX+zz).
SET 1, - (IY+zz)
SET 2, -
SET 3, -
SET 4, -
SET 5, -
SET 6, -
SET 7, -
RES 0, -
RES 1, -

```
RES 2,      -
RES 3,      -
RES 4,      -
RES 5,      -
RES 6,      -
RES 7,      -
```

Flags werden keine verändert.

Sprungbefehle:

```
JP              - xxyy.(HL).(IX).(IY)

JP NZ,          - xxyy
JP Z,           -
JP NC,          -
JP C,           -
JP PO,          -
JP PE,          -
JP P,           -
JP M,           -

JR tt

JR NZ,          - tt
JR Z,           -
JR NC,          -
JR C,           -
DJNZ            -
```

Flags werden keine verändert.

Unterprogrammbefehle:

```
CALL            - xxyy
CALL NZ,        -
```

CALL Z, -
CALL NC -
CALL C, -
CALL PO, -
CALL PE, -
CALL P, -
CALL M, -

RET
RET NZ
RET Z
RET NC
RET C
RET PO
RET PE
RET P
RET M
RETI
RETN

RST - &00.&08.&10.&18.&20.&28.&30.&38

Flags werden keine verändert.

Kontrollbefehle:

NOP
HALT
DI
EI
IM 0
IM 1
IM 2

Flags werden keine verändert.

I/O-Befehle:

```
IN A,(C)
IN B,(C)
IN C,(C)
IN D,(C)
IN E,(C)
IN H,(C)
IN L,(C)
```

```
Flags:  S  : 1, wenn Resultat neg.
        Z  : 1, wenn Resultat 0
        AC : 0
        P  : 1, wenn Parität gerade
        N  : 0
```

```
IN A,(xx)
OUT (xx),A
OUT (C),A
OUT (C),B
OUT (C),D
OUT (C),E
OUT (C),H
OUT (C),L
```

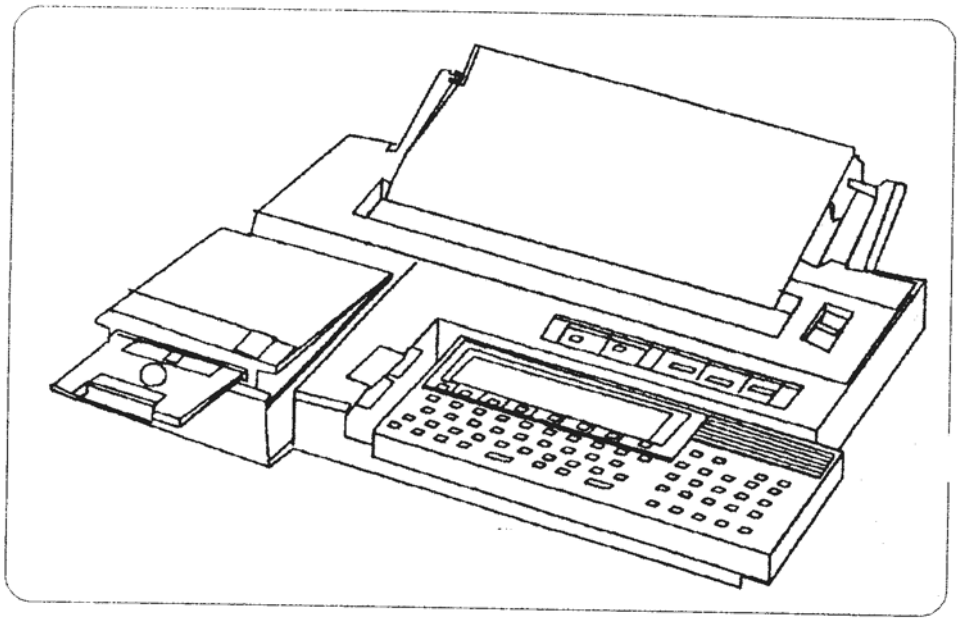
Flags werden keine verändert.

```
INI
IND
OUTI
OUTD
```

```
Flags:  S  : unbekannt
        Z  : 1, wenn B = 0 wurde
        AC : unbekannt
        P  : unbekannt
        N  : 1
```

INIR
INDR
OTIR
OTDR

Flags: S : unbekannt
Z : 1
AC : unbekannt
P : unbekannt
N 1



Auf den nächsten Seiten ist ein BASIC-Programm abgedruckt. Es dient zur Übersetzung von Assemblercode in Maschinensprache. Zu diesem Zwecke wird es nötig, den Assemblercode durch eine Liste zu vergleichen, mit der der ML-Code bestimmt werden kann. Eine solche Liste ist im Anhang abgedruckt. Nun können Sie es sich wahrscheinlich denken: Diese Liste muss abgetippt werden. Zu Ihrer Information: es sind um die 17000 Zeichen, die Sie in mühsamer Kleinarbeit in Ihren PC-1600 eingeben werden. Es sei denn, Sie entschliessen sich die Programmdiskette mit allen nötigen Programmen und Dateien kommen zu lassen. Benützen Sie bitte den Bestellschein am Schluss des Buches.

Zur Eingabe des Programms:
Geben Sie als erstes folgende Zeile im **Direktmodus** (d.h. nicht als Programm) ein:

```
MAXFILES = 1
OPEN "X:OBJ-CODE-DAT" FOR OUTPUT AS #1
CLOSE
```

Durch diese Zeilen wird die Listen-Datei generiert. Als nächstes tippen Sie folgendes Programm ab und speichern es unter "EINGABE.BAS" auf Diskette ab:

```
10:CLS :MAXFILES =1:DIM A$(1)*30
20:PRINT "EINGABE DES OBJECT-CODES:"
30:OPEN "X:OBJ-CODE.DAT" FOR APPEND AS #1
40:PRINT
45:N$="":A$(1)=""
50:INPUT "BEZEICHNUNG:";N$:IF INSTR(N$,",")<>0 THEN
  GOTO 50
60:A$(1)=N$+LEFT$("                    ",18-LEN N$):REM
  18 Leerzeichen
```

```
70:AB$="":INPUT "ANZ.BYTES:";AB$
80:A$(1)=A$(1)+LEFT$("00",2-LEN AB$)+AB$
90:C$="":INPUT "CODE$:";C$
100:A$(1)=A$(1)+C$
110:PRINT A$(1)
120:PRINT #1,A$(1)
130:PRINT "BEI 'BRK' > CLOSE ... "
140:PRINT :GOTO 45
```

Nun starten Sie dieses Programm und geben die Umrechnungsliste, die Sie im Anhang dieses Buches finden, ein.
Beispiel:

```
"ADC A,(HL)        8E"   steht in der Liste.
Ihre Eingabe: "ADC A;(HL)" (Enter)
              "1" (Enter) (da nur 1 Byte)
              "8E"(Enter)
-------------------------------------------------
"BIT 2,(IX+zz)     DD CB zz 56"
Ihre Eingabe: "BIT 2;(IX+zz)" (Enter)
              "4" (Enter) (da aus 4 Byte best.)
              "DDCBzz56"
```

Wichtig: Geben Sie keine (Komma) ein, da diese als Datentrenner interpretiert würden.
Geben Sie eventuell vorkommende "xx","tt", etc. ebenfalls ein und zwar kleingeschrieben ! Benützen Sie stattdessen das Zeichen ";".

Haben Sie bei "ANZ.BYTES:" bereits den Code eingegeben, stürzt das Programm ab. Sie können es mit "GOTO 70" wieder starten. Es frägt Sie nun erneut nach "ANZ.BYTES:". Geben Sie nun den Wert ein und arbeiten Sie normal weiter. Haben Sie genug eingegeben, drücken Sie bei der Frage "BEZEICHNUNG:" die BRK-Taste. Geben Sie nun "CLOSE" ein und schalten Sie den Computer aus. Sie können zu einem späteren Zeitpunkt mit RUN starten und an der aktuellen Stelle in der Liste weiterarbeiten.

Haben Sie die Liste fertig eingetippt (wofür ich Ihnen gratulieren möchte), können Sie sich an das Hauptprogramm wagen.

HAUPTPROGRAM:

```
  5 MAXFILES =3:DIM E$(1)*80,K$(9)
  7 FOR I=0TO 9:K$(I)="0000":NEXT I
 10 GOTO 1000
 20 "D>H"H$=HEX$ (VAL B$)
 22 IF LEN H$=10R LEN H$=3THEN LET H$="0"+H$
 24 RETURN
 30 "B>H"B=0:FOR I=0TO LEN B$-1
 40 IF MID$ (B$,LEN B$-I,1)="1"THEN LET B=B+2^I
 50 NEXT I:B$=STR$ B
 60 GOSUB "D>H":RETURN
 70 "A>H"H$=HEX$ ASC B$
 72 IF LEN H$=10R LEN H$=3THEN LET H$="0"+H$
 75 RETURN
100 "LIES"H$="":IF EOF(1)=1THEN LET EL=99:GOTO
    "ERROR"
110 INPUT #1,E$(0)
115 EF=INSTR(E$(0),":")
116 IF EF=0 THEN 130
120 E$(0)=LEFT$(E$(0),INSTR (E$(0=LEFT$(E$(0),"|")
130 IF RIGHT$(E$(0),1)=" "THEN LET E$(0)=LEFT$
    (E$(0),LEN E$(0)-1):GOTO 130
140 IF E$(0)=""THEN GOTO 100
150 E=INSTR (E$(0),"#'')+INSTR (E$(0),"$")+INSTR
    (E$(0),"%")+INSTR (E$(0),"&")+INSTR (E$(0),"K")
155 IF E=0THEN RETURN
160 E$=MID$ (E$(0),E,1)
170 IF E$="#"THEN LET B$=MID$ (E$(0),E+1,1):GOSUB
    "A>H":GOTO 240
```

```
172 IF E$="K"THEN LET H$=K$(VAL MID$ (E$(0),
    E+1,1)):C=1:GOTO 240
175 IF E$="&"THEN 260
180 B$="":I=1:I$=""
190 I$=MID$ (E$(0),E+I,1)
200 IF I$>="0"AND I$<="9"THEN LET B$=B$+I$:I
    =I+1:GOTO 190
210 IF B$=""THEN EL=98:GOTO "ERROR"
215 C=LEN B$
220 IF E$="%''THEN GOSUB "D>H"
230 IF E$="$"THEN GOSUB "B>H"
240 E$(0)=LEFT$ (E$(0),E-1)+"&"+H$+RIGHT$ (E
    $ (0),LEN E$(0)-C-E)
250 RETURN
260 H$="":I=1:I$=""
270 I$=MID$ (E$(0),E+I,1)
280 IF (I$>="0"AND I$<="9" )OR (I$>="A"AND I$
    <="F")THEN LET H$=H$+I$:I=I+1: GOTO 270
285 C=LEN H$
290 IF H$=""THEN EL=98:GOTO "ERROR"
300 IF LEN H$=10R LEN H$=3THEN LET H$="0"+H$
310 GOTO 240
400 "OBJ":IF EOF(3)=1THEN LET EL=97:GOTO "ERROR"
410 INPUT #3,E$(1)
415 IF LEFT$ (E$(1),3)<>J$ THEN 400
420 CO$=MID$ (E$(1),19,LEN E$(1)-18)
425 E$(1)=LEFT$ (E$(1),18)
430 IF RIGHT$ (E$(1),1)=" "THEN LET E$(1)=LEFT$
    (E$(1),LEN E$(1)-1):GOTO 430
435 IF INSTR (E$(1),",x")+INSTR (E$(1),"y")+INSTR
    (E$(1),"z")+INSTR (E$(1),"t")=0 THEN RETURN
440 E1=INSTR (E$(1)1,"x")
450 IF EL>0 THEN LET E$(1)=LEFT$ (E$(1),E1-1)
    +RIGHT$ (E$(1),LEN E$(1)-E1-1):GOTO 440
460 E1=INSTR (E$(1),"y")
```

```
470  IF E1>0THEN LET E$(1)=LEFT$ (E$(1),E1-1)
     +RIGHT$ (E$(1),LEN E$(1)-E1-1):GOTO 460
480  EL=INSTR (E$(1),"z")
490  IF E1>0THEN LET E$(1)=LEFT$ (E$(1),E1-1)
     +RIGHT$ (E$(1),LEN E$(1)-E1-1):GOTO 480
500  EL=INSTR (E$(1),"t")
510  IF E1>0THEN LET E$(1)=LEFT$ (E$(1),E1-1)
     +RIGHT$ (E$(1),LEN E$(1)-E1-1):GOTO 500
520  RETURN
600  "H>D"M$=LEFT$ (CD$,1):N$=RIGHT$ (CD$,1)
610  N=ASC N$-48:IF N>10THEN LET N=N-7
615  M=ASC M$-48:IF M>10THEN LET M=M-7
620  DE=M*16+N:RETURN
800  "ERROR"CLS :CLOSE
805  IF EL=99THEN 1700
810  BEEP 10,35,50
820  PRINT "**** PROGRAMM-ABBRUCH ****"
830  PRINT "-------------------------"
840  PRINT "ERRORLEVEL:";EL
990  END
991  "* * * * * * * * * * *"
992  "* * * * * * * * * * *"
993  "*   HAUPTPROGRAMM   *"
994  "* * * * * * * * * * *"
995  "* * * * * * * * * * *"
1000 CLS
1010 PRINT "-*-POCKET-ASSEMBLER-*-"
1020 PRINT "----------------------"
1030 INPUT "SRC-FILE: ";FN$
1040 IF INSTR (FN$,".")<>OTHEN GOTO 1030
1045 INPUT "S/Peicher ¦ D/isk:';SD$
1047 IF SD$<>"S" AND SD$<>"D" THEN 1045
1050 OPEN "X:"+FN$+".SRC"FOR INPUT AS #1
1055 OPEN "X:"+FN$+".OBJ"FOR OUTPUT AS #2
1060 GOSUB 100
1070 IF E$(0)="BEGINNE"THEN "START"
1080 BE$=LEFT$ (E$(0),2)
```

```
1090 IF BE$="AA"THEN LET AA$=H$
1091 IF BE$="AB"THEN LET AB$=H$
1100 IF LEFT$ (BE$,1)="k"THEN LET K$(VAL RIGH
     T$ (BE$,1))=H$
1110 GOTO 1060
1500 "START"PRINT #2,AA$:PPINT #2,AB$:CLS :PR
     INT "Startadr. &";AA$;" Banknr.";AB$
1505 CD$=LEFT$ (AA$,2):GOSUB 600:AA=DE*256:CD
     $-RIGHT$ (AA$,2):GOSUB 600:AA=AA+DE:AB=V
     AL AB$
1506 PRINT AA,AB
1510 OPEN "X:OBJ-CODE.DAT"FOR INPUT AS #3
1520 GOSUB 100:J$=LEFT$ (E$(0),3):IF H$=""THE
     N LET F$=E$(0):PRINT F$:GOT0 1540
1530 F$=LEFT$ (E$(0),E-1)+RIGHT$ (E$(0),LEN E
     $(0)-E-LEN H$):PRINT E$(0)
1540 GOSUB 400
1550 IF E$(1)<>F$THEN 1540
1560 CLOSE #3
1570 C=INSTR (CO$,"x")
1580  IF  C>0THEN  LET  CO$=LEFT$  (CO$,C-1)+LEFT$
      (H$,2)+RIGHT$ (CO$,LEN
1590 C=INSTR (CO$,"y")
1600 IF C>0THEN LET CO$=LEFT$ (CO$,C-1)+RIGHT
     $ (H$,2)+RIGHT$ (CO$,LEN CO$-C-1)
1610 C=INSTR (CO$,"z")
1620 IF C>0THEN LET CO$=LEFT$ (CO$,C-1)+RIGHT
     $ (H$,2)+RIGHT$ (CO$,LEN CO$-C-1)
1630 C=INSTR (CO$,"t")
1640 IF C>0THEN LET CO$=LEFT$ (CO$,C-1)+RIGHT
     $ (H$,2)+RIGHT$ (CO$,LEN CO$-C-1)
1650 CO$=RIGHT$ (CO$,1,LEN CO$-2)
1660 FOR I=1TO LEN CO$STEP 2
1670 CD$=MID$ (CO$,I,2):PRINT #2,CD$
1675 IF SD$="S"THEN GOSUB 600:POKE #(AB),(AA)
     ,(DE):AA=AA+1
1680 NEXT I
```

```
1690   GOTO 1510
1700   IF SD$="S"THEN END
1705   OPEN "X:+FN$+".BLD"FOR OUTPUT AS #3
1707   OPEN "X:"+FN$+".OBJ"FOR INPUT AS #2:INPU
       T #2,A$:INPUT #2,A$
1710   PRINT #3, "10AA=&";AA$;":AB=&";AB$
1720   I=0:ZL=20
1730   E$(0)=STR$ (ZL)+ "POKE# AB,AA+"+STR$ I
1740   FOR J=OTO 9
1750   IF EOF (2)<>0THEN LET BY$="00":GOTO 1770
1760   INPUT #2,BY$
1770   E$(0)=E$(0)+ ",&"+BY$
1780   NEXT J
1790   PRINT #3,E$(0)
1800   IF EOF (2)<>0THEN CLOSE :END
1810   I=I+10:ZL=ZL+10:GOTO 1730
```

Speichern Sie es auf Diskette mit dem Befehl SAVE "X:MAIN.BAS".
Zur Bedienung des-Programmes:

Beim Schreiben von Assemblerprogrammen und späteren Assemblieren mit obenstehendem Programm, können folgende Zahlensysteme benützt werden. Es ist wichtig, das Sonderzeichen davor einzugeben, da sonst die Zahl nicht als solche erkannt wird.

```
&FFFF   = Hexadezimal
%49152  = Dezimal
$11011  = Binär
#x      = ASCII (Immer nur 1 Zeichen)
```

Weiter ist es möglich, Konstanten zu gebrauchen. Bei der Konstantendeklaration können ebenfalls alle oben genannten Zahlensysteme benützt werden.

Ihr Assemblerprogramm muss aus 2 Teilen bestehen: Aus einem 1. Deklarationsteil und aus einem 2. Programmteil.

Der Deklarationsteil muss folgende Elemente aufweisen:

AA= Anfangsadresse (0-66535)
AB= Block der Anfangsadresse (0-7)
kx= Konstaritenbelegung (x=0-9)

Wichtig ist, dass man bei der Konstantenbelegung den Buchstaben "k" klein schreibt, hingegen beim Gebrauch der Konstanten im Progranmmteil diesen gross ("K") schreibt.

Am Schluss einer jeden Zeile (egal ob im Deklarations- oder Programmteil) dürfen Kommentare stehen, doch müssen diese mit dem Zeichen "|" Shift + 0) abgetrennt sein.

Der Programmteil. wird durch das Befehlswort "BEGINNE" eingeleitet. Nun folgt Ihr Assemblerprogramm. Wichtig dabei ist, dass Sie in der Assemblersprache das Zeichen "," (Komma) nicht gebrauchen, sondern das Zeichen ";benützen.
Durch obenstehendes Programm werden alle Befehle korrekt übersetzt - ausser die Befehle:
LD (IX+zz);xx und LD (IY+zz);xx. Dies, weil diese Befehle 2 verschiedene Eingaben verlangt (xx und zz). Diese fehlenden Befehle können aber wiefolgt ersetzt werden: LD B; xx - LD (IX+zz);B

Bei dem Befehl RST ist darauf zu achten, dass nachfolgender Term im Hexadezimalen Format geschrieben wird (also z.B.: "RST &18 ").
Verwenden Sie Konstanten im Programm, so ist es wichtig, diese gross zu schreiben (Z.B. LD A;K2)

Nachfolgend ist ein Beispielprogramm abgedruckt. Tippen Sie es nicht ab, es handelt sich im Prinzip nur um eine Ansammlung von Befehlen, die kein Programm darstellen sondern nur als Vorlage für Ihre Programme dienen.

Beispielprogramm:

```
10:'AA=&C400
20:'AB=%0          |Anfangsadresse &C400 - Bank 0
30:'
40:'|Konstantdendefinition
50:'
60:'k1=$10010101
70:'k0=&C3FF
80:'
90:'BEGINNE
100:'LD A;(K0)
110:'CP K1          |Vergleiche mit K1
120:'RET NZ         |Zurück wenn ungleich
130:'LD A;%124
140:'LD BC;&1000
150:'CALL &01B7     |Beep Höhe A Länge BC
160:'RET            |Zurück
```

Speichern Sie Ihr Assemblerprogramm mit dem Befehl SAVE *"X:Name.SRC" ab. Vergessen Sie hier den * nicht. Dieser bewirkt, dass alle REM-Zeilen eines Programmes als ASCII-Datei gespeichert werden.

Laden Sie nun das Übersetzungsprogramm und starten es mit RUN. Sie, werden nun nach dem SRC-File gefragt. Geben Sie hier den Namen ohne Extension (.SRC) ein. Als nächstes werden Sie gefragt, ob Sie direkt in den Speicher (S) oder auf die Diskette (D) assemblieren wollen. Dies wurde notwendig, da man ohne Speichererweiterung nicht mehr über all zuviel RAM verfügt. Mit der Option D erstellt das Programm einen sogenannten BASIC-Loader. Diesen Können Sie nach dem Durchlauf mit LOAD "Name.BLD" laden und mit RUN starten.
Wenn es beim Übersetzen einen Fehler gibt, zeigt Ihnen das das Programm an. Folgende Errorlevels können erscheinen

98 Formatfehler. Dez/Bin-Zahl sind falsch
97 Code unbekannt, Syntax Error. Dies kommt u.a. vor, wenn Sie ein Komma, statt ein ; eingegeben haben.

Während dem Übersetzen zeigt Ihnen das Programm immer die gerade aktuelle Zeile an.
Besitzen Sie eine RAM-Erweiterung von 32KByte, können Sie die Datei "OBJ-CODE.DAT" auch auf RAM-Disk speichern und in Zeile 1510 den Wert "X:" in "S1: " ändern. Die übersetzung gewinnt so an Geschwindigkeit.
Ausdrucken können Sie Ihre Assemblerprogramme mit LLIST*.
Wenn Ihnen die Übersetzung zu langsam ist, können Sie die Datei "OBJ-CODE.DAT " umstrukturieren und die "LD", "CALL", "RET" und "JP" Befehle nach vorne nehmen, da diese mehr vorkommen als z.B. der "BIT"-Befehl.

Mit den Ladebefehlen belegt man Register oder Speicheradressen mit Werten und Inhalten von anderen Registern.

Beispiele:

LD A,&3C	Der Wert &3C wird dem A-Register zugewiesen.
LD A,C	Der Inhalt vom C-Register Wird in das A-Register kopiert.
LD (xxyy),A	Die Adresse xxyy wird mit dem Wert vom A-Register belegt.
L,D BC,xxyy	Der Wert xxyy wird dem BC-Register zugewiesen.
LD BC,(xxyy)	Ins BC-Register kommt der Wert: Inhalt von (xxyy) + Inhalt von (xxyy+1) * 256

Die Transportbefetile PUSH und POP dienen zum Ablegen von Registerinhalten auf den Stack. Der Stack ist wie ein Speicher, bei dem man oben auflegen und nur oben wieder wegnehmen kann.
Zur Verdeutlichung ein Beispiel:

PUSH AF	AF-Register wird auf Stack gelegt
PUSH BC	BC-Register wird auf Stack gelegt
POP AF	AF erhält oberster Stack-Wert (also alter BC-Wert)
POP BC	BC erhält oberster Stack-Wert (also alter AF-Wert)

Der Stack-Pointer das SP-Register, zeigt immer auf die Adresse des obersten Elementes des Stapels. Eigentlich müsste der Stapel. definiert werden, da aber der PC-1600 ein eigenes ROM besitzt, können wir davon absehen.
PUSH und POP werden also vorallem für das Sichern und Rekonstruieren von Registern benötigt.

Austauschbefehle:

Man unterscheidet 3 Arten von Austauschbefehlen:

1) Austauschen von Stackpointerinhalt und 16-Bit-Register
2) Austauschen von DE und HL
3) Austauschen von primären und sekundären Registern

Zu der ersten Gruppe gehören die Befehle:

EX (SP),HL
EX (SP),IX
EX (SP),IY

Mit diesen Befehlen wird das oberste Stackelement mit dem Inhalt des HL-, IX- oder IY Register ausgetauscht.

Zur zweiten Gruppe gehört nur der Befehl:

EX DE,HL

Dieser tauscht DE und HL um. In HL ist also der Wert von DE und in DE der Wert von HL.

Zur letzten Gruppe der Austauschbefehle gehören folgende Befehle:

```
EX AF,AF'
EXX
```

Der Befehl EXX vertauscht folgende Register miteinander: BC mit BC' DE mit DE' HL mit HL'

Blockbefehle:

Die Befehle LDD LDDR, LDI und LDIR stehen für blockweises verschieben. Bei den Befehlen LDDR (Load, decrement and repeat) und LDIR (Load, increment and repeat) wird wiefolgt vorgegangen:

```
 LDDR                        LDIR
 ----                        ----
wiederhole                  wiederhole
  LD (DE),(HL)                LD (DE),(HL)
  DE=DE-1                     DE=DE+1
  HL=HL-1                     HL=HL+1
  BC=BC-1                     BC=BC-1
bis                         bis
  BC=0                        BC=0
```

In HL muss die Anfangsadresse des zu kopierenden Bereiches stehen, in DE die Anfangsadresse des freien Bereiches und in BC die Anzahl Bytes, die kopiert werden sollen.
Die Befehle LDD und LDI machen genau das gleiche, bis auf dass sie nicht repetieren, d.h. man kann die Repetitionsschleife beliebig vergrössern. Ist nämlich BC=0 geworden, wird das Übertragflag 0 gesetzt. Somit können wir eine einfache Zählschleife mit Abbruch erste11en.

Such-/Vergleichsbefehle:

Der Befehl CP dient zum Vergleich eines, Registers oder eines Wertes mit dem A-Register. Stimmen die beiden Register überein, so wird das Null-Flag=1 (Z=1) gesetzt.

Bsp:

```
CP (IX+&22)          Der  Inhalt der Adresse, auf
                     Die das Register IX+&22
                     zeigt, wird mit A verglichen
JP Z,&C3F2           Wenn übereinstimmung, weiter
                     nach &C3F2
NOP                  Wenn nicht, hier weiter
```

Die Befehle CPD, CPDR, CPI und CPIR gehören züi den Suchbefehlen. Mit Ihnen kann man ganze Blöcke nach einem bestimmten Wert durchsuchen. Bei den Befehlen CPDR (compare, decrement and repeat) und CPIR (compare, increment and repeat) wird wiefolgt vorgegangen:

```
CPDR                        CPIR
----                        ----
wiederhole                  wiederhole
   CP (HL)                    CP (HL)
   HL=HL-1                    HL=HL+1
   BC=BC-1                    BC=BC-1
bis BC=0 oder Z=1           bis BC=0 oder Z=1
```

In HL muss die Anfangsadresse des zu durchsuchendes Bereiches stehen, in BC die Anzahl durchsuchende Bytes. Ist eine Adresse mit dem gleichen Wert wie im A-Register gefunden, bricht die Schlaufe ab und im HL-Register steht die Adresse. Die Schlaufe bricht

ebenfalls ab, sofern BC=0 wurde, dann wurde der Wert nie gefunden.
Die Befehle CPD und CPI machen genau das gleiche, bis auf dass sie nicht repetieren, d.h. man kann die Repetitionsschleife beliebig vergrössern. Ist nämlich der Wert gefunden, wird Z=1 (das Nullflag) gesetzt oder wenn BC=0 wurde, wird das Übertragflag=0 gesetzt.

Arithmetikbefehle:

Die Befehle ADD und ADC bewirken ein Addieren des Terms hinter dem Komma zum Term vor dem Komma. Der Unterschied zwischen beiden Befehlen besteht darin, dass beim Befehl ADC das CY-Flag mitaddiert wird. Beispiele:

```
ADC HL,BC          HL=HL+BC+CY
ADD A,(HL)         A=A+PEEK(HL)
ADD IX,DE          IX=IX+DE
```

Die Befehle SUB und SBC funktionieren genau gleich, nur dass sie zum Subtrahieren von Termen geschaffen wurden. SUB subtrahiert ohne CY-Flag, SBC subtrahiert das CY-Flag mit. SUB gilt nur für das A-Register. Beispiele:

```
SBC A,xx           A=A-xx-CY
SUB B              A=A-B
SBC HL,DE          HL=HL-DE
```

Die Befehle INC und DEC sind zum erhöhen (INCrement) und erniedrigen (DECrement) um

Jeweils den Wert 1 geschaffen worden.
Beispiele:

```
INC (HL)          PEEK(HL)=PEEK(HL)+1
DEC BC            BC=BC-1
INC IX            IX=IX+1
```

Der Befehl DAA Wird benötigt, um den Inhalt des A-Registers nach einem arithmetischen Befehl so zu modifizieren, dass er das Ergebnis dezimaler Artihmetik darstellt.

Der Befehl NEG negiert der Inhalt des A-Registers. Dies entpricht der Subtraktion des Inhaltes vom A-Register an 0 (A=0-A)

Logikbefehle:

Zu dieser Gruppe gehören die Befehle AND, OR, XOR und CPL. Die Befehle AND, OR und XOR verknüpfen das A-Register mit einem Operanden wiefolgt:

A-Register	Operand	AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	0

Daraus folgt, das bei der AND-Verknüpfurig alle Bits, die im Operanden 0 gesetzt sind, im A-Register auch 0 gesetzt werden. Beispiel:

```
A-Register:  01001101    (dez. 77)
Operand      11010011    (dez. 211)
---------------------
AND          01000001    (dez. 65)
```

Dieser Wert ist nach der AND-Operation im A-Register vorzufinden.

Bei der OR-Verknüpfung werden alle Bits, die im Operanden 1 gesetzt sind, auch im A-Register 1 gesetzt. Beispiel:

```
A-Register:  01001101    (dez. 77)
Operand      11010011    (dez. 211)
---------------------
OR           11011111    (dez. 223)
```

Bei der XOR-Verknüpfung werden alle Bits, die im Operanden 1 gesetzt sind, im A-Register invertiert, d.h. sind diese Bits im A-Register gesetzt, werden sie gelöscht. Sind sie =0, werden sie gesetzt. Beispiel:

```
A-Register:  01001101    (dez. 77)
Operand      11010011    (dez. 211)
---------------------
XOR          10011110    (dez. 158)
```

Das Ergebnis der Operation steht immer im A-Register. Dieses wird also zerstört und muss u.U. zuerst mit PUSH AF gerettet werden.
Der Befehl CPL bewirkt eine Invertierung des ganzen A-Register (entspricht also XOR &FF).

Flag-Befehle:

Mit den Befehlen SCF (Set Carry-Flag) und CCF (complement Carry-Flag) können Sie den Zustand des Carry-Flags beeinflussen.

Mit SCF belegen Sie die Flags wiefolgt:

S: unverändert
Z: unverändert
AC: 0
P: unverändert
N: 0
CY: 1

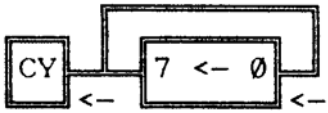
Mit CCF belegen Sie die Flags wiefolgt:

S: unverändert
Z: unverändert
AC: Alter Wert des CY-Flags
P: unverändert
N: 0
CY: Invertierung vom alten Wert
d.h. CY=1, wenn CY(alt)=0
CY=0, wenn CY(alt)=1

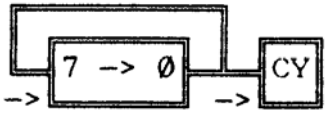
Rotationsbefehle:

Die Befehle RLC, RL, SLA, RRC, RR, SRA und SRL stehen für BIT-weises Verschieben innerhalb eines BYTE. Für BYTE kann ein Register oder eine Adresse, die durch ein 16Bit-Register definiert ist, angegeben werden. Die Befehle RLCA, RLA, RRCA und RRA sind die gleichen Befehle wie RLC, RL, RRC und RR mit dem Unterschied, dass sie auf das A-Register angewendet werden. Die Flags und die Ausführungszeiten sind jedoch anders als wenn man sie mit den Befehlen RLC A, RL A, etc ersetzt (Siehe Befehlsgruppen)

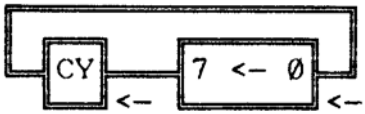
RLC (rotate left circular) rotiert die Bits innerhalb des anzugebenden Registers nach links. Das Bit0 bekommt den Wert des Bit7. Dieses wird aber auch in das CY-Flag kopiert. Schema:



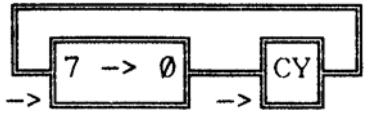
Dasselbe nur in die andere Richtung bewirkt der Befehl RRC (rotate right circular).
Schema:



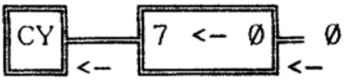
Der Befehl RL (rotate left) verschiebt das ganze Byte nach links. Bit7 gelangt ins CY-Flag und Bit0 erhält den Wert des CY-Flag.
Schema:



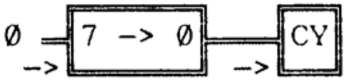
Dieselbe Wirkung zeigt der Befehl RR (rotate right); er rotiert aber nach rechts.
Schema:



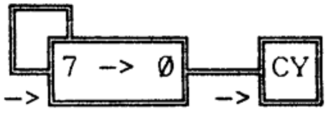
SLA (Shift left arithmetically) funktioniert ähnlich wie RL, nur wird das BIT0 nicht mit dem Inhalt des CY-Flags belegt, sondern wird 0 gesetzt.
Schema:



Das Pendant für dieselbe Rotation aber rechtsherum beisst SRL (Shift right logically).
Schema:

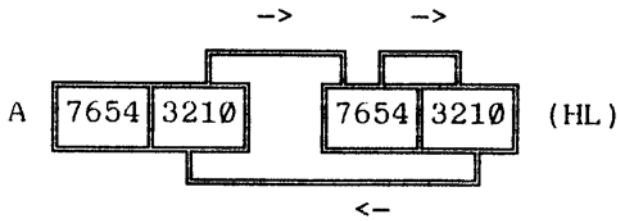


Mit dem Befehl SRA (Shift right arithmetically) verschiebt man ebenfalls bitweise nach rechts, das Bit7 bleibt aber bestehen.
Schema:

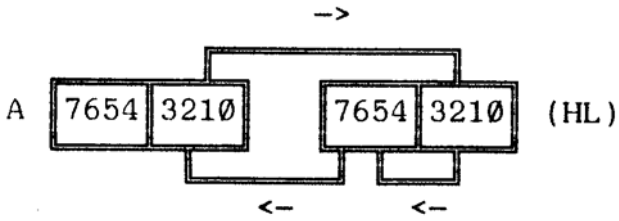


Die Befehle RRD und RLD verschieben nibbleweise. Ein Nibble ist im Prinzip ein halbes Byte, also 4 Bit, und wird zur Darstellung von Dezimalzahlen benötigt. Mit 4 Bit kann man ja Zahlen von 0-15 darstellen, also auch Zahlen von 0-9. 4 Bit werden demnach für eine Dezimalstelle gebraucht. Eine solche Stelle nennt man Nibble oder Digit. Der Befehl RRD (Rotate right digit) rotiert nun im

ergänzten Register A-HL die 3 Stellen nach rechts. Das ergänzte Register A-HL sieht so aus, dass die ersten 4 Bit des A-Register und die 8 Bit der Speicherzelle, auf die das HL-Register zeigt, zusammen ein Register darstellen. Das Schema des RRD-Befehles sieht demnach so aus:



Der RLD-Befehl ist, wie bereits erwähnt, für dieselbe Rotation linkswärts verantwortlich.
Schema:



Bit-Befehle:

Mit den Befehlen BIT, SET und RES können einzelne Bits im Argument, welches dem Befehl folgt, kontrolliert resp. manipuliert werden.

Der Befehl Bit dient zur Kontrolle, ob einzelne Bits gesetzt sind oder nicht. Ist das betreffende nicht gesetzt, also = 0, so wird das Z-Register 1 gesetzt. Dies erlaubt folgende Programmierung:

```
BIT 5,A          Testet ob Bit5 des A-Register =1
JP Z,&1FFF       Wenn nicht, springe zu &1FFF
weiter           wenn ja, weiter im Programm
```

Um gezielt Bits zu setzen, verwenden wir den Befehl SET. Beispiel:

```
SET 7,D          macht das Vorzeichen negativ
                 (1-Byte Zahlen gehen von
                  -127 bis 128)
```

Um gezielt Bits zu löschen (0 zu setzen), benützen wir den RES-Befehl. Beispiel:

```
RES 5,A          wandelt den ASCII-codierten
                 Buchstaben in einen Grossbuch-
                 staben um
```

Sprungbefehle:

Sprungbefehle bewirken eine Verzweigung im Programm. Sie können von einer Bedingung abhängen. Man unterscheidet zwischen relativen (JR-Befehl) und absoluten (JP-Befehl) Sprungbefehlen. Dem JP-Befehl folgt direkt eine Adresse, an die das Programm ggf. springen soll. Beim relativen Sprung werden soviele Bytes "übersprungen" wie der Term nach dem Befehl ausdrückt.

Beispiel. eines absoluten Sprunges:

```
JP Z,&1FFF        Springt nach &1FFF, wenn Z=1
```

Beispiel eines relativen Sprunges:

```
JR &23            Springt 35 Bytes nach vorne
JR &E7            Springt 25 Bytes zurück
```

Ein relativer Sprung kann von -127 bis +128 Bytes erfolgen. Für die negative Sprungrichtung nimmt man die Anzahl zu überspringender Bytes und subtrahiert diesen Wert von 256 um den Term zu erhalten, der nach dem JR Befehl hingehört.

Relative und absolute Sprünge kann man von einer Bedingung abhängig machen. So wird z.B. beim Befehl JP M,&1FFF nur gesprungen, wenn das Negativ-Flag gesetzt ist.

Folgende Bedingungen sind nützbar:

Befehl	Sprung, wenn
JP C,xxyy	CY=1 (Übertrag)
JP M,xxyy	S=1 (negativ)
JP NC,xxyy	CY=0 (kein Übertrag)
JP NZ,xxyy	Z=0 (ungleich 0)
JP P,xxyy	S=0 (positiv, also >=0)
JP PE,xxyy	P=1 (Überlauf)
JP PO,xxyy	P=0 (kein Überlauf)
JP Z,xxyy	Z=1 (gleich 0)
JP xxyy	springt immer

Auch relative Spränge kann man von einer Bedingung abhängig machen:

Befehl	Sprung, wenn
JR NZ,tt	Z=0 (ungleich 0)
JR z,tt	Z=1 (gleich 0)
JR NC,tt	CY=0 (kein Ubertrag)
JR C,tt	CY=1 (Übertrag)
JR tt	springt immer

Absolute Sprünge ohne Bedingung können auch an die Adresse erfolgen, die im HL-, IX- oder IY-Register steht. Z.B. JP (HL)

Ein weiterer Sprungbefehl ist der **DJNZ**-Befehl.
Mit ihm kann man konfortabel eine Schlaufe programmieren. Der Befehl setzt sich aus folgenden Befehlen zusammen:

```
DEC B
JR NZ,tt
```

Er decrementiert das B-Register und springt solange zu seinem relativen Sprungziel, bis B=0 wird. Um auch hier einen Sprung in die negative Richtung zu bewirken, muss die Anzahl der zu überspringenden Bytes zuerst am Wert 256 subtrahiert werden, da Sprünge von max. 128 Bytes in jede Richtung zugelassen sind.

Unterprogrammbefehle:

Ähnlich wie im Basic mit den Befehlen GOSUB und RETURN lassen sich auch in Maschinensprache. Unterprogramme anfertigen. Im Z80 Assemblerdialekt heissen diese Befehle **CALL**, für den Aufruf eines Unterprogrammes, und **RET**, für den Rücksprung ins Hauptprogramm.

Beim **CALL**-Befehl wird die aktuelle Adresse im Programm auf den Stack gelegt und an die angegebene Adresse gesprungen. Diesen Sprung kann man (wie den JP-Befehl) von einer Bedingung abhängig machen.
Der **RET**-Befehl bewirkt ein Laden von der Adresse (die durch den CALL-Befehl auf den Stack gelegt wurde) in den Programmzeiger. Das Programm wird also an der alten Stelle weiterabgearbeitet. Auch diesen Rücksprung kann man von einer Bedinung abhängig machen.

Folgende Bedingungen sind bestimmbar:

Befehle		(Rück-)Sprung, wenn
CALL xxyy	/ RET	erfolgt ohne Bed.
CALL NZ,xxyy	/ RET NZ	Z=0 (ungleich 0)
CALL Z, xxyy	/ RET Z	Z=1 (gleich 0)
CALL NC,xxyy	/ RET NC	CY=0 (kein Übertrag)
CALL C, xxyy	/ RET C	CY=1 (Übertrag)
CALL PO,xxyy	/ RET PO	P=0 (kein Überlauf)
CALL PE,xxyy	/ RET PE	P=1 (Überlauf)
CALL P, xxyy	/ RET P	S=0 (positiv)
CALL M, xxyy	/ RET M	S=1 (negativ)

Um einen Rücksprung vom Maschinenprogramm in den Basic-Interpreter zu erzwingen (Am Ende jedes ML-Prg.), muss der Befehl RET (ohne Bedingung) erfolgen.
Benötigt man im Unterprogramm den Befehl POP, so wird als erster Wert die Rücksprungadresse geliefert. Da dies wahrscheinlich nicht im Sinne-des Programmierers ist, muss folglich zuerst ein POP IX erfolgen und statt dem RET ein JP (IX) um den normalen Rücksprung sicherzustellen.

Zu der Gruppe der Unterprogramme gehören auch noch die Befehle **RETI**, **RETN** und **RST.**
Der RST-Befehl dient zum aufrufen von bestimmten Unterprogrammen. Und zwar besitzen diese Unterprogramme die Adressen &0000, &0008, &0010, &0018, &0020, &0028, &0030 und &0038. Der Befehl bewirkt also nichts anderes als ein CALL-Befehl. Der Unterschied besteht darin, das der RST-Befehl schneller arbeitet und nur ein Byte als Objectcode besitzt. Da beim PC-1600 der Bereich von &0000 bis &7FFF als ROM deklariert ist, bewirken diese Befehle also einen Aufruf eines Systemunterprogrammes.

Die Befehle **RETI** und **RETN** dienen zur Rückkehr von einem Interruptprogramm. Ein Interruptprogramm fängt dann an zu laufen, wenn z.B. ein externes Gerät den Impuls dazu gibt. Mit diesen Befehlen springt man dann vom Interruptprogramm wieder ins Hauptprogramm zurück. Der Unterschied zwischen beiden Befehlen besteht, darin, dass RETI das Ende einer maskierten Unterbrechungsroutine signalisiert und RETN einer unmaskierten.

Kontrollbefehle:

Zu der Gruppe der Kontrollbefehle zählt man die Befehle, die den Prozessor kontrollieren oder beeinflussen. Als ersten solchen Befehl haben wir da den **NOP**-Befehl. Dieser führt rein gar nichts aus. NOP ist die Abkürzung für "No operation" (Keine Ausführung) . Ihn kann dafür benützen, um Speicherplatz vorzumerken (wenn man in ein bestehendes Programm noch etwas einfügen will) oder um überflüssiges in einem Programm zu löschen. Dieser Befehl braucht wie jeder andere auch Speicherplatz und bei der Abarbeitung eine gewisse Zeit.

Der Befehl **HALT** hält den Prozessor solange an, bis ein Rücksetzen oder eine Unterbrechung (z.B. eine Interruptanforderung eines externen Gerätes) erfolgt.
Die Befehle **DI** und **EI** dienen zum Sperren (DI-Befehl) resp. zum zulassen (EI-Befehl) von maskierbaren Interrupts. Das Sperren geschieht durch Löschen der Interruptflipflops IFF1 und IFF2. Das Zulassen durch ein Setzen von IFF1 und IFF2.

Als letzte der Kontrollbefehle stehen uns die Befehle **IM 0**, **IM 1** und **IM 2** zur Verfügung. Mit ihnen setzt man den Interruptmodus.
Der Interruptmodus 0 bewirkt, dass der Prozessor bei einer maskierbaren Unterbrechung durch Aktivieren der INT-Leitung des Z80, ein Byte liest und dieses als Objekt-Code eines Maschinenbefehls interpretiert. Dieses Byte wird natürlich ein RST-Befehl (siehe dort) darstellen. Somit können acht verschiedene Interruptquellen unterschieden werden.
Bei setzten des IM 1 erfolgt bei jedem Interrupt ein Sprung nach &0038. Es wird das Gerät also nicht bereits vom Prozessor identifiziert.
Bei IM 2 müssen die Anfangsadressen der Unterprechnungsroutinen in einer Tabelle gespeichert sein. Das externe Gerät, welches einen Impuls gab, muss den korrekten Index für diese Tabelle liefern. Dieser Index ist ein 7-Bit Wert (Bit7-Bitl). Bit0 wird nullgesetzt und so der ganze Index zu einer Relativadresse gemacht.
Die zuvorbeschriebene Tabelle muss am Anfang einer Seite (als Seite wird ein Block von 256 Bytes benannt) stehen. Im A-Register muss diese Seitenzahl stehen und durch den Befehl LD I,A in das Unterprechurigsvektorregister I gebracht werden.

Beispiel für IM 2:

Tabelle ab &1400 (Also ab "Seite" 14)

&33,&FF,&40,&22,&52,&4D,
1.Gerät,2.Gerät,3.Gerät,etc..

Den Interruptvekor setzt man mit folgendem Programm

DI	maskierbare Interrupts sperren (während setzten von I)
IM 2	Interruptmodus 2 setzen
LD A,&14	Seitennummer laden (gemäss obigem Beispiel)
LD I,A	Unterbrechungsvektor laden
EI	Interrupts wieder zulassen, da I nun gesetzt.

Wenn nun ein externes Gerät einen Interrupt erzeugt und am Datenbus z.B. der Wert &02 (oder &03; ist egal, da Bit0 ja gelöscht wird) anliegt, so wird dementsprechend nach &33FF gesprungen und die dort stehende Interruptroutine für das entsprechende Gerät abgearbeitet. Dies muss mit dem Befehl RETI abgeschlossen- sein, damit das Hauptprogramm wieder aufgenommen wird.

I/O-Befehle:

Die Kommunikation mit externen Geräten (wie Drucker, Floppy-Disk, RS-232, etc.) kann über bestimmte Speicherzellen und Routinen erfolgen aber auch über sogenannte Ports. Meistens gibt es ein Statusport, welches den Zustand des ext. Gerätes wiedergibt (beim Drucker z.B. Kein

Papier vorhanden, Drucker nicht angeschlossen, Drucker nicht betriebsbereit, etc..). Über die Datenports kann man Daten an das betreffende Gerät schicken (Zeichen, die es ausdrucken soll, Codes für die Änderung des Zeichensatzes, etc...).
Der Z80-Prozessor verfügt über I/O-Befehle, mit denen man 256 verschiedene Ports adressieren kann. Die Bedeutung der verschiedenen Ports fürden PC-1600 wird im Systemhandbuch von Winfried Baum (ISBN 3-924327-31-9, Fischel GmbH, Berlin) ausführlich erklärt. Ich empfehle Ihnen den Kauf dieses Buches, wenn Sie es nicht bereits schon besitzen.
In der Maschinensprache gibt es folgende Befehle für die Handhabung der Ports:

Mit dem Befehl **IN A,(xx)** holt man den Inhalt des Ports xx in das A-Register. Beim PC-1600 hat z.B. der Port &78 die Aufgabe, den Status des Diskettenlaufwerkes anzuzeigen.
Dabei bedeutet:

Bit 3: 0: Laufwerk ist leer
1: Laufwerk enthält Diskette
Bit 6: 0: Laufwerk steht oder Schutz ist aktiv
1: Laufwerk läuft und Schutz ist inaktiv
Bit 7: 0: Laufwerk läuft
1: Laufwerk steht

Nach dem Befehl IN A,(&78) kann man mit BIT 3,A testen, ob eine Diskette im Laufwerk liegt. Ist dies nicht der Fall, so wird das Z-Flag gesetzt und man kann mit JP Z,&adr, in eine Routine springen.

Ports können natürlich auch indirekt adressiert werden. Dies geschieht durch das C-Register. Man lädt die Portadresse mit LD C,&78 in das CRegister und lädt dann mit **IN E,(C)** den Inhalt der oben angegebenen Portadresse (in unserem Bsp. die Adresse &78) in das E-Register. Statt dem E-Register kann man irgendein 8-BitRegister nehmen.
Mit den **OUT**-Befehlen schreibt man den Inhalt eines 8-Bit-Registers in eine Portadresse. Auch hier kann man direkt (mit dem A-Register) oder indirekt (mit dem C-Register) programmieren.
Der Befehl **OUT (xx),A** schreibt den Inhalt des A-Registers auf das Port xx. Anhand eines Beispieles wird auch dieser Befehl klarer:
Die Portadresse &7A ist beim PC-1600 ebenfalls für das Diskettenlaufwerk zuständig. Mit dieser Adresse kann man bestimmen, ob der Motor des Laufwerkes ein- oder ausgeschalten sein soll.
Dabei gilt:

Bit 7: 0: Motor ausschalten
1: Motor einschalten

Mit folgendem Mini-Programm schalten Sie den Motor ein:

```
LD A,&00            A-Register löschen
SET 7,A             Bit 7 von A =1 setzen
OUT (&7A),A         A auf Port &7A ausgeben
RET                 Ende Unterprogramm
```

Lassen Sie den SET-Befehl weg, so wird der Motor angehalten (da ja im A-Register der Wert 0 steht und somit Bit 7 auch nicht gesetzt ist.)
Indirekt adressiert sähe dieses Programm wiefolgt aus:

```
LD C,&7A
LD D,&00            Statt D kann irgendein
SET 7,D             8-Bit-Register verwendet
OUT (C),D           werden.
RET
```

Den Befehl **INI** kann am besten wiefolgt umschrieben werden:
Der Inhalt der Portadresse, die in C steht, wird in die Speicherstelle, die das HL-Register bezeichnet, kopiert. B wird um 1 subtrahiert, HL um 1 erhöht. In einem Mini-Programm ausgedrückt, bewirkt dieser Befehl also:

```
IN (HL),(C)         (Diesen Befehl gibt es in
                    Wirklichkeit nicht)
INC HL
DEC B
```

Mit diesem Befehl kann man also ganze Speicherbereiche mit dem Inhalt der durch C bestimmten Portadresse belegen. Bei INI (In and increment) wird HL immer um eins erhöht. Diesselbe Wirkung, nur dass HL immer um eins erniedrigt wird, besitzt der Befehl **IND.** Mit IND und INI kann man ohne Probleme Schleifen programmieren. Wenn B=0 wird, wird Z=1.
Bereits vorprogrammierte Schleifen beschreiben die Befehle **INIR** und **INDR.** Sie bewirken dasselbe wie die Befehle INI und IND, nur dass sie solange die Portadresse einlesen, bis B=0 wird.

Die Befehle **OUTI** und **OUTD** bewirken das Gegenteil von INI und IND. Sie geben den Inhalt der Speicherzelle, beschrieben durch HL an die Portadresse, beschrieben durch C, aus. OUTI erhöht HL um 1 und OUTD erniedrigt HL um 1. Beide erniedrigen B um 1, damit B einmal 0 wird und man so wieder eine Schlaufe programmieren kann .
Bereits programmierte Schlaufen bezeichnen die Befehle **OTIR** und **OTDR.** Mit Ihnen kann man ganze Speicherblöcke auf einen Port ausgeben. Wollen wir z.B. 200 Bytes von der Adresse &1D33 an aufsteigend auf den Port &7F (ohne Wirkung beim PC-1600) ausgeben, so müssen wir wiefolgt programmieren:

```
LD C,&7F          Portadresse laden
LD HL,&1D33       Anfangsadresse laden
LD B,&C8          Anzahl Bytes (dez. 200)
OTIR              Diesen Block auf Port aus
                  ,geben
```

Mit OTIR geben Sie also den Block &1D33 - &1DFB auf den Port &7F aus. Möchten Sie den Block von &1D33 - &1C6B (also 200 Bytes rückwärts) ausgeben, so schreiben Sie statt OTIR den Befehl OTDR (Out, decrement and repeat) .

Im 2. Teil dieses Buches werden wir PC-1600 spezifisch. Als erstes ist unten die Speicherübersicht abgedruckt. Dies ist wichtig, damit Sie wissen, wo das RAM und wo welches ROM des PC-1600 liegt.

Speicherübersicht PC-1600:

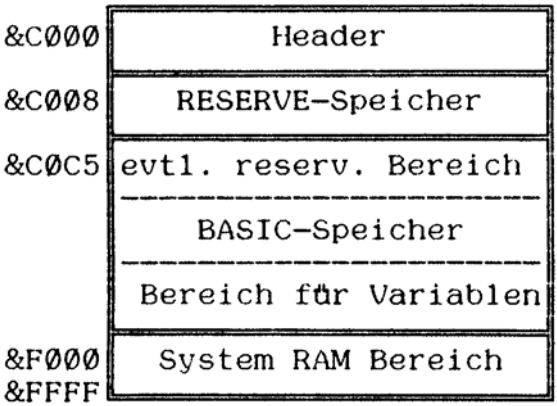
Banknr.	Ø	1	2	3	4	5	6	7
&0000	syst ROM	unbe legt						
&4000	syst ROM	RAM S1t2	unbe legt	syst ROM	P1tr ROM	Disk ROM	unbe legt	unbe legt
&8000	RAM S1t1	RAM S1t1	RAM S1t2	RAM S1t2	unbe legt	unbe legt	syst ROM	unbe legt
&C000	RAM 1600	unbe legt						

Die leeren Felder sind Speicherbereiche die nicht verfügbar sind, d.h. weder RAM noch ROM darstellen.

Der RAM-Speicher beginnt beim PC-1600 ohne Speichererweiterung bei der Adresse &C000 in Bank #0. Bei einer Speichererweiterung von mindestens 16Kb in Slot 1 beginnt der RAM-Speicher bei &8000 in Bank #0.

In der Programmiersprache BASIC sind folgende Befehle für die Maschinensprache zuständig:

Mit NEW "S0:",xx reserviert man sich einen Bereich am Anfang des vorhandenen RAM-Speichers für Maschinenspracheprogramme. Die Anfangsadresse berechnet sich aus: RAM-Anfang + &C5. Der Wert &C5 muss dazuaddiert werden, da diese 197 Bytes für den Reservespeicher (Funktionstasten) benötigt werden.



Um nun die effektive Adresse herauszufinden, müssen Sie RAM-Anfang wissen und diesen um &C5 dazuaddieren. Dazu kommt noch, dass Sie an dieser Stelle nur ML-Programme mit relativen Sprungadressen schreiben können, da sonst die Programme bei einer anderen Speicherkonfiguration zuerst umgeschrieben werden müssen.
Um dies all-es zu umgehen, kann man sich auch eines kleinen Tricks bedienen. Wird der BAR-Code-Lesestift nicht benötigt, kann man sich

einen Bereich wie dies die Software zu diesem Stift auch tut, am Ende des BASIC-RAMs reservieren. Dies hat den Vorteil, dass die Anfangsadresse ohne langes Umrechnen auslesbar ist und- diese, egal welche Speicherkonfiguration momentan eingestellt ist, immer diesselbe ist, da der reservierte Bereich im RAM vom Grundspeicher liegt. Einen solchen Bereich reserviert man wiefolgt:

```
A= (Anzahl zu reservierende Bytes)
CALL &02DD,A
```

Die Anfangsadresse (das erste Byte des reservierten Bereiches) berechnet sich wiefolgt:

```
PEEK &F035*256 + PEEK &F034
```

Um diesen Speicher wieder aufzuheben (Er ist vom Überschreiben durch Variablen und BasicProgranun geschützt), gibt man folgende Zeilen ein:

```
A=0
CALL &02DD,A
```

Den **PEEK**-Befehl dürfte wohl jedem bekannt sein. Mit ihm liest man bestimmte Speicherzellen in einer bestimmten Bank aus. Dies geschieht wiefolgt:

```
PEEK #(banknr,adresse)
```

Liegt die Adresse zwischen &C000 und &FFFF der Bank 0, kann man auch "PEEK adresse " benützen.

Der **POKE**-Befehl funktioniert genau umgekehrt wie der PEEK-Befehl. Mit ihm schreibt man in bestimmte Speicherzellen in einer bestimmten Bank.

POKE # banknr,adresse,Wert1,Wert2,Wert3, etc..

Die Werte 1-x werden wiefolgt in den Speicher geschrieben:

Banknr.	Adresse
banknr	adresse Wertl
banknr	adresse+1 Wert2
banknr	adresse+2 Wert3 etc.

Will man in die Banknummer #0 schreiben, kann man den Befehl auch so verwenden:

POKE adresse,Wert1,Wert2,Wert3, etc ...

Mit dem **CALL**-Befehl rufen Sie eine ML-Routine auf. Liegt diese in der Bank #0, können Sie dies mit "CALL, adresse" tun; liegt sie ausserhalb Bank #0, so müssen Sie wiefolgt auch die Banknummer bestimmen:

CALL # banknr,adresse

Dem Term "adresse" kann noch eine Variable folgen. Bei einer numerischen Variablen wird der Inhalt (sofern er im Bereich -32768 und 32767 liegt) dem DE-Register überreicht. Ist das ML-Programm beendet und das CF=1 gesetzt, so wird das DE-Register in diese numerische Variable kopiert.

Handelt es sich um eine String-Variable (z.B. A$), so wird dem DE-Register die Adresse übermittelt, wo diese Variable beginnt. Bei Beendigung der ML-Routine, sofern CY=1 ist, wird dieser Variablen die Zeichenkette zugewiesen, die sich aus Adresse im DE-Register und Länge im B-Register ergibt.

Die Befehle **INP** und **OUT** dienen, wie die Befehle IN und OUT in der Maschinensprache, zur Verwaltung der 256 Ports des PC-1600. Diese werden im Systembandbuch für den PC-1600 der Fischel GmbH ausführlich erklärt.

Ein im Speicher befindliches ML-Programm kann mit dem BSAVE-Befehl auf Diskette oder mit dem CSAVEM-Befehl auf Kassette gespeichert werden.
Dies geschieht wiefolgt (am Beispiel von CSAVEM):

CSAVE M "Name";#banknr,&anfang,&ende (,&auto)

Die Terme banknr,anfang und ende beschreiben Standort und Länge des Maschinenprogrammes. Mit (dem Term auto kann eine Adresse bestimmt werden, an der das Programm startet, wenn es vollständig eingeladen wurde. Wird er weggelassen, so muss mit CALL #banknr,Startadr gestartet werden.

Der **BLOAD-** resp. **CLOADM-**Befehl funktioniert ähnlich (Am Beispiel von BLOAD) :

BLOAD "Dateibez." (,#banknr,adresse)

Wird die Banknummer und Adresse nicht angegeben, so wird das ML-Prg. an dieselbe Stelle wie beim Abspeichern geladen. Wird eine Ladeadresse angegeben, so muss darauf geachtet werden, dass vor dem Start alle absoluten Adressen geändert werden.

Die Adressen von &F000 bis &FFFF in Bank, #0 sind für das System reserviert. Hier werden Einstellungen für den Drucker, für die Speicherverwaltung, für die Disketten, etc. gespeichert. Nachfolgend eine Tabelle, die Ihnen diese Speicherzellen vorstellt. Zuerst eine Grobeinteilung des Speichers von &F000 bis &FFFF.

A	&F000	
	&F05C	IOCS-Bereich
	&F1B2	Interpreterbereich 1
	&F21D	Editierbuffer (256 Bytes)
	&F31D	Interpreterbereich 2
	&F3C7	Momentanes FCB
	&F500	Z80-Stack (256 Bytes)
B	&F600	PC-1500(A)-Display 1
	&F650	Variablen E$ - O$
	&F700	PC-1500(A)-Display 2
	&F750	Variablen P$ - Z$
C	&F800	Systembreich für PC-1500
D	&FC00	RAM-File Bereich
	&FCB0	Interpreterbreich 3
E	&FF21	Reserviert für System

Nunfolgend ist der Arbeitsbereich für das System abgedruckt.

Adresse	Erklärungen
F02D	MAXFILES-Wert
F04C	RS-232-Bufferstartadresse Lowbyte
F04D	RS-232-Bufferstartadresse Highbyte
F04E	FCB-Bufferstartadresse Lowbyte
F04F	FCB-Bufferstartadresse Highbyte
F05C	LCD-Display-Startlinie
F05D	LCD-Arbeitsbereich 1 Bit0:LCD-Mode (=0) Bit2:Zeichengenerator 0:PC-1600 1:PC-1500 (A) Bit3:Controlzeichen 0:nicht angezeigt 1:angezeigt Bit4:Cursor Blinkgeschwindigkeit 0:langsam 1:schnell
F05E	LCD-Arbeitsbereich 2 Bit0:Cursor Blinkarbeitsbereich Bit1:Interruptrequest-Maske für LCD
F05F	Cursor X-Koordinate
F060	Cursor Y-Koordinate
F061	Adresse (Low) Controlzeichengrafik
F062	Adresse (High) Controlzeichengrafik
F063	Banknummer für obenstehende Adresse
F064	Adresse (L) Grafik Zeichen &80-&FF
F065	Adresse (H) Grafik Zeichen &80-&FF
F066	Banknummer für obenstehende Adresse
F067	Cursortyp &00:kein Cursor &01:Strich-Cursor &02:Quadrat-Cursor &03:Space-Cursor
F068	Cursorblinken-Zähler

Adresse	Erklärungen
F079	Tasten-Arbeitsbereich 1 Bit1:Druck-Beep 0:Aus 1:Ein Bit2:Wiederholung 0:Aus 1:Ein Bit3:zu wiederholende Tasten 0:alle, ausser Spezialtasten 1:alle Tasten Bit4:Wiederholungsverzögerung 0:1 Sekunde 1:0.8 Sekunden Bit7:Tastencodeumrechnung 0:Ein 1:Aus
F07A	Tasten-Arbeitsbereich 2
F07B	Tasten-Arbeitsbereich 3
F182	Zeichenabstand des PITCH-Befehls
F183	Zeilenabstand des PITCH-Befehls
F184	Farbwahl Plotter
F185	Zeichen pro Zeile
F187	Einstellung ob LF,CR,LF+CR Bit5: Bit6: =1 + =1: LF =1 + =0: CR + LF =0 + =1: CR
F188	Schreibbereichzähler Lowbyte
F189	Schreibbereichzähler Highbyte
F18F	Rechtes Ende Schreibbereich Lowbyte
F190	Rechtes Ende Schreibbereich Highbyte
F191	Linkes Ende Schreibbereich Lowbyte
F192	Linkes Ende Schreibbereich Highbyte
F194	Schreibkopfposition (von links)
F88F	Ausgabebuffer-Zeiger
F890	Stack-Zeiger für FOR ... NEXT

Adresse	Erklärungen
F891	Zeiger für GOSUB
F894	Zeiger für Stringbuffer
F895	USING format (Punkt oder Komma)
F896	USING M (Integerteil von USING)
F897	USING & (für Buchstaben)
F898	USING m (Dezimalpunkt von USING)
F899	Variablenzeiger Highbyte
F89A	Variablenzeiger Lowbyte
F89B	Errornummer des entstandenen Fehlers
F89C	Momentane Programmlinie (Higlibyte)
F89D	Momentane Programmlinie (Lowbyte)
F89E	Programmstartadresse der Zeile (H)
F89F	Programmstartadresse der Zeile (L)
F8A6	Adresse (H) von der Linie beim Suchen
F8A7	Adresse (L) von der Linie beim Suchen
F8A8	Zeilennr (H) von Linie nach Suchen
F8A9	Zeilennr (L) von Linie nach Suchen
F8AA	Startadresse (H) von gesuchtem Block
F8AB	Startadresse (L) von gesuchtem Block
F8AC	Adresse (H) wo BREAK entstand
F8AD	Adresse(L) wo BREAK entstand
F8AE	Zeilennummer (H) wo BREAK entstand
F8AF	Zeilennummer (L) wo BREAK entstand
F8B0	Startadresse (H) von Block, wo BREAK
F8B1	Startadresse (L) von Block, wo BREAK
F8B2	Adresse (H) wo ERROR entstand
F8B3	Adresse (L) wo ERROR entstand
F8B4	Zeilennummer (H) wo ERROR entstand
F8B5	Zeilennummer (L) wo ERROR entstand
F8B6	Startadresse (H) von Block, wo ERROR
F8B7	Startadresse (L) von Block, wo ERROR
F8B8	Adresse (H) wo ON ERROR hinspringt
F8B9	Adresse (L) wo ON ERROR hinspringt
F8BA	Zlnnummer (H) wo ON ERROR hinspringt
F8BB	Zlnnummer (L) wo ON ERROR hinspringt
F8BC	Adr.(H) von Block wo ON ERROR hinspr.
F9BD	Adr.(L) von Block wo ON ERROR hinspr.

Adresse	Erklärungen
F8C0-F8CF	Inhalt von A$
F8D0-F8DF	Inhalt von B$
F8E0-F8EF	Inhalt von C$
F8F0-F8FF	Inhalt von D$
F900 -F9CF	Inhalte von A-Z (jeweils 8 Byte lang)
F9D1	Spezifikation der Peripheriegeräte
F9E0	Zähler für X-Koordinate Plotter (H)
F9E1	Zähler für X-Koordinate Plotter (L)
F9E2	Zähler für Y-Koordinate Plotter (H)
F9E3	Zähler für Y-Koordinate Plotter (L)
F9E4	Schreibfläche: Zähler Y-Richtung (H)
F9E5	Schreibfläche: Zähler Y-Richtung (L)
F9E6	absolute X-Position des Stifts
F9E7	Schreibfläche: Zähler X-Richtung (H)
F9E8	Schreibfläche: Zähler X-Richtung (L)
F9EA	Linientyp
F9EB	Zähler für gepunktete Linien
F9EC	Status des Stiftes (Oben/unten)
F9ED	Zähler für Motor in X-Richtung
F9EE	Momentane Motor hase
F9EF	Zähler für Motor in Y-Richtung
F9F0	Modus: &FF=GRaph / &0=Text
F9F2	Rotate-Richtung
F9F3	Schreibfarbe
F9F4	CSIZE
F9EF	Plottermodi:

```
Bit0:Printmodus
       1:Grapfh
       0:Text
Bit1:Papierart
       0:Einzelblatt
       1:Rolle
Bit5:Drucker Ready-Status
       0:Printer Ready
       1:Printer not Ready
```

Adresse	Erklärungen
	Bit6:Stiftplatzierung 0:Stift nicht am Wechselort 1:Stift am Wechselort Bit7:Initialisieren 0:Printer ist nicht init. ist initialisiert
F9F4	Werte des ROTATE-Befehls Bit0-3 :Schreibrichtung 0-3 Bit4-7 :Rotate 0-3
F9F5	Wert des CSIZE Befehls Bit0-3 :CSIZE 1-9
F9F6	Linientyp Bit0-3 :Typ 1-9
F9F7	PZONE-Wert
F9F8	Arbeitsbereich Plotter Bit0:LLIST 0:Normalmodus 1:LLIST in bel. CSIZE-Grösse Bit1:Stift 0:Normalmodus 1:bleibt nach RLINE oder LLINE auf Papier (wird z.B bei Linientyp 20 verwendet) Bit5:Papierbewegung 0:Normalmodus 1:Stift bewegt sich nicht bei Druck auf "hoch"-Taste Bit6:Schreibfläche 0:Normalmodus 1:Y wird nicht qezätilt im Grafikmodus
F9FF	LOCK/UNLOCK
FB00-FB07	benötigt für Zufallszahlen

In der Maschinensprache kann man natürlich auch die im ROM verwendeten Routinen für- die diversen Basic-Befehle benützen. So lässt sich z.B. auch in ML mit den Befehlen SIN, TAN, RND, etc arbeiten. Aus diesem Grund müssen zuerst die verschiedenen Zahlendarstellungen erklärt werden.

Als erstes sei hier die Verwaltung von Nachkommazahlen beschrieben. Eine Nachkommazahl kann mit 8 Bytes dargestellt wenden. Dieses Format enthält Exponent, Mantissenvorzeichen und Mantisse. Das 8-Byte-Format lässt Zahen zwischen $-9.999999999 \times 10^{99}$ und $9.999999999 \times 10^{99}$ zu.

Der Exponent wird in einem Byte dargestellt. Negative Exponenten sind vom Wert 256 zu subtrahieren. Das Vorzeichen der ganzen Zahl benötigt ebenfalls ein ganzes Byte. &00 bedeutet positives Vorzeichen, &80 bedeutet negatives Vorzeichen. Die folgenden 5 Bytes stellen die Mantisse dar (im BCD-Format).
Anhand eines Beispieles ist diese Darstellung am besten erklärt.

Beispiel:
Um den Wert 7218 (oder 7.218×10^3) darzustellen, müssen die 8 Bytes wiefolgt belegt sein:

A	B	C	D	E	F	G	H
&03	&00	&72	&18	&00	&00	&00	&00

A : Exponent=3, da Tausender
B : 0, da positives Vorzeichen
C-G: Mantisse im BCD-Format
(immer 4 Bit stellen eine Zahl dar)
H : Ist bei Nachkommazahlen immer 0

Ein anderes Beispiel für die Zahl -0.00028182 folgt nun, damit auch negative Exponenten gezeigt wurden. ($-0.00028182 = -2.8182 \times 10^{-4}$)

A	B	C	D	E	F	G	H
&FC	&80	&28	&18	&20	&00	&00	&00

A : Exponent=-4 (a1.so 256-4=&FC), da Zehntausendstel
B : &80, da negatives Vorzeichen
C-G: Maritisse im BCD-Format immer 0

Eine ganze Zahl von -32768 bis 32767 kann mit folgendem Format dargestellt werden. (Negative Zahlen müssen zuerst vom Wert &10000 oder dez. 66536 subtrahiert werden)

Beispiel:

Die Zahl 1.4855 soll als Binärausdruck (so nennt man dieses Darstellungsformat) dargestellt werden:

A	B	C	D	E	F	G	H
*	*	*	*	&B2	&3A	&07	*

A-D:egal
E :muss &B2 enthalten um einen Binärausdruck darzustellen
F :Highbyte des Ausdruckes (HEX$(14855)=&3A07
G :Lowbvte des Ausdruckes (HEX$(14855)=&3A07

Die Zahl -7762 ergäbe in F und G folgende Werte:
F :&E1
G :&AE

65536-7762 ergibt 57774 HEX$(57774) = &E1AE.

Ein letztes Darstellungsformat muss angewendet werden für die Verwaltung von Zeichenketten. Dabei ist es wichtig zu wissen, dass die 8 Bytes nur die Adresse an der der String steht und dessen Länge speichert. Die Adresse wird aber nicht einfach mit High- und Lowbyte speichert, sondern das 7.Bit des High-Byte wird zuerst noch invertiert Steht das High-Byte im A-Register, wird das 7.Bit mit XOR &80 invertiert.

Beispiel:

Der Text "Taschencomputer" steht, ab im Speicher. Er soll nun als String behandelt werden. Wir geben folgendes in die 8 Byte ein:

A	B	C	D	E	F	G	H
*	*	*	*	&D0	&BB	&2F	&0F

A-D :egal
E :&D0, da es sich um einen String handelt
F :Higb-Byte der Adresse (mit invertiertem 7.Bit)
G :Low-Byte der Adresse
H :Länge der Zeichenkette (15 Zeichen)

Wir haben uns nun für die 3 Darstellungsformen Interessiert, doch wie arbeitet man mit diesen?
Um die Arbeit mit dem Betriebssystem zu ermöglichen, wurden 7 sogenannte arithmetische Register geschaffen. Sie sind natürlich nur fiktive Register, d.h. nicht mit ML-Befehlen ansprechbar. Sie repräsentier-E~n nur einen für diesen Zweck reservierten Speicherbereich.
Folgende 7 arithmetische Register, sind verfügbar:

Register Adresse

X	FA00-FA07
Y	FA08-FA0F
Z	FA10-FA17
U	FA18-FA1F
V	FA20-FA27
W	FA28-FA2F
S	FA30-FA38

Mit diesen "Registern" kann man auch in ML auf die vom Betriebssystem offerierten Befehle wie SIN, INT, RND, etc. zurückgreifen.

Funktionen mit zwei Registern:

1. Schreiben Sie die Argumente in das X und Y Register. Dabei. ist darauf züi achten, das der erste Term im X und der 2.Term im Y-Register zu stehen kommt (Bsp.: 2^5: X=2,Y=5)
2. Schreiben Sie in die Adresse &F88C den Wert &02, da es sich um eine Funktion mit 2 Registern handelt
3. Rufen Sie die Adresse der gewünschten Funktion auf:

Funktion Adresse

+	&021A
-	&021D
*	&0220
/	&0223
^	&01F3
AND	&01F6
OR	&01F9

Ist die Berechnung erfolgt, so wird das CY=0 gesetzt und das Resultat wird im "Register" X gespeichert. Ist ein Fehler aufgetreten, so wird CY=1 gesetzt und im A-Register steht der Fehlercode (entsprechend dem BASIC-Fehlercode). Bei der Ausführung einer der obengenannten Routinen werden folgende Register zerstört, d.h. mit anderen Werten belegt: HL,DE,BC,AF,AF'

Um Vergleichsoperationen (zwischen 2 Zahlen) durchzuführen, ist folgendes auszuführen:

1. Beide zu vergleichenden Argumente im BCD-Format in X und Y speichern.
2. Je nach Vergleichsoperator folgender Wert in DE einladen:

Operator\Code für DE

< >	&8000
<	&8001
>	&8002
=	&8004
< =	&8005
> =	&8006

3. Den Wert &02 in die Adresse &F88C schreiben (da 2 Register verwendet werden)
4. CALL &01FC durchführen

Ist der Vergleich "wahr", so wird der Wert 1 im X-Register gespeichert, ist er falsch, so steht 0 im X-Register. Die Z80-Reqister HL,DE,BC,AF und AF' werden zerstört.

Funktionen mit einem Register:

Für Funktionen, die nur ein Argument annehmen, ist folgendermassen vorzugehen:

1. Das Argument ins X-Register schreiben (natürlich im BCD-Format)
2. DE nach untenstehender Liste laden:

Funktion\DE-Register

SQR X	&F16B
LN X	&F176
LOG X	&F177
EXP X	&F178
SIN X	&F17D
COS X	&F17E
TAN X	&F17F
ASN X	&F173
ACS X	&F174
ATN X	&F175
DEG X	&F165
DMS X	&F166
ABS X	&F170
SGN X	&F179
INT X	&F171
NOT X	&F16D
RND X	&F17C
XPEEK X	&F16E
PEEK X	&F16F
STATUS X	&F167
POINT X	&F168

3. Den Wert &01 in die Adresse &F88C laden (Da nur 1 Register, verwendet wird).
4. CALL &0202

Nach Rückkehr aus der Routine ist das CY-Flag =0 gesetzt, wenn kein Fehler auftrat. Im X "Register" steht das Resultat. Entstand ein Fehler, so wird CY=1 gesetzt und im A-Register steht der Errorcode (entsprechend dem Basic-Fehler). Die Register HL,DE,BC,AF und AF' werden zerstört.

String-Funktionen mit einem Register:

Dieses Format kommt nur beim Befehl STR$ vor. Diesen Befehl rufen Sie in ML wiefolgt auf:

1. Setzen des Argument in das X-Register (entweder im BCD- oder im Binärformat)
2. DE-Register mit &F161 laden (Interner Code für den Befehl STR$)
3. Die Adresse &F88C mit &01 laden
4. Die Adresse &F894 mit &10 laden (Diese Adresse zeigt auf den Stringbuffer)
5. CALL &0202

Nach Durchlauf der Routine, wird CY=0 gesetzt und die Adresse der generierten Zeichenkette steht im X-Register. Vergessen Sie nicht, um die korrekte Adresse züi erhalten, das 7.Bit des High-Byte zu invertieren (Siehe S.64). Die Register HL,DE,BC,AF und AF' werden zerstört.

Funktionen mit einem String-Register:
Um die Basic-Befehle VAL, ASC und LEN in ML zu gebrauchen, muss wiefolgt vorgegangen werden:
1. X-Register mit betreffendem String belegen.
2. Folgendes ins DE-Register einladen:

Fkt.	DE-Register
VAL X	&F162
ASC X	&F160
LEN X	&F164

3. Die Adresse &F88C mit &01 laden.
4. CALL &0202

Ist die Funktion korrekt durchlaufen worden, so wird CY=0 gesetzt und im X-Register steht das numerische Resultat (entweder im BCD- oder im Binärformat). Ist ein Fehler entstanden, so wird CY=1 gesetzt und im A-Register steht der Fehlercode, welcher dem des Basics entspricht.
Die Register HL, DE, BC, AF und AF' werden zerstört.

Die Befehle RIGHT$, LEFT$ und MID$ können auch emuliert werden, doch ist die Anwendung komplizierter als eine eigene Routine für diese Befehle zu schreiben.

Vergleichsoperationen zwischen Strings:

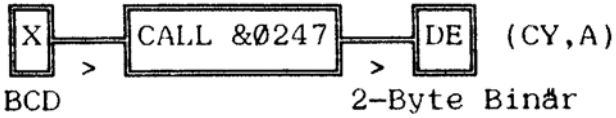
Um 2 Zeichenketten miteinander zu vergleichen, gehen Sie wiefolgt vor:
1. Ins X-Register den ersten String und ins Y-Register den zweiten String laden.
2. Ins DE-Register den folgenden Wert einladen, je nach gewünschter Operation:

Operator	Code für DE
< >	&8000
<	&8001
>	&8002
=	&8004

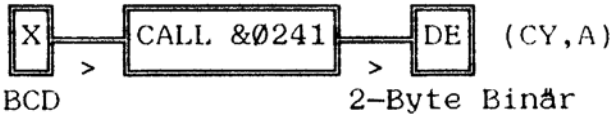
3. CALL &01FF

Ist der Vergleich richtig, wird "1" ins X-Register geschrieben, ist er falsch, so steht dort "0".
Zerstörte Register: HL,BC,DE,AF,AF'

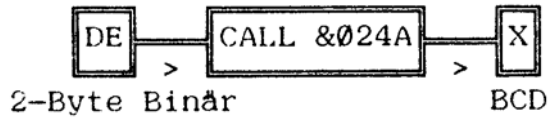
Nunfolgend sind Unterprogramme erklärt, die Sie für Ihre ML-Programme benützen können. Es handelt sich hier in einem ersten Teil um Zahlenkonvertierungsprogramme dann um Textbehandlung.



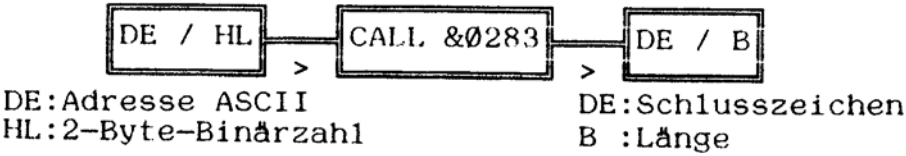
Wandelt eine BCD-Zahl von 0-65535 im X-Register in eine 2-Byte-Binärzahl um die in DE-Register zu stehen kommt. Hat die Routine einen Fehlerergeben, so wird CY=1 gesetzt und der Fehlercode ins A-Register geschrieben. Die Register AF, BC, DE und HL werden zerstört.



Wandelt eine BCD-Zahl von -32768- 32767 im X- Register in eine 2-Byte-Binärzahl um, die in DE-Register zu stehen kommt. Hat die Routine einen Fehler ergeben, so wird CY=1 gesetzt und der Fehlercode ins A-Register geschrieben. Alle Register werden zerstört.



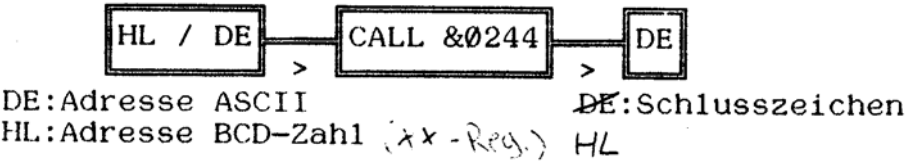
Wandelt eine 2-Byte Binärzahl im DE-Register in eine BCD-Zahl ins X-Register um. Alle Register werden zerstört.



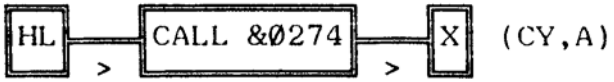
Wandelt eine 2-Byte Binärzahl (im HL-Register) in einen ASCII-String um. Dazu wird die Adresse, wo der ASCII-String abgelegt werden soll benötigt (im DE-Register). Nach Aufruf steht in B die Länge des Strings und in DE die Adresse des letzten Zeichens.
Zerstörte Register: HL, DE, BC, IX, AF

Beispiel: Wenn HL = &829E (oder 33438 dez.)

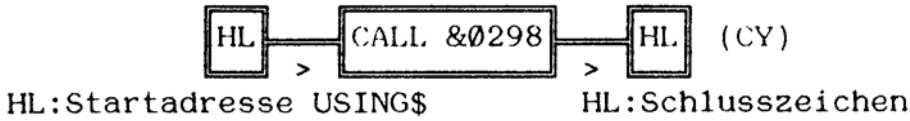
(DE) vor Aufruf (DE) nach Aufruf
&33 &33 &34 &33 &38 B=&05 (5 Ziffern)



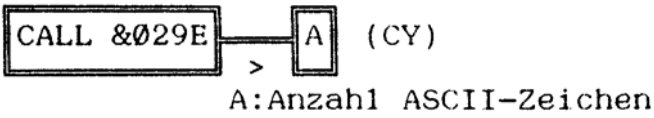
Wandelt eine BCD-Zahl in einen ASCII-String (ab Adresse im DE-Register) um. Ist der absolute Wert des Exponenten 10 oder grösser, so wird in die Exponentialschreibweise umgeformt (z.B. 2.42819 E23), wenn nicht, als Zahl ohne Exponent. Ein evt. Vorhandenes "+" wird durch ein Leerzeichen ersetzt. Die BCD-Zahl muss an der Adresse beschrieben durch HL stehen.
Zerstörte Register: AF, BC, DE, HL, IX, IY, AF'



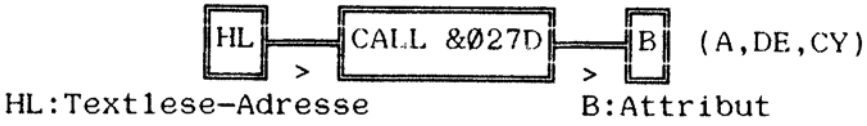
Diese Routine berechnet einen Ausdruck im Intermediate Code-Format (Im BASIC-Format) ab Adresse im HL-Register und schreibt das Resultat als BCD-Zahl ins X-Register. CY wird 1, falls ein Fehler auftrat (Fehlercode im ARegister). Zerstört: AF, BC, DE, HL, IX und IY.



Kontrolliert den ASCII-String im USING-Format ab Adresse im HL-Register und überträgt das USING-Format in die USING-Peeks &F895-&F898.
Entstand Fehler, wird CY=1 gesetzt. Zerstört die Register HL und AF.



Konvertiert den numerischen Wert im X-Register in einen ASCII-Stririq formatiert durch USING (falls vorhanden). Das Resultat wird in den Registern Y, U, V, W und S (also von &FA10 bis &FA37) gespeichert. CY=1, wenn Fehler auftrat. Zerstörte Register: AF, AF' und BC.

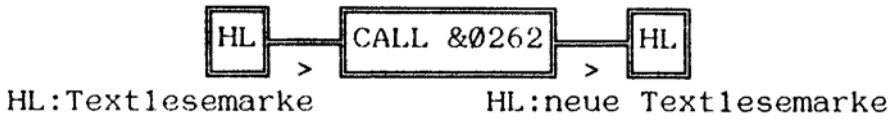


Liest das Attribut (ob Zeilennummer, reserviertes BASIC-Wort, ASCII-Zeichen) vom Inhalt der Adresse (HL-Register). Das Attribut sieht folgendermassen aus:

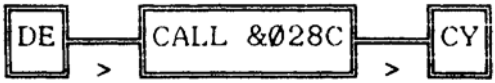
B-Register:

Bit0: 1, wenn ASCII-Zeichen (ASCII > A-Register)
Bit1: 1, wenn BASIC-Wort (Code > DE-Register)
Bit2: 1, wenn Zeilennummer (Wert > DE-Register)

Zeilennummern werden ins DE-Register übertragen, wenn Sie im Binärformat (d.h. &lF, &high, &low, &00) stehen, ansonsten (ASCII-Format: GOTO 100 = &F1,&92 (für GOTO) &31,&30,&30 (für "1"+"0"+"0") wird nacheinander ins A-Register übertragen. HL wird zum nächsten Wert verschoben. CY wird wenn der End-of-text-Code (&FF) erkannt wurde.
Zerstörte Register: AF, B, DE und HL.



Lässt HL die momentan bearbeitete Zeile über-springen und springt zum CR der Zeile. AF und HL, werden zerstört.



DE:gesuchte Linie CY:gefunden ?

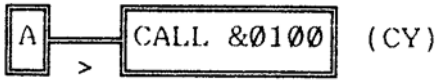
Sucht die Linie mit der Zeilennummer beschrieben durch das DE-Register. CY=0, dann wurde die Zeile (oder die nächst grössere) gefunden. Die Zeilennummer steht in &F8A8 und &F8A9 (high/low-Byte), die Adresse (mit dem 7.Bit des High-Byte invertiert) steht in &F8A6 und &F8A7 (high/low-Byte). In &F1C3 steht die Banknummer). Ist CY=1 wurde die Zeile nicht gefunden. Die obengenannten Speicherzellen beinhalten in dem Fall die Daten der letzten Zeile im Programm. AF wird zerstört.

HL:Adresse der Sprungmarke (Bsp.:"START")
BC:Anzahl Zeichen Marke CY:Marke gefunden ?

Sucht die Zeilennummer, in der die Marke (bezeichnet durch Adresse (HL) und Länge (BC) der Marke) steht. CY=0, wenn Marke gefunden. &F8A8 und &F8A9 (High/Low) speichern die Zeilennummer, &F8A6 und &F8A7 (High/Low) speichern die Adresse (Achtung:7.Bit des Byte ist invertiert). Ist CY=1, wurde die Marke nicht gefunden. Da unter umständen Bankswitching betrieben werden muss, kann es vorkommen, das die CALL-Seite in die Originalbank zurückswitchen muss.
Zerstört werden: AF, BC, DE, HL und AF'.

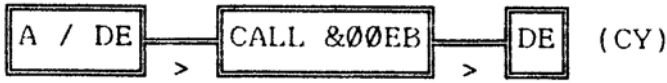
Der PC-1600 hat viele verschiedene IOCS-Routinen mit ihnen lassen sich Vorgänge, die bei BASIC- Befehlen benötigt werden, auch in Maschinensprache Verwenden. Viele IOCS-Routinen wären zwar überflüssig, da sie durch spezielle INP- und OUT- Befehle (siehe Systemhandbuch von Fischel) ersetzbar sind, doch ist nicht gewährleistet, dass diese Port-adressen in zukünftiqen Versionen des PC-1600 beibehalten werden. Die IOCS-Startadressen werden aber auf allen PC-1600 Versionen die gleichen sein. Als erstes werden die IOCS-Routinen für das Display behandelt:

D i s p l a y



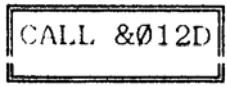
A:Character-Code CY:Letztes Linienzeichen?

Diese Routine stellt ein ASCII-Zeichen (ARegister) auf dem Display bei der aktuellen Cursorposition dar Diese kann durch Ändern der Werte in &F05F (Y-Koord.) und &F060 (X-Koord.) verändert werten). Die X-Koordinate wird um eins erhöht. Ist das Zeichen auf die letzte Position der Linie geraten, so wird das CY=1, gesetzt. Die X-Koordinate wird beibehalten und der Cursor wird ausgeschalten.
Zerstörte Register: AF, (&F060), (&F067)

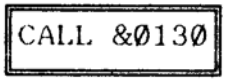


A :Erkennung letztes Zeichen CY:Displayende ?
DE:Adresse wo String abgelegt DE:Schlusszeichen

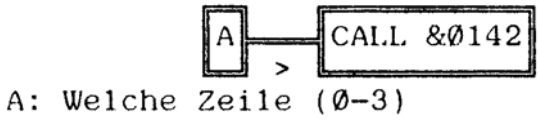
Diese Routine stellt fortlaufend Zeichen aus dem Speicher dar (beginnend bei Adresse gespeichert in DE). Liest die Routine aber ein Zeichen ein,das dem Code im A-Register entspricht, bricht sie ab. Dieses Zeichen wird nicht mehr angezeigt. CY erhält sich gleich wie bei der Routine vorher. Nach Durchlauf der Routine enthält DE die Adresse des Abbruchzeichens.
Es wird zerstört: AF, DE,(&F060), (&F067).



Scrollt den Bildschirm um eine Zeile aufwärts.
Die letzte Zeile wird gelöscht und der Cursor ausgeschaltet. Keine zerstörten Register.



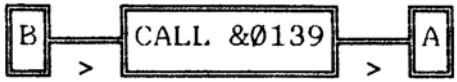
Scrollt den Bildschirm um eine Zeile abwärts. Die erste Zeile wird gelöscht und der Cursor ausgeschalten. Keine zerstörten Register.



Fügt eine leere Zeile an der durch das A-Register bestimmten Stelle ein. Was Innerhalb dieser Linie steht, wird abwärts gescrollt. Der Cursor wird ausgeschalten. Zerstört wird AF.

A :Welche Zeile (Ø-3)

Löscht die, durch das A-Register angegebene Zeile. Das Darunterliegende wird nicht gescrollt. Zerstörte Register: AF.



B :Symbolsatz A :Status

Liest die aktuelle Belegung der StatuslineSymbole. B beschreibt den Satz der Symbole (siehe folgende Grafik). A enthält nach Aufruf die Belegung.

B=&ØØ	DEF	I	II	III	SMLL		SHFT	BUSY
B=&Ø1		RUN	PRO	RSRV		RAD	G	DE
B=&Ø2	KBII				<S>		CTRL	BATT
A=	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	BitØ

Im A-Register werden die jeweiligen Bits gesetzt. Zerstörtes Register: AF

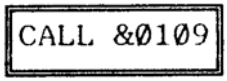
A :Symbolstatus
B :Symbolsatznr.

Setzt die Symbole der Statuslinie. Im BRegister muss der Satz gespeichert sein, der verändert werden soll und im A-Register die neue Belegung qemäss Skizze auf der vorhergehenden Seite. Keine zerstörten Register.

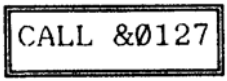
D :X-Koordinate (Ø-25)
E :Y-Koordinate (Ø-3)
A :Anzahl Zeichen

CY:X & Y im Display ?

Invertiert auf dem Display eine Zeichenkette (auch Grafik). Beginnend von den Koordinaten in den Registern D und E nach rechts soviele Zeichen wie im A-Register stehen. CY=1, wenn D und/oder E einen Wert enthalten, der nicht im Bereiche des Displays liegt. Cursor wird ausgeschaltet.



Setzt den Cursor in die linke obere Ecke im Display (0,0) und schaltet in den CharacterModus. (&F05F) (Y-Koordinate) und (&F060) (X-Koordinate) werden auf 0 gesetzt.



Diese Routine wirkt gleich wie der PSET-Befehl in BASIC.

Um festzusetzen, ob ein Punkt erscheinen, gelöscht. oder invertiert werden soll, muss die Speicherstelle &F096 entsprechend gesetzt-
werden. Dabei gilt: &00 Punkt setzen
&01 Punkt löschen
&02 Punkt invertieren.
Vor dem Aufruf der Routine, müssen die X- und Y-Koordinate gesetzt werden. Dies geschieht wiefolgt:

X-Koord.: &F08E Lowbyte
&FO8F Highbyte
Y-Koord.: &F090 Lowbyte
&F091 Highbyte

Die effektiven Grafikkoordinaten des Display lauten:

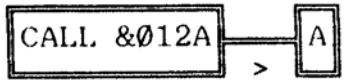
X-Richtung: 0 <= X <= 155
Y-Richtung: 0 <= Y <= 31

CALL &Ø12Ø

Zeichnet eine Linie auf dem Display. Dazu muss die Speicherzelle &F096 gemäss obenstehender Liste belegt werden. Dazu kommt, dass die Koordinaten von Anfangs- und Endpunkt definiert werden müssen. Dies betrifft folgende Speicherzellen:

X (Anfang):&F08E Lowbyte
&F08F Highbyte
Y (Anfang):&F090 Lowbyte
&F091 Highbyte
X (End): &F092 Lowbyte
&F093 Highbyte
Y (End): &F094 Louvbyte
&F095 Highbyte

Die Linienform (Bitmuster) müssen Sie in den Speicherzellen &F097 (Lowbyte) und &F098 (Highbyte) bestimmen. Wie dies geht, entnehmen Sie der Bedienungsanleitung auf Seite 14-105.Nach dem Aufruf sind in den Koordinaten für den Anfangspunkt die des letzten Endpunktes Gespeichert (Dies erleichtert fortlaufendes Linienzeichnen). Auch das Bitmuster ist so verändert, dass man gleich weiterzeichnen kann.
Andere Veränderungen: AF, BC, DE, HL.



A :Punkt gesetzt?

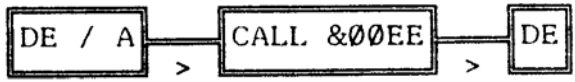
Entspricht dem BASIC-Befehl POINT. A wird mit dem Wert &01 belegt, wenn der Punkt auf dein Display (mit den Koordinateri X (&F08E Low &F08F High) und Y (&F090 Low &F091,High) gesetzt ist, ansonsten enthält das A-Register den Wert &00. Ausser AF wird nichts zerstört.

A :ASCII-Zeichen

Stellt das Zeichen im A-Register an der aktuellen Grafikcursor-Position dar und verschiebt die X-Koordinate des Cursors um 6 nach rechts. Dabei hat die Belegung der Speicherzelle &F096 folgende Bedeutung:

&00 Was darunter liegt, löschen
&01 Nicht löschen, darüber schreiben (OR)
&02 Gesetzte Punkte invertieren (XOR)

Zerstört wird beim Aufruf dieser Routine nur das AF-Register.

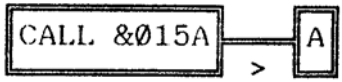


DE:Startadresse Text DE:Endadresse Text
A :Abbruchzeichen

Stellt solange Text (gespeichert ab DE-Register) ab der aktuellen GrafikCursorposition dar, bis ein Zeichen, welches dem des A-Register entspricht, eingelesen wird. Die Zelle &F096 bewirkt gleiches wie bei. der vorhergehenden Routine. Zerstört DE, AF.

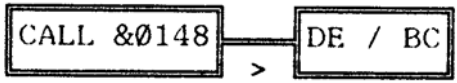
A :Bitmuster

Stellt das Bitmuster des A-Reqisters an der aktuellen Position des Grafikcursors (X:(F099 L/F09A H), Y:(F09B L/F09C H)) dar. Dabei bewirkt der Inhalt von &F096 dasselbe wie bei den 2 zuvor beschriebenen Routinen. Die XKoordinate wird um eins vergrössert und das AF-Register zerstört.



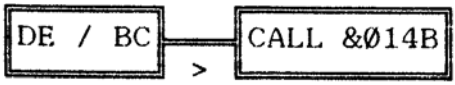
A :Bitmuster

Liest das Bitmuster, das an der aktuellen Grafikcursorposition steht in das A-Register. Dabei wird ersichtitlicherweise das AF-Register zerstört.



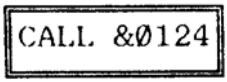
DE:X-Koordinate
BC:Y-Koordinate

Liest die aktuelle Position des Grafik-Cursors. In DE steht die X- und in BC die Y-Koordinate.
Belegt den Grafikcursor mit den Werten vom DE-(X-Koordinate) und vom BC-Register (Y-Koord.)

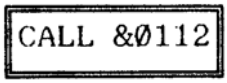


DE:X-Koordinate
BC:Y-Koordinate

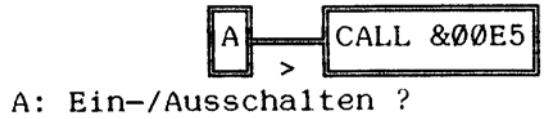
Belegt den Graphikcursor mit den Werten vom DE-(X-Koordinate) und vom BC-Register (Y-Koordinate)



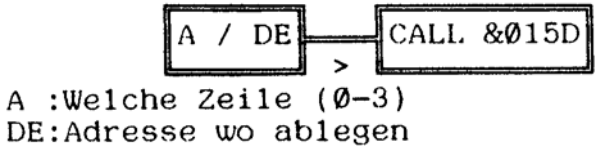
Zeichnet ein gefülltes Rechteck. Die Speicherzellen, die belegt werden müssen, entsprechen denen des LINE-Befehls. Dies ist auf Seite 79 beschrieben. Die Linie, die Sie mit den Koordinaten beschreiben, ist die Diagonale des Rechteckes.



Löscht das Display und schaltet den Cursor aus.



Schaltet das Display ein (bei A=&01) respektiv aus (bei A=&00). Zerstört wird das AF-Register.



Speichert den Inhalt einer Displayzeile (Bestimmt durch A-Register) in einen Speicherbereich beginnend ab Adresse im DERegister. Die Zeile benötigt logischerweise 156 Bytes an Speicher und liegt von (DE) bis (DE)+155 (also (DE)+&9B) im Speicher.

A :Welche Zeile (Ø-3)
DE:Adresse wo im Speicher

Bewirkt das Umgekehrte der vorhergehenden Routine. Hier wird vom Speicherbereich (beginnend mit der Adresse, die in DE steht) in die Displayzeile kopiert, die im A-Reqister steht.

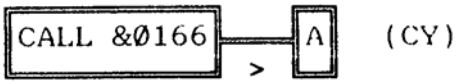
BEISPIEL :

Anhand folgendem Beispiels soll aufgezeigt werden, wie man mit diesen IOCS-Routinen arbeiten kann. Hierbei handelt es sich um eine ML-Routine, die den Inhalt der ersten Zeile solange nach links scrollt (Softscrolling), bis eine Taste gedrückt, wird. Das ML-Programm ist nicht frei verschiebbar (ggf. Adressen ändern):

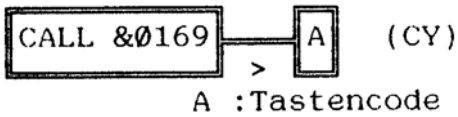
Startadresse: &C000

```
LD A, &00
L,D DE,&C100        1.Displayzeile ab &C100 saven
CALL &015D
LD A,(&C100)        - erstes Bitmuster retten
LD HL,&C101
LD DE,&C100         Um ein Byte linksverschieben
LDIR
LD (&C19B),A        - Gerettetes an letzte ctelle
LD A,&00
LD DE,&C1000        Displayzeile anzeigen
CALL &0160
CALL &0169          Testet, ob Taste gedrückt
JP C,&C000          Nein, also neu beginnen
RET                 Ja, also zurück zum BASIC
```

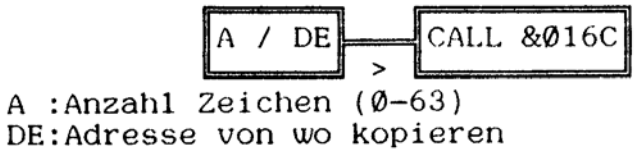
TASTATUR



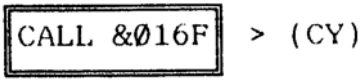
Diese Routine liest ein Zeichen vom Tastaturbuffer. Ist dieser leer, so wartet die Routine (10 Minuten lang, dann wird ausgeschalten). A enthält das erste Zeichen aus dem Tastaturbuffer. Ist CY=1, wurden die 10 Minuten überschritten.



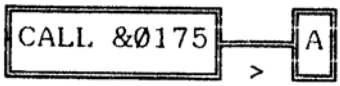
Entspricht obenstehender Routine, nur dass nicht auf Eingabe gewartet wird. Ist der Buffer leer, so wird CY=1 gesetzt.



Löscht den Tastaturbuffer und lädt soviele Zeichen wie in A angegeben von der Adresse (in DE angegeben) in den Tastaturbuffer.



Testet, ob die BREAK/ON- Taste gedrückt wird. Wenn ja, wird CY=1 gesetzt und der Tastaturbuffer gelöscht. Ansonsten bleibt CY=0. Das AF-Register wird zerstört.

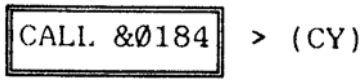


A :Tastencode
(&ØØ=keine Taste)

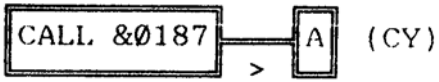
Schreibt den Code der Taste, die beim Aufruf gedrückt wird, in das A-Register. Tastaturbuffer wird gelöscht und AF-Register wird zerstört.

A :&ØØ=Tastatur
&Ø2=RS232C-Schnittstelle

Definiert das Eingabemedium. Bei A=&02 werden alle Tasteneingabenroutinen auf die RS-232C-Schnittstelle umgeleitet. AF wird zerstört.



Liest, ob die OFF-Taste gedrückt wurde. CF=0, die OFF-Taste wurde nicht gedrückt. CF=1, sie wurde gedrückt. AF wird zerstört.



A :Tastencode

Liest ein Zeichen vom Tastaturbuffer in das ARegister ohne den Tastaturbuffer zu verändern. Ist CY=1, war der Buffer leer.

DATEIEN:

Das BASIC vom PC-1600 unterscheidet 3 Typen von Dateien:
1. ASCII-File
2. Basicprogrammm
3. ML-Programm

Basic- und ML-Programme starten mit einem 16Byte Kopf am Anfang des Files.

+&00:	File-Kopf existiert	&FF = Basic / ML, <> &FF = ASCII-File
+&01:	ID-Code	&10
+&02:	Reserviert	&00
+&03:	Reserviert	&00
+&04:	Modus	&10 = ML-Prg. &21 = BASIC-Prg.
+&05:	Grösse (Lowbyte)	
+&06:	Grösse (Middlebyte)	
+&07:	Grösse (Highbyte)	
+&08:	Ladeadresse (Low)	nur für ML-Prg.
+&09:	Ladeadresse (High)	nur für ML-Prg
+&0A:	Ladeadresse (Bank)	nur für ML -Prg.
+&0B:	Startadresse (Low)	nur für ML-Prg.
+&0C:	Startadresse (High)	nur für ML-Prg.
+&0D:	Startadresse (Bank)	nur für ML-Prg.
+&0E:	Reserviert	&00
+&0F:	Reserviert	&0F
+&10:	Beginn der Daten	

Für die Arbeit mit ASCII-Dateien stellt der Rechner für die Anzahl Dateien markiert durch den MAXFILES-Befehl einen Dateikontrollblock FCB (File Control Block) zur Verfügung. Dieser Block benötigt 57 Byte plus den File-Buffer von 256 Byte, also insgesamt 313 Bytes pro angegebene Datei. Um mit Dateien in ML zu arbeiten, muss man selber einen solchen FCB erstellen.

Um einen FCB zu erstellen, müssen Sie in einem Speicherbereich für jede Datei folgende 313 Bytes reservieren und belegen:

Adresse	Erklärungen
+&00	Filenummer
+&01	Datenquelle z.B. “X”,”S1”. Ohne ":"
+&02	Datenquelle nicht gebrauchte Bytes
+&03	Datenquelle enthalten &00
+&04	Datenquellle
+&05	Modus: &01:INPUT &02:OUTPUT &03:APPEND
+&06	Filebuffer-Zeiger-(Länge der Daten)
+&07	Filebuffer-Lesezeiger: Zeigt auf die Daten, die als nächstes gelesen werden (im INPUT-Modus).
+&08	Status: &01, wenn EOF
+&09-&10	Filename (8 Bytes im ASCII-Format)
+&11-&13	Extension (3 Bytes im ASCII-Format)
+&14	Attribut: &20:nicht geschützt &21:geschützt
+&15-&1E	Reserviert
+&1F-&20	Zeit der ersten Eröffnung. Format: +&20 +&1F Bit: 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 -STUNDE-- ---MINUTE-- -SEK/2--- 4 3 2 1 0 5 4 3 2 1 0 4 3 2 1 0
+&21-&22	Datum der ersten Eröffnung. Format: +&21 +&22 Bit: 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 ----JAHR----- -MONAT- ---TAG--- 6 5 4 3 2 1 0 3 2 1 0 4 3 2 1 0 Bem.:Dem Wert von JAHR ist 1980 dazu zuaddieren.

Adresse	Erklärungen
+&23-&24	erste Clusternummer
+&25-&28	Länge der Datei in Bytes
+&29-&2F	Arbeitsbereich für jede Datenquelle
+&30	aktueller Record
+&31-&32	aktueller Block
+&33-&38	Reserviert
+&39-&139	Filebuffer

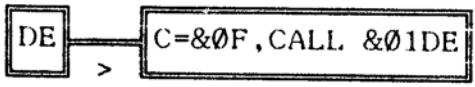
Auf den folgenden Seiten sind die IOCS-Routinen für die Dateibehandlung abgedruckt. Um die Routinen aufzurufen, muss die “IOCS-Nummer” in das C-Register geladen werden und der Befehl CALL &01DE ausgeführt werden. Ist nichts anderes angegeben, muss im DE-Register die Adresse des FCB stehen. A wird mit dem Wert &00 belegt, wenn kein Fehler aufgetreten ist. Ist ein Fehler entstanden, sind im A-Register folgende Bits gesetzt:

A-Register:

Bit7:Die angegebene Datenquelle ist nicht vorhanden.
Bit6:Auf die angegebene Datenquelle konnte nicht korrekt zugegriffen werden.
Bit5:Keine Diskette eingelegt.
Bit4:Diese IOCS-Routine wird nicht unterstützt
Bit3:Anderer Fehler.
Bit2:Keine Daten züm Lesen mehr.
Bit1:Kein Platz mehr zum Schreiben.
Bit0:Filenamen nicht gefunden, oder Directory besitzt zuviele Eintragungen.

Es ist möglich, das mehrere Bits gesetzt sind. Dementsprechend sind mehrere Fehler aufgetreten.

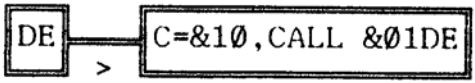
OPEN FILE



DE:Adresse des FCB

Öffnet eine Datei mit den Parametern, angegeben im FCB, welcher an Adresse im DE-Register steht.

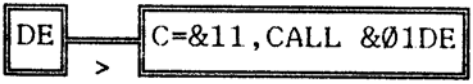
CLOSE FILE



DE:Adresse des FCB

Schliesst die Datei, die durch den FCB bestimmt ist.

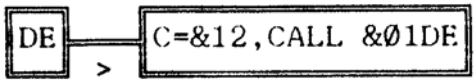
SEARCH FIRST



DE:Adresse des FCB

Sucht nach dem ersten Directory-Eintrag mit demselben Namen wie im FCB angegeben (hier sind "*" und "?" erlaubt) und überträgt den Inhaltes des Directory-Eintrages zur Adresse, welche mit der SETDMA-Routine festgelegt wurde (siehe nächste Seiten). Als Datenquelle darf nicht "CAS" oder "COM" angegeben werden.

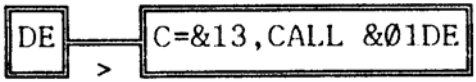
SEARCH NEXT



DE:Adresse des FCB

Sucht den nächsten Directory-Eintrag und macht sonst dasselbe wie SEARCH FIRST.

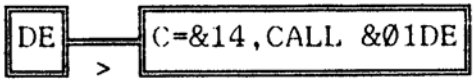
DELETE FILE



DE:Adresse des FCB

Löscht die Datei, die durch FCB bestimmt. ist. Die Jokerzeichen "*" und "?" dürfen benutzt werden. Ein Fehler entsteht, wenn auf "CAS" oder "COM" zugegriffen wird.

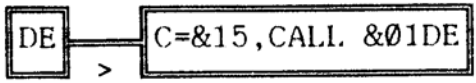
SEQ. READ



DE:Adresse des FCB

Liest ein Rekord (256 Bytes) von der geöffneten Datei an die Speicherstelle, die mit der SETDMA-Routine bestimmt wurde.

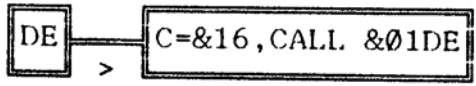
SEQ. WRITE



DE:Adresse des FCB

Schreibt ein Rekord (256 Bytes) von der Speicherstelle an, die durch die SETDMA-Routine bestimmt wurde, in eine geöffnete Datei.

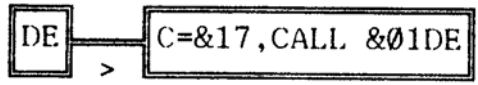
CREATE FILE



DE:Adresse des FCB

Erstellt eine Datei mit den Angaben aus dem FCB. Existiert bereits eine solche Datei, so wird diese gelöscht und die neue Datei, erstellt. Das Datum und die Zeit werden in die entsprechenden Speicherzellen des FCB geschrieben und der Filebufferzeiger, First-Clusternummer, Grösse, Aktueller Record und aktueller Block werden auf Null gesetzt.

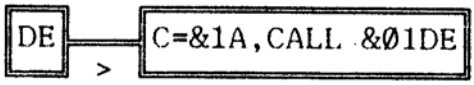
RENAME FILE



DE:Adresse des FCB

Ändert den Namen einer Datei. Dazu muss der neue Namen und die neue Extensions an die Adressen +&29 bis +&33 des FCB geschrieben werden. Die Angabe von “COM” oder “CAS” oder eines bereits vorhandenen Namens führt zu einer Fehlermeldung.

SETDMA



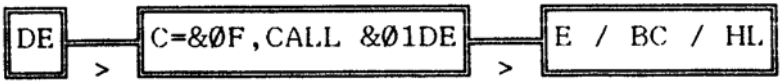
DE:Disktransfer-Adresse

Setzt die Disktransferadresse an die Adresse, die im DE-Register steht.

SET ATTRIBUTE

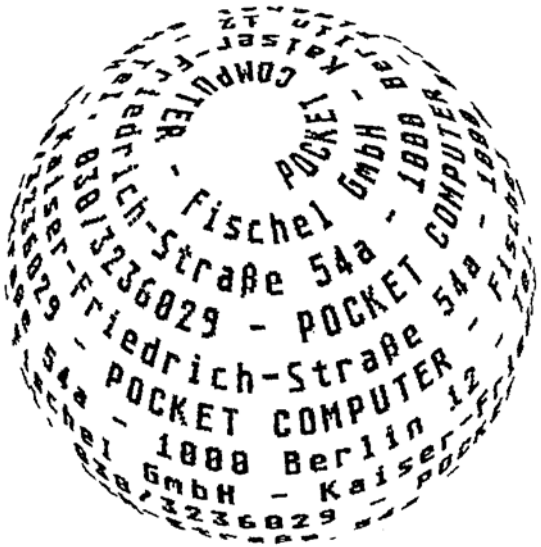
DE:Adresse des FCB

Setzt das in der Adresse DE+&14 angegebene Attribut.



DE:Adresse des FCB E :Anzahl Sektoren pro Clstr
BC:Sektorgrösse
HL:Anzahl leerer Clusters

Liest Informationen über Speichermedium und Datei. Angabe von “COM” oder “CAS” fährt zu einer Fehlermeldung.

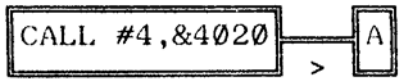


P R I N T E R

Der Aufruf von IOCS die den Printer / Plotter betreffen, ist wesentlich komplizierter als die zuvor, betrachteten IOCS-Routinen, da das Printer-ROM in der Bank #4 liegt. Wurde eine folgernder Routinen durchgeführt , so müssen folgende Routinen aufgerufen werden, die den Drucker ausschalten und den Interrupt freigeben, da sonst u.U. der Drucker unnötig ,Strom verbraucht oder die Tastatur nicht mehr richtig funktioniert:

Bank #4, CALL &4029
Bank #4, CALL &6777

Eine Routine in einer anderen Bank ruft man auf, indem man die Banknummer im A'-Register und die Adresse im HL'-Register (Achtung: sekundäre Register !!!) speichert und dann die Routine CALL &019F (in Bank #0) aufruft !!
Folgende 2 Routinen rufen Sie einfach mit einem CALL-Befehl (in Bank #4) auf:

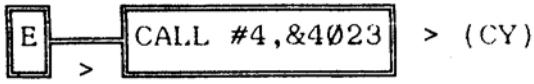


A :Printerstatus

Überprüft den Status des Druckers. Dabei bedeuten die einzelnen Bits des A-Registers:

Bit0:Die Printerer-Batterie ist leer
Bit1:Der Druckkopf ist in Stiftwechselposition
Bit4:Printer noch nicht initialisiert.

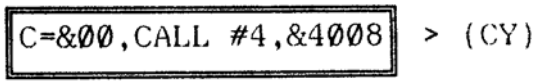
Alle Register ,werden zerstört.



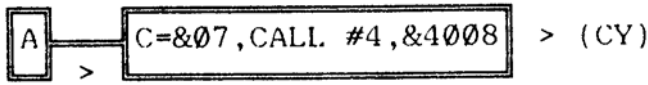
E :ASCII-Zeichen

Sendet das Zeichen, das im E-Register steht zum Printer. Ist CY=1 beim Verlassen der Routine, so wurde die BREAK-Taste gedrückt (A=&00) oder ein anderer Fehler (Fehlercode im A-Register) ist aufgetaucht. Zerstört alle Register.

Die nun folgenden IOCS-Routinen müssen mit der IOCS-Nummer gestartet werden. Laden Sie diese ins C-Register und starten Sie dann CALL &4008 in Bank #4. Alle Register werden durch diese Routinen zerstört. Auch hier gilt, die beiden Routinen aufzurufen, nachdem eine der folgenden IOCS-Routinen erfolgt ist. Unterproqramme für Grafik funktionieren nur dann, wenn der Drucker auch im Grafik-Modus ist. Die meisten Routinen liefern bei einem entstandenen Fehler ein gesetztes Carry-Flag (CY=1) mit dem Fehlercode (wie in Basic, ausser A=&00 heisst: BREAK) im A-Register.

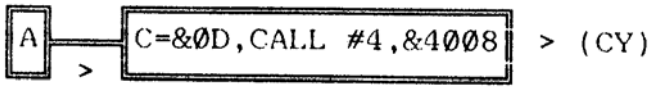


Initialisiert den Printer-Arbeitsbereich.



A :Position (Ø-79)

Setzt im Textmodus den linken Textrand.

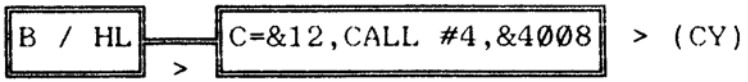


A : = &FF Stift auf Papier
<> &FF Stift weg vom Papier

Setzt den Stift auf das Papier oder nimmt ihn weg.

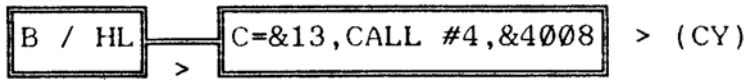
Die nächsten 4 Routinen sind für die Bewegung des Stiftes verantwortlich. Als Parameter muss man in HL die Adresse angegeben, wo die Koordinaten-Daten gespeichert sind. In B muss stehen, wieviele Koordinaten es sind. Der Stift bewegt sich von X0/Y0 nach X1/Y1 , X2/Y2 bis zu X5/Y5. Bei relativen Bewegungen wird der Punkt X0/Y0 als Startpunkt gewertet (also absolute Koordinaten) und alle weiteren Koordinaten als relative Bewegung in die entsprechende Richtung. Der Speicherbereich (HL) muss wiefolgt gegliedert sein:

Adresse	Inhalt	Wert in B
HL	XØ (Low)	
HL+ 1	XØ (High)	1
HL+ 2	YØ (Low)	
HL+ 3	YØ (High)	
HL+ 4	X1 (Low)	
HL+ 5	X1 (High)	2
HL+ 6	Y1 (Low)	
HL+ 7	Y1 (High)	
.	.	.
.	.	.
.	.	.
HL+2Ø	X5 (Low)	
HL+21	X5 (High)	6
HL+22	Y5 (Low)	
HL+23	Y5 (High)	



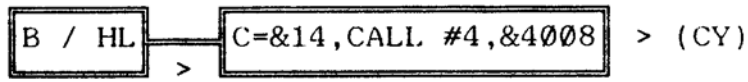
B,HL gem. Beschreibung S.95

Bewegt Stift absolut ohne zu zeichnen.



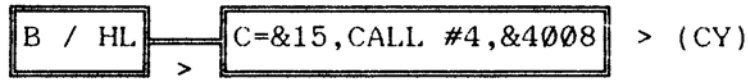
B,HL gem. Beschreibung S.95

Bewegt Stift relativ ohne zu zeichnen.



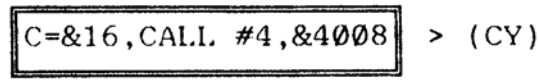
B,HL gem. Beschreibung S.95

Bewegt Stift absolut zeichnend.

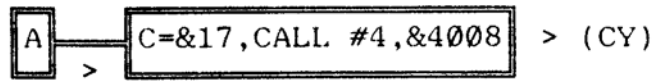


B,HL gem. Beschreibung S.95

Bewegt Stift relativ zeichnend.



Führt den Druckertest aus.



A :Position (Ø-79)

Führt Stift (im Textmode) zur Position in A.

```
C=&18,CALL #4,&4008
```

Schaltet die Stromzufuhr aus.

```
A ──> C=&1A,CALL #4,&4008 > (CY)
```

A :Verschiebung (&00-&FF)

Fährt Kopf um den Wert im A-Register nach oben. Der Kopf bleibt u.U. am oberen Ende stehen.

```
A ──> C=&1B,CALL #4,&4008 > (CY)
```

A :Verschiebung (&00-&FF)

Fährt Kopf um den Wert im A-Register nach unten. Kopf bleibt u.U. am unteren Ende stehen.

```
A ──> C=&1C,CALL #4,&4008 > (CY)
```

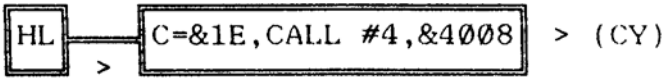
A :Verschiebung (&00-&FF)

Fährt Kopf um den Wert im A-Register nach links. Kopf bleibt u.U. am linken Ende stehen.

```
A ──> C=&1D,CALL #4,&4008 > (CY)
```

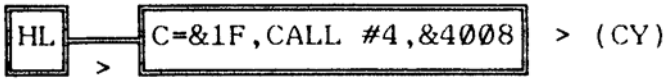
A :Verschiebung (&00-&FF)

Fährt Kopf um den Wert im A-Register nach rechts. Er bleibt u.U. am rechten Ende stehen.



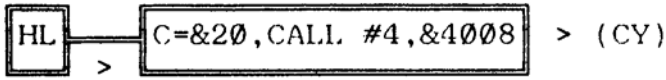
HL:Verschiebung (Ø-2047)

Fährt den Kopf in Grafikpunkten nach oben.



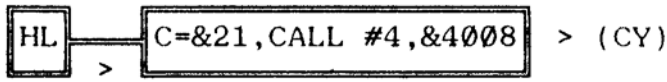
HL:Verschiebung (Ø-2047)

Fährt den Kopf in Grafikpunkten nach unten.



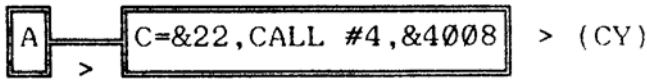
HL:Verschiebung (Ø-2047)

Fährt den Kopf in Grafikpunkten nach links



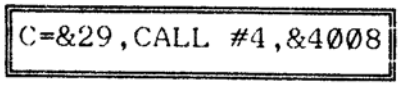
HL:Verschiebung (Ø-2047)

Fährt den Kopf in Grafikpunkten nach rechts.

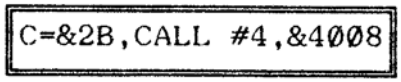


A :&ØØ-&Ø1

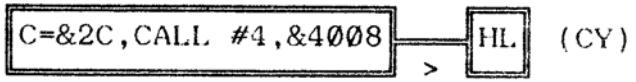
Fährt den Kopf zur Stiftwechselposition (Bei A=&00), dreht Zylinder (Bei A=&00 und Position bereits dort) oder fährt zur ursprünglichen Position zurück (Bei A=&01)



Fährt Kopf an linkes Ende (Carriage Return).

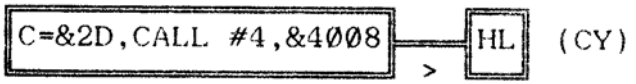


Fährt Kopf an linkes Ende mit Line-Feed.



HL:Verbleibende Anzahl

Überprüft, wieviele Linien in negativer Y-Richtung übrigbleiben (HL=&FFFF bei Rollpapier)



HL:Verbleibende Anzahl

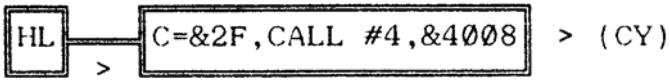
Überprüft, wieviele Linien in positiver Y-Richtung übrigbleiben (HL=&FFFF bei Rollpapier)

Folgende 2 Routinen sind für das Zeichnen von Rechtecken verantwortlich Dabei muss das HLRegister als Parameter übergeben werden. Dieses zeigt auf eine Adresse wo die Koordinaten des Anfangspunktes und des Diagonalpunktes gespeichert sind Wie sie, abgelegt müssen entnehmen Sie folgender Tabelle:

Adresse	Inhalt
HL	Start X-Koordinate (Lowbyte)
HL+1	Start X-Koordinate (Highbyte)
HL+2	Start Y-Koordinate (Lowbyte)

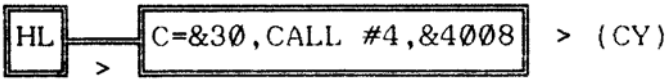
Fortsetzung Tabelle

HL+3	Start Y-Koordinate (Highbyte)
HL+4	Diagonal X-Koordinate (Lowbyte)
HL+5	Diagonal X-Koordinate (Highbyte)
HL+6	Diagonal Y-Koordinate (Lowbyte)
HL+7	Diagonal Y-Koordinate (Highbyte



HL:Wie auf S.99 beschrieben

Zeichnet Rechteck mit. absoluten Koordinaten



HL:Wie auf S.99 beschrieben

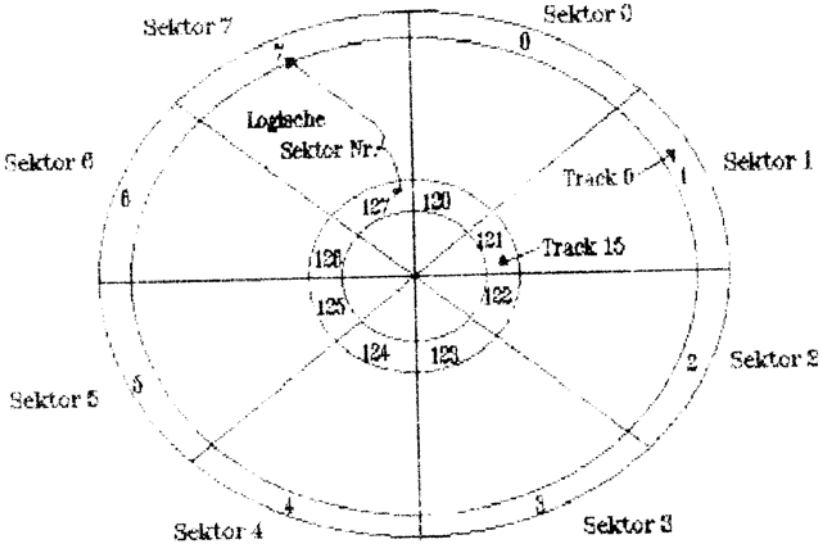
Zeichnet Rechteck mit relativen Koordinaten

Initialisiert die Printer-Hardware. Diese Routine muss nach einer erfolgten IOCS-NR. &01-Routine ausgeführt werden.

Eine Diskette für den PC-1600, hat folgende Daten:

Anzahl Tracks pro Seite:	16
Anzahl Sektoren pro Track:	8
Anzahl Bytes pro Sektor:	512
Anzahl Sektoren pro FAT:	1
Anzahl FATS:	2
Anzahl logischer Sektoren pro Cluster:	1
Max. Anzahl Files pro Seite:	48

Jeder Sektor hat eine logische Sektornummer von 0-127. Der Sektor 0 von Track 0 trägt- die log. Sektornummer 0, der Sektor 7 von Track 15 die log. Sektornr. 127:



Die ersten sechs logischen Sektoren sind fürbestimmte Zwecke reserviert:

Sektor 0 :Boot-Sektor: Beim Einschalten wird getestet, ob hier ein Auto-BootProgramm steht.

Sektor 1 :FAT: Dies ist eine Karte, die Ihnen sagt, welche Dateien welche Sektoren auf der Diskette belegen. Files sind in verschiedene Clusters aufgeteilt. Jedes Cluster ist verwaltet von einem Byte im FAT. erste Byte lautet &F2 (Format ID-Code. Die weiteren Bytes enthalten die übersichtsdaten des Datenbereichs. Diese Karte beinhaltet 122 Bytes und jedes dieser Bytes repräsentiert Informationen über einen bestimmten Cluster. Sie sind in der Folge von Cluster 1 bis Cluster 122 geschrieben (dabei belegt Cluster 1 den logischen Sektor 6 und der Cluster 122 den logischen Sektor 127). Die Cluster Information bedeutet folgendes:

&00	: Cluster ist leer
&01-&7A	: Cluster gebraucht. Wert zeigt auf nächsten Cluster.
&F0	: Letzter Cluster des Files

Sektor 2 :FAT (Sicherheitskopie)

Die logischen Sektoren 3 bis 5 beinhalten das Directory (Die Liste aller Files auf der Disk). Jede Dateieintragung benötigt 32 Bytes, 3 Sektoren stehen dafür zur Verfügung, d.h. es sind max. 512*3/32=48 Files eintragbar.

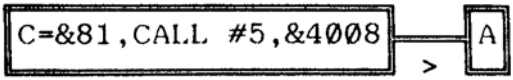
Diese 32 Bytes pro Dateien beschreiben folgendes:

&00-&07 :Filenamen (8 Zeichen). Nicht benötigte Bytes werden mit &20 aufgefüllt.
&08-&0A :Extension (3 Zeichen).
&0B :Attribut (0=Lesen/Schreiben,1=nur lesen)
&0C-&0F :Reserviert (immer &00)
&16-&17 :Zeit des letzten SAVE (Format Siehe s.87).
&18-&19 :Datum des letzten SAVE (S.87).
&1A :Erste Cluster-Nr. der Datei.
&1B :Immer &00
&1C-&1F :Dateigrösse (Low > High)

Die nachfolgenden IOCS-Routinen müssen, ähnlich wie die für den Plotter, umständlich aufgerufen werden. Die IOCS-NR. muss ins C-Register geladen werden und dann die Routine ab Adresse &4008 in Bank #5 aufgerufen werden. Laden Sie zu diesem Zwecke das A'-Register mit dem Wert &05 und das HL'-Register mit dem Wert &4008 (Achtung: Sekundäre Register !) und führen Sie ein CALL &019F durch. Somit wird ein CALL &4008 in Bank #5 durchgeführt.
Bei allen Routinien, ausser der ersten, wird das A-Register als Parameter verlangt. Dieses beschreibt das Laufwerk. Belegen Sie A für das entsprechende Laufwerk wiefolgt:
A=&01 für “X:” A=&02 für "Y:" Ist beim Aufruf ein Fehler entstanden, wird CY=1 und im A-Register steht der Fehlercode, dabei bedeuten:

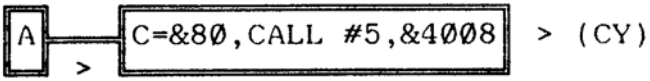
Bit0 :Laufwerk nicht bereit
Bit1+2 :Fehler beim Zugriff
Bit3 :Verlangte Sektor nicht gefunden
Bit4 :Lesekopf-Fehler
Bit5 :Schreibschutz in Funktion
Bit6 :Schwache Batterie
Bit7 :Andere Fehler, wie: Laufwerk ist leer

Bei allen folgenden IOCS-Routinen (Ausser bei der ersten) werden alle Register - ausser die Sekundärregister - zerstört.



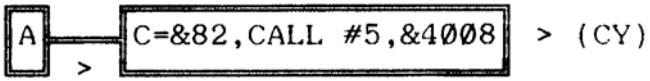
A :Anzahl Laufwerke

Liest die Anzahl vorhandener Laufwerke.



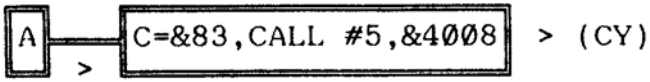
A :Laufwerknr.

Initialisiert das Laufwerk (Dies formattiert aber nicht die Diskette, dies schaltet nur das Laufwerk ein !)



A :Laufwerknr.

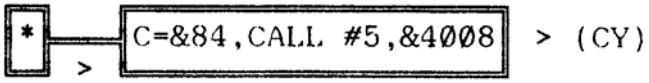
Fährt den Lesekopf zum Track 0.



A :Laufwerknr.

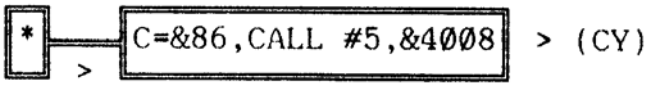
Formattiert die Diskette (Datenverlust !)

Ein "*" als Übergabeparameter
bedeutet:
A :Laufwerksnr.
B : Anzahl Sektoren
D :Tracknr. des ersten Sektor (0-15)
E :Sektornr. des ersten Sektor (0-7)
HL:Startadresse im Speicher

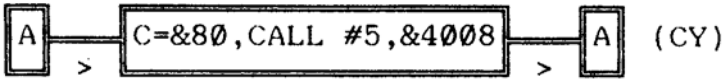


Liest sequentiell Daten von den bezeichneten Sektoren in den Speicher (ab HL).

Schreibt sequentiell Daten vom Speicher (ab HL) in die bezeichneten Sektoren.



Überprüft den Speicherbereich (ab HL) mit dem Inhalt der bezeichneten Sektoren.



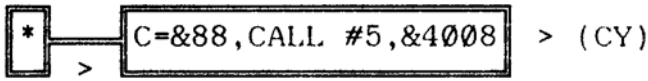
A :Laufwerknr. A :Status

Liest Status vom Laufwerk und Diskette. CY=1:Laufwerk nicht vorhanden.
CY=0:A= Status. Dabei bedeuten:

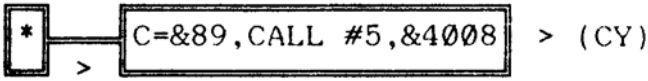
Bit0:1, wenn Laufwerk läuft
Bit1:immer 0

Bit2:1, wenn Laufwerk geöffnet wurde
Bit3:1, wenn Diskette eingesetzt
Bit4: immer 0
Bit5: immer 0
Bit6:1, wenn geschützt. Gilt nur, wenn Bit3=1 und Bit7=0.
Bit7:1, wenn Laufwerk nicht bereit.

Ein "*" in den folgenden 2 IOCS-Routinen als Übergabeparameter bedeuten dasselbe wie auf S. 105 beschrieben, ausser dass B nicht übergeben wird.



Liest die ersten 256 Bytes des bezeichneten Sektors in den Speicher (Ab HL).

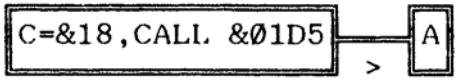


Vergleicht die ersten 256 Bytes des bezeichneten Sektors mit dem Speicherbereich ab HL-Register.

Ist das 4. Byte im Boot-Sektor (Loq.Sektor 0) gleich &C3, beinhaltet der Boot-Sektor ein Auto-Startprogramm, welches in diesem Falle ab dem 33. Byte in diesem Sektor steht und ausgeführt wird.

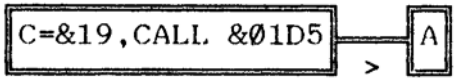
ANALOGPORT

Um eine solche Routine aufzurufen, muss man die IOCS-NR. ins C-Register lesen und ein CALL &01D5 durchführen.



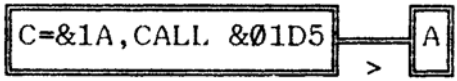
A :Digitaler Wert

Liest die Batteriespannung der PC-1600 Batterien. Liegt dieser Wert unterhalb &AF, so wird das Signet "BATT" zum Leuchten gebracht. Dieser erlischt erst, wenn dieser Wert wieder grösser als &BE wird.



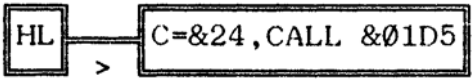
A :Digitaler Wert

Liest die Spannung am analogen Port.



A :Digitaler Wert

Liest die Spannung des Akku im CE-1600P.



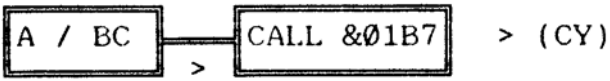
H :Obere Limite
L :untere Limite

Setzt die Limite für den Analogen Port. H bezeichnet die obere Limite, L die untere. Beides müssen digitale Werte sein. Das AF-Register wird zerstört.

BEEP

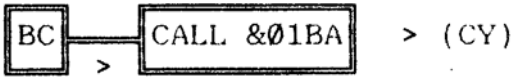
A :Tonhöhe (&ØØ-&FF)
BC:Tondauer (&ØØØØ-&FEFF)
DE:Anzahl Töne (&ØØØØ-&FFFF)

Erzeugt Beeps mit den angegebenen Daten. Die Frequenz des durch A bestimmten Tones, lässt sich so berechnen: f=1300000/(166+22*A)(Hz)
Um die effektive Dauer in Sekunden eines Tones zu berechnen, bedienen wir uns folgender Formel:
Dauer = BC*(166+22*A)/1300000 [Sek]



A :Tonhöhe (&ØØ-&FF)
BC:Tondauer (&ØØØØ-&FEFF)

Wie die vorhergehende Routine, erzeugt aber nur einen BEEP.



BC:Dauer (&ØØØØ-&FEFF)

Wartet eine bestimmte Dauer (BC/64=,Sekunden)

Erlaubt oder verbietet Beeps. Dazu muss das Bit0 der Adresse (F86B) =1 sein, um Beep zu verbieten.

Nachfolgende Liste hilft dem Programmierer, da sie eine Vergleichsliste zwischen 3 Zahlensystemen und dem ASCII-Zeichensatz des PC-1600 bietet.

Dez.	Hex.	Bin.	Asc.	Dez.	Hex.	Bin.	Asc.
000	00	00000000		026	1A	00011010	
001	01	00000001	Shft	027	1B	00011011	DEF
002	02	00000010	SML	028	1C	00011100	
003	03	00000011	CTRL	029	1D	00011101	
004	04	00000100	KBII	030	1E	00011110	
005	05	00000101	BS	031	1F	00011111	MODE
006	06	00000110		032	20	00100000	leer
007	07	00000111		033	21	00100001	!
008	08	00001000	CuLe	034	22	00100010	"
009	09	00001001	CH f	035	23	00100011	#
010	0A	00001010	CuDo	036	24	00100100	$
011	0B	00001011	CuUp	037	25	00100101	%
012	0C	00001100	CuRi	038	26	00100110	&
013	0D	00001101	Enter	039	27	00100111	'
014	0E	00001110	ON	040	28	00101000	(
015	0F	00001111	OFF	041	29	00101001	)
016	10	00010000		042	2A	00101010	*
017	11	00010001	f1	043	2B	00101011	+
018	12	00010010	f2	044	2C	00101100	,
019	13	00010011	f3	045	2D	00101101	-
020	14	00010100	f4	046	2E	00101110	.
021	15	00010101	f5	047	2F	00101111	/
022	16	00010110	f6	048	30	00110000	0
023	17	00010111		049	31	00110001	1
024	18	00011000	CL	050	32	00110010	2
025	19	00011001	RCL	051	33	00110011	3

Dez.	Hex.	Bin.	Asc.	Dez.	Hex.	Bin.	Asc.
052	34	00110100	4	084	54	01010100	T
053	35	00110101	5	085	55	01010101	U
054	36	00110110	6	086	56	01010110	V
055	37	00110111	7	087	57	01010111	W
056	38	00111000	8	088	58	01011000	X
057	39	00111001	9	089	59	01011001	Y
058	3A	00111010	:	090	5A	01011010	Z
059	3B	00111011	;	091	5B	01011011	[
060	3C	00111100	<	092	5C	01011100	\
061	3D	00111101	=	093	5D	01011101	]
062	3E	00111110	>	094	5E	01011110	^
063	3F	00111111	?	095	5F	01011111	_
064	40	01000000	@	096	60	01100000	~
065	41	01000001	A	097	61	01100001	a
066	42	01000010	B	098	62	01100010	b
067	43	01000011	C	099	63	01100011	c
068	44	01000100	D	100	64	01100100	d
069	45	01000101	E	101	65	01100101	e
070	46	01000110	F	102	66	01100110	f
071	47	01000111	G	103	67	01100111	g
072	48	01001000	H	104	68	01101000	h
073	49	01001001	I	105	69	01101001	i
074	4A	01001010	J	106	6A	01101010	j
075	4B	01001011	K	107	6B	01101011	k
076	4C	01001100	L	108	6C	01101100	l
077	4D	01001101	M	109	6D	01101101	m
078	4E	01001110	N	110	6E	01101110	n
079	4F	01001111	O	111	6F	01101111	o
080	50	01010000	P	112	70	01110000	p
081	51	01010001	Q	113	71	01110001	q
082	52	01010010	R	114	72	01110010	r
083	53	01010011	S	115	73	01110011	s

Dez.	Hex.	Bin.	Asc.	Dez.	Hex.	Bin.	Asc.
116	74	Ø1110100	t	148	94	10010100	ö
117	75	Ø1110101	u	149	95	10010101	ò
118	76	Ø1110110	v	150	96	10010110	û
119	77	Ø1110111	w	151	97	10010111	ù
120	78	Ø1111000	x	152	98	10011000	ÿ
121	79	Ø1111001	y	153	99	10011001	Ö
122	7A	Ø1111010	z	154	9A	10011010	Ü
123	7B	Ø1111011	{	155	9B	10011011	¢
124	7C	Ø1111100	¦	156	9C	10011100	#
125	7D	Ø1111101	}	157	9D	10011101	¥
126	7E	Ø1111110	~	158	9E	10011110	P
127	7F	Ø1111111	▉	159	9F	10011111	f
128	80	10000000	Ç	160	A0	10100000	á
129	81	10000001	ü	161	A1	10100001	í
130	82	10000010	é	162	A2	10100010	ó
131	83	10000011	â	163	A3	10100011	ú
132	84	10000100	ä	164	A4	10100100	ñ
133	85	10000101	à	165	A5	10100101	Ñ
134	86	10000110	a	166	A6	10100110	a
135	87	10000111	ç	167	A7	10100111	o
136	88	10001000	ê	168	A8	10101000	¿
137	89	10001001	ë	169	A9	10101001	⌐
138	8A	10001010	è	170	AA	10101010	¬
139	8B	10001011	ï	171	AB	10101011	½
140	8C	10001100	î	172	AC	10101100	¼
141	8D	10001101	ì	173	AD	10101101	¡
142	8E	10001110	Ä	174	AE	10101110	«
143	8F	10001111	Å	175	AF	10101111	»
144	90	10010000	É	176	B0	10110000	▒
145	91	10010001	æ	177	B1	10110001	▓
146	92	10010010	Æ	178	B2	10110010	▓
147	93	10010011	ô	179	B3	10110011	│

Dez.	Hex.	Bin.	Asc.	Dez.	Hex.	Bin.	Asc.
180	B4	10110100	┤	212	D4	11010100	╘
181	B5	10110101	╡	213	D5	11010101	╒
182	B6	10110110	╢	214	D6	11010110	╓
183	B7	10110111	╖	215	D7	11010111	╫
184	B8	10111000	╕	216	D8	11011000	╪
185	B9	10111001	╣	217	D9	11011001	┘
186	BA	10111010	║	218	DA	11011010	┌
187	BB	10111011	╗	219	DB	11011011	▉
188	BC	10111100	╝	220	DC	11011100	▄
189	BD	10111101	╜	221	DD	11011101	▌
190	BE	10111110	╛	222	DE	11011110	▐
191	BF	10111111	┐	223	DF	11011111	▀
192	C0	11000000	└	224	E0	11100000	α
193	C1	11000001	┴	225	E1	11100001	β
194	C2	11000010	┬	226	E2	11100010	Γ
195	C3	11000011	├	227	E3	11100011	π
196	C4	11000100	─	228	E4	11100100	Σ
197	C5	11000101	┼	229	E5	11100101	σ
198	C6	11000110	╞	230	E6	11100110	µ
199	C7	11000111	╟	231	E7	11100111	τ
200	C8	11001000	╚	232	E8	11101000	Φ
201	C9	11001001	╔	233	E9	11101001	Θ
202	CA	11001010	╩	234	EA	11101010	Ω
203	CB	11001011	╦	235	EB	11101011	δ
204	CC	11001100	╠	236	EC	11101100	∞
205	CD	11001101	═	237	ED	11101101	ø
206	CE	11001110	╬	238	EE	11101110	ε
207	CF	11001111	╧	239	EF	11101111	∩
208	D0	11010000	╨	240	F0	11110000	≡
209	D1	11010001	╤	241	F1	11110001	±
210	D2	11010010	╥	242	F2	11110010	≥
211	D3	11010011	╙	243	F3	11110011	≤

Dez.	Hex.	Bin.	Asc.	Dez.	Hex.	Bin.	Asc.
244	F4	11110100	⌠	250	FA	11111010	·
245	F5	11110101	⌡	251	FB	11111011	√
246	F6	11110110	÷	252	FC	11111100	η
247	F7	11110111	≈	253	FD	11111101	²
248	F8	11111000	°	254	FE	11111110	■
249	F9	11111001	·	255	FF	11111111	

Die Zeichen die in der Kolonne ASC stehen, kommen zum Vorschein bei PRINT CHR$(Dez.). Dies gilt für Dez > 31. Die Zeichen < 32 sind Werte von Tasten, welche Sie mit der INKEY$-Funktion erhalten.

Erklärungen von Abkürzungen:

Shft	SHIFT	SML	SMALL
CTRL	Control	KBII	2.Tastatur
BS	Korrektur	CuLe	Pfeil links
CH f	I ->II ->III	CuDo	Pfeil unten
CuUp	Pfeil auf	CuRi	Pfeil recht
f1-f6	Funktionstasten	RCL	Recall

* * * *

ADC A,(HL)	8E	AND A	A7	BIT 3,A	CB5F
ADC A,(IX+zz)	DD8Ezz	AND B	A0	BIT 3,B	CB58
ADC A,(IY+zz)	FD8Ezz	AND C	A1	BIT 3,C	CB59
ADC A,A	8F	AND D	A2	BIT 3,D	CB5A
ADC A,B	88	AND E	A3	BIT 3,E	CB5B
ADC A,C	89	AND H	A4	BIT 3,H	CB5C
ADC A,D	8A	AND L	A5	BIT 3,L	CB5D
ADC A,E	8B	AND xx	E6xx	BIT 4,(HL)	CB66
ADC A,H	8C	BIT 0,(HL)	CB46	BIT 4,(IX+zz)	DDCBzz66
ADC A,L	8D	BIT 0,(IX+zz)	DDCBzz46	BIT 4,(IY+zz)	FDCBzz66
ADC A,xx	CExx	BIT 0,(IY+zz)	FDCBzz46	BIT 4,A	CB67
ADC HL,BC	ED4A	BIT 0,A	CB47	BIT 4,B	CB60
ADC HL,DE	ED5A	BIT 0,B	CB40	BIT 4,C	CB61
ADC HL,HL	ED6A	BIT 0,C	CB41	BIT 4,D	CB62
ADC HL,SP	ED7A	BIT 0,D	CB42	BIT 4,E	CB63
ADD A,(HL)	86	BIT 0,E	CB43	BIT 4,H	CB64
ADD A,(IX+zz)	DD86zz	BIT 0,H	CB44	BIT 4,L	CB65
ADD A,(IY+zz)	FD86zz	BIT 0,L	CB45	BIT 5,(HL)	CB6E
ADD A,A	87	BIT 1,(HL)	CB4E	BIT 5,(IX+zz)	DDCBzz6E
ADD A,B	80	BIT 1,(IX+zz)	DDCBzz4E	BIT 5,(IY+zz)	FDCBzz6E
ADD A,C	81	BIT 1,(IY+zz)	FDCBzz4E	BIT 5,A	CB6F
ADD A,D	82	BIT 1,A	CB4F	BIT 5,B	CB68
ADD A,E	83	BIT 1,B	CB48	BIT 5,C	CB69
ADD A,H	84	BIT 1,C	CB49	BIT 5,D	CB6A
ADD A,L	85	BIT 1,D	CB4A	BIT 5,E	CB6B
ADD A,xx	C6xx	BIT 1,E	CB4B	BIT 5,H	CB6C
ADD HL,BC	09	BIT 1,H	CB4C	BIT 5,L	CB6D
ADD HL,DE	19	BIT 1,L	CB4D	BIT 6,(HL)	CB76
ADD HL,HL	29	BIT 2,(HL)	CB56	BIT 6,(IX+zz)	DDCBzz76
ADD HL,SP	39	BIT 2,(IX+zz)	DDCBzz56	BIT 6,(IY+zz)	FDCBzz76
ADD IX,BC	DD09	BIT 2,(IY+zz)	FDCBzz56	BIT 6,A	CB77
ADD IX,DE	DD19	BIT 2,A	CB57	BIT 6,B	CB70
ADD IX,IX	DD29	BIT 2,B	CB50	BIT 6,C	CB71
ADD IX,SP	DD39	BIT 2,C	CB51	BIT 6,D	CB72
ADD IY,BC	FD09	BIT 2,D	CB52	BIT 6,E	CB73
ADD IY,DE	FD19	BIT 2,E	CB53	BIT 6,H	CB74
ADD IY,IY	FD29	BIT 2,H	CB54	BIT 6,L	CB75
ADD IY,SP	FD39	BIT 2,L	CB55	BIT 7,(HL)	CB7E
AND (HL)	A6	BIT 3,(HL)	CB5E	BIT 7,(IX+zz)	DDCBzz7E
AND (IX+zz)	DDA6zz	BIT 3,(IX+zz)	DDCBzz5E	BIT 7,(IY+zz)	FDCBzz7E
AND (IY+zz)	FDA6zz	BIT 3,(IY+zz)	FDCBzz5E	BIT 7,A	CB7F

BIT 7,B	CB78	DEC DE	1B	INC IX	DD23
BIT 7,C	CB79	DEC E	1D	INC IY	FD23
BIT 7,D	CB7A	DEC H	25	INC L	2C
BIT 7,E	CB7B	DEC HL	2B	INC SP	33
BIT 7,H	CB7C	DEC IX	DD2B	IND	EDAA
BIT 7,L	CB7D	DEC IY	FD2B	INDR	EDBA
CALL C,xxyy	DCyyxx	DEC L	2D	INI	EDA2
CALL M,xxyy	FCyyxx	DEC SP	3B	INIR	EDB2
CALL NC,xxyy	D4yyxx	DI	F3	JP (HL)	E9
CALL NZ,xxyy	C4yyxx	DJNZ tt	10tt	JP (IX)	DDE9
CALL P,xxyy	F4yyxx	EI	FB	JP (IY)	FDE9
CALL PE,xxyy	ECyyxx	EX (SP),HL	E3	JP C,xxyy	DAyyxx
CALL PO,xxyy	E4yyxx	EX (SP),IX	DDE3	JP M,xxyy	FAyyxx
CALL Z,xxyy	CCyyxx	EX (SP),IY	FDE3	JP NC,xxyy	D2yyxx
CALL xxyy	CDyyxx	EX AF,AF'	08	JP NZ,xxyy	C2yyxx
CCF	3F	EX DE,HL	EB	JP P,xxyy	F2yyxx
CP (HL)	BE	EXX	D9	JP PE,xxyy	EAyyxx
CP (IX+zz)	DDBEzz	HALT	76	JP PO,xxyy	E2yyxx
CP (IY+zz)	DFBEzz	IM 0	ED46	JP Z,xxyy	CAyyxx
CP A	BF	IM 1	ED56	JP xxyy	C3yyxx
CP B	B8	IM 2	ED5E	JR C,tt	38tt
CP C	B9	IN A,(C)	ED78	JR NC,tt	30tt
CP D	BA	IN A,(xx)	DBxx	JR NZ,tt	20tt
CP E	BB	IN B,(C)	ED40	JR Z,tt	28tt
CP H	BC	IN C,(C)	ED48	JR tt	18tt
CP L	BD	IN D,(C)	ED50	LD (BC),A	02
CP xx	FExx	IN E,(C)	ED58	LD (DE),A	12
CPD	EDA9	IN H,(C)	ED60	LD (HL),A	77
CPDR	EDB9	IN L,(C)	ED68	LD (HL),B	70
CPI	EDA1	INC (HL)	34	LD (HL),C	71
CPIR	EDB1	INC (IX+zz)	DD34zz	LD (HL),D	72
CPL	2F	INC (IY+zz)	FD34zz	LD (HL),E	73
DAA	27	INC A	3C	LD (HL),H	74
DEC (HL)	35	INC B	04	LD (HL),L	75
DEC (IX+zz)	DD35zz	INC BC	03	LD (HL),xx	36xx
DEC (IY+zz)	FD35zz	INC C	0C	LD (IX+zz),A	DD77zz
DEC A	3D	INC D	14	LD (IX+zz),B	DD70zz
DEC B	05	INC DE	13	LD (IX+zz),C	DD71zz
DEC BC	0B	INC E	1C	LD (IX+zz),D	DD72zz
DEC C	0D	INC H	24	LD (IX+zz),E	DD73zz
DEC D	15	INC HL	23	LD (IX+zz),H	DD74zz

LD (IX+zz),L	DD75zz	LD B,E	43	LD H,(HL)	66
LD (IX+zz),xx	DD36zzxx	LD B,H	44	LD H,(IX+zz)	DD66zz
LD (IY+zz),A	FD77zz	LD B,L	45	LD H,(IY+zz)	FD66zz
LD (IY+zz),B	FD70zz	LD B,xx	06xx	LD H,A	67
LD (IY+zz),C	FD71zz	LD BC,(xxyy)	ED4Byyxx	LD H,B	60
LD (IY+zz),D	FD72zz	LD BC,xxyy	01yyxx	LD H,C	61
LD (IY+zz),E	FD73zz	LD C,(HL)	4E	LD H,D	62
LD (IY+zz),H	FD74zz	LD C,(IX+zz)	DD4Ezz	LD H,E	63
LD (IY+zz),L	FD75zz	LD C,(IY+zz)	FD4Ezz	LD H,H	64
LD (IY+zz),xx	FD36zzxx	LD C,A	4F	LD H,L	65
LD (xxyy),A	32yyxx	LD C,B	48	LD H,xx	26xx
LD (xxyy),BC	ED43yyxx	LD C,C	49	LD HL,(xxyy)	2Ayyxx
LD (xxyy),DE	ED53yyxx	LD C,D	4A	LD HL,(xxyy)	ED6Byyxx
LD (xxyy),HL	22yyxx	LD C,E	4B	LD HL,xxyy	21yyxx
LD (xxyy),HL	ED63yyxx	LD C,H	4C	LD I,A	ED47
LD (xxyy),IX	FF22yyxx	LD C,L	4D	LD IX,(xxyy)	DD2Ayyxx
LD (xxyy),IY	FD22yyxx	LD C,xx	0Exx	LD IX,xxyy	DD21yyxx
LD (xxyy),SP	ED73yyxx	LD D,(HL)	56	LD IY,(xxyy)	FD2Ayyxx
LD A,(BC)	0A	LD D,(IX+zz)	DD56zz	LD IY,xxyy	FD21yyxx
LD A,(DE)	1A	LD D,(IY+zz)	FD56zz	LD L,(HL)	6E
LD A,(HL)	7E	LD D,A	57	LD L,(IX+zz)	DD6Ezz
LD A,(IX+zz)	DD7Ezz	LD D,B	50	LD L,(IY+zz)	FD6Ezz
LD A,(IY+zz)	FD7Ezz	LD D,C	51	LD L,A	6F
LD A,(xxyy)	3Ayyxx	LD D,D	52	LD L,B	68
LD A,A	7F	LD D,E	53	LD L,C	69
LD A,B	78	LD D,H	54	LD L,D	6A
LD A,C	79	LD D,L	55	LD L,E	6B
LD A,D	7A	LD D,xx	16xx	LD L,H	6C
LD A,E	7B	LD DE,(xxyy)	ED5Byyxx	LD L,L	6D
LD A,H	7C	LD DE,xxyy	11yyxx	LD L,xx	2Exx
LD A,I	ED57	LD E,(HL)	5E	LD R,A	ED4F
LD A,L	7D	LD E,(IX+zz)	DD5Ezz	LD SP,(xxyy)	ED7Byyxx
LD A,R	ED5F	LD E,(IY+zz)	FD5Ezz	LD SP,HL	F9
LD A,xx	3Exx	LD E,A	5F	LD SP,IX	DDF9
LD B,(HL)	46	LD E,B	58	LD SP,IY	FDF9
LD B,(IX+zz)	DD46zz	LD E,C	59	LF SP,xxyy	31yyxx
LD B,(IY+zz)	FD46zz	LD E,D	5A	LDD	EDA8
LD B,A	47	LD E,E	5B	LDDR	EDB8
LD B,B	40	LD E,H	5C	LDI	EDA0
LD B,C	41	LD E,L	5D	LDIR	EDB0
LD B,D	42	LD E,xx	1Exx	NEG	ED44

NOP	ØØ	RES Ø,C	CB81	RES 4,D	CBA2
OR (HL)	B6	RES Ø,D	CB82	RES 4,E	CBA3
OR (IX+zz)	DDB6zz	RES Ø,E	CB83	RES 4,H	CBA4
OR (IY+zz)	FDB6zz	RES Ø,H	CB84	RES 4,L	CBA5
OR A	B7	RES Ø,L	CB85	RES 5,(HL)	CBAE
OR B	BØ	RES 1,(HL)	CB8E	RES 5,(IX+zz)	DDCBzzAE
OR C	B1	RES 1,(IX+zz)	DDCBzz8E	RES 5,(IY+zz)	FDCBzzAE
OR D	B2	RES 1,(IY+zz)	FDCBzz8E	RES 5,A	CBAF
OR E	B3	RES 1,A	CB8F	RES 5,B	CBA8
OR H	B4	RES 1,B	CB88	RES 5,C	CBA9
OR L	B5	RES 1,C	CB89	RES 5,D	CBAA
OR xx	F6xx	RES 1,D	CB8A	RES 5,E	CBAB
OTDR	EDBB	RES 1,E	CB8B	RES 5,H	CBAC
OTIR	EDB3	RES 1,H	CB8C	RES 5,L	CBAD
OUT (C),A	ED79	RES 1,L	CB8D	RES 6,(HL)	CBB6
OUT (C),B	ED41	RES 2,(HL)	CB96	RES 6,(IX+zz)	DDCBzzB6
OUT (C),C	ED49	RES 2,(IX+zz)	DDCBzz96	RES 6,(IY+zz)	FDCBzzB6
OUT (C),D	ED51	RES 2,(IY+zz)	FDCBzz96	RES 6,A	CBB7
OUT (C),E	ED59	RES 2,A	CB97	RES 6,B	CBBØ
OUT (C),H	ED61	RES 2,B	CB9Ø	RES 6,C	CBB1
OUT (C),L	ED69	RES 2,C	CB91	RES 6,D	CBB2
OUT (xx),A	D3xx	RES 2,D	CB92	RES 6,E	CBB3
OUTD	EDAB	RES 2,E	CB93	RES 6,H	CBB4
OUTI	EDA3	RES 2,H	CB94	RES 6,L	CBB5
POP AF	F1	RES 2,L	CB95	RES 7,(HL)	CBBE
POP BC	C1	RES 3,(HL)	CB9E	RES 7,(IX+zz)	DDCBzzBE
POP DE	D1	RES 3,(IX+zz)	DDCBzz9E	RES 7,(IY+zz)	FDCBzzBE
POP HL	E1	RES 3,(IY+zz)	FDCBzz9E	RES 7,A	CBBF
POP IX	DDE1	RES 3,A	CB9F	RES 7,B	CBB8
POP IY	FDE1	RES 3,B	CB98	RES 7,C	CBB9
PUSH AF	F5	RES 3,C	CB99	RES 7,D	CBBA
PUSH BC	C5	RES 3,D	CB9A	RES 7,E	CBBB
PUSH DE	D5	RES 3,E	CB9B	RES 7,H	CBBC
PUSH HL	E5	RES 3,H	CB9C	RES 7,L	CBBD
PUSH IX	DDE5	RES 3,L	CB9D	RET	C9
PUSH IY	FDE5	RES 4,(HL)	CBA6	RET C	D8
RES Ø,(HL)	CB86	RES 4,(IX+zz)	DDCBzzA6	RET M	F8
RES Ø,(IX+zz)	DDCBzz86	RES 4,(IY+zz)	FDCBzzA6	RET NC	DØ
RES Ø,(IY+zz)	FDCBzz86	RES 4,A	CBA7	RET NZ	CØ
RES Ø,A	CB87	RES 4,B	CBAØ	RET P	FØ
RES Ø,B	CB8Ø	RES 4,C	CBA1	RET PE	E8

RET PO	E0	RRC A	CB0F	SET 0,H	CBC4
RET Z	C8	RRC B	CB08	SET 0,L	CBC5
RETI	ED4D	RRC C	CB09	SET 1,(HL)	CBCE
RETN	ED45	RRC D	CB0A	SET 1,(IX+zz)	DDCBzzCE
RL (HL)	CB16	RRC E	CB0B	SET 1,(IY+zz)	FDCBzzCE
RL (IX+zz)	DDCBzz16	RRC H	CB0C	SET 1,A	CBCF
RL (IY+zz)	FDCBzz16	RRC L	CB0D	SET 1,B	CBC8
RL A	CB17	RRCA	0F	SET 1,C	CBC9
RL B	CB10	RRD	ED67	SET 1,D	CBCA
RL C	CB11	RST &00	C7	SET 1,E	CBCB
RL D	CB12	RST &08	CF	SET 1,H	CBCC
RL E	CB13	RST &10	D7	SET 1,L	CBCD
RL H	CB14	RST &18	DF	SET 2,(HL)	CBD6
RL L	CB15	RST &20	E7	SET 2,(IX+zz)	DDCBzzD6
RLA	17	RST &28	EF	SET 2,(IY+zz)	FDCBzzD6
RLC (HL)	CB06	RST &30	F7	SET 2,A	CBD7
RLC (IX+zz)	DDCBzz06	RST &38	FF	SET 2,B	CBD0
RLC (IY+zz)	FDCBzz06	SBC A,(HL)	9E	SET 2,C	CBD1
RLC A	CB07	SBC A,(IX+zz)	DD9Ezz	SET 2,D	CBD2
RLC B	CB00	SBC A,(IY+zz)	FD9Ezz	SET 2,E	CBD3
RLC C	CB01	SBC A,A	9F	SET 2,H	CBD4
RLC D	CB02	SBC A,B	98	SET 2,L	CBD5
RLC E	CB03	SBC A,C	99	SET 3,(HL)	CBDE
RLC H	CB04	SBC A,D	9A	SET 3,(IX+zz)	DDCBzzDE
RLC L	CB05	SBC A,E	9B	SET 3,(IY+zz)	FDCBzzDE
RLCA	07	SBC A,H	9C	SET 3,A	CBDF
RLD	ED6F	SBC A,L	9D	SET 3,B	CBD8
RR (HL)	CB1E	SBC A,xx	DExx	SET 3,C	CBD9
RR (IX+zz)	DDCBzz1E	SBC HL,BC	ED42	SET 3,D	CBDA
RR (IY+zz)	FDCBzz1E	SBC HL,DE	ED52	SET 3,E	CBDB
RR A	CB1F	SBC HL,HL	ED62	SET 3,H	CBDC
RR B	CB18	SBC HL,SP	ED72	SET 3,L	CBDD
RR C	CB19	SCF	37	SET 4,(HL)	CBE6
RR D	CB1A	SET 0,(HL)	CBC6	SET 4,(IX+zz)	DDCBzzE6
RR E	CB1B	SET 0,(IX+zz)	DDCBzzC6	SET 4,(IY+zz)	FDCBzzE6
RR H	CB1C	SET 0,(IY+zz)	FDCBzzC6	SET 4,A	CBE7
RR L	CB1D	SET 0,A	CBC7	SET 4,B	CBE0
RRA	1F	SET 0,B	CBC0	SET 4,C	CBE1
RRC (HL)	CB0E	SET 0,C	CBC1	SET 4,D	CBE2
RRC (IX+zz)	DDCBzz0E	SET 0,D	CBC2	SET 4,E	CBE3
RRC (IY+zz)	FDCBzz0E	SET 0,E	CBC3	SET 4,H	CBE4

SET 4,L	CBE5	SET 7,E	CBFB	SRL B	CB38
SET 5,(HL)	CBEE	SET 7,H	CBFC	SRL C	CB39
SET 5,(IX+zz)	DDCBzzEE	SET 7,L	CBFD	SRL D	CB3A
SET 5,(IY+zz)	FDCBzzEE	SLA (HL)	CB26	SRL E	CB3B
SET 5,A	CBEF	SLA (IX+zz)	DDCBzz26	SRL H	CB3C
SET 5,B	CBE8	SLA (IY+zz)	FDCBzz26	SRL L	CB3D
SET 5,C	CBE9	SLA A	CB27	SUB (HL)	96
SET 5,D	CBEA	SLA B	CB20	SUB (IX+zz)	DD96zz
SET 5,E	CBEB	SLA C	CB21	SUB (IY+zz)	FD96zz
SET 5,H	CBEC	SLA D	CB22	SUB A	97
SET 5,L	CBED	SLA E	CB23	SUB B	90
SET 6,(HL)	CBF6	SLA H	CB24	SUB C	91
SET 6,(IX+zz)	DDCBzzF6	SLA L	CB25	SUB D	92
SET 6,(IY+zz)	FDCBzzF6	SRA (HL)	CB2E	SUB E	93
SET 6,A	CBF7	SRA (IX+zz)	DDCBzz2E	SUB H	94
SET 6,B	CBF0	SRA (IY+zz)	FDCBzz2E	SUB L	95
SET 6,C	CBF1	SRA A	CB2F	SUB xx	D6xx
SET 6,D	CBF2	SRA B	CB28	XOR (HL)	AE
SET 6,E	CBF2	SRA C	CB29	XOR (IX+zz)	DDAEzz
SET 6,H	CBF4	SRA D	CB2A	XOR (IY+zz)	FDAEzz
SET 6,L	CBF5	SRA E	CB2B	XOR A	AF
SET 7,(HL)	CBFE	SRA H	CB2C	XOR B	A8
SET 7,(IX+zz)	DDCBzzFE	SRA L	CB2D	XOR C	A9
SET 7,(IY+zz)	FDCBzzFE	SRL (HL)	CB3E	XOR D	AA
SET 7,A	CBFF	SRL (IX+zz)	DDCBzz3E	XOR E	AB
SET 7,B	CBF8	SRL (IY+zz)	FDCBzz3E	XOR H	AC
SET 7,C	CBF9	SRL A	CB3F	XOR L	AD
SET 7,D	CBFA			XOR xx	EExx

Wenn es Ihnen Mühe bereitet, das abgedruckte Programm mit der Assemblierungsliste einzugeben, oder wenn Sie nur keine Lust dazu haben, können Sie bei uns eine Diskette bestellen, worauf das lauffähige Programm abgespeichert ist. Benützen Sie dazu untenstehenden Bestellzettel:

Bestellung: Ich bestelle hiermit ein Exemplar der Programmdiskette zum Buch "Maschinensprachehandbuch für den PC-1600" zum Preis von DM 98 .- (inkl. 14% MwSt.)

Der Betrag

( ) liegt als Verrechnungscheck bei
( ) wurde am auf das Postgirokonto der Fischel GmbH Nr.461533-103, BLZ 10010010, Giroamt Berlin- überwiesen
( ) Bitte als Nachnahmesendung ausliefern (plus 6 DM Nachnahmekosten)

Name, Vorname:

Adresse

PLZ / Ort

Datum, Unterschrift

Einzusenden an: Fischel GmbH
Kaiser-Friedrichstr. 54a
D-1000 Berlin 12
Deutschland