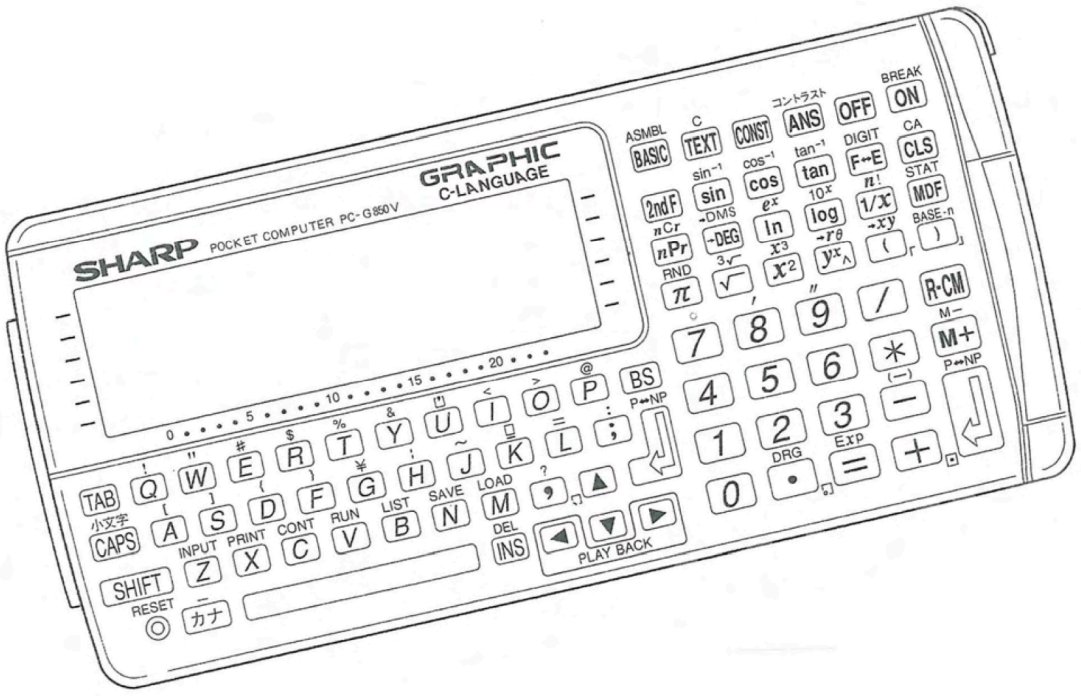
SHARP

TASCHENCOMPUTER

MODELL
PC-G850V(S)

BEDIENUNGSANLEITUNG



Version 1.3, 12/2017

Diese Bedienungsanleitung wurde NICHT durch die SHARP COOPERATION erstellt. Da diese Anleitung aber auf Basis verschiedener originaler Sharp-Anleitungen von Vorgänger-Modellen entstanden ist, ist die Verbreitung dieser Anleitung nur vorbehaltlich der freundlichen (bisher NICHT vorliegenden) Genehmigung durch die Firma Sharp gestattet.

Über die normale Sharp-Dokumentation hinaus beinhaltet diese Anleitung auch Ideen und Informationen die ich aus dem Internet zusammengetragen habe (Vielen Dank an die vielen fleißigen Forscher). Falls dadurch jemand der Meinung ist, dass dadurch seine Rechte verletzt wurden bitte unverzüglich an den Verfasser wenden.

Vielen Dank an Tom Stahl für den Anhang A (11-Pin Interface) und die Zuarbeit zum Anhang J (Display-Port). Durch ihn ist dieser Teil jetzt ordentlich verifiziert.

Für Fehler im Text, in den technischen Beschreibungen etc., sowie deren Folgen kann keine Haftung übernommen werden.

Jörg Wrabetz
joerg@wrabetz.com

EINFÜHRUNG

Der Pocket-Computer Sharp PC-G850V(S) ist das letzte Modell einer langen Reihe von Pocket-Computern die ihren Ursprung Ende der siebziger Jahre des vorigen Jahrhunderts hatten.
Gleichzeitig sticht er aus den anderen Modellen wegen einiger besonderen Merkmale stark heraus. Es gibt also zum G850 keine echten Vorgängermodelle. Trotzdem kann man sagen, dass als Basisfunktionalität der PC-E200/PC-E220/PD-G815 herangezogen wurden. Dazu flossen wichtige Funktionen des PC-1600 und PC-E500S ein. Auch einige mathematische Funktionen aus den PC-14xx-Modellen fanden Einzug.
Als eine Besonderheit ist im PC-G850V(S) ein C-Compiler/Interpreter integriert worden. Dieser ist am ehesten mit dem C-Interpreter des Casio Z-1GR zu vergleichen. Darüber hinaus wurde ein CASL-Assembler und die dazugehörige COMET-Umgebung integriert.

Diese Anleitung ist für den Sharp G850V und Sharp G850VS erstellt. Der Unterschied zwischen beiden Modellen ist sehr minimal. Der Unterschied besteht im leicht geringeren Gewicht (10g) und dem flashbaren Speicher des Betriebssystems im VS. Da aber dem Autor dieser Anleitung nicht bekannt ist, das es je ein Update für das Betriebssystem gab ist diese Eigenschaft irrelevant. Auch für die beiden Vorgänger G850 und G850S müsste dieses Dokument weitestgehend gültig sein, dies wurde aber nicht verifiziert.

Leider wurden die G-Modelle von Sharp nur noch innerhalb Japans vertrieben, so dass sie im Rest der Welt kaum zu finden sind und auch keine offizielle nicht-japanische Dokumentation verfügbar ist.

Grundlegende Eigenschaften des Sharp PC-G850V(S):

1. Eingebauter Assembler: Der Rechner ist mit einem eingebauten Assembler ausgestattet der es erlaubt Programme in der Maschinensprache des Z80 zu schreiben.

2. Programmierung in Basic. Der G850V(S) verfügt über eine mächtige Basic-Sprache die an den PC-1600 angelehnt ist und um Elemente des PC-E500S erweitert wurde.

3. Programmierung in C: Dazu verfügt der Rechner über einen eingebauten Compiler um einfache C-Programme ausführen zu können.

4. Programmieren in CASL. CASL ist eine Assembler-Sprache für einen virtuellen Computer COMET. Dieses System aus virtuellem Computer und dazugehörigen Assembler wurde vom japanischen Bildungsministerium ins Leben gerufen um für Schüler und Studenten eine einheitliche Ausbildung ohne eine spezielle Hardware zu ermöglichen.

5. Wissenschaftliche Berechnungen: Einfaches und leichtes Ausführen von wissenschaftlichen Berechnungen.

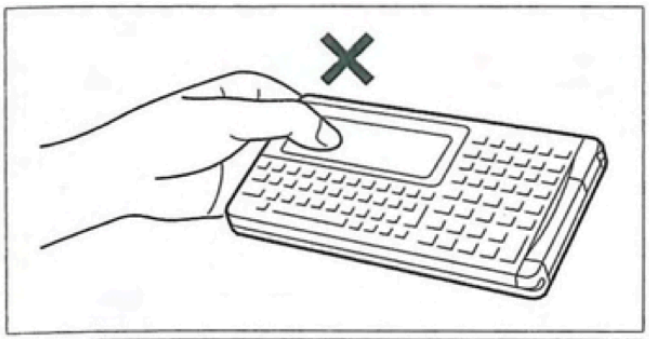
6. RAM-Disk: Ein Teil des internen Speichers kann wie eine RAM-Diskette genutzt werden um Programme und Daten abzulegen.

7. Serielles Interface: Damit ist es möglich, Programme und Daten zwischen verschiedenen Pocketcomputern oder auch einem PC auszutauschen.

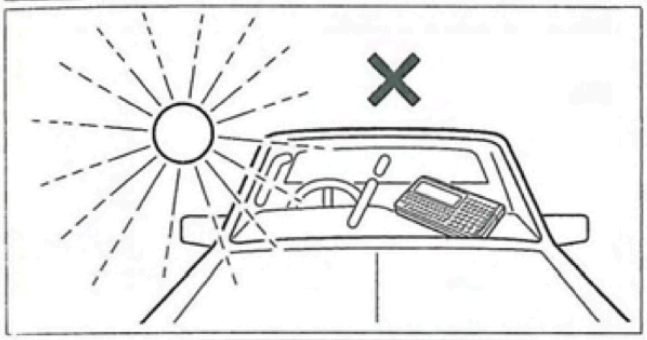
8. Anschluss des programmierbaren PIC-Microcontrollers

Inhaltsverzeichnis

1. UMGANG MIT DEM RECHNER



Bitte die Flüssigkristallanzeige nicht stark drücken. Die Anzeige kann dabei zerbrechen.



Bitte nicht in der Nähe von Heizungen aufbewahren oder direkter Sonneneinstrahlung aussetzen (z.B. in einem Auto). Aufgrund der hohen Temperaturen kann es zu Verformungen kommen.

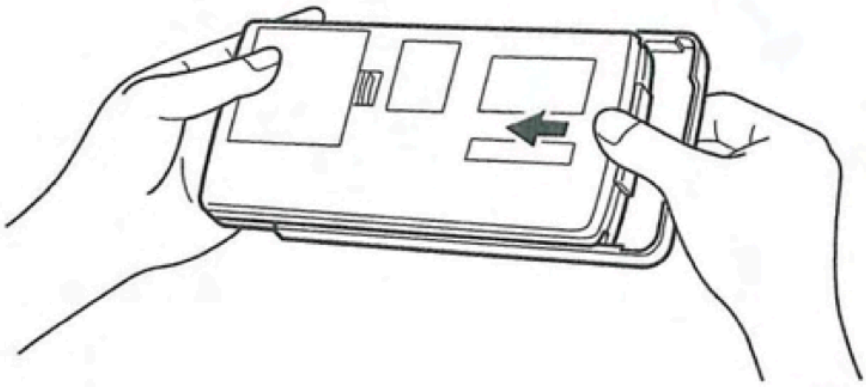
Nicht fallen lassen, drücken oder einer sonstigen großen Kraft aussetzen- Das Gerät kann dabei zerbrechen.

Reinigen Sie die Oberfläche mit einem weichen, trockenen Tuch. Verwenden Sie keine Lösungsmittel wie Verdünner, Benzin oder ein nasses Tuch Es kann zu Farbveränderungen oder zur Zerstörung der Oberfläche kommen.

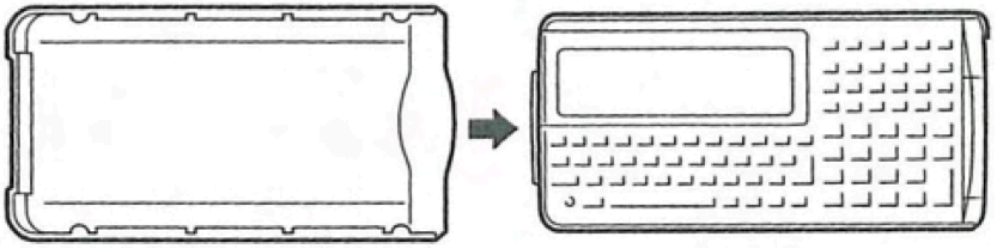
Bitte nicht zusammen mit harten oder spitzen Gegenständen in der Tasche aufbewahren. Das Gerät kann zerkratzt werden. Verwenden Sie es immer die Abdeckung. Das Produkt ist nicht wasserdicht.

Das Hardcover dient zum Schutz des Rechners gegen Stöße. Immer wenn Sie nicht mit dem Pocket-Computer, arbeiten installieren Sie bitte das Hardcover, z. B. wenn Sie den Rechner in die Tasche stecken.

Entfernen der Schutzhülle:

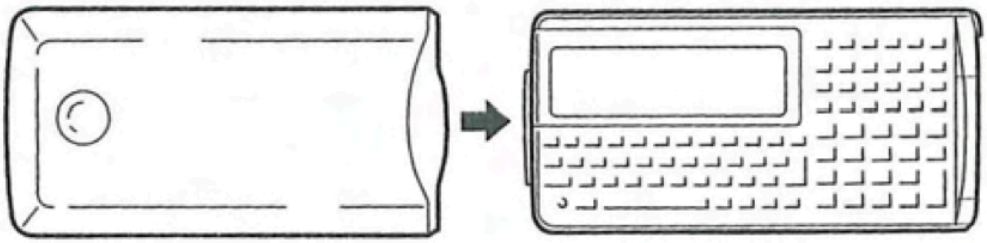


Bei Verwendung



Bei Nicht-

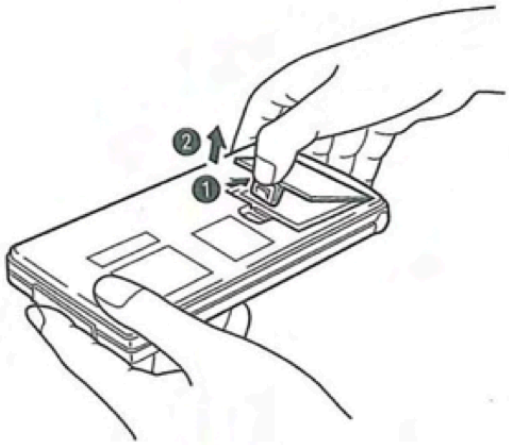
Verwendung:



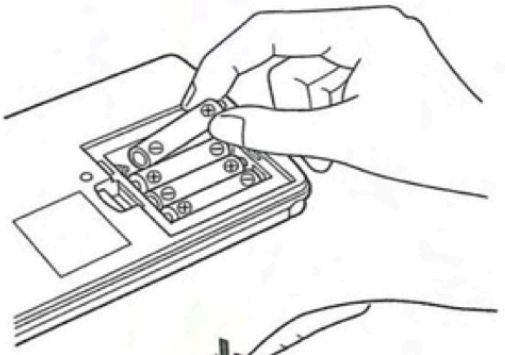
2. ERSTE VERWENDUNG DES SHARP PC-G850V

(1) Batterien einsetzen

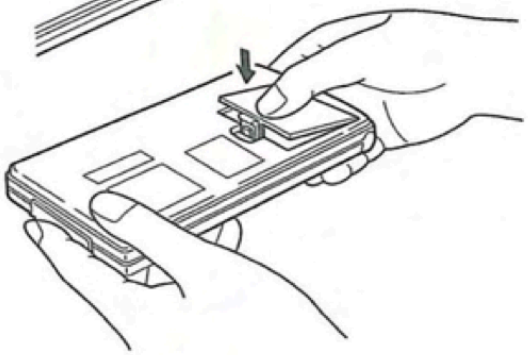
Bitte legen Sie die Batterien ein. Entfernen Sie dazu die Abdeckung des Batteriefachs auf der Rückseite wie in der Abbildung gezeigt.



Legen Sie die Batterien richtig herum ein. Richten Sie sich dabei nach den Piktogrammen im Batteriefach.



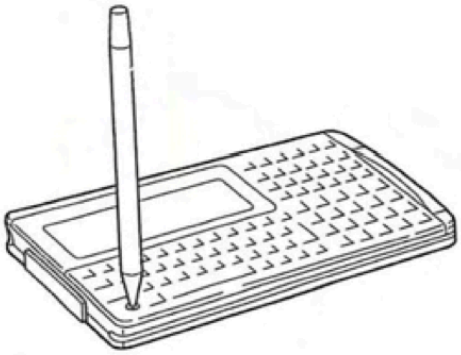
Danach bitte den Batteriedeckel wieder schließen.



(2) Rücksetzen

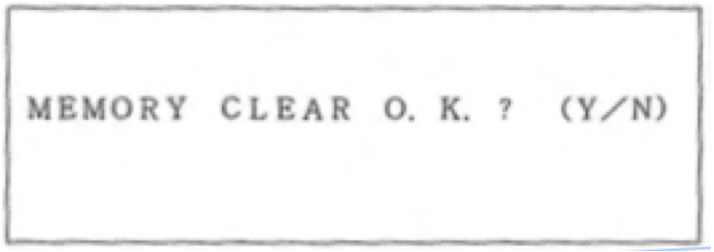
Direkt nach dem Einlegen der Batterien in den Computer ist der interne Status des PC-G850V noch nicht eingestellt. Dazu der Computer erst initialisiert werden.

1. Die ON-Taste drücken und dann den Reset-Taster unter der SHIFT-Taste mit einem Kugelschreiber oder einem ähnlichen Gerät eindrücken. Den Reset-Taster dann wieder loslassen.

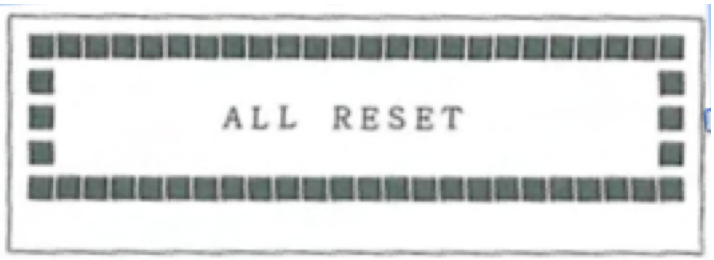


Direkt nach dem Drücken des RESET-Tasters zeigt der PC-G850V die folgende Anzeige. Falls irgendeine andere Anzeige erscheint, muss der obige Vorgang wiederholt werden.

Der PC-G850V fragt nach der Bestätigung zum Löschen des Speichers:



2. Die Taste Y drücken. Die folgende Meldung blinkt auf und zeigt damit an, daß der Computer initialisiert wurde und alle Speicherinhalte gelöscht sind.



3. Eine beliebige Taste drücken. Folgende Anzeige erscheint:

(3) Überprüfen der normalen Computerfunktion

Zur Sicherstellung der normalen Computerfunktion folgende Tasten drücken:

Wenn die oben dargestellte Anzeige erscheint, funktioniert der Computer normal und ist bereit für die Eingaben.
Die Zahl 30179 stellt die Speicherkapazität für Programme und Daten dar.

Hinweis:
Wenn der PC-G850V nach den oben beschriebenen Schritten nicht diese Anzeige darstellt, sollte die Anweisung für den entsprechenden Schritt noch einmal gelesen werden. Versuchen Sie noch einmal die richtige Eingabe für diesen Schritt.

Falls die Anzeige BATT erscheint müssen die Batterien gewechselt werden

Auswechseln der Batterien

Der Computer verwendet für den Betrieb vier Batterien vom Typ AAA.
Wenn die Batterien zu schwach sind, während der Drucker CE-126P gleichzeitig mit dem Computer verwendet wird, kann er auch über den CE-126P mit Strom versorgt werden. Dadurch wird die Belastung der internen Batterie vermindert.

Zeitpunkt des Wechsels der Batterien

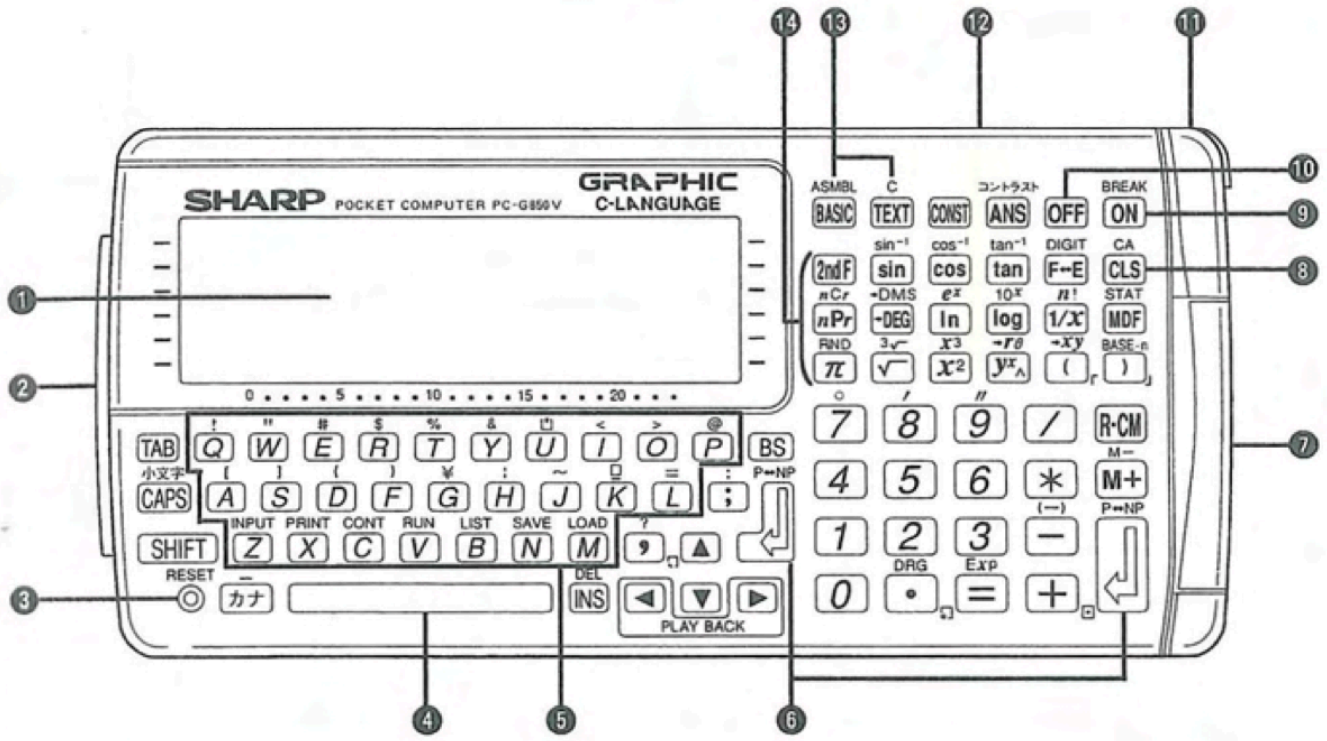
Wenn in der unteren linken Ecke des Displays die Warnanzeige BATT erscheint, bedeutet dies, dass die Batterien zu schwach sind. Sie sollten sofort durch neue ersetzt werden. Wenn der Computer weiterhin verwendet, obwohl BATT angezeigt wird, schaltet sich der Computer nach einiger Zeit aus. Danach kann er auch durch drücken der ON-Taste nicht mehr eingeschaltet werden.
Hinweis: Der Pocket-Computer behält seine Programme und Dateien über einen längeren Zeitraum auch ohne Batterien. Sicherheitshalber sollten Sie dennoch die Batterien nicht länger als 5 Minuten aus dem Computer entfernen.

Wichtig: Die Batterien **NIE** bei eingeschalteten Pocket-Computer entfernen da dann nach Neueinlegen der Batterien der Computer immer zurückgesetzt werden muss und damit alle Daten verliert. Auch sollten Sie eventuell alle Programme und Daten vorher auf einen PC sichern oder ausdrucken.

Wenn ein zusätzliches Peripheriegerät angeschlossen wurde, kann der Computer über dieses Gerät mit Strom versorgt werden. In diesem Fall erscheint die Warnanzeige BATT nicht, obwohl die Batterien des Computers zu schwach sind. Vor dem Gebrauch sollte das Peripheriegerät kurzfristig abgetrennt werden um zu prüfen, ob die Warnanzeige BATT auf dem Display erscheint oder nicht.
Desweiterem befindet sich an der hinteren rechten Seite ein Anschluss um den Computer mit einem externen Netzteil mit Strom zu versorgen (6V,0.2W)

3. GERÄTEÜBERSICHT

Die Ausstattung des SHARP Computers besteht aus einer QWERTY-Tastatur, die der einer herkömmlichen Schreibmaschine ähnelt und einer LCD-Anzeige mit einstellbarem Kontrast. Auf der linken Seite befindet sich das SHARP-11-Pin-Interface und auf der rechten Seite das Interface zum PIC-Mikrokontroller. Rechts oben befindet sich ein Anschluss für ein externes Netzteil mit 6V und 0.2W (z.B. Netzteil Sharp EA-23E).



1 Display (6 Zeilen, 24 Zeichen/Zeile) 144x48 Pixel)
2 SHARP-11-Pin-Interface für Drucker, serielles Interface usw.
3 Reset-Taster (versenkt)
4 Leertaste
5 Schreibmaschinen-Tastatur
6 Enter-Taste(n)
7 Interface für PIC-Microkontroller
8 Löschtaste
9 Power-On-Taste, Taste zum Aufwecken
10 Power-Off-Taste
11 Anschluss für Netzteil (6V, 0.2W z.B. Netzteil Sharp EA-23E).
12 Batteriefachdeckel (auf der Rückseite)
13 Modus-Umschalttasten (Basic RUN/PRO, Assembler, C, CASL, Text-Editor)
14 Funktionstasten

2 GRUNDLEGENDE FUNKTIONEN UND MODI

Es gibt beim PC-G850V eine Reihe von wichtigen Buchstaben, Zahlen und verschiedene Symbole.

3.1. ANSCHALTEN DES COMUTERS

Die ON-Taste auf der rechten Seite der Computer-Tastatur drücken. Der Computer befindet sich nach dem Einschalten im RUN-Modus.

3.2. AUTOMATISCHE ABSCHALTUNG

Zum Schutz der Batterien schaltet sich der Computer automatisch selbst aus, wenn etwa 11 Minuten lang keine Tasten gedrückt werden.
Die ON-Taste zum erneuten Einschalten drücken, wenn sich der Computer selbst abgeschaltet hat.

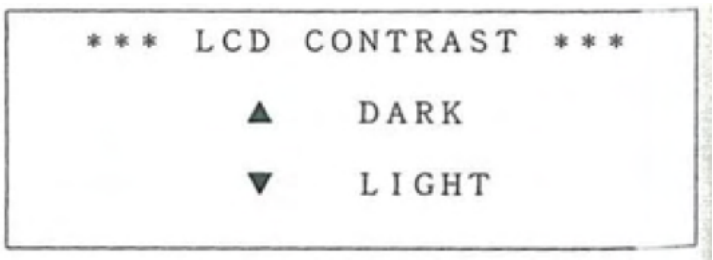
Während der Computer einen INKEY$-Befehl ausführt ist die automatische Abschaltfunktion wirkungslos. Sie ist wirksam während der Computer einen INPUT-Befehl ausführt.

Wenn der Computer längere Zeit nicht benutzt wird, während die automatische Abschaltung nicht wirksam ist, wird Batteriestrom verbraucht. Es kann dabei zum Verlust von gespeicherten Programmen bzw. Daten kommen.

3.3. EINSTELLEN DES KONTRASTES

Die Anzeige zum Einstellen des Kontrasts wird durch drücken von Shift und ANS aufgerufen.

Bitte stellen Sie den Kontrast so ein, so dass Sie die Anzeige gut erkennen können.



Durch drücken der Cursortasten ▲ (Kontrast verstärken) und ▼ (Kontrast verringern) kann der Kontrast eingestellt werden.

Ist die Anzeige richtig eingestellt kann die Einstellung durch drücken entweder der BASIC-, TEXT- oder CLS-Taste verlassen werden.

3.4. DIE MODI DES SHARP PC-G850

Der Sharp PC-G850V verfügt über 7 verschiedene Modi:

RUN-Modus	führen von BASIC-Programmen oder BASIC-Kommandos, Eingabe von mathematischen Funktionen
PRO-Modus	Zum Schreiben oder Korrigieren von BASIC-Programmen
TEXT-Modus	Zur Eingabe, Bearbeitung, Löschen und Speichern(Ramdisk,SIO), Laden(Ramdisk,SIO) eines Text-Programms im ASCII-Format, Umwandlung nach BASIC oder umgekehrt, Anlegen und löschen von Daten-Dateien.
ASMBL-Modus (Assembler-Modus)	Zum assemblieren eines Assembler-Programms (Erzeugung von Maschinencode (Z80),
CASL-Modus	Übersetzen und Ausführen von CASL-Programmen (Erreichbar über ASMBL)
PIC-Modus	Übersetzen von Quellprogrammen des PIC und zum PIC übertragen. (Erreichbar über ASMBL)
C-Modus	Kompilieren und Ausführen von C-Programmen.

Modus-Umschaltung

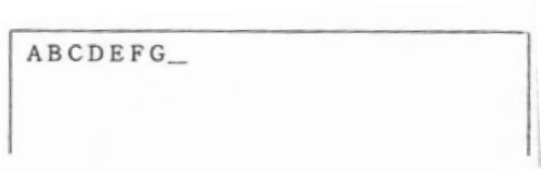
Modus	Tasten
RUN-Modus	BASIC
PRO-Modus	BASIC oder BASIC BASIC (um von außerhalb des RUN-Modus in den PRO-Modus zu gelangen, drücken Sie BASIC zweimal)
TEXT-Modus	TEXT
ASMBL-Modus	SHIFT + BASIC(ASMBL) , dann A
CASL-Modus	SHIFT + BASIC(ASMBL) , dann C
PIC-Modus	SHIFT + BASIC(ASMBL) , dann P
C-Modus	SHIFT + TEXT(C)

Statt SHIFT kann vorher auch 2ndF gedrückt werden.

3.5. GRUNDLEGENDE BEDIENUNG

Schalten Sei den Computer ein. Durch drücken der CLS wird der Bildschirm gelöscht und die Eingabe beginnt in der oberen linken Ecke.

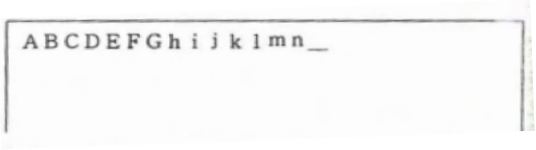
(1) Eingabe von Zeichen



Auf dieser Art und Weise erhalten Sie Großbuchstaben.

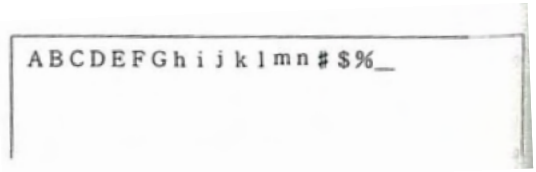
(2) Eingabe von Kleinbuchstaben

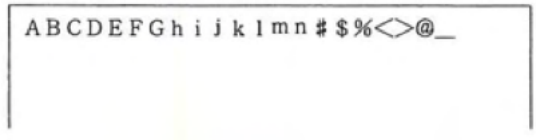
Durch das Drücken der CAPS-Taste wird der CAPS-Modus verlassen (immer eingeschaltet nach Power-On) und bei der Eingabe von Buchstaben werde diese klein geschrieben.



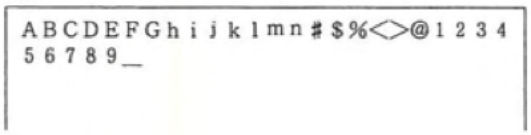
(3) Eingabe von Sonderzeichen

Durch gleichzeitiges Drücken der Taste SHIFT und der dazugehörigen Taste wird das entsprechende Sonderzeichen geschrieben. Alternativ kann auch VOR dem drücken der entsprechenden Taste die Taste 2ndF gedrückt werden. Der folgende Tastendruck schreibt dann immer das Sonderzeichen oder die mathematische Funktion.





Genauso werden Zahlen geschrieben



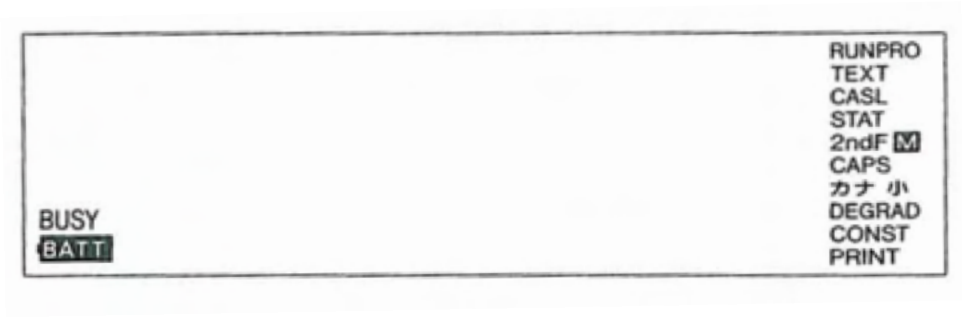
(4) Cursorsteuerung

Um die eingegeben Zeichen im nachhinein zu verändern können die vier Cursortasten verwendet werden.(◄ ► ▲ ▼)

Befindet sich der Cursor am Ende einer Eingabezeile erschein ein Unterstrich ab der die Zeile weitergeführt wird. Befindet sich der Cursor mitten im Textfeld blinkt das entsprechende Zeichen schwarz auf. Die Eingabe von Zeichen überschreibt die bestehenden Zeichen ab dieser Position. Wenn sie die gewählte Cursortaste gedrückt halten bewegt sich der Cursor schnell über den Bildschirm.

3.6. DAS DISPLAY

Der Computer hat ein 6-Zeilen-Flüssigkristall-Display mit 24 Zeichen pro Zeile und einer Statuszeile am oberen und am unteren Rand. Jedes Zeichen besetzt eine 5 x 7 Punktmatrix. Das Display zeigt Tastenbezeichnungen und Berechnungsvorgänge, Die Display-Beispiele dieser Anleitung geben nur die für die jeweilige Erklärung der Funktion notwendigen Symbole wieder.



In der BASIC-Betriebsart zeigt das Display folgendes:

>

Bereitschaftssymbol. Dieses Symbol erscheint wenn der Computer in der BASIC-Betriebsart bereit ist, eine Eingabe anzunehmen. Beim Tippen. Beim Tippen verschwindet das Bereitschaftssymbol und wird durch den Cursor ersetzt

Der Cursor. Dieses Symbol markiert die Stelle des nächsten einzugebenden Zeichens. Wenn man mit dem Tippen beginnt, ersetzt der Cursor das Bereitschaftssymbol. Als Markierungssymbol wird der Cursor auch im Zusammenhang mit den INSert- und DELete-Funktionen benutzt.

Der Unterstrich-Cursor ändert sich zum Block-Cursor wenn er sich nicht auf einem Zeichen befindet.

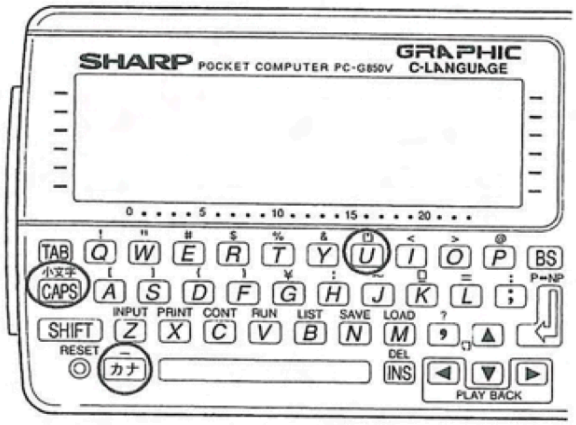
Die Statuszeilen geben folgendes wieder:

BUSY	Dieses Wort erscheint auf dem Display, wenn der Computer ein Programm oder ein Kommando ausführt.
BATT	Dieses Symbol weist Sie darauf hin, daß die Batterien schwach sind und ausgewechselt werden müssen.
RUN	Dieses Symbol zeigt den RUN-Modus für den Computer an.
PRO	Dieses Symbol zeigt den BASIC-Programmierungs-Modus für den Computer an.
TEXT	Zeigt an, dass sich der Computer in der TEXT-Betriebsart befindet.
CASL	Dieses Symbol zeigt den CASL-Programmierungs-Modus für den Computer an. In diesen Modus gelangen sie durch drücken der Taste ASMBL (SHIFT+BASIC) und danach C
STAT	Dieses Symbol zeigt an, dass sich der Computer im Statistik-Modus befindet. In diesen Modus gelangen sie durch drücken der Taste STAT (SHIFT+MDF)
2ndF	Die Anzeige erscheint, wenn die 2ndF-Taste gedrückt wurde und erlischt mit dem folgenden Tastenkommando. Denken Sie daran, dass die 2ndF-Taste vor dem Druck auf eine andere Taste freigegeben werden muss, wenn die zweite
M	Zeigt an, dass eine andere Zahl außer Null für manuelle Berechnungen gespeichert ist.
CAPS	Zeigt an, dass sich der Computer in der Betriebsart für Großbuchstaben (CAPS) befindet. Wenn diese Anzeige nicht auf dem Display erscheint, werden alle Buchstaben des Alphabets als Kleinbuchstaben eingegeben. Mit der CAPS-Taste kann die CAPS-

	Betriebsart ein- und wieder ausgeschaltet werden.
カナ	Wenn Sie die Taste drücken können Katakana-Silben mit lateinischen Buchstaben eingeben werden. (Siehe Seite 15) Durch Drücken der Taste können Sie diese Funktion Ein- und Ausschalten
小	Zeigt an, dass sich der Computer im Katanka-Modus befindet und durch Ausschalten der CAPS-Funktion Kleinbuchstaben eingegeben werden können.
DEG RAD **GRAD**	Zeigt in welchem Winkel-Modus sich der Computer befindet: DEG (Altgrad-Modus) RAD (Bogenmaß-Modus) GRAD (Neugrad-Modus)
CONST	Zeigt an, dass im Computer eine Konstante für Berechnungen mit Konstanten gespeichert ist. Wenn dieses Symbol angezeigt wird, führt der Computer jedes mal beim Drücken der Taste eine Berechnung mit dieser Konstante aus. Wenn die Konstante nicht mehr benötigt wird, kann sie mit SHIFT+ CA gelöscht werden.
PRINT	Dieses Symbol zeigt an, dass der Computer bereit ist, im RUN-Modus Daten zum Drucker zu senden. Drücken Sie SHIFT + P↔NP zum Ein- und Ausschalten (ON und OFF). (Nur möglich mit einem angeschlossenen zusätzlichen Drucker.)

3.7. EINGABE VON Kanji-Zeichen

Der PC-G850V bietet Ihnen die Möglichkeit japanische Wörter im Kanji-Format einzugeben. Diese Funktion wird durch drücken der Taste カナ ein- bzw. ausgeschaltet.



カナ .....Ein- und Ausschalten der Kanji-Eingabe

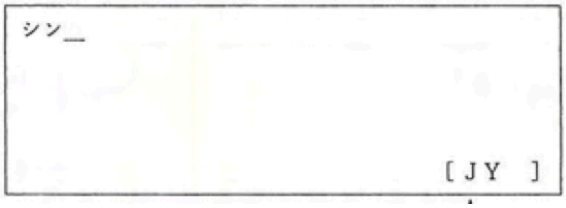
小文字 CAPS .....Umschalten auf Eingabe von Kleinbuchstaben

SHIFT + Ü ... Eingabe von Konsonanten einleiten (???)

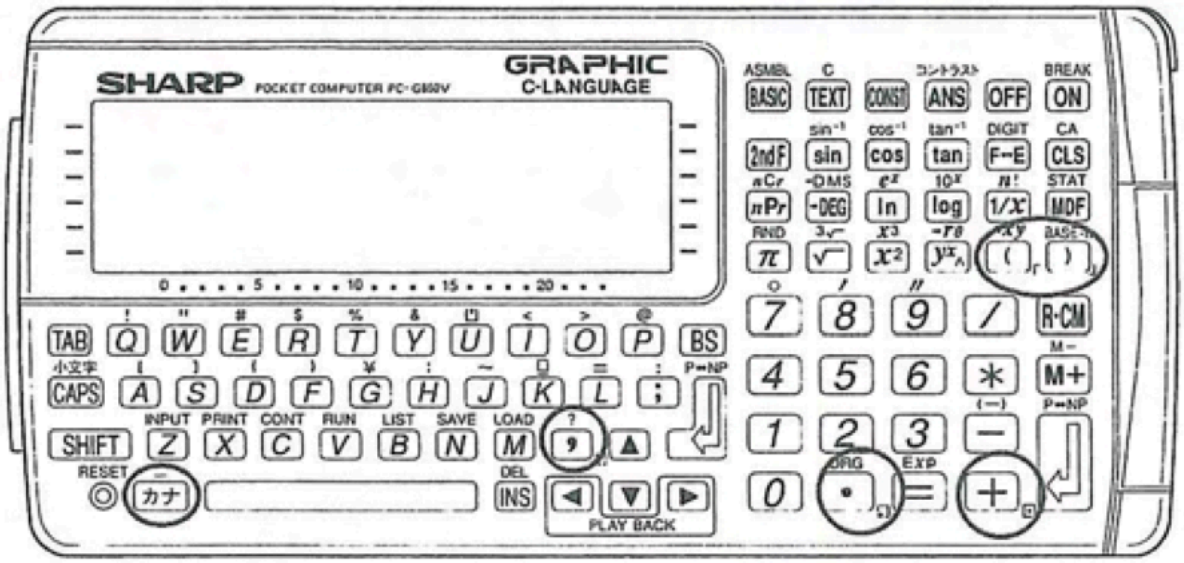
Beispiele:	Ergebnis der Ausgabe		Tastatur-Eingabe
	カタカナ	->	KATAKANA
	ガッコウ	->	GAKKOU
	ヘンカン	->	HENNKAN SHIFT + Ü
	ハンイ	->	DHISUKU
	ディスク	->	HAN SHIFT + Ü I
	ウォッチ	->	U 小文字 CAPS OTTI

Achtung es können nicht immer alle Zeichen der Eingabe in der unteren Zeile angezeigt werden:

SINJY



Spezielle Kanji-Symbole:



ー（長音符） : SHIFT + カナ

、（読点） : , □

。（句点） : . □

・（中点） : + □

「（カギカッコ） : (「

」（カギカッコ） :) 」

3 MANUELLE BERECHNUNGEN

Der Computer kann wie ein Taschenrechner mit 10 Stellen verwendet werden. Dazu muss der Computer auf den BASIC-RUN-Modus eingestellt werden. Die Anzeige RUN erscheint dazu oben rechts im Display.

	Math. Funktion	Math. Operator	Eingabe
(1)	Addition	+	+
(2)	Subtraktion	-	-
(3)	Multiplikation	x	*
(4)	Division	÷	/
(5)	Integer-Divison		¥
(6)	Modulo-Division		MOD
(7)	Vorzeichen		-,+
(8)	Ausführen der Operation	=	↵

Beispiele:

51¥5 → 10	51 MOD 5 → 1	(51÷5=10...1)
51¥−5.7 →− 8	51 MOD − 5.7 → 3	(51÷−6=−8...3)
87.57¥5.4 → 17	87.57 MOD 5.4 → 3	(88÷5=17...3)
30°36′¥14°36′ → 2	30°36′ MOD 14°36′ → 1	(31÷15=2...1)
30°36′¥4.4 → 7	30°36′ MOD 4.4 → 3	(31÷4=7...3)
87.57¥14°36′ → 5	87.57 MOD 14°36′ → 13	(88÷15=5...13)

5+15*2/4 ↵ —› 12.5

3.1 TASTENBEDIENUNG

Durch die Eingabe der Cursor-Taste kann die Eingabe wieder editiert und damit verändert werden. Dazu den Cursor mit Hilfe der Cursortasten auf die entsprechende Position setzen und das Zeichen mit einem neuen überschreiben. Sollen Zeichen eingefügt werden muss vor der Eingabe die INS-Taste betätigt werden. Der Einfüge-Modus bleibt aktiv bis die INS-Taste noch einmal gedrückt wird. Soll dagegen das Zeichen unter dem Cursor gelöscht werden muss die Taste DEL (SHIFT+INS) gedrückt werden. Soll dagegen das Zeichen vor dem Cursor gelöscht werden kann dazu die BS-Taste benutzt werden.

3.2 TASTEN FÜR MATHEMATISCHE OPERATIONEN

ANS: Weiterverwendung eines Ergebnisses

Wird bei der Eingabe einer mathemischen Funktion die ANS verwenden, wird an der aktuellen Stelle das Ergebnis der letzten Berechnung eingefügt.

Beispiel:

5+15*2/4 —› 12.5

4*ANS (fügt 12.5 ein) —› 50

Exponenten-Funktionen EXP, 10^x und e^x

Die Eingabe des Exponenten einer Zahl wird durch SHIFT+Exp (.) eingeleitet. Statt EXP kann auch einfach der Buchstabe E verwendet werden.

Beispiel:

36**EXP**3 —› 36E3 —› 36000

52**10^x** —› 1.E 52

5 **e^x** —› 148.4131591

DIGIT: Festlegung der Anzahl von Dezimalstellen

Diese Taste dient, in Verbindung mit einer Zahlentaste, zur Spezifizierung der Anzahl der Dezimalstellen (Nachkommastellen).

Wir anstatt der Zahlentaste der Punkt(.) verwendet, wird der Computer wieder auf die normale Darstellung zurückgestellt.

Mit der Taste F<->E schalten Sie zwischen FIX-Modus und Exponentialmodus um.

2ndF DIGIT 2 ↵

5/8 ↵ —› 0.63

2ndF DIGIT 5 ↵ —› 0.62500

2ndF DIGIT . ↵ —› 0.625

USING: Festlegung des Ausgabeformats

Format : USING [<formatstring>]

USING erlaubt eine formatierte Datenausgabe.
Das Format wird durch einen **<Format-String>** bestimmt, der sich aus folgenden Zeichen zusammensetzen kann:

Das jeweilige Format wird durch einen <Format-String>, der der USING-Anweisung als Parameter beizugeben ist, bestimmt.

Der <Format-String> besteht aus einer Reihe von speziellen Zeichen, die in Anführungsstriche eingeschlossen sein müssen.

Die Zeichen, die den <Format-String> bilden, sind:

#	Rechtsbündiges Zeichen eines numerischen Feldes
.	Begrenzer zwischen der ganzen Zahl und dem Dezimalanteil
,	verwendet das Komma als Trennzeichen nach 3 Ziffern in numerischen Feldern
^	Zur Anzeige der Zahl in wissenschaftlicher Notation

Das Nummernzeichen und das Kaufmannsund sind sogenannte Platzhalter. Für jedes im <Formatstring enthaltene # kann eine Ziffer des numerischen Wertes angezeigt werden,

für jedes & dagegen ein Zeichen eines Strings. Alle weiteren Formatsymbole dienen der näheren Beschreibung numerischer Formate. Bei den numerischen Formaten lassen sich sowohl positive als auch negative Werte darstellen. Das Vorzeichen wird jedoch nur bei den negativen Werten angezeigt.

Das Format für USING entspricht dem BASIC-Befehl USING (siehe Seite 249). USING ohne Parameter setzt das Ausgabeformat auf den Standard zurück.

Beispiel:

USING "###.##" [↵]

8/3 [↵] —› 2.66

17/3 [↵] —› 5.66

Ist das Ergebnis größer als die das definierte Ausgabeformat zulässt wird ein Fehler (ERROR 70) ausgegeben

USING "###.##" [↵]

17865/3 [↵] —› ERROR 70

MDF: Modifizierungs-Funktion

Mit der Funktion für Dezimalstellen zeigt der Computer nur die festgelegte Anzahl der Dezimalstellen an, intern speichert er aber insgesamt immer alle Stellen. Daher können sich die angezeigten Daten von den internen Daten unterscheiden. Um die internen mit den angezeigten Daten in Übereinstimmung zu bringen, wird die Modifizierungs-Funktion verwendet.

Beispiel:

Ohne MDF:

キ ー 操 作	表 示 部
([2nd F] [CA])	
55.4 [÷] 9 [↵]	6.155555556
[×] 9	6.155555556*9_
[↵]	55.4

mit MDF:

キ ー 操 作	表 示 部
2ndF DIGIT 3	
55.4 ÷ 9 ↵	6.156
MDF	6.156
* 9	6.156*9_
↵	55.404

Vorzeichenwechsel

Mit Hilfe der Taste (-) (2ndF + -) wird das Vorzeichen des angezeigten Ergebnisses umgekehrt (von Plus nach Minus und umgekehrt).

Speicherberechnung

Der unabhängige Speicher kann mit den Tasten **M+**, **M-** und **R-CM** angewählt werden.

R-CM	Zeigt den Inhalt der gespeicherten Zahl und fügt sie in die Berechnung ein. Wird **R-CM** zweimal gedrückt, wird der Inhalt des Speichers gelöscht. Das M-Symbol erlischt.
M+	Fügt das angezeigte Ergebnis oder das Ergebnis einer mathematischen Funktion dem Speicher hinzu. War vorher der Speicher leer (also 0) erscheint das M-Symbol
M-	Subtrahiert das angezeigte Ergebnis oder das Ergebnis einer mathematischen Funktion vom Speicher. War vorher der Speicher leer (also 0) erscheint das M-Symbol

Berechnungen mit Konstanten

Mit der Taste CONST können Konstanten wie im folgenden beschrieben für die Grundrechenarten verwendet werden.

Verwendung von Konstanten

Addition: + a **CONST** oder a + **CONST**

Subtraktion: - a **CONST** oder a - **CONST**
Multiplikation: * a **CONST** oder a * **CONST**
Division: / a **CONST** oder a / **CONST**

Dabei bedeutet "a" die Konstante.
Nach dem Drücken von **CONST** erscheint "CONST" unten rechts auf dem Display.

Hinweis
Wenn die Konstanten-Funktion nicht verwendet wird, muss sichergestellt werden, dass die Anzeige "CONST" nicht auf dem Display erscheint.

Überprüfung der Konstanten-Einstellung
Zur Überprüfung der zuletzt eingegebenen Konstante wird **CONST** während "CONST" angezeigt wird.

2ndF CONST (SHIFT + CONST)

Löschen der letzten Konstante
Zum Löschen der zuletzt eingegebenen Konstante folgendermaßen vorgehen. Sie kann auch durch Abschalten des Gerätes gelöscht werden.

2ndF CA (SHIFT + CA)

Beispiel:

Speichern von "+ (4,8 + 3,6)" als Konstante und Berechnung von "24 — 18,5 + (4,8 + 3,6)" und "8,2 x 6 + (4,8 + 3,6)"
Eingabe: +4.8+ 3.6 **CONST**
Die Konstante muss nicht in Klammern geschrieben werden.

Eingabe: 24-18.5  Ergebnis: 13.9

Eingabe:8,2*6  Ergebnis: 57.6

4 WISSENSCHAFTLICHE UND MATHEMATISCHE BERECHNUNGEN

Der Computer ist mit zahlreichen eingebauten Funktionen für wissenschaftliche, mathematische und statistische Berechnungen ausgestattet. Eine alphabetische Liste folgt weiter unten. All diese Funktionen können bei der Anwendung des Computers im RUN-Modus als Teil von Berechnungen benutzt werden bzw. auch als BASIC-Anweisungen in einem Programm.

Für trigonometrische Funktionen können nach Bedarf Altgrad-, Bogenmaß- oder Neugrad-Werte eingegeben werden:

ALTGRAD: Stellen Sie den Computer auf den Altgrad-Eingabe-Modus durch Eingeben des Befehls T DEGREE (auf der Statuszeile im Displayerscheint DEG). Dies ist die Grundeinstellung.

BOGENMASS: Stellen Sie den Computer in den Bogenmaß-Eingabe-Modus durch die Eingabe des Befehls RADIAN (auf der Statuszeile im Display erscheint RAD).

NEUGRAD: Stellen Sie den Computer in den Neugrad-Eingabe-Modus durch die Eingabe des Befehls GRAD (auf der Statuszeile im Display erscheint GRAD).

Diese drei Modi (DEG, RAD und GRAD) können auch innerhalb eines Programms eingestellt werden. Ist ein Modus eingestellt, müssen alle trigonometrischen Funktionen der Einstellung des Computers angepasst werden (Altgrad-, Bogenmaß- oder Neugrad-Werte), bis der Modus von Hand oder innerhalb eines Programmes geändert wird. Die folgenden Beispiele gelten für Direkt-Eingabe der in Altgrad eingegebenen Funktionen.

Die meisten Funktionen können auch durch Drücken der entsprechenden Funktionstaste eingegeben werden.

Es ist nicht möglich manuelle Berechnung im PRO-Modus direkt ausführen.

ABS |X|

Funktion: Absoluter Wert

Anmerkungen: Zeigt den absoluten Wert des numerischen Arguments an. Der absolute Wert ist der Wert einer Ziffer unabhängig von seinem Vorzeichen. ABS-10 ist 10.

Beispiel: ABS -10 10

ACS $\cos^{-1}x$

Funktion: Reziproker oder Arcuscosinus

Anmerkungen: Zeigt den Arcuscosinus des numerischen Arguments an. Der Arcuscosinus ist der Winkel, dessen Cosinus gleich dem Ausdruck ist. Der Wert ist abhängig von der Einstellung in Altgrad, Bogenmaß oder Neugrad.
Die entsprechende Funktionstaste ist **$\cos^{-1}$**

Beispiel: DEGREE

ACS -0.5 120

AHC $\cosh^{-1}x$

Funktion: Reziproker hyperbolischer Cosinus (Areafunktion)

Anmerkungen: Zeigt den Cosinus der Areafunktion des numerischen Arguments an.

Beispiel: AHC 10 2.993222846

AHS $\sinh^{-1}x$

Funktion: Reziproker hyperbolischer Sinus

Anmerkungen: Zeigt den Sinus der Areafunktion des numerischen Arguments an.

Beispiel: AHS 27.3 4.000369154

AHT $\tanh^{-1}x$

Funktion: Reziproker hyperbolischer Tangens

Anmerkungen: Zeigt den Tangens der Areafunktion des numerischen Arguments an.

Beispiel: AHT 0.7 0.867300527

ASN $\sin^{-1}x$

Funktion: Reziproker oder Arcussinus

Anmerkungen: Zeigt den Arcussinus des numerischen Arguments an. Der Arcussinus ist der Winkel, dessen Sinus gleich dem Ausdruck ist. Der angezeigte Wert ist abhängig vom eingestellten Modus (Altgrad, Bogenmaß oder Neugrad)
Die entsprechende Funktionstaste ist **$\sin^{-1}$**

Beispiel: DEGREE
ASN 0.5 30

ATN $\tan^{-1}x$

Funktion: Reziprok oder Arcustangens

Anmerkungen: Zeigt den Arcustangens des numerischen Arguments an. Der angezeigte Wert ist abhängig vom eingestellten Modus (Altgrad, Bogenmaß oder Neugrad).
Die entsprechende Funktionstaste ist **$\tan^{-1}$**

Beispiel:
```
DEGREE
ATN 1                    45
```

COS $\cos x$

Funktion: Cosinus

Anmerkungen: Zeigt den Cosinus des Winkelarguments an. Der angezeigte Wert ist abhängig vorn eingestellten Modus (Altgrad, Bogenmaß oder Neugrad).
Die entsprechende Funktionstaste ist **COS**

Beispiel:
```
DEGREE
COS 120                  -0.5
```

CUB x^3

Funktion: Kubikwert

Anmerkungen: Zeigt den Kubikwert des Arguments an.

Beispiel:
```
CUB 3                    27
```

CUR $\sqrt[3]{x}$

Funktion: Kubikwurzel

Anmerkungen: Zeigt den Kubikwurtel des Arguments an.

Beispiel:
```
CUR 125                  5
```

DEG dd°mm'ss" —> ddd.dddd°

Funktion: Umwandlung von Altgrad/Minuten/Sekunden in eine dezimale Form

Anmerkungen: Wandelt das Argument eines Winkels in der DMS-Form (Altgrad, Minuten, Sekunden) in die DEG-Form (Dezimal-Grad) um. In der DMS-Form repräsentiert die ganze Zahl die Gradzahl, die erste und zweite Stelle die Minuten und die dritte und vierte Stelle hinter dem Komma repräsentieren die Sekunden. Jede weitere Stelle gibt Dezimalsekunden an.
Die entsprechende Funktionstaste ist **->DEG**

Beispiel: DEG 30.5230 (30'52'30") 30.875

DMS ddd.dddd° —> dd°mm'ss"

Funktion: Umwandlung von Altgrad/Minuten/Sekunden in eine dezimale Form

Anmerkungen: Wandelt das Argument eines Winkels in der DMS-Form (Altgrad, Minuten, Sekunden) in die DEG-Form (Dezimal-Grad) um. In der DMS-Form repräsentiert die ganze Zahl die Gradzahl, die erste und zweite Stelle die Minuten und die dritte und vierte Stelle hinter dem Komma repräsentieren die Sekunden. Jede weitere Stelle gibt Dezimalsekunden an.
Die entsprechende Funktionstaste ist **->DMS**.

Beispiel: DMS 124.8055 124.48198 (124°48'19"8)

EXP e^x

Funktion: Exponentialfunktion

Anmerkungen: Zeigt den Wert von e (2.718281828... die Basis des natürlichen Logarithmus) erhoben zum Wert des numerischen Arguments an.
Die entsprechende Funktionstaste ist **e^x**.

Beispiel: EXP 1.2 3.320116923

FACT n!

Funktion: Fakultät n

Anmerkungen: Zeigt die Fakultät des eigenen Arguments an.

Beispiel: FACT 7 5040

HCS cosh x

Funktion: Hyperbolischer Cosinus

Anmerkungen: Zeigt den hyperbolischen Cosinus des numerischen Arguments an.

Beispiel: HCS 3 10.067662

HSN sinh x

Funktion: Hyperbolischer Sinus

Anmerkungen: Zeigt den hyperbolischen Sinus des numerischen Arguments an.

Beispiel: HSN 4 27.2899172

HTN tanh x

Funktion: Hyperbolischer Tangens

Anmerkungen: Zeigt den hyperbolischen Tangens des numerischen Arguments an.

Beispiel: HTN 0.9 0.71629787

INT

Funktion: Ganze Zahl

Anmerkungen: Zeigt die ganze Zahl des eigenen Arguments an.

Beispiel: INT -1.9 2

LN $\log_e x$

Funktion: Natürlicher Logarithmus

Anmerkungen: Zeigt den Logarithmus der Basis e (2.718281828...) des numerischen Arguments an.
Die entsprechende Funktionstaste ist **ln**.

Beispiel: LN 2 0.69314718

LOG $\log_{10} x$

Funktion: Dekadischer Logarithmus mit der Basis 10

Anmerkungen: Zeigt den Logarithmus zur Basis 10 des numerischen Arguments an.
Die entsprechende Funktionstaste ist **log**.

Beispiel: LOG 1000 3

MDF

Funktion: Modifizierungs-Funktion

Anmerkungen: Mit der Funktion für Dezimalstellen zeigt der Computer nur die festgelegte Anzahl der Dezimalstellen an, intern speichert er aber insgesamt immer alle Stellen. Daher können sich die angezeigten Daten von den internen Daten unterscheiden. Um die internen mit den angezeigten Daten in Übereinstimmung zu bringen, wird die Modifizierungs-Funktion verwendet (Siehe Seite 20)

Die entsprechende Funktionstaste ist **MDF**.

NCR $_nC_r=n!/r!(n-r)!$

Funktion: Kombination

Anmerkungen: Geben Sie die Werte als NCR (n,r) ein.

Beispiel: NCR (6,3) 20

NPR $_nP_r=n!/(n-r)!$

Funktion: Permutation

Anmerkungen: Geben Sie die Werte als NPR (n,r) ein.

Beispiel: NPR (6,3) 120

PI π

Funktion: PI

Anmerkungen: Pi ist eine numerische Pseudovariable mit dem Wert π. Die Anwendung von Pi ist identisch mit dem Gebrauch der π -Taste.

Beispiel: PI 3.141592654

POL (x,y) -> (r,ø)

Funktion: Umwandlung einer rechtwinkligen Koordinate in eine Polarkoordinate

Anmerkungen: Wandelt die numerischen Argumente in einem rechtwinkligen Koordinaten-Format in ein Polarkoordinaten-Format um.

Das erste Argument bezeichnet die Entfernung von der y-Achse und das zweite die von der x-Achse. Die umgewandelten Werte entsprechen der Entfernung und dem Winkel in den Polarkoordinaten und werden den festen Variablen Y und Z zugeordnet. Der umgewandelte Winkel hängt ab von der Einstellung (Altgrad,

Bogenmaß oder Neugrad).

Beispiel:

```
DEGREE
POL (8,6)                    10  (r=10)
Z                            36.86989765

                             (ø = 36.9°)
```

^ y^x

Funktion: x-te Potenz

Anmerkungen: Zeigt die x-te Potenz eines numerischen Werts an. Eingabe als y^x. Die entsprechende Funktionstaste ist y^x.

Beispiel:

```
4^2.5                        32
```

RCP 1/x

Funktion: Reziprokwert

Anmerkungen: Zeigt den Reziprokwert des numerischen Arguments an.

Beispiel:

```
RCP 4                        0.25
```

REC (r,ø) -> (x,y)

Funktion: Umwandlung polarer in rechtwinklige Koordinaten

Anmerkungen: Wandelt numerische Argumente im Polarkoordinaten-Format in ein rechtwinkliges Koordinaten-Format um.

Das erste Argument bezeichnet die Entfernung und das zweite den Winkel. Die Winkelangabe hängt ab von der Einstellung des Computers (DEG, RAD oder GRAD). Die umgewandelten Werte, die den Entfernungen von der x-Achse und der y-Achse entsprechen, werden den festen Variablen Y und Z zugewiesen.

Beispiel:

```
DEGREE
REC (12,30)                  10.39230485 (x ≈ 10.4)
Z                            6 (y = 6)
```

RND

Funktion: Zufallszahl

Anmerkungen: Siehe RND und RANDOMIZE im Basic-Kommando-Lexikon.

SGN

Funktion: Vorzeichen des Arguments

Anmerkungen: Zeigt den auf das Vorzeichen des Arguments basierenden Wert an.

Wenn x > 0, wird 1 angezeigt.
Wenn x < 0, wird -1 angezeigt.
Wenn x = 0, wird 0 angezeigt.

SIN sin x

Funktion: Sinus

Anmerkungen: Zeigt den Sinus des Winkelarguments an. Der angezeigte Wert hängt von der Einstellung ab (Altgrad, Bogenmaß oder Neugrad). Die entsprechende Funktionstaste ist **sin.**

Beispiel:
DEGREE
SIN 30 0.5

SQR √x

Funktion: Quadratwurzel

Anmerkungen: Zeigt die Quadratwurzel des Arguments an.
Die entsprechende Funktionstaste ist √**.**

Beispiel: SQR 3 1.732050808

SQU x^2

Funktion: Quadrat

Anmerkungen: Zeigt das Quadrat des Arguments an.
Die entsprechende Funktionstaste ist **x^2.**

Beispiel: SQU 4 16

TAN tan x

Funktion: Tangens

Anmerkungen: Zeigt den Tangens des Winkelarguments an. Der angezeigte Wert hängt von der Einstellung ab (Altgrad, Bogenmaß oder Neugrad).
Die entsprechende Funktionstaste ist **tan.**

Beispiel: DEGREE
TAN 45 1

TEN 10^x

Funktion: Antilogarithmus

Anmerkungen: Zeigt den Wert von 10 (die Basis des dekadischen Logarithmus) erhoben auf den Wert des eigenen numerischen Arguments.
Die entsprechende Funktionstaste ist **10^x**.

Beispiel: TEN 3 1000

&H

Funktion: Umformung von hexadezimal nach dezimal

Anmerkungen: Wandelt einen hexadezimalen Wert in einen dezimalen um.

Beispiel: &H F82 3970

5 BASIC

5.1 BEGRIFFE UND AUSDRÜCKE IN BASIC

Zeichenfolgen-Konstanten

Der Computer ist in der Lage, außer Zahlen auch Buchstaben und spezielle Symbole in vielfacher Weise zu verarbeiten. Diese Buchstaben, Zahlen und speziellen Symbole werden Zeichen genannt.

Im BASIC wird eine Folge von Zeichen als Zeichenfolge bezeichnet. Damit der Computer den Unterschied zwischen einer Zeichenfolge und anderen Programmteilen, wie z.B. Befehlen oder Variablen-Bezeichnungen, erkennen kann, muss man die Zeichen, die zu einer Zeichenfolge gehören, in Anführungszeichen (") einschließen. Zur Verwendung von Anführungszeichen als Zeichen wird "CHR$&H22" eingegeben.

Es folgen einige Beispiele für Zeichenfolgen-Konstanten:

"HELLO"
"Goodbye"
"SHARP COMPUTER"

Die folgenden Beispiele werden nicht als Zeichenfolgen-Konstanten akzeptiert:

"COMPUTER	Anführungszeichen am Ende fehlen.
"VALUE OF "A" IS"	Anführungszeichen dürfen nicht innerhalb einer Zeichenfolge benutzt werden.

Hexadezimalzahlen

Das Dezimalsystem ist nur eines von verschiedenen Zahlensystemen. Ein anderes, dessen Bedeutung im Zusammenhang mit Computern stark zugenommen hat, ist das Hexadezimalsystem. Das Hexadezimalsystem basiert auf der Zahl 16 statt auf der Zahl 10. Um hexadezimale Ziffern zu schreiben, benutzt man die Ziffern 0 bis 9, sowie sechs weitere "Ziffern" A, B, C, D, E und F. Diese entsprechen den Zahlen 10, 11, 12, 13, 14 und 15, Wenn der Computer eine Zahl als hexadezimal auffassen soll, setzen Sie ein UND-Zeichen (&) und "H" vor die Zahl:

&HA	= 10
&H10	= 16
&H100	= 256
&HFFFF	= 65535

5.2 Variablen

Computer sind aus einer Vielzahl von kleinsten Speichereinheiten aufgebaut, genannt Bytes, Jedes Byte kann man sich als einzelnes Zeichen vorstellen. Das Wort "Byte" erfordert beispielsweise vier Speicherbytes, weil es vier Buchstaben enthält. Um herauszufinden, wie viele Bytes zum Arbeiten zur Verfügung stehen, geben Sie einfach im RUN-Modus den Befehl FRE ein. Die angezeigte Zahl gibt an, wie viel Bytes zum Programmieren frei sind.

Dieses Verfahren funktioniert gut für Worte, ist aber zum Speichern von Zahlen sehr unzulänglich. Aus diesem Grund werden Zahlen in codierter Form gespeichert. Aufgrund dieser Codierung ist der Computer in der Lage, auch lange Zahlen in nur 8 Bytes zu speichern, Die größtmögliche Zahl, die gespeichert werden kann, ist +9.999999999E + 99. Die kleinste Zahl ist 1.E -99. So erhalten Sie einen recht großen Zahlenbereich zum Arbeiten. Wenn jedoch das Ergebnis der Rechnung diesen Rahmen übersteigt, teilt der Computer dies mit, indem er eine Fehlermeldung auf dem Display ausgibt.

Beispiel:

R=556

Bislang haben wir uns nur mit numerischen Variablen befasst. Wie speichert man nun alphanumerische Zeichen? Grundsätzlich ist das Prinzip das gleiche, aber damit der Computer nun alphabetische Zeichen? Grundsätzlich ist das Prinzip das gleiche, aber damit der Computer den Unterschied zwischen den beiden Variablentypen erkennen kann, muss nun ein $ zum Namen der Variablen gesetzt werden. Wir wollen z.B. das Wort BYTE unter der Variablen B$ speichern. Beachten Sie das $-Zeichen hinter dem B. Dies sagt dem Computer, dass der Inhalt der Variablen B$ alphanumerisch bzw. eine Zeichenfolge ist. Damit dies klarer wird, hier ein Beispiel:

B$=“BYTE“

Arten von Variablen

Die Variablen, mit denen der Computer arbeitet, sind folgendermaßen aufgegliedert:

Numerische Variablen:
- Feste numerische Variablen (A bis Z)
- Einfache numerische Variablen (AB, C1, usw.)
- Numerische Fe!dvariablen

Zeichenfolge-Variablen:
- Feste Zeichenfoige-Variablen (A$ bis Z$)
- Einfache Zeichenfolge-Variable (BB$, C2$, etc.)

Zeichenfolge-Feldvariablen

Feste Variablen

Bei der ersten Art, den festen Variablen, handelt es sich um Variablen mit bereits zugewiesenen Speicherbereichen. Mit andern Worten, egal, wie viel Speicherplatz das Programm in Anspruch nimmt, ihnen stehen immer mindestens 26 Variable zur Speicherung von Daten offen. Es handelt sich dabei um zwei Arten von Variablen: numerische oder Zeichenfolge-Variable (alphanumerische Zeichen). Die zugewiesenen Speicherbereiche haben eine Kapazität von acht Bytes und können jeweils nur für eine Art von Variablen verwendet werden.

Beispiel:

```
A= 123
A$
```

Die folgende Meldung wird ausgegeben:

```
ERROR 91
```

Dies bedeutet, dass numerische Daten in einen als "A" bezeichneten Speicherbereich zugewiesen wurden und danach der Computer die Anweisung erhielt, diese Informationen wieder als Zeichenfolge auszugeben. Der Computer ist dadurch verwirrt und gibt eine Fehlermeldung aus. CLS/CA drücken um die Fehlermeldung zu löschen. Nun wird folgendes eingegeben;

```
A $ = " A B C "
A
```

Wieder ist der Computer verwirrt und gibt die Fehlermeldung 91aus. Die Variablenbezeichnung A entspricht im Speicher dem gleichen Bereich wie die Variablenbezeichnung A$, weiterhin entspricht B dem Bereich B$ und so weiter für alle Buchstaben des Alphabetes.

Jede feste Zeichenvariable kann bis zu 7 Zeichen und Symbole enthalten.

Einfache Variablen

Einfache Variablenbezeichnungen werden durch alphanumerische Zeichen charakterisiert, z.B. AB oder C8$, Anders als feste Variablen haben die einfachen Variablen keinen fest im Speicher reservierten Bereich. Der Speicherbereich für einfache Variablen wird automatisch bereitgestellt (Im Programm oder im Datenbereich), sobald eine einfache Variable erstmalig benutzt wird.

Da für einfache numerische und einfache Zeichenfolge-Variablen verschiedene Speicherbereiche vorgesehen sind, können Variablen mit dem gleichen Namen, z.B. AB und AB$ gleichzeitig benutzt werden.

Für die Bezeichnungen von einfachen Variablen können alphanumerische Zeichen verwendet werden; das erste Zeichen muss allerdings immer ein Großbuchstabe sein. Zur Bestimmung eines Variablennamens können zwei oder mehr Zeichen verwendet werden, der Computer liest allerdings nur die ersten beiden.

Hinweise:
- Die im Computer residenten Bezeichnungen für Funktionen und BASIC-Befehle, z,B. PI, IF, TO, ON, SIN u.a., können nicht für Variablenbezeichnungen verwendet werden.
- Jede einfache Zeichenvariable kann bis zu 16 Zeichen und Symbole enthalten. Jede feste Zeichenvariable kann bis zu 7 Zeichen und Symbole enthalten.

Feldvariablen

In einigen Fällen ist es sinnvoll, Zahlen in organisierten Gruppen zu verarbeiten, z.B. eine Tabelle der Fußballergebnisse oder eine Steuertabelle. Im BASIC werden solche Gruppen Felder genannt. Ein Feld kann eindimensional sein, z.B. eine Liste, es kann aber auch zweidimensional sein, z.B. eine Tabelle.

Um ein Feld zu definieren, benutzt man den DIM-Befehl (Kürzel für Dimension). Felder müssen vor Gebrauch immer definiert werden. (Dies war nicht der Fall bei den Einwert-Variablen, die wir bislang benutzt haben.) Die Form für die DIMensionierung numerischer Felder ist:

DIM Name der Feldvariable (Größe)

Dabei bedeutet:

Name der Feldvariable ist eine Bezeichnung der Variablen gemäß den oben besprochenen Benennungsregeln für numerische oder Feld-Variablen.

Größe bedeutet die Anzahl der Speicherplätze und sollte eine Zahl im Bereich von 0 bis 255 sein, Bei der Zuweisung einer Zahl für die Größe wird ein Speicherplatz mehr als zugewiesen bereitgestellt.

Beispiel für zugelassene Befehle für numerische oder Zeichenfolge-Dimensionierung:

```
DIM X(5)      ->      X(0), X(1), X(2), X(3), X(4), X(5)
DIM AA(24)
DIM Q5(0)
```

Der erste Befehl schafft ein Feld X mit 6 Speicherplätzen. Der zweite Befehl baut ein Feld AA mit 25 Speicherplätzen auf, der dritte ein Feld mit einem Speicherplatz, was unsinnig (zumindest für Zahlen) ist, da man ebenso gut eine einwertige numerische Variable definieren könnte.

Es ist wichtig zu wissen, dass eine Feldvariable X und eine Variable X vom Computer unterschieden werden. Das erste X bezeichnet eine Serie von numerischen Speicherplätzen, das zweite einen einzelnen und getrennten Speicherplatz.

Nachdem Sie nun wissen, wie man Felder aufbaut, mögen Sie sich fragen, wie man die einzelnen Speicherplätze anspricht. Da die gesamte Gruppe unter einem einzigen Namen abgelegt ist, sprechen wir einen einzelnen Speicherplatz ("Element" genannt) an, indem wir an den Namen der Gruppe eine Zahl in Klammern anschließen. Diese Zahl wird "Index" genannt. So müsste man z.B., um die Zahl 8 an fünfter Stelle in unserem (vorher definierten) Feld X unterzubringen, schreiben:

```
X(4) = 8
```

Wenn Sie die Zahl 4 verwirrt, bedenken Sie, dass die Nummerierung der Elemente in einem Feld mit Null beginnt und dann bis zu der in der DIM-Anweisung definierten Anzahl von Elementen fortläuft.

Der besondere Vorteil von Feldern liegt in der Möglichkeit, einen längeren Ausdruck oder eine Variable als Index zu benutzen.

Zur Bestimmung einer Zeichenfolge-Feldvariable wird eine etwas andere Form der DIM-Anweisung verwendet:

DIM Name der Zeichenfolge-Variable (Größe) *Länge

Dabei bedeutet:

Name der Zeichenfolge-Variable ist eine Bezeichnung für die Variable, die den bereits besprochenen Regeln für Bezeichnungen entspricht.

Größe bedeutet die Anzahl der Speicherplätze und sollte eine Zahl im Bereich von 0 bis 255 sein. Bei der Zuweisung einer Zahl für die Größe wird ein Speicherplatz mehr als zugewiesen bereitgestellt.

***Länge** ist optional. Falls diese Option verwendet wird, wird damit die Länge für jede Zeichenfolge dieser Feldvariablen bestimmt. Die Länge muss durch eine Zahl von 1 bis 255 angegeben werden, Falls hier keine Eingabe erfolgt, wird die Grundeinstellung von 16 Zeichen für die Zeichenfolge angenommen.

Beispiele für zulässige Bestimmungen von Zeichenfolge-Feldvariablen:

```
DIM X$(4)
DIM NM$(10)x10
DIM IN$(1)*255
DIM R$(0)*26
```

Beim ersten Beispiel wird ein Feld von fünf Zeichenfolgen geschaffen, in denen jeweils 16 Zeichen gespeichert werden können. Beim zweiten Beispiel wird durch die DIM-Anweisung ein Feld NM bestimmt, dass 11 Zeichenfolgen mit jeweils 10 Zeichen enthält. Die eindeutige Zuweisung von Zeichenfolgen, die kürzer als die Grundeinstellung von 16

Zeichen sind, hilft bei der Einsparung von Speicherplatz, Beim dritten Beispiel wird ein Zwei-Elemente-Feld mit 255 Zeichenfolgen bestimmt und beim letzten Beispiel eine einzelne Zeichenfolge mit 26 Zeichen.

Neben den einfachen Feldvariablen, die bisher besprochen wurden, kann der Computer auch "zweidimensionale" Felder bearbeiten. Bei einem eindimensionalen Feld wird eine Folge von Daten in einer einzigen Spalte aufgelistet. Ein zweidimensionales Feld entspricht einer Tabelle mit Zeilen und Spalten. Zweidimensionale Felder werden durch die folgende Anweisung bestimmt:

DIM Bezeichnung des numerischen Feldes (Zeilen, Spalten)

oder

DIM Bezeichnung der Zeichenfolge-Feldvariable (Zeilen, Spalten) *Länge

Dabei bedeutet:

Zeilen bedeutet die Anzahl der Zeilen in einem Feld. Es muss eine Zahl im Bereich von 0 bis 255 sein. Bei der Zuweisung einer Zahl für die Zeilen wird eine Zeile mehr als zugewiesen bereitgestellt.

Spalten bedeutet die Anzahl der Spalten in einem Feld Es muss eine Zahl im Bereich von 0 bis 255 sein. Bei der Zuweisung einer Zahl für die Spalten wird eine Spalte mehr als zugewiesen bereitgestellt.

Die folgende Tabelle illustriert die Speicherplätze, die sich aus der Anweisung DIM T(2,3) und den Indizes, die zu den jeweiligen Speicherplätzen gehören, ergeben (in diesem Beispiel zwei Zahlen):

	Spalte 0	Spalte1	Spalte 2	Spalte 3
Zeile 0	T (0,0)	T (0,1)	T (0,2)	T (0,3)
Zeile 1	T (1,0)	T (1,1)	T (1,2)	T (1,3)
Zeile 2	T (2,0)	T (2,1)	T (2,2)	T (2,3)

Hinweis:
Zweidimensionale Felder nehmen viel Speicherplatz in Anspruch. Z.B. benötigt ein Feld mit 25 Zeilen und 35 Spalten 875 Speicherplätze!

Die folgende Tabelle zeigt die Anzahl der Bytes, die zur Definierung jeder einzelnen Variable benötigt werden, sowie die Anzahl der für jeden einzelnen Programm-Befehl erforderlichen Bytes.

Variablen-Typ	Anzahl der verwendeten Byte	
	Variablen-Name	Daten
Numerische Variable Numerische Feldvariable	7 Byte	8 Byte
Zeichenfolge-Variable	7 Byte	16 Byte
Zeichenfolge-Feldvariable	7 Byte	Zugewiesene Anzahl

* Wenn zum Beispiel DIM Z$(2,3)*10 ist, werden 12 Variable mit einem Speicherplatz für jeweils 10 Zeichen bereitgestellt. Dafür werden 127 Byte benötigt: 7 Byte (Variablenname) + 10 Byte (Anzahl der Zeichen) x 12.

Element	Zeilennummer	Befehl & Funktion	ENTER und andere
Anzahl der verwendeten Byte	3 Byte	2 Byte	1 Byte

5.3 Programm-Dateien auf der RAM-Disk

Programm-Dateien stellen den wichtigsten Aspekt beim Gebrauch des Computers dar. Ein Teil des internen Speichers des Computers kann als RAM-Disc verwendet werden. Auf der RAM-Disc gespeicherte Programme müssen vor der Ausführung in den Programmdaten-Bereich (Benutzerbereich) geladen werden. (Siehe BASIC-KOMMANDO LEXIKON für Hinweise zu den Befehlen SAVE, LOAD, KILL und FILES.)

5.4 Daten-Dateien auf der RAM-Disk

Es können Daten-Dateien auf der RAM-Disc gespeichert werden. Auf der RAM-Disc abzulegende Dateien müssen vor der ersten Nutzung im TEXT-Modus unter Rfile angelegt werden. (Siehe TEXT-Modus unter Rfile)

5.5 Dateinamen

Vor dem Speichern auf der RAM-Disc, muss die Datei einen Namen bekommen. Dieser Name wird benutzt zum Laden der Programm-Datei in den Computerspeicher oder beim Öffnen einer Datei-Datei mit dem Befehl OPEN. Der Dateiname ist beliebig und kann aus bis zu 8 der folgenden Zeichen bestehen:

A — Z, a — z, 0 — 9, 1, $,%, 8., ', (,), (, }, —, @

5.6 Dateinamen-Erweiterung

Die Erweiterung ist eine zusätzliche Art, Datei-Typen (wie z.B. BASIC-Programm-Dateien, Daten-Dateien oder Text-Dateien) zu identifizieren. Die Erweiterung besteht aus drei Zeichen, die am Ende des Dateinamens durch einen Punkt getrennt angehängt werden. Die Erweiterung wird spezifiziert, wenn eine Datei gesichert wird.

BASIC-Programme erhalten automatisch die Erweiterung .BAS, wenn sie mit dem SAVE-Kommando gesichert werden. Beim Laden in den Speicher mit dem LOAD-Kommando muss die .BAS-Erweiterung nicht spezifiziert werden.

Wird mit den Anweisungen FILES oder LFILES ein Verzeichnis der Dateien der RAM-Disc aufgelistet, erscheinen die BASIC-Programme mit der .BAS-Erweiterung, unabhängig davon, ob irgendeine andere Erweiterung bei der Sicherung der Datei bestimmt wurde.

5.7 Ausdrücke

Ein Ausdruck ist eine Kombination von Variablen, Konstanten und Operatoren, die auf einen einzigen Wert zusammengefasst werden. Die Rechenbeispiele, die Sie vorher eingegeben haben, waren Beispiele für solche Ausdrücke. Ausdrücke sind ein wesentlicher Bestandteil von BASIC-Programmen. Zum Beispiel kann ein Ausdruck eine Formel sein, mit der das Ergebnis einer Gleichung errechnet wird oder ein Test zur Bestimmung des Verhältnisses zwischen zwei Größen oder ein Mittel, um eine Reihe von Zeichenfolgen zu formatieren.

5.8 Numerische Ausdrücke

Ein numerischer Ausdruck wird in der gleichen Weise konstruiert, wie Sie komplexe Rechenbefehle eingegeben haben. Numerische Ausdrücke können jede aussagefähige Kombination von numerischen Konstanten, numerischen Variablen und numerischen Operatoren beinhalten. Es gibt folgende numerische Operatoren:

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
^	Exponent

Dies sind die arithmetischen Rechenzeichen. Beispiele für zulässige numerische Ausdrücke:

```
(A*B)^2
A(2,3)+A(3,4)+5.0-C
(A/B)*(C+D)
```

5.9 Zeichenfolge-Ausdrücke

Zeichenfolge-Ausdrücke sind den numerischen Ausdrücken ähnlich, allerdings gibt es nur einen einzigen Zeichenfolgen-Operator: die Verkettung (+), Das benutzte Symbol ist dasselbe wie das Pluszeichen. Wird es mit einem Zeichenfolge-Paar benutzt, knüpft das + die zweite Zeichenfolge an das Ende der ersten an und schafft dadurch eine längere Zeichenfolge. Wenn Sie eine komplexere Zeichenfolgen-Verkettung und andere Zeichenfolgen-Operationen vornehmen, bedenken Sie bitte, dass der Computer nur 255 Zeichen annimmt.

Hinweis:
Zeichenfolge-Einheiten und numerische Einheiten können nicht in demselben Ausdruck definiert werden, es sei denn, man benutzt eine der Funktionen, die Zeichenfolge-Werte in numerische Werte umwandeln oder umgekehrt:

"15"+10	ist unzulässig.
"15"+"10"	ist "1510", nicht "25".

5.10Verhältnis-Ausdrücke

Ein Verhältnis-Ausdruck vergleicht zwei Ausdrücke und gibt an, ob das festgestellte Verhältnis wahr oder unwahr ist. Die Verhältnis-Operatoren sind:

>	größer als
>=	größer oder gleich
=	gleich
<>	ungleich
<=	kleiner oder gleich
<	kleiner als

Die folgenden Ausdrücke werden als Verhältnis-Ausdrücke akzeptiert:

```
A<B
C(1,2)>=5
D(3)<>8
```

Wenn A gleich 10 wäre, B gleich 12, C(1,2) gleich 6 und D(3) gleich 9, wären alle diese Ausdrücke wahr.

Zeichenfolgen können ebenfalls mit Hilfe von Verhältnis-Ausdrucken verglichen werden. Die beiden Zeichenfolgen werden Zeichen für Zeichen gemäß dem Wert ihres ASCII-Codes verglichen (siehe Anhang H: Tabelle der Zeichencodes auf Seite 284) Seite st eine Zeichenfolge kürzer als eine andere, wird 0 oder NULL in die freibleibenden Positionen eingesetzt. Alle folgenden Beispiele sind wahr:

```
"ABCDEF"="ABCDEF"
"ABCDEF"<> ABCDE"
"ABCDEF">"ABCDE"
```

Verhältnis-Ausdrücke beurteilen nach wahr oder unwahr. Beim Computer wird wahr durch -1 angegeben, unwahr durch eine 0.

5.11Logische Ausdrücke

Logische Operationen benutzen zum Bau von Verbindungen zwischen Verhältnis-Ausdrücken die Funktionen der Booleschen Algebra **AND** (und), **OR** (oder), **XOR**(exklusiv-oder) und **NOT** (nicht). Die logischen Operationen in einem einzigen Ausdruck werden nach arithmetischen und Verhältnis-Operationen bewertet.

Auf diese Weise können logische Operatoren benutzt werden, die ihre Programmentscheidungen auf Grund mehrerer Bedingungen unter Verwendung der

Anweisungen IF.T..H EN treffen.

Beispiel:

IF A<=32 AND B>=90 THEN 150

Diese Anweisung bewirkt, dass die Ausführung auf Zeile 150 springt, wenn der Wert der numerischen Variable A kleiner oder gleich 32 ist und wenn gleichzeitig der Wert der numerischen Variablen B größer oder gleich 90 ist.

IF X<>I3 OR Y=0 THEN 50

Diese Anweisung bewirkt, dass die Ausführung auf Zeile 50 springt, es sei denn die Variable X hat den Wert 13, oder die Variable Y ist ungleich 0.

In einer logischen Operation, in der zwei Ziffern im Bereich -32768 bis +32767 involviert sind, werden die zwei Ziffern in 16-stellige Binärzahlen umgewandelt (in die Zweierkomplementform) und die logische Verbindung wird dann für jedes entsprechende Bit-Paar in den zwei Ziffern bewertet.

Die Ergebnisse, angezeigt durch die logischen Operatoren für diese Bit-Bewertungen, werden im folgenden aufgelistet:

5.12Klammerung und Vorrang der Operatoren

Bei der Bearbeitung komplexer Ausdrücke folgt der Computer einer Reihe definierter Prioritäten, die bestimmen, in welcher Reihenfolge die Operatoren bearbeitet werden.

5+2*3 kann sein

5+2=7 oder 2*3=6

7*3 =21 6+5=11

Damit Sie sich nicht alle Regeln merken müssen und damit Sie Ihre Programme eindeutiger gestalten, benutzen Sie immer Klammern, um die Reihenfolge der Bearbeitung von Operatoren vorzugeben. Das obige Beispiel wird eindeutig, wenn Sie schreiben:

(5+2) 3 oder 5+(2*3)

6 PROGRAMMIEREN MIT BASIC

Im vergangenen Abschnitt haben wir einige Begriffe und Ausdrücke der Programmiersprache BASIC kennengelernt. In diesem Abschnitt wollen wir nun diese Elemente benutzen, um Programme zu schreiben. Wir möchten noch einmal darauf hinweisen, dass dieses Handbuch nicht als Einführung in die BASIC- Programmierung verstanden werden soll. Dieser Abschnitt soll Sie lediglich in den besonderen Gebrauch des BASIC auf diesem Computer einführen.

6.1 Programme

Ein Programm besteht aus einer Reihe von an den Computer gerichteten Befehlen. Denken Sie daran, dass der Computer nur eine Maschine ist, die genau die Operationen durchführt, die Sie angeben. Sie als Programmierer sind dafür verantwortlich, dass korrekte Befehle eingegeben werden.

6.2 BASIC-Anweisungen

Der Computer setzt Programme entsprechend einem bestimmten Format um. Dieses Format wird Anweisung genannt. Sie geben die BASIC-Anweisungen immer nach einem bestimmten Muster ein. Anweisungen müssen mit einer Zeilennummer beginnen:

Beispiel:
10: INPUT A
20: PRINT A*A
30: END

6.3 Zeilennummern

Jede Programmzeile muss eine eigene Nummer haben, und zwar muss diese eine ganze Zahl zwischen 1 und 65279 sein. Zeilennummern sind die Bezugspunkte des

Computers. Sie geben dem Computer an, in welcher Reihenfolge er ein Programm abarbeiten muss. Es ist nicht erforderlich, dass Sie die Programmzeilen folgerichtig eingegeben haben (obwohl das sicher weniger verwirrend sein dürfte, besonders wenn Sie noch Anfänger sind). Der Computer beginnt beim Durcharbeiten eines Programms immer mit der niedrigsten Zeilennummer und arbeitet die folgenden in ansteigender Reihenfolge ab.

Mit dem AUTO-Kommando können Sie automatisch Zeilennummern einfügen. Jedes Mal, wenn Sie die Enter-Taste drücken, wird eine neue Zeilennummer, in der richtigen aufsteigenden Reihenfolge, automatisch eingefügt. (Eine vollständige Beschreibung dieser nützlichen Funktion finden Sie im BASIC KOMMANDO LEXIKON.)

Beim Programmieren ist es sinnvoll, genug Raum für spätere Einschübe zwischen den einzelnen Zeilen zu lassen (10, 20, 30,... 10, 30, 50, usw.). Dies ermöglicht den Einschub von zusätzlichen Zeilen, falls notwendig.

Wenn Sie gleiche Zeilennummer mehr als einmal benutzen, wird die ältere Zeile gelöscht, wenn Sie eine neue mit derselben Nummer eingeben,

6.4 Programme mit Labels

Es wird häufiger vorkommen, dass Sie mehrere verschiedene Programme zur gleichen Zeit im Speicher abspeichern wollen. (Vergessen Sie nicht, verschiedene Zeilennummern zu vergeben.) Um ein Programm mit einem RUN- oder einem GOTO-Befehl zu starten, müssen Sie sich normalerweise an die erste Zeilennummer eines jeden Programms erinnern. Aber es gibt eine einfachere Möglichkeit! Sie können jedes Programm alphanumerisch benennen und starten.

Die erste Zeile jedes Programms benennen, das als Referenz verwendet werden soll. Das Label besteht aus einem Buchstaben und alphanumerischen Zeichen;

davor muss entweder ein * stehen oder der Ausdruck muss in Anführungszeichen eingeschlossen sein, gefolgt von einem Doppelpunkt.

Beispiel:
10; *A: PRINT "FIRST"
20: END
80: "B": PRINT "SECOND"
90: END

Die Formate *Label oder "Label" können beide verwendet werden. Das Format *Label wird allerdings empfohlen, da es schneller zur Ausführung kommt und in einer Programmauflistung besser erkennbar ist.

6.5 BASIC-Kommandos (Befehle)

Alle BASIC-Anweisungen müssen Befehle enthalten. Diese Befehle sagen dem Computer, welche Operation er durchführen soll. Ein Befehl ist immer ein Programmbestandteil, insofern erfolgt die Operation nicht direkt. Einige Anweisungen erfordern oder erlauben den Gebrauch eines Operanden.

Beispiel:
10: DATA "HELLO"
20: READ B$
30: PRINT B$
40: END

Operanden informieren den Computer darüber, auf welche Daten sich die vom Befehl angeordnete Operation bezieht. Einige Befehle erfordern Operanden, bei andern Befehlen. sind sie fakultativ. Einige Befehle erlauben keine Operanden. (Im BASIC KOMMANDO LEXIKON finden Sie die BASIC-Befehle und ihre Anwendung auf dem Computer.)

Hinweis:
Kommandos, Funktionen und Variable müssen in Großbuchstaben eingegeben werden.

6.6 Direkt-Kommandos

Direkt-Kommandos sind Anweisungen an den Computer, die außerhalb eines Programms eingegeben werden. Sie fordern den Computer auf, bestimmet Vorgänge durchzuführen oder einen bestimmten Modus zu setzen, der dann wieder die Art der Programmbearbeitung determiniert.

Direkt-Kommandos haben unmittelbare Wirkung — sobald Sie die Eingabe des Direkt-Kommandos beendet haben (durch Betätigen der ENTER-Taste), wird die Anweisung ausgeführt. Direkt-Kommandos gehen keine Zeilennummern voraus:

RUN
NEW
RADIAN

6.7 Modi (Betriebsarten)

Sie erinnern sich sicherlich, dass Sie, als Sie den Computer als Rechner benutzt haben, im im RUM-Modus gearbeitet haben. Der RUN-Modus wird ebenfalls gebraucht, um die von ihnen geschriebenen Programm abzuarbeiten.

Der PRO-Modus wird gewählt, wenn Sie Programme eingeben oder editieren wollen.

6.8 Fehlersuche

Nachdem Sie ein neues BASIC-Programm eingeben haben, wird es in der Regel beim ersten Startversuch nicht laufen. Selbst wenn Sie ein Programm nur abtippen, von dem Sie wissen, dass es korrekt ist, wie z.B. die in diesem Handbuch vorgestellten, dürfte ihnen normalerweise mindestens ein Tippfehler unterlaufen sein. Handelt es sich um ein längeres Programm, wird es oft auch mindestens einen logischen Fehler enthalten.

Es folgen einige grundsätzliche Hinweise, wie Sie solche Fehler finden und korrigieren. Sie lassen das Programm laufen und erhalten eine Fehlermeldung:

1. Schalten Sie zurück in den PRO-Modus und benutzen Sie Cursortasten ↓ oder ↑ um die fehlerhafte Zeile ins Display zu rufen. Der Cursor befindet sich an der Stelle, an welcher der Fehler auftrat.
2. Wenn Sie aus der Art, wie die Programmzeile geschrieben ist, keinen offensichtlichen Syntaxfehler entnehmen können, kann das Problem auch an den von Ihnen verwendeten Werten liegen. So erzeugt beispielsweise CHR$(A) eine Leerstelle, wenn A den Wert 1 hat. Überprüfen Sie die Werte der von Ihnen verwendeten Variablen, indem Sie entweder im RUN- oder im PRO-Modus die einzelnen Variablennamen gefolgt von ENTER eingeben.

Sie lassen das Programm mit RUN laufen und erhalten keine Fehlermeldung, aber das Programm tut nicht, was Sie von ihm erwarten:

1. Überprüfen Sie das Programm Zeile für Zeile unter Verwendung von LIST und den ↓ und ↑-Tasten um herauszufinden, ob Sie das Programm korrekt eingegeben haben. Es ist erstaunlich, wie viele Fehler beim bloßen erneuten Durchsehen eines Programms gefunden werden können!
2. Versuchen Sie jede einzelne Zeile beim Durchlesen so zu interpretieren, als wären Sie ein Computer. Nehmen Sie einfache Werte und realisieren Sie die Operationen der einzelnen Zeilen, um herauszufinden, ob Sie die gewünschten Ergebnisse erhalten.
3. Fügen Sie eine oder mehrere zusätzliche PRINT-Anweisungen in das Programm ein, um wichtige Werte und wichtige Positionen zur Anzeige zu bringen. Benutzen Sie diese, um die korrekten Teile des Programms von den möglicherweise fehlerhaften zu isolieren. Diese Vorgehensweise ist auch nützlich um zu bestimmen, welche Teile eines Programms schon abgearbeitet wurden. Sie können den Programmablauf auch an kritischen Stellen vorübergehend mit STOP unterbrechen, um dann einzelne Variable zu überprüfen.
4. Verwenden Sie TRON (TRace ON) und TROFF (TRace OFF) als Direkt-Kommandos innerhalb des Programms, um den Programmablauf durch die einzelnen Zeilen hindurch verfolgen zu können. Halten Sie das Programm an kritischen Punkten an, um den Inhalt von wichtigen Variablen zu überprüfen. Dies ist zwar eine sehr langsame Methode, Fehler aufzuspüren, aber es ist manchmal die einzige.

Trace-Modus

Unabhängig davon, wie vorsichtig Sie auch programmieren, wird es vorkommen, dass Sie ein Programm schreiben, das nicht das macht, was Sie von ihm erwarten. Dann geht es darum, den Fehler zu finden. BASIC enthält deshalb eine spezielle Programmablauf-Routine, den "Trace"-Modus.

TRON (TRace ON) startet den Trace-Modus. Die TRON-Anweisung kann als Direkt-Kommando (im RUN-Modus) oder auch innerhalb eines Programms als Befehl verwendet werden. Wird TRON als Direkt-Kommando verwendet, informiert es den Computer, dass eine Ablaufverfolgung während der Ausführung aller nachfolgenden Programme durchgeführt wird. Die zu verfolgenden Programme werden dann ganz normal gestartet, mit Hilfe des GOTO- oder RUN-Kommandos. Wird TRON innerhalb eines Programms verwendet, so wird der Trace-Modus erst beim Erreichen der entsprechenden Zeile eingeschaltet. Wenn aus irgendeinem Grund die Zeile nie erreicht wird, bleibt der Trace-Modus ineffektiv.

Vorgehen bei der Fehlersuche

1. Die RUN-Betriebsart einstellen.
2. Eingabe vonTRON, um den Trace-Modus zu aktivieren.
3. Eingabe von RUN, um das Programm ausführen zu lassen. Nach jeder Zeile unterbricht der Computer die Ausführung und zeigt die Zeilennummer der gerade ausgeführten Zeile an.

4. Die [↓]-Taste drücken, um auf die zu prüfende Zeile zu gelangen. Durch Gedrückthalten der [↓]-Taste wird das Programm Schritt für Schritt ausgeführt. Beim Loslassen der Taste hält die Ausführung an. Der Inhalt der zu prüfenden Zeile kann durch Gedrückthalten der [↑]-Taste angesehen werden. (Beim Loslassen von [↑] erscheint die Befehlseingabezeile zur Fortsetzung der Ausführung im Trace-Modus die Taste [↓] drücken.)
5. Zur Weiterführung CONT eingeben. Wenn die Ausführung während der Dateneingabe mit dem INPUT-Befehl unterbrochen wurde, braucht zur Fortführung des Programms nur die ENTER-Taste gedrückt werden.
6. Den Trace-Vorgang weiterführen und prüfen, ob das Programm ordnungsgemäß ausgeführt wird, indem nach jeder Zeile die Reihenfolge der Ausführung und der Inhalt der Variablen überprüft wird. Wenn das Programm nicht ordnungsgemäß abläuft, muss die Logik verbessert werden.
7. Nach der Fehlersuche durch Eingabe von TROFF die Trace-Betriebsart wieder deaktivieren.

Beispiel:

```
10 INPUT "A=";A,"B =";B
20 C=A*2
30 D=B*3
40 PRINT "C=";C;" D=";D
50 END
```

Das Programm laufen lassen.

```
RUN-Modus
TRON                >
RUN                 A=–
8                   B=_
9
                          10:
                [↓]       20:
                [↓]       30:
                [↓]       C=16.     D=27.
                           40:
```

Wenn die Ausführung durch die BREAK-Taste unterbrochen wird, können die Variablen manuell abgerufen und ihre Werte überprüft werden. Durch Drücken von [↓] wird jeweils eine Zeile ausgeführt. Eingabe von CONT führt zur fortlaufenden Ausführung der Zeilen.

Hinweise:

- Wenn an der mit LOCATE spezifizierten Stelle ein Ergebnis oder eine andere Information ausgegeben wird, erscheint die folgende Zeilennummer auf der Zeile nach dieser Information. (Siehe BASIC KOMMANDO LEXIKON für Hinweise zum Befehl LOCATE)
- Wenn eine Variable manuell abgerufen oder eine manuelle Berechnung durchgeführt

wurde, nachdem eine Stelle mit dem LOCATE-Befehl zugewiesen wurde, wird diese Zuweisung gelöscht.
- Der Trace-Modus bleibt aktiv, bis TROFF eingegeben wird, die SHIFT+CA-Tasten gedrückt werden oder die Stromversorgung unterbrochen wird.
- Bei der Ausführung einer Kommentarzeile im Trace-Modus wird für diese Zeile keine Zeilennummer angezeigt. In diesem Fall bleibt die Nummer der zuletzt ausgeführten Zeile auf der Anzeige.

Zur Fehlersuche durch Unterbrechen eines laufenden Programms gehen Sie folgendermaßen vor:
• Drücken der BREAK-Taste während der Ausführung des Programms.
• Eingabe des STOP-Befehls an der entsprechenden Stelle.

Die Meldung für den Abbruch erfolgt und die Ausführung wird unterbrochen.
Danach:
1. Die Inhalte der Variablen werden manuell überprüft.
2. Drücken der ↓-Taste zur Ausführung der Anweisungen, Zeile für Zeile CONT eingeben, um auf den vorherigen Vorgang zurückzugehen.

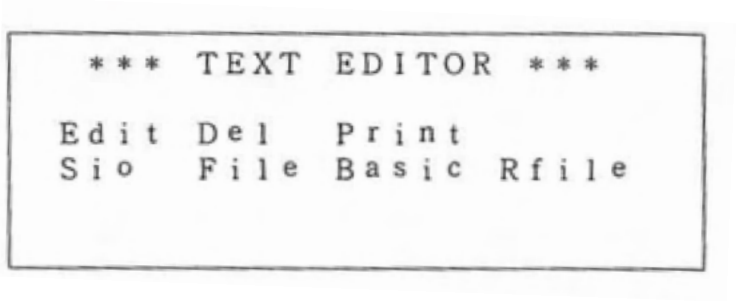
Ein Programm, welches mit der BREAK-Taste oder dem STOP-Befehl unterbrochen wurde, kann mit der ↓-Taste Zeile für Zeile ausgeführt werden.

7 TEXT-MODUS

In der TEXT-Betriebsart (Text-Editor) können Sie Programme (BASIC, C, Assembler oder CASL) im ASCII-Format schreiben und editieren. Genauso können Daten-Dateien angelegt, editiert oder gelöscht werden. Programme wie auch Daten können auf der RAM-Disc gespeichert werden oder auch über die serielle E/A-Schnittstelle ein- oder ausgeben werden.

BASIC-Anweisungen für den Computer werden in einem 2-Byte-Format gespeichert, dem sog. "Zwischencode". Da dieser Code je nach verwendeter Hardware bzw. BASIC-Interpreter unterschiedlich ist, kann er nicht für die Kommunikation zwischen Personal Computern oder anderen Geräten verwendet werden, Der ASCII-Code wird allgemein für die Datenkommunikation zwischen Personal Computern verwendet, da die ASCII-Darstellung von alphanumerischen Zeichen und Grundsymbolen die gleiche ist, unabhängig von der verwendeten Hardware. Mit der TEXT-Betriebsart dieses Computers können Sie Programme in ASCII schreiben, editieren und speichern. Die Programme können auch vom Zwischencode (BASIC) in ASCII konvertiert werden und umgekehrt. Dieser Abschnitt beschreibt die Funktionen der TEXT-Betriebsart.

Wenn Sie die TEXT-Taste drücken, sieht man einen Bildschirm wie auf der rechten Seite.



Der Textmodus kann jederzeit durch Wechsel in einen anderen Modus beendet werden (RUN, PRO, ASMBL, CASL, C). Bereits eingegeben Daten gehen dabei nicht verloren und können durch erneutes Drücken von TEXT + E (für Edit) weiter bearbeitet werden.

Um von einem beliebigen Untermenu des TEXT-Modus in das Hauptmenu zu gelangen, drücken Sie bitte einfach nochmals die TEXT-Taste. Um in das jeweils nächst höhere Menu zurück zu gelangen die Break-Taste (ON) drücken.

Funktionen des Text-Modus

In der TEXT-Betriebsart stehen die folgenden Funktionen zur Verfügung:

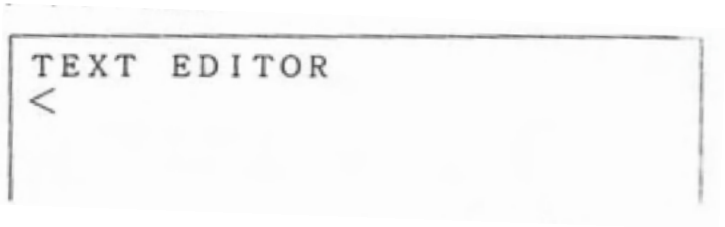
TEXT -> **TEXT-Betriebsart**

- **E**dit Programmieren und Editieren von Programmen oder Dateien
- **D**el Löschen des Programms oder der Dateien im Editor
- **P**rint Ausgabe der Programmauflistung oder Daten auf dem Drucker)
- **S**io Serieller E/A-Port

  - **S**ave Senden eines Programms oder Daten über die serielle Schnittstelle
  - **L**oad Laden eines Programme oder Daten von der seriellen Schnittstelle
  - **F**ormat Einstellung der Parameter für die serielle Schnittstelle
- **F**ile Ein- und Ausgabe von Programmen auf der RAM-Disc
  - **S**ave Speichern eines Programms auf der RAM-Disc
  - **L**oad Laden eines Programms von der RAM-Disc
  - **K**ill Löschen eines Programms von der RAM-Disc
  - **F**iles Abrufen/Anzeigen aller Programme auf der RAM-Disc
- **B**asic Programm-Konvertierung zwischen BASIC- und TEXT- Formaten
  - **B**asic<-text Konvertierung von TEXT nach BASIC
  - **T**ext<-basic Konvertierung von BASIC nach TEXT
- **R**file Anlegen, Löschen, Laden, Speichern von Daten auf der RAM-Disc
  - **I**nit Anlegen von Daten-Dateien
  - **S**ave Speichern von Daten-Dateien
  - **L**oad Laden von Daten
  - **K**ill Löschen von Daten-Dateien
  - **F**iles Abrufen/Anzeigen aller Daten-Dateien auf der RAM-Disc

7.1 EDIT – Editieren von Programmen und Dateien

Zur Wahl der Editierfunktion vom Hauptmenü wird E gedrückt.



In der Editierfunktion ist die Eingabeaufforderung in der Befehlseingabe-Zeile wie folgt: "<" (anstatt ">" wie in BASIC).

Wie bei einem BASIC-Programm beginnt jede Zeile eines TEXT-Programms mit einer Zeilennummer. Bei einem TEXT-Programm fügt der Computer allerdings nicht automatisch nach der Zeilennummer einen Doppelpunkt (:) ein, wie das bei BASIC-Programmen der Fall ist. Auch wird nicht automatisch eine Leerstelle zwischen den eingegebenen Befehlen eingefügt. Jede Zeile erscheint genau so, wie sie eingetippt wird.

Hinweise:

- Die Nummern der Programmzeilen werden automatisch in aufsteigender Reihenfolge sortiert.
- Der Bereich der möglichen Zeilennummern für ein Programm ist von 1 bis 65279. Wenn dieser Bereich überschritten oder keine Zeilennummer eingegeben wird,

erfolgt die Anzeige einer Fehlermeldung (LINE NO. ERROR). Zum Löschen der Fehlermeldung CLS/CA drücken.

Zum Zurückgehen auf das Hauptmenü BREAK drücken.

Hinweis:
Eine TEXT-Zeile kann nicht mit einer Zahl beginnen, die der Zeilennummer direkt folgt. Wenn die Zeile unbedingt mit einer Zahl beginnen soll, muss zwischen der Zeilennummer und der Zahl ein Hochkomma (') eingefügt werden.

```
50 ' 100 FORMAT (17X, A)
```

Hochkomma

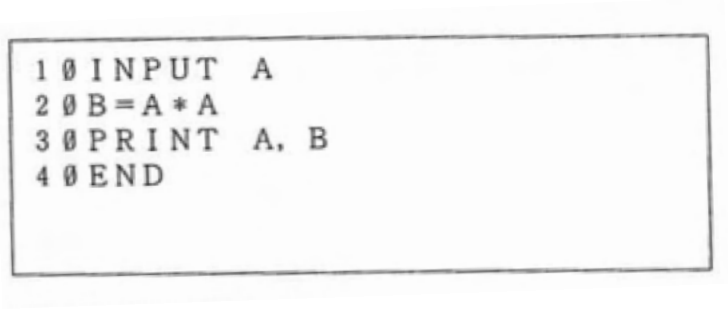
Zeilennummer

(Beispiel-Programm) Geben Sie das folgende Programm ein:

```
10INPUT A
20B=A*A
30PRINT A, B
40END
```

```
10INPUT SPACE A RETURN
20B=A*A  RETURN
30PRINT SPACE A,B RETURN
40END RETURN
```

10INPUT SPACE A RETURN
20B=A*A RETURN
30PRINT SPACE A,B RETURN
40END RETURN



Editieren von Programmen

Ein TEXT-Programm wird genau so wie ein BASIC-Programm editiert. (Siehe die Erklärungen zum Programmieren in BASIC)

Die Edit-Befehle der TEXT-Betriebsart entsprechen den Befehlen im BASIC. (Für Details

über die Befehle siehe die Erklärungen im BASIC KOMMANDO LEXIKON.

Befehle:

AUTO	A	Auto-Nummerierung (siehe auch Basic-Befehl AUTO auf Seite 185)
LIST	L	Auflisten der Zeilen (siehe auch Basic-Befehl LIST auf Seite 218)
RENUM	R	Neunummerierung (siehe auch Basic-Befehl RENUM auf Seite 238)
DELETE	D	Löschen von Zeilen (siehe auch Basic-Befehl DELETE auf Seite 197)
LCOPY	C	Kopieren von Zeilen (siehe auch Basic-Befehl LCOPY auf Seite 214)
	S	Suchen einer Zeichenkette im Editor
	E	Suchen und ersetzen einer Zeichenkette

Wenn der R-Befehl in einem TEXT-Programm ausgeführt wird, das von einem BASIC-Programm konvertiert wurde, werden lediglich die Zeilennummern am Anfang einer Zeile neu nummeriert, wahrend die Zeilennummern innerhalb der Anweisungen GOTO, THEN, GOSUB bzw. RESTORE nicht neu nummeriert werden. In diesem Fall läuft das Programm nicht, wenn es wieder in BASIC zurückkonvertiert wird.

Befehlsformat für A:
A [[<Start-Zeilennummer>][,<Ergänzungswert>]]

Nach Aktivierung von A erscheint die erste generierte Zeilennummer in der Anzeige mit einem nachgestellten Cursor, Nun kann der gewünschte Zeileninhalt eingegeben werden. Schließt man die Eingabe dann durch Betätigung der ENTER Taste ab, so wird in der folgenden Zeile die nächste Zeilennummer generiert und so fort.

Befehlsformat für L:
(1) L
(2) L <zeilennummer>
(3)L <Label>

Listet das Programm von vorne oder ab der angegeben Zeilennummer oder Labels auf.

Befehlsformat für R:

R [<alteZeile>[,<neueZeile>][,<zeilenabstand>]]

R führt eine Neunummerierung aller Zeilen oder der angegebenen Zeilen durch.

Befehlsformat für D:

D <zeilennummer>[, [<endzeilennummer>]]

Löscht die angegebene Zeile oder alle Zeilen von der angegeben Startzeile bis einschließlich der zweiten angegebenen Zeilennummer. Die restlichen Syntax-Varianten entsprechen dem Basic-Befehl DELETE.

Befehlsformat für C:

C <startzeile>,<endzeile>,<zielzeile>

Kopiert die Zeilen von <startzeile> bis <endzeile> nach <zielzeile>.
Achtung: Dabei werden Sprungadressen in Basic-Befehlen nicht angepasst

Befehlsformat für S:

S [0|1,] “<zeichenkette>“

Sucht eine Zeichenkette in den Daten des Editors. Wird die Zeichenkette gefunden, wird der Cursor auf die erste Stelle der gefundenen Zeichenkette gesetzt. Durch ENTER wird der Cursor auf die nächste gefundene Zeichenkette gesetzt. CLS beendet die Suche. Die zu suchende Zeichenkette darf eine maximale Länge von 16 Zeichen besitzen.

Durch die Angabe von 0 oder 1 wird die Suchrichtung bestimmt:
1: Die Suche erfolgt vorwärts ab Anfang der Datei
0: Die Sucher erfolgt Rückwärts vom Ende der Datei
Wird dieser Parameter nicht angegeben, wird von Beginn der Datei an gesucht (1).

Bei der Suche nach einem “ (doppelte Anführungszeichen) muss ¥“ als String" angegeben werden.
Beispiel: S “ ¥“ “

Befehlsformat für E:

S [0|1,] “<suchzeichenkette>“,“<austauschzeichenkette>“

Sucht und ersetzt eine Zeichenkette in den Daten des Editors. Wird die Zeichenkette gefunden, wird der Cursor auf die erste Stelle der gefundenen Zeichenkette gesetzt.

- Durch ENTER wird die gefundene Zeichenkette mit der zweiten angegebenen Zeichenkette ausgetauscht und der Cursor auf die nächste gefundene Zeichenkette gesetzt.
- Mit der Leertaste SPACE wird diese Zeichenkette nicht ausgetauscht und der Cursor auf die nächste gefundene Zeichenkette gesetzt.

CLS beendet die Suche. Die Zeichenketten dürfen eine maximale Länge von 16 Zeichen besitzen.

Durch die Angabe von 0 oder 1 wird die Suchrichtung bestimmt:
1: Die Suche erfolgt vorwärts ab Anfang der Datei
0: Die Sucher erfolgt Rückwärts vom Ende der Datei
Wird dieser Parameter nicht angegeben, wird von Beginn der Datei an gesucht (1).

Bei der Suche nach einem “ (doppelte Anführungszeichen) muss ¥“ als String"

angegeben werden.
Beispiel: S “ ¥“ “

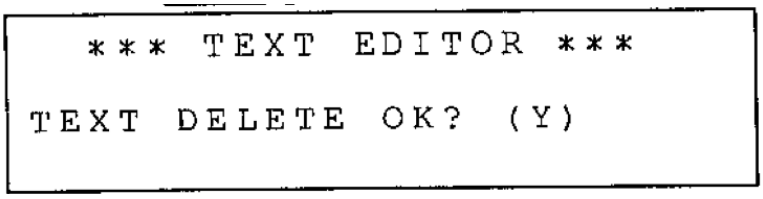
Die Taste TAB

In der Editierfunktion kann der Cursor mit der Taste TAB an die nächste Tabellenposition bewegt werden.

Beim ersten Drücken von TAB bewegt sich der Cursor auf die Spalte 8. Beim nächsten Drücken bewegt er sich auf die Spalte 14 (6 Stellen nach der ersten Tabellenposition). Bei jedem folgenden Drücken auf TAB bewegt sich der Cursor um sieben Stellen vorwärts (Bis 21)

7.2 Löschen eines TEXT-Programms (Del)

Zur Wahl der Löschfunktion vom TEXT-Hauptmenü wird [D] gedrückt.



Beim Drücken von [Y] wird der gesamte TEXT-Speicherbereich vollständig gelöscht, einschließlich dem TEXT-Programm, und das Hauptmenü wird wieder angezeigt.

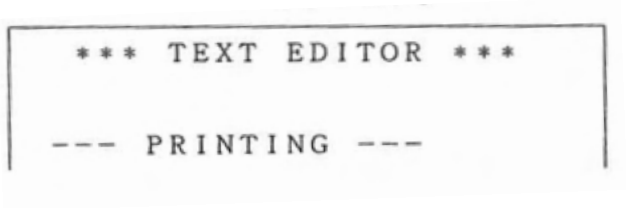
Wenn eine andere Taste als [Y] gedrückt wird, geht der Computer auf das Hauptmenü zurück, ohne etwas zu löschen.

Hinweis:

Wenn im TEXT-Bereich kein Text gespeichert ist, reagiert der Computer bei angezeigtem Hauptmenü nicht auf das Drücken von [D].

7.3 Ausdrucken einer TEXT-Programmauflistung (Print)

Den Drucker CE-126P mit dem Computer verbinden und den Computer sowie den Drucker einschalten, Das TEXT-Hauptmenü zur Anzeige bringen und [P] drücken, um das gespeicherte TEXT-Programm auszudrucken.



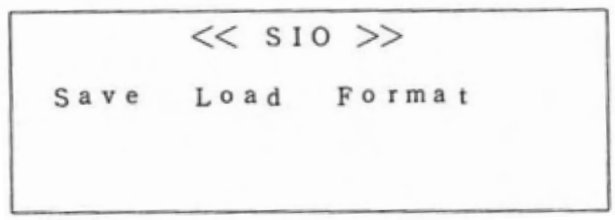
Nach dem Ausdrucken zeigt der Computer wieder das Hauptmenü an.

Hinweis:

Zum Abbrechen des Ausdrucks BREAK drücken. Wenn der Drucker nicht eingeschaltet oder nicht mit dem Computer verbunden ist, reagiert der Computer bei angezeigtem Hauptmenü nicht auf das Drücken von [P].

7.4 Serielle Eingabe/Ausgabe (Sio)

Beim Drücken von [S] bei angezeigtem TEXT-Hauptmenü gibt der Computer das Menü für die serielle Eingabe/Ausgabe (SIO-Menü) aus.



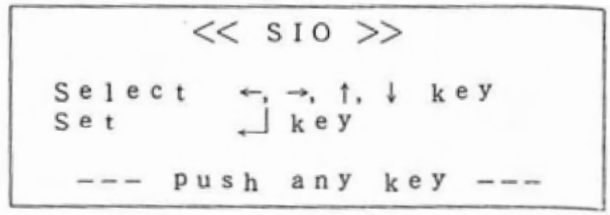
Vom SIO-Menü wird die entsprechende Funktion gewählt — Senden(Save), Empfangen(Load) oder Formatieren(Format) —, indem der erste Buchstabe der Funktion eingegeben wird (S, L oder F).

Einstellung der E/A-Parameter (Format)

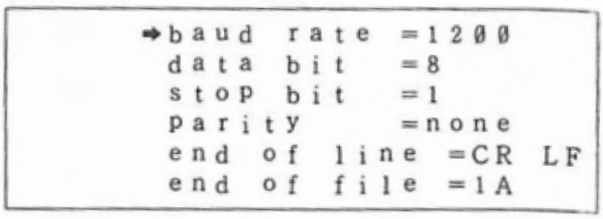
Mit dieser Funktion können die Parameter der seriellen Kommunikation eingestellt werden. Die Kommunikations-Parameter müssen mit demjenigen Gerät übereinstimmen, mit dem dieser Computer kommunizieren soll.

Zur Anzeige eines Hilfsmenüs vom SIO-Menü wird [F] gedrückt. Danach eine

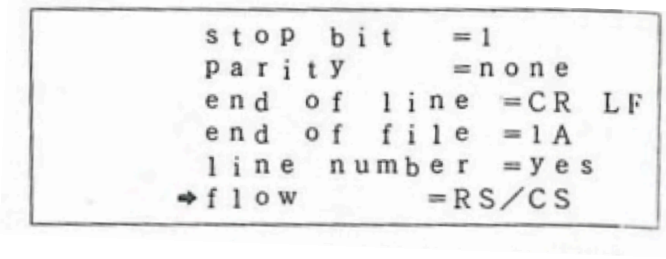
beliebige Taste drücken oder etwas warten, bis die Einstellung der Kommunikations-Parameter auf dem Display angezeigt werden.



Eine beliebige Taste drücken oder etwas warten, bis die folgende Anzeige erscheint.



➡zeigt den gewählten Parameter an. Mit den Tasten [↓] bzw. [↑] kann ➡ auf einen zu ändern den Parameter bewegt werden. Es können insgesamt sieben Parameter eingestellt werden. Mit der Taste [↓] können die Parameter auf dem Display durchlaufen werden.



Mit [►] und [◄] werden die Einstellungen geändert. Die Einstellung für den Parameter "end of file" (Ende der Datei) muss allerdings manuell eingegeben werden, Nach der Eingabe der Änderungen wird ENTER gedrückt, um die Änderungen zu speichern. Wenn die neuen Einstellungen nicht gespeichert werden, verwendet der Computer die vorher eingestellten Parameter.

Erläuterung der Kommunikations-Parameter

• Baudrate : 300, 600, 1200, 2400, 4800, 9600
Die Baudrate entspricht der Geschwindigkeit der Datenübertragung. Je größer die Baudrate, desto schneller die Geschwindigkeit der Datenübertragung. Die Baudrate kann zwischen 300, 600, 3 200, 2400, 4800 und 9600 bps (Baud pro Sekunde) gewählt wer- den.

• Data bit : 7oder8
Die Datenlänge (Datenbit) ist die Anzahl der Bit, die benötigt werden, um ein Zeichen darzustellen. Es können entweder 7 oder 8 Bit eingestellt werden.

• Stop bit : 1oder2
Das Stopbit bestimmt die Länge des Stoppbits am Ende jedes Zeichens.

• Parity : none, even oder odd
Die Parität bestimmt die Art der Datenüberprüfung (Paritätsprüfung).
keine (none) ... Zu den übertragenen Daten wird kein Paritätsbit hinzugefügt (keine Paritätsprüfung).
gerade (even) ... Bestimmt eine gerade Parität.
ungerade (odd) ... Bestimmt eine ungerade Parität.

• End of line : CR,LF oder CR + LF
Mit "Ende der Zeile" wird der Begrenzungscode für das Ende jeder Programmzeile bestimmt.
CR ... Bestimmt den Code für einen Wagenrücklauf.
LF ... Bestimmt den Code für einen Zeilenvorschub.
CR + LF ... Bestimmt den Code für CR und LF.

• Ende of file : 0 0bis FF (zweistellige Hexadezimalzahl)
Mit "Ende der Datei" wird ein Code für das Ende des Textes bestimmt, der das Ende eines Programms oder einer anderen Datei anzeigt.

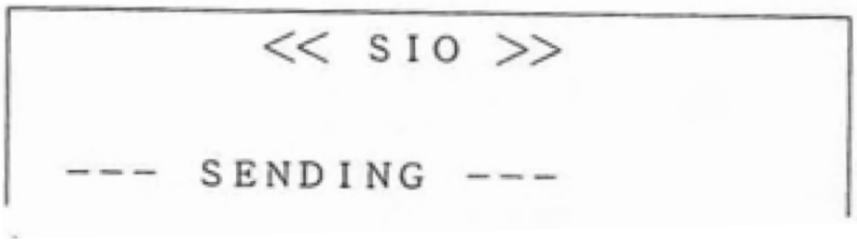
• line number: yes oder no
Mit der "Zeilennummer" wird bestimmt, ob ein TEXT-Programm mit oder ohne Zeilennummern gesendet wird.
ja (yes) ... Das Programm wird mit Zeilennummern gesendet.
nein (no) ... Das Programm wird ohne Zeilennummern gesendet.
Mit "Zeilennummer" wird weiterhin bestimmt, ob beim Empfang automatisch eine Zeilennummer (in Schritten von 10) zu jeder Programmzeile hinzugefügt werden soll.
ja (yes) ... Es werden keine Zeilennummern hinzugefügt. "ja" wird gewählt, wenn das Programm bereits Zeilennummern enthält.
nein (no) ... Es werden automatisch Zeilennummern hinzugefügt. Wenn die empfangene Datei keine Zeilennummern enthält, obwohl "ja" eingestellt war, wird eine Fehlermeldung (LINE NO, ERROR) angezeigt.

• flow : RS/CS, Xon/Xoff, oder none
Gibt an, wie die Flußkontrolle bei der Kommunikation erfolgen soll
RS / CS ... Die Flußkontrolle wird durch die RS/CS-Signale gesteuert.
Xon/Xoff ... Die Flußkontrolle wird durch das Xon/Xoff-Protokoll gesteuert.
none ... Die Übertragung wird ohne Flußkontrolle durchgeführt.

Die eingestellten Werte gelten auch für alle folgenden fopen(“stdaux1“, oder OPEN "COM1".
Nachdem die Werte geändert und gespeichert wurden, gelten diese neuen Parameter, bis die RESET-Taste gedrückt wird, um den Speicher zu löschen oder bis die Batterie ausgewechselt wird oder bis die Einstellungen erneut geändert werden.

7.5 Senden von Programmen (Save)

Beim Drücken von [S] bei angezeigtem SIO-Menü beginnt der Computer ein TEXT-Editor abgelegtes Programm oder Daten über den seriellen I/O-Port zu senden.



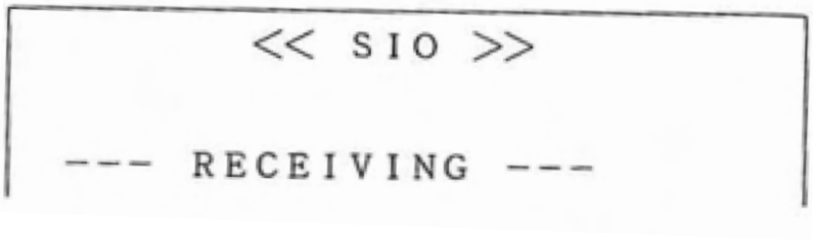
Nach dem Senden geht der Computer auf das SIO-Menü zurück.

Hinweise:

- Zum Abbrechen des Sendens BREAK drücken. Der Computer geht auf das SIO-Menü zurück.
- Wenn im TEXT-Bereich kein Programm oder Daten gespeichert sind, reagiert der Computer nicht auf das Drücken von [S].

7.6 Empfangen von Programmen (Load)

Beim Drücken von [L] bei angezeigtem SIO-Menü beginnt der Computer, die Daten über den seriellen I/O-Port zu empfangen.



Nach dem Empfangen geht der Computer auf das SIO-Menü zurück.

Hinweise:

- Zum Abbrechen des Empfangs BREAK drücken. Der Computer geht auf das SIO-Menü zurück.
- Wenn das Programm nicht richtig empfangen wurde oder wenn ein Paritätsfehler auftrat, erscheint eine Fehlermeldung (I/O DEVICE ERROR). Zum Löschen der Fehlermeldung CLS-CA drücken.

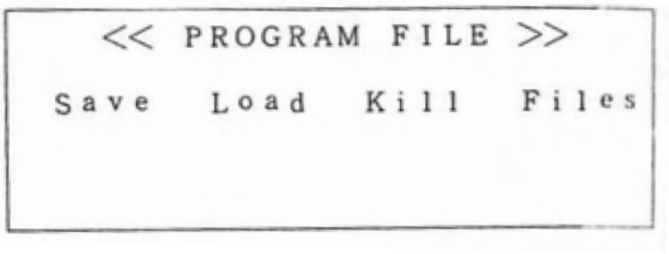
7.7 parallele Schnittstelle

Wenn im TEXT-Hauptmenü [L] gedrückt wird erfolgt der Ausdruck über die parallele Schnittstelle (Centroncis-Protokoll).

Weitere Informationen zur Nutzung der parallelen Schnittstelle siehe auch INP- und OUT-Befehl.

7.8 Ein- und Ausgabe von Programmen mit Hilfe der Ram-Disk (File)

Beim Drücken von [F] bei angezeigtem Hauptmenü gibt der Computer das Ram-Disk-Datei-Menü aus.

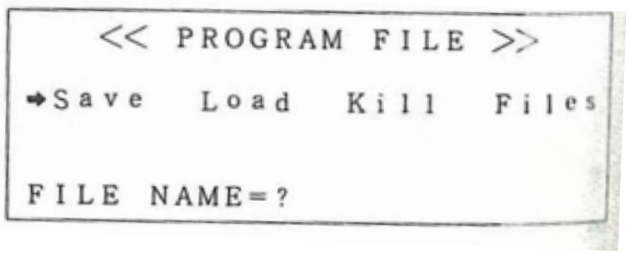


Von diesem Menü wird die entsprechende Funktion gewählt — Speichern (Save), Laden (Load), Löschen (Kill) oder Dateien ansehen (Files) —, indem der erste Buchstabe der Funktion eingegeben wird (S, L, K oder F).

Speichern eines TEXT-Programms (Save)

Beim Speichern muss dem TEXT-Programm einen Namen zugewiesen werden.

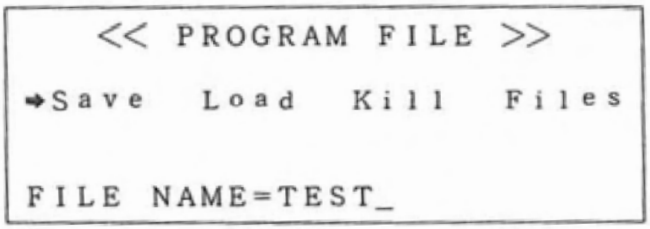
Beim Drücken von [S] bei angezeigtem Programm-Datei-Menü fordert der Computer zur Eingabe des Namens der zu speichernden Datei auf.



Den Dateinamen eingeben und ENTER drücken. Der Computer speichert nun diese Datei.

Beispiel:
Speichern einer Datei mit dem Dateinamen "TEST".

TEST



Der Computer speichert die Datei "TEST" und geht dann wieder auf das Programm-Datei-Menü zurück.

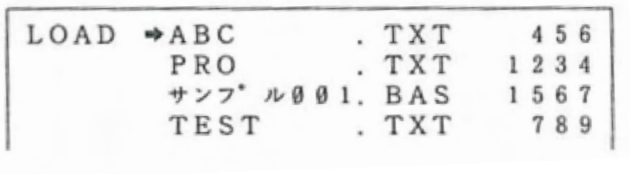
Hinweise:

- Nach der Wahl der Speicher-Funktion aus dem Datei-Menü muss unbedingt ein Dateiname eingegeben werden. Wenn die Taste ENTER ohne Eingabe eines Namens gedrückt wird, erfolgt eine Fehlermeldung (ILLEGAL FILE NAME). Zum Löschen der Fehlermeldung CLS drücken.
- Ein Dateiname kann aus bis zu acht Zeichen bestehen und eine Erweiterung von
- bis zu drei Zeichen enthalten. Wenn keine Erweiterung eingegeben wird, weist der Computer automatisch die Erweiterung ".TXT" zu.
- Wenn kein TEXT-Programm im TEXT-Bereich gespeichert ist, kann auch kein Speichern ausgeführt werden.

Laden einer TEXT-Datei (load)

Beim Drücken von [L] bei angezeigtem Programm-Datei-Menü gibt der Computer eine Liste der gespeicherten Dateien aus; dabei weist "LOAD ➡" auf den ersten Dateinamen (wenn kein Programm gespeichert wurde, reagiert der Computer nicht auf das Drücken von [L]).

(Hier wird ein Beispiel für eine Liste mit registrierten Dateien angeführt.)



Mit den Tasten [↓] und [↑] wird "LOAD ➡" auf den Namen der zu ladenden Datei bewegt; danach ENTER drücken. Der Computer lädt den Inhalt der gewählten Datei in den TEXT-Bereich und geht dann auf das Programm-Datei-Menü zurück.

Hinweis:
In der TEXT-Betriebsart können nur Programme und Dateien abgerufen werden, die in der TEXT-Betriebsart registriert wurden. Beim Versuch des Abrufens eines BASIC-

Programms, dass mit einem SAVE-Befehl von BASIC gesichert wurde, wird eine Fehlermeldung (FILE MODE ERROR) angezeigt. Zum Löschen der Fehlermeldung CLS drücken.

Löschen einer Programm-Datei (Kill)

Mit dieser Funktion wird eine bestimmte Datei gelöscht.

Beim Drücken von [K] bei angezeigtem Programm-Datei-Menü fragt der Computer nach dem Namen der zu löschenden Datei.

```
<< PROGRAM FILE >>
 Save  Load ➡Kill  Files

FILE NAME=?
```

Den Namen der zu löschenden Datei eingeben und ENTER drücken.
Daraufhin fragt der Computer ob die Datei wirklich gelöscht werden soll

FILE DELETE OK? (Y)

Durch die Eingabe von [Y] bestätigen Sie den Löschvorgang. Jede andere Taste bricht den Löschvorgang ab und es wird wieder das Datei-Menü angezeigt.
Wenn beim Dateinamen keine Endung angegeben wird, wird standardgemäß die Endung .TXT angefügt.
Wenn die angegebene Datei nicht vorhanden ist gibt der Computer die Fehlermeldung (FILE NOT FOUND) aus.

Auflisten von Dateinamen (Files)

Beim Drücken von [F] bei angezeigtem Programm-Datei-Menü gibt der Computer eine Liste aller gespeicherten Dateien aus. Dabei zeigt ➡ auf den ersten Dateinamen der Liste. (Wenn keine Dateien gespeichert sind, reagiert der Computer nicht auf das Drücken von [F].)

(Hier wird ein Beispiel für eine Liste mit registrierten Dateien angeführt.)

```
➡ABC      .TXT   456
 PRO      .TXT  1234
 ｻﾝﾌﾟﾙ００1.BAS  1567
 TEST     .TXT   789
```

Die noch nicht angezeigten Teile der Liste können durch Drücken von [↓] oder [↑] angesehen werden.

Um ein Programm zu laden, das durch ➡ markiert ist, wird [SHIFT] + [LOAD M] gedrückt (oder [2nd F] [LOAD M]).

Über die Dateigröße:

Die Größe der Textdatei ergibt sich aus der Gesamtzahl der Bytes für jede Zeile. Die Anzahl der Bytes in jeder Zeile berechnet sich aus Zeilennummer (3Byte), line-feed(1Byte) und der Anzahl der Zeichen im Text der Zeile.

Beispiel:

10␣INPUT␣A [↵]

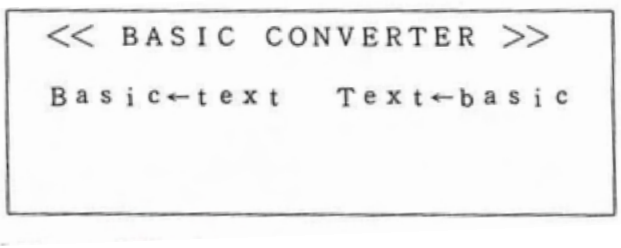
ergibt 3 + 8 + 1 = 12 Bytes für diese Zeile.

Bei der Umwandlung in BASIC-Code verkürzt sich die Programmlänge, da die BASIC-Schlüsselwörter weniger Bytes benötigen.

7.9 Der BASIC-Konverter (Basic)

Mit dieser Funktion wird ein BASIC-Programm im Zwischencode-Format in eine TEXT-Datei im ASCII-Format umgewandelt oder umgekehrt. Diese Funktion ist nützlich, um BASIC-Programme mit einem Personal Computer zu bearbeiten, die auf diesem Computer geschrieben wurden.

Beim Drücken von [B] bei angezeigtem Hauptmenü gibt der Computer das BASIC-Konverter-Menü aus.



Von diesem Menü kann die Konvertierung von TEXT nach BASIC oder von BASIC nach TEXT gewählt werden. Den ersten Buchstaben des Formates eingeben, in das umgewandelt werden soll.

Umwandlung von TEXT- und BASIC-Programmen (zwischen ASCII-Format und Zwischencode-Format)

Beim Drücken von [B] bei angezeigtem BASIC-Konverter-Menü wandelt der

Computer das TEXT-Programm im TEXT-Bereich in ein BASIC-Programm um und speichert es im Programmdaten-Bereich.
Beim Drücken von [T] bei angezeigtem BASIC-Konverter-Menü wandelt der Computer das BASIC-Programm im Programmdaten-Bereich in ein TEXT-Programm um und speichert es im TEXT-Bereich.

Beispiel:
Umwandeln eines TEXT-Programms in BASIC.

[B]

```
<< BASIC CONVERTER >>

--- CONVERTING ---
```

Nach der Konvertierung geht der Computer auf das Hauptmenü zurück. (Bei Umwandlung eines kurzen Programms beansprucht die Konvertierung nur sehr wenig Zeit.)

Wenn sich im Programm-Bereich bereits ein BASIC-Programm befindet, während ein TEXT-Programm in ein BASIC-Programm umgewandelt wird oder wenn andererseits im TEXT-Bereich bereits ein Programm vorhanden ist, während ein BASIC-Programm umgewandelt wird, fragt der Computer zur Sicherheit, ob das vorhandene Programm vor der Konvertierung gelöscht werden soll.

```
➡Basic←text   Text←basic

BASIC DELETE OK? (Y)
```

Beim Drücken von [Y] löscht der Computer das vorhandene BASIC-Programm und beginnt mit der Konvertierung.

Beim Drücken einer anderen Taste als [Y] wird die Konvertierung abgebrochen und der Computer geht auf das Hauptmenü zurück.

Im allgemeinen behält der Computer das Originalprogramm bei, nachdem es in ein anderes Format umgewandelt wurde. Wenn allerdings nach der Umwandlung eines Programms nicht genügend Speicherplatz zur Verfügung steht, fragt der Computer zur Sicherheit, ob das Originalprogramm gelöscht werden soll.

```
--- CONVERTING ---

TEXT DELETE OK? (Y)
```

Beim Drücken von [Y] löscht der Computer das Originalprogramm im Verlauf der

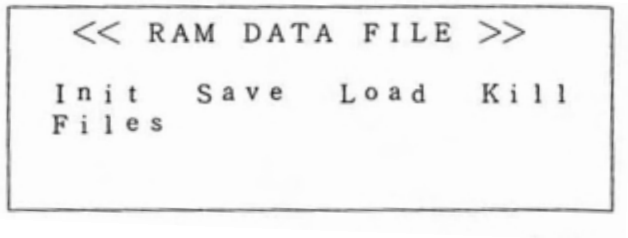
Konvertierung. Am Ende der Konvertierung ist das Originalprogramm vollständig gelöscht.

Beim Drücken einer anderen Taste als [Y] wird die Konvertierung abgebrochen und der Computer geht auf das Hauptmenü zurück.

7.10RAM-DISK - Datendateien (RFILE)

Mit dieser Funktion werden Daten-Dateien angelegt, gelöscht, in den TEXT-Editor geladen oder vom Editor gespeichert.

Beim Drücken von [R] bei angezeigtem Hauptmenü gibt der Computer das Daten-Datei-Menü (RFILE) aus.

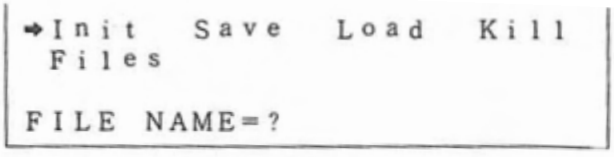


In diesem Menü wird die entsprechende Funktion gewählt — Datei Anlegen(Init) Speichern (Save), Laden (Load), Löschen (Kill) oder Dateien ansehen (Files) —, indem der erste Buchstabe der Funktion eingegeben wird (I,S, L, K oder F).

Anlegen einer Datei (Init)

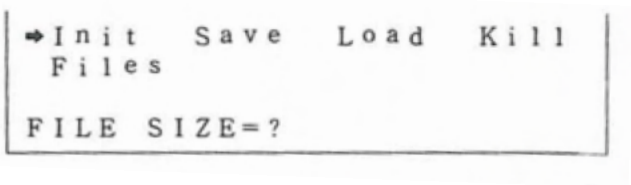
Beim Anlegen einer Daten-Datei muss ihr ein Namen zugewiesen werden.

Beim Drücken von [I] bei angezeigtem Daten-Datei-Menü fordert der Computer zur Eingabe des Namens der anzulegenden Datei auf.



Beispiel für das Anlegen der Datei TEST.

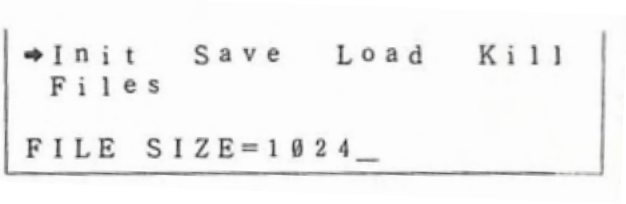
TEST.DAT



Danach muss die Größe der Datei in Bytes angegeben werden. Die Größe muss so gewählt werde, dass alle benötigten Daten in die Datei hineinpassen. Als Beispiel

werden hier 1024 Bytes (1KByte) angegeben.

1024



Wenn die angegebene Datei bereits angelegt wurde fragt der Computer durch die Ausgabe von FILE INITIALZE OK? (Y) ob die Datei neu initialisiert werden soll und damit alle Daten gelöscht werden sollen.

Wenn Sie die Taste [Y] drücken wird die Datei neu initialisert. Jede andere Taste bricht das Anlegen ab

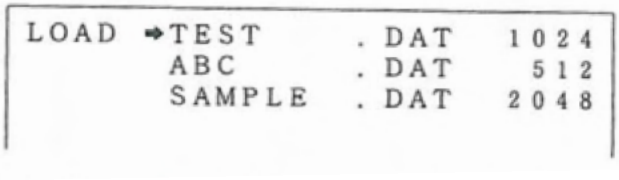
Hinweise:

- Geben Sie bei Dateinamen keine Dateiendung an wird automatisch die Endung .DAT angefügt.
- Der Dateinamen darf ein maximale Länge von 8 Zeichen haben.
- Eine Datei belegt die angegebene Menge Bytes im Speicher und 34 zusätzliche Bytes
- Wenn nicht mehr genug Platz im Speicher zum Anlegen der Datei vorhanden ist, gibt der Computer die Meldung "MEMORY OVER" aus,

Laden einer Daten-Datei (load)

Beim Drücken von [L] bei angezeigtem Daten-Datei-Menü gibt der Computer eine Liste der gespeicherten Dateien aus; dabei weist "LOAD ➡" auf den ersten Dateinamen (wenn keine Dateien gespeichert wurde, reagiert der Computer nicht auf das Drücken von [L]).

(Hier wird ein Beispiel für eine Liste mit gespeicherten Dateien angeführt.)



Mit den Tasten [↓] und [↑] wird "LOAD ➡" auf den Namen der zu ladenden Datei bewegt; danach ENTER drücken. Der Computer lädt den Inhalt der gewählten Datei in den TEXT-Bereich und geht dann auf das Programm-Datei-Menü zurück.

Löschen einer Daten-Datei (Kill)

Mit dieser Funktion wird eine bestimmte Datei gelöscht.

Beim Drücken von [K] bei angezeigtem Daten-Datei-Menü fragt der Computer nach dem Namen der zu löschenden Datei.

```
<< PROGRAM FILE >>
 Save  Load ➡Kill  Files

FILE NAME=?
```

Den Namen der zu löschenden Datei eingeben und ENTER drücken.
Daraufhin fragt der Computer ob die Datei wirklich gelöscht werden soll

FILE DELETE OK? (Y)

Durch die Eingabe von [Y] bestätigen Sie den Löschvorgang. Jede andere Taste bricht den Löschvorgang ab und es wird wieder das Datei-Menü angezeigt.
Wenn beim Dateinamen keine Endung angegeben wird, wird standardgemäß die Endung .DAT angefügt.
Wenn die angegebene Datei nicht vorhanden ist gibt der Computer die Fehlermeldung (FILE NOT FOUND) aus.

Auflisten von Dateinamen (Files)

Beim Drücken von [F] bei angezeigtem Programm-Datei-Menü gibt der Computer eine Liste aller gespeicherten Dateien aus. Dabei zeigt ➡ auf den ersten Dateinamen der Liste. (Wenn keine Dateien gespeichert sind, reagiert der Computer nicht auf das Drücken von [F].)

```
LOAD ➡TEST     .DAT  1024
       ABC     .DAT   512
       SAMPLE  .DAT  2048
```

(Hier wird ein Beispiel für eine Liste mit gespeicherten Dateien angeführt.)

Die noch nicht angezeigten Teile der Liste können durch Drücken von [↓] oder [↑] angesehen werden.

Um eine Datei zu laden, die durch ➡ markiert ist, wird [SHIFT] + [LOAD M] gedrückt (oder [2nd F] [LOAD M]).

Speichermangel bei Arbeiten um den TEXT-EDITOR/Basic-Konvertierung:

Wenn der Computer während einer BASIC-Konvertierung erkennt, dass der Speicher nicht ausreicht um beide Versionen vorzuhalten, wird während der Konvertierung Stück für Stück die Quellversion gelöscht. Wächst das Zielprogramm zu stark an kann es zum Abbruch der Funktion führen. Als Folge liegt eine Teil des Programms im Quellformat und der Rest im Zielformat vor und ist damit nicht mehr brauchbar.
Wenn eine solche Situation zu erwarten ist (z.B. bei knappen Speicher), sollten sie daher vorher das Quellprogramm über das serielle Interface sichern oder zur Not ausdrucken.

Speichern von Daten-Dateien (Save)

Beim Speichern muss der Datei ein Name zugewiesen werden.

Beim Drücken von [S] bei angezeigtem Daten-Datei-Menü fordert der Computer zur Eingabe des Namens der zu speichernden Datei auf.

```
<< RAM DATA FILE >>

Init ➡Save  Load  Kill
Files

FILE NAME=?
```

Den Dateinamen der vorher mit der Init-Funktion angelegten Datei eingeben und ENTER drücken. Der Computer speichert nun diese Datei.

Beispiel:
Speichern einer Datei
mit dem Dateinamen "TEST".

TEST

```
<< PROGRAM FILE >>

➡Save  Load  Kill  Files

FILE NAME=TEST_
```

Der Computer speichert die Datei "TEST.DAT" und geht dann wieder auf das Daten-Datei-Menü zurück. Dazu muss die Frage des Computers „FILE OVERWRITE OK? (Y)“ mit [Y] beantwortet werden. Jede andere Eingabe bricht die Funktion SAVE ab.

Wurde die Datei vorher nicht mit INIT angelegt gibt der Computer die Fehlermeldung „FILE NOT FOUND“ aus und bricht die Funktion ab.

Hinweise:

- Wenn die zu speichernde Datei größer ist als die bei der Init-Funktion angegeben Größe bricht der Computer die Aktion mit der Meldung „MEMORY OVER“ ab.
- Nach der Wahl der Speicher-Funktion aus dem Datei-Menü muss unbedingt ein Dateiname eingegeben werden. Wenn die Taste ENTER ohne Eingabe eines Namens gedrückt wird, erfolgt eine Fehlermeldung (ILLEGAL FILE NAME). Zum Löschen der Fehlermeldung CLS drücken.
- Ein Dateiname kann aus bis zu acht Zeichen bestehen und eine Erweiterung von
- bis zu drei Zeichen enthalten. Wenn keine Erweiterung eingegeben wird, weist der Computer automatisch die Erweiterung ".DAT" zu.

- Wenn keine Daten im TEXT-Bereich gespeichert sind, kann auch kein Speichern ausgeführt werden.

8 DIE PROGRAMMIERSPRACHE C

Dieses Kapitel beschreibt den Unterschied zwischen der C-Sprache auf großen Computern (z.B. unter UNIX) und wie auf dem SHARP PC-G850 C-Programme erstellt werden können.

Für die Programmiersprache C selber sind im Handel im großen Umfang Bücher erhältlich.

8.1 Eigenschaften der Programmiersprache C

C ist eine sehr kompakte Sprache. Einerseits handelt es sich bei C um eine höhere Programmiersprache, andererseits ist die Verwendung detaillierter Verarbeitungsnotationen möglich, die sich sehr nahe an der Maschinensprache anlehnen.

Während bei höheren Programmiersprachen wie BASIC, FORTRAN usw. Zugriffe auf die Hardware nur über PEEK und POKE möglich sind, kann man in der Sprache C Programme erzeugen die direkt auf die Hardware und dem Speicher zugreifen, ähnlich wie ein Assembler-Programm. Die Programmierung in C gegenüber der Assemblersprache aber deutlich effizienter.

Mit seiner strukturierten Programmierung ist C leicht zu lesen und leicht zu verstehen. Damit ist es möglich eine sehr effektive leistungsfähige Programmentwicklung durchzuführen. Darüber hinaus sind eine Vielzahl von Datentypen für die Verarbeitung von Daten und numerischen Funktionen vorhanden. Daher ergibt sich für die Sprache C eine weite Palette an Anwendungsfällen, sei es beruflich, privat oder im wissenschaftlichen Bereich.

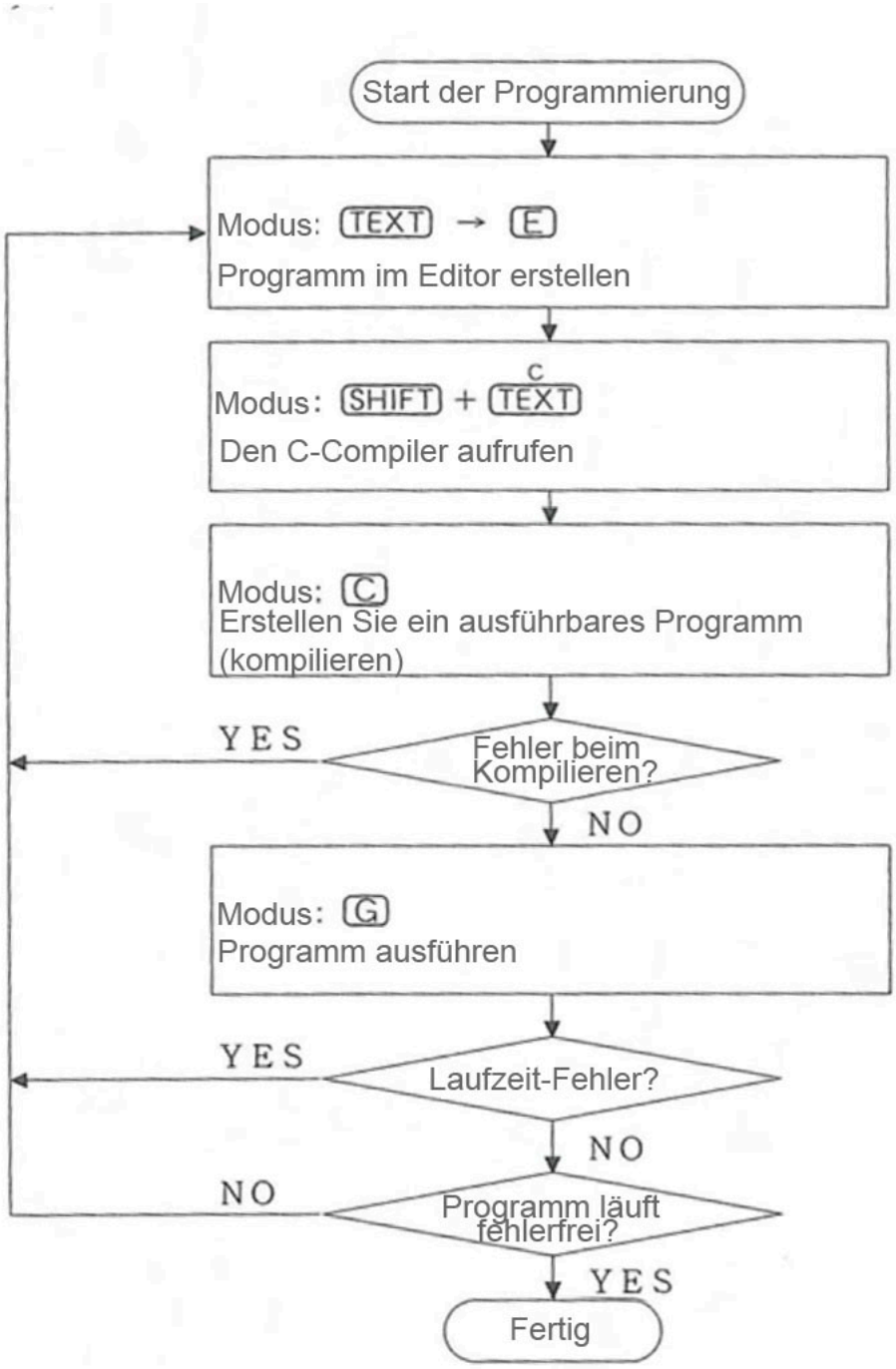
C-Programme sind sehr kompakt. Programme sind darüber hinaus durch die Möglichkeit der Pointer-Nutzung sehr effizient.

Das Portabilität der C-Programme ist trotz der hardwarenahen Programmierung sehr hoch. C-Programme können meist mit wenig Änderungen auf anderen Rechnern lauffähig gemacht werden.

Die C-Sprache ist sehr mächtig. Daraus ergibt sich auch der Nachteil, dass Programme sehr unübersichtlich geschrieben werden können, da es oft mehrere Varianten gibt eine Aufgabe zu lösen.

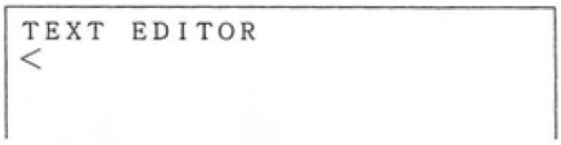
8.2 Grundlegende Arbeitsweise mit dem C-Compiler

Da der Computer die C-Anweisungen nicht direkt versteht muss das C-Programm vor der Ausführung kompiliert werden. Dazu sind in der Regel folgende grundlegenden Schritte notwendig:



Aufruf des Text-Editors:

[TEXT] → [E]



Eingabe des C-Source-Programms:

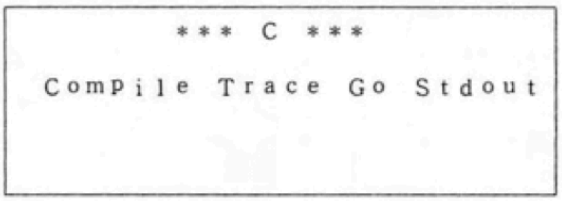
```
10 main()
20 {
30   printf(“Hello World¥n“);
40 }
```

Schalten Sie zur Eingabe der Befehle in den CAPS-Modus. Ähnlich wie in BASIC muss jeder Zeile eine Zeilennummer vorangestellt werden (ohne folgenden Doppelpunkt). Der C-Compiler selber kann dagegen nichts mit den Zeilennunmern anfangen. Daher werden Sie intern beim kompilieren ignoriert. Sie dienen daher nur dem Editor.
Die Editor-Funktionen entnehmen Sie bitte dem entsprechenden Kapitel zum TEXT-Modus.

Kompilieren des Source-Programms

Rufen Sie das C-Compiler-Menu auf:

[SHIFT] + [TEXT]



Folgende Befehle stehen zur Verfügung:

Compile : Kompilieren des im TEXT-Editor befindlichen Programms
Trace : Ausführen des Programms im Trace-Modus (Schritt für Schritt)
Go : Ausführen des Programms
Stdout : Schaltet die Standardausgabe auf den Drucker um
Der jeweilige Befehl wird durch Eingabe des ersten Buchstabens ausgewählt.

Kompilieren:

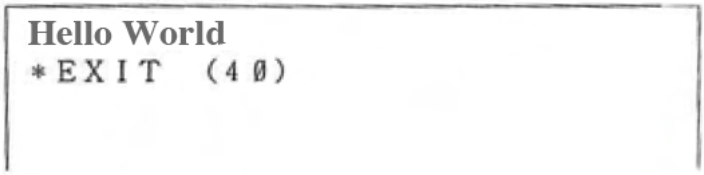
Drücken Sie die Taste C. Es erscheint kurz die Ausgabe „compiling“.
Konnte das Programm ordnungsgemäß kompiliert werden erscheint kurze Zeit später die Ausgabe „complete !“

Erscheint stattdessen eine Fehlermeldung muss das Programm mit dem Editor (TEXT-E) korrigiert und neu kompiliert werden.

Erscheint die Meldung „memory full“ reicht der freie Speicher nicht mehr aus um das ausführbare Programm zu erstellen.

Ausführen des Programms

Führen Sie das Programm aus in dem Sie im C-Menü die Taste [G] drücken.



- EXIT zeigt die Nummer der Zeile an, wo die Ausführung des Programms abgeschlossen wurde. Zur Rückkehr in das C.Menü die Taste CLS oder BREAK drücken.

Weiter unten sind die Laufzeitfehler beschrieben.

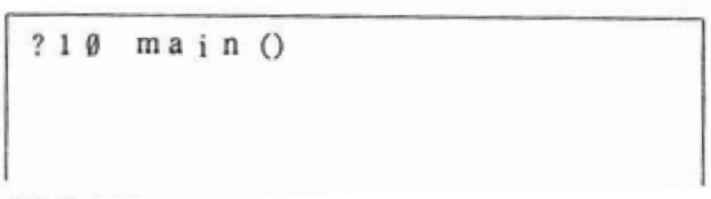
8.3 Trace

Um Fehler in einem Programm zu lokalisieren kann es hilfreich sein das Programm Schritt für Schritt auszuführen und zu beobachten was das Programm im einzelnen tut und wie der Inhalt der Variablen aussieht. Dazu kann im C-Compiler-Menü die Funktion TRACE genutzt werden.
Die Trace-Funktion wird anhand eines Beispiels erklärt.

```
10 main()
20 {
30  int i, gokei=0;
40  for (i=1;i<51;i++) {
50     gokei +=i;
60     printf(“ 1+... +%d = %d¥n“,i, gokei);
70  }
80 }
```

Starten des TRACE-Modus

Der Trace-Modus wird durch Drücken der Taste [T] im Compiler-Menü gestartet.



Es wird jeder Befehl im Display angezeigt und durch das Drücken von ENTER ausgeführt. Jeder weitere Befehl muss wiederum durch ENTER bestätigt werden. Mit der BREAK-Taste gelangen Sie in den Pause-Modus.

Funktionen im Pause-Modus:

ENTER : Verlassen des Pause-Modus und Programm fortsetzen
[C] : Verlassen des Pause-Modus und Programm fortsetzen
[A] : Abbrechen des Trace-Modus und Rückkehr in das Compiler-Menü
[T] : Trace fortsetzen
[N] : normales Fortsetzen des Programms (ohne weitere Trace-Funktionen)
[D] : Zeigt das Variablen-Menü

Im Variablen-Menü geben Sie bitte den Namen der Variablen ein um ihren Inhalt anzuzeigen.

[BREAK]

[D] i [↵]

```
1+...+1 = 1
 40   for(i=1;i<51;i++)  {
Break>D
ヘンスウ>i
int : 2(0x0002)
ヘンスウ>_
```

In diesem Beispiel hat die Variable augenblicklich den Wert 2.

Mit BREAK verlassen sie wieder das Variablen-Menü.

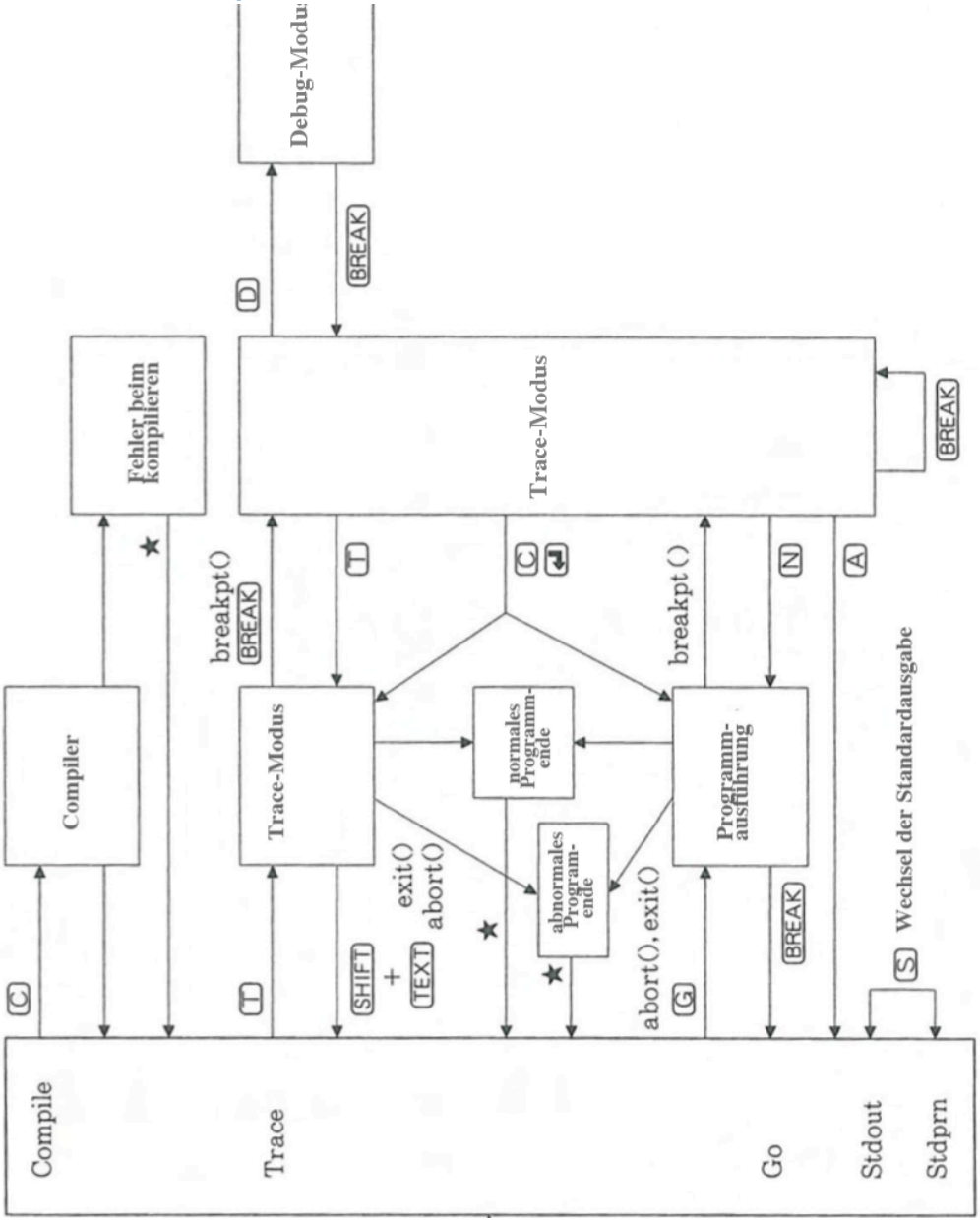
8.4 Umleiten der Bildschirmausgabe auf den Drucker

Wenn der Drucker C E-126P (separat erhältlich) angeschlossen und betriebsbereit ist, drücken Sie [S] auf dem C-Compiler-Menü-Bildschirm.
Damit wechselt die Anzeige von Stdout (Bildschirmausgabe) in stdprn (Druckerausgabe).
Um wieder die Bildschirmeingabe zu aktivieren drücken Sie erneut die Taste [S] .
Wenn im Programm explizit stdprn verwendet wird, erfolgt die Ausgabe auf dem Drucker unabhängig von der Angabe im Compiler-Menü .

Stdout: Ausgabe auf dem Bildschirm
Stdprn: Ausgabe auf dem Drucker

Folgende C-Befehle richten sich nach der Einstellung im Menü:
putc
fputc
fputs
fprintf

8.5 Funktionsbild des C-Compilers



8.6 Grundlagen der C-Programmiersprache

Dieses Kapitel geht nur auf die spezifischen Eigenschaften des C-Compilers im SHARP PC-G850 ein.

Unterstützte %-Konstruktionen für die Ausgabe (z.B. bei printf)

%-Konstruktion	Ausgabe
%d	Ganze Dezimalzahl
%x	Ganze Hexadezimalzahl
%o	Ganze Oktalzahl
%f	Gleitkommazahl
%s	Zeichenfolge
%c	Einzelzeichen

Variablentypen

Art	Variablentyp	Bereich	Größe/Länge
Integer	char	-128 bis +127	8-bit
	unsigned char	0 bis 255	8-bit
	int	-32768 bis +32767	16-bit
	unsigned int	0 bis 65535	
	short	-32768 bis +32767	16-bit
	unsigned short	0 bis 65535	16-bit
	long	-2147483648 bis +2147483647	32-bit
	Unsigned long	0 bis 4294967295	32-bit
Reale Zahlen	float	$\pm 1 \times 10^{-99}$ bis $\pm 9.999 \times 10^{+99}$	32-bit
	double	$\pm 1 \times 10^{-99}$ bis $\pm 9.999999999 \times 10^{+99}$	64-bit
	long double	$\pm 1 \times 10^{-99}$ bis $\pm 9.999999999 \times 10^{+99}$	64-bit

unsigned Durch unsigned wird ohne Vorzeichen gearbeitet. Damit steht die volle Bit-Anzahl für die Zahl zur Verfügung.

Variablennamen

- Variablenname dürfen aus Klein-, Großbuchstaben und Zahlen bestehen (keine Kana-Zeichen).
- Ein Variablename muss immer mit einem Buchstaben beginnen.
- Es dürfen keine Sonderzeichen vorkommen
- Die Länge eines Variablennamens ist maximal 31 Zeichen. Alle Zeichen ab der 32. Stelle werden ignoriert.
- Es dürfen keine Schlüsselwörter als Name verwendet werden.

Vergleichsoperatoren

Ausdruck	Erklärung
a==b	Wahr wenn a gleich b ist
a!=b	Wahr wenn a ungleich b ist
a<b	Wahr wenn a kleiner als b ist
a>b	Wahr wenn a größer b ist
a<=b	War wenn a kleiner oder gleich b ist
a>=b	Wahr wenn a größer oder gleich b ist

Arithmetische Operatoren

Operator	Bedeutung	Beispiel
+	Addition	a+b
-	Subtraktion	a-b
*	Multiplikation	a*b
/	Division	a/b
%	Modulo	a%b

Zuweisungsoperationen

Operator	Beispiel	Erklärung	Arithmetische Operation
=	a=b	Ersetze a durch b	
+=	a+=b	Addiere zu a den Inhalt von b	a=a+b
-=	a-=b	Subtrahiere von a den Inhalt von b	a=a-b
*=	a*=b	Multipliziere A mit dem Inhalt von b	a=a*b
/=	a/=b	Dividiere A mit dem Inhalt von b	a=a/b
%=	a%=b	A ergibt sich auch dem Rest der Division von a/b	a=a%b

Inkrement- und Dekrementzuweisungen

Operator	Beispiel	Bedeutung	Arithmetische Operation
++	++a	a wird um 1 erhöht und dann benutzen	a=a+1
++	a++	a wird benutzt und dann um 1 erhöhen	
--	--a	a wird um 1 erniedrigt und dann benutzt	a=a-1
--	a--	A wird benutzt und dann um 1 erniedrigt	

logische Operationen

Operator	Beispiel	Bedeutung
&&	a&&b	Logisches AND von a und b (1 wenn weder a noch b 0 ist)
||	a||b	Logisches OR von a und b (1 wenn weder a noch b 0 ist)
!	!a	Logisches NOT (wenn a<>0, dann 0,wenn a=0 dann 1)

Bit-Operationen

Operator	Beispiel	Bedeutung
&	a&b	Bitweises AND
|	a|b	Bitweises OR
^	a^b	Bitweiser XOR
~	~a	Bitweises NOT

Verschiebe-Operationen

Operator	Beispiel	Bedeutung
<<	a<<b	a bitweise nach links b-mal verschieben
>>	a>>b	A bitweise nach rechts b-mal verschieben

Schlüsselwörter

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Escape-Steuerzeichen

Steuerzeichen	Hexwert	Erklärung
¥b	0x08	Gehe zum einem Zeichen vor dem Cursor) return
¥n	0x0A	Zeilenumbruch (zum Anfang der nächsten Zeile)
¥r	0x0D	Gehe zum Anfang der Zeile
¥t	0x09	Tab (Sprung zum nächsten Tabstop)
¥¥	0x5C	Ausgabe des Zeichens ¥
¥’	0x2C	Ausgabe des Zeichens ’
¥“	0x22	Ausgabe des Zeichens “
¥?	0x3F	Ausgabe des Zeichens ?
¥ddd		Ausgabe der Zeichen, das der 3-stellige Oktalzahl ddd entspricht
¥xhh		Ausgabe der Zeichen im Hexadezimalcode

8.7 Syntax von C

Zusammengesetzte Anweisungen

Mehrere Anweisungen, die in geschweiften Klammern stehen, behandelt der Computer als eine Gruppe bzw. als Einzelanweisung. Der einzige Unterschied ist, dass hinter der Klammer am Ende der zusammengesetzten Anweisung kein Semikolon stehen muss

```
{
        Anweisung 1
        Anweisung 2
        ....
        Anweisung n
}
```

Bedingte Sprünge (if-else, switch-case)

If-else

hat folgendes Format:

1. if (ausdruck) anweisung	Wenn der ausdruck wahr ist wird die anweisung ausgeführt.
2. if(ausdruck) anweisung1 else anweisung2	Wenn der ausdruck wahr ist wird die anweisung1 ausgeführt, ist ausdruck falsch wird anweisung2 ausgeführt
3. if(ausdruck1) anweisung1 else if (ausdruck2) anweisung2 else anweisung3	Wenn der ausdruck1 wahr ist wird die anweisung1 ausgeführt, ist ausdruck falsch und ausdruck2 wahr wird anweisung2 ausgeführt. Ansonsten wird anweisung3 ausgeführt.

switch-case

hat folgendes Format_

```
switch(ausdruck) {
        case konst-ausdruck1 : anweisung1
                               [break;]
        case konst-ausdruck2 : anweisung2
                               [break;]
            . . . . . .
        case konst-ausdruckn : anweisungn
                               [break;]
        default: anweisung
}
```

Wiederholungsschleifen (for, while)

for-Schleife

```
for(ausdruck1;ausdruck2;ausdruck3)
        anweisung
```

ausdruck1: wird nur im ersten Schleifendurchgang zur Initialisierung ausgeführt.

ausdruck2: nach der Ausführung von ausdruck1 wird ausdruck2 geprüft und wenn wahr die Anweisung ausgeführt.

ausdruck3: nach der Ausführung der Anweisung wird ausdruck3 ausgeführt

Dieser Vorgang wird solange ausgeführt bis ausdruck2 unwahr wird.

while-Schleife

```
while(ausdruck)
        anweisung
```

Die anweisung wird solange wiederholt, wie der ausdruck wahr ist.

Do-while-Schleife

do

anweisung

while(ausdruck)

Zuerst wird die Anweisung ausgeführt und dann die Bedingung geprüft. Wenn wahr, wird die Anweisung erneut ausgeführt.

Unbedingter Sprung (goto, coninue, break, return)

goto-Anweisung

goto label
. . . . . .
label: anweisung

continue-Anweisung

Mit der continue-Anweisung wird der aktuelle Zyklus abgebrochen und der nächste Zyklus einer while, do-while oder for Schleife gestartet.

```
for(i=0;i<100;i++) {
    . . . . . .
   if(i%2==0)
        continue;
   printf("%d¥n",I);
}
```

break-Anweisung

Mit der break-Anweisung erfolgt der sofortige Abbruch der nächstäußeren switch, while, do-while, for Anweisung.

```
for(i=0;i<100;i++) {
    . . . . . .
   if(a[i]<0)
         break;

   . . . . . .

}
```

return-Anweisung

Mit return wird zum aufrufenden Programm zurückgekehrt. Dabei kann ein Rückgabewert übergeben werden (nicht bei void).

Beispiel:
return(ausdruck);
oder
return;

8.8 Speicherklassen

Speicherklassen dienen zur Definition von Speicherbereichen für vereinbarte Variablen sowie zur Definition des Umfangs (Bereich, aus dem/in den das Programm lesen bzw. schreiben kann).

Speicherklasse	Gültigkeitsbereich	
auto	Für kurzfristige Speicherung innerhalb eines Programms	
register	Für häufigen Zugriff. Variablen für die Erhöhung der Ausführungsgeschwindigkeit durch Zuordnung von Werten auf register (sonst wie auto)	
static	Reserviert einen Bereich während der Programmausführung. Wertezugriff und entsprechende Handlungen durch das ganze Programm.	
extern	Datei-externe oder Funktions-externe globale Variablen.	Beim

8.9 Arrays

Der C-Compiler unterstützt die Nutzung von bis zu acht-dimensionalen Arrays.

Beispiel für ein zweidimensionales Feld:

char color [3] [6] (3 Zeilen zu 6 Spalten(Zeichen))

Die gleiche Anweisung mit Zuweisung:
char color[3][6]={ “white“, “red“, “blue“};

8.10 Strukturen (struct)

Mit Hilfe von struct können Datenstrukturen erzeugt werden. Die Struktur definiert einen neuen Datentyp welcher Komponenten unterschiedlichen Typs vereint.

Die Typdeklaration

```
struct <struct_bezeichner> {
      <Datendeklaration>
    };
```

erlaubt die Deklaration von Variablen diesen Typs

```
<struct_bezeichner> <var_bezeichner>;
```

Beispiel: Deklaration eines Datentyps zur Speicherung der persönlichen Daten eines Studenten.

```
// Structure
{
// new structure
  struct Student
   {
     long long int matrikel;
     int skz;
     char name[30], vorname[20];
   };
// Variable of type Student
  Student arni,robbi;
/ Data input
  cout << endl << " Vorname : ";
  cin >> arni.vorname;
   ...
  robbi = arni;                          // complete copy
  cout << robbi.vorname << endl;
}
```

Die Zuweisung robbi = arni; kopiert den kompletten Datensatz von einer Variablen zur anderen. Der Zugriff auf die Komponente vorname der Variablen arni (des Typs Student) erfolgt über
arni.vorname
Abgespeichert werden die Daten in der Form

matrikel	skz	name	vorname

8.11 Ausführungsfunktionen vor der Kompilierung (#include,#define,#if,#ifdef)

#include "datei"

Diese Präprozessoranweisung fügt vor dem eigentlichen Kompilieren den Inhalt der Datei an der entsprechenden Stelle im Quellfile ein. Analog können bestimmte Teile des Quelltextes beim Kompilieren eingebunden oder ignoriert werden, je nach Abhängigkeit des Tests.

Mit dem #include-Befehl können z.B. Header-Dateien eingefügt werden. Dies ist in der Regel bei diesem Computer nicht notwendig.

#define name [wert]

Hiermit werden Symbole, Konstanten oder Makros des Präprozessors definiert (z.B. auch um Tests durchzuführen, siehe # ifdef und # ifndef).

Beispiele:

```
#define TEST

#define P I  3.141592
#define NULL 0
#define EOF  -1
#define FILE int
```

Es können auch Makros in definiert werden:

```
#define SQR(x)  ((x)*(x))
```

#if...#elif...#else...#endif

Mit #if kann ähnlich wie mit #ifdef eine bedingte Übersetzung eingeleitet werden, jedoch können hier konstante Ausdrücke ausgewertet werden.

```
#if <expression>
        anweisung
[#elif <expression>
        anweisung ]
[#else <expression
```

anweisung]

#endif

#ifdef name ... #endif

Mit der #ifdef-Direktive kann geprüft werden, ob ein Symbol definiert wurde. Falls nicht, wird der Code nach der Direktive nicht an den Compiler weitergegeben. Eine #ifdef-Direktive muss durch eine #endif-Direktive abgeschlossen werden.

#ifndef name ... #endif

Die #ifndef-Direktive ist das Gegenstück zur #ifdef-Direktive. Sie prüft, ob ein Symbol nicht definiert ist. Sollte es doch sein, wird der Code nach der Direktive nicht an den Compiler weitergegeben. Eine #ifndef-Direktive muss ebenfalls durch eine #endif-Direktive abgeschlossen werden.

8.12Library-Funktionen

In diesem Abschnitt werden die Bibliothekfunktionen des C-Compilers erklärt

In diesem Computer sind die Standardeingabe- und Ausgabegeräte (stream) wie folgt definiert:

Eingabe	stream : stdin	(Tastatur)
Ausgabe	stream : stdout	(Bildschirm (oder stdprn (Drucker))
Seriell	stream : stdaux	(Halbduplex-Kommunikation über 11-Pin)
	Stream : stdaux1	Vollduplex-Kommunikation über 11-Pin)

Darüber hinaus sind folgenden Konstanten standardmässig definiert:

```
#define NULL 0
#define EOF  -1
#define FILE int
```

Bei Umleitung auf den Drucker mit SHIFT (oder 2ndF) + ENTER (P<->NP) geben Input-Funktionen folgendes zurück:

getch	: 0xFF
aller anderen Inputfunktionen	: EOF

Trennzeichen bei der Eingabe über die serielle Schnittstelle:

Zeilentrennung : 0x0d, 0x0a oder 0x0d+0x0a
Dateiende : 0x1a
(Die Eingabe von 0x0d+0x0a wird nach 0x0a umgewandelt)
Sie sollten im Normalfall 0x0a als Zeilentrennzeichen verwenden.

Trennzeichen bei der Ausgabe über die serielle Schnittstelle:

Separator : null
(Die Ausgabe von 0x0a (Zeilentrennung) wird nach 0x0d+0x0a umgewandelt)

8.13Standard I/O-Funktion

getc, getchar, fgetc

Format:
```
int   getc  (FILE* stream);
int   getchar (void);
int   fgetc (FILE* stream);
```

Funktion: Es wird ein Zeichen gelesen. Wird von stdin gelesen, wird das Zeichen erst übergeben nachdem ENTER gedrückt wurde.

getc	Liest ein Zeichen aus dem angegebenen Stream
getchar	Liest ein Zeichen von stdin
fgetc	Liest ein Zeichen aus dem angegebenen Stream

Rückgabewert: ist das gelesene Zeichen

gets, fgets

Format:
```
char*  gets  (char* s);
char*  fgets (char* s, int n, FILE* stream);
```

Funktion: Es werden Zeichen gelesen und im String s gespeichert.

gets	Liest Zeichen aus stdin bis ENTER. Vor dem Abspeichern des Strings wird Wagenrücklauf/Zeilenvorschub durch 0x00 (¥0) ersetzt.
fgets	Liest Zeichen aus dem angegebenen Stream. Die Zeichen werden von der gegenwärtigen Position des Datenflusses bis zum Wagenrücklauf/Zeilenvorschub-Zeichen, auf das gestoßen wird, gelesen oder bis zum Ende der Datei (EOF) oder bis die Anzahl der gelesenen Zeichen der Zählung n-1 gleich ist. Es wird ein Nullzeichen (¥0) an das Ende der übergebenen Zeichenfolge angehängt.

Rückgabewert: Es wird Null zurückgegeben wenn das Dateiende (EOF) erreicht ist.

scanf, fscanf, sscanf

Format:
```
int   scanf  (const char* format [, address, . . . ]);
int   fscanf  (FILE* stream, const char* format [, address, . . . ]);
int   sscanf  (char* s, const char* format [, address, . . . ]);
```

Funktion: Die Funktionenfamilie scanf() prüft Eingaben in Bezug auf ein format, wie es im Folgenden beschrieben wird. Dieses Format darf Umwandlungsspezifikationen enthalten; die Ergebnisse solcher Umwandlungen, falls vorhanden, werden an den Stellen gespeichert, auf die die Zeiger-Argumente verweisen, die sich an das format halten.
Jedes Zeiger-Argument muss einen geeigneten Typ für den Rückgabewert durch die zugehörige Umwandlungsspezifikation haben.
Falls die Anzahl der Umwandlungsspezifikation in format die Anzahl der Zeiger-Argumente übersteigt, sind die Ergebnisse undefiniert. Falls die Anzahl der Zeiger-Argumente die Anzahl der Umwandlungsspezifikation übersteigt, werden die überzähligen Zeiger-Argumente ausgewertet, aber ansonsten ignoriert.

scanf	Liest Zeichen aus stdin bis ENTER.
fscanf	Liest Zeichen aus vom Eingabe-bis Wagenrücklauf/Zeilenvorschub.
sscanf	Die Zeichen werden aus dem angegebenen String s gelesen.

Rückgabewert: Anzahl der zugeordneten Argumente. Es wird EOF zurückgegeben wenn das Dateiende erreicht ist.

Format-Definition:
Die Zeichenkette format besteht aus einer Abfolge von Richtlinien, die beschreiben, wie die Abfolge der Eingabezeichen verarbeitet wird. Wenn das Verarbeiten einer Richtlinie fehlschlägt, werden keine weiteren Eingaben gelesen und scanf() kehrt zurück.

1. Leerzeichen/Wagenrücklauf
Die Eingabe wird gelesen, bis auf ein Zeichen, das kein Leerzeichen (wird ohnehin nicht gelesen) ist, gestoßen wird oder wenn keine Zeichen mehr vorhanden sind. Die Ausführung der Funktion wird beendet, wenn au fein Zeichen gestoßen wird, das kein Leerzeichen ist.

2. Normalzeichen (Andere als Leerzeichen und %)
Das nächste Zeichen wird gelesen und die Ausführung der Funktion wird beendet, wenn es kein Normalzeichen ist, und das Eingabezeichen wird nicht gelesen.

3. Umwandlungs-Definitionen

Umwandlungs -Symbol	erwartetes Format	Konvertierung
%d	String im binären Integer-Format (dezimal)	int
%i	String im binären Integer-Format (dezimal, oktal oder hexadezimal)	int
%o	String im binären Integer-Format (oktal)	int
%u	String mit einer ganzen Dezimalzahl ohne Vorzeichen.	unsigned int
%x	String mit einer ganzen Hexadezimalzahl	int
%f	String mit einer Gleitkommazahl	float
%e	“	“
%g	“	“
%c	String mit einer Zeichenfolge (Zeichenanzahl 1 oder angegebene Feldbreite)	char
%s	String (am Ende wird Null (¥0) angefügt)	
%p	String aus 4 Hex-Zeichen (z.B. 89ab)	Zeiger

Format der Umwandlungs-Anweisung

% [*] [feldbreite] [l] Umwandlungs-Zeichen

*** (Zuordnung verhindert)**
Ein Lesen in das Eingabefeld ist möglich, die Zuordnung des Umwandlungsresultats auf ein Argument ist jedoch nicht möglich.

feldbreite
die größtmögliche Feldbreite wird durch eine ganze Zahl ohne Vorzeichen definiert.

l

l Integer-Zahlen werden zu long integer umgewandelt
Gleitkommazahlen werden zum double-Format umgewandelt

L Gleitkommazahlen werden zu long double umgewandelt.

putc, putchar, fputc

Format:
```
int   putc  (int c, FILE* stream);
int   putchar (int c);
int   fputc (int c, FILE* stream);
```

Funktion: Es wird ein einzelnes Zeichen ausgegeben.

putc	Es wird ein Zeichen in den angegebenen Stream geschrieben
putchar	Es wird ein Zeichen nach stdout geschrieben
fputc	Es wird ein Zeichen in den angegebenen Stream geschrieben

Rückgabewert: ist das geschriebene Zeichen. Wenn ein Fehler beim Schreiben auftritt wird EOF zurückgegeben.

puts, fputs

Format:
```
int   puts  (const char* s);
int   fputs (const char* s, int n, FILE* stream);
```

Funktion: Es werden Zeichen aus dem String s geschrieben.

puts	Schreibt einen String nach stdout. Das das Ende der Zeichenfolge kennzeichnende Nullzeichen wird durch Wagenrücklauf/ Zeilenvorschub ersetzt.
fputs	Schreibt einen String in den angegebenen Stream beginnend an der gegenwärtigen Position des Ausgabeflusses. Das das Ende der Zeichenfolge kennzeichnende Nullzeichen wird nicht geschrieben,

Rückgabewert: ist das zu letzt geschriebene Zeichen. Wenn ein Fehler beim Schreiben auftritt wird EOF zurückgegeben.

printf, fprintf, sprintf

Format:
```
int   printf  (const char* format [, arg, . . . ]);
int   fprintf  (FILE* stream, const char* format [, arg, . . . ]);
int   sprintf  (char* s, const char* format [, arg, . . . ]);
```

Funktion: Die Funktionenfamilie printf() wandelt das „argument“ gemäß der „format“-Definition um und gibt es entweder in einem stream aus, schreibt es nach stdout oder übergibt das Ergebnis in einem String.
Die Format-Zeichenfolge ist eine Zeichenfolge größer 0 und kann sich aus Normalzeichen, ESC-Sequenzen und Umwandlungsdefinitionen zusammensetzen. Normalzeichen und ESC-Sequenzen werden in der Reihenfolge ihres Auftretens ausgegeben. Umwandlungs-Definitionen

hingegen werden durch sequentielle Extraktion der Argumente ausgeführt, wobei erst die Umwandlung und dann die Ausgabe erfolgt. Gibt es mehr Argumente als Umwandlungs-Definitionen, dann werden diese überschüssigen Argumente ignoriert. Bei zu wenig vorhanden Argumenten sind die Resultate ungewiss.

printf Schreibt die Zeichen nach stdout.
fprintf Schreibt die Zeichen in den angegebenen Stream ab der aktuellen Position.
sprintf Die Zeichen werden in den angegebenen String s geschrieben.

Rückgabewert: Anzahl der ausgegebenen Zeichen. Es wird EOF zurückgegeben wenn ein Fehlerfall eintritt.

Format-Definition:
Die Zeichenkette format besteht aus einer Abfolge von Richtlinien, die beschreiben, wie die Abfolge der Ausgabezeichen erzeugt werden.

Umwandlungs-Definitionen

Umwandlungs -Symbol	erwartetes Format	Argument
%d	Anzeige als Dezimalzahl mit Vorzeichen	int
%i	Anzeige als Dezimalzahl mit Vorzeichen	int
%o	Anzeige als Oktalzahl ohne Vorzeichen	int
%u	Anzeige als Dezimalzahl ohne Vorzeichen	int
%x	Anzeige als Hexadezimalzahl ohne Vorzeichen (abcdef)	int
%X	Anzeige als Hexadezimalzahl ohne Vorzeichen (ABCDEF)	int
%f	Anzeige als Dezimalnotation in der Form [-]ddd.ddd, wobei ddd ein einstelliger oder längerer Dezimalwert ist.	double
%e	Anzeige als Dezimalnotation in der Form [-]d.ddde±dd , wobei d ein einstelliger Dezimalwert ist, ddd ein- oder mehrstellig ist.	double
%E	Anzeige als Dezimalnotation in der Form [-]d.dddE±dd , wobei d ein einstelliger Dezimalwert ist, ddd ein- oder mehrstellig ist.	double
%g	Wandelt f oder e in eine verkürzte Form um	double
%G	Wandelt f oder E in eine verkürzte Form um	double
%c	Umwandlung in ein Zeichen ohne Vorzeichen	int
%s	Zeichen der Zeichenfolge werden ausgegeben, bis die Null (¥0) erreicht wird (wird nicht mit ausgegeben) oder bis die angegebene Anzahl von Zeichen erreicht sind.	String (char *)
%p	Ausgabe als Zeigerargument	Zeiger

Format der Umwandlungs-Anweisung

% [flag] [feldbreite][.genauigkeit] [l] Umwandlungs-Zeichen

flag

-	linksbündige Ausgabe
+	Vorzeichen wird immer ausgegeben
#	bei einer %o-Konvertierung wird eine 0 vorangestellt bei einer %x und %X-Konvertierung wird ein 0x (oder 0X) vorangestellt.
0	Ergebnis mit führenden Nullen auffüllen (bei %d, %i, %o, %u, %x, %X)
(weggelassen)	rechtsbündige Ausgabe

feldbreite

n	Gibt die Anzahl der auszugebenden Stellen an. Sind weniger Zeichen vorhanden mit wird Leerzeichen aufgefüllt.
0n	Das Feld bekommt die Länge n. Ist das Ergebnis der Umwandlung kürzer als n wird das Ergebnis mit Nullen aufgefüllt
(weggelassen)	Die Länge wird durch das Umwandlungsergebnis definiert.

.genauigkeit

n	Definiert bei %d, %o, %u, %x, %f die kleinste Anzahl von Stellen zur Ausgabe (Standard ist 1) Bei Gleitkomma-Zahlen (%e,%E%f) wird die Anzahl der Dezimalstellen nach dem Komma definiert (Standard ist 6) Bei dem Format %g und %G wird damit die größte Anzahl der ausgegeben Zeichen festgelegt (Standard: alle bedeutsamen Zeichen)

l

Definiert bei %d, %i, %o, %, %x, %X die Ausgabe als langes Argument (long)

fflush

Format: int fflush (FILE* stream) ;

Funktion: Schreibt den Inhalt des Pufferspeichers in die Datei bei einem Ausgabe-Stream. Bei einem Eingabe-Stream wird der Inhalt des Pufferspeichers gelöscht. Diese Funktion schließt nicht den Stream. Der Pufferspeicher wird automatisch abgeschlossen (flush) wenn er voll ist.

Rückgabewert: Null. Wenn ein Fehler beim Schreiben auftritt wird EOF zurückgegeben.

clearerr

Format: void clearerr (FILE* stream) ;

Funktion: Diese Funktion löscht einen Datenfluß-EOF und einen Fehlerzustand

8.14Character-Funktionen

isalnum, isalpha, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit

Format:
```
int isalnum (int c) ;
int isalpha (int c) ;
int iscntrl (int c) ;
int isdigit (int c) ;
int isgraph (int c) ;
int islower (int c) ;
int isprint (int c) ;
int ispunct (int c) ;
int isspace (int c) ;
int isupper (int c) ;
int isxdigit (int c) ;
```

Funktion: Charakterisierung eines Zeichens

isalnum	prüft ob Zeichen ein Buchstabe oder eine Ziffer ist
isalpha	prüft auf Buchstaben
iscntrl	prüft auf Kontrollzeichen
isdigit	prüft auf Ziffern
isgraph	prüft auf druckbare Zeichen außer Leerzeichen
islower	prüft auf Kleinbuchstaben
isprint	prüft auf druckbare Zeichen inklusive Leerzeichen
ispunct	prüft auf druckbare Zeichen, das kein Leerzeichen und kein alphanumerisches Zeichen ist.
isspace	prüft auf Freizeichen (Leerzeichen, Tabulatoren, Zeilenumbrüche Zeilenumbrüche und so weiter, 0x09~0x0d,0x20)
isupper	prüft auf Großbuchstaben
isxdigit	prüft auf ein hexadezimales Zeichen (0-F,0-f)

Rückgabewert: wahr (Wert ungleich null) oder falsch (null)

tolower, toupper

Format: int tolower (int c) ;
int toupper (int c) ;

Funktion:

tolower	wandelt das Zeichen zu Kleinbuchstaben um
toupper	wandelt das Zeichen zu Großbuchstaben um

Rückgabewert: das umgewandelte Zeichen

8.15String-Funktionen

strcat

Format: char * strcat (char * s1, const char * s2) ;

Funktion: Hängt die Zeichenkette s2 an die Zeichenkette s1 an.

Rückgabewert: Pointer auf Zielstring s1

strchr

Format: char * strchr (const char * s, int c) ;

Funktion: Sucht einen String nach dem ersten Auftreten eines bestimmten Zeichens ab.

Rückgabewert: strchr liefert einen Zeiger auf die erste Fundstelle des Zeichens c im String s zurück bzw. den Wert NULL, wenn der String dieses Zeichen nicht enthält.

strcmp

Format: int strcmp (const char * s1, const char * s2) ;

Funktion: Vergleicht zwei Strings miteinander.
Beginnend mit dem ersten Zeichen werden die beiden Strings zeichenweise verglichen, bis zwei korrespondierende Zeichen ungleich sind oder das Ende der Strings erreicht wird.

Rückgabewert:
strcmp liefert einen Wert
< 0 wenn s1 kleiner als s2 ist
== 0 wenn s1 gleich s2 ist
> 0 wenn s1 größer als s2 ist

strcpy

Format: char * strcpy (char * s1, const char * s2) ;

Funktion: Kopiert einen String in einen anderen.
strcpy kopiert den Inhalt des über s2 angegebenen Strings in den durch s1 angegebenen Speicherbereich.
Das abschließende Nullzeichen von s2 wird als letztes Zeichen kopiert.

Rückgabewert: Pointer auf Zielstring s1

strlen

Format: int strlen (const char * string) ;

Funktion: Bestimmt die Länge einer Zeichenkette

Rückgabewert: strlen liefert die Anzahl der Zeichen des Strings zurück. Das abschließende Nullzeichen wird dabei nicht mitgezählt.

8.16Speicher-Funktionen

calloc

Format: void *calloc (unsigned n, unsigned size) ;

Funktion: reserviert eine Gruppierung von n Elementen, wobei jedes Element von der Größe size ist.

Rückgabewert: Zeiger auf den reservierten Speicherbereich. Bei erfolgloser Speicherreservierung (z.B. wenn RAM-Größe überschritten wurde) wird Null zurückgegeben.

malloc

Format: void * malloc (unsigned size) ;

Funktion: reserviert einen Speicherbereich mit der Größe size.

Rückgabewert: Zeiger auf den reservierten Speicherbereich. Bei erfolgloser Speicherreservierung (z.B. wenn RAM-Größe überschritten wurde) wird Null zurückgegeben.

free

Format: void *free(void * ptr) ;

Funktion: gibt den durch calloc oder malloc reservierten Speicherbereich frei. Das Argument ptr muss der Zeiger des von calloc oder malloc reservierten Speicherbereichs sein.

Rückgabewert: kein

8.17Mathematische Funktionen

abs

Format: int abs(int x) ;

Funktion: gibt den absoluten Wert einer ganzen Zahl zurück.

Rückgabewert: die absolute Zahl (0-32767).

asin, acos, atan

Format:
```
double  asin(double x) ;
double  acos(double x) ;
double  atan(double x) ;
```

Funktion: diese Funktionen geben einen Wert zurück der ihrer jeweiligen inversen trigonometrischen Funktion entspricht. Die Angabe kann in DEG, RAD oder GRAD erfolgen.
Der Kalkulationsbereich bei asin und acos liegt zwischen -1 und 1

Rückgabewert: der jeweilige Wert für das Ergebnis. Null bei einem Fehler.

Funktion	Wertebereich		
	DEG	RAD	GRAD
asin	-90° bis 90°	-π/2 bis π/2	-100° bis 100°
acos	0° bis 180°	0 bis π	0° bis 200°
atan	-90° bis 90°	-π/2 bis π/2	-100° bis 100°

asinh, acosh, atanh

Format:
```
double  asinh(double x) ;
double  acosh(double x) ;
double  atanh(double x) ;
```

Funktion: diese Funktionen geben einen Wert zurück der ihrer jeweiligen inversen Hyperbelfunktion entspricht.

Rückgabewert: der jeweilige Wert für das Ergebnis.

exp

Format: double exp(double x) ;

Funktion: errechnet die Exponentialfunktion von x (e^x)

Rückgabewert: das Ergebnis.

log, log10

Format: double log(double x) ;
double log10(double x) ;

Funktion: errechnet den natürlichen Logarithmus von x. Die log10-Funktion errechnet den Zehnerlogarithmus von x.

Rückgabewert: das Ergebnis.

pow

Format: double pow(double x, doube y) ;

Funktion: diese Funktion erhebt x zur Potenz von y.

Rückgabewert: das Ergebnis.

sin, cos, tan

Format: double sin(double x) ;
double cos(double x) ;
double tan(double x) ;

Funktion: diese Funktionen geben einen Wert zurück der ihrer jeweiligen trigonometrischen Funktion entspricht. Die Angabe kann in DEG, RAD oder GRAD erfolgen.

Rückgabewert: der jeweilige Wert für das Ergebnis.

sinh, cosh, tanh

Format: double sinh(double x) ;
double cosh(double x) ;
double tanh(double x) ;

Funktion: diese Funktionen geben einen Wert zurück der ihrer jeweiligen Hyperbelfunktion entspricht.

Rückgabewert: der jeweilige Wert für das Ergebnis.

sqrt

Format: double sqrt(double x) ;

Funktion: errechnet die Quadratwurzel von x

Rückgabewert: das Ergebnis.

8.18Hardware-Schnittstellen-Funktion

Dieser Abschnitt beschreibt die Hardware-spezifischen I/O-Funktionen.

Mini-I/O-Funktionen

miniget

Format: int miniget(void) ;

Funktion: Liest ein Byte aus dem Mini-I/O-Port
Bit 2: Xin, Bit 1: Din, Bit: Ack

Rückgabewert: das gelesene Byte

miniput

Format: void miniput(char byte) ;

Funktion: Schreibt ein Byte auf den Mini-I/O-Port
Bit 2: Busy, Bit 1: Dout, Bit: Xout

Rückgabewert: keiner

8-Bit PIO-Steuerung über das 11-Pin-Interface

fclose

Format: int fclose(FILE* stream) ;

Funktion: Abschließen des Streams

Rückgabewert: bei erfolgreicher Ausführung wird NULL zurückgegeben. Im Falle eines Fehlers wird EOF zurückgegeben.

fopen

Format: FILE * fopen(char *path, char *type) ;

Funktion: Öffnet einen Stream zum Gerät das mit path spezifiziert ist mit dem unter type angegeben Modus.
Der path ist bei 8-Bit-PIO „pio“ und der Modus kann Eingabe “r+“, Ausgabe “w+“ oder “a+“ Ein- und Ausgabe sein.

Rückgabewert: bei normaler Ausführung wird der Zeiger auf die FILE-Struktur zurückgegeben. Im Fehlerfall wird NULL zurückgegeben.

pioget

Format: int pioget(void) ;

Funktion: liest ein Byte vom PIO-Port ein.

Rückgabewert: das gelesene Byte.

pioput

Format: void pioput(char byte) ;

Funktion: schreibt ein Byte zum PIO-Port.

Rückgabewert: keinen

pioset

Format: void pioset(char byte) ;

Funktion: setzte den Signalein- und ausgangsmodus des PIO-Ports.
1 setzt den Eingabemodus und 0 setzt den Ausgabemodus

Rückgabewert: keinen

SIO (RS-232C) Steuerung über das 11-Pin-Interface

fclose

Format: int fclose(FILE* stream) ;

Funktion: Abschließen des Streams. Beim Ausgabe-Modus wird abschließend das EOF-Zeichen geschrieben, das durch die Eintragung im TEXT-Modus unter SIO mit dem end-of-file-Parameter bestimmt wurde.

Rückgabewert: bei erfolgreicher Ausführung wird NULL zurückgegeben. Im Falle eines Fehlers wird EOF zurückgegeben

fopen

Format: FILE * fopen(char *path, char *type) ;

Funktion: Öffnet einen Stream zum Gerät das mit path spezifiziert ist mit dem unter type angegeben Modus.
path-Definition beim seriellem Interface:
“stdaux“ : Halbduplex-Kommunikation
“stdaux1“ : Vollduplex-Kommunikation
Spezifikation des Modus (type):
“r+“ : Eingabemodus
“w+“ : Ausgabemodus
“a+“ : Ein- und Ausgabemodus

Rückgabewert: bei normaler Ausführung wird der Zeiger auf die FILE-Struktur zurückgegeben. Im Fehlerfall wird NULL zurückgegeben.

Puffer-/Kommunikationssteuerung

feof

Format: int feof(FILE* stream) ;

Funktion: prüft, ob der Stream das Ende der Datei (EOF) erreicht hat.

Rückgabewert: Bei Erreichen des Dateiendes erfolgt die Rückgabe eines Wertes von -1. Ist das Dateiende noch nicht erreicht wird NULL zurückgegeben.

I/O-Port-Funktion

inport

Format: unsigned char inport(unsigned char port) ;

Funktion: Liest ein Byte aus der angegeben I/O-Port-Adresse (0x20-0x3F)

Rückgabewert: das gelesene Byte

outport

Format: void outport(unsigned char port, unsigned char byte) ;

Funktion: Schreibt ein Byte in die angegebe I/O-Port-Adresse (0x20-0x3F)

Rückgabewert: keinen

Speicher-Funktionen, Programmaufruf

call

Format: unsigned call(unsigned adr, void* arg_HL) ;

Funktion: ruft ein Maschinenspracheprogramm auf, das beginnend ab der Adresse adr steht. Der Wert des Arguments arg_HL wird im HL-Register übergeben.

Rückgabewert: Inhalt des HL-Registers

peek

Format: unsigned char peek(unsigned adr) ;

Funktion: Liest ein Byte von der Adresse adr.

Rückgabewert: Inhalt der Speicheradresse adr

poke

Format: void poke(unsigned adr, unsigned char byte) ;

Funktion: Schreibt ein Byte an die Adresse adr.

Rückgabewert: keinen

8.19Datendatei-Funktionen

fclose

Format: int fclose(FILE* stream) ;

Funktion: Abschließen des Datei-Streams. Wenn die Datei im Modus “w“ oder “a“ geöffnet wurde, wird noch ein 0x1A als Ende-Zeichen geschrieben.

Rückgabewert: bei erfolgreicher Ausführung wird NULL zurückgegeben. Im Falle eines Fehlers wird EOF zurückgegeben

feof

Format: int feof(FILE* stream) ;

Funktion: prüft, ob der Stream das Ende der Datei (EOF) erreicht hat.

Rückgabewert: Bei Erreichen des Dateiendes erfolgt die Rückgabe eines Wertes von -1. Ist das Dateiende noch nicht erreicht wird NULL zurückgegeben.

flof

Format: unsigned long flof(FILE* stream) ;

Funktion: ermittelt die noch ungenutzten Bytes in der Datei.

Rückgabewert: die Anzahl der noch ungenutzten Bytes.

fopen

Format: FILE * fopen(char *path, char *type) ;

Funktion: Öffnet einen Stream zu einer Datei die mit path spezifiziert ist mit dem unter type angegeben Modus.
Die path-Definition für eine Datendatei entspricht dem Dateinamen der vorher im TEXT-Mode definiert/angelegt wurde.
Spezifikation des Modus (type):
“r“ : Eingabemodus
“w“ : Ausgabemodus
“a“ : Ein- und Ausgabemodus

Rückgabewert: bei normaler Ausführung wird der Zeiger auf die FILE-Struktur zurückgegeben. Im Fehlerfall wird NULL zurückgegeben.

8.20Grafik-Funktionen

Die hier beschriebenen Grafik-Funktionen entsprechen den Basic-Funktionen. Für genauere Angaben nutzen Sie bitte auch die Beschreibungen der entsprechenden Basic-Befehle.

circle

Format: int circle(int x, int y, int r, double s-angle, double e-angle, double ratio, int reverse, unsigned short kind) ;

Funktion: zeichnet einen Kreis
x,y : Koordinate des Mittelpunkts
r : Radius
s-angle : Startwinkel
e-angle : Endwinkel
ratio : Verhältnis für die Ellipse
reverse : 0: Punkt setzen

1: Punkt löschen
2: Punkt invertieren
kind : Füllmuster (Siehe die Beschreibung des Basic-Befehls)

Rückgabewert: bei erfolgreicher Ausführung wird NULL zurückgegeben. Im Falle eines Fehlers wird -1 zurückgegeben

gcursor

Format: int gcursor(int x, int y) ;

Funktion: positioniert den Grafik-Cursor auf dem gewünschten Display-Punkt x,y.

Rückgabewert: bei erfolgreicher Ausführung wird NULL zurückgegeben. Im Falle eines Fehlers wird -1 zurückgegeben

gprint

Format: void gprint(char * image) ;

Funktion: zeichnet Grafik-Muster auf dem Display.

Rückgabewert: keinen

line

Format: int line(int x, int y, int x2, int y2, int reverse, unsigned short mask, int rectangle) ;

Funktion: zeichnet eine Linie oder ein Rechteck.
reverse : 0: Punkt setzen
1: Punkt löschen
2: Punkt invertieren
mask : Linienart (Siehe die Beschreibung des Basic-Befehls)
rectangle : 0: zeichnet eine Linie
1: zeichnet ein Rechteck
2: zeichnet ein gefülltes Rechteck

Rückgabewert: bei erfolgreicher Ausführung wird NULL zurückgegeben. Im Falle eines Fehlers wird -1 zurückgegeben

paint

Format: int paint(int x, int y, unsigned short kind) ;

Funktion: füllt ab der Koordinate <x>,<y> die Fläche mit dem Muster aus.
kind : Füllmuster (Siehe die Beschreibung des Basic-Befehls)

Rückgabewert: bei erfolgreicher Ausführung wird NULL zurückgegeben. Im Falle eines Fehlers wird -1 zurückgegeben

point

Format: int point(int x, int y) ;

Funktion: liefert eine Information über den Zustand des spezifizierten Display-Punktes

Rückgabewert: Ist der Punkt dunkel, also gesetzt, so wird 1 zurückliefert. Ist der Punkt nicht gesetzt, wird der Wert 0 zurückgegeben. Befindet sich der Punkt außerhalb des Bildschirms wird -1 zurückgegeben.

preset

Format: int preset(int x, int y) ;

Funktion: löscht den Display-Punkt.

Rückgabewert: bei erfolgreicher Ausführung wird NULL zurückgegeben. Im Falle eines Fehlers wird -1 zurückgegeben

pset

Format: int pset(int x, int y, int reverse) ;

Funktion: Setzt den Display-Punkt.
reverse : 0 setzt den Punkt
1 invertiert den Punkt

Rückgabewert: bei erfolgreicher Ausführung wird NULL zurückgegeben. Im Falle eines Fehlers wird -1 zurückgegeben

8.21Sonstige Funktionen

abort, exit

Format: void abort(void) ;
void exit (int status) ;

Funktion: verlässt/beendet das Programm.
abort : bricht das Programm ab. A B O R T wird auf dem Bildschirm angezeigt.
exit : normale Programmbeendigung mit Return-Code

Rückgabewert: keinen

angle

Format: void angle(unsigned n) ;

Funktion: setzt den Modus für die trigonometrischen Funktionen.
n=0 : DEG
n=1 : RAD
n=2 : GRAD

Rückgabewert: keinen

breakpt

Format: void breakpt(void) ;

Funktion: unterbricht die Programmausführung und geht in den BREAK-Modus.

Rückgabewert: keinen

clrscr

Format: void clrscr(void) ;

Funktion: Löscht den Bildschirm

Rückgabewert: keinen

getch

Format: int getch(void) ;

Funktion: Wartet auf ein Zeichen von der Tastatur. Es wird nicht auf ENTER gewartet.

Rückgabewert: gibt das gelesene Zeichen zurück

gotoxy

Format: void gotoxy(unsigned x, unsigned y) ;

Funktion: Setzt den Text-Cursor auf die angegebene Koordinate des Bildschirms.
(0,0) ist die obere linke Ecke.

Rückgabewert: keinen

kbhit

Format: int kbhit(void) ;

Funktion: Liest ein Zeichen von der Tastatur ohne zu warten.

Rückgabewert: gibt die gelesene Taste zurück. Ist keine Taste gedrückt wird 0 zurückgegeben.

8.22Fehlermeldungen

Compiler-Fehlermeldung

Fehlermeldung	Beschreibung
Null dimension	Eine Dimension eines Arrays ist null in einem Kontext, wo das verboten ist.
array of function is illegal	Array of function ist nicht erlaubt
can't find include file	Die include-Datei kann nicht gefunden werden
case not in switch	case-Anweisung befindet sich nicht innerhalb einer switch-Anweisung
constant expected	- Die Elementeanzahl des Arrays ist nicht ganzzahlig - Der Ausdruck der case-Marke ist kein Ganzzahlkonstantenausdruck
default not in switch	default-Anweisung befindet sich nicht innerhalb einer switch-Anweisung
define buffer full	Zu viele #define-Anweisungeen
different s/u	unterschiedliche structure/union
division by 0	Division durch 0
duplicate #define <name>	Doppelt definiertes Macro
duplicate case	Es wurden im Switch-Statement zwei gleiche case-Anweisungen gefunden.
duplicate default	Es wurden im Switch-Statement mehr als eine default-Anweisung gefunden.
duplicate label : <name>	Ein Label wurde mehr als einmal definiert.
empty character constant	Die Konstante hat keinen Inhalt
float overflow	Gleitkomma-Konstante außerhalb des Bereichs
float underflow	Gleitkomma-Konstante außerhalb des Bereichs
function illegal in s/u	Function in Structure/Union nicht erlaubt
function returns illegal type	Die Art des Rückgabewertes ist in dieser Funktion nicht erlaubt
if nest too deep	Zu viele Verschachtelung von #if/#ifdef-Anweisungen
if nesting error	#if/#ifdef-#endif Strukturfehler
if –less elif	Kein #if/#ifdef zum #elif gefunden
if –less else	Kein #if/#ifdef zum #else gefunden
if –less endif	Kein #if/#ifdef zum #endif gefunden
illegal # line	Falsche #define-Syntax
illegal break	Eine break-Anweisung innerhalb einer do-, for-, while- oder switch-Schleife ist nicht erlaubt
illegal character	Illegales Zeichen im Source-Code
illegal class	Die definierte Klasse kann nicht verwendet werden
illegal continue	Eine continue-Anweisung innerhalb einer do-, for-, while-Schleife ist nicht erlaubt
illegal digit in octal	Illegale Ziffer in einer Oktalzahl (8 oder 9).
illegal function	Aufruf einer Function die nicht dem Typ entspricht

illegal if	Falscher Ausdruck in der #if/#ifdef-Anweisung
illegal include	Die Syntax der #include-Anweisung ist falsch
illegal indirection	Operand der unären Operators * ist ungültig
illegal initialisation	Die rechten Seite der Initialisierung ist nicht ein konstanter Ausdruck
illegal main	Es wurde ein Argument in der Funktion main deklariert
illegal operand of <operator>	Der Typ des Operanden des Operators falsch
illegal operand of U+	Der Typ des Operanden des unären Operator + ist ungültig
illegal operand of U -	Der Typ des Operanden des unären Operator - ist ungültig
illegal operand of ARG	Die Typen der Argumente der Funktion sind ungültig
illegal operand of RET	Die Art des Ausdrucks der return-Anweisung ist falsch
illegal s/u	struct/union wurden falsch benutzt.
illegal size	Die Größe der Struktur / Union zu groß
illegal switch expression	Im switch-Befehl ist die expression ungültig
illegal type	Es ist eine ungültige Typumwandlung aufgetreten
illegal void	Die Verwendung von Typ void ist nicht korrekt
Include nest too deep	Die #include Verschachtelung ist zu tief
macro recursion	Makro ist rekursiv
memory full	Der Speicher ist voll
missing argument : <name>	Im Funktionsaufruf fehlt ein Argument
missing declarator	Es gibt keine Deklaration
missing function : <name>	Die Funktion <name> wurde nicht deklariert
missing label	- Kein Label in der goto-Anweisung - Das Label in Goto wurde nicht definiert
missing main	Main() wurde nicht definiert
missing member	- Die struct / union Mitglieder wurden nicht definiert - Es wurden die Mitglieder, die nicht in dem Ausdruck verwendet werden definiert
missing member in s/u	In struct/union fehlt das Member
missing name in prototype	Es fehlt ein Argument –Name in der Prototyp-Definition
missing type	Der Typ wurde nicht definiert.
missing type in prototype	Es gibt eine Syntax Regelverletzung im Prototyp
newline in character constant	Zeilenumbruch in der Zeichenkonstante
newline in string constant	Zeilenumbruch in der String-Konstante
prototype unmatch	Expression des Funktionsaufrufs passt nicht zu dem Prototyp
redeclaration : <name>	Es wurde ein anderer Name definiert
reserved : <name>	Der <name> ist reserviert
syntax error	Im Programm wurden die Syntax-Regeln verletzt
token buffer full	Die Macro-Expansion ist zu komplex
too complicated declarator	Die Definition ist zu komplex
too complicated declaration	Die Definition ist zu komplex
too complicated initialize	Die Initialisierung ist zu komplex
too deep statement	Zu tiefe Verschachtelung
too long initializer	String-Konstante ist zu lang in der Initialisierung

too long macro	Der Macro-Text ist zu lang
too many #define	Anzahl der #define überschreiten den Grenzwert
too many case	Anzahl der case-Anweisungen überschreiten den Grenzwert
too many characters in character constant	Anzahl der Zeichen in der Konstante größer als der Grenzwert
too many characters in string constant	Anzahl der Zeichen in der String-Konstanten überschreitet den Grenzwert
too many initializers	Es wurden zu viele Initialisierungs-Ausdrücke deklariert
too many label	Die Anzahl der goto-Markierungen ist größer als der Grenzwert
too many prototype	Die Anzahl der Prototyp-Definitionen überschreitet den Grenzwert
unacceptable operand of &	Der &-Operand des Operators ist ungültig
unexpected EOF	Das Quell-Programm endet mitten in der Syntax
unknown size	Die Größe ist unbestimmt.
void function	Sie gibt einen Wert in der return-Anweisung, obwohl es eine void-Funktion ist
zero or negative subscript	Negative oder Null Anzahl der Elemente in dem Array

Run-time-Error Meldungen

Fehlermeldung	Beschreibung
NO MEMORY	Speicherüberlauf
BAD POINTER	Der Pointer zeigt außerhalb des erlaubten Bereichs
DIVISION BY 0	Division durch 0
UNKNOWN ERROR	Durch falsche Zeiger wurden die Bereich des C-Programms zerstört
BAD FUNCTION	Beim Aufruf einer Funktion ist der Pointer-Wert falsch
BAD STREAM	Der Datenstrom für Ein- und Ausgabe ist nicht korrekt
ARITHMETIC ERROR	Berechnungsfehler (z.B. bei Gleitkomma-Brechnungen)
FRAME ERROR	Der Funktions-Rahmen wurde zerstört
I/O OPEN ERROR	Serielles Interface wurde zu oft geöffnet
I/O ERROR	Nicht geöffnetes Mini-I/O

9 CASL

9.1 Der CASL-Assembler

Das CASL-Assembler-System wurde für das Erlernen der Assembler-Sprache entwickelt, um die internen Prozesse eines Computers zu verstehen.
Das System besteht aus zwei Teilen, dem CASL-Assembler und der virtuellen COMET-Maschine. Beim Entwickeln des COMET-Systems wurde sehr darauf geachtet, dass sämtliche Zustände des Systems überwacht und verfolgt werden können. Leider hat sich dieses System nicht über die Grenzen von Japan und den Philippinen hinaus etabliert. Um in Japan die Prüfung für die „Japanese Information Technology Standards Examination“ (JITSE) zu bestehen muss (musste, Stand 2013 wahrscheinlich nicht mehr!?!) auch eine Prüfung in CASL/COMET abgelegt werden. Die Erweiterung zu CASL II und COMET II von 2001 unterstützt dieser Computer nicht.

Dieses Handbuch erklärt die Spezifikation CASL II / COMET II mit dem Schwerpunkt auf dem Umgang damit beim Sharp PC-G850

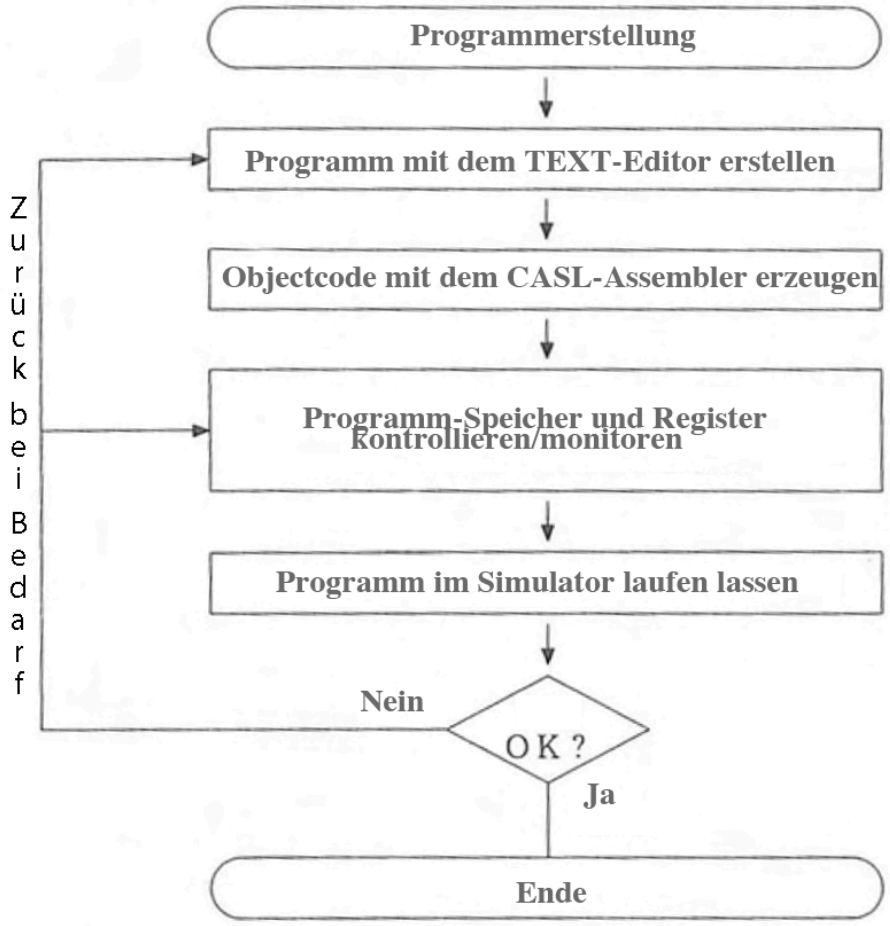
9.2 Konfiguration de CASL-Modus

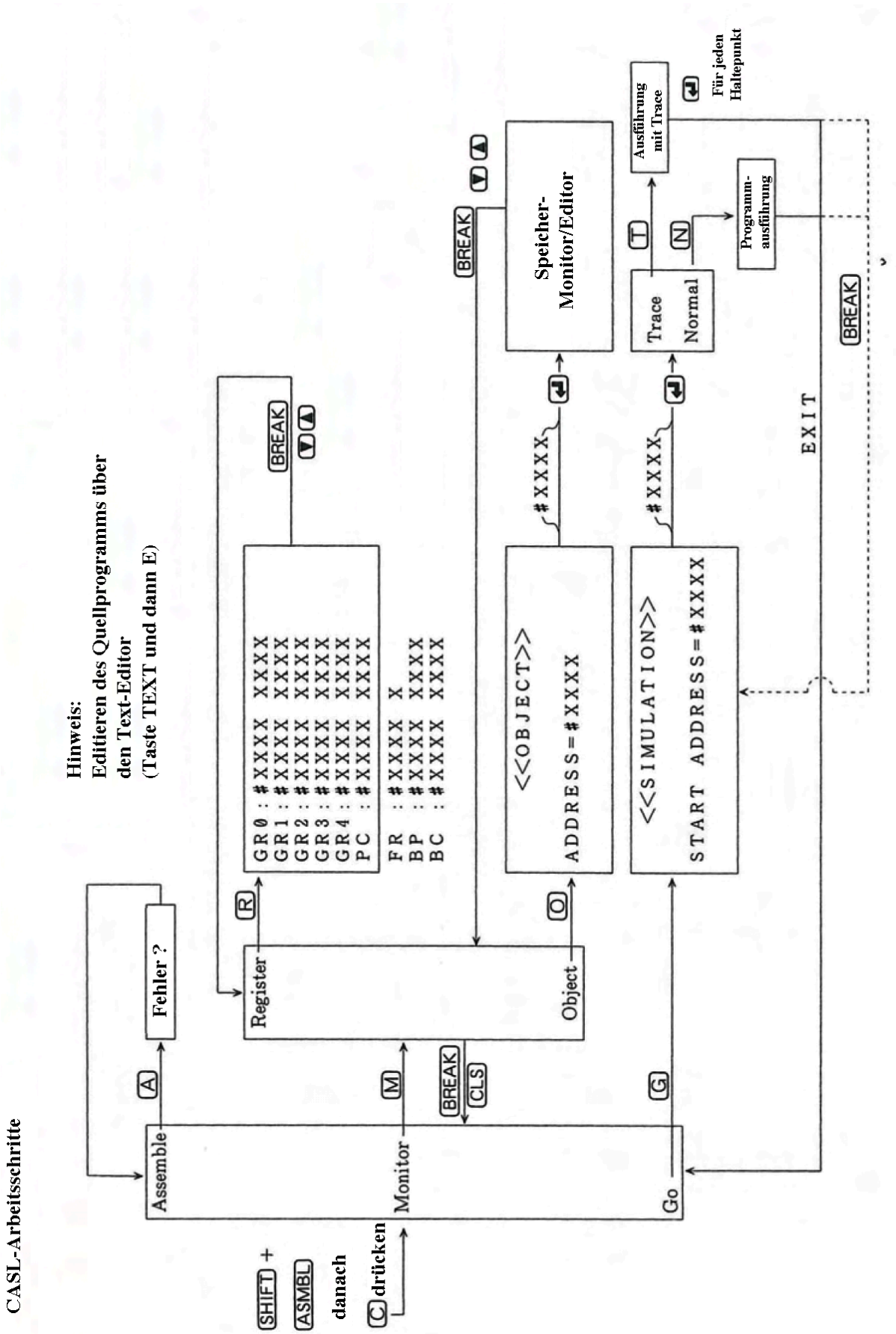
Der CASL-Modus besteht aus drei Funktionen:

Assembler:	Sie können mit dem TEXT-Editor den CASL-Programm-Source-Code wie gewohnt schreiben und speichern. Dieses Programm können sie dann im CASL-Modus assemblieren und ausführen. Wenn der Drucker CE-126P angeschlossen ist, können die Ausgaben auf den Drucker umgeleitet werden.
Überwachung:	Sie können das Programm und den Inhalt der Register anzeigen und ändern. Genauso können sie die mit dem DS-Befehl definierten Datenbereiche ändern.
Simulierte Ausführung:	Das Programm wird in einer abgeschlossenen Shell ausgeführt. Das Programm kann normal oder auch im Trace-Modus ausgeführt werden. Die Ausführung des Programms kann an definierten Haltepunkten gestoppt werden.

9.3 Prinzipielle Vorgehensweise bei der Programmierung

Prinzipielle Reihenfolge der Vorgehensweise:




CASL-Arbeitsschritte
SHIFT +
ASMBL
danach
C drücken
Assemble
Monitor
Go
A
M
G
Fehler ?
BREAK
CLS
Register
Object
R
O
BREAK
GR0 : #XXXX XXXX
GR1 : #XXXX XXXX
GR2 : #XXXX XXXX
GR3 : #XXXX XXXX
GR4 : #XXXX XXXX
PC : #XXXX XXXX
FR : #XXXX X
BP : #XXXX XXXX
BC : #XXXX XXXX
<<OBJECT>>
ADDRESS=#XXXX
#XXXX
<<SIMULATION>>
START ADDRESS=#XXXX
#XXXX
Speicher-
Monitor/Editor
Trace
Normal
T
N
Ausführung
mit Trace
Programm-
ausführung
Für jeden
Haltepunkt
EXIT
BREAK
Hinweis:
Editieren des Quellprogramms über
den Text-Editor
(Taste TEXT und dann E)

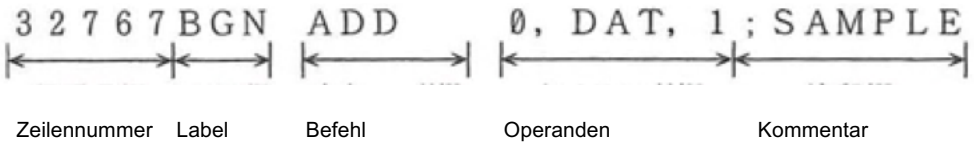
9.4 Eingabe/Bearbeiten des Source-Programms

Das CASL-Source-Programm wird erstellt und modifiziert im TEXT-Modus mit Hilfe des Editors.

Detaillierte Informationen wie man TEXT-Modus zu verwenden, finden Sie in Kapitel "TEXT-Modus".

Eingabeformat des Source-Programms

Struktur des Quellprogramms:



Die Trennung zwischen den einzelnen Operanden kann mit Leerzeichen oder mit der Tab-Taste erfolgen.

Zeilennummer: Für die Zeilennummer kann eine Zahl zwischen 1 bis 65279 verwendet werden. Wird eine Zahl außerhalb des Bereichs von 1 bis 65279 angegeben wird die Meldung „LINE NO. ERROR“ ausgegeben.

Label: Das Label darf 6 Stellen aus Alphanummerischen Zeichen besitzen. Alle folgenden Zeichen werden ignoriert. Das Label muss mit einem Buchstaben beginnen

Befehl: Dies ist der Befehl der ausgeführt werden soll.

Operanden: GR-Register, Adress-Operanden oder XR. Jeder Operand muss durch ein Komma getrennt werden. XR kann weggelassen werden.

Kommentar: Kommentare müssen mit einem Semikolon (;) anfangen und dienen dazu in dem Programm Notizen einzufügen.

Eine Zeile darf einschließlich des Kommentars eine maximale Länge von 254 Zeichen besitzen

Beispielprogramm

Dieses Programm erzeugt die Ausgabe 「♠♥CARDS♦♣」

```
10L1    START  L2
20L2    OUT    DSP, N
30      EXIT
40N     DC     9
50DSP   DC     #E8
60      DC     #E9
70      DC     'CARDS'
80      DC     #EA
90      DC     #EB
100     END
```

[TEXT] [E]

```
10L1 [TAB] START [TAB] L2 [↵]
20L2 [TAB] OUT [TAB] DSP, N [↵]
30 [TAB] EXIT [↵]
40N [TAB] DC [TAB] 9 [↵]
50DSP [TAB] DC [TAB] #E8 [↵]
60 [TAB] DC [TAB] #E9 [↵]
70 [TAB] DC [TAB] 'CARDS' [↵]
80 [TAB] DC [TAB] #EA [↵]
90 [TAB] DC [TAB] #EB [↵]
100 [TAB] END [↵]
```

L2 ist Startpunkt des Programms
Ausgabe der Zeichen bei DSP mit Länge in N
Rücksprung, Ende der Programmausführung

```
「♠」
「♥」
「CARDS」
「♦」
「♣」
```

Programmende

9.5 Der CASL-Assembler

Nach dem der Sourcecode im Text-Editor erstellt wurde werden alle weiteren Schritte im CASL-Modus durchgeführt.

[SHIFT] + [ASMBL] Danach [C] drücken
um in den CASL-Modus zu gelangen

```
*** CASL ***
Assemble Monitor Go
```

[A] drücken um den Sourcecode zu assemblieren

```
*** CASL ***
Assemble Monitor Go
complete !
```

Es wird „assembling“ in der unteren Zeile des Bildschirms angezeigt. Wurde der Assembler-Lauf erfolgreich ausgeführt erscheint die Meldung „complete !“. Tritt beim Assemblieren ein Fehler auf wird der Vorgang beendet und eine Fehlermeldung ausgegeben.

Das fertige Object-Programm wird ab Adresse 1000 gespeichert.

Das CASL-Assembler-Protokoll

Während des Assemblerlaufs wird ein Protokoll erzeugt mit folgendem beispielhaften Format:

```
ADD : OBJECT   : LINE NO.
               : 10
1000:7000 100B: 20
1002:7000 100A: 20
1004:8000 0002: 20
1006:1244 0002: 20
1008:6400 0004: 30
100A:0009      : 40
100B:00E8      : 50
100C:00E9      : 60
100D:0043      : 70
100E:0041      : 70
100F:0052      : 70
1010:0044      : 70
1011:0053      : 70
1012:00EA      : 80
1013:00EB      : 90
               : 100
```

```
←①→←——②——→←——③——→
LABEL :ADDRESS
L1      1000
L2      1000
N       100A
DSP     100B
```

①Adresse (16bit)
②Objekt (16bit)
③Zeilennummer des Source-Progr.
④Label

Hinweis:

Die Ausgabe des Protokolls kann auch auf einen angeschlossen Drucker CE-126P umgeleitet werden. (z.B. durch SHIFT + P↔NP)

9.6 CASL-Assembler-Fehlermeldungen

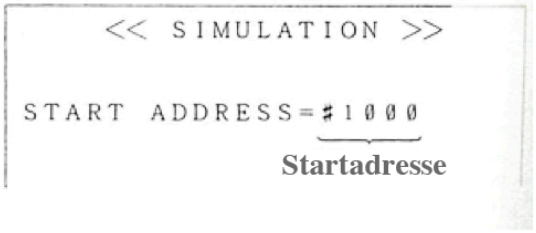
Wenn der Assembler während der Ausführung denen Fehler erkennt werden diese Fehlermeldungen ausgegeben. Durch drücken von CLS wird die Fehlermedlung auf dem Bildschirm gelöscht. Danach können Sie das Programm mit dem Text-Editor korrigieren.

Art des Fehlers	Fehlermeldung	Ursache
Operationscode-Error	OP-CODE ERROR (zeilennummer)	Der Befehlscode in der angegeben Zeile ist falsch.
	OP-CODE ERROR (0)	Kein Quellprogramm
Operanden-Fehler	OPERAND ERROR (zeilennummer)	Es wurde ein Fehler bei einem Operanden des Befehls an der angegeben Zeilennummer gefunden.
Label-Fehler	LABEL ERROR (zeilennummer)	Es ist ein Label-Fehler bei der angegebenen Zeilennummer erkannt worden
Speicherfehler	MEMORY ERROR (0)	- Speicher zum Speichern des Objekts fehlt. - Der Stack-Speicherplatz reicht nicht aus.
Allgemeiner Fehler	OTHER ERROR	Es wurde kein START- bzw. END-Befehl gefunden oder das Quellprogramm hat einen sonstigen Strukturfehler

9.7 Simulation

Drücken Sie die Taste G im CASL-Menü um das Programm auszuführen.

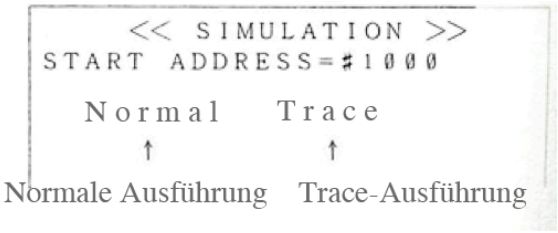
G



Es wird die Startadresse angezeigt. Diese kann geändert werden. Wenn nichts weiter eingegeben wird und ↵ gedrückt wird, wird das Programm die Startadresse #1000 verwenden.

Sie könne die gewünschte Adresse in dezimaler oder in hexadezimaler Form (mit vorangestelltem #-Zeichen) eingeben. Danach drücken sie ↵.

Danach erscheint folgende Anzeige:

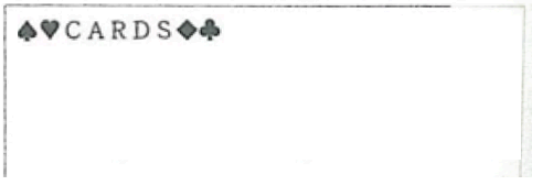


Drücken Sie N oder T.

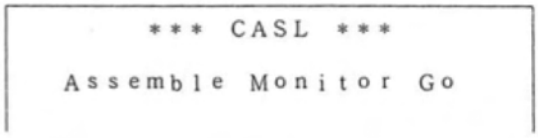
Normale Ausführung

Drücken sie N um das Programm zu starten.

N



↵



Laufende Programme können jederzeit mit der BREAK-Taste abgebrochen werden um z.B. Register oder Speicherinhalte zu prüfen oder zu ändern. Durch G im CASL-Menu (GO) kann das Programm danach wieder weitergeführt werden.

Trace-Modus

Drücken Sie dazu die Taste T. Es werden die Register (GR0-GR4), Der Programmzähler (PC), das Flag-Register (FR) und der aktuelle Befehl angezeigt.

Mit jedem Drücken der ↵-Taste wird der nächste Befehl ausgeführt.

```
1000: GR0:0000 GR4:1B0B
      GR1:0000 PC  :1002
      GR2:0000 FR  :0000
      GR3:0000 <PUSH>
```

Hinweis: Der Inhalt von GR4 kann je nach Speicher anders sein

↵

```
1002: GR0:0000 GR4:1B0A
      GR1:0000 PC  :1004
      GR2:0000 FR  :0000
      GR3:0000 <PUSH>
```

Der ausgeführte Befehl

Wird die Ausgabe durch SHIFT + P<->NP umgeleitet wird die Trace-Ausgabe wie folgt auf dem Drucker ausgegeben:

```
ADD :GR0  GR1  GR2  GR3
1000:0000 0000 0000 0000
1002:0000 0000 0000 0000
1004:0000 0000 0000 0000
♠♥CARDS♦♣
0002:0000 0000 0000 0000
1006:0000 0000 0000 0000
1008:0000 0000 0000 0000
←①→|←②→|←③→|←④→|←⑤→|
```

①Adresse
②GR0
③GR1
④GR2
⑤GR3

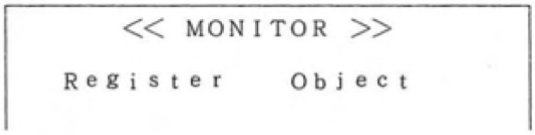
Trace-Fehlermeldungen

Fehlermeldung	Ursache
OBJECT ERROR	Keine Objektprogramm gefunden
* MEM * * ERR *	JMP adressiert einen Bereich außerhalb des Adressraumes Der verfügbare Speicher wurde überschritten
* OPR * * ERR *	Die Ausgabe von OUT hat mehr als 97 Zeichen

9.8 Monitor-Funktion

Mit der Monitorfunktion überprüfen Sie die Inhalte der Register der virtuellen COMET-Maschine. Genauso können Sie das Objekt-Programm oder die Register verändern.
Sie können darüber hinaus mit Breakpoints arbeiten.
Die Monitor-Funktion können Sie durch das Drücken der Taste M im CASL-Hauptmenu erreichen.

Ⓜ



Jetzt können Sie folgendes wählen:

Taste	Beschreibung
R	Zum Anzeigen und Ändern der Register
O	Hier können Sie das Object-Programm und den Speicher anzeigen und ändern.

Anzeige der Registerinhalte

Drücken Sie die Taste R um die Registerinhalte anzuzeigen.

R

```
GR0  #0000  0
GR1 :#0000  0
GR2 :#0000  0
GR3 :#0000  0
GR4 :#1B0B  6923
PC  :#1000  4096
```

Register Inhalt dezimaler Inhalt

Mit den Cursortasten ▼▲ oder ENTER kann auf der Anzeige hin und her geblättert werden. Beim gerade aktuellen Register verschwindet der Doppelpunkt.

:
▼

```
GR3 :#0000  0
GR4 :#1B0B  6923
PC  :#1000  4096
FR  :#0000  0
BP  :#FFFF  65535
BC  :#0000  0
```

Register	Name	Beschreibung
GR0-GR4	Allgemeine Register	Universal-Register. GR4 wird als Stack-Pointer verwendet
PC	Programmzähler	Zeigt auf den nächsten auszuführenden Befehl
FR	Flag-Register	Ergebnis der Ausführung eines Befehls (positiv, null, negativ)
BP	Break-Zeiger	Wird verwendet um die Ausführung der Simulation zu steuern.
BC	Break-Zähler	

Setzen der Register

Der Inhalt des ausgewählten Registers (dort wo der Doppelpunkt ausgeschaltet ist) kann wie folgt mit Werten gesetzt werden:

Dezimal: Eingabe einer dezimalen Zahl von -32768-65535
Beispiel: 123 ↵ Ausgabe: #007B 123

Hexadezimal: Eingabe einer hexadezimalen Zahl von 0-FFFF
Beispiel: #007B ↵ Ausgabe: #007B 123

Eingabe eine Labels: Label eingeschlossen in Doppelhochkomma
Beispiel: " L1" ↵ Ausgabe: #100A 4106 (L1 zeigt auf #100A)

Eingabe eines Zeichens: Zeichen eingeschlossen in Hochkomma
Beispiel 'A'↵ Ausgabe: #0041 65

Das Drücken von CLS bricht die Eingabe ab ohne den ursprünglichen Wert zu ändern.

Hinweis: Beim Inhalt des Register FR werden nur 2 Bits verwendet (Werte 0,1 und 2) alle anderen Bits werden ignoriert.

Inhalte der Register nach Reset/Start des Assemblers:

GR0-GR3: 0

GR4 : obere Adresse + 1 des Objektbereichs

PC : Startadresse des Programms (Adresse des Labels der Startanweisung)

FR : 0

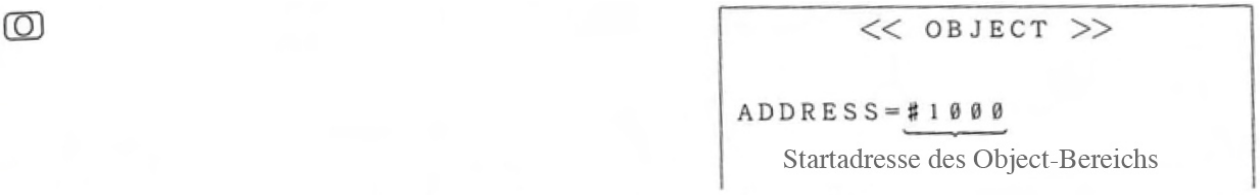
BP : FFFF (Hex) 65535 (dezimal), Kein Haltepunkt gesetzt

BC : 0

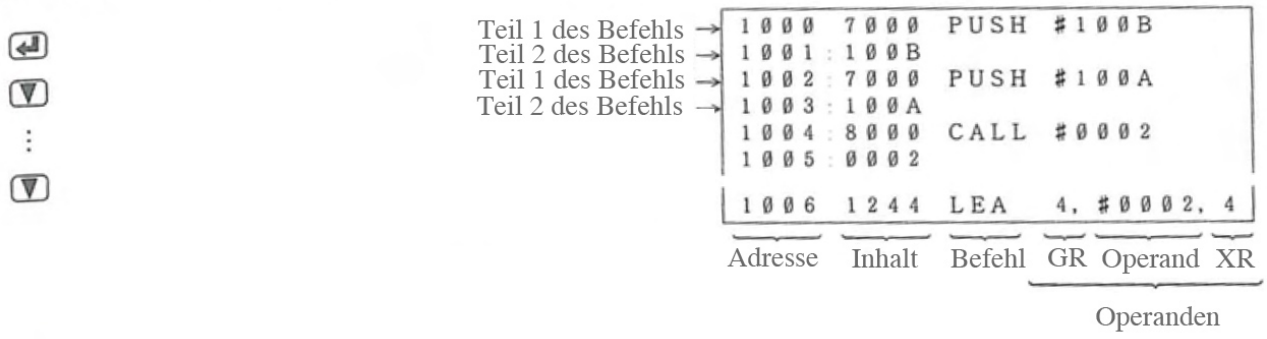
Hinweis: GR4 kann im Normalfall frei verwendet werden. Die Adresse des Objektbereichs kann im Monitor oder im Programm verändert werden.

Anzeige des Object-Codes

Drücken Sie die Taste O im CASL-Monitor-Menü um die Registerinhalte anzuzeigen.



Mit Enter die Startadresse bestätigen oder diese Adresse vorher ändern.



Hinweis: Wurde vorher kein Programm über den CASL-Assembler geladen wird der Fehler „OBJECT ERROR“ ausgegeben.

Mit den Cursortasten ▼▲ oder ENTER kann auf der Anzeige hin und her geblättert werden. Bei der gerade aktuellen Adresse verschwindet der Doppelpunkt.

Der Inhalt der ausgewählten Adresse (dort wo der Doppelpunkt ausgeschaltet ist) kann wie folgt mit Werten gesetzt werden:

Dezimal: Eingabe einer dezimalen Zahl von -32768-65535
Beispiel: 123 ↵ Ausgabe: #007B 123

Hexadezimal: Eingabe einer hexadezimalen Zahl von 0-FFFF
Beispiel: #007B ↵ Ausgabe: #007B 123

Eingabe eine Labels: Label eingeschlossen in Doppelhochkomma
Beispiel: “ L1“ ↵ Ausgabe: #100A 4106 (L1 zeigt auf #100A)

Eingabe eines Zeichens: Zeichen eingeschlossen in Hochkomma
Beispiel ’A’↵ Ausgabe: #0041 65

Das Drücken von CLS bricht die Eingabe ab ohne den ursprünglichen Wert zu ändern.

CASL-Beispielprogramm

Im Folgenden ein Beispielprogramm, das 5 Zahlen addiert:

Dieses Programm addiert die Zahlen in Zeile 130-170 in DAT (Zeile 120) ab.

```
10EXAM    START
20BGN     LEA     GR0,0          In das Register 0 den Wert 0 schreiben
30        LEA     GR1,0          In das Register 1 den Wert 0 schreiben
40        JMP     AGN1           unbedingter Sprung nach AGN1
50AGN     ADD     GR0,DAT,GR1    Addiert zu Register 0 den Inhalt von
                                 DAT (mit Verschiebung GR1)
60        LEA     GR1,1,GR1      Register 1 um den Wert 1 erhöhen
70AGN1    CPA     GR1,N          Vergleicht die Zahlen in GR1 und N
80        JMI     AGN            Ist das Ergebnis negativ
                                 (also N größer GR1) Sprung zu AGN
90        ST      GR0,TTL        Speichert die Zahl Im Register 0 nach TTL
100       EXIT                   Programmende/Rücksprung
110N      DC      5
120TTL    DS      1              Definiert einen Speicherplatz
130DAT    DC      #000C
140       DC      #07F3
150       DC      #0231
160       DC      #0009
170       DC      #000F
180       END                    Ende des Programmcodes
```

Beispiel für die Bedienung:

Eingabe	Ausgabe
	*** CASL *** Assemble Monitor Go
[M]	<< MONITOR >> Register Object
[O]	<< OBJECT >> ADDRESS=#1000
[↵]	1000 1200 LEA 0, #0000 1001:0000 1002:1210 LEA 1, #0000 1003:0000 1004:6400 JMP #100A 1005:100A ...
[▼] ⋮	1006:2001 ADD 0, #1014, 1 1007:1014 1008:1211 LEA 1, #0001, 1 1009:0001 100A:4010 CPA 1, #1012 100B:1012 100C:6100 JMI #1006 100D:1006 100E:1100 ST 0, #1013 100F:1013 1010:6400 JMP #0004 1011:0004 1012:0005 * 1013:0000 1014:000C * 1015:07F3 1016:0231 ***** 1017:0009 1018:000F * 1019:0000
BREAK [ON]	<< MONITOR >> Register Object

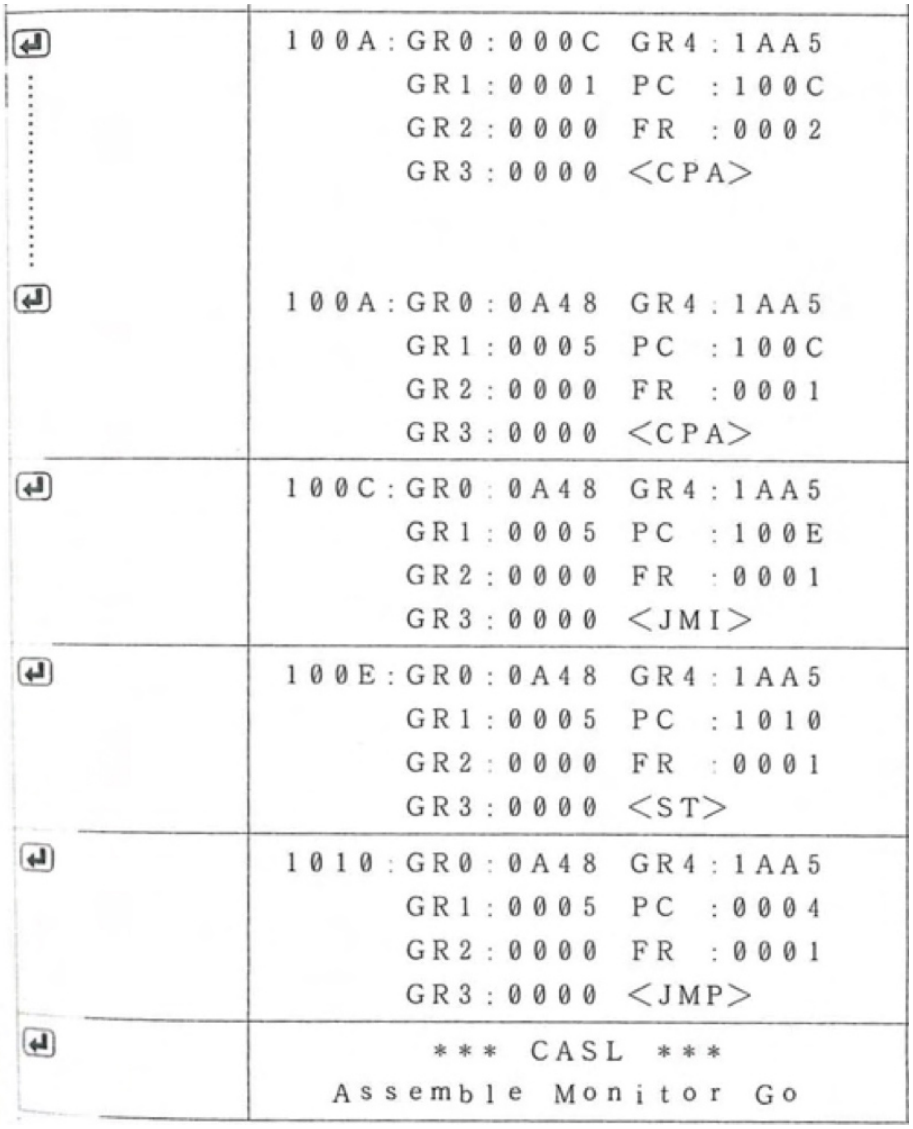
Eingabe	Ausgabe
	*** CASL *** Assemble Monitor Go
[M]	<< MONITOR >> Register Object
[R]	GR0 #0000 0 GR1:#0000 0 GR2:#0000 0 GR3:#0000 0 GR4:#1AA5 6821 PC :#1000 4096
[▼]	
:	
[▼]	BP #FFFF 65535
#100C	BP #FFFF #100C_
[↵]	BP #100C 4108

[▼]	BC #0000 0
2 [↵]	BC #0002 2
[BREAK] [BREAK]	*** CASL *** Assemble Monitor Go
[G] [↵]	<< SIMULATION >> START ADDRESS=#1000 Normal Trace
[N]	100C: GR0:000C GR4:1AA5 * * GR1:0001 PC :100C *STP* GR2:0000 FR :0002 * * GR3:0000 <JMI>
[↵]	100C: GR0:07FF GR4:1AA5 * * GR1:0002 PC :100C *STP* GR2:0000 FR :0002 * * GR3:0000 <JMI>

BREAK BREAK	*** CASL *** Assemble Monitor Go
M R	GR0 #07FF 2047 GR1:#0002 2 GR2:#0000 0 GR3:#0000 0 GR4:#1AA5 6821 PC :#100C 4108
▼ ⋮ ▼	BC #0000 0
4 ↵	BC #0004 4
BREAK BREAK	*** CASL *** Assemble Monitor Go
G ↵	<< SIMULATION >> START ADDRESS=#100C Normal Trace
N	100C: GR0:0A48 GR4:1AA5 * * GR1:0005 PC :100C *STP* GR2:0000 FR :0001 * * GR3:0000 <JMI>
BREAK BREAK	*** CASL *** Assemble Monitor Go

Trace:

Eingabe	Ausgabe	
	*** CASL *** Assemble Monitor Go	
G ↵	<< SIMULATION >> START ADDRESS=#1000 Normal Trace	
T	1000:GR0:0000 GR4:1AA5 GR1:0000 PC :1002 GR2:0000 FR :0001 GR3:0000 <LEA>	LEA GR0,0
↵	1002:GR0:0000 GR4:1AA5 GR1:0000 PC :1004 GR2:0000 FR :0001 GR3:0000 <LEA>	LEA GR1,0
↵	1004:GR0:0000 GR4:1AA5 GR1:0000 PC :100A GR2:0000 FR :0001 GR3:0000 <JMP>	JMP AGN1
↵	100A:GR0:0000 GR4:1AA5 GR1:0000 PC :100C GR2:0000 FR :0002 GR3:0000 <CPA>	CPA GR1,N
↵	100C:GR0:0000 GR4:1AA5 GR1:0000 PC :1006 GR2:0000 FR :0002 GR3:0000 <JMI>	JMI AGN
↵	1006:GR0:000C GR4:1AA5 GR1:0000 PC :1008 GR2:0000 FR :0000 GR3:0000 <ADD>	ADD GR0,DAT,GR1
↵	1008:GR0:000C GR4:1AA5 GR1:0001 PC :100A GR2:0000 FR :0000 GR3:0000 <LEA>	LEA GR1,1,GR1



CPA GR1, N

JMI AGN

ST GR0,TTL

EXIT

9.9 COMET-Spezifikation

Ausgehend von der COMET/CASL Spezifikation hat das japanische Ministerium für Wirtschaft, Handel und Industrie im Jahre 2001 folgende Spezifikation von COMET II und CASL II entworfen:

(1) START
Definiert den Beginn eines Programms. Standardmäßig ist das die Adresse #1000 .

(2) DC
Definiert einen Speicherbereich mit einem dezimalen Wert von -32768-65535 (hex #0000-#FFFF) oder eine Zeichenkette

(3) IN (CALL #0000)
Eingabe von Zeichen vom Bildschirm. Als Eingabeanforderung wird ein Fragezeichen (‚?') ausgegeben. Die Eingabe wird durch ENTER abgeschlossen. Der erste Operand ist die Adresse an der die Eingabe hingeschrieben werden soll. Die Anzahl der gelesenen Zeichen wird an die Adresse des zweiten Opernanden geschrieben. Durch Eingabe von ▼ wird die Eingabe übergangen, dann wir als Anzahl 65636 (#FFFF) übergeben.

(4)OUT (CALL #0002)
Ausgabe einer Zeichenkette. Dieser Befehl entspricht dem Basic-Befehl PRINT. Der erste Operand ist die Adresse an der die auszugebenden Zeichen stehen. Im zweiten Operand wird die Adresse angegeben, in der die Anzahl der auszugebenden Zeichen abgelegt ist.

(5)WRITE CALL #0006)
Gibt die Inhalte der Register auf dem Bildschirm aus. Durch ENTER wird das Programm fortgesetzt.

(6)END
Legt das Ende der Programms fest.

9.10COMET-Architektur

Für ein besseres Verständnis des CASL-Assemblers ist es notwendig die technischen Daten der COMET-Architektur zu kennen.

Wortlänge:	16 Bit (Jede Speicheradresse hat eine Länge von 16Bit, im Gegensatz zu der Länge von 8Bit eines normales Computers)
Architektur:	von Neumann
Zahlen:	16-Bit-Binärzahlen, negative Zahlen werden durch das Zweierkompliment dargestellt.
Register:	GR0-GR4 (16bit) General Register (Allzweckregister) GR1-GR4 werden auch als Index-Register verwendet. Das Register GR4 wird aber hauptsächlich als Stapelzeiger (SP) verwendet. Der Stapelspeicher beginnt an der obersten freien Adresse der Comet-Maschine und wächst mit jedem neuen Eintrag nach unten. PC (16bit) Program Counter (Programmzähler) Dieses Register enthält die Adresse des als nächstes auszuführenden Befehls. FR(2bit) Flag Register Dieses Register enthält das Ergebnis aus Vergleichsoperationen 00=größer(positiv), 01=null(geich),10=kleiner(negativ)
Stapelspeicher:	Der Stapelspeicher beginnt an der obersten freien Adresse der Comet-Maschine und wächst mit jedem neuen Eintrag nach unten. Das Register GR4 zeigt auf den aktuellsten abgelegten Wert. Ist kein Wert abgelegt zeigt GR4 auf die letzte Adresse+1. Wir ein Wert mit PUSH eingetragen verringert sich die Adresse in GR4 um 1. POP hingegen erhöht die Adresse wieder um 1.

9.11Befehlsübersicht

Die Befehle haben eine Länge von 2 16Bit-Worten

Wort 1			Wort 2			
OP	GR	XR	adr	Syntax		Beschreibung
				Befehl	Operanden	
0 0						
1 0				LD	GR, adr, XR	load
1 1				ST	GR, adr, XR	store
1 2				LEA	GR, adr, XR	load effective address
2 0				ADD	GR, adr, XR	add arithmetic
2 1				SUB	GR, adr, XR	subtract arithmetic
3 0				AND	GR, adr, XR	and
3 1				OR	GR, adr, XR	or
3 2				EOR	GR, adr, XR	exclusive or
4 0				CPA	GR, adr, XR	compare arithmetic
4 1				CPL	GR, adr, XR	compare logical
5 0				SLA	GR, adr, XR	shift left arithmetic
5 1				SRA	GR, adr, XR	shift right arithmetic
5 2				SLL	GR, adr, XR	shift left logical
5 3				SRL	GR, adr, XR	shift right logical
6 0	0			JPZ	adr, XR	jump on plus or zero
6 1	0			JMI	adr, XR	jump on minus
6 2	0			JNZ	adr, XR	jump on non zero
6 3	0			JZE	adr, XR	jump on zero
6 4	0			JMP	adr, XR	unconditional jump
7 0	0			PUSH	adr, XR	push effective address
7 1		0	0 0 0 0	POP	GR	pop up
8 0	0			CALL	adr, XR	call subroutine
8 1	0	0	0 0 0 0	RET		return from subroutine
9 ~ F				nicht verwendet		

Typen und Funktionen der Befehlsoperanden

In CASL sind in für diesen Pocket-Computer 23 Befehle definiert. Im nächsten Abschnitt werden die Befehle und die Operanden erklärt:

GR : GR0-4 sind die Alzweckregister
XR : XR0-4 entsprechen der optionalen Nutzung von Index-Registern. (Es gibt keine speziellen XR-Register, diese entsprechen den GR-Registern)
SP : Ist der Stackpointer. Dieser wird durch das Register GR4 abgebildet.
adr : Ist eine 16bitige Zahl die einem Label entspricht oder eine zu verarbeitende Zahl. Der Wert geht dezimal von -32758-65535 oder hexadezimal von #0000-#FFFF.
gültige Adresse: Ist eine Adresse die sich aus dem Adresswert adr und dem Index XR ergibt.
[ ] : Optionaler Parameter

Befehle

LD GR,adr [,XR]
Der Inhalt der Adresse wird in das angegebene Register GR0-GR4 gespeichert.

ST GR,adr [,XR]
Der Inhalt des Registers wird in die angegebene Speicheradresse geschrieben.

LEA GR,adr [,XR]
Die Wert von adr wird in das Register geschrieben.

Beispiele:

LEA GR1,100	Lädt den Wert 100 in das Register GR1
LEA GR1,10,GR1	Erhöht den Inhalt des Registers GR1 um 10
LEA GR1,0,GR2	Der Inhalt von GR2 wird nach GR1 kopiert

ADD GR,adr [,XR]
Das Register GR wird um den Wert auf der Adresse adr addiert.

SUB GR,adr [,XR]
Das Register GR wird um den Wert auf der Adresse adr subtrahiert.

AND GR,adr [,XR]
OR GR,adr [,XR]
EOR GR,adr [,XR]
Der Inhalt auf Adresse adr wird bitweise (16Bit) logisch mit and, or oder exclusive or

verknüpft.

CPA GR,adr [,XR]
Der Inhalt auf Adresse adr wird verglichen mit dem Inhalt des Registers GR.
CPA vergleicht arithmetisch und interpretiert den Werte als Zahlen (-32768-32767)
CPL vergleicht logisch und interpretiert den Inhalt bitweise

(GR) > Wert FR=00 (0)
(GR) = Wert FR=01 (1)
(GR) < Wert FR=10 (2)

JPZ adr [,XR] Sprung zur Adresse wenn Vergleich positiv oder Null (FR=00 oder 01)
JMI adr [,XR] Sprung zur Adresse wenn Vergleich negativ (FR=10 (2))
JNZ adr [,XR] Sprung zur Adresse wenn Vergleich nicht Null (FR=00 oder 10)
JZE adr [,XR] Sprung zur Adresse wenn Vergleich Null (FR=01 (1))
Verzweigt das Programm zur angegeben Adresse wenn die Bedingung erfüllt ist.

JMP adr [,XR]
Verzweigt das Programm zur angegeben Adresse

SLA GR,adr [,XR]
SRA GR,adr [,XR]
Arithmetische bitweise Verschiebung
Der Inhalt des Registers wird bitweise nach links bzw rechts um die Anzahl bits die mit adr (plus dem optionalen Inhalt von XR) angegeben sind verschoben. Dabei bleibt das Vorzeichen (Bit 15) immer erhalten. Bei negativen Zahlen wird bei der Rechtsverschiebung eine 1 statt einer 0 eingeschoben. Das FR-Register wird je nach Ergebnis gesetzt.

SLL GR,adr [,XR]
SRL GR,adr [,XR]
Arithmetische bitweise Verschiebung
Der Inhalt des Registers wird bitweise nach links bzw rechts um die Anzahl bits die mit adr adr (plus dem optionalen Inhalt von XR) angegeben sind verschoben.

PUSH adr [,XR]
Schreibt den Inhalt der Adresse in den Stack. Die Stapeladresse (SP) im Register GR4 wird auf diese neue TOP-Stapeladresse gesetzt.

POP GR
Schreibt den Inhalt aus der TOP-Adresse des Stacks in das angegebene Register. Die Die TOP-Stapeladresse (SP) im Register GR4 wird auf die vorige Stapeladresse gesetzt.

CALL adr [,XR]
Dieser Befehl ruft ein Unterprogramm an der angegeben Adresse auf. Die Rücksprungadresse (Adresse nach CALL) wird auf den Stack gelegt.

RET
Springt aus einem Unterprogramm zurück in das aufrufende Programm (CALL). Dazu wird die Rücksprungadresse vom Stack genommen.

Assembler-Syntax

Label	Befehl	Operand	Erklärung
[label]	START	[Startlabel]	Kennzeichnet Beginn des Porgramms
	END		Ende des Programms
[label]	DC	Konstante	Definiert Zahlen oder Zeichenketten
[label]	DS	Anzahl Worte	Definiert einen Speicherbereich
[label]	IN	Zeichenkette,Länge	Liest Zeichen vom Bildschirm
[label]	OUT	Zeichenkette,Länge	Schreibt Zeichen auch den Bildschirm
[label]	EXIT		Programmrücksprung
[label]	WRITE		Ausgabe der Register auf dem Bildschirm

START [label]
Mit der START-Anweisung beginnt ein CASL-Programm. Optional kann ein Label angegeben werden an dem die Programmausführung begonnen werden soll. Ansonsten wird der der START-Anweisung folgende Befehl ausgeführt.

END
Mit der END-Anweisung wird das Ende des CASL-Programms festgelegt.

DC n
Definiert eine Zahlen-Konstante(dezimal). Der Wert der Konstante muss zwischen -32768-65535 liegen.

DC #h
Definiert eine Zahlen-Konstante(hexadezimal). Der Wert der Konstante muss zwischen #0000-#FFFF liegen.

DC 'string'
Definiert eine Zeichenkette. Jedes Byte wird in der rechten Hälfte einer Adresse (16Bit) abgelegt. Von der Zeichencodetabelle können die Zeichen 32-38 (&H20-&H26), 40-95 (&H28-&H5F), 97-122 (&H61-&H7A), 166-223 (&HA6-&HDF) genutzt werden. Es wird innerhalb der Zeichenkette keine Länge abgelegt. Das Programm muss also selber wissen wie lang die Zeichenkette ist.

DC label
Definiert eine Konstante die die Adresse des angegeben Labels enthält.

DS [n]
Definiert einen Speicherbereich der n Worte enthält. Wird als Anzahl 0 angegeben, wird nur das Label für die nächstfolgende Adresse definiert.

Besonderer Hinweis:
Wird während der Programmausführung auf #0000 getroffen wird die Ausführung des Programms unterbrochen. In der Register-Anzeige erscheint *STP*. Dies kann dazu genutzt werden, um das Programm an einer Stelle zu stoppen um Register und Speicher zu prüfen oder zu ändern. Der Befehlscounter wird um 2 erhöht so dass beim nächsten Start das Programmmit dem nächsten Befehl fortgesetzt werden kann. Dabei ist jedoch zu beachten, dass im Programm dazu zwei Worte mit #0000 (z.B. mit 2x 'DC 0') definiert sein müssen, da der Programmzähler immer um zwei erhöht wird.

Makro-Befehle

IN adr,länge
Eingabe von Zeichen vom Bildschirm. Als Eingabeanforderung wird ein Fragezeichen (‚?') ausgegeben. Die Eingabe wird durch ENTER abgeschlossen. Der erste Operand ist die Adresse an der die Eingabe hingeschrieben werden soll. Die Anzahl der gelesenen Zeichen schreibt die Funktion an die Adresse des zweiten Opernanden. Durch Eingabe von ▼ wird die Eingabe übergangen, dann wir als Anzahl 65636 (#FFFF) übergeben.

OUT adr,länge
Ausgabe einer Zeichenkette. Dieser Befehl entspricht dem Basic-Befehl PRINT. Der erste Operand ist die Adresse an der die auszugebenden Zeichen stehen. Im zweiten Operand wird die Adresse angegeben, in der die Anzahl der auszugebenden Zeichen abgelegt werden muss. Das Programm wird nach der Ausgabe unterbrochen und muss durch drücken von ENTER weitergeführt werden.

EXIT
Beendet die Ausführung des Programms.

WRITE
Gibt die Inhalte der Register auf dem Bildschirm aus. Durch ENTER wird das Programm fortgesetzt.

Beispiel:

```
10          START
20          IN          A,C          Einlesen der Zeichen in A
25          OUT         NL,N         Ausgabe von 9xP als Trennung
30          OUT         A,B          Ausgabe der ersten 2 Zeichen von A
40          EXIT                     Ende des Programms
```

```
50A     DS      20          Eingabepuffer mit 20 Worten(Zeichen)
60B     DC      2           Ausgabelänge 2
70C     DS      1           Ablage der Anzahl der gelesenen Zeichen
80N     DC      9           Ausgabelänge für 9x'P'
90NL    DC      'PPPPPPPPP  Zeichenkette mit 9x'P'
        '
100     END                 Ende des Quellprogramms
```

Ablage des Beispiel-Programms im Speicher:

```
IN      A,C     7000  101A    PUSH    A
                7000  102F    PUSH    C
                8000  0000    CALL    #0000
                1244  0002    LEA     GR4,2,GR4
OUT     NL,N    7000  1031    PUSH    NL
                7000  1030    PUSH    N
                8000  0002    CALL    #0002
                1244  0002    LEA     GR4,2,GR4
OUT     A,B     7000  101A    PUSH    A
                7000  102E    PUSH    B
                8000  0002    CALL    #0002
                1244  0002    LEA     GR4,2,GR4
EXIT            6400  0004    JMP     #0004

A  DS  20       101A..102D    #0000 (20x)
B  DC  2        102E          #0002
C  DS  1        102F          #0000
N  DC  9        1030          #0009
NL DC 'PPPP...  1031..1039    #0050  (9x)
```

10 Maschinensprache-Monitor

Mit diesem Computer können Sie Programme sowohl in Maschinensprache als auch in BASIC schreiben. Um Ihnen bei der Programmierung im Maschinencode zu helfen hat der Computer einen Maschinensprachen-Monitor (im folgenden als "der Monitor" bezeichnet). Dieser Monitor ist eines der besonderen Merkmale dieses Systems, mit dem Sie eine bestimmte Folge von Befehlen eingeben bzw. ausgeben oder ein in Maschinencode geschriebenes Programm ausführen können. Dieser Abschnitt beschreibt die Funktionen der Befehle dieses Maschinensprachen-Monitors für diesen Computer.
Die CPU des Computers ist ein Z80-Mikroprozessor (entsprechend CMOS Z80A), der häufig in den besonders leistungsfähigen 8-Bit-Computern verwendet wird. Im Handel sind zahlreiche Bücher über den Prozessor Z80 vorhanden aus denen Sie Hinweise zur Maschinensprache des Z80 und andere wichtige Informationen entnehmen können. Dieses Kapitel beschreibt das Verhalten der Befehle im Maschinensprache-Monitor wie man ein Source-Programm erstellt und wie man es ausführt.

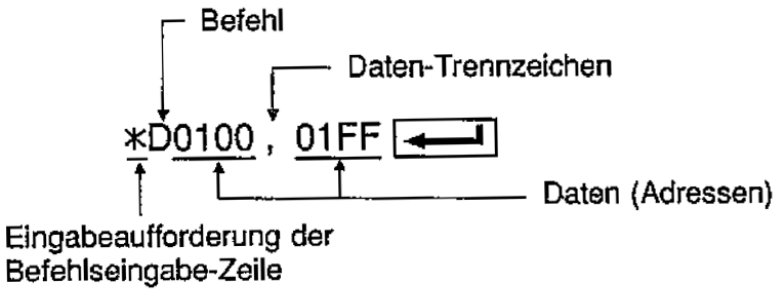
10.1 Verwendung des Maschinensprachen-Monitors

Die Monitor-Betriebsart wird durch Eingabe von MON in der BASIC-Betriebsart (RUN oder PRO) gewählt. Die folgende Anzeige erscheint.

```
MACHINE LANGUAGE MONITOR
*
```

Das Sternchen (*) auf der Anzeige bedeutet in der Monitor-Betriebsart die Eingabeaufforderung der Befehlseingabezeile. Hier werden alle Befehle eingegeben. Alle notwendigen Adressen oder weitere Daten können nach dem Befehl hier eingegeben werden. Am Ende jeder Zeile muss die Eingabe durch das Drücken von ↵ zur Ausführung gebracht werden.

Beispiel:



Hinweis:

- Bei aktivierter Speicherschutzfunktion mit Passwort kann der Computer nicht in die Monitor-Betriebsart eingestellt werden.
- Alle Adressen und Daten müssen im hexadezimalen Code sein.
- Um mehr als eine Adresse oder um Datenteile voneinander zu trennen, wird ein Komma (,) verwendet.
- Wenn kein hexadezimaler Code verwendet wird oder ein anderes Symbol außer dem Komma zur Datentrennung eingegeben wird, erfolgt ein Fehler (SYNTAX ERROR).
- Die Monitor-Betriebsart kann durch Wahl einer anderen Betriebsart oder durch Aus- und wieder Einschalten des Computers verlassen werden.
- Da die Maschinensprache sehr kompliziert ist, kommt es häufig zu Programmfehlern. Bei der Ausführung eines Programms in Maschinensprache kann es vorkommen, dass BASIC-Programme, Daten oder andere Teile des Computerspeichers zerstört werden. Aus diesem Grund wird empfohlen, alle BASIC-Programme, Daten oder andere Informationen auf einem PC zu sichern bevor ein Programm in Maschinensprache ausgeführt wird.
- Bei der Verwendung des Monitors kann der Zugang zu einem anderen als dem Maschinensprachen-Bereich (der mit dem USER-Befehl zugewiesen wurde) zur Zerstörung von BASIC- oder TEXT-Programmen führen oder andere Daten in diesem Bereich zerstören oder zu Fehlfunktionen führen. Stellen Sie sicher, dass Sie nur den für Maschinencode vorgesehenen Bereich verwenden.

10.2Auflistung der Befehle für den Maschinensprachen-Monitor

USER - Benutzerbereich

Wirkung: Zuweisung eines Bereiches für Maschinensprache und Anzeige der Adressen dieses Bereiches.

Format: (1) USER01FF↵
(2) USER↵
(3) USER00FF↵

Anmerkungen: • Mit Format (1) wird der Speicher-Adressbereich von 0100H (der ersten Adresse) bis 01FF (der letzten Adresse) als Bereich für den Maschinencode zugewiesen. Die erste Adresse ist automatisch auf 0100H eingestellt.

Zugewiesener Bereich →

```
*USER01FF
FREE:0100-01FF
*
```

- Mit Format (2) wird der Bereich der zugewiesenen Adressen für den Maschinencode-Bereich angezeigt.

```
*USER
FREE:0100-01FF
*
```

- Wenn kein Bereich für den Maschinencode zugewiesen wurde, wird "FREE: NOT RESERVED" angezeigt.
- Mit Format (3) wird ein bereits vorhandener Maschinencode-Bereich aus dem Speicher gelöscht und die Meldung "FREE: NOT RESERVED" angezeigt.
- Bei Eingabe eines unzulässigen Adressbereiches für den Maschinencode-Bereich wird eine Fehlermeldung (MEMORY ERROR) angezeigt

S - Aktualisierung des Speichers

Wirkung: Aktualisierung des Speicherbereichs.
Format: (1) S0100↵
(2) S↵
Anmerkungen:
- Mit Format (1) wird der Inhalt von Adresse 0100H (der ersten Adresse) angezeigt und zur Eingabe einer neuen Einstellung aufgefordert.

```
*S0100
0100:10-
```

Vorhandener Speicherinhalt

- Um die Daten zu ändern, wird ein Byte eingegeben (zweistellig hexadezimal) und dann ↵ gedrückt. Der Computer gibt nun den Inhalt der darauf folgenden Adresse an und fordert wieder zur Eingabe von Daten auf.
- Wenn vorhandene Daten nicht geändert werden sollen, drückt man ↵ ohne Daten einzugeben. Der Computer zeigt dann den Inhalt der folgenden Adresse an und fordert wieder zur Eingabe von Daten auf.
- Es können maximal zwei Stellen hexadezimaler Daten eingegeben werden. Um eine Eingabe vor dem Drücken von ↵ zu löschen wird ◄ oder CLS gedrückt
- Zum Abrufen des Inhalts der vorherigen Adresse ▲ und zum Abrufen des Inhalts der folgenden Adresse ▼ drücken.
- Mit Format (2) wird der Inhalt derjenigen Adresse angezeigt, die direkt neben der letzten mit dem S-Befehl dargestellten Adresse liegt.
- Zum Zurückgehen auf die Befehlseingabe-Zeile wird BREAK gedrückt.

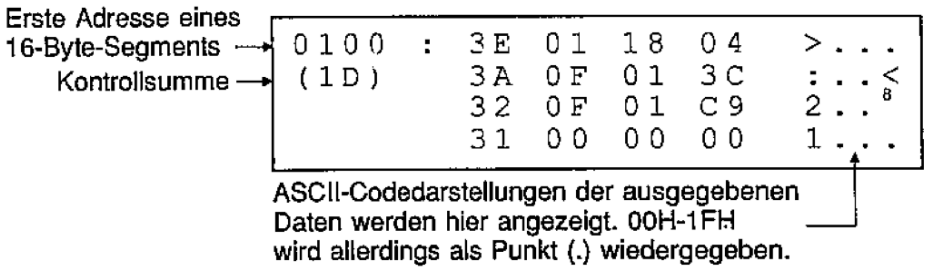
D – Speicherauszug ausgeben

Wirkung: Ausgabe des Speicherbereichs.

Format: (1) D0100↵
(2) D↵
(3) D0100,01FF↵

Anmerkungen: • Mit Format (1) werden 16 Byte aus dem Adressbereich von 0100H (erste Adresse) bis 010FH ausgegeben. (Die ausgegebenen Daten werden in der Drucker-Betriebsart ausgedruckt.)

Beispiel:



Hinweise:

- Die Größe des ausgebbaren Bereiches ist auf XXX0H-XXXFH festgelegt. Der Inhalt des gleichen Segments wird immer dann ausgegeben, wenn eine Adresse innerhalb eines 16-Byte-Segments angegeben wird. Wenn Sie zum Beispiel die Adresse 0104H angeben wird der Inhalt der festgelegten 16 Segmente dieser Adresse ausgegeben, in diesem Fall 0100H-010FH.
- Zur Ausgabe des vorhergehenden 16-Byte-Segments und ▲ zur Ausgabe des folgenden 16-Byte-Segment ▼ drücken.
- Mit Format (2) wird der Inhalt desjenigen Segments ausgegeben, das direkt neben dem letzten mit dem D-Befehl ausgegebenen Segments liegt.
- Wenn das Format(3) in der Drucker-Betriebsart ausgeführt wird, gibt der Computer den Inhalt der Bereiche, 0100H (erste Adresse)-01FFH (letzte Adresse), in 16-Byte-Inkrementen auf dem Drucker aus. Nach Beendigung der Ausgabe wird wieder die Befehlseingabe-Zeile dargestellt.
- Die Drucker-Betriebsart wird mit dem P-Befehl (siehe später) oder mit den Tasten SHIFT + P↔NP aktiviert oder beendet.
- Wenn das Format (3) ausgeführt wird, ohne dass sich der Computer in der Drucker-Betriebsart befindet, gibt der Computer den Inhalt der 16-Byte-Segmente auf dem Bildschirm aus, beginnend mit Adresse 0100H (der ersten Adresse). Während der Ausgabe auf dem Bildschirm in dieser Betriebsart berücksichtigt der Computer nicht die letzte bestimmte Adresse.
- Zum Zurückgehen auf die Befehlseingabe-Zeile BREAK drücken.

Kontrollsumme
Kontrollsumme bezieht sich auf die Summe der Werte eines spezifischen Datensatzes. Diese Summe wird berechnet und einem Datensatz zugewiesen, wenn dieser Datensatz geschrieben oder ausgegeben wird. Der Computer berechnet die Summe der Inhalte eines 16-Byte-Segments,das mit dem D-Befehl ausgegeben wurde, und zeigt das wertniedrigste Einzelbyte der Summe als Ergebnis der Kontrollsumme an.
Wenn Sie zum Beispiel manuell ein Maschinencode-Programm eingeben, das von einem gedruckten Programm kopiert wird, dann können Sie in jedem ausgegebenen 16-Byte-Segment nach Fehlern suchen, indem die Ergebnisse der Kontrollsumme mit den Werten des Originalprogramms verglichen werden. Wenn das Programm allerdings mehr als einen Fehler enthält, kann die Kontrollsumme fälschlicherweise mit der des Originalprogramms übereinstimmen.

E – Speicher-Editor

Wirkung: Editieren des Speicherbereichs.
Format: (1) E<startadresse>↵
(2) E↵
Anmerkungen: • Mit Format (1) wird der Speicher ab der angegeben Adresse. Format (2) setzt das Editieren mit dem darauffolgenden Speicherblock des letzten Editierens fort.

- Genau wie mit dem Befehl S können mit E die Speicherinhalte verändert werden. Der Unterschied ist, dass der Editor deutlich bequemer zu verwenden ist.
- Das Umschreiben der Speicherinhalt, können Sie auch den Befehl S, beispielsweise Umwandlung kann einen bequem zu den E-Befehl verwenden.
- Der editierbare Bereich geht von 0000H-07FFFH
- Um sich in dem Editor zwischen den Speicherbereichen zu bewegen benutzen Sie die Cursortasten (◄ ► ▲ ▼)
- Die Daten werden in hexadezimaler Schreibweise 0-F eingegeben. Neben dem normalen Buchstabenfeld kann dazu auch das Zahlenfeld wie angegeben verwendet werden:

7	8	9	／ (F)
4	5	6	* (E)
1	2	3	− (D)
0	. (A)	= (B)	+ (C)

- Mit Hilfe der TAB-Taste kann zwischen hexadezimaler Eingabe (links) und ASCII-Eingabe (rechts) hin und her gewechselt werden.
- Der Kana-Modus ist in während des Editierens nicht möglich.

P – Druckereinstellung

Wirkung: Aktivieren oder Deaktivieren der Drucker-Betriebsart.

Format: P[↵]

Anmerkungen:
- Die Drucker-Betriebsart wird jedes Mal beim Drücken von P[↵] aktiviert bzw. deaktiviert. (Bei aktivierter Drucker-Betriebsart erscheint in der unteren rechten Ecke des Displays die Anzeige "PRINT".) Die Drucker-Betriebsart kann auch durch Drücken von [SHIFT] + [P↔NP] aktiviert bzw. deaktiviert werden.

Hinweis:
- Der P-Befehl wird nicht ausgeführt, wenn kein Drucker angeschlossen oder ein angeschlossener Drucker nicht eingeschaltet ist.

G – GOSUB

Wirkung: Ausführen von Maschinencode-Programmen von einer bestimmten Adresse aus.

Format: G<adresse>[↵]

Anmerkungen:
- Dieser G-Befehl entspricht dem GOSUB-Befehl im BASIC. Damit wird ein Maschinencode-Programm auf einer bestimmten Adresse ausgeführt. Der Rücksprung erfolgt beim Auftreten eines RET-Befehls (Return-Anweisung). Nach der Ausführung der Return-Anweisung zeigt der Computer wieder die Befehlseingabe-Zeile an.

Hinweis:
- Am Ende des Programms muss unbedingt eine Return-Anweisung (RET-Befehl) eingefügt werden, andernfalls kann das Programm nicht richtig ausgeführt werden.

Außer Kontrolle geratene Programme

Ein "Runaway"-Programm kann nicht richtig ausgeführt werden, weil es außer Kontrolle geraten ist. Das Reset des Systems ist die einzige Möglichkeit ein derartiges Programm zu unterbrechen. In den meisten Fällen zerstört ein aus der Kontrolle geratenes Programm die Speicherinhalte, einschließlich der Maschinencode-Programme, BASIC-Programme und anderer Daten.

Ein Maschinencode-Programm kann bereits außer Kontrolle geraten, wenn es auch nur einen einzigen kleinen Fehler enthält. Aus diesem Grund wird empfohlen, alle BASIC-Programme und andere Informationen auf einem PC zu sichern oder auszudrucken bevor ein Programm in Maschinensprache ausgeführt wird.

R – Empfangen von Daten über das serielle Interface

Wirkung: Empfangen von Daten über den seriellen I/O-Port (SIO). Dieser Befehl dient dem Empfangen von Maschinencodes von einem Personal Computer oder einem anderen Gerät.

Format: (1) R↵
(2) R<adresse>↵

Anmerkungen: Mit dem R-Befehl werden Daten im Intel-Hex-Format über SIO übertragen/empfangen.

- Mit Format (1) werden empfangene Daten in bestimmte Adressen geladen, die von den empfangenen Daten spezifiziert werden.
- Mit Format (2) werden nachfolgende Daten geladen, die mit dem Beginn ab der angegeben Adresse (z.B. 0100H) empfangen werden.
- Nach Übertragung aller Daten wird der Adressbereich angezeigt, in den die Daten geladen wurden.
- Um die Übertragung von Daten abzubrechen, wird die Taste BREAK gedrückt gehalten bis die Befehlseingabezeile angezeigt wird.
- Die Parameter für die serielle Schnittstelle werden im TEXT-Modus eingestellt.

W – Senden von Daten über das serielle Interface

Wirkung: Ausgabe von Daten über den seriellen I/O-Port (SIO). Dieser Befehl dient dem Senden von Maschinencodes zu einem Personal Computer oder einem anderen Gerät.

Format: W<startadresse>,<endadresse>↵

Anmerkungen:
- Bei der Ausführung des W-Befehls sendet der Computer Daten im Intel-Hex-Format aus dem Speicherbereich ab der angegebenen Start-Adresse bis zur End-Adresse (z.B. W0100h,01FF) über den seriellen I/O-Port.
- Um das Senden von Daten abzubrechen, wird die Taste BREAK gedrückt gehalten, bis die Befehlseingabe-Zeile angezeigt wird.

Hinweis:
Wenn ein Drucker an den peripheren Interface-Stiftstecker (11-Stifte) angeschlossen ist und der W-Befehl ausgeführt wird, kann es zu Fehlfunktionen sowohl des Computers als auch des Druckers kommen. In diesem Fall den Drucker ausschalten und dann die Taste BREAK gedrückt halten, bis die Befehlseingabe-Zeile an-gezeigt wird.

BP – Unterbrechungspunkt setzen

Wirkung: Einfügen eines Unterbrechungspunktes an einer bestimmten Adresse.
Format: (1) BP<adresse>[,anzahl]↵
(2) BP↵
(3) BP 0↵

Anmerkungen:

- Mit Format (1) wird an der angegebenen Adresse ein Unterbrechungspunkt eingefügt. In einem Programm können mit dem Format (1) bis zu 4 Unterbrechungspunkte an verschiedenen Adressen eingefügt werden. Der mögliche Adressbereich geht von 0000H bis 7FFFH.
- Mit der Anzahl können Sie festlegen nach dem wievielten Durchlaufen der angegebenen Adresse erst die Programmausführung gestoppt werden soll. Es kann ein Wert von 0-255 angegeben werden. Die Angabe von 0 löscht den Unterbrechungspunkt Format(3). Wenn die Anzahl nicht angegeben wird, wird der standardmäßig der Wert auf 1 gesetzt, dass heißt die Ausführung wird beim ersten Erreichen des Unterbrechungspunktes gestoppt
- Beim Versuch der Eingabe eines fünften Unterbrechungspunktes wird der erste Unterbrechungspunkt gelöscht. Es können daher niemals mehr als vier Unterbrechungspunkte in einem Programm enthalten sein.
- Ein Unterbrechungspunkt sollte unbedingt bei einer Befehlsadresse (OP-Code) eingefügt werden. Wenn der Unterbrechungspunkt an einer Operandenadresse eingefügt wird, kann das Programm den Unterbrechungspunkt nicht lesen und er wird daher auch nicht richtig ausgeführt.
- Mit Format (2) wird die Adresse des Unterbrechungspunktes angezeigt. Wenn kein Unterbrechungspunkt eingefügt wurde, erscheint nur die Eingabeaufforderung der Befehlseingabe-Zeile (*) auf der folgenden Zeile.
- Mit Format (3) werden alle vorhandenen Unterbrechungspunkte gelöscht.
- Ein Unterbrechungspunkt wird nach seiner Ausführung unwirksam, Wenn sich daher ein Unterbrechungspunkt innerhalb einer Programmschleife befindet, wird er nur bei der ersten oder n-ten (laut Anzahl) Ausführung der Schleife berücksichtigt. Er kann allerdings mit dem G-Befehl wieder wirksam gemacht werden.
- Der Computer behält Unterbrechungspunkte bei, die bei der letzten Verwendung des Monitors bestimmt wurden. Wenn der Computer aus einer anderen Betriebsart wieder in die Monitor-Betriebsart zurückgestellt wird, können diese Unterbrechungspunkte mit dem G-Befehl wieder wirksam gemacht werden.

Hinweis:
Der Inhalt einer Adresse, die eine Unterbrechungspunkt-Anweisung enthält, wird zeitweise durch "F7H" ersetzt, während das Programm ausgeführt wird. Wenn der RESET-Schalter vor der Ausführung dieses Unterbrechungspunktes gedrückt wird, bleibt der Inhalt "F7H". In diesem Fall muss "F7H" durch den ursprünglichen Inhalt ersetzt werden.

10.3 Fehlermeldungen des Monitor-Modus

Es folgt eine Liste mit Fehlermeldungen, die während der Monitor-Betriebsart ausgegeben werden. Zum Löschen der Fehlermeldung CLS drücken.

Fehlermeldung	Beschreibung
SYNTAX ERROR	Unzulässige Befehlssyntax
MEMORY ERROR	Es wurde versuch, einen Maschinencode-Bereich außerhalb des zulässigen Bereiches zuzuweisen.
I/O DEVICE ERROR	Fehler bei der Datenübertragung oder Fehler der Kontrollsumme während eines I/O-Vorgangs
OTHER ERROR	Andere Fehler.

11 ASSEMBLER

Referenz
Es folgt eine Liste der Fachvokabeln, die häufig beim Umgang mit Maschinensprachen-Programmen verwendet werden.

Assemblieren, Übersetzen:
Ein Quellprogramm der Assemblersprache in eine Maschinensprache übersetzen. Ein übersetztes Maschinencode-Programm wird "Objektprogramm" oder kurz "Objekt" genannt.

Assembler:
Übersetzungsprogramm zur Übersetzung eines Quellprogramms in ein Objektprogramm.

Generieren:
Ein Objekt aus einem mnemonischen Code erzeugen.

Assemblieren von Hand:
Manuelle Übersetzung eines Quellprogramms ohne einen Assembler.

Maschinensprache:
Eine Computersprache, die direkt von einer Maschine interpretiert und deren Befehle ausgeführt werden. Als Hexadezimalcode dargestellt (intern als Binärcode verarbeitet.)

Mnemonische Code:
Symbole, die dem Programmierer helfen sollen, sich die Befehle für Maschinencode-Anweisungen zu behalten. Z.B. die Abkürzung "ADD" für einen zusätzlichen Befehl (additional command). Eine Sprache, deren mnemonische Anweisungen eine spezifische Übereinstimmung mit dem Maschinencode haben, wird "Assemblersprache" genannt.

Objektprogramm:
Ein vollständig assembliertes Programm, das bereit zum Laden in einen Computer ist. Die Bezeichnung bezieht sich im allgemeinen auf ein Maschinencode-Programm, dass von einem Quellprogramm übersetzt wurde. Manchmal einfach als "Objekt" bezeichnet. (" Objekt" kann sich entweder auf einen individuellen Maschinencode beziehen, der nach einer Übersetzung entstand, oder es kann sich auf ein ganzes Maschinensprachen-Programmbeziehen.)

Pseudobefehl:
Eine Folge von Assembler-Steuerungsbefehlen, die nicht in ein Maschinencode-Programm übersetzt werden. Eine derartige Folge wird verwendet, um
eine Adresse zu bestimmen, ein Maschinencode-Programm zu speichern oder Daten zu generieren.

Quellprogramm:
Ein Programm, dass in einem mnemonischen Code (Assemblersprache) geschrieben wurde. Ein Maschinencode-Programm ist eine Übersetzung eines Quellprogramrns

11.1Programmieren mit Assembler

Vor der Beschreibung des Assemblers soll zunächst ein Beispiel-Programm assembliert und das daraus entstehende Objekt (das Maschinencode-Programm) ausgeführt werden. Zunächst muss allerdings darauf hingewiesen werden, dass verschiedene störende Situationen bei der Ausführung von Programmen auftreten können. Wenn das Maschinenprogramm einen oder mehrere Fehler enthält, können die folgenden Fehlersituationen auftreten:

- Der Computer führt das Programm mit einer Endlosschleife aus und reagiert nicht mehr auf das Drücken von Tasten.
  Zum Unterbrechen dieser Endlosschleife muss die RESET-Taste gedrückt werden.
- Das Programm führt zur Anzeige von zufälligen oder unsinnigen Zeichen oder zeigt andere Auffälligkeiten.
  In einigen Fällen kann das Programm mit der Taste BREAK gestoppt werden, aber in anderen Fällen muss der RESET-Schalter gedrückt werden.
- Teile oder das ganze Programm werden zerstört oder gehen verloren. Hier liegt ein Speicherfehler vor. Es kann auch zur Zerstörung von Quellprogrammen (TEXT), BASIC-Programmen oder aller Daten des Computers kommen, einschließlich dem Maschinencode-Programm.

Diese Probleme können einzeln oder gleichzeitig alle zusammen auftreten. Falls eines dieser Probleme auftritt und Sie nicht feststellen können, was gerade vorgeht, drücken Sie die RESET-Taste um den gesamten Speicherinhalt zu löschen.
• Die obigen Probleme (1) und (2) werden "Runaway-Programme" genannt.

Eine kurze Anleitung zum Programmierung des Z80-Prozessor ist im Anhang auf Seite 299 zu finden.

Beispiel-Programm
Mit dem folgenden Programm werden die Hexadezimalzahlen von 20H bis 9FH in dem Speicherbereich von 0400H bis047FH geladen (das H am Ende zeigt an das es sich um eine hexadezimale Notation handelt):

```
10          ORG  0100H
20START:    LD   A,20H
30          LD   HL,0400H
40LBL:      LD   (HL),A
50          INC  A
60          INC  HL
70          CP   0A0H
80          JP   NZ,LBL
90          RET
100         END
```

Hinweis: Mit der Taste SPACE oder mit TAB können eine oder mehrere Leerstellen eingefügt werden.

Beschreibung des Beispiel-Programms:
10: (Laden des Objekts in den Bereich, der mit der Adresse 0100H beginnt.)
20: 20H in das Register A laden.
30: 0400H in das Registerpaar HL laden.
40: Den Inhalt von Register A in eine Adresse laden, die durch das Registerpaar HL bestimmt wird.
50: Den Wert von Register A um eins erhöhen und das Ergebnis in Register A laden.
60: Den Wert des Registerpaars HL um eins erhöhen und das Ergebnis in HL laden.
70: Den Inhalt von Register A mit dem Wert AOH vergleichen (AOH vom Inhalt von A subtrahieren).
80: Wenn das Ergebnis des letzten Vorgangs nicht Null ist (Inhalt von A ist nicht gleich AOH), erfolgt ein Sprung auf das Label LBL. Das Label wird in eine Adresse (0105H) übersetzt.
90: Rückkehr (Rückkehr von der Subroutine).
100: (Ende des Quellprogramms)

Die Zeilen 10 und 100 dieses Quellprogramms werden Pseudobefehle genannt. Sie dienen der Steuerung des Assemblers und werden nicht in Maschinencodes (Objekte) umgewandelt.
Nach der Eingabe aller Zeilen dieses Beispiel-Programms mu6 eine Fehlersuche ausgeführt werden. Vor der Assemblierung des Quellprogramms muss ein Bereich zum Speichern für den Maschinencode zugewiesen werden, andernfalls ist es nicht möglich, das Quellprogramm zu assemblieren.

Zuweisen eines Maschinencode-Bereiches

Zur Zuweisung eines Maschinencode-Bereiches wird in der Monitor-Betriebsart der USER-Befehl verwendet.

Zunächst wird die Monitor-Betriebsart gewählt.
[BASIC] MON [↵]

```
MACHINE LANGUAGE MONITOR
*
```

Nun wird mit de USER-Befehl der Maschinencode-Bereich zugewiesen, in diesem Beispiel von 0100H bis 04FFH.

USER 04FF[↵]

Der Computer zeigt den zugewiesenen Maschinencode-Bereich (Benutzer-Bereich) an.

```
MACHINE LANGUAGE MONITOR
*USER04FF
FREE:0100-04FF
*
```

Assemblieren des Quellprogramms

Nun kann das Quellprogramm dieses Beispiels in Maschinencode umgewandelt werden.

Die Assembler-Funktion wählen.

SHIFT + ASMBL

```
***** ASSEMBLER *****
user area=0100H-04FFH
work area=29221bytes
< Asm  Display  Print >
```

(Die Größe des Arbeitsbereiches kann anders sein als die in diesem Beispiel dargestellte.)

Die Taste **A** drücken um mit der Assemblierung zu beginnen.

```
***** ASSEMBLER *****

  --- assembling ---
```

Nach Beendigung der Assemblierung erscheint eine Anzeige wie die rechts abgebildete.

```
object:0100H-010DH
size  :000EH(   14)bytes
label :    2
error :    0   complete !
```

Wenn während der Assemblierung ein Fehler auftritt, zeigt der Computer die entsprechende Fehlermeldung und die Zeilennummer an, bei der der Fehler auftrat.

```
***** ASSEMBLER *****
*FORMAT ERROR          (1)
0105 ****              40
       LBL:  LD     HL),A
```

In diesem Fall zum Editor zurückgehen und das Quellprogramm korrigieren.

Überprüfung des generierten Objektprogramms

Das generierte Objektprogramrn wird mit dem Monitor überprüft. Das Programm ist in einem Bereich von 0100H bis 010DH gespeichert.

Zur Einstellung der Monitor-Betriebsart CLS drücken(oder BASIC MON ↵ eingegeben).

```
MACHINE LANGUAGE MONITOR
*
```

Das Objektprogramm mit dem D-Befehl ausgeben.
D0100↵
Der Computer zeigt den abgebildeten Speicherauszug des Objektprogramms an.

```
0100 : 3E 20 21 00  > !.
(88)   04 77 3C 23  .w<#
       FE A0 C2 05  .  .
       01 C9 00 00  .  ..
```

(Teile des letzten Speicherinhaltes können in einem Bereich beginnend mit 010DH (C9) angezeigt werden.)
(88) ist die Kontrollsumme.

Ausführen des Objektprogramms (Maschinencode-Programms)

Nun soll das generierte Objektprogramm ausgeführt werden. Dafür wird der Monitorbefehl G (GOSUB) verwendet.

Die Eingabeaufforderung der Befehlseingabe-Zeile der Monitor-Betriebsart zur Anzeige bringen.
BREAK

```
*
```

Den folgenden G-Befehl verwenden, um das Objektprogramm laufen zu lassen.
G0100↵
Nach der Ausführung wird wieder die Befehlseingabe-Zeile des Monitors angezeigt.

```
*G0100
*
```

Nun wird das Ergebnis der Programmausführung überprüft.
D0400↵

```
0400 :  20 21 22 23   !" #
(78)    24 25 26 27  $%&'
        28 29 2A 2B  ( ) *+
        2C 2D 2E 2F  , -. /
0410 :  30 31 32 33  0123
        34 35 36 37  4567
```

▼
Die hexadezimalen Zahlen von 20H bis
9FH wurden in den Adressbereich von 0400H bis 047FH geschrieben.

```
0410 :  30 31 32 33  0123
(78)    34 35 36 37  4567
        38 39 3A 3B  89:;
        3C 3D 3E 3F  <=>?
0420 :  40 41 42 43  @ABC
        44 45 46 47  DEFG
```

11.2Codieren und Editieren eines Quellprogramms

Der Assembler des Computers übersetzt (assembliert) das im TEXT- Bereich gespeicherte Quellprogramm in ein Maschinencode-Programm. Das assemblierte Maschinencode-Programm wird sequentiell in einen Speicherbereich geladen, an der spezifizierten Adresse beginnt.

In diesem Abschnitt werden die Konventionen und Regeln (Eingabeformate u.a,) beschrieben, die bei der Erstellung eines Quellprogramms verwendet werden.

Konfiguration des Quellprogramms

Jede Zeile eines Quellprogramms enthält normalerweise eine einzige Anweisung. Ein Programm besteht im allgemeinen aus einigen Zeilen. Ein Quellprogramm in der Assemblersprache beginnt mit einer ORG-Anweisung und endet mit einer END-Anweisung (die ORG- und die END-Anweisungen können weggelassen werden).

Beispiel:

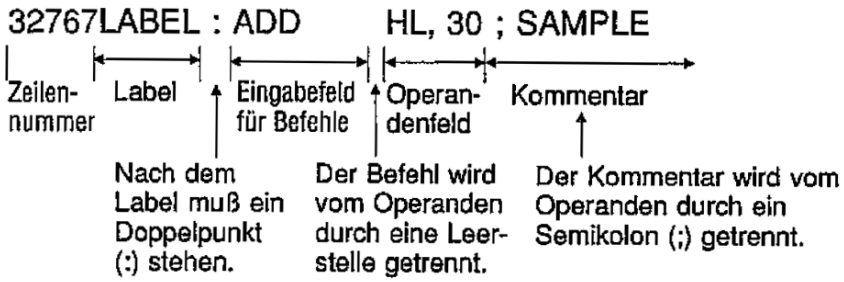
```
10      ORG         0100H
        :::::::
100     END
```

- Die ORG-Anweisung dient der Spezifizierung der ersten Adresse des Speicherbereiches, in dem das generierte Maschinencode-Programm gespeichert werden soll. Dies bedeutet, dass die Zeilen des Maschinencode-Programms der Reihe nach gespeichert werden, beginnend von der Adresse, die mit der ORG-Anweisung bestimmt wurde. Wenn keine Adresse bestimmt wird, nimmt der Computer die Adresse 0100H als erste Adresse an.
- Die END-Anweisung zeigt das Ende des Quellprogramms an. Der Computer beendet die Assemblierung, wenn er diese Anweisung im Quellprogramm erreicht.

Diese Anweisungen dienen der Steuerung des Assemblers; sie werden nicht in einen Maschinencode umgewandelt.

Zeilenkonfiguration (Anweisungen)

Jede Zeile des Quellprogramms besteht aus einer Zeilennummer, einem Label, einem Befehl einem Operanden, einem Kommentar bzw. einem Pseudobefehl.



- Eine Zeile kann aus bis zu 254 Zeichen bestehen, einschließlich dem Kommentar.
- Klein- und Großbuchstaben werden wie Großbuchstaben verarbeitet, außer wenn sie bei Operanden oder Kommentaren benutzt werden.

(1) Zeilennummern

Wenn eine Zeilennummer außerhalb des zulässigen Bereiches von 1 bis 65279 eingegeben wird, erfolgt die Fehlermeldung "LINE NO, ERROR".

(2) Labels

- Ein Label kann direkt nach der Zeilennummer eingefügt werden (zwischen der Zeilennummer und dem Label darf keine Leerstelle sein, andernfalls erfolgt ein Fehler).
- Das Label kann aus bis zu sechs Zeichen bestehen. Bei mehr als sechs Zeichen erfolgt ein Fehler.
- Die folgenden Zeichen können für Label verwendet werden:
  Buchstaben: A bis Z (a bis z werden wie A bis Z gelesen).
  Zahlen: 0 bis 9
  Symbole: [,], @, ? und _
- Das erste Zeichen eines Labels muss allerdings unbedingt ein Buchstabe oder ein Symbol sein (eine Zahl kann nicht von der Zeilennummer unterschieden werden).
- Als Label können genau die gleichen wie die im folgenden aufgelisteten einzelnen Zeichen oder Zeichenpaare für Register oder Bedingungscodes nicht verwendet werden:
  Einzelregister: A, B, C, D, E, H, L, I, R
  Registerpaare: AF, BC, DE, HL, IX, IY, SP
  Bedingungscode: NZ, Z, NC, C, PO, PE, P, M
- Ein Label muss von einem Doppelpunkt (:) gefolgt werden, andernfalls entsteht ein Fehler. Eine Ausnahme bildet die Definition eines Wertes für ein Label mit dem Pseudobefehl EQU; in diesem Fall muss nach dem Label kein Doppelpunkt folgen.
- Wenn kein Label benötigt wird, muss eine oder mehrere Leerstellen zwischen der Zeilennummer und dem folgenden Befehlswort eingefügt werden. Zum Einfügen von Leerstellen kann die Taste **SPACE** oder **TAB** verwendet werden.

(3) Eingabefeld für Befehle (OP-Code)

- In dem Eingabefeld für Befehle kann ein Z80-Befehl als mnemonisches Symbol eingegeben werden. Auch andere Pseudobefehle können hier eingefügt werden. Ein Befehl ist Teil einer Anweisung, der Anweisungscode oder OP-Code genannt wird.
- Der eingegebene Befehl muss durch eine oder mehrere Leerstellen vom folgenden Operanden getrennt werden. Zum Einfügen von Leerstellen kann die Taste **SPACE** oder **TAB** verwendet werden.

(4) Operandenfeld

Als Operanden werden Register, Adressen oder Konstanten bezeichnet, die bei der Befehlsausführung verwendet werden.

- Zur Trennung von Operanden werden Kommas (,) verwendet.
- Jeder Operand kann aus bis zu 32 Zeichen bestehen.
- Die folgenden Arten von Konstanten können ebenfalls als Operanden benutzt werden:

[Numerische Konstanten]

Binäre, dezimale oder hexadezimale Zahlen:

Binäre Zahlen: Dargestellt als Folge von 1 und 0, mit einem "B" am Ende.
Beispiele: 10111100B, 100000B

Dezimalzahlen: Dargestellt als Basis 0 bis 9
Beispiele: 188, 32

Hexadezimale Zahlen: Dargestellt durch Dezimalzahlen 0 bis 9 sowie den Großbuchstaben A bis F; mit einem "H" am Ende.
Wenn eine hexadezimale Zahl mit einem Buchstaben beginnt, muss am Anfang eine "0"stehen, um sie von einem Befehl zu unterscheiden.
Beispiele: 0BCH, 20H

[Zeichenfolge-Konstanten]

Zeichenfolgen bei Operanden müssen in einfachen Hochkommas (') eingeschlossen werden. ASCII-Darstellungen von Zeichen werden bei Operanden als Konstanten verwendet.

Beispiel:

(Spezifizierung)	(Zeichenfolge)	(Konstante)
'A'	A	41H
'AB'	AB	41H, 42H
'B''C'	B'C	42H, 27H, 43H
'''D'	'D	27H, 44H
,E'''	E'	45H, 27H
''''	'	27H
''	(NULL)	00H

[Label-Konstanten]
Wenn eine Konstante für ein Label mit dem Befehl EQU definiert wird, kann dieses Label als Konstante in einem Operanden benutzt werden.

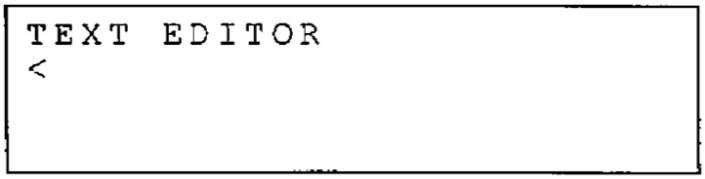
- Ausdrücke (einschließlich arithmetischer Operatoren) können als Operanden verwendet werden. Die folgenden Zeichen und arithmetischen Operatoren können in Operanden benutzt werden; dabei hat jedoch kein Operator Vorrang gegenüber einem anderen.
  Vorzeichen: positiv (+), negativ (-)
  Operatoren: *, /, +,-
- Der Computer führt interne Vorgänge mit 16-Bit-Daten aus. Eine Kapazitätsüberschreitung wird nicht beachtet (es tritt kein Fehler auf). Das Objekt wird mit dem 8-Bit- oder 16-Bit-Ergebnis generiert.
- Bei Anweisungen mit Ausdrücken überprüft der Computer nicht die Richtigkeit des eingegebenen Ausdrucks.
  Beispiele:
  LD A, 4142H → Gelesen als LD A, 42H
  DB 1234H → Gelesen als DB 34H

(5) Kommentare
Jeder Zeile eines Quellprogramms kann ein Kommentar folgen, getrennt durch ein Semikolon (;). Der Anteil der Zeile von einem Semikolon bis zum Ende der Zeile wird als Kommentar angesehen und nicht in Maschinencode (Objekt) übersetzt,

Löschen eines Quellprogramms

Das Hauptmenü in der TEXT-Betriebsart zur Anzeige bringen und **D** drücken, um die Löschfunktion zu wählen. Der Computer fragt zur Sicherheit, ob der Inhalt des TEXT-Bereiches gelöscht werden soll. (Wenn kein Programm im TEXT-Bereich gespeichert ist, reagiert der Computer nicht auf das Drücken von **D** .)
TEXT DELETE OK? (Y)
Zum Löschen aller Informationen aus dem TEXT-Bereich die Taste **Y** drücken. Der Computer geht wieder auf das Hauptmenü der TEXT-Betriebsart zurück. Beim Drücken einer anderen Taste als **Y** geht der Computer ohne etwas zu löschen auf das Hauptmenü der TEXT-Betriebsart zurück.

Eingabe eines Quellprogramms
Das Hauptmenü in der TEXT-Betriebs-
art zur Anzeige bringen und **E** drücken, um die Editierfunktion zu wählen.



Durch Drücken der Taste ▼ bzw. ▲ kann der Inhalt des TEXT-Bereiches dargestellt werden, z.B. ein Quellprogramm. Wenn nichts gespeichert ist, ändert sich die Anzeige nicht.
Ein neues Programm kann erst dann in den TEXT-Bereich geladen werden, wenn der vorhandene Inhalt vollkommen gelöscht wurde. **BREAK** drücken, um auf das Hauptmenü zurückzugehen, die Löschfunktion wählen und die Inhalte des TEXT-Bereiches löschen.

Folgen Sie dabei den Schritten, die im obigen Abschnitt über das Löschen von Quellprogrammen beschrieben wurden.

Eingabe eines Quellprogramms:
(1) Die Zeilennummer eingeben.
(2) Wenn kein Label benötigt wird, kann durch Drücken von **TAB** eine oder mehrere Leerstellen eingefügt werden. Der Cursor bewegt sich wieder auf das Eingabefeld für Befehle. (Die Taste **SPACE** kann ebenfalls anstelle von **TAB** verwendet werden.)
Ein Label wird direkt nach der Zeilennummer eingegeben, ohne eine Leerstelle. Am Ende des Labels muss ein Doppelpunkt (:) stehen. Nach dem Doppelpunkt können nach Belieben eine oder mehrere Leerstellen eingefügt werden.
(3) Einen Befehl eingeben. Wenn nach dem Befehl ein Operand folgt, muss er durch eine oder mehrere Leerstellen von dem Befehl getrennt werden; dazu **TAB** oder **SPACE** drücken.
(4) Die Operanden eingeben.
Die Operanden werden durch Kommas (,) getrennt,
(5) Wenn diese Zeile mit einem Kommentar versehen werden soll, muss vor dem Kommentar ein Semikolon (;) eingegeben werden.
(6) Nach Eingabe der gesamten Zeile wird ↵ gedrückt, um die Zeile zu speichern. Der Cursor verschwindet beim Drücken von ↵.

Zur Eingabe der folgenden Zeile werden die obigen Schritte wiederholt.

11.3Assemblieren

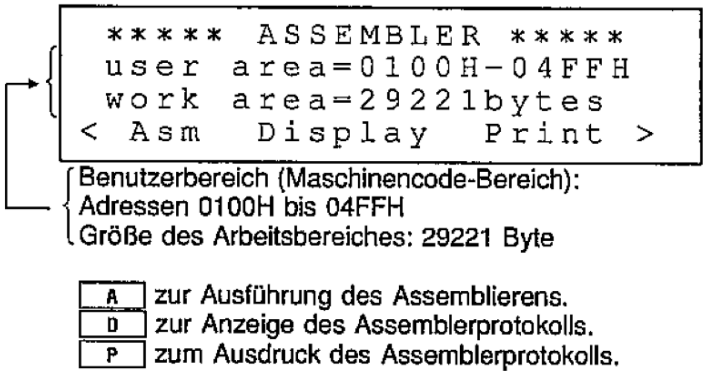
in diesem Abschnitt wird ausführlich beschrieben, wie ein Quellprogramm assembliert wird, das in der TEXT-Betriebsart eingegeben wurde. Eine Voraussetzung dabei ist, dass das Beispiel-Programm bereits im Computer geladen ist. Andernfalls muss es vor dem Ausführen der folgenden Schritte geladen werden.

Das Menü der Assembler-Funktion

Um ein Quellprogramm zu assemblieren, muss zunächst die Assembler-Funktion aktiviert werden.

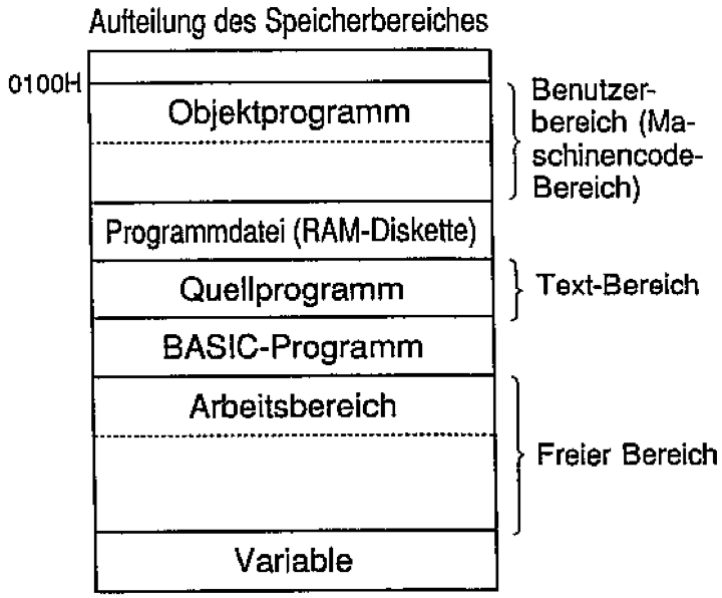
SHIFT + ASMBL

Das rechts abgebildete Menü der Assembler-Funktion erscheint.



- Das Menü zeigt auf der zweiten Zeile den zugewiesenen Maschinencode-Bereich. Zum Zuweisen des Maschinencode-Bereiches wird in der Monitor-Betriebsart der USER-Befehl verwendet. Wenn kein Maschinencode-Bereich zugewiesen wurde oder der Bereich zu klein ist, um das Objekt zu speichern, wird während dem Assemblieren eine Fehlermeldung (NOT RESERVED oder USER AREA OVER) angezeigt. In diesem Fall [BASIC]MON[↵] eingeben, um die Monitor-Betriebsart zu wählen und den Maschinencode-Bereich mit dem USER-Befehl zuzuweisen bzw. zu vergrößern.
- In der dritten Zeile des Assembler-Menüs wird die Größe des vorhandenen Arbeitsbereiches in Byte angegeben. Diese Information zeigt die Byteanzahl des freien Bereiches im Speicher. Der Wert entspricht der Anzahl, die mit dem FRE-Befehl von BASIC erhalten wird.
  Der für den Umwandlungsvorgang im Computer benötigte Arbeitsbereich wird automatisch im freien Bereich zugewiesen. Wenn der Arbeitsbereich nicht zugewiesen werden kann, wird eine Fehlermeldung (WORK AREA OVER) angezeigt. In diesem Fall den freien Bereich durch Löschen von vorhandenen BASIC-Programmen oder anderen Daten vergrößern oder den Maschinencode-Bereich verkleinern.

Hinweis:
Ein Fehler tritt auf wenn nicht wenigstens 307 Byte für den freien Bereich zugewiesen sind während sich der Computer in der Assembler-Betriebsart befindet. Wenn ein Quellprogramm Label enthält, stellt der Assembler einen Label-Arbeitsbereich mit der notwendigen Größe bereit. Ein Fehler tritt auf, wenn der Assembler diesen notwendigen Bereich nicht zuweisen kann.



Assemblieren

Erfolgreiches Assemblieren

Zum Starten des Assemblierens wird **A** bei angezeigtem Assembler-Menü gedrückt.

```
*****  ASSEMBLER  *****

   ---  assembling  ---
```

Während dem Assemblieren wird "--assembling--" angezeigt. Nach der Beendigung des Vorgangs wird "complete !" sowie der Objektbereich und seine Größe, die Anzahl der Labels und die Anzahl der Fehler angezeigt.

```
object:0100H-010DH
size  :000EH(    14)bytes
label :    2
error :    0    complete !
```

Objektbereich: 0100H bis 010DH
Größe des Objektes: 0EH (14) Byte
Anzahl der Labels: 2
Anzahl der Fehler: 0

Nach Anzeige der Meldung "complete!" **CLS** drücken um auf die Monitor-Betriebsart zurückzugehen. In der Monitor-Betriebsart können Sie das assemblierte Objektprogramm mit dem D-Befehl überprüfen oder es mit dem G-Befehl ausführen lassen.

Nicht erfolgreiches Assemblieren

Wenn während dem Assemblieren im Quellprogramm ein Fehler gefunden wird beendet der Assembler den Vorgang und zeigt eine entsprechende Fehlermeldung und die Zeilennummer an bei der der Fehler gefunden wurde. Zum Fortsetzen des Assemblierens ▼ drücken.

Beispiel:
Es wird angenommen, dass das Beispiel-Programm in den Zeilen 50 und 80 je einen Fehler enthält:

```
50    INB   A          ....."INC A" ist richtig.
80    JP    NZ,KBL     ....."JP NZ, LBL" ist richtig.
```

Die Taste **A** bei angezeigtem Assembler-Menü drücken um das Programm mit den angegebenen Fehlern zu assemblieren.

Beim Auffinden des ersten Fehlers wird die rechts abgebildete Fehlermeldung angezeigt.

```
*****  ASSEMBLER  *****
*OPECODE  ERROR
0106  ****                    50
                        INB   A
```

Adresse
Objekt (siehe Anmerkung)
Befehl
Operand
Zeilennummer
Fehlermeldung (zeigt einen OP-Code-Fehler an)

Anmerkung:
Wenn der Assembler das richtige Objektprogramm nicht generieren kann, weil im Quellprogramm ein Fehler vorliegt, wird nach der entsprechenden Adresse eine Folge von Sternchen (****) angezeigt.

Die Taste ▼ drücken um das Assemblieren fortzusetzen. Nun wird die Fehlermeldung für den zweiten Fehler in Zeile 80 angezeigt.

```
  *****  ASSEMBLER  *****
*UNDEFINED  SYMBOL
0109  ****                  80
                  JP      NZ,KB
```

Fehlermeldung (ein undefiniertes Symbol wurde für ein Label verwendet)

Die Taste ▼ erneut drücken.
Die letzte Anzeige des Assemblers erscheint, diesmal allerdings ohne die Meldung "complete !".

```
object:0100H-010CH
size   :000DH(    13)bytes
labe   :    2
error  :    2
```

Bei der letzten Anzeige **CLS** drücken, um auf das Assembler-Menü zurückzugehen.

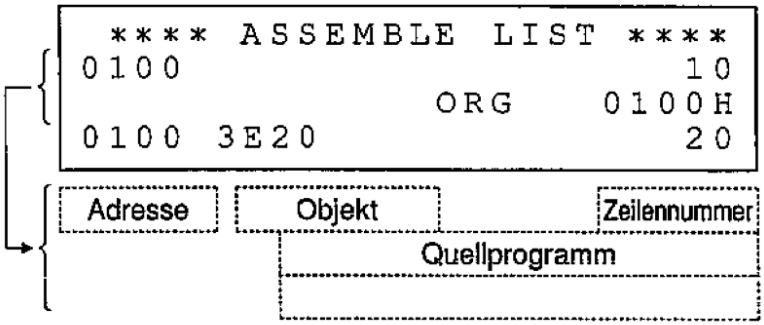
Hinweise:

- Der Assembler ignoriert die Anweisung von Zeile 50 und nimmt an, dass das Label von Zeile 80 die Adresse 0000W spezifiziert; an dieser Stelle wird das Programm dieses Beispiels assembliert.
- Wenn ein Fehler im Quellprogramm aufgefunden wird weist auch das generierte Objekt Fehler auf. Beim Ausführen eines derartigen Objektprogramms kann das Programm außer Kontrolle geraten oder zu Zerstörungen des Speicherinhaltes führen. Das Quellprogramm muss unbedingt korrigiert und neu assembliert werden, damit das Objektprogramm fehlerfrei ausgeführt werden kann.

Anzeige des Assemblerprotokolls

Mit der Anzeige-Möglichkeit des Assembler-Menüs kann ein Objektprogramm überprüft werden, bevor das Quellprogramm übersetzt wird. Das Assemblerprotokoll enthält das zu generierende Maschinencode-Programm, seine Adressen und weitere Objekt-informationen.
Beim Drücken von **D** bei angezeigtem Assembler-Menü wird die erste Zeile des Assemblerprotokolls dargestellt. Durch Drücken von ▼ können die weiteren Zeilen eingesehen werden.
Laden Sie die obige das Beispiel-Programm wenn es nicht bereits geladen ist und überprüfen Sie sein Assemblerprotokoll.

Bei angezeigtem Assembler-Menü **D** drücken.



Wenn das Objektfeld leer ist, gibt es kein Objekt, das generiert werden kann. Wenn der Maschinencode mehr als acht Stellen enthält, werden die übrigen Stellen auf der folgenden Zeile dargestellt.

Die Taste ▼ mehrfach drücken um die folgenden Zeilen des Assemblerprotokolls einzusehen.

```
  **** ASSEMBLE LIST ****
0100                          10
                 ORG    0100H
0100  3E20                    20
        START:LD       A,20H
0102  210004                  30
                 LD     HL,04
        00H
0105  77                      40
        LBL:   LD     (HL),
        A
0106  3C                      50
                 INC    A
0107  23                      60
                 INC    HL
0108  FEA0                    70
                 CP     0A0H
010A  C20501                  80
                 JP     NZ,LB
        L
010D  C9                      90
                 RET
010E                         100
               END

  **** SYMBOL TABLE ****
START  :0100  LBL    :0105
object:0100H-010DH
size   :000EH(    14)bytes
label  :    2
error  :    0    complete !
```

Liste der Symbole (*) →
Objekt-Adressen →
Größe des Objektprogramms →
Anzahl der Labels →
Anzahl der Fehler →

* Die Liste der Symbole zeigt die Werte, die den Labels in hexadezimal zugewiesen wurden.

Hinweise:

- **CLS** drücken, um vom Assemblerprotokoll wieder auf das Assembler-Menü zurückzugehen.
- Mit der Anzeige-Möglichkeit können Sie das Assemblerprotokoll überprüfen; der Maschinencode des Protokolls kann damit aber nicht in den Maschinencode Bereich geladen werden. Zum Laden muss das Quellprogramm ordnungsgemäß assembliert werden.

Ausdrucken des Assemblerprotokolls

Das Assemblerprotokoll kann mit dem Drucker-Befehl des Assembler-Menüs ausgedruckt werden. Den zusätzlichen Drucker CE-126P mit dem Computer verbinden, den Drucker einschalten und bei angezeigtem Assembler-Menü **P** drücken.

Hinweise:

- Bei Wahl der Drucker-Option, ohne dass der Drucker CE-126P angeschlossen bzw. eingeschaltet ist, erscheint eine Fehlermeldung (*PRINTER ERROR). In diesem Fall die Meldung mit **CLS** löschen und den Drucker überprüfen.
- Ein Assemblerprotokoll kann ausgedruckt werden, unabhängig davon, ob auf der unteren rechten Seite des Displays "PRINT" angezeigt wird.
- Nach dem Ausdruck zeigt der Assembler den letzten Assembler-Bildschirm. **CLS** drücken, um wieder auf das Assembler-Menü zurückzugehen.
- Das Protokoll wird in der gleichen Weise ausgedruckt, in der es auf dem Display angezeigt wird.
- Zum Abbrechen des Ausdrucks die Taste **BREAK** gedrückt halten, bis der Drucker stoppt. Auf dem Display wird "--break--" angezeigt. **CLS** drücken um auf das Assembler-Menü zurückzugehen.

Ausgabe/Druck des Assemblerprotokolls über das serielle Interface (SIO)
Das Assemblerprotokoll kann auch mit der Eingabe von **L** im Assembler-Menü über das serielle Interface ausgegeben werden. Die weitere Bedienung ist identisch zur Ausgabe mit „Print“ (siehe oben).
Achtung: Im Gegensatz zu den anderen Assembler-Menü-Befehlen ist der L-Befehl nicht auf dem Bildschirm mit aufgeführt.

11.4Pseudobefehle für den Assembler

Pseudobefehle dienen der Steuerung des Assemblers selbst und werden nicht in Maschinencode umgewandelt. Dieser Computer kennt die folgenden Pseudobefehle:

- ORG: Bestimmt die erste Adresse des Maschinencode-Bereiches.
- DEFB/DB/DEFM/DM, DEFS/DS und DEFW/DW: Definieren von Daten innerhalb
- der Operanden.
- EQU: Definieren von Labelwerten.
- END: Zeigt das Ende des Assemblierens an.

Im folgenden werden einige Konventionen und Regeln beschrieben, die bei den Erklärungen für die Pseudobefehle verwendet werden.

Ausdruck: Ausdrücke können Zahlen, Formeln, Labels oder "Zeichenfolgen" sein.
Formeln: Formeln können Zahlen, Labels oder irgendwelche arithmetischen Ausdrücke sein, bei denen Zahlen oder Labels verwendet werden.
{ }: Wenn mehrere Elemente durch eine geschweifte Klammer zusammengefasst werden, muss eines dieser Elemente gewählt werden.
[ ]: Ein Element innerhalb eckiger Klammern bezeichnet eine optionale Anweisung.
[ ]...: Punkte für Folgeinformationen nach eckigen Klammern bedeuten, dass das Element in den Klammern optional ist oder wiederholt werden kann.

ORG - Beginn

Format: ORG Ausdruck
Wirkung: Bestimmt die erste Adresse des Maschinencode-Bereiches.
Anmerkung: • Der Ausdruck bestimmt die erste Adresse eines Bereiches, in dem der generierte Maschinencode gespeichert wird. Das Maschinencode-Programm wird sequentiell in einen Speicherbereich geladen, der an derjenigen Adresse beginnt, die mit diesem Ausdruck bestimmt wurde.
• Wenn das Quellprogramm keine ORG-Anweisung enthält, nimmt der Assembler die Anweisung "ORG 100H" an; damit wird 100H zur ersten Adresse, von der aus der Maschinencode gespeichert wird.

Beispiel:
ORG 0400H
Mit dieser Anweisung wird der Maschinencode in einen Bereich gespeichert, der bei der Adresse 0400H beginnt.

DEFB/DB/DEFM/DM — Definiere Byte/Definiere Meldung

Format:

[Label:] { DEFB / DB / DEFM } Ausdruck [,Ausdruck]...

Wirkung: Übersetzt eine Zeichenfolge oder das wertniedrigste Einzelbyte einer Zahl des Ausdrucks in den Maschinencode.
Anmerkung: • Mit dieser Anweisung wird das wertniedrigste Einzelbyte einer gegebenen Zahl des Ausdrucks in den Maschinencode umgewandelt.

Beispiel:
DEFB 1234H;Übersetzt1234H in den Maschinencode "34H".
DB 1234; Übersetzt 1234 in den Maschinencode "D2H".

• Eine Zeichenfolge in einem Operanden muss in Hochkommas (') eingeschlossen werden. Sie kann aus bis zu 32 Zeichen bestehen, Einzelne Zeichen einer Operanden-Zeichenfolge werden in den entsprechenden ASCII-Codes übersetzt.

Beispiel:
DEFM 'DATA'; Übersetzt die einzelnen Zeichen der Folge 'DATA' in die Maschinencodes 44H, 41H, 54H und 41H.

• Die Operanden werden durch Kommas (,) getrennt.

Beispiel:
DB 32w4+5,'X2';85H, 58H und 32H werden im Maschinencode generiert.

Beispiel-Programm

	Quellprogramm		Maschinencode-Programm
10	ORG	0100H	
20	LD	HL,DATA	21 0C 01
30	LD	DE,300H	11 00 03
40	LD	BC,5	01 05 00
50	LDIR		ED B0
60	RET		C9
70DATA:	DB	'ABCDEFGH'	41 42 43 44 45 46 47 48
80	END		

Die einzelnen Zeichen der Operanden-Zeichenfolge von Zeile 70 werden in die entsprechenden ASCll-Codes übersetzt.
Bei diesem Beispiel-Programm werden fünf Datenbyte kopiert, die sich in einem Bereich befinden, dessen erste Adresse durch das Label DATA spezifiziert ist; sie werden in einen Bereich kopiert, der mit der Adresse 300H beginnt. Dies bedeutet also, dass die Daten 41H, 42H, 43H, 44H und 45H an den Adressbereich von 300H bis 304H kopiert werden.

DEFW/DW/ — Definiere Wort

Format:

[Label:] { DEFW / DW } Ausdruck [,Ausdruck]...

Wirkung: Übersetzt die wertniedrigsten zwei Byte einer Zahl oder einer Zeichenfolge des Ausdrucks (zwei Zeichen oder weniger) in den Maschinencode.
Anmerkung: • Mit dieser Anweisung werden die wertniedrigsten zwei Byte einer Zahl übersetzt, die im Operanden spezifiziert sind.

Beispiel:
DW 1234H; Übersetzt 1234H in die Maschinencodes 34H und 12H (in der Reihenfolge von wertniedrigstem und werthöchstem Byte).
DEFW 34H; Übersetzt 34H in die Maschinencodes 34H und 00H.

• Eine Zeichenfolge in einem Operanden muss von Hochkommas (') eingeschlossen sein. Es können bis zu zwei Zeichen für eine Zeichenfolge definiert werden.

Beispiel:
DEFW 'DA', Übersetzt die Zeichenfolge 'DA' in den Maschinencode 41H und 44H.
DW 'Z'; Übersetzt die Zeichenfolge 'Z' in den Maschinencode 5AH und 00H.

• Die Operanden werden durch Kommas (,) getrennt.

Beispiel:
DW 'AB','CD', 5678H; Im Maschinencode werden 42H, 41H, 44H, 43H, 78H und 56H generiert.

DEFS/DS/ — Definiere Speicher

Format:

[Label:] { DEFS / DS } Ausdruck [,Ausdruck]...

Wirkung: Generierung von Null-Codes (00H) mit einer Anzahl, die im Operanden spezifiziert wird.
Anmerkung: • Mit dieser Anweisung werden Null-Codes (00H) entsprechend der spezifizierten Byteanzahl generiert.

Beispiel:
DS 12; Generiert 12 Byte mit 00H.

Beispiel:

Quellprogramm			Maschinencode-Programm
10	ORG	0100H	
20	LD	HL,DATA	21 10 01
30	LD	DE,300H	11 00 03
40	LD	BC,5	01 05 00
50	LDIR		ED B0
60	RET		C9
65	DS	4	00 00 00 00
70DATA:	DB	'ABCDEFGH'	41 42 43 44 45 46 47 48
75NXT00:	DS	500H-NXT00	(00H wird an allen folgenden Adressen bis 04FFH eingefügt.)
80	END		

Dieses Beispiel-Programm ist genau so wie das obige, es enthält aber die zusätzlichen Zeilen 65 und 75. Mit Zeile 65 wird ein Speicherbereich für den späteren Gebrauch zugewiesen. Mit Zeile 75 werden Null-Codes (00H) eingefügt um unnötigen Speicherinhalt zu löschen.
00H ist ein "Nulloperations-Code" (NOP), der den Computer anweist, nichts zu tun.

EQU — EQU

Format: [Label:] EQU Ausdruck
Wirkung: Dem Label wird ein Wert zugewiesen, der durch den Operanden spezifiziert wird.
Anmerkung:
- Mit dieser Anweisung wird dem Label ein Wert zugewiesen, der durch den Ausdruck gegeben ist.
- Der Wert kann eine Zahl oder eine Zeichenfolge von ein oder zwei Byte sein.
- Nach dem Label darf kein Doppelpunkt (:) stehen

Beispiel:

```
START EQU 1000H;    Zuweisung des Wertes 1000H zum Label
                    START; das Label kann dann als Konstante
                    1000H bearbeitet werden.
OK    EQU 'Y';      Zuweisung des Wertes 59H zum Label OK.
```

END — Ende

Format: END
Wirkung: Zeigt das Ende eines Quellprogramms an.
Anmerkung:
- Mi t der END-Anweisung wird das Ende eines Quellprogramms bestimmt; der Assembler beendet an dieser Stelle den Umwandlungsvorgang. Informationen, die nach dieser Anweisung folgen, werden nicht mehr assembliert.
- Wenn am Ende eines Quellprogramms keine END-Anweisung vorhanden ist assembliert der Assembler bis zum letzten Teil des TEXT-Bereiches.

11.5Assemblerfehler

Dieser Abschnitt enthält eine Liste mit Fehlermeldungen, die während dem Assemblieren angezeigt werden können, sowie Erklärungen zu diesen Meldungen. Zum Löschen der Fehlermeldung **CLS** drücken. Wenn das Assemblieren beim Auftreten eines Fehlers im Quellprogramm abgebrochen wird kann zum Wiederaufnehmen die Taste ▼ gedrückt werden. Die Fehlermeldung wird ebenfalls gelöscht, wenn der Computer auf eine andere Betriebsart eingestellt wird.

Fehlerart	Beschreibung (Ursache)
OPECODE ERROR	Ungültiger OP-Code (Befehlscode),
FORMAT ERROR (1)	Ungültiges Trennzeichen für Operatoren
FORMAT ERROR (2)	Ungültiger Code (ASCII-Code 01H-1FH o.a.) oder Zeichen in einem Operanden (derartige Codes bzw. Zeichen können normalerweise allerdings nicht eingegeben werden).
FORMAT ERROR (3)	Ungültige Anzahl von Operanden

FORMAT ERROR (4)	Ungültige Zeichen wurden in einem Label verwendet.
FORMAT ERROR (5)	Ein Label hat mehr als sechs Zeichen
FORMAT ERROR (6)	Die Zeichenfolge im Operanden ist nicht in Hochkommas eingeschlossen.
FORMAT ERROR (7)	Die Anzahl der Zeichen in einer Anweisung oder einem einzelnen Operanden überschreitet 32 (z.B. Wert der Adresse o.a. in einem Operanden hat zu viele voranstehende Nullen.)
QUESTIONABLE OPERAND (1)	Ungültiger Operand.
QUESTIONABLE OPERAND (2)	Ungültige Bedingung (NZ, Z, NC o.a.)
QUESTIONABLE OPERAND (3)	Der Wert des Operanden überschreitet die zulässige Grenze.
QUESTIONABLE OPERAND (4)	Die Zeichenfolge im Operanden überschreitet die zulässige Länge von 32 Zeichen.
QUESTIONABLE OPERAND (5)	Versuch der Division durch Null.
QUESTIONABLE OPERAND (6)	Andere ungültige Werte oder Ausdrücke.
UNDEFINED SYMBOL	Ein nicht definiertes Symbol (Label) wurde verwendet.
MULTI DEFINE SYMBOL	Das gleiche Symbol (Label) wurde mehr als einmal definiert.
FILE NOT EXIST	Das zu assemblierende Programm befindet sich nicht im TEXT-Bereich.
USER AREA OVER	Das Objekt konnte nicht in den Maschinencode- Bereich geladen werden. (Die erste Adresse des Objektbereiches, die mit der ORG-Anweisung spezifiziert wurde, liegt außerhalb des Maschinencode-Bereiches oder das Objekt hat beim Laden die Kapazität des Maschinencode-Bereiches überschritten,)
WORK AREA OVER	Die Größe des freien Bereiches ist zu klein für den notwendigen Arbeitsbereich zum Assemblieren (wenn sich der Computer in der Assembler-Betriebsart befindet oder assembliert).
PRINTER ERROR	Der Drucker ist nicht startbereit oder funktioniert nicht. (Der Drucker ist nicht angeschlossen, ausgeschaltet oder wegen einer entladenen Batterie nicht funktionsfähig.)

12 PIC

An den SHARP PC-G850V(S) verfügt über ein Interface für PIC-Devices (Peripheral Interface Controller). Damit können diese Controller mit dem Pocket-Computer programmiert werden.

Folgende Geräte werden unterstützt (Stand 2001):

Gerät	Programmspeicher	PIN-Anzahl
PIC16F627	1K-Worte	18
PIC16F83	512-Worte	18
PIC16F84	1K-Worte	18
PIC16F84A	1K-Worte	18

Der PIC-Modus besteht aus zwei Funktionen:

Assembler Programme werden im TEXT-Modus erstellt und anschließend assembliert.

Loader Übertragen des Objektprogramms in das PIC-Modul

Auf der Webseite http://www.sprut.de/electronic/pic/index.htm kann man detaillierte Hinweise in deutsch für die Programmierung der PIC-Controller finden.

12.1 Definition des Maschinensprachebereiches

Stellen Sie sicher, dass genug Speicher im Maschinensprache-Bereich definiert ist.

Für das PIC-Interface benötigt das System mehr als 1K-Worte. Daher sollten mindestens 3KByte definiert werden. Dazu mit dem USER-Befehl im Maschinensprache-Monitor einen freien Bereich von 3K definierten:

BASIC MON ↵

USER0CFF

```
MACHINE LANGUAGE MONITOR
*USER0CFF
FREE:0100-0CFF
*
```

(Der Speicherbereich von 0100H-0CFFH wird als freier Bereich definiert)

12.2Erstellen und Bearbeiten eines Quellprogramms

Das Quellprogramm wird genauso erstellt oder bearbeitet wie Z80- oder CASL-Assemblerprogramme. Die Assemblerprogramme müssen Konform zur MPLAB-Spezifikation geschrieben werden.

Wie bei den anderen Assembler-Sprachen darf in pro Zeile nur ein Befehl geschrieben werden.

Eine Befehlszeile besteht aus einer Zeilennummer, Label, Befehlscode, Operand und Kommentar.

<zeilennummer>[label] Opcode operand [;kommentar]

Beispiel:

32767LABEL MOVLW 0x0F9;SAMPLE

Vor und nach dem Befehl muss mindestens jeweils ein Leerzeichen oder TAB geschrieben werden.

Eine Zeile darf einschließlich des Kommentars maximal 254 Zeichen lang sein.

zeilennummer: Jede Zeile muss eine Zeilennummer enthalten. Diese Nummer kann einen Wert zwischen 1 und 65279 annehmen. Wird vom Assembler ein Wert außerhalb dieses Bereichs erkannt wird die Meldung „LINE NO. ERROR“ ausgegeben.

label: Das optionale Label muss unmittelbar nach der Zeilennummer beginnen. Die Länge des Labels darf zwischen 1 und 8 Zeichen liegen. Verwendet werden dürfen nur alphanumerische Zeichen (A-Z und 0-9). Das Label muss aber mit einem Buchstaben beginnen.

OpCode: Hier werden die Operationscodes der 35 de 14-Bit-Kernels angegeben. Dazu gehören auch die speziellen Assembler-Befehle. Als Abgrenzung zum Operanden muss mindestens ein Leerzeichen oder ein TAB eingegeben werden.

operand: Ein oder mehrere Operanden (durch Komma getrennt).

Folgende Arten Konstanten sind möglich:

[Numerische Konstanten]
Zahlen können in dezimaler und hexadezimaler Schreibweise geschrieben werden:
Dezimal: (0-9), Beispiel : 188, 32
Hexadezimal: (beginnend mit 0x 0-9,A,B,C,D,E,F)

Beispiel: 0xBC, 0x20

[Zeichenkonstante]
Zeichenkonstanten müssen durch ein Hochkomma eingeschlossen werden. Soll z.B.

Beispiel:	Zeichen	Operand	Zahlenwert
	A	'A'	0x41
	(NULL)	' '	0x0

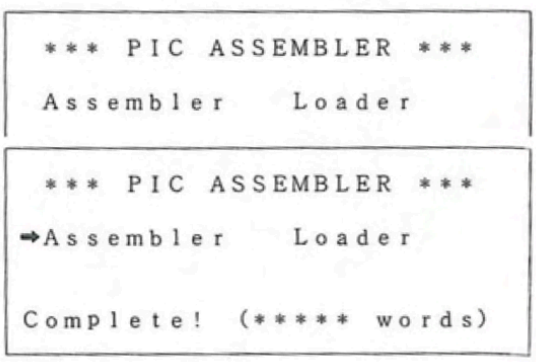
[Adresskonstante]
Der Operand ist ein Label z.B. einer EQU-Anweisung

kommentar: der optionale Kommentar muss mit einem Semikolon beginnen. Bis zum Ende der Zeile werden dann alle folgenden Zeichen als Kommentar behandelt. Diese Zeichen werden nicht mit assembliert, gehören also nicht mit zum Objektprogramm.

12.3PIC-Assembler

Das Quellprogramm muss in den TEXT-Editor eingegeben oder geladen werden. Danach wechseln Sie in den PIC-Modus:

Drücken Sie [SHIFT] + [ASMBL] und danach **P** um in den PIC-Modus zu gelangen.



Zum Assemblieren des Programms drücken Sie dann **A**.

Während des Assemblierens erscheint im unteren linken Bereich "Assembling....". Ist der Vorgang abgeschlossen erscheint dort die Ausschrift "Complete! (***** words)". Für ***** wird die Größe des Programms in Worten angezeigt.

12.4Richtlinien des PIC-Assemblers

Der Assembler verfügt über Befehle um den Assembler selber zu steuern und Definitionen zu deklarieren. Diese Befehle werden nicht direkt Teil des Objektprogramms.

Festlegen der Konfiguration: __CONFIG

Angabe der Programm-Startadresse: ORG

Wertedefinition: EQU

Definition von Daten: DW

__CONFIG Konfiguration

Funktion: Festlegen der Konfiguration

Format: __CONFIG ausdruck

Beschreibung: Konfigurationsbit für jeden PIC. Die anzugebenen Werte entnehmen Sie bitte der Dokumentation des PIC-Moduls.

Laut MPASM-Spezifikation können die Bits mit „&“ (AND) verknüpft werden. Dies funktioniert aber nicht bei diesem Computer.

Beispiel: __CONFIG 0x3FA8

ORG Startadresse festlegen

Funktion: Festlegen der Startadresse des Programms

Format: ORG adresse

Beschreibung: Legt die Startadresse des Objektprogramms an. Wird die ORG-Anweisung nicht angegeben, wird die Startadresse 0 (ORG 0) angenommen.

Es kann ein Wert von 0x0 bis 0x1FFF angegeben werden, abhängig von den Erfordernissen des PIC-Moduls

Beispiel: ORG 0x0006

EQU Definition einer Konstante

Funktion: Definiert eine Konstante, die dann über das Label im Programm verwendet werden kann.

Format: label EQU ausdruck

Beschreibung: Der Ausdruck kann ein numerischer Wert oder ein Zeichen sein.

Beispiel:	START	EQU	0x1000	Definiert für START die Konstante 0x1000
	OK	EQU	'Y'	Definiert für OK den Wert 0x59

DW Definition eines Wortes

Funktion: Definiert ein Wort (2 Byte) im Programm.

Format: [label] EQU ausdruck

Beschreibung: Der Ausdruck kann ein numerischer Wert oder ein Zeichen sein. Beachten Sie, dass dies ein 14-Bit-System ist und der Wert nur bis 0x3FFF gehen darf.

Beispiel: DW 0x1234

12.5#INCLUDE

Funktion: Mit dem #INCLUDE-Befehl wird während der Assemblierung eine Datei für die speziellen PIC-Module in das Quellprogramm eingefügt. Diese Include-Datei enthält für das spezifische Modul Standarddefinitionen.

Format: #INCLUDE "datei"

Beschreibung: Diese Dateien enthalten LABEL-Definitionen der MPASM-Spezifikation für ein spezielles Modul. Die im Operanden angegebene Datei muss in Doppelhochkommas (Gänsefüßchen) eingeschlossen werden.

- Folgende Dateien können verwendet werden:
  Für die PIC-Module: P16F627.INC, P16F83.INC, P16F84.INC, P16F84A.INC
  Für den 14bit-Flash-Speicher: PIC.INC
- Die durch die INCLUDE-Anweisung definierten Labels sind nicht Bestandteil der 102 Labels die vom User definiert werden können.
- In jedem Programm kann nur eine INLCUDE-Anweisung enthalten sein. Diese sollte am Anfang des Programms stehen.
- Labels der MPASM-Spezifikation die mehr als 8 Zeichen lang sind werden auf 8 bei diesem Computer auf 8 Zeichen beschränkt:

MPASM-Label	Label in diesem Computer
OPTION_REC	OPTION_R
NOT_T1SYNC	NOT_T1SY

12.6Fehlermeldungen

Bei der Verwendung des Assemblers können Fehler auftreten (siehe Tabelle). Durch Drücken von CLS löschen sie den Fehler. Dann können sie den Fehler korrigieren (z.B. im TEXT-Editor).

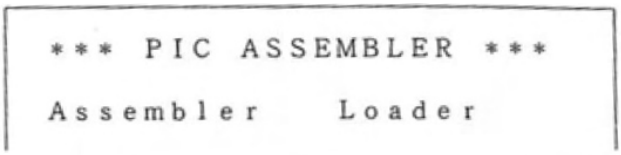
Fehlermeldung	Beschreibung
File not exist!	Kein Programm im TEXT-Editor enthalten
No USER AREA!	Es wurde kein Maschinensprachen-Bereich definiert
Not __CONFIG data!	Es gibt keine __CONFIG-Richtlinie
Syntax error! (*****)	Falscher __CONFIG_Parameter
	Falscher ORG-Parameter
	Der EQU-Befejhl hat kein Label
	Unerlaubte Speicheradresse
	Nach dem Operand folgt kein Space/TAB/CR
	Nach dem OpCode folgt kein Space/TAB/CR
	Die Operanden wurden mit Space/TAB getrennt.
	Falscher Operand
	Falscher OPCode
	Falscher Preprozessor-Befehl
Out of range! (*****)	Adresse, Inhalt befindet sich außerhalb des erlaubten Bereichs.
Undefined label! (*****)	Das angegebene Label existiert nicht.
Undefined line! (*****)	Die angegebene Adresse ist höher als die vom Maschinensprache-Bereich

Label too long! (*****)	Das Label hat mehr als 8 Zeichen
Out of memory! (*****)	Adresse im ORG-Befehl über 8K, Das Programm ist aus dem Maschinensprache-Bereich hinausgelaufen (zu groß), zu viele Label.
Multi define! (*****)	Es darf nur ein INCLUDE-Befehl im Programm enthalten sein. Es gibt 2 oder mehr identische Labels
'Not include file! (*****)	Die angegebene Include-Datei ist ungültig.

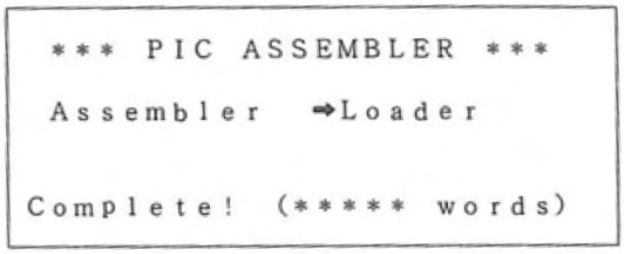
12.7PIC-LOADER

Der Loader überträgt das erfolgreich assemblierte PIC-Programm aus dem Maschinensprache-Bereich in das PIC-Modul.

Drücken Sie [SHIFT] + [ASMBL] und danach **P** um in den PIC-Modus zu gelangen.



Zum Laden des Programms drücken Sie dann **L**



Während des Übertragens erscheint im unteren linken Bereich "Loading....". Ist der Vorgang abgeschlossen erscheint dort die Ausschrift "Complete! (***** words)". Für ***** wird die Größe des Programms in Worten angezeigt.

Fehlermeldungen des PIC-Loaders

Bei der Verwendung des Loaders können Fehler auftreten (siehe Tabelle). Durch Drücken von CLS löschen sie den Fehler. Dann können sie den Fehler korrigieren (z.B. im TEXT-Editor).

Fehlermeldung	Beschreibung
No USER AREA!	Es wurde kein Maschinensprachen-Bereich definiert
Not PIC data!	Die PIC-Datengröße ist 0
Illegal PIC data!	PIC Daten sind größer als der Maschinensprache-Bereich. Das Wort im __CONFIG-Parameter ist falsch
Connection error!	Die Verbindung zum PIC-Modul kann nicht aufgebaut werden.
Low battery!	Während des Schreibens wurde eine zu schwache Batterie erkannt.
Verify error!	Beim Vergleichen/Überprüfen des übertragenen Programms wurde ein Fehler erkannt.
Break!	Die Übertragung oder der Vergleich wurde abgebrochen.

13 BASIC KOMMANDO LEXIKON

ABS

Format: **ABS(<zahl>)**

Die Funktion ABS liefert den Absolutbetrag eines numerischen Ausdruckes.

ACS

Format: **ACS(<zahl>)**

Der Befehl ACS liefert einen zum Argument <zahl> gehörenden Wert der Arcus-Cosinus-Funktion.

Da die Arcus-Cosinus-Funktion die Umkehrung der Cosinus-Funktion ist, stellt der gelieferte Wert folglich einen Winkel dar. In Abhängigkeit von dem derzeit gültigen Winkelmodus (DEGREE, GRAD, oder RADIAN) ist das Ergebnis somit entweder in Altgrad, Neugrad oder im Bogenmaß zu werten. Das zulässige Funktionsargument <zahl> ist auf den Wertebereich -1 s X s 1 beschränkt. Der gelieferte Funktionswert liegt stets in folgenden Hauptwertebereichen:

DEG-Modus
RAD-Modus
GRAD-Modus
0•.... 180• O .... ≤ 0 .... 200 gon

```
1O:DEGREE
20:PRINT "arccos(0.5)=";ACS(.5);" Grad"
30:PRINT "arcoos(O) =";ACS(O);" Grad"
40:END
>
RUN
arccos(O.5) = 60 Grad
arccos(O)   = 90 Grad
>
```

AHC

Format: **AHC(<zahl>)**

Der Befehl AHC liefert einen zum Argument <zahl> gehörenden Wert des reziproken hyperbolischen Kosinus

AHS

Format: **AHS(<Zahl>)**

Der Befehl AHS liefert einen zum Argument <zahl> gehörenden Wert des reziproken hyperbolischen Sinus

AHT

Format: **AHT(<zahl>)**

Der Befehl AHT liefert einen zum Argument <zahl> gehörenden Wert des reziproken hyperbolischen Tangens

ASC

Format: **ASC(<string>)**

Die Funktion ASC liefert den ASCII-Code des Argumentes <string>, welches ein String aus einem oder mehreren Zeichen sein kann.

Wird als Funktionsargument ein String genommen, der aus mehr als einem Zeichen besteht, so wird der ASCII-Code seines ersten Zeichens geliefert.

Der Zusammenhang zwischen dem gelieferten Code und dem zugehörigen Zeichen ist aus dem Anhang H: Tabelle der Zeichencodes auf Seite 284 ersichtlich.

Siehe auch: CHR$

```
10:WAIT 0:CLS
20:PRINT "BITTE EIN ZEICHEN ODER"
30:INPUT "EINEN STRING EINGEBEN:",S$
40:WAIT 100
50:PRINT "DER ASCII-CODE LAUTET:";ASC(S$)
```

```
60:END
RUN
BITTE EIN ZEICHEN ODER
EINEN STRING EINGEBEN:
SHARP
DER ASCII-CODE LAUTET: 83
>
RUN
BITTE EIN ZEICHEN ODER
EINEN STRING EINCEBEN:
*
```

ASN

Format: **ASN(<Zahl>)**

Der Befehl ACS liefert einen zum Argument <zahl> gehörenden Wert der Arcus-Sinus-Funktion.

Da die Arcus-Sinus-Funktion die Umkehrung der Sinus-Funktion ist, stellt der gelieferte Wert folglich einen Winkel dar. In Abhängigkeit von dem derzeit gültigen Winkelmodus (DEGREE, GRAD, oder RADIAN) ist das Ergebnis somit entweder in Altgrad, Neugrad oder im Bogenmaß zu werten. Das zulässige Funktionsargument <zahl> ist auf den Wertebereich -1 _ X _ 1 beschränkt. Der gelieferte Funktionswert liegt stets im Hauptwertebereich. Hierbei gilt:

DEG-Modus : -90°.... 90°
RAD-Modus : π2 .... π/2
GRAD-Modus: (-100 .... 100)

Siehe auch ACS, ATN, SIN

```
10:DEGREE
20:GOSUB 100
30:FOR DX=-10 TO 10
40:X=DX/10
50:F=ASN(X):Z=Z+1
60:IF Z=3 THEN GOSUB 100
70:PAUSE "   ";STR$(X),F
80:NEXT DX
90:END
100:CLS:PRINT"ARGUMENT","ARCUS-SINUS"
110:Z=0:RETURN
```

ATN

Format: **ATN(<Zahl>)**

Der Befehl ATN liefert einen zum Argument <zahl> gehörenden Wert der Arcus-Tangens-Funktion.

Da die Arcus-Tangens-Funktion die Umkehrung der Tangens-Funktion ist, stellt der gelieferte Wert folglich einen Winkel dar. In Abhängigkeit von dem derzeit gültigen Winkelmodus (DEGREE, GRAD, oder RADIAN) ist das Ergebnis somit entweder in Altgrad, Neugrad oder im Bogenmaß zu verstehen. Das Funktionsargument <zahl> unterliegt keiner wertmäßigen Beschränkung. Der Funktionswert wird je nach Winkelmodus innerhalb folgender Hauptwertebereiche geliefert:

DEG-Modus : -90°.... 90°
RAD-Modus : -π/2 .... π/2
GRAD-Modus: (-100 .... 100)

Siehe auch: ACS, ASN, TAN

```
10:DEGREE
20:GOSUB 100
30:FOR DX=0 TO 100
40:X=DX*.1
50:F=ATN(X):Z=Z+1
60:IF Z=3 THEN GOSUB 100
70:PAUSE " ";STR$(X),F
80:NEXT DX
90:END
100:CLS:PRINT "ARGUMENT","ARCUS-TANGENS"
110:Z=0:RETURN
```

AUTO

Format: **AUTO [[<Start-Zeilennummer>][,<Ergänzungswert>]]**

Mit dem Kommando AUTO kann zur Erleichterung des Programmierens im PRO-Modus eine automatische Zeilennummerierung vorgenommen werden.

Nach Aktivierung von AUTO erscheint die erste generierte Zeilennummer in der Anzeige mit einem nachgestellten Cursor, Nun kann der gewünschte Zeileninhalt eingegeben werden. Schließt man die Eingabe dann durch Betätigung der ENTER Taste ab, so wird in der folgenden Zeile die nächste Zeilennummer generiert und so fort.

Ergibt sich bei der Generation der Zeilennummern die Nummer einer bereits

existierenden Zeile, so wird diese Zeile angezeigt,

Die erzeugten Zeilennummern hängen davon ab, ob AUTO mit oder ohne Parameter versehen wird und welche Werte für diese gewählt werden:

AUTO

Wird das Kommando AUTO ohne Parameter angegeben, so beginnt die Nummerierung mit der Zeile 1O und setzt sich im Zehnerabstand, also mit den Zeilen 20, 30 usw. fort.

AUTO <Zeilennummer>

Ist dem Befehlswort AUTO eine einzelne Integerzahl beigefügt, gilt diese als die erste Zeilennummer. Alle weiteren Zeilennummern folgen im Zehnerabstand.

AUTO
AUTO 100
AUTO 400,20
Generierte Zeilennummern
10,20,30,40,.....
100,110,120,.....
400,420,440,.....

AUTO <Zeilennummer>,<Zeilenabstand>

Mit einem weiteren Parameter kann die Schrittweite der Zeilennummerierung bestimmt werden. Die beiden Parameter sind durch ein Komma voneinander zu trennen.

Die automatische Zeilennummerierung lässt sich mit Betätigung der BREAK- oder CL Taste aufheben. Es kann aber auch die ENTER Taste zu diesem Zweck verwendet werden, wenn man diese gleich nach dem Erscheinen einer neuen Zeilennummer betätigt, ohne zuvor etwas in die Zeile hineinzuschreiben.

Siehe auch: RENUM

BEEP

Format: **BEEP <Anzahl>[,[<Ton>][,<Dauer>]]**

BEEP erzeugt eine Anzahl von Tönen spezifizierte Tonhöhe und Dauer. ACHTUNG BEIM G850 NICHT ANGESCHLOSSEN.

<Anzahl> bestimmt, wie oft der standardmäßige oder aber näher spezifizierte Ton vom Computer erzeugt werden soll. Die zu- lässigen Werte sind: 0 ... 65535.

<Ton>legt die Frequenz des Tones fest und darf durch einen Integer-Wert von 0 bis 255

vertreten sein. Es lassen sich Frequenzen von 230 Hz bis 7 kHz erzeugen, wobei Frequenz und Wert des Parameters in einem umgekehrt "proportionalen" Verhältnis stehen. Je höher der Wert, desto niedriger die Frequenz:

0 bedeutet ca. 7 kHz
255 bedeutet ca. 230 Hz

Fehlt dieser Parameter, wird als Standard eine Frequenz von 4 kHz geliefert.

<Dauer> bestimmt die Dauer eines Tones. Sie ist abhängig von der Frequenz, also dem Wert von <Tonhöhe>. Je tiefer die Frequenz, umso länger die Dauer. Der Wert darf im Bereich 0...65279 liegen und wird bei fehlendem Parameter zu 160 angenommen.

BLOAD

Format: **BLOAD**

BLOAD lädt ein Basic-Programm von einem anderen Sharp PC-G850 über das serielle Interface (11-Pin) in den Speicher. Dazu muss zeitgleich auf dem 2. Sharp der BSAVE eingegeben werden.

Achtung: Diese Übertragung verwendet ein internes Protokoll und ist daher nicht geeignet um Daten zwischen dem Sharp PC-G850 und einem PC auszutauschen. Genauso werden die Parameter für das serielle Interface im TEXT-Modus unter SIO-Format ignoriert.

Siehe auch: BSAVE

BLOAD M

Format: **BLOAD M [<beginnadresse>]**

BLOAD lädt Binärcode von einem anderen Sharp PC-G850 über das serielle Interface (11-Pin) in den Speicher. Es muss gleichzeitig am zweiten Sharp der BSAVE M-Befehl einegeben werden.

<beginnadresse> überschreibt die im Programm vorgesehene Adresse.

Achtung: Diese Übertragung verwendet ein internes Protokoll und ist daher nicht geeignet um Daten zwischen dem Sharp PC-G850 und einem PC auszutauschen. Genauso werden die Parameter für das serielle Interface im TEXT-Modus unter SIO-Format ignoriert.

Siehe auch: BSAVE M

BLOAD ?

Format: **BLOAD ?**

BLOAD ? liest ein Programm eines anderen Sharp PC-G850 vom seriellen Interface und vergleicht es mit dem im Speicher vorhandenen Programm.

Achtung: Diese Übertragung verwendet ein internes Protokoll und ist daher nicht geeignet um Daten zwischen dem Sharp PC-G850 und einem PC auszutauschen. Genauso werden die Parameter für das serielle Interface im TEXT-Modus unter SIO-Format ignoriert.

Siehe auch: BLOAD

BSAVE

Format: **BSAVE**

BSAVE überträgt ein Basic-Programm über das serielle Interface (11-Pin) auf einen anderen Sharp PC-G850. Auf diesem zweiten Sharp muss vorher der BLOAD Befehl eingegeben werden.

Achtung: Diese Übertragung verwendet ein internes Protokoll und ist daher nicht geeignet um Daten zwischen dem Sharp PC-G850 und einem PC auszutauschen. Genauso werden die Parameter für das serielle Interface im TEXT-Modus unter SIO-Format ignoriert.

Siehe auch: BLOAD

BSAVE M

Format: **BSAVE M [<beginnadresse>,<endadresse>[,<startadresse>]]**

Überträgt Binärcode über das serielle Interface auf einen zweiten Sharp PC-G850. Auf diesem zweiten Sharp muss vorher der BLOAD-M-Befehl eingegeben werden.

Die Sicherung beginnt bei <beginnadresse> und endet bei <endadresse>. Optional kann noch die startadresse angegeben werden.

Achtung: Diese Übertragung verwendet ein internes Protokoll und ist daher nicht geeignet um Daten zwischen dem Sharp PC-G850 und einem PC auszutauschen. Genauso werden die Parameter für das serielle Interface im TEXT-Modus unter SIO-Format ignoriert.

Siehe auch: BLOAD M

CALL

Format: **CALL [#<Bank>,]<Adresse>**

Mit CALL kann von einem BASIC-Programm oder der RUN-Ebene aus, ein Maschinensprache-Programm gestartet und anschließend in die aufrufende Ebene zurückgekehrt werden.

<Bank> bestimmt die Speicherbank aus dem Bereich 0...7, in der das Maschinenspracheprogramm gespeichert ist. Wird dieser Parameter nicht angegeben, gilt Speicherbank 0.

<Adresse> nennt die Anfangsadresse innerhalb der gültigen Speicherbank, bei der das Programm beginnt. Es sind die Adressen von 0...65535 (&0...&FFFF) zulässig. Die Adressangabe muss erfolgen und darf nicht weggelassen werden.

Siehe auch: POKE

CHR$

Format: **CHR$(<integer-zahl>)**

Die Funktion CHR$ liefert das Zeichen, dessen zugehöriger ASCII-Code als Funktionsargument angegeben ist.
Das betreffende Argument kann entweder eine Konstante, eine Variable oder ein Ausdruck sein. Der numerische Wert dieser Größen muss aber in jedem Falle vom Typ Integer sein.

Mit dieser Funktion können Steuerzeichen, die nicht über die Tastatur zugänglich sind, an die verschiedensten Peripheriegeräte wie Drucker usw. oder aber an die seriellen Schnittstellen weitergeleitet werden.

In welcher Zuordnung die Zeichen zu den ASCII- Codes stehen, können Sie aus Anhang H: Tabelle der Zeichencodes auf Seite 284 ersehen.

```
10:FOR X= 33 TO 126
20:PAUSE CHR$(X);
```

```
30 NEXT X
40:END
```

Dieses Programm zeigt alle darstellbaren Zeichen des standardmäßigen ASCII-Bereiches (Codes: &21 bis &7E) auf dem Display an. Das erste Zeichen ist hierbei das Ausrufezeichen !, das letzte die sogenannte Tilde ~.

Siehe auch: CHR$

CIRCLE

Format:
CIRCLE (<x>,<y>),<radius>[,[startwinkel],[endwinkel],[Verhältnis][,S|R|X],[Muster]]]

Der Befehl CIRCLE kann zur Darstellung von Kreisen, Kreisbögen, Sektoren und Ellipsen mit durchgehender Linie verwendet werden.

x	X-Koordinate des Bildschirms für den Kreismittelpunkt 0 (links) - 143 (rechts), Angaben von -32768 bis 32767 sind erlaubt.
y	y-Koordinate des Bildschirms für den Kreismittelpunkt 0 (oben) - 47 (unten), Angaben von -32768 bis 32767 sind erlaubt.
Radius	Radius des Kreises (1 - 32767)
startwinkel	Beginn des Kreisbogens in Grad (0-360), Standard ist 0 (Voller Kreis)
endwinkel	Ende des Kreisbogens in Grad (0-360), Standard ist 360 (Voller Kreis)
Verhältnis	Gibt das Verhältnis für die Ellipse an Verhältnis=ry/rx. Standard ist 1 (Kreis)
S|R|X	S:Setzt die Pixel des Kreises(Standard), R: Löscht die Pixel des Kreises, X:Invertiert die Pixel des Kreises
Muster	Muster für das Füllen des Kreises 0-6, 0:ohne Füllung (Standard), 1:waagerechte Linien, 2:=senkrechte Linien, 3:Linie nach rechts oben, 4:Linie nach rechts unten, 5:3+4, 6:voller Kreis

Beispiele:

CIRCLE (71,23),20	Einfacher Kreis mit Radius 20
CIRCLE (71,23),20,,,0.5,,2	abgeflachter Kreis mit senkrechter Füllung
CIRCLE (71,23),20,-45,-135	Kreisbogen von -45 nach -135

Siehe auch: LINE

CLEAR

Format: **CLEAR**

CLEAR löscht sämtliche im Speicher befindliche Variablen. Dies gilt auch für die reservierten, sprich Standard-Variablen.

Die numerischen Standardvariablen A bis Z bzw. @(1) bis @(26) werden dabei mit dem Wert 0 belegt und den Stringvariablen A$ bis Z$ bzw. @$(1) bis @$(26) ein Nullstring (ASCII-Code 0) zugewiesen.

Man kann das CLEAR-Kommando auch innerhalb eines Programmes verwenden. In jedem Falle 1äßt sich mit ihm Speicherplatz wiedergewinnen, der durch die Erzeugung und Belegung von Variablen für das eigentliche Programm verlorengegangen ist. Zum Beispiel mögen im ersten Teil eines Programmes so viele Variablen verwendet worden sein, dass kein freier Speicherplatz mehr übrig bleibt. Braucht man diese Variablen im zweiten Programmteil nicht mehr, so kann mit CLEAR wieder Platz für neue Variablen geschaffen werden.

BEISPIEL :

```
5:WAIT 30
10:DIM C(5)
20:FOR N= 1 TO 5
30:READ A:LET C(N)=A
40:PRINT C(N)
50:NEXT N
'Setzt Wartezeit für PRINT
'Dimensioniert Array C(N)
'Diese Zeilen lesen die
'DATA-Werte ein und
'zeigen sie an.
60:DATA 10,20,30,40,50'Stellt die Daten bereit 70:CLEAR 'Löscht alle Variablen
80:PRINT A 'Beweise Löschung   90:END
```

Prüfen Sie nach Ablauf des Programmes, ob auch das Array gelöscht worden ist, indem Sie eine Zuweisung versuchen, z.B.: C(2)=99. Existiert das Array nicht, erscheint die Meldung: ERROR 6.

Siehe auch: DIM, NEW, ERASE

CLOSE

Format: **CLOSE [#Dateinummer[,#Dateinummer,….]]**

CLOSE schließt alle spezifizierten Dateien.
Mit CLOSE wird die Möglichkeit des Zugriffs auf Dateien beendet, d.h. diese

geschlossen.

Ohne Parameterangabe schließt CLOSE alle offenen Dateien. Mit Angabe der Parameter <Dateinummer> werden nur die Daten geschlossen, die unter der jeweils gleichen Nummer zuvor mit OPEN zu einem bestimmten Zweck geöffnet worden sind.

Eine einmal geöffnete Datei muss, bevor sie für einen anderen Zugriffszweck (Eingabe, Ausgabe, Datenanhang) geöffnet werden kann, zuvor mit CLOSE geschlossen werden.

Versucht man, eine bereits geöffnete Datei zu öffnen, wird ein ERROR-Code ausgegeben.

Alle geöffneten Dateien werden automatisch bei Ausführung der Befehle END, NEW, RUN und LOAD und bei Ausschaltung des Computers geschlossen. Eine Dateischließung erfolgt auch dann, wenn man ein Programm editiert.

```
10:OPEN "E:PAYMENT" FOR INPUT AS #1
20:OPEN "E:UPDATE" FOR INPUT AS #2
:
:
400:CLOSE #1,#2
```

Siehe auch: END, OPEN

CLS

Format: **CLS**

CLS löscht das Display.
Das Kommando CLS löscht den Inhalt sämtlicher Display-Zeilen und setzt den Cursor an die linke obere Display-Ecke. Diese "Home-Position" trägt den Koordinatenpunkt (0,0).

CONT

Format : **CONT**

Das Kommando CONT setzt abgebrochene oder unterbrochene Programmabläufe fort.

Eine solche Fortsetzung ist nur bei folgenden Abbruch oder Unterbrechungsursachen möglich:
- Abbruch durch STOP-Anweisung
- Abbruch durch Betätigung der Taste BREAK
- Unterbrechung durch PRINT-Anweisung

In diesen Fällen wird CONT jedoch ignoriert:
- Programm abgearbeitet oder durch END beendet
- Programm im PRO-Modus geändert
- Programm mit ERROR-Code abgebrochen

Anstelle des CONT-Kommandos lässt sich auch eine GOTO-Anweisung mit spezifizierter Zeilennummer verwenden. Noch einfacher geht es aber, wenn man die Taste ⇓ betätigt.

```
10:PRINT "PROGRAMM STOPPT HIER"
20:STOP
30:PRINT "PROGRAMM FORTGESETZT"
40:PRINT "PROGRAMM BEENDET
50:END
RUN
PROGRAMM STOPPT HIER
BREAK IN 20
>
CONT
PROGRAMM FORTGESETZT
```

```
PROGRAMM BEENDET
>
```

COS

Format: **COS(<zahl>)**

Die Funktion COS liefert den Cosinus-Wert eines angegebenen Winkelargumentes.
Der durch einen numerischen Ausdruck vertretene Winkel kann entweder in Altgrad, Bogenmaß oder Neugrad vorliegen. Damit der Computer den dazugehörigen Funktionswert liefern kann, muss er im passenden Winkelmodus betrieben werden. Hierbei gilt:

Winkelmaß	Winkelmodus
Altgrad	DEGREE
Bogenmaß	RADIAN
Neugrad	GRAD

```
1O:DEGREE
20:G$=CHR$ (&F8)
30:PRINT "cos(60";G$;") = ";COS(60)
40:PRINT "cos(90";G$;") = ";COS(90)
50:END
>RUN
cos(60°) = 0.5
cos(90°) = 0
>
```

Siehe auch: ACS, SIN, TAN

CUB

Format: **CUB(<zahl>)**

Der Befehl CUB liefert einen zum Argument zahl gehörenden Kubikwert

Siehe auch: CUB

CUR

Format: **CUR(<zahl>)**

Der Befehl CUB liefert die zum Argument zahl gehörende Kubikwurzel

Siehe auch: CUB

DATA

Format: **DATA Liste der Werte**

DATA dient zur Auflistung von numerischen oder String-Konstanten, die mit der READ-Anweisung gelesen werden können.

Eine DATA-Anweisung kann gleichzeitig numerische und String-Konstante in gemischter Aufzählung enthalten. Die Konstanten müssen jeweils durch ein Komma voneinander getrennt sein.

Mit jeder neuen READ-Anweisung lässt sich immer eine Konstante nach der anderen aus dieser Liste ablesen. Damit das Zusammenspiel zwischen READ und DATA funktioniert, muss die gerade zu lesende Konstante vom selben Typ sein wie die in der READ-Anweisung angegebene Variable.

Welches Element der Liste gerade lesbar ist, wird durch einen internen Zeiger bestimmt, der automatisch nach jedem Lesevorgang entsprechend um eine Position weiter gesetzt wird.

Sind alle in der DATA-Liste enthaltenen Elemente gelesen, kann keine weitere READ-Anweisung ausgeführt werden, bevor nicht eine Rücksetzung des internen Zeigers mit Hilfe des RESTORE-Befehles erfolgt.

DATA gehört wie REM zu den sogenannten nichtausführbaren Anweisungen, was bedeuten soll, dass bei einem Sprung auf eine solche Anweisung der Computer nach der nächsten Anweisung sucht, die nicht mit dem Befehlswort DATA beginnt und dort den weiteren Programmablauf fortsetzt.

DATA-Anweisungen können an beliebiger Stelle im BASIC-Programm vorkommen. Sie brauchen nicht so platziert zu werden, dass sie vor dem READ-Befehl, der ihre Inhalte liest, stehen. Der Computer sucht sich nämlich die nächste auffindbare und noch nicht abgelesene DATA-Anweisung selbst.

```
10:DATA "MICHAEL",23
20:FOR J=1 TO 5
30:READ A$,B
40:PAUSE A$,B
50:NEXT J
```

60:END 70:DATA "SUSANNE",-24,"NICOLE",38," PETER",57

Dieses Programmbeispiel wird mit Ausgabe eines ERROR-Codes beendet, da nach 4 Durchläufen der durch die Zeilen 20 bis 50 gebildeten Programmschleife die nächste READ-Anweisung keine les- baren Daten mehr vorfindet. Die Programmschleife möchte zwar gerne fünf Datenpaare A$,B lesen, es stehen in den DATA-Anweisungen des Programmes jedoch nur vier solche Datenpaare bereit.

```
10:FOR I=1 TO 5
20:READ N
30:PAUSE N
40:NEXT I
50:END
60:DATA 10,2*I,I+N,4,ACS(I/10)
```

Siehe auch: READ, RESTORE

DEG

Format: **DEG(<zahl>)**

Wandelt einen Winkel, der in Altgrad gemessen wird, von der sexagesimalen Form, also der Darstellung in Stunden, Minuten und Sekunden, in seine dezimale Entsprechung um. Der zu wandelnde Winkel muss dabei im Format hh.mmssrr vorliegen, wobei die Ziffern:

hh die Stunden,
mm die Minuten,
ss die Sekunden und
rr den dezimalen Sekundenrest bestimmen.

Folgende Werte sind dabei einzuhalten:

hh : 0 bis ..
mm : 00 bis 59
ss : 00 bis 59
rr : 00 bis 99

Das Ergebnis wird mit bis zu zehn signifikanten Ziffern angezeigt.

BEISPIEL :

```
10:X=DEG 50.3000
20:PRINT X
30.END
> RUN
```

50.5

>

Siehe auch: DMS

DEGREE

Format: **DEGREE**

Versetzt den Computer in den Winkelmodus DEGREE. In diesem Modus werden alle Winkelangaben als in Altgrad gegeben angesehen und auch in diesem Maß ausgegeben.

Zur Kennzeichnung dieser Betriebsart wird in der Status-Zeile das Symbol DEG angezeigt.

Die Argumente der Funktionen SIN, COS und TAN werden dann als in Altgrad vorliegend angesehen und die Werte der Funktionen ASN, ACS und ATN in dezimalen Altgraden ausgegeben.

```
1O:DEGREE
20:PAUSE "WINKELANGABEN IN ALTGRAD"
30:PRINT ASN(0.5),ASN(1)
40:PRINT ACS(0.5),ASN(1)
50:PRINT ATN(0.5),ATN(1)
60:END
```

Lassen Sie dieses Programm zum Vergleich auch in den beiden anderen Winkel-Modi (GRAD und RADIAN) laufen.

Siehe auch: RADIAN, GRAD

DELETE

Format: **DELETE [Zeilennummer][-][Zeilenummer]**

DELETE löscht die spezifizierten Zeilen eines BASIC-Programmes.

DELETE <Zeilennummer>

löscht genau diese spezifizierte Zeile, sofern sie im Programm vorhanden ist.

DELETE <Zeilennummer>-

löscht, mit der genannten Zeile beginnend, alle weiteren Programm-Zeilen bis zum Programm-Ende.

DELETE <Zeilennummer>-<Zeilennummer>

löscht alle Zeilen eines Programmes, beginnend mit der ersten und endend mit der zweiten angegebenen Zeile. Die zweite Zeilennummer muss dabei höherwertiger a1s die erstgenannte Nummer sein.

DELETE -<Zeilennummer>

löscht alle Zeilen eines Programmes, beginnend mit der ersten Programmzeile bis einschließlich der spezifizierten Zeile.

Um ein Programm komplett zu löschen, sollte das Kommando NEW verwendet werden.

DELETE 150
DELETE 200-
DELETE 50-150
DELETE -35

Siehe auch: NEW, RENUM

DIM

Format: **DIM variable[$](index1[,index2]][*stringlänge][,……]**

DIM dient der Reservierung von Speicherplatz für die Aufnahme von Array-Variablen des numerischen oder des String-Typs. Daneben bestimmt DIM auch die von einem Array erfassbare Elemente-Anzahl. Bei String-Arrays läßt sich zusätzlich die Länge der aufzunehmenden Strings definieren.

Mit Ausnahme der Standard-Variablen A bis Z und A$ bis Z$, die gleichbedeutend mit den beiden eindimensionalen Standard-Arrays @(1) bis @(26) und @$(1) bis @$(26) sind, müssen alle anderen Array-Variablen mit DIM dimensioniert werden, um für diese ausreichend Platz im Speicher bereit- zustellen.

Solange diese Dimensionierung fehlt, können die nicht zu den Standard-Variablen zählenden Arrays nicht benutzt werden.

<index1 bestimmt bei eindimensionalen Arrays die Anzahl der Elemente, die in einem solchen Array, dass auch als Liste betrachtet werden kann, aufnehmbar sind. Die zulässigen Werte für <Größe> liegen im Bereich von 0...255, wobei sich die Elemente-Anzahl aus <Größe> + 1 ergibt.

<index2> bestimmt bei zweidimensionalen Arrays, die man auch als Tabellen auffassen

kann, die Nummer der höchsten vorkommenden Zeile einer solchen Tabelle. Die zulässigen Werte für <Zeile> liegen ebenfalls im Bereich: 0...255.

<stringlänge> bestimmt bei den String-Arrays wie lang die aufzunehmenden Strings sein dürfen. Weisen die Strings jedoch mehr Zeichen auf als mit <Länge> vorgegeben, werden sie auf das entsprechende Maß reduziert und alle überzähligen Zeichen gekappt. Fehlt die Angabe des Parameters <Länge>, können die Strings standardmäßig bis zu 16 Zeichen enthalten. Die maximale Stringlänge beträgt 255 Zeichen.

Die Gesamtheit der Tabellenelemente ergibt sich wegen des Nummerierungsbeginns bei Null aus dem Zusammenhang:

Elementanzahl = (<Zeile> + 1) * (<Spalte> + 1)

Nachdem ein Array DIMensioniert worden ist, kann man es nicht umdimensionieren, solange nicht ein Reset des Computers oder aber einer der Befehle CLEAR, NEW, RUN oder ERASE ausgeführt wird. Ein laufendes Programm bricht mit der Ausgabe eines ERROR-Codes ab, wenn es entweder auf ein nicht mit DIM deklariertes Array trifft oder eine DIM-Anweisung vorfindet, die sich auf ein bereits dimensioniertes Array bezieht. Indizes, die mit <Größe>, <Spalte> oder <Zeile> vereinbarten Maximalwerte überschreiten, führen ebenfalls zu einem Programmabbruch. Negative Indizes sind nicht erlaubt!

```
10:DIM C(13)
20:DIM F$(10)
30:DIM H(4,6)
40:DIM G$(7,5)*25
```

Siehe auch: CLEAR, ERASE, READ

DMS

Format: **DMS(<zahl>)**

DMS wandelt einen in Altgrad vorliegenden Winkel von der dezimalen Darstellungsart in das Format "Stunden,Minuten,Sekunden" um, das man auch als sexagesimale Notation bezeichnet.
Das Wandlungsergebnis wird im Format hh.mmssrr ausgegeben, wobei folgendes gilt:

hh	Stunden	00.....
mm	Minuten	00...59
ss	Sekunden	00...59
rr	Dezimaler Sekundenrest	00...99

```
10:X=DMS 50.5
20:PRINT X
30:END
```

DMS$

Format: **DMS$(<zahl>)**

DMS wandelt einen in Altgrad vorliegenden Winkel von der dezimalen Darstellungsart in das Format "Stunden,Minuten,Sekunden" um, das man auch als sexagesimale Notation bezeichnet.
Das Wandlungsergebnis wird im String-Format hh˚ mm' [ss[.rr]]" ausgegeben, wobei folgendes gilt:

hh	Stunden	00.....
mm	Minuten	00...59
ss	Sekunden	00...59
rr	Dezimaler Sekundenrest	00...99

```
10:X=DMS$(50.5)
20:PRINT X
30:END
> RUN

50˚30'
```

END

Format: **END**

END beendet ein laufendes Programm und schließt alle offenen Dateien und Schnittstellen.

Es ist nicht zwingend erforderlich, ein Programm mit einer END-Anweisung in der letzten Programmzeile abzuschließen. Bei speziellen Strukturen, die sich bei der Verwendung von Unter-Routinen ergeben, oder in Fällen, wo mehrere voneinander unabhängige Programme in den Speicher zu laden sind, stellt die Einfügung von END-Anweisungen sicher, dass der Computer nicht aus Versehen in unerlaubte Programmteile verzweigt.

Fehlt die END-Anweisung, endet das Programm mit Ausführung der letzten Programmzeile.

```
10: GOSUB 50
```

```
20:PRINT "NACH WIEDERKEHR ENDET DAS"
30:PRINT "HAUPTPROGRAMM MIT ZEILE 40"
40:END
50:PRINT "HIER IST DAS UNTERPROGRAMM"
60:RETURN
>
RUN
HIER IST DAS UNTERPROGRAMM
NACH WIEDERKEHR ENDET DAS
HAUPTPROGRAMM MIT ZEILE 40
>
```

Siehe auch STOP

EOF

Format: **EOF(<dateinummer>)**

EOF liefert einen Wert, aus dem ersichtlich ist, ob beim Lesen einer sequentiellen Datei deren Ende erreicht worden ist.

Der Parameter **<Dateinummer>** sorgt für die Anwahl der richtigen Datei und muss mit der Nummer übereinstimmen, unter der die Datei geöffnet wurde.

Die möglichen gelieferten Werte sind 0 oder -1. Sie haben folgende Bedeutung:

0 Dateiende noch nicht erreicht -1 Dateiende erreicht

ERASE

Format: **ERASE <variable>|<array>[,...]**

ERASE löscht einfache Variablen und Arrays.
Hierbei sind nur solche numerischen Variablen und String-Variablen löschbar, die nicht zu den Standard-Variablen A bis Z bzw. @(1) bis @(26) und A$ bis Z$ bzw. @$(1) bis @$(26) gehören.

Mit der ERASE-Anweisung lassen sich zwar gezielt einfache Variablen und String-Variablen löschen, jedoch nicht individuelle Elemente eines Arrays. Arrays sind nur komplett löschbar und müssen mit zwei nachfolgenden, aber leeren Klammern kenntlich gemacht sein.

Die zu löschenden Variablen können in Form einer Parameterliste an das Befehlswort

ERASE angefügt werden, wobei diese durch Kommas voneinander zu trennen sind.

10:ERASE AB,Z$()

Siehe auch: CLEAR

EXP

Format: **EXP(<zahl>)**

Die Funktion EXP(zahl) liefert zu einem Argument zahl die zur Basis der Zahl e gebildete Potenz.

Das Argument zahl kann eine numerische Konstante oder Variable oder aber ein numerischer Ausdruck sein. Dabei muß X im Wertebereich: -227.9559242 bis +230.2585092 liegen. Werte außerhalb dieses Bereiches liefern den Funktionswert 0.

Die transzendente Basiszahl e wird intern mit 2.7181828 angenähert.

PRINT EXP(1O)
>
220026.46579

FACT

Format: **FACT(<zahl>)**

Zeigt den Faktor von zahl an.

PRINT FACT(7)
>
5040

FILES

Format: **FILES**

FILES liefert eine Liste der auf der RAM-Disk (Disk E) befindlichen Dateien, also ein Inhaltsverzeichnis.

Das Inhaltsverzeichnis zeigt jede Datei unter Nennung folgender Einzelinformationen an:

-Dateiname
-Extension (z.B.: .BAS für BASIC-Programme, TXT für Assembler, C, CASL)
-Dateilänge

Siehe auch: LFILES

FIX

Format: **FIX(<zahl>)**

Die Funktionen FIX und INT entfernen beide die Nachkommastellen von zahl und geben die daraus resultierende ganze Zahl zurück.
Der Unterschied zwischen INT und FIX ist folgender: Wenn zahl negativ ist, gibt INT die erste negative ganze Zahl zurück, die kleiner oder gleich zahl ist, während FIX die erste negative ganze Zahl zurückgibt, die größer oder gleich zahl ist. INT konvertiert beispielsweise –8,4 in –9, während Fix –8,4 in –8 konvertiert.

PRINT FIX(-8.4)
>
-8

FOR .. NEXT

Format:

FOR <variable>=<startausdruck> TO <endausdruck> [STEP <erhöhungsausdruck>]

NEXT <variable>

Mit FOR und NEXT lassen sich Programmschleifen bilden und damit Anweisungen mehrfach in vorbestimmter Weise (determiniert) ausführen.

Der erste Parameter ist eine numerische <Variable> und bestimmt, welche Variable als Schleifenzähler (Laufvariable) dienen soll.

Der zweite Parameter <startausdruck> weist dieser Laufvariablen einen Anfangswert zu, der durch jeden beliebigen num. Ausdruck direkt oder in vorm einer Variablen gebildet sein kann.

Der dritte Parameter <endausdruck> gibt den Endwert der Laufvariablen an. Ist er über- oder unterschritten, wird die Programmschleife verlassen und mit der Anweisung nach dem NEXT-Befehl fortgefahren. Dieser Parameter kann ebenfalls ein beliebiger num.

Ausdruck (auch Übergabe in einer Variablen) sein.

Der vierte Parameter <erhöhungsausdruck> ist optional und gibt die Schrittweite, also den Betrag an, der nach jedem Schleifendurchlauf zur Laufvariable hinzugezählt wird (unter Berücksichtigung des Vorzeichens). Fehlt die Angabe des STEP-Wertes, gilt für ihn standardmäßig 1.

Trifft der Computer auf einen NEXT-Befehl, sucht er sich diejenige zuvor stehende FOR-Anweisung, die mit derselben Laufvariablen behaftet ist, wie sie in der NEXT-Anweisung spezifiziert ist. Dann addiert er die Schrittweite zum derzeitigen Wert der Laufvariablen und überprüft, ob ihr Endwert "überschritten" wird. Bei positiver Schrittweite überprüft er, ob der neue Variablenwert größer als der Endwert ist und bei negativen, ob dieser kleiner als der Endwert ist. Trifft dieses zu, wird die Programmschleife verlassen. Im anderen Falle werden die in der Schleife aufgeführten Anweisungen erneut ausgeführt: diesmal mit dem neuen Wert der Laufvariablen.
Damit der Computer die richtige FOR-Anweisung finden kann, muss in der NEXT-Anweisung die zugehörige Laufvariable als Parameter angegeben werden.

```
10:FOR I=1 TO 20
50:NEXT I
230:FOR K=2 TO 17 STEP 2
290:NEXT K
```

FOR-NEXT-Schleifen lassen sich auch ineinander verschachteln:

```
10:FOR M=1 TO 10
20:FOR N=5 TO 20 STEP 5
80:NEXT N
90:NEXT M
```

```
10:A=2:B=5
20:FOR I=A TO B STEP 0.2
30 NEXT A
```

FOR-NEXT-Anweisungen treten immer paarweise auf. Zu jeder FOR-Anweisung muss in logischer Folge eine passende NEXT-Anweisung vorzufinden sein. Ist diese Bedingung nicht gegeben, weil ein Teil des Paares fehlt oder aber durch eine verkehrte Schachtelung vorliegt, stoppt der Computer das Programm und gibt einen ERROR-Code aus.

FRE

Format: **FRE**

FRE zeigt den freien Speicher in Bytes an. FRE kann als Funktion den Wert z.B. an eine Variable übergeben.

GCURSOR

Format: **GCURSOR (<x>,<y>)**

GCURSOR positioniert den Grafik-Cursor auf dem gewünschten Display-Punkt.

Im Grafik-Modus lässt sich das Display als Matrix mit 144 x 48 einzeln adressierbaren Punkten ansprechen. Der Grafik-Cursor kann hierbei nicht nur auf eine innerhalb dieser Matrix liegende Koordinate positioniert werden, sondern auch außerhalb davon. Soll der Cursor im sichtbaren Bereich liegen, so sind folgende Koordinaten einzuhalten:

<X-Koordinate> : 0…143
<Y-Koordinate> : 0…47

Beide Koordinaten können jedoch auch Werte im Bereich von -32768 bis 32767 annehmen.

GOSUB ... RETURN

Format: **GOSUB <zeilennummer>|"<label>"|*label**
RETURN

GOSUB ruft das durch eine <Zeilennummer> oder eine <Label> spezifizierte Unterprogramm auf.

Ein Unterprogramm (subroutine) ist eine Gruppe von aufeinanderfolgenden Programmzeilen, die im Ablauf des Gesamtprogrammes mehrfach benötigt werden. Jedes Unterprogramm muss mit dem Befehl RETURN abgeschlossen sein, damit eine Rückkehr in das aufrufende Hauptprogramm möglich ist und der normale Programmablauf hinter der Anweisung GOSUB fortgesetzt werden kann.

Ein Unterprogramm lässt sich beliebig oft von diversen GOSUB-Anweisungen aufrufen, wobei es selbst wiederum andere Unterprogramme aufrufen kann und so fort. Somit entstehen verschachtelte Unterprogramme mit einer maximalen Tiefe von 10 Ebenen. Ist diese Verschachtelungstiefe größer, wird das laufende Programm abgebrochen und ein ERROR-Code 50 auf dem Display angezeigt.

Unterprogramme können sich auch selbst aufrufen, wenn man das Programm entsprechend geschickt auslegt und dafür sorgt, daß das Unterprogramm verlassen werden kann und die zulässige Tiefe der Verschachtelung nicht überschritten wird.

```
 20:GOSUB 90

 20:GOSUB "A"
……

 90:"A" PRINT"UNTERPROGRAMM GESTARTET"
 95:RETURN
```

Siehe auch: GOTO, ON..GOTO, ON..GOSUB

GOTO

Format: **GOTO <zeilennummer>|"<label>"|*label**

GOTO verzweigt den weiteren Programmablauf zu der durch eine <Zeilennummer> oder einem <Label> spezifizierten Zeile.

Die GOTO-Anweisung führt einen nichtbedingten Sprung aus, d.h., der Sprung zur spezifizierten Zeile wird unweigerlich ausgeführt und hängt von keiner Bedingung ab (es sei denn man wählt die Anweisung IF..THEN GOTO).

Wird als Sprungziel eine Zeile bestimmt, die die nichtausführbaren Befehle DATA oder REM enthält, wird der Programmablauf mit der nächsten Zeile (bzw. ausführbaren Anweisung) fortgesetzt.

Im RUN-Modus lässt sich mit der GOTO-Anweisung auch ein Programm ab einer speziellen Zeile starten. Im Gegensatz zum Kommando RUN werden hierbei keine Variablen gelöscht.

Mit der Anweisung GOTO kann auch ein Programm, welches mit der BREAK-Taste unterbrochen worden ist, fortgesetzt werden. (Siehe auch: CONT).

```
10:INPUT A$
20:IF A$="J" THEN 40
30:PRINT "NEIN" : GOTO 50
40:PRINT "JA"
```

50:END

Siehe auch: GOSUB, ON..GOTO

GPRINT

Format: **GPRINT <bit-muster>[;bit-muster…….]**
GPRINT <string>

GPRINT zeichnet Grafik-Muster auf dem Display.
Mit GPRINT können jeweils acht direkt übereinander liegende Display-Punkte durch geeignete Bit-Muster beeinflusst und somit Grafiken auf dem Display abgebildet werden. Jede vertikale Punktreihe hat dabei die Höhe eines im Text-Modus dargestellten Zeichens.

Die Bit-Muster können in dezimaler oder aber hexadezimaler Aufzählung angegeben werden, wobei sie jeweils durch ein Semikolon voneinander zu trennen sind. Hexadezimale Werte sind dabei wie üblich durch ein vorangestelltes & zu kennzeichnen, Des weiteren lassen sich diese Muster auch mit einem <Hex-String> vereinbaren, wobei jedes Byte zwischen 00 und FF stets mit zwei Ziffern ohne Verwendung des & darzustellen ist. Der Verbund der so aufgeführten Bytes ist dann in Anführungsstriche einzuschließen. Wird innerhalb eines solchen Strings eine Ziffer vergessen und damit seine Zeichenanzahl ungerade, so interpretiert der Computer alle Zweiergruppen an Ziffern als gültiges Byte und lässt die letzte verbleibende Ziffer unbeachtet.

Nachstehende Anweisungen sind in ihrer Wirkung identisch:

GPRINT 16;40;18;253;18;40;16 (dezimal) GPRINT &10;&28;&12;&FD;&12;&28;&10 (hexadezimal) GPRINT "102812FD122810" (Hex-String)

GPRINT ohne jegliche Parameterangabe setzt den Grafik-Cursor um eine Punktzeile nach unten ohne dabei den auf dem Display befindlichen Inhalt zu beeinflussen. Schließt eine GPRINT-Anweisung mit einem Semikolon ab, so bleibt der Grafik-Cursor an der letzten Position stehen. Die nächste GPRINT-Anweisung setzt sich dann an dieser Stelle fort. Wählt man ein Komma als Trennzeichen, so wird zwischen die so getrennten Punktmuster eine Lücke von einer Punktbreite gesetzt.

Siehe auch: GCURSOR

GRAD

Format: **GRAD**

Versetzt den Computer in den Winkelmodus GRAD. In diesem Modus werden alle Winkelangaben als in Neugrad gegeben angesehen und auch in diesem Maß

ausgegeben. Zur Kennzeichnung dieser Betriebsart wird in der Status-Zeile das Symbol GRAD angezeigt.

Alle Argumente der Funktionen SIN, COS und TAN werden als in Neugrad vorliegend angesehen und die Werte der Umkehrfunktionen ASN, ACS und ATN im Winkelmaß Neugrad geliefert.

```
1O:GRAD
20:PAUSE "WINKEL IN NEUGRAD"
30:PRINT ASN(0.5),ASN(1.O)
40:PRINT ACS(0.5),ACS(1.O)
50:PRINT ATN(0.5),ATN(1.O)
60:END
```

Siehe auch: DEGREE, RADIAN

HCS

Format: **HCS(<zahl>)**

Der Befehl HCS liefert einen zum Argument zahl gehörenden Wert der hyperbolischen Cosinus.

HEX$

Format: **HEX$(<zahl>)**

Die Funktion HEX$ liefert einen String, der sich aus den Hex-Ziffern des als Hexadezimalwert aufgefassten Argumentes zusammensetzt.
Das Argument wird als ganzzahlige Dezimalzahl betrachtet.

HEX$(64) liefert den String: "&40"

```
10:PRINT "WANDLUNG: DEZIMAL ZU HEX"
20:INPUT " DEZIMAL-ZAHL = ";X
30:IF X>65535 THEN 100
40:IF X<0 THEN 110
50:PRINT "HEXADEZIMAL-WERT = ";HEX$(X):PRINT
60:INPUT "NOCH EINE ZAHL (J/N) ";A$
70:IF A$="J" THEN 20
80:IF A$="N" THEN END
90:GOTO 60
100:PRINT "FEHLER: MAXIMUM=65535 !":GOTO 20
110:PRINT "FEHLER: MINIMUM=0 !":GOTO 20
120:END
```

Siehe auch: VAL

IF .. THEN .. ELSE

Format:

IF <bedingung> THEN Zeilennummer|*label|<anweisung> [ELSE Zeilennummer|*label]

IF...THEN...ELSE entscheidet über den weiteren Programmverlauf, je nachdem, ob eine Bedingung erfüllt ist oder nicht.

Die Entscheidung hängt dabei von einer zwischen den Befehlswörtern IF...THEN zu überprüfenden **<Bedingung>** ab. Ist diese erfüllt, wird mit der hinter dem Befehlswort THEN angegebenen Zeile, die durch eine **<Zeilen-Nr.>** oder eine **<*label>** spezifiziert ist, oder den dortigen stehenden **<Anweisungen>** fortgefahren. Im anderen Falle wird die nächste Zeile ausgeführt.

Enthält die Anweisung IF...THEN den Zusatz ELSE, so wird bei nicht erfüllter Bedingung das Programm nicht mit der nächsten Zeile fortgesetzt, sondern mit der hinter ELSE spezifizierten Zeile oder den dortigen Anweisungen.

Folgt dem Befehlswort ELSE keine Zeilenspezifikation, werden alle Anweisungen (auch die durch einen Doppelpunkt voneinander getrennten) ausgeführt, solange diese sich in derselben Programmzeile befinden.

BEISPIEL :

Beispiele logischer Ausdrücke:
X=1

Bedingung ist erfüllt, wenn X den Wert 1 hat.

(P=2 AND Q=4) OR P=1

Die Bedingung ist erfüllt, wenn entweder P den Wert 1 besitzt (unabhängig von Q) oder aber P=2 und Q=4 gilt.

```
10:INPUT "SOLL ICH PIEPSEN ",A$
20:IF A$="N" THEN 60
30:IF A$="J" THEN BEEP 3:GOTO 10
40:PRINT "NUR J ODER N EINGEBEN !"
50:GOTO 10
60:PRINT "SCHADE !"
70:END
```

Die Anweisungen IF..THEN..ELSE können innerhalb einer Programmzeile auch ineinander verschachtelt werden.
Die zwischen den Befehlswörtern IF...THEN abzufragende **<Bedingung>** wird durch einen **logischen Ausdruck** gebildet, der sich, so komplex er auch immer sein mag, stets aus folgenden Grundformen aufbauen läßt:

X=Y	Vergleich auf Gleichheit
X<>Y	Vergleich auf Ungleichheit
X<Y	Vergleich, ob X kleiner als Y
X>Y	Vergleich, ob X größer als Y
X<=Y	Test, ob X kleiner oder gleich Y
X>=Y	Test, ob X größer oder gleich Y
X AND Y	Logische UND-Verknüpfung
X OR Y	Logische ODER-Verknüpfung
NOT X	Logische Verneinung

IF .. THEN .. ELSE .. ENDIF

Format: **IF <bedingung> THEN**
Anweisung
[ELSE
Anweisung]
ENDIF

IF...THEN...ELSE..ENDIF entscheidet über den weiteren Programmverlauf, je nachdem, ob eine Bedingung erfüllt ist oder nicht.

Wenn die Bedingung der IF-Anweisung zutrifft, wird die die Anweisung nach THEN ausgeführt. Danach wird die Anweisung nach ENDIF ausgeführt. Wenn die Bedingung nicht zutrifft, wird die Anweisung nach ELSE ausgeführt. Danach wird wieder die Anweisung ENDIF ausgeführt.

IF, ELSE, ENDIF müssen immer direkt nach einer Zeilennummer folgen, nicht nach einem Label

Eine Anweisung, ein Ausdruck oder eine Bemerkung sollte nicht in der gleichen Zeile nach THEN (oder ELSE) folgen. Ansonsten wird die Anweisung wie eine normale IF..THEN..ELSE-Anweisung behandelt.

Die Verwendung und Interpretation des Bedingungsausdrucks entspricht der IF..THEN..ELSE-Anweisung.

BEISPIEL :

```
 10 IF(4*A)<B OR (2*A)>B THEN
 20 PRINT "IMPOSSIBLE"
 30 ELSE
 40  C=B-INT(B/2)*2
 50  IF C=1 THEN
 60    PRINT "XX"
 70  ELSE
 80    X=(2*A)-B/2:Y=(B/2)-A
 90    PRINT X
100    PRINZ Y
110  ENDIF
120 ENDIF
```

INKEY$

Format: **INKEY$**

INKEY$ überprüft während der Programmausführung den Tastaturpuffer, ob zwischenzeitlich Zeichen über die Tastatur eingegeben worden sind, und liest ein einzelnes Zeichen in die spezifizierte Stringvariable ein.

Wurde zwischenzeitlich kein Zeichen über die Tastatur eingegeben, liefert INKEY$ einen Nullstring (ASCII-Code 0).

BEISPIEL :

```
300:A$=INKEY$
310:IF A$="" THEN 300
320:IF A$="*" THEN 500
330:GOTO 300
:
500:PRINT "HALLO"
```

Siehe auch: INPUT

INP

Format: **INP(<port>)**
INP

INP liefert ein Datenbyte vom spezifizierten Port des Z80-kompatiblen Mikroprozessors.

Der Wert <port> bestimmt den Eingabe-Port von dem ein Byte zu holen ist. Die Spezifikation Ports geschieht durch eine Adresse, also mit einem 16-Bit-Wert im Bereich 0....65535 bzw. &0-&FFFF

:
300:A=INP(20)

INP ohne Parameter ließt vom 11-Pin-Interface die Werte von XIN, DIN und ACK

Beispiel:

10:A=INP
20:PRINT A
> 3

0x4 + 1x2 + 1x1 = 3
XIN=Lo DIN=HI ACK=HI

Siehe auch: OUT

INPUT

Format: **INPUT Variable[,Variable……]**

INPUT "Meldung",variable[["Meldung"],variable……]

INPUT "Meldung";variable["Meldung";variable……]

INPUT erlaubt während eines laufenden Programms Variablen Werte zuweisen zu können.

Mit Ausführung einer INPUT-Anweisung wird der Programmablauf gestoppt und eine <Meldung> auf dem Display angezeigt, sofern eine solche in der Anweisung spezifiziert worden ist. Fehlt diese, erscheint an ihrer Stelle ein Fragezeichen.

Während dieser Pause im Programmablauf können Daten über die Tastatur eingegeben werden. Die entgegengenommenen Daten werden den a1s Parameter aufgelisteten <Variablen> der Reihe nach zugeordnet. Die Variablen sind in der Liste durch Kommas, die entsprechenden Tastatureingaben durch Betätigung der ENTER-Taste voneinander zu trennen.

Wird keine <Meldung> vereinbart, so erscheint in der Anzeige nur ein Fragezeichen, um auf eine notwendige Dateneingabe aufmerksam zu machen. Ist dagegen eine <Meldung> vereinbart, erfolgt nach dem Fragezeichen die Ausgabe dieser <Meldung>. Die Ausgabe des Fragezeichens lässt sich unterdrücken, wenn man dem Parameter <Meldung> ein Semikolon nachstellt.

In allen eben beschriebenen Fällen wird der Cursor hinter dem Fragezeichen bzw. der <Meldung> positioniert. Folgt der <Meldung> jedoch ein Komma, wird der Cursor auf den

Anfang der Zeile gesetzt, die sich oberhalb der Zeile befindet, in der die Meldung angezeigt wird.

```
10:INPUT A
20:INPUT "A=";A
30:INPUT "A=",A
40:INPUT "X=?";X,"Y=?";Y
```

INPUT#

Format : **INPUT#<Dateinummer>,Variable[,Variable…….]**

liest Datensätze aus einer sequentiellen Datei, die sich auf der RAM-Disk befinden oder vom seriellen Interface eingelesen werden.

Dateinummer ist die Nummer der Datei die ihr bei der Öffnung durch den OPEN-Befehl zugewiesen worden ist. Ein Versuch, eine ungeöffnete Datei zu lesen, endet mit der Ausgabe eines ERROR-Codes.
Für das Lesen vom seriellen Interface ist die Dateinummer 1. Für Dateien der RAM-Disc entweder 2 oder 3.

Die Liste der Variablen bestimmt die Namen der Variablen, in die die Daten der Datensätze einzulesen sind. Die Aufzählung der Variablen kann sowohl aus einfachen Variablen, Standardvariablen oder Arrays bestehen. Reihenfolge und Typ der gelieferten Daten müssen zu den bereitgestellten Variablen passen. Stringvariable sind dabei in ausreichender Länge zu dimensionieren. Arrays müssen in der Variablen-Liste mit dem Pseudo-Index (*) versehen sein, z.B.: A(*). Als Begrenzung beim Einlesen von Daten zu numerischen Variablen werden Komma, Leerstelle, LF, CF oder CR+LF verwendet. Als Begrenzung bei der Eingabe von String-Variablen werden Komma, CR, LF und CR+LF verwendet. Wenn die Daten motz einem doppelten Hochkomma (") beginnen werden alle Daten bis zum folgenden Hochkomma einer Variablen zugewiesen.

```
 10:A$="AB"+CHR$ 34+"CDE"+CHR$ 34
 20:B$=CHR$ 34 + "CD,EF"+CHR$ 34
 30:PRINT A$
 40 PRINT B$
 50:OPEN "E:ABC.DAT" FOR OUTPUT AS #2
 60:PRINT #2,A$;",";B$
 70:CLOSE 70
 80:OPEN "E:ABC.DAT" FOR INPUT AS #2
 90:INPUT #2,C$,D$
100:PRINT C$
110:PRINT D$
120:CLOSE:END
```

Siehe auch: DIM, INPUT, OPEN, PRINT

INT

Format: **INT(<zahl>)**

Die Funktion INT(zahl) schneidet vom Argument zahl alle Nachkommastellen ab und liefert lediglich dessen ganzzahligen Anteil (Integer-Wert).

Der Funktionswert ergibt sich aus der Abrundung des Argumentes auf den nächstniedrigeren ganzzahligen Wert.

Die Rundung ist völlig unabhängig davon, ob ein positiver oder negativer Wert als Argumentes angegeben ist. So werden positive Werte betragsmäßig kleiner, negative hingegen größer.

```
1O:PRINT INT(3.3)
20:PRINT INT(-3.3)
30:PRINT INT(.2)
```

KILL

Format: **Kill "<dateiname>[.BAS]"**

KILL löscht Basic-Programme, die sich auf der RAM-Disk befinden.

Der **<Dateiname>** bestimmt welche Datei zu löschen ist. Die Extension BAS muss nicht mit angegeben werden. Es kann auch nicht der Name der Ramdisk (E:) oder sonstiger Geräte angegeben werden.
Die Benutzung von mehrdeutigen Namen unter Verwendung sogenannter "wildcards" (* oder ?) ist nicht möglich.
Alle anderen Dateien werden über den TEXT-Monitor angelegt oder gelöscht.

Beispiel:
KILL "TEST"
Diese Anweisung löscht das Basic-Programm TEST von der RAM-Disk

Siehe auch: SAVE

LCOPY

Format: **LCOPY <startzeile>,<endzeile>,<zielzeile>**

LCOPY Kopiert die Basic-Zeilen von <startzeile> bis <endzeile> nach <zielzeile>. Dabei werden Sprungadressen in Basic-Befehlen nicht angepasst (im Gegensatz zu RENUM).

LEFT$

Format: **LEFT$(<zeichenkette>,<anzahl>)**

Die Funktion LEFT$ liefert den linksbündigen Teil einer vorgegebenen Zeichenkette, wobei bestimmt werden kann, wie viele Zeichen von rechts aus der Zeichenkette ausgelesen werden.

Die Anzahl der Zeichen des Teilstrings muss im Bereich von 0 bis 255 liegen. Gibt man die Anzahl als gebrochene Zahl an, wird sie zur nächsten ganzen Zahl hin gerundet. Ist die Anzahl größer als die Zeichenanzahl des vorgegebenen Strings, wird der gesamte vorgegebene String geliefert.

```
10:X$="SHARP"
20:FOR N=1 TO 6
30:TS$=LEFT$(X$,N)
40:PRINT TS$
60:NEXT N
>
RUN
S
SH
SHA
SHAR
SHARP
SHARP
SHARP
```

LEN

Format: **LEN(<string>)**

LEN ermittelt die Länge eines Strings, d.h. die Anzahl der in ihm enthaltenen Zeichen. Diese Anzahl berücksichtigt auch solche Zeichen, die einem Leerzeichen (space) entsprechen oder aber zu den nicht darstellbaren Zeichen, wie den Steuercodes (control codes) gehören. Ein solches
Steuerzeichen könnte z.B. ein "carriege return" (Symbol: <CR>, Code: &OD) sein.

Beispiel:

```
10:INPUT "GIB EIN WORT EIN :",W$
20:N=LEN(W$)
30:PRINT "DAS WORT HAT";N;" BUCHSTABEN"
40:END
```

Beachten Sie was passiert, wenn die Wörter aus
mehr als 16 Zeichen bestehen.
Beispiel 2

```
10:A$="EINS";B$="ZWEI";C$="DREI"
20:S$=A$+CHR$(13)+B$+CHR$7+C$
30:PAUSE S$
40:PRINT "ANZAHL DER ZEICHEN = ";LEN S$
50:END
>
RUN
EINS ZWEI DREI
ANZAHL DER ZEICHEN = 14
>
```

LET

Format:
[LET] <variable>=<expression>[,<variable>=<expression>……]

LET weist Variablen Werte zu. Numerischen Variablen lassen sich nur numerische Werte zuweisen und Stringvariablen nur Strings.
Das Befehlswort LET ist optional und kann somit auch weggelassen werden. Damit sind die beiden folgenden Zuweisungen identisch:

LET A=5 oder einfach: A=5

Sind Wertzuweisungen hinter den Befehlswörtern THEN und ELSE erwünscht, so muß das Befehlswort LET verwendet werden, wenn eine solche Zuweisung direkt hinter dem Befehlswort erfolgt:

LFILES

Format: **LFILES**

LFILES druckt eine Liste der auf der RAM-Disk (Disk E) befindlichen Dateien, also ein Inhaltsverzeichnis auf dem angeschlossenen Drucker aus. Das Inhaltsverzeichnis zeigt jede Datei unter Nennung folgender Einzelinformationen an:

-Dateiname
-Extension (z.B.: .BAS für BASIC-Programme, TXT für Assembler, C, CASL)

Siehe auch: FILES

LINE

Format:
LINE [(<x-start>,<y-start>)]-(<x-end>,<y-end>)[,S|R|X][,<linien-art>][,B|BF]

LINE zeichnet eine Linie oder ein Rechteck.

Die gezeichnete Linie erstreckt sich dabei vom 1. Punkt mit den Koordinaten (X-start,Y-start) bis zum 2. Punkt mit den Koordinaten (X-end,Y-end) des Displays. Der Ursprung (0,0) des zugrunde gelegten Koordinatensystems befindet sich dabei in der linken oberen Display-Ecke.

Wird die Angabe des 1. Punktes weggelassen, so wird hierfür die momentane Position des Grafik-Cursors angenommen.

x	X-Koordinate des Bildschirms 0 (links) - 143 (rechts), Angaben von -32768 bis 32767 sind erlaubt.
y	y-Koordinate des Bildschirms 0 (oben) - 47 (unten), Angaben von -32768 bis 32767 sind erlaubt.
S|R|X	S:Setzt die Pixel des Kreises(Standard), R: Löscht die Pixel des Kreises, X:Invertiert die Pixel des Kreises
linien-art	Festlegung der Art der Linie mit einem Wert von 0 bis 65535. (&0-&FFFF). Diese Zahl repräsentiert ein Bitmuster für die zu zeichnenden Linie.
B|BF	Diese Optionen dienen der Darstellung eines Rechteckes. Die erste Koordinate geben dann dabei die gegenüberliegenden Ecken an. B zeichnet ein Rechteck mit Umrisslinie und BF ein ausgefülltes Rechteck.

Beispiele:
10:CLS
20:FOR N=10 TO 100 STEP 30

```
30:M=N+20
40:LINE (N,10)-(M,20),,,BF
50:NEXT N
60:END
```

```
10:LINE -(124,31)
```

```
10:LINE (24,0)-(124,47),&HF18F,B
```

```
10:LINE (34,3)-(114,44),X,BF
```

Siehe auch: CIRCLE

LIST

Format: **LIST [<zeilennummer>|"<label>"]**

LIST zeigt die Zeilen eines BASIC-Programmes auf dem Display an.

Ohne Angabe einer <Zeilennummer> oder "<label>" beginnt LIST die zu erstellende Auflistung mit der ersten im Programm enthaltenen Zeile. Sie wird in der oberen Display-Zeile dargestellt. Soweit es der Platz zulässt, zeigt das Display auch die nachfolgenden Programmzeilen an. Der Cursor wird unsichtbar hinter der ersten Zeilennummer positioniert.

Alle weiteren Zeilen können dadurch zur Anzeige gebracht werden, indem man mit der Taste m den Cursor nach unten bewegt und den Display-Inhalt nach oben hin wegrollen (scrollen) 1ässt.

Wird LIST bei der Parameter <Zeilennummer> oder "<label>" angefügt, so beginnt die Auflistung eben mit dieser Zeile. Existiert im Programm keine Zeile mit dieser Zeilennummer, so wird die Auflistung mit der Zeile begonnen, die die nächsthöhere Nummer aufweist. Liegt die <Zeilen-Nr.> über der höchsten im Programm vorkommenden Zeilennummer oder wird das angegebene Label nicht gefunden, so wird ein ERROR-Code angezeigt.

Ein Programm, dass über PASS mit einem Kennwort geschützt ist, lässt sich nicht auflisten, da der Zugang zum PRO-Modus in diesem Falle verwehrt ist, der LIST-Befehl aber nur in diesem Modus akzeptiert wird.

Siehe auch LLIST

LLIST

Format: **LLIST [<zeilennummer>|"<label>"][-[<zeilennummer>|"<label>"]]**

LLIST druckt die Zeilen eines BASIC-Programmes auf dem Display an.

Ohne Angabe einer <Zeilennummer> oder "<label>" beginnt LIST die zu erstellende Auflistung mit der ersten im Programm enthaltenen Zeile.

LLIST wird zwar in ähnlicher Weise verwendet wie der Befehl LIST, ist jedoch flexibler in seinen möglichen Parameterangaben.
Ausgabe zum Drucker:

LLIST sendet das komplette Programm, also alle Zeilen des Programms zum Drucker.

LLIST <Zeilennumer>
LLIST "<label>" bringt nur die gewünschte Zeile zu Papier.

LLIST <Zeilennmmer>-<Zeilennummer>
beginnt die Auflistung mit der Zeile der zuerst angegebenen Zeilennummer oder Label und beendet diese mit der Zeile der als zweites angegebenen Zeilennummer oder Label.

LLIST <Zeilen-Nr.>- beginnt die Auflistung mit der spezifizierten Zeile und setzt diese bis hin zum Programmende fort.

LLIST -<Zeilen-Nr.> beginnt die Auflistung mit der ersten Programmzeile und beendet sie mit Ausdruck der angegebenen Zeile.

Beispiele:

LLIST
LLIST 10-100
LLIST 10-"A"
LLIST "A"-

LNINPUT#

Format: **LNINPUT#<Dateinummer>,Stringvariable[,Stringvariable…….]**

liest Datensätze aus einer sequentiellen Datei, die sich auf der RAM-Disk befinden oder vom seriellen Interface eingelesen werden.

Dateinummer ist die Nummer der Datei die ihr bei der Öffnung durch den OPEN-Befehl zugewiesen worden ist. Ein Versuch, eine ungeöffnete Datei zu lesen, endet mit der Ausgabe eines ERROR-Codes.
Für das Lesen vom seriellen Interface ist die Dateinummer 1. Für Dateien der RAM-Disc entweder 2 oder 3.

Die Liste der Stringvariablen bestimmt die Namen der Variablen, in die die Daten der Datensätze einzulesen sind. Die Aufzählung der Variablen kann sowohl aus einfachen Stringvariablen, Standardvariablen oder Arrays bestehen. Die Variablen sind dabei in ausreichender Länge zu dimensionieren. Arrays müssen in der Variablen-Liste mit dem Pseudo-Index (*) versehen sein, z.B.: A(*). Als Begrenzung beim Einlesen von Daten wird nur CR+LF verwendet.

```
10:LNINPUT #2,AA$

10:LNINPUT #2,AA$,AB$,AC$

10:DIM AA$(4)*16
20:LNINPUT #2,AA$(*)                    Liest 5 Datensätze ein
```

LN

Format: **LN(<zahl>)**

Die Funktion LN(<zahl>) liefert den natürlichen Logarithmus des Argumentes.

Die Basis dieser Logarithmus-Funktion ist die Zahl e. Die Funktion LN ist die Umkehrung der Funktion EXP.
A1s Argument ist jeder beliebige numerische Ausdruck erlaubt, sofern sein Resultat innerhalb des zulässigen Wertebereiches liegt.
Das Argument muss größer oder gleich 1E-99 sein. Werte, die darunter liegen, bewirken die Anzeige des ERROR-Codes 39.

```
10:CLS: INPUT "Argument = ";X
20:PRINT "Der Logarithmus zur Basis"
30:PRINT "e lautet: ";LN(X)
40:INPUT "Weitere Berechnung (J/N)";A$
50:IF A$="J" THEN 10
60:END
```

LOAD

Format: **LOAD "<dateiname>[.BAS]"**

LOAD lädt eine Datei in den internen Speicher, die sich auf einer Ramdisk befindet.

Der **<Dateiname>** bestimmt welche Basic-Datei zu laden ist. Die Extension BAS muss nicht mit angegeben werden. Es kann auch nicht der Name der Ramdisk (E:) oder sonstiger Geräte angegeben werden.

Alle offenen Dateien werden durch LOAD geschlossen

Siehe auch: RUN, SAVE

LOCATE

Format: **LOCATE <spalte>[,<zeile>]**

LOCATE setzt den Text-Cursor des Displays an die gewünschte Zeichenposition.

<Spalte> bestimmt dabei die gewünschte Spalte (X-Position) im Bereich von 0 bis 23. Wenn Spalte nicht angegeben wird, wird die aktuelle Spaltenposition beibehalten.

<Zeile> bestimmt dagegen die gewünschte Zeile (Y-Position) im Bereich von 0 bis 5. Wenn Zeile nicht angegeben wird, wird die aktuelle Zeilenposition beibehalten.

```
10:LOCATE 5
20:PRINT "TEXT1";
30:LOCATE ,4
40:PRINT "TEXT2";
50:LOCATE 0,3
60:PRINT "TEXT3"
```

LOF

Format: **LOF(<Dateinummer>)**

LOF gibt die Anzahl der Bytes an, aus denen eine Datei besteht. LOF ermittelt die Größe von Dateien, die sich auf der RAM-Disk befinden.

Dazu muss die Datei über den OPEN-Befehl geöffnet und von diesem mit einer <Dateinummer> versehen worden sein.

Über diese **<Dateinummer>** ist die Datei, deren Größe ermittelt werden soll, anzusprechen.

```
10:OPEN "E:DATEI1.TXT" FOR INPUT AS #2
20:N=LOF(2)
30:PRINT "DATEI1 BESTEHT AUS ";N;" BYTES"
40:CLOSE #2
```

50:END

LOG

Format: **LOG(<zahl>)**

Die Funktion LOG liefert den dekadischen Logarithmus des angegebenen Argumentes.
Um einen Logarithmus zu einer anderen Basis als 10 zu erhalten, z.B. zur beliebigen Basis B, ist die folgende Formel zu verwenden:

LOG(X)/LOG(B)

Die Umkehrung des Zehnerlogarithmus kann mit der Hilfe des Potenzoperators ^ gebildet werden, wenn man als Potenzbasis die Zahl 10 wählt.

Siehe auch: LN

LPRINT

Format :
LPRINT [USING "format"]
[ausdruck|zeichenfolge[,|;[ausdruck|zeichenkette……][;]

LPRINT gibt Daten über den Drucker aus.

Die Anweisungen LPRINT und LPRINT USING werden dabei in gleicher Weise verwendet wie PRINT und PRINT USING.

LPRINT ohne Parameter sorgt für den Vorschub des Papieres um eine Zeile.

LPRINT mit Parameterangabe sendet die Werte der aufgelisteten Ausdrücke nacheinander aus. Diese Ausdrücke können sowohl numerisch sein oder auch einem String entsprechen. Wird ein Semikolon zur Trennung der Ausdrücke verwendet, so werden ihre Werte unmittelbar hintereinander ausgegeben. Ist a1s Trennzeichen ein Komma gesetzt, so wird der Wert des nachfolgenden Ausdruckes auf die nächste Spalte gesetzt (Beim CE-126P
beginnt die Spalte bei Position 13 oder 0)
Endet die Liste der Ausdrücke mit einem Semikolon, wird die folgende LPRINT-Anweisung an der darauffolgenden Position fortgesetzt. Schließt kein Semikolon die besagte Liste ab, wird ein Zeilenvorschub ausgegeben.
LPRINT USING verhält sich in Analog zu PRINT USING, so das man hierzu auf die dort gemachten Beschreibungen zurückgreifen kann.

Siehe auch: PRINT

MID$

Format: **MID$(<zeichenkette>,<position>,<anzahl>)**

MID$ liefert eine Zeichenkette von <anzahl> Zeichen, beginnend ab <position> der <zeichenkette>.
Die **<Position>** bestimmt, bei welcher Position der vorgegebenen <zeichenkette> die Kopie beginnen soll. Die <Position> kann im Bereich 1…255 liegen. Werte, die sich außerhalb davon befinden, haben die Anzeige eines ERROR-Codes zur Folge. Ist die <Position> größer a1s die Anzahl der im String enthaltenen Zeichen, so wird ein Nullstring erzeugt
<Anzahl> legt fest, wie viele Zeichen von der vorgegebenen <zeichenkette> zu kopieren sind. Sie kann im Bereich 0…255 liegen. Werte mit Nachkommastellen werden auf die nächste ganze Zahl abgerundet.

BEISPIEL :

```
10:Z$="ABCDEFG"
20:Y$=MID$(Z$,3,4)
30:PRINT Y$
> RUN CDEF >
```

Siehe auch: LEFT$, RIGHT$

MON

Format: **MON**

Schaltet in den Maschinensprache-Monitor.

NCR

Format: **NCR(<n>,<r>)**

Liefert die Kombination aus n und r.

Siehe auch: NPR

NEW

Format: **NEW**

NEW löscht das Basic-Programm im Speicher und Daten (alle Variablen). Mit Kennwort (PASS) geschützt Programme können nicht gelöscht werden.

Siehe auch: DELETE, CLEAR

NPR

Format: **NPR(<n>,<r>)**

Liefert die Permutation aus n und r.

Siehe auch: NCR

ON .. GOSUB/GOTO

Format:
ON <ausdruck> GOSUB <zeilennummer>|"<label>",<zeilennummer>|"<label>",....
ON <ausdruck> GOTO <zeilennummer>|"<label>",<zeilennummer>|"<label>",….

ON...GOSUB und ON...GOTO dienen der bedingten Programmverzweigung in Abhängigkeit des Integerwertes eines numerischen Ausdruckes.

Die als Parameter aufgelisteten **<Zeilennummern>** oder **<Marken>** bestimmen jene Programmzeilen, an die nach folgenden Regeln gesprungen wird:

Weist das Verzweigungskriterium **<num. Ausdruck>** den Wert 1 auf, so findet ein Sprung zur ersten Zeile, die in der Parameter-Liste definiert ist, statt. Ergibt der numerische Ausdruck den Wert 2, so wird zur zweiten spezifizierten Zeile gesprungen und so fort. Ist der <num. Ausdruck> mit Nachkommastellen behaftet, wird er auf die nächstniedrigere ganze Zahl (Integer-Wert) abgerundet.

Ist der Wert größer a1s die Anzahl der in der Liste enthaltenen Elemente, wird das Programm mit der nächsten Programmzeile fortgeführt, was auch für jene Fälle gilt, in denen der Wert kleiner als 1 lautet.

GOSUB verzweigt in das Unterprogramm. Nach einem Return (RETURN) wird das

Programm auf den, dem ON-GOSUB folgenden Befehl fortgesetzt. ON-GOTO verzweigt dagegen zur <zeilennummer> oder "<label>" ohne Wiederkehr.

```
10:INPUT "NUMMER (1-3) = ";N
20:ON N GOSUB 100,200,300
:
90:END
100:REM ERSTES UNTERPROGRAMM
190:END
200:REM ZWEITES UNTERPROGRAMM
290:END
300:REM DRITTES UNTERPROGRAMM
380:END
```

Siehe auch: GOSUB, GOTO

OPEN

Format: **OPEN "E:<datei>“ FOR INPUT|OUTPUT|APPEND AS #<dateinummer>**
OPEN "COM:|COM1:|LPRT:|PIO:"

OPEN öffnet eine Datei oder die Ein-/Ausgabe auf einem Gerät. Damit wird das Lesen, Schreiben oder Anhängen von Daten eingeleitet.

Open mit **"E:<datei>"** öffnet eine Datei auf der Ramdisk. Je nach gewünschter Art des Zugriffes ist dem OPEN-Befehl ein entsprechendes Attribut (INPUT, OUTPUT oder APPEND) beizufügen und die Datei in diesem Modus zu öffnen. Auf die Datei kann dann nur zu genau diesem Zwecke zugegriffen werden. Bevor eine Datei zu einem anderen Zweck geöffnet werden kann, muss sie zuvor mit CLOSE geschlossen werden. **<datei>** gib dabei den vollständigen Dateinamen, einschließlich der Extension an.

INPUT erlaubt eine Dateiöffnung, um von der Datei Datensätze sequentiell mittels INPUT# oder LNINPUT# lesen zu können.

OUTPUT gestattet eine Öffnung der Datei, um in diese sequentiell Datensätze mit PRINT# hineinschreiben zu können. In diesem Modus können keine Datensätze an die Datei angefügt werden.

APPEND öffnet die Datei in der Weise, dass an ihren bisherigen Inhalt mit Hilfe von PRINT# weitere Datensätze angefügt werden.

Der Parameter **<Dateinummer>** darf nur den Wert 2 oder 3 annehmen. Alle weiteren Ein- und Ausgabebefehle wie z.B. PRINT# LNINPUT# müssen sich auf diese Nummer

beziehen. Daraus ergibt sich auch das maximal 2 Ramdisk-Dateien gleichzeitig geöffnet sein dürfen.

Dateien können mit OPEN nicht erzeugt werden. Dateien müssen im TEXT-Modus vorher unter RFILES angelegt werden. Auch jede weitere Verwaltung muss dort erfolgen.

Das öffnen von I/O-Geräten erfordert dagegen keine weiteren Optionen.

COM: Serielle Ein-/Ausgabe. Die Ein-/Ausgabe erfolgt über die üblichen Ein- und Ausgabebefehle. Die Dateinummer ist dabei grundsätzlich #1 (z.B. PRINT #1,“Hallo Welt“. Genauso ist eine Kommunikation über die Befehle INP und OUT möglich.
Die Einstellungen für das serielle Interface werden aus dem im TEXT-Modus unter SIO-Format angegeben entnommen.

COM1: Entspricht COM:

PIO: Kommunikation mit dem PIO. Ein-/Ausgabe erfolgt über die Befehle PIOSET, PIOGET, PIOPUT, INP, OUT

LPRT: Ausgabe auf einem seriellen Drucker. Befehle wie LPRINT, LLIST werden auf die serielle Schnittstelle umgeleitet

10:**OPEN** "E:DATA.TXT" FOR OUTPUT AS #2

20:FOR J=ITO 5 30:PRINT #2,J 40:NEXT J 50:CLOSE #2 60:OPEN "E:DATA"FOR INPUT AS #2
70:IF EOF(2)THEN 110 80:INPUT #2,J 90:PRINT J 100:GOTO 70 110:REM DATEIENDE ERREICHT 120:CLOSE #2 130:END

10:OPEN "LPRT:"

10:OPEN "COM1:"

10:OPEN: "COM:"

OUT

Format: **OUT <portadresse>,<byte…..>**
OUT <pattern>

Das erste Format von OUT gibt ein Byte über den gewünschten Port des Z80-kompatib1en Mikroprozessors aus.

<Portadresse> ist eine Adresse (16-Bit-Wert) im Werte- Bereich von 0...65535 (&0...&FFFF), die den gewünschten Port selektiert.

<Byte> gibt das an den Port zu liefernde Byte an. Werden mehrere Bytes a1s Parameter aufgelistet, so wird jedes folgende Byte an die nächste Port-Adresse abgegeben. Aufeinanderfolgende Bytes werden also an aufeinanderfolgende Ports übergeben.

Beispiel:

OUT 80,187

Diese Anweisung sendet den Wert 187 = &BB an den Port 80 (=&50).

Das zweite Format setzt die Bits für BUSY, DOUT und XOUT auf dem 11-Pin-Interface. Vorher muss ein OPEN auf COM:, COM1: oder LPRT erfolgen.

Beispiel:

OUT 6

6= 1x4 + 1x2 + 0x1
BUSY=HI DOUT=HI XOUT=LO

Siehe auch: INP

PAINT

Format: **PAINT (<x>,<y>),<Muster>**

Der Befehl PAINT füllt ab der Koordinate <x>,<y> die Fläche mit dem Muster aus. Die Grenzen zum Füllen werden durch die umschließenden Pixel definiert.

x	x-Koordinate des Bildschirms für den Kreismittelpunkt 0(links) - 143(rechts)
y	y-Koordinate des Bildschirms für den Kreismittelpunkt 0(oben) - 47(unten).
Muster	Muster für das Füllen der Fläche 0-6, 0:ohne Füllung (Standard), 1:waagerechte Linien, 2:=senkrechte Linien 3:Linie nach rechts oben, 4:Linie nach rechts unten, 5:3+4, 6:voller Kreis

Beispiele:

PAINT (71,23),3

PASS

Format: **PASS "<passwort>"**

Das PASS-Kommando erlaubt es, ein Programm durch die Vergabe eines Kennwortes (password) gegen den unerlaubten Zugriff fremder Personen zu schützen. Ein solches **<passwort>** besteht aus bis zu acht, beliebig kombinierten Zeichen, die wie eine Stringkonstante in Anführungsstriche einzuklammern sind. Das Anführungszeichen " kann somit nicht innerhalb des Kennwortes verwendet werden.

Nachdem ein Kennwort gesetzt worden ist, kann der Computer nicht mehr in den PRO-Modus versetzt werden. Die nachstehend genannten Befehle bleiben dann ebenso wirkungslos wie die Tasten ↑ und ↓ :

AUTO, RENUM, LCOPY, LIST, LLIST, BSAVE, SAVE, BLOAD, LOAD, NEW, DELETE

Damit kann das Programm weder aufgelistet, gespeichert noch verändert werden. Ebenso ist ein Überschreiben durch Laden eines anderen Programmes verhindert. Befinden sich mehrere Programme im Speicher, so gilt diese Schutzwirkung für alle diese Programme. Der einzige Weg, um den Schutz aufzuheben, ist, das PASS-Kommando mit dem betreffenden Kennwort nochmalig einzugeben.

Das PASS-Kommando ist nur dann anwendbar, wenn sich im Programmspeicher auch tatsächlich ein Programm befindet.

PASS "GEHEIM"	Dieses Kommando schützt alle im Speicher vorhandenen Programme durch das Kennwort "GEHEIM".

Siehe auch: LOAD, BLOAD

PEEK

Format: **PEEK(<adresse>)**

PEEK liefert den Inhalt einer spezifizierten Speicheradresse.

<Adresse> bestimmt die Adresse im Bereich von &0bis &FFFF (0 bis 65535).

Beispiel:
A=PEEK (100)

A=PEEK &H4001

Siehe auch: POKE, CALL

PI

Format: **PI**

PI liefert die Zahl Pi

PIOGET

Format: **PIOGET**

PIOGET liest ein Byte vom PIO-Port ein. Vorher müssen die Befehle OPEN "PIO:" und PIOSET ausgeführt werden.

PIOPUT

Format : **PIOPUT <byte>**

PIOGET schreib ein Byte zum PIO-Port. Vorher müssen die Befehle OPEN "PIO:" und PIOSET ausgeführt werden.

PIOSET

Format: **PIOSET <byte>**

PIOSET setzte den Signalein- und ausgangsmodus des PIO-Ports.

Das <byte> wird bitweise interpretiert:

Bit 7	EX2	
Bit 6	EX1	
Bit 5	ACK	
Bit 4	Din	PIOSET &HF0
Bit 3	Xout	Din, ACK, EX1, EX2

Bit 2 Xin
Bit 1 Dout
Bit 0 Busy

POINT

Format: **POINT (<X>,<Y>)**

POINT liefert eine Information über den Zustand des spezifizierten Display-Punktes.

Bei POINT (X,Y) bestimmen die Koordinaten X und Y den Display-Punkt. Ist der Punkt dunkel, also gesetzt, so liefert POINT (X,Y) den Wert 1, im anderen Falle den Wert 0. Die Koordinaten können in folgenden Bereichen liegen:

x	X-Koordinate des Bildschirms 0 (links) - 143 (rechts), Angaben von -32768 bis 32767 sind erlaubt.
y	y-Koordinate des Bildschirms 0 (oben) - 47 (unten), Angaben von -32768 bis 32767 sind erlaubt.

Obwohl die <X-Koordinate> und die <Y-Koordinate> in diesen Bereichen liegen sollten, um einen realen Display-Punkt zu adressieren, können sie dennoch jeweils im Bereich -32768...32767 angegeben werden. Das gelieferte Ergebnis lautet in diesen Fällen 0.

POKE

Format: POKE <adresse>,<byte>[,<byte>……]

Mit POKE besteht ein direkter Zugriff auf die Speicherzellen des Computers. Es können hiermit Daten, die in Form von Bytes vorliegen, gezielt in die spezifizierten RAM-Adressen geschrieben werden.

<Adresse> bestimmt, in welche Speicherzelle das (erste) angegebene Byte zu schreiben ist. Die im Bereich von 0..65535 bzw. &0...&FFFF liegende Adresse bezieht sich dabei auf die derzeit gültige oder die über <Bank> spezifizierte Speicherbank.

<Byte> bestimmt einen 8-Bit-Wert im Bereich von 0 bis 255 (bzw.: &0..&FF), der in die durch <Adresse> genau festgelegte Speicherzelle geschrieben werden soll.

Gibt man mehrere Bytes an, die dann mit Kommas voneinander getrennt sein müssen, so werden sie der Reihe nach in aufeinanderfolgende Adressen geschrieben. Der Parameter <Adresse> wirkt dabei als Anfangsadresse.

Reicht der zur Verfügung stehende Speicherplatz nicht aus, um alle aufgelisteten Bytes unterzubringen, so wird ein ERROR-Code angezeigt.

Beispiel:
POKE &FF00,&13,&B7,&37,&C9

POL

Format: **POL(<entfy>,<entfx>)**

Wandelt die numerischen Argumente in ein einem rechtwinkligen Koordinatenformat in ein Polarkoordinatenformat um.

Das erste Argument bezeichnet die Entfernung von der Y-Achse und das zweite von der X-Achse. Die Werte werden umgekehrt. Die Entfernung und der Winkel in den Polarkoordinaten werden den festen Variablen Y und Z zugeordnet. Der umgewandelte Winkel hängt ab von der Einstellung Altgrad, Bogenmaß oder Neugrad.

PRESET

Format: **PRESET (<x>,<y>)**

PRESET löscht den Display-Punkt, der durch die angegebenen Koordinaten bestimmt ist.

Die Koordinaten X und Y können im Bereich von -32768 bis 32767 liegen, ohne daß ein ERROR-Code generiert wird. Es sollten jedoch die folgenden Bereiche beachtet werden, wenn ein tatsächlich existierender Display-Punkt anzusprechen ist:

x	X-Koordinate des Bildschirms 0 (links) - 143 (rechts), Angaben von -32768 bis 32767 sind erlaubt.
y	y-Koordinate des Bildschirms 0 (oben) - 47 (unten), Angaben von -32768 bis 32767 sind erlaubt.

Siehe auch: PSET, LINE

PRINT

Format:
PRINT [USING "format"] [ausdruck|zeichenfolge[,|;[ausdruck|zeichenkette……][;]

PRINT gibt Daten auf dem Bildschirm aus.

PRINT sendet die Werte der aufgelisteten Ausdrücke nacheinander aus. Diese Ausdrücke können sowohl numerisch sein oder auch einem String entsprechen. Wird ein Semikolon zur Trennung der Ausdrücke verwendet, so werden ihre Werte unmittelbar hintereinander ausgegeben. Ist a1s Trennzeichen ein Komma gesetzt, so wird der Wert des nachfolgenden Ausdruckes auf die nächste Spalte 13 (oder wieder 1) gesetzt.

Bevor Sie die nachfolgenden Erklärungen schrittweise durch einzelne Experimente nachvollziehen, sollten Sie wissen, dass das BASIC jede Zeile des Displays in zwei gleichgroße Zonen unterteilt. Jede Zone besteht damit aus 12 Spalten. Diese Zonen-Einteilung wird immer dann von der PRINT-Anweisung berücksichtigt, wenn einzelne Daten vorliegen oder in der Auflistung ein Komma enthalten ist. Mit einem Semikolon voneinander getrennte Daten werden auf dem Display unmittelbar hintereinander gefügt angezeigt.

Wird die Zoneneinteilung berücksichtigt, wei1 in der PRINT-Anweisung nur ein Parameter beigefügt ist oder aber in der Auflistung ein Komma vorliegt, so erfolgt die Anzeige innerhalb einer solchen Zone :

bei numerischen Daten rechtsbündig,
bei String-Daten jedoch linksbündig.

Numerische Einzel-Daten werden in der rechten, einzelne String-Daten in der linken Zone dargestellt.

Endet die Liste der Ausdrücke mit einem Semikolon, wird die folgende LPRINT-Anweisung an der darauffolgenden Position fortgesetzt. Schließt kein Semikolon die besagte Liste ab, wird ein Zeilenvorschub ausgegeben.

LPRINT USING verhält sich in Analog zu PRINT USING, so das man hierzu auf die dort gemachten Beschreibungen zurückgreifen kann.

Eine PRINT-Anweisung ohne jegliche Parameterangabe bewirkt die Ausgabe einer Leerzeile, was praktisch mit einem Zeilenvorschub beim Drucker verglichen werden kann.

USING

Ein PRINT-Befehl kann mit einer oder mehreren USING-Anweisungen versehen sein, so dass sich die Daten formatiert anzeigen lassen. Das jeweilige Format wird durch einen <Format-String>, der der USING-Anweisung als Parameter beizugeben ist, bestimmt.

Der <Format-String> besteht aus einer Reihe von speziellen Zeichen, die in Anführungsstriche eingeschlossen sein müssen.

Die Zeichen, die den <Format-String> bilden, sind:

#	Rechtsbündiges Zeichen eines numerischen Feldes
.	Begrenzer zwischen der ganzen Zahl und dem Dezimalanteil
,	verwendet das Komma als Trennzeichen nach 3 Ziffern in numerischen Feldern
^	Zur Anzeige der Zahl in wissenschaftlicher Notation
&	Linksbündiges alphanumerisches Feld

Das Nummernzeichen und das Kaufmannsund sind sogenannte Platzhalter. Für jedes im <Formatstring enthaltene # kann eine Ziffer des numerischen Wertes angezeigt werden, für jedes & dagegen ein Zeichen eines Strings. Alle weiteren Formatsymbole dienen der näheren Beschreibung numerischer Formate. Bei den numerischen Formaten lassen sich sowohl positive als auch negative Werte darstellen. Das Vorzeichen wird jedoch nur bei den negativen Werten angezeigt.

Zusammen mit dem Stringformat lassen sich insgesamt sechs grundlegende Formate unterscheiden, die in den Erklärungen zu USING näher erläutert sind.

(1) "###" 43

(2) ###." 98.

(3) "###.##"
64.29
5.00
-13.44
-2.00

(4) "##.## "
23.11E 05
-4.33E-02
7.00E 00

(5) "###,###."

34,567.
2.
-230.
2,345.

(6) "&&&&&&" ABCDEF

Die maximale Anzahl der Formatsymbole # beträgt in den Formaten (1), (2), (3), (4) und (6) 11 und im Format (5) 14.

PRINT#

Format : **PRINT#<Dateinummer>,<Variable>[,|;Variable……][,|;]**

PRINT# und wird benutzt, um Daten sequentiell in eine Datei oder auf dem seriellen Interface zu schreiben.

<Dateinummer> ist die Nummer, unter der die Datei mittels OPEN-Befehl geöffnet wurde. Der Versuch, etwas in eine ungeöffnete Datei zu schreiben bewirkt die Ausgabe eines ERROR-Codes. Dieser Wert muss 1(seriell), 2 oder 3 sein.

Sind die Daten durch Semikolons voneinander getrennt, so werden sie in der Datei ohne einen Zwischenraum aneinandergefügt. Wird ein Komma in der Auflistung der Daten verwendet, werden dagegen die Daten in Zonen zu 20 Zeichen getrennt. Soll das Komma selbst mit in die Datei gelangen, ist es in Anführungsstriche zu setzen.

Die Aufzählung der Daten kann sowohl numerische als auch String-Variablen enthalten. In beiden Fällen ist auf die Verwendung der richtigen Trennungszeichen zu achten. Ansonsten kann es Probleme beim Einlesen mittels INPUT# geben. Mit Array-Variablen können keine individuellen Elemente angesprochen werden. Es muss das gesamte Array in der Form A(*) angegeben sein.

PRINT->LPRINT

Format: **PRINT=LPRINT**
PRINT=PRINT

PRINT=LPRINT lenkt den PRINT-Befehl auf den Drucker um.

PRINT=PRINT setzt die Ausgabe zurück auf den Bildschirm

PSET

Format: **PSET (<x>,<y>)[,X]**

PSET setzt oder löscht einen Display-Punkt, der durch zwei Koordinaten bestimmt ist. Ein gesetzter Punkt erscheint dunkel, ein gelöschter dagegen hell.

Die Koordinaten X und Y können im Bereich von -32768 bis 32767 liegen, ohne dass ein ERROR-Code generiert wird. Es sollten jedoch die folgenden Bereiche beachtet werden, wenn ein tatsächlich existierender Display-Punkt anzusprechen ist:

x	X-Koordinate des Bildschirms 0 (links) - 143 (rechts), Angaben von -32768 bis 32767 sind erlaubt.
y	y-Koordinate des Bildschirms 0 (oben) - 47 (unten), Angaben von -32768 bis 32767 sind erlaubt.

Benutzt man PSET ohne den 3. Parameter, so wird der mit den Koordinaten bestimmte Display-Punkt gesetzt. Ist der Parameter vorhanden, invertiert PSET den gerade derzeitigen Zustand des Punktes. Ein heller Punkt wird dunkel, ein dunkler hell. Der Parameter muss aus dem Buchstaben X bestehen.

Siehe auch: PRESET, LINE, CIRCLE

RADIAN

Format: **RADIAN**

Versetzt den Computer in den Winkelmodus RADIAN
In diesem Modus werden alle Winkelangaben als im Bogenmaß gegeben angesehen und auch in diesem Maß ausgegeben.

Zur Kennzeichnung dieser Betriebsart wird in der Status-Zeile das Symbol RADIAN angezeigt.

Die Argumente der Funktionen SIN, COS und TAN werden dann als im Bogenmaß vorliegend angesehen und die Werte der Funktionen ASN, ACS und ATN in diesem Maß geliefert.

```
10:RADIAN
20:PAUSE "WINKEL IM BOGENMASS"
30:PRINT ASN(0.5),ASN(1)
40:PRINT ACS(0.5),ACS(1)
50:PRINT ATN(0.5),ATN(1)
60:END
RUN   WINKEL IM BOGENMASS 5.235987E-01 1.570796327 1.047197551 0
0.463647609 7.853981E-01 >
```

Lassen Sie dieses Programm zum Vergleich auch in den beiden anderen Winkel-Modi (DEGREE und GRAD) laufen.

Siehe auch: DEGREE, GRAD

RANDOMIZE

Format: **RANDOMIZE**

RANDOMIZE bestimmt die Serie der Zufallszahlen, die über die Funktion RND geliefert werden. Benutzt man RANDOMIZE zu Beginn eines Programms, wird mit jedem Einschalten des Computer eine neue Reihe von Zufallszahlen erzeugt .

```
10:RANDOMIZE
20:FOR I=ITO 5
30:PRINT I; " . ZUFALLSZAHL = ";
40:PAUSE RND ( 10 )
50:NEXT I
60:END
RUN
1. ZUFALLSZAHL = 2
2. ZUFALLSZAHL = 1
3. ZUFALLSZAHL = 10
4. ZUFALLSZAHL = 8
5. ZUFALLSZAHL = 10
>
OFF ON

> RUN

1. ZUFALLSZAHL = 2
2. ZUFALLSZAHL = 1
3. ZUFALLSZAHL = 10
4. ZUFALLSZAHL = 8
5. ZUFALLSZAHL = 10
>
```

Siehe auch: RND

RCP

Format: **RCP(<zahl>)**

Gibt die Reziproke des numerischen Arguments zurück.

READ

Format: **READ variable[,variable……]**

READ liest aus den im Programm befindlichen DATA-Zeilen die dort enthaltenen Werte ab und weist sie den gewünschten Variablen zu.

Damit eine READ-Anweisung durchgeführt werden kann, muss innerhalb des Programms mindestens eine DATA-Zeile vorhanden sein.

Jeder als Parameter aufgeführten Variablen wird dann der Reihe nach das nächste in einer DATA- Zeile auffindbare Element zugeordnet. Die in der Parameterauflistung enthaltenen Variablen müssen jeweils typenmäßig den in der DATA-Anweisung auftretenden Konstanten entsprechen.

Die einzulesenden Werte brauchen nicht allesamt in einer gemeinsamen DATA-Zeile stehen, sondern können auf beliebig viele DATA-Zeilen verteilt sein. Es gilt dabei immer, dass mit der nächsten READ-Variablen der nächste in einer DATA-Zeile auffindbare Wert gelesen wird.

Sind die in den DATA-Zeilen stehenden Werte alle gelesen, führt die nächste READ-Anweisung zur Ausgabe eines ERROR-Codes.

```
10:DIM B(10)
20:FOR I=1 TO 10 : READ B(I)
30:PRINT B(I) : NEXT I
40:DATA 10,20,30,40,50
50:DATA 60,70,80,90,100
```

Siehe auch: DIM, RESTORE

REC

Format: **REC(<entfernung>,<winkel>)**

REC wandelt numerische Argumente im polaren Koordinatenformat in ein rechtwinkliges Koordinatenformat um.
Das erste Argument bezeichnet die Entfernung und das zweite den Winkel. Die Winkelangabe hängt ab von der Einstellung des Computers (DEG, RAD oder GRAD).
Die umgekehrten Werte, die Entfernungen von der x-Achse und der y-Achse, werden den festen Variablen Y und Z zugewiesen.

Beispiel:
REC(12,30)

REM

Format: **REM**

REM erlaubt die Einfügung von Kommentaren in den Programmtext. Diese Kommentare dienen zur Kennzeichnung von Programmteilen, so dass man ihre Funktionen noch zu einem späterem Zeitpunkt wieder erkennen und verstehen kann. Sie erscheinen ausschließlich im Programmlisting. Sie bleiben also bei der Programmausführung unsichtbar. Anstelle von REM kann auch der Apostroph ' verwendet werden. Die so gekennzeichneten Programmzeilen zählen zu den nicht ausführbaren Anweisungen. Werden diese durch GOTO oder GOSUB angesprungen, erfolgt die Programmausführung mit der nächsten Zeile , die nicht als Kommentarzeile ausgewiesen ist. Kommentare können an Anweisungen einer Zeile angefügt werden, wenn man vor dem Befehlswort REM den Doppelpunkt als Trennzeichen benutzt.

```
10:V=G*H/3: REM VOLUMEN DER PYRAMIDE
```

Einem Apostroph darf kein Doppelpunkt vorausgehen , da dieser die Trennfunktion beinhaltet:

```
10:V=G*H/3 'VOLUMEN DER PYRAMIDE
```

Nach der Einleitung eines Kommentares gilt der gesamte Rest der Zeile als Kommentar. Hinter einem Kommentar stehende Anweisungen, die zur selben Zeile gehören, werden ignoriert:

10:REM VOLUMEN DER PYRAMIDE**:** V=G*H/3 20:'VOLUMEN DER PYRAMIDE: V=G*H/3

RENUM

Format: **RENUM [<alteZeile>[,<neueZeile>][,<zeilenabstand>]]**

RENUM nimmt eine Neunumerierung der Zeilen eines BASIC-Programmes vor.

<alteZeile> bestimmt, ab welcher Zeile mit der neuen Numerierung begonnen wird.

<neueZeile> bestimmt, wie die erste neue Zeilennummer lautet.

<zeilenabstand> bestimmt, in welchen Abstand die Nummern generiert werden sollen. Fehlt diese Angabe, erfolgt die Numerierung in Zehnerschritten.

Bei der Neunumerierung werden automatisch alle Zeilennummern in GOTO- und GOSUB-Anweisungen entsprechend korrigiert. Dies gilt auch für die Zeilennummern, die in anderen Verzweigungsanweisungen, wie z.B. IF..THEN..ELSE, enthalten sind. Taucht

dabei aber eine nichtexistente Zeilennummer auf, wird die Meldung:

Undefined in <Zeilen-Nr.>

angezeigt und die Neunummerierung unterlassen. Die <Zeilen-Nr.> verweist dabei auf die Zeile, in der der Fehler entdeckt worden ist.

REPEAT .. UNTIL

Format: **REPEAT**
Anweisung
UNTIL <bedingung>

Die Anweisungen nach REPEAT werden so lange in einer Schleife ausgeführt bis die Bedingung nach UNTIL erfüllt ist.

REPEAT und UNTIL müssen immer zusammen verwendet werden.

REPEAT und UNTIL schließen eine Gruppe von Anweisungen ein, die wiederholt werden sollen. Nach der Ausführung der Anweisung nach REPEAT wird die Bedingung von UNTIL geprüft. Wenn die Bedingung erfüllt ist, wird die Ausführung mit der Anweisung nach Until fortgesetzt. Damit ist die Schleife beendet. Wenn die Bedingung nicht erfüllt ist, werden die Anweisungen nach REPEAT so lange ausgeführt bis die Bedingung erfüllt ist.

Eine REPEAT-UNTIL-Schleife kann mit einer anderen verschachtelt sein. Die innere Schleife muss vollkommen in der äußeren Schleife verschachtelt sein.

Wenn die Ausführung die REPEAT-UNTIL-Schleife verlässt bevor die Schleifenbedingung erfüllt ist, kann ein Verschachtelungsfehler auftreten, je nach dem auf welche Weise die Schleifen ausgeführt werden (wenn das Programm z.B. verschiedene REPEAT-Anweisungen enthält)

Innerhalb einer REPEAT-UNTIL-Schleife dürfen die Kommandos CLEAR, DIM bzw. ERASE nicht verwendet werden.

RESTORE

Format: **RESTORE [<zeilennummer>|"<label>"]**

RESTORE setzt den internen DATA-Zeiger zurück oder auf den Anfang der gewünschten DATA-Zeile. Damit lassen sich die in den DATA-Zeilen bereitgestellten Werte erneut lesen.

Wird RESTORE ohne Parameter verwendet, zeigt der interne Zeiger anschließend auf den ersten Wert der zuerst im Programm auffindbaren DATA-Zeile.

Bei der Angabe einer <zeilennummer> oder eines "<Labels>" wird der Zeiger auf das erste Element der in dieser Zeile vorkommenden DATA-Anweisung gesetzt.

Nach dem Setzen des Zeigers lassen sich die Werte der momentan vom Zeiger bestimmten DATA-Liste mit jeder folgenden READ-Anweisung der Reihe nach ablesen und den durch READ bestimmten Variablen zuweisen. Sind alle Werte einer DATA-Anweisung gelesen, sucht der Computer nach der nächsten im Programmspeicher befindlichen Zeile mit einer DATA-Anweisung und setzt den Zeiger auf deren Anfang und so fort.

Enthält die als Parameter angegebene Zeile keine DATA-Liste, wird der Zeiger auf den Anfang der nächsten auffindbaren DATA-Anweisung gesetzt.

BEISPIEL :
```
100:DIM A$(3*10)
110:GOSUB "OBST"
120:RESTORE
130:GOSUB "OBST"
140:RESTORE 310
150:GOSUB "OBST"
160:END
200:"OBST"
210:FOR N=1 TO 3
220:READ A$(I)
230:PAUSE A$
240:NEXT N
250:PAUSE
260: RETURN
300:DATA "PFLAUME","PFIRSICH","NEKTARINE"
310:DATA "APFEL","BIRNE","MANDARINE"
>
RUN
PFLAUME
PIRSICH
NEKTARINE
PFLAUME
PIRSICH
NEKTARINE
APFEL
BIRNE
MANDARINE
```

Siehe auch: DIM, READ

RIGHT$

Format: **RIGHT$(<zeichenkette>,<anzahl>)**

Die Funktion RIGHTS liefert den rechtsbündigen Teil eines vorgegebenen Strings, wobei bestimmt werden kann, wie viele Zeichen von rechts-aus dem String abgelesen werden. Die <Anzahl> der Zeichen des Teilstringes muss im Bereich von 0 bis 255 liegen. Gibt man die Anzahl als gebrochene Zahl an, wird sie zur nächsten ganzen Zahl hin gerundet. Ist die Anzahl größer als die Zeichenanzahl des vorgegebenen Strings, wird der gesamte vorgegebene String geliefert

```
5:WAIT 32
10:X$="SHARP"
20:FOR N=1TO 6
30:S$=RIGHT$(XS,N)
40:PRINT S$
50:NEXT N
60:WAIT
70:END
>RUN

P
RP
ARP
HARP
SHARP
SHARP
>
```

Siehe auch: MID$, RIGHT$

RND

Format: **RND(<zahl>)**

Die Funktion RND liefert eine Zufallszahl, deren maximale Größe vom angegebenen Argument abhängt.

Mit Hilfe der RND-Funktion lassen sich scheinbar willkürlich verteilte Zahlen erzeugen. Schaltet man jedoch den Computer an und startet das mit der RND-Funktion versehene Programm erneut, so stellt man fest, dass genau dieselben "Zufallszahlen" geliefert werden wie zuvor. Sollen die gelieferten Werte einem "echten Zufallsgesetz" folgen, muss vor Aufruf von RND-Funktionen im Programm das Befehlswort RANDOMIZE

aufgeführt sein.

Ist das Argument der RND-Funktion kleiner als Null liefert jeder erneute Funktionsaufruf immer wieder dieselbe Zufallszahl.

Ist das Argument größer als Null, aber kleiner als 1, liegen die gelieferten Funktionswerte im Bereich von 0 bis 1, ohne Einschluss der 1. Alle Werte werden hierbei mit bis zu 10 signifikanten Stellen geliefert.

Ist das Argument jedoch größer a1s 1, liegen die Funktionswerte im Bereich von 1 bis zum Argument selbst. Hierbei werden allerdings nur Integerwerte ausgegeben.

```
10:FOR I=1 TO 3
20:FOR J=1 TO 10 : R=RND(9) : PRINT R; : NEXT J
30:PRINT : NEXT I
40:END

>RUN   6 425682768 5 577126536 3 157345742
```

Siehe auch: RANDOMIZE

RUN

Format: **RUN [<zeilennummer>|"<label>"]**

RUN startet ein im Speicher befindliches BASIC-Programm.

Ohne Parameterangabe beginnt die Ausführung des Programms mit dessen erster Zeile, also der, der kleinsten vorkommenden Zeilennummer.

Mit Angabe der <Zeilennummer> oder einer <label> wird das Programm ab der spezifizierten Stelle gestartet.

In jedem Falle löscht RUN Variablen und setzt den internen Zeiger für die Auswahl der in den DATA-Zeilen bereitgestellten Daten auf die erste mögliche Position.

```
> RUN

>  RUN 100

>  RUN "F"

>  RUN "MARKE"
```

Siehe auch: END, STOP

SAVE

Format: **SAVE "<dateiname>[.BAS]"**

SAVE sichert das Basic-Programm im Speicher auf der RAM-Disk.

Existiert bereits eine Datei unter gleichem Namen, so wird diese mit der neuen Datei überschrieben.

Lässt man in der Dateibezeichnung die Extension weg, wird sie automatisch zu .BAS angenommen.

BEISPIEL :

```
SAVE "TEST"
```

Siehe auch: LOAD, KILL

SGN

Format: **SGN(<zahl>)**

Die Funktion SGN liefert das Vorzeichen des Argumentes zahl.

<zahl> kann dabei jeder beliebige numerische Ausdruck sein. Die dazu gelieferten Funktionswerte lauten -1,0 oder 1 und ergeben sich wie folgt:

Argument

```
5:WAIT 100
10:FOR N=-3 TO 3
20:PRINT N,SGN(N)
30:NEXT N
40:END
> RUN

>
```

Funktionswert SGN
1 0 -1

SIN

Format: **SIN(<zahl>)**

Diese Funktion liefert den Sinus-Wert eines Winkelargumentes,

Der angegebene Winkel kann in Altgrad, Bogenmaß oder Neugrad vorliegen. Damit der richtige Wert der Sinusfunktion geliefert werden kann, muß der Computer in den richtigen Winkelmodus geschaltet sein. Hierbei gilt:

Winkelmaß	Winkelmodus
Altgrad	DEGREE
Bogenmaß	RADIAN
Neugrad	GRAD

```
10:DEGREE
20:G$=CHR$ (&F8)
3O:PRINT "sin(3O";G$;") = ";SIN(3O)
40:PRINT "sin(45";G$;") = ";SIN(45)
50:END
>
RUN
sin(30°) = 0.5
sin(45°) = 7.071067812E-O1
>
```

SQR

Format: **SQR(<zahl>)**

Die Funktion SQR liefert den positiven Wert der Quadratwurzel von <zahl>.

Das Argument <zahl> darf ein beliebiger numerischer Ausdruck sein, dessen Wert jedoch im positiven Zahlen-Bereich liegen oder Null lauten muss. Negative Werte sind nicht erlaubt und führen daher zur Ausgabe eines ERROR-Codes.

BEISPIELE :

```
> SQR(5)

>
```

SQU

Format: **SQU(<zahl>)**

Die Funktion SQU liefert die Quadratwurzel von <zahl>.

Das Argument <zahl> darf ein beliebiger numerischer Ausdruck sein, dessen Wert jedoch im positiven Zahlen-Bereich liegen oder Null lauten muss.

BEISPIELE :

> SQU(5)

>

STOP

Format: **STOP**

STOP dient der Unterbrechung eines Programms, um so während der Testphase bei fehlerhaften Programmteilen anhalten und diese untersuchen zu können.

Kommt ein STOP-Befehl zur Ausführung, wird das laufende Programm abgebrochen und eine Meldung angezeigt, die angibt, bei welcher Zeile dieser Abbruch erfolgte:

BREAK IN <Zeilennummer>

Nun kann durch Abfrage der Variablen und anderer Untersuchungen auf die Ursache der ermittelten Programmfehler geschlossen werden. Ebenso lassen sich nun den falsch belegten Variablen über die Tastatur die erwarteten Werte zuweisen und durch Start des Programms mit der Anweisung CONT das weitere Verhalten getestet werden. Diese Fortsetzung ist aber nur möglich, wenn inzwischen keine Änderungen der Programmzeilen vorgenommen worden sind. -

Im Gegensatz zum Befehl END werden bei STOP keine geöffneten Dateien geschlossen.

```
10:FOR N=1 TO 10
20:LET S=N*5
30:STOP
40:GRAPH
50:LINE (0,0)-(N,S)
60:NEXT N
```

STR$

Format: **STR$(<zahl>)**

Die Funktion STR$ wandelt einen numerischen Wert in einen String um.

Der gelieferte String setzt sich aus genau jenen Zeichen zusammen wie der numerische Wert auch. Jedoch hat der String keinen Wert, mit dem sich Berechnungen durchführen lassen.

Die Funktion STR$ kann man als die Umkehrung der VAL-Funktion ansehen.

Ist der numerische Wert negativ, enthält auch der String das betreffende Vorzeichen.

Ist der numerische Wert zu groß, um ihn mit zehn Ziffern darstellen zu können, erscheint er auch im String in der Fließkomma-Schreibweise.

```
:
110:N=N*3
120:A$=STR$ (N)
130:B$=LEFT$ (A$,1)
140:M=VAL (B$)
:
```

SWITCH..CASE..DEFAULT..ENDSWITCH

Format: **SWITCH <variable>**
CASE <zahlenfolge>|<buchstabenfolge>
anweisung
CASE <zahlenfolge>|<buchstabenfolge>
anweisung
……
[DEFAULT
anweisung]
ENDSWITCH

Ausführung einer Anzahl von bestimmten Anweisungen entsprechend dem Wert einer gegebenen Variablen.

SWITCH und ENDSWITCH müssen immer zusammen verwendet werden. Dieses Kommando vergleicht den Wert der Variable die nach SWITCH folgt mit einer Zahlen

oder Buchstabenfolge, die jedes mal nach CASE folgt. Bei Übereinstimmung werden die Anweisungen zwischen der übereinstimmenden CASE-Anweisung und dem nächsten CASE-Befehl ausgeführt. Wenn der Wert der Variablen mit keiner CASE-Anweisung übereinstimmt wird die DEFAULT-Anweisung ausgeführt. Wenn keine DEFAULT-Anweisung verfügbar ist, wird statt dessen die ENDSWITCH-Anweisung ausgeführt.

CASE, DEFAULT und ENDSWITCH müssen immer direkt nach einer Zeilennummer folgen, nicht nach einem Label.

Wenn die gleiche Zahlenfolge oder Zeichenkette in mehr als einer CASE-Anweisung verwendet wird, wird bei Übereinstimmung diejenige CASE-Anweisung ausgeführt, die am nächsten zu der SWITCH-Anweisung ist.

SWITCH-Anweisung dürfen nicht ineinander verschachtelt werden.

TAN

Format: **TAN(<zahl>)**

Die Funktion TAN liefert den Tangens eines als Winkel aufzufassenden numerischen Wertes.

Der angegebene Winkel kann in Altgrad, Bogenmaß oder Neugrad vorliegen. Damit der richtige Wert der Tangensfunktion geliefert wird, muss sich der Computer im entsprechenden Winkelmodus befinden:

Winkelmaß	Winkelmodus
Altgrad	DEGREE
Bogenmaß	RADIAN
Neugrad	GRAD

Da die Tangensfunktion sogenannte Polstellenaufweist, an denen der Wert ins Unendliche geht, wird bei diesen Winkelargumenten (z.B. 90) ein ERROR-Code ausgegeben.

```
10:DEGREE
20:PAUSE "WINKEL SIND IN ALTRGRAD !"
30:PAUSE "WINKEL: 0, TANGENS:";TAN(0)
40:PAUSE "WINKEL: 45, TANGENS:";TAN(45)
50:PAUSE "WINKEL: 90, TANGENS:";TAN(90)
>RUN
WINKEL SIND IN ALTGRAD !
WINKEL: 0, TANGENS: 0
WINKEL: 46, TANGENS: 1
WINKEL: 90, TANGENS:
```

ERROR 37 IN 70 (Taste CL drücken !)

TEN

Format: **TEN(<zahl>)**

Die Funktion TEN liefert den Antilogarithmus des angegebenen Argumentes. Zeigt den Wert von 10(die Basis des normalen Logarithmus) erhoben auf dem Wert des eigenen numerischen Arguments.

TRON / TROFF

Format : **TRON**

TROFF

Mit TRON und TROFF kann der TRACE-Modus ein- bzw. ausgeschaltet werden.

HINWEISE :Ist dieser Modus über den Befehl TRON aktiviert, hält der Computer nach Ausführung einer jeden BASIC-Zeile an und gibt die Nummer der betreffenden Zeile auf dem Displays aus.

Die Taste jedes mal erneut betätigt werden, damit die folgende Programmzeile zur Ausführung kommt. Wird diese Taste dauernd gedrückt, arbeitet der Computer die Programmzeilen nacheinander ab, ohne die zugehörigen Zeilennummern anzuzeigen. Eine gerade abgearbeitete Zeile kann durch Betätigung der Tasten und sichtbar gemacht werden.

Stoppt das Programm infolge eines PRINT- oder INPUT-Befehles, kann es durch die Betätigung der ENTER Taste weitergefahren werden.

Hält das Programm aufgrund eines STOP-Befehles an oder wurde es durch die BREAK-Taste abgebrochen, kann mit der CA Taste oder der Taste das Programm wieder in Betrieb genommen werden.

TROFF schaltet den Trace-Modus aus.

TRON und TROFF können auch programmiert werden. Der Trace-Modus bleibt dann solange bestehen, bis im Programm der nächste TROFF-Befehl vorgefunden wird.

USING

Format: **USING [<formatstring>]**

USING erlaubt eine formatierte Datenausgabe.
Das Format wird durch einen **<Format-String>** bestimmt, der sich aus folgenden Zeichen zusammensetzen kann:

Das jeweilige Format wird durch einen <Format-String>, der der USING-Anweisung als Parameter beizugeben ist, bestimmt.

Der <Format-String> besteht aus einer Reihe von speziellen Zeichen, die in Anführungsstriche eingeschlossen sein müssen.

Die Zeichen, die den <Format-String> bilden, sind:

#	Rechtsbündiges Zeichen eines numerischen Feldes
.	Begrenzer zwischen der ganzen Zahl und dem Dezimalanteil
,	verwendet das Komma als Trennzeichen nach 3 Ziffern in numerischen Feldern
^	Zur Anzeige der Zahl in wissenschaftlicher Notation
&	Linksbündiges alphanumerisches Feld

Das Nummernzeichen und das Kaufmannsund sind sogenannte Platzhalter. Für jedes im <Formatstring enthaltene # kann eine Ziffer des numerischen Wertes angezeigt werden, für jedes & dagegen ein Zeichen eines Strings. Alle weiteren Formatsymbole dienen der näheren Beschreibung numerischer Formate. Bei den numerischen Formaten lassen sich sowohl positive als auch negative Werte darstellen. Das Vorzeichen wird jedoch nur bei den negativen Werten angezeigt.

Zusammen mit dem Stringformat lassen sich insgesamt sechs grundlegende Formate unterscheiden, die in den Erklärungen zu USING näher erläutert sind.

(1) "###" 43

(2) ###." 98.

(3) "###.##"
64.29
5.00
-13.44
-2.00

(4) "##.## "
23.11E 05
-4.33E-02

7.00E 00

(5) "###,###."

34,567.
2.
-230.
2,345.

(6) "&&&&&&" ABCDEF

Die maximale Anzahl der Formatsymbole # beträgt in den Formaten (1), (2), (3), (4) und (6) 11 und im Format (5) 14.

VAL

Format: **VAL(<zeichenkette>)**

VAL wandelt einen String in einen numerischen Wert um.

Die VAL-Funktion kann als Umkehrung der beiden Funktionen STR$ und HEX$ angesehen werden. Sie konvertiert einen String, der aus numerischen Zeichen besteht, in einen numerischen Wert.

Ist das Funktionsargument ein dezimaler String, muss er aus den Zeichen 0 bis 9 zusammengesetzt sein. Er darf einen Dezimalpunkt und die Angabe eines Exponenten enthalten sowie ein Vorzeichen für die Mantisse und eines für den Exponenten aufweisen. In einem solchen Falle entspricht VAL genau der Umkehrung von STR$.

Ist das Funktionsargument dagegen ein hexadezimaler String, muss das erste Stringzeichen ein & sein und die nachfolgenden Zeichen aus solchen Symbolen bestehen, die zur Darstellung von Hex- Ziffern verwendet werden. In diesem Falle wirkt VAL als Umkehrung der HEX$-Funktion.

Enthält das Argument unzulässige Zeichen, wird der Funktionswert 0 geliefert.

```
10:INPUT "FREQUENZ = ";A$
15:IF ASC(A$)<48 OR ASC(A$)>57 THEN 100
20:F=VAL(A$)
30:PRINT F
40:END
:
100:PRINT "NUR ZAHLENEINGABE ERLAUBT"
```

VDEG

Format: **VDEG(<zeichenkette>)**

VDEG wandelt einen String in sexagesimale Notation im Format "Stunden,Minuten,Sekunden" zu einem Winkel in Altgrad um, wobei folgendes gilt:
hh Stunden mm Minuten ss Sekunden rr Dezimaler Sekundenrest 00...99

```
10:AA$="1° 30' 36""
20:B=VDEG AA$
30:PRINT B
> RUN

1. 51
```

WAIT

Format: **WAIT [<zahl>]**

WAIT bestimmt die Zeitdauer, die nach Ausführung einer PRINT-Anweisung abzuwarten ist, bevor mit der nächsten Anweisung fortgefahren wird.

WAIT ohne Parameterangabe setzt die Wartezeit auf unendlich. Nach einer PRINT-Anweisung stoppt das Programm also vollständig. Wie man aber aufgrund des angezeigten Bereitschaftszeichens > erkennen kann, ist der Programmablauf aber nicht abgebrochen. Er pausiert lediglich und kann mit Betätigung der ENTER-Taste fortgesetzt werden.

Wird der Parameter **<Dauer>** angegeben, bestimmt dieser die Wartezeit als ein Vielfaches einer 1/64 Sekunde. Der Wert 64 ergibt also eine Pause von 1 Sekunde, der Wert 128 eine von 2 Sekunden usw. Der erlaubte Wertebereich erstreckt sich von 0 bis 65535.

Ohne Angabe eines Wertes wird der Wert 0 angenommen. Damit wird der Print-Befehle nicht angehalten. Dies ist auch der Standard nach dem Start des Programms.

Beispiel:

```
10:FOR I=1 TO 10
20:WAIT (64*I)
30:PRINT "*";
```

```
40:NEXT I
50:WAIT
60:END
>
RUN
**********
>
```

Jeder Stern erscheint 1 Sekunde später als der vorhergehende.

WHILE .. WEND

Format: **WHILE <bedingung>**
Anweisung
WEND

Die Anweisungen zwischen WHILE UND WEND werden so oft ausgeführt, solange die Bedingung erfüllt ist.

WHILE und WEND müssen immer zusammen verwendet werden.

Die Bedingung der WHILE-Anweisung wird zuerst geprüft. Wenn die Bedingung nicht erfüllt ist, geht die Ausführung zu der Anweisung nach WEND.

Wenn die Bedingung erfüllt ist, werden die Anweisungen zwischen WHILE und WEND so lange wiederholt bis die Bedingung nicht mehr erfüllt ist.

Ist die Bedingung der WHILE-Anweisung nicht erfüllt wird die Schleife nicht ausgeführt.

Innerhalb einer WHILE-Schleife dürfen die Kommandos CLEAR, DIM bzw. ERASE nicht verwendet werden.

Anhang A: 11-Pin Interface

Signale und Belegungen

Der PC-G850V(S) besitzt auf der linken Seite ein 11-poliges Interface zur Kommunikation mit anderen Geräten. Es handelt sich um ein multifunktionales Interface, welches in verschiedenen (Sub-)Modi betrieben werden kann. Der aktuelle Modus wird dabei durch die Betriebsarten des PC-G850V(S) gesteuert:

1. SIO / RS-232C-Modus (z.B. `OPEN"COM:"`)
2. SSIO-Modus (Synchronous Serial Input/Output)
   a. CE-126P Druckerprotokoll (z.B. `LPRINT` *ohne* vorheriges `OPEN`)
   b. LPRT-Protokoll (z.B. `OPEN"LPRT:"`)
3. PWM-Modus (Pulsweitenmodulation)
   a. CE-126P Bandprotokoll (z.B.. `BSAVE/BLOAD` via CE-126P)
   b. Generisches PWM-Protokoll (z.B. `BSAVE/BLOAD` mit zweitem PC-G850V)
4. PIO-Modus (z.B. `OPEN"PIO:"`)
   Programmierbare, 8-Bit parallele Port-Schnittstelle
5. PIC-Modus (Aktivierung des PIC-Loaders im Assembler-Menue)
   PIC-Programmierschnittstelle

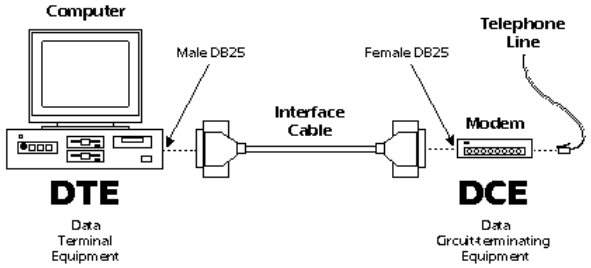
Sowohl die Signal-Bedeutungen als auch die -Konfigurationen als Eingang (I) oder Ausgang (O) des physischen Interfaces sind spezifisch für den aktiven Modus. Folgende Tabelle gibt einen Überblick. Wenn man von der linken Seite des PC-G850 auf das Interface schaut, befindet sich Pin-1 links und Pin-11 rechts.

Pin #	SIO-Modus		SSIO/PWM-Modus		PIO-Modus		PIC-Modus	
	Signal	I/O	Signal	I/O	Signal	I/O	Signal	I/O
1	-	-	-	-	-	-	-	-
2	VCC(+5V)	-	VCC(+5V)	-	VCC(+5V)	-	VCC(+5V)	-
3	GND	-	GND	-	GND	-	GND	-
4	RTS	O	BUSY	O	Bit0	I/O	CP	O
5	DTR	O	DOUT	O	Bit1	I/O	CLK#	O
6	RXD	I	XIN	I	Bit2	I/O	DATAIN	I
7	TXD	O	XOUT	O	Bit3	I/O	DATAOUT	O
8	CD	I	DIN	I	Bit4	I/O	LOWBATT#	I
9	CTS	I	ACK	I	Bit5	I/O	-	-
10	DSR	I	EX1	I	Bit6	I/O	-	-
11	CI	I	EX2	I	Bit7	I/O	-	-

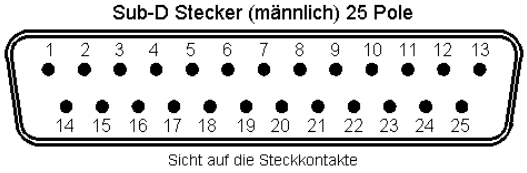
Es werden nun zunächst der SIO-Modus und entsprechende Anschlussmöglichkeiten genauer beschrieben. Das serielle Interface des PC-G850V(S) im SIO-Modus verwendet die Signale des RS-232C Standards (allerdings mit TTL-Pegel – s.u).

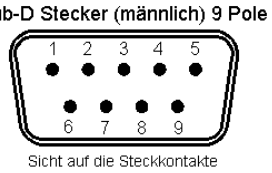
SIO-Modus: RS-232 Standard und Konventionen

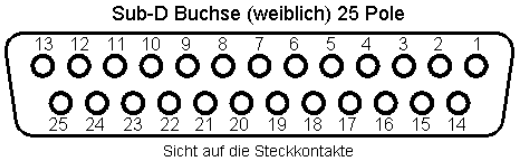
Der RS-232 Standard verwendet die Begriffe DTE (Data Terminal Equipment) und DCE (Data Communication Equipment). Das DTE ist dabei z.B. der PC-G850 und das DCE ein Modem oder eine andere Peripherie (serieller Drucker).

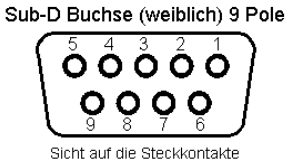


Sollen zwei Computer direkt (d.h. ohne Modem) verbunden werden, ist ein Nullmodem-(Kabel) erforderlich, welches die Eingänge des einen DTE mit den Ausgängen des anderen verbindet und umgekehrt („gekreuzte“ Signale).
Allgemein sind 25-polige (Sub-D 25 / DB-25) oder 9-polige (Sub-D 9 / DB-9) Stecker und Buchsen zur Verbindung von RS-232-fähiger Peripherie gebräuchlich.









Die Pin-Belegungen, Signale und ihre Bedeutungen sind in der folgenden Tabelle zusammengefasst.

Signal	Name	Alternativ Bezeich-nung	Richtung (DTE-Sicht)	Bedeutung	Pin# DB-9	Pin# DB-25
TXD	Transmitted Data	SD	Out	Daten vom DTE zum DCE	3	2
RXD	Received Data	RD	In	Daten vom DCE zum DTE	2	3
RTS	Request To Send (Ready To Send)	RS	Out	DTE erbittet Sendeerlaubnis vom DCE	7	4
RTR	Ready To Receive			DTE kann Daten empfangen		
CTS	Clear To Send	CS	In	DCE kann Daten empfangen	8	5
DTR	Data Terminal Ready	ER	Out	DTE Schnittstelle bereit	4	20
DSR	Data Set Ready	DR	In	DCE Schnittstelle bereit	6	6
CD	Carrier Detect		In	DCE detektiert fernes DCE (z.B. Telefonverbindung)	1	8
CI	Call Indicator	RI	In	Anruf eines fernen DCE	9	22
GND	Signal Ground	SG	None	Signal-Referenzmasse	5	7
FG	Frame Ground	PG	None	Abschirmung	-	1

Hinweis:
Ca. Ende der 1980er Jahre hat sich ein Wechsel in der Interpretation des RTS-Signals durchgesetzt:
In der ursprünglichen Bedeutung fragt das DTE (Computer) das DCE (Modem), ob das DTE senden darf und das DCE „antwortet" entsprechend via CTS. Dieses Protokoll ist jedoch asymmetrisch und das DTE hat keine Möglichkeit, dem DCE zu signalisieren, dass es warten soll, wenn das DCE sendet. Aus diesem Grund wurde „Request To Send" neu interpretiert: Das DTE „bittet" das DCE, Daten zu senden – anders ausgedrückt: Das DTE ist empfangsbereit (Ready To Receive – RTR). RTR und CTS sind damit unabhängig voneinander und das Protokoll zwischen DTE und DCE symmetrisch. Die Bezeichnung „Request To Send" (RTS) wurde jedoch in der Regel beibehalten, zumal sie hinreichend uneindeutig ist.

Der PC-G850V(S) implementiert die neuere, symmetrische RTR-Semantik, die Signalbezeichnung ist jedoch RTS – im Gegensatz zum Vorgängermodell PC-E500(S), welcher die originäre RTS-Bedeutung implementiert und daher zusätzlich auf das XON/XOFF-Protokoll angewiesen ist, wenn er Daten/Programme von einem PC empfängt. Das DTR-Signal wird vom PC-G850V(S) auf HIGH gesetzt, wenn das SIO-Interface aktiv ist, der DSR-Eingang wird jedoch nicht ausgewertet. Es gibt also kein DTR/DSR-Handshake. Das RTS/CTS-Handshake, oder ersatzweise das XON/XOFF-Protokoll, kann im TEXT/Sio/Format-Submenü unter dem Eintrag "flow" konfiguriert werden.

SIO-Modus: Signalpegel

Die folgende Tabelle zeigt die Signalpegel und die verwendete Logik des RS-232 Standards im Vergleich zum PC-G850V(S).

	Level	Pegel	Bedeutung für Datensignale (RXD,TXD)	Bedeutung für Kontrollsignale (RTS,CTS,etc.)
RS-232	LOW	-15V bis -3V	1 (Mark), Idle, Stop	Inaktiv
	HIGH	+3V bis +15V	0 (Space), Start	Aktiv
UART-TTL	LOW	0V	0 (Space), Start	Aktiv
	HIGH	+3,3V / +5V	1 (Mark), Idle, Stop	Inaktiv
PC-G850V(S)	LOW	0V	1 (Mark), Idle, Stop	Inaktiv
	HIGH	5V	0 (Space), Start	Aktiv

Der PC-G850V(S) verwendet also, wie fast alle anderen SHARP Pocket-Computer auch, invertierte UART-TTL-Pegel und –Logik. D.h. die Logik ist identisch zum RS-232 Standard (HIGH=0/Aktiv), aber die Pegel sind TTL-Level.

⇨ **Sollen Peripheriegeräte an den PC-G850 angeschlossen werden, die mit RS-232 Pegeln arbeiten, ist zwingend ein Pegel-Konverter erforderlich!**

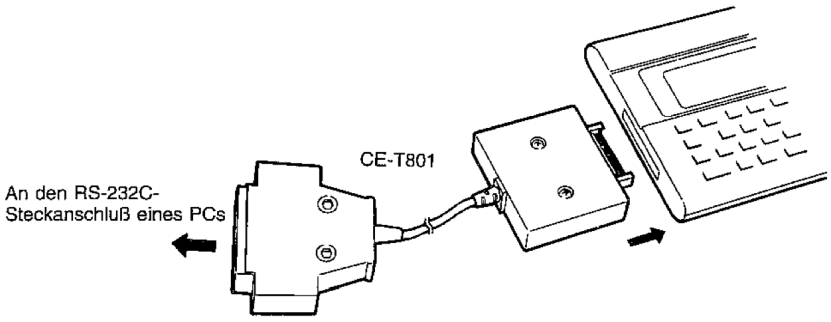
Der Status der Signale TXD und RTS im SIO-Modus ist außer in den folgenden Fällen undefiniert:
(1) Das Interface ist im SIO-Modus in der BASIC-Betriebsart geöffnet (`OPEN"COM:"`)
(2) R- oder W-Befehle werden in der Monitor-Betriebsart ausgeführt.
(3) Daten werden in der TEXT-Betriebsart über SIO übertragen.

SIO-Modus: Datenübertragungskabel CE-T800 und CE-T801

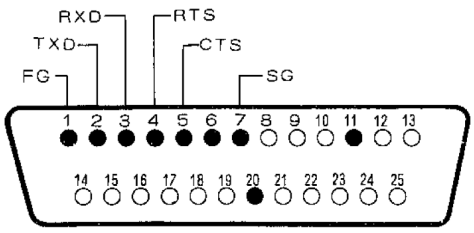
Die Datenübertragungskabel CE-T800 und CE-T801 sind serielle Kommunikationskabel, mit deren Hilfe der PC-G850V(S) mit einem Personal Computer oder einem anderen seriellen Gerät kommunizieren kann. Mit diesen Kabeln können Daten, Programm-Quellcode oder Maschinensprachen-Programme unter Verwendung der SIO-Funktion der TEXT-Betriebsart oder den SIO-Befehlen des Monitors zu und von einem PC übertragen werden.

Die Datenübertragungskabel sind mit einem RS-232 Pegelkonverter sowie einer internen Nullmodem-Schaltung ausgestattet. Der DB-25 Stecker kann entweder (ggf. per DB-9 Adapter) an den physischen COM-Port eines PC angeschlossen werden, sofern ein solcher vorhanden ist, oder mittels eines Seriell-USB-Adapters an einen USB-Port. Ein Nullmodem-Kabel bzw. -Baustein darf dabei nicht verwendet werden.

Soll hingegen der Plotter CE-515P/516P oder ein anderes Peripherie-Gerät über das CE-T800/801 angeschlossen werden, so *muss* eine Nullmodem-Schaltung in den Signalweg eingefügt werden, um das integrierte Nullmodem des CE-T800/801 zu neutralisieren.



Stiftbelegung des CE-T801-Steckers ⟨DB-25(W)⟩



Stift 6 und 20 sind verbunden.

Beim Datenübertragungskabel CE-T800 ist der Stift 6 und 20 nicht belegt. Stift 11 ist in beiden Varianten unbelegt.

Vorsicht: Niemals die Steckstifte des Datenübertragungskabels mit bloßen Händen berühren. Statische Entladungen des Körpers können die internen Schaltkreise beschädigen.

Für Datenübertragung von einem Personal Computer muss ein Arbeitsbereich von etwa 300 Byte zur Verfügung stehen.

SIO-Modus: USB PC-Adapterkabel mit Hardware-Handshake

Eine elegante, leistungsfähige und dennoch einfache Selbstbau-Alternative zur Verbindung des PC-G850V(S) mit einem modernen PC basiert auf der Verwendung eines vorgefertigten USB-UART Adapter Kabels mit offenem Ende.

Spezifikation:
FTDI USB-UART/TTL Adapter Kabel mit FT232R-Chip, 5V, 6-polig (GND,5V,RXD,TXD,RTS,CTS)

Als Steckverbinder für das 11-polige Interface des PC-G850V kann eine handelsübliche Stiftleiste mit 2,54mm Rastermaß verwendet werden. Die Adern des UART-Endes des Adapterkabels müssen dann nur noch in Nullmodem-Schaltung mit der Stiftleiste verlötet werden:

FTDI-UART Signal (Farbe)	PC-G850V(S) Signal (Pin)
GND (schwarz)	GND (3)
RXD (gelb)	TXD (7)
TXD (orange)	RXD (6)
CTS (braun)	RTS (4)
RTS (grün)	CTS (9)
VCC (rot)	-

Zusätzlich sollte auf der Steckerseite noch ein 10KOhm Widerstand zwischen Pin 4 und Pin 3 verbaut werden. Dieser dient als sog. Pulldown für das RTS-Signal, um in jedem Fall einen definierten LOW-Pegel zu erzeugen. Ohne diesen kann es zu I/O-Fehlern bei der Datenübertragung vom PC zum PC-G850V(S) kommen, weil der PC Warteanforderungen des Pocket-Computers ggf. nicht erhält.

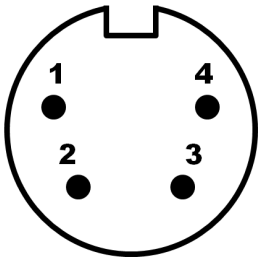
Anschließend laden Sie das Tool FT_PROG von der Webseite des Herstellers: www.ftdichip.com. Mit diesem Werkzeug müssen die Signale RXD, TXD, RTS und CTS des FTDI-Chips logisch invertiert werden, da die serielle Schnittstelle des PC-G850V(S) mit *invertierter* UART-Logik arbeitet (s.o.). Dieser Vorgang ist einmalig durchzuführen, die Einstellungen werden im integrierten EEPROM des FTDI-Chips dauerhaft gespeichert.

SIO-Modus: RS-232-Drucker

Über den SIO-Modus lassen sich auch Drucker ansteuern, die über eine RS-232C Schnittstelle verfügen, wie z.B. der 4-Farb-Plotter CE-515P bzw. CE-516P.

Schließen Sie einen RS-232-Drucker niemals ohne Pegelkonverter an den PC-G850V(S) an! Sie können das Datenübertragungskabel CE-T800/801 in Kombination mit einer nachgelagerten Nullmodem-Schaltung verwenden. Für den CE-515P/516P wird ein DIN-4 Stecker in folgender Beschaltung (Nullmodem inklusive) benötigt:

DIN-4 Stecker		CE-T800/1 DB-25 Stecker	
Pin#	Signal	Pin#	Signal
1	+12V	-	-
2	BUSY#	4	RTS
3	GND	7	GND
4	DATA#	3	RXD



DIN-4 Stecker, Sicht auf die Kontakte

Achten Sie auf die korrekte Konfiguration der DIP-Schalter auf der Rückseite des CE-515P/516P (Details siehe Handbuch des Druckers).
Für eine erfolgreiche Kommunikation mit dem CE-515P müssen im TEXT/Sio/Format-Menue des PC-G850V(S) folgende Parameter eingestellt werden:

- baud rate = 1200
- data bit = 8
- stop bit = 1
- parity = none
- end of line = CR
- flow = RS/CS

Um einen RS-232-Drucker anzusteuern, muss das 11-Pin Interface explizit im SIO-Modus geöffnet (`OPEN"COM:"`) und abschließend wieder geschlossen werden (`CLOSE`). Die zu druckenden Zeichenketten oder Steuerzeichen werden über den Befehl `PRINT#1,"<string>"` an den Drucker gesendet.

```
OPEN"COM:"
PRINT#1,"HELLO WORLD"
...
CLOSE
```

Die Kommandos `LPRINT`, `LLIST`, `LFILES` werden im SIO-Modus *nicht* auf das 11-Pin Interface umgeleitet.

SSIO-Modus

Der SSIO-Modus dient der *synchronen*, seriellen Datenübertragung – im Gegensatz zur *asynchronen* seriellen Datenübertragung des SIO-Modus. „Synchron“ bedeutet, dass der Sender zusätzlich ein Taktsignal (Strobe/Clock) aussendet, an dem sich der Empfänger ausrichtet. Dadurch wird u.a. die Vorgabe einer beidseitig einzuhaltenden Baud-Rate obsolet. Die SIO-Parameter im TEXT/Sio/Format-Menü haben daher im SSIO-Modus keine Relevanz.

Der SSIO-Modus des PC-G850V(S) besitzt verschiedene Untermodi bzw. Protokolle.

SSIO-Modus: CE-126P Druckerprotokoll

Dieser Submodus ist das Default-Protokoll für das 11-Pin Interface des PC-G850V(S). Es ist immer dann aktiv, wenn die Schnittstelle in keinem der anderen (Sub-)Modi geöffnet ist. Es dient der Ansteuerung des Druckers CE-126P: Kommandos wie LPRINT, LLIST und LFILES werden an den Drucker übertragen.

Das integrierte Kassetten-Interface des CE-126P läßt sich ebenfalls über den PC-G850V ansteuern. Das entsprechende Protokoll besitzt das gleiche Handshake wie das Druckerprotokoll, aber die eigentliche Datenübertragung erfolgt über PWM anstatt SSIO (s.u.).

Die Signale des 11-Pin Interface des PC-G850V(S) haben beim CE-126P Druckerprotokoll folgende Bedeutung:

Pin #	Signal	Richtung	Funktion
4	BUSY	Out	Taktgeber (Clock) für die synchrone, serielle Datenübertragung
5	DOUT	Out	Datenleitung
6	XIN	In	Keine Funktion
7	XOUT	Out	HIGH: CE-126P Sub-Device Select (Drucker vs. Kassetten-Interface) bzw. Kommando-Übertragung, LOW:Pause oder Datenübertragung
8	DIN	In	Keine Funktion
9	ACK	In	Empfangsbereitschaft des CE-126P (Handshake)
10	EX1	In	Keine Funktion
11	EX2	In	Keine Funktion

Das folgende Diagramm zeigt den zeitlichen Signalverlauf für den Befehl LPRINT"X" bei angeschlossenem CE-126P:

Der PC-G850V(S) wartet jeweils auf das ACK-Signal, bevor BUSY auf HIGH gesetzt wird. Dieses synchrone, serielle Protokoll entspricht dem der Druckerschnittstelle des PC-E500(S).

SSIO-Modus: LPRT-Protokoll und Mini-I/O-Port

Der Mini-I/O-Port des PC-G850V(S) wird gebildet durch die logische Aufteilung der sechs Hauptsignale des SSIO-Modus in zwei Gruppen mit je drei Signalen/Bits:

- Mini-I/O Port „Schreiben“ (Output)
  - XOUT (Mini-I/O Output-Port Bit-0)
  - DOUT (Mini-I/O Output-Port Bit-1)
  - BUSY (Mini-I/O Output-Port Bit-2)
- Mini-I/O Port “Lesen” (Input)
  - ACK (Mini-I/O Input-Port Bit-0)
  - DIN (Mini-I/O Input-Port Bit-1)
  - XIN (Mini-I/O Input-Port Bit-2)

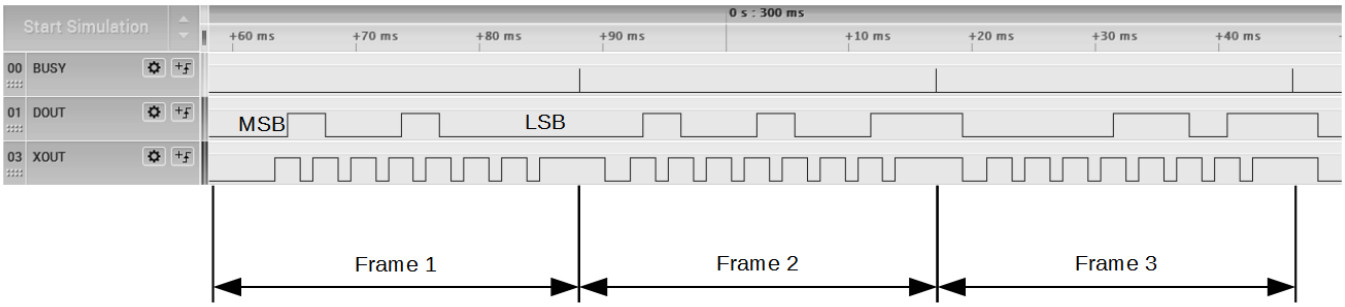
Die Bits des Mini-I/O Ports lassen sich durch die Funktionen `OUT/miniput()` und `INP/miniget()` einzeln steuern, so dass auf dieser Basis bspw. individuelle Kommunikationsprotokolle implementiert werden können.

Daneben bietet der PC-G850V(S) ein synchrones, serielles Protokoll zur Datenübertragung an entsprechende externe Peripherie. Dazu muss das 11-Pin Interface mit `OPEN("LPRT:")` geöffnet werden. Die Datenströme der Befehle LPRINT, LLIST und LFILES werden dann im ASCII-Code über dieses Protokoll versendet.

Die Bedeutung der Signale des LPRT-Protokolls ist dabei wie folgt:

Pin #	Signal	Richtung	Funktion
4	BUSY	Out	Rahmenende-Indikator nach jedem Byte
5	DOUT	Out	Datenleitung
7	XOUT	Out	Taktgeber mit Pause nach jedem Byte
9	ACK	In	LOW: Empfänger ist bereit HIGH: PC-G850V(S) muss warten

Folgendes Diagramm zeigt die zeitlichen Signalverläufe:



Daten werden Byte-weise mit dem höchstwertigen Bit zuerst (MSB-first) an den steigenden Flanken des Taktsignals übertragen. Das BUSY-Signal liefert zusätzlich einen Begrenzungsrahmen (Frame) für jedes Byte.

PWM-Modus: CE-126P Bandprotokoll

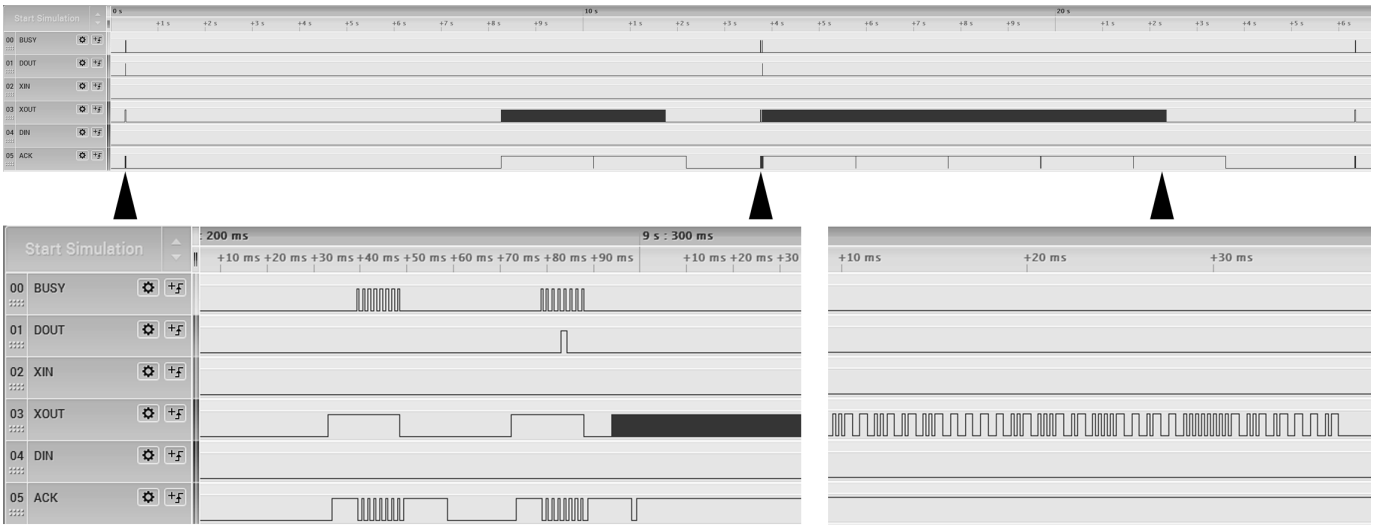
Dieses Protokoll wird durch die Befehle BSAVE, BSAVEM, BLOAD, BLOADM, BLOAD? bei angeschlossenem CE-126P (oder kompatiblem Kassetten-Interface) aktiviert. Es dient dem Speichern und Laden von BASIC-Programmen oder binären Daten (wie z.B. Maschinenprogrammen) mittels eines Kassettenrecorders (CE-152).

Das Handshake des CE-126P Bandprotokolls und Druckerprotokolls ist identisch, aber das Bandprotokoll verwendet Pulsweitenmodulation (das digitale Äquivalent analoger Wellenformen) zur eigentlichen Datenübertragung. Es handelt sich also um eine Mischung aus SSIO- und PWM-Protokoll.

Hier die Bedeutung der Signale des CE-126P Bandprotokolls:

Pin #	Signal	Richtung	Funktion
4	BUSY	Out	Taktgeber für synchrones, serielles Handshake
5	DOUT	Out	Datenleitung für Handshake
6	XIN	In	PWM-codierte Daten vom Kassetten-Interface (Laden)
7	XOUT	Out	Handshake: Siehe CE-126P Druckerprotokoll Daten: PWM-codierte Daten zum Kassetten-Interface (Speichern)
8	DIN	In	Keine Funktion
9	ACK	In	Empfangsbereitschaft des CE-126P (Handshake)

Das folgende Diagramm zeigt die Signalverläufe des CE-126P Bandprotokolls einer BSAVE-Ausführung (Speichern eines einzeiligen BASIC-Programmes):



CE-126P Handshake PWM Data Transfer

Die Dynamik einer BLOAD-Ausführung is äquivalent mit dem Unterschied, dass die Daten via PWM auf dem XIN-Signal empfangen werden.

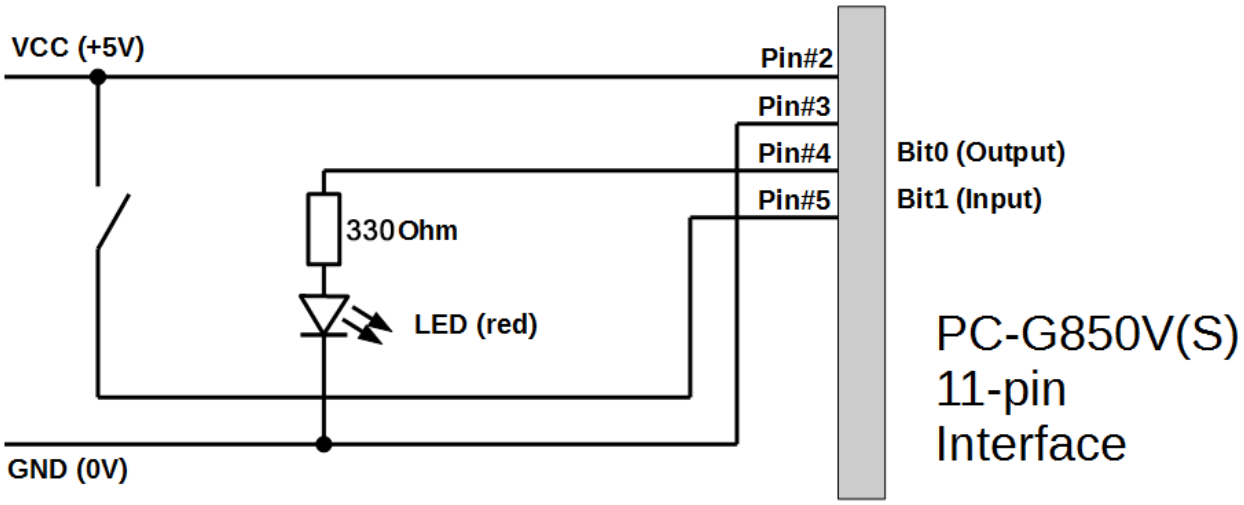
PWM-Modus: Generisches PWM-Protokoll

Dieses Protokoll entspricht dem CE-126P Bandprotokoll reduziert auf XOUT und XIN (d.h. das Handshake entfällt). Es wird durch die Befehle BSAVE, BLOAD etc. aktiviert, wenn *kein* CE-126P (oder kompatibles Kassetten-Interface) angeschlossen ist. Dies ist typischerweise der Fall, wenn zwei PC-G850V(S) mittels Datenaustauschkabel (EA-129C) verbunden sind. Gleich zu Beginn einer BSAVE/BLOAD-Operation setzt der PC-G850V(S) den XOUT-Ausgang auf HIGH und beobachtet anschließend kurz den ACK-Eingang. Wird dieser von der Gegenseite *nicht* auf HIGH gesetzt, wird das generische PWM-Protokoll aktiviert, andernfalls das CE-126P Bandprotokoll-Handshake.

PIO-Modus

Der PIO-Modus des PC-G850V(S) ist weniger zur Datenkommunikation als vielmehr zur digitalen Hardwaresteuerung gedacht. Mit Hilfe dieser Funktionalität verwandeln Sie den Pocket Computer in einen Mikrocontroller.
Das 11-Pin Interface wird zu einem programmierbaren 8-Bit Port, dessen logische Pegel (LOW/HIGH) über die PIO-API (API = Application Programming Interface) in BASIC oder C gesetzt oder gelesen werden können - je nachdem, ob das entsprechende Signal als Eingang oder als Ausgang konfiguriert wurde. Die Konfiguration der Richtung für jedes Signal/Bit erfolgt über die Funktion `pioset/PIOSET` (vgl. Referenzteil). Die Funktion `pioput/PIOPUT` setzt die einzelnen Signalpegel (Signal(n)=LOW <=> Bit(n)=0, Signal(n)=HIGH <=> Bit(n)=1), wobei Signale, die als Eingang konfiguriert wurden, ignoriert werden. Die Funktion `pioget/PIOGET` liest entsprechend die acht Signalpegel des PIO-Ports und fasst sie in einem Byte zusammen.

Das folgende, sehr einfach gehaltene Beispiel soll die Möglichkeiten und die korrekte Verwendung demonstrieren.



Bit-0/Pin-4 soll in diesem Szenario als Ausgang dienen, der im aktiven Zustand (HIGH) eine LED zum Leuchten bringt. Bit-1/Pin-5 hingegen wird als Eingang konfiguriert und soll den Zustand eines Tasters repräsentieren. Ein offener Eingang (d.h. nicht definierter Eingangspegel) wird als logisch 0 interpretiert. Genau dies ist der Fall, wenn der Taster geöffnet ist. Um den geschlossenen Zustand davon unterscheiden zu können, ist der Taster auf der anderen Seite mit VCC (d.h. HIGH/logisch 1) und nicht mit GND verbunden.

Ziel der Software-Steuerung in diesem Beispiel soll es sein, bei einer Betätigung des Tasters die LED anzuschalten und bei der folgenden Betätigung wieder auszuschalten. Wir wollen hier die Programmiersprache C verwenden, die Programmierung in BASIC ist ähnlich, aber weniger strukturiert.

Folgender Code bildet das gewünschte Szenario ab:

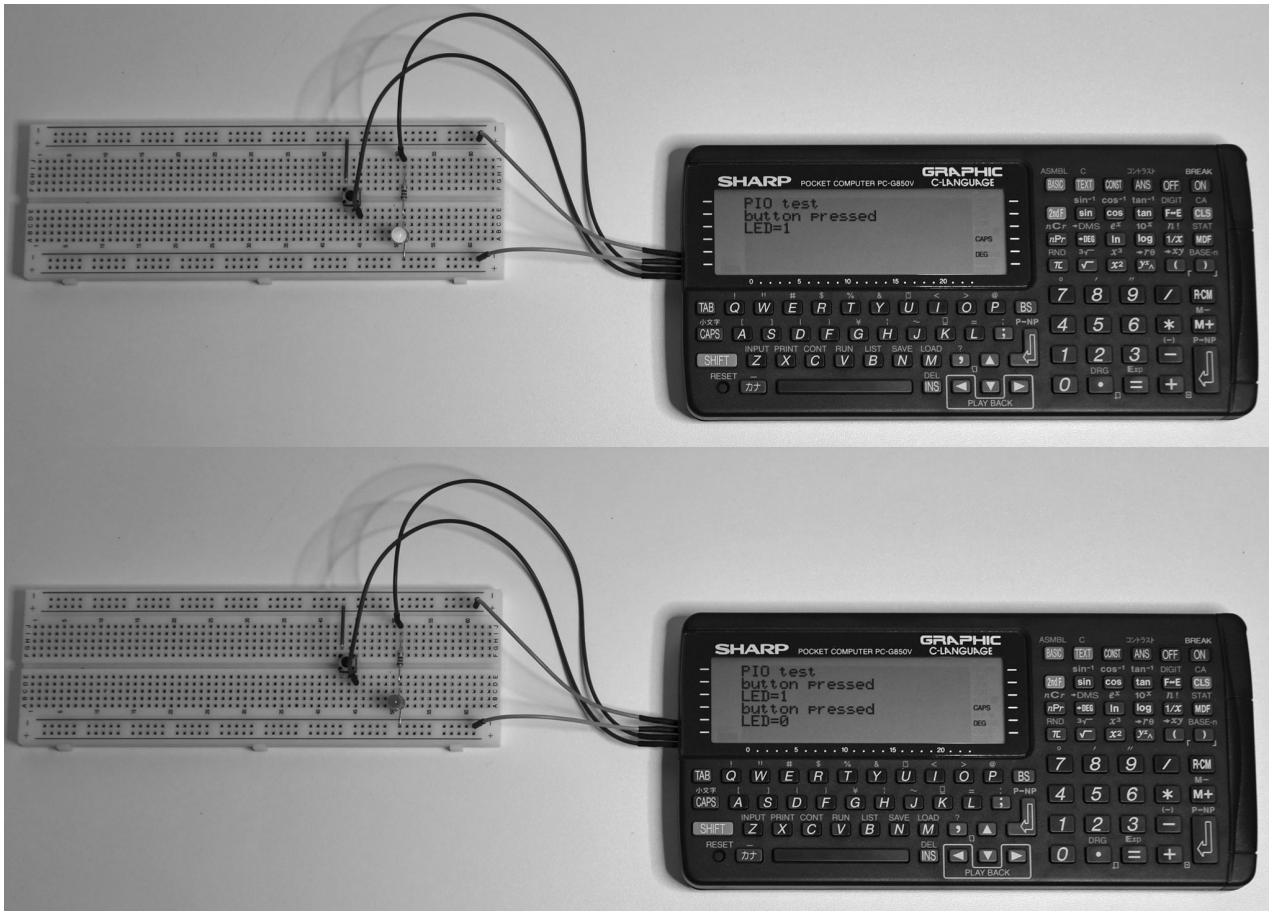
```
1   #define BOOL  char
2   #define TRUE  1
3   #define FALSE 0
4   #define BTN   0x02
5
6   char BTNstate = 0;
7   char LEDstate = 0;
9
10  BOOL setupPIO() {
11   if(!fopen("pio","a+")) {
12    printf("can't open port\n");
13    return FALSE;
14   }
15   pioset(BTN);
16   return TRUE;
17  }
19
20  BOOL pressed() {
21   BOOL rtn=FALSE;
22   char btn;
23   btn=pioget()&BTN;
24   if(btn && BTNstate==0)
25    rtn=TRUE;
26   BTNstate=btn;
27   return rtn;
28  }
29
30  toggleLED() {
31   LEDstate=!LEDstate;
32   printf("LED=%x\n",LEDstate);
33   pioput(LEDstate);
34  }
39
100 main() {
101  printf("PIO test\n");
102  if(!setupPIO())
103   abort();
104  while(TRUE) {
105   if(pressed()){
106    printf("button pressed\n");
107    toggleLED();
108   }
109  }
110 }
```

Das Symbol '\' wird beim PC-G850V(S) als SHIFT-G im TEXT-Modus eingegeben und als ¥ auf dem Display dargestellt.

Hinweise zum Code:

- Zeile 4: Maske für Bit-1 (0b00000010), d.h. Taster-Eingang
- Zeile 6: Globale Zustandsvariable für den Taster
- Zeile 7: Globale Zustandsvariable für die LED
- Zeile 11: Das Interface wird im PIO-Modus für Lesen und Schreiben geöffnet.
- Zeile 15: Bit-1 = Pin#5 wird als Eingang konfiguriert, alle anderen Signale als Ausgang.
- Zeile 20: Diese Funktion detektiert die Transition von Bit-1=0 auf Bit-1=1, d.h. das Schließ-Ereignis des Tasters.
- Zeile 23: Der PIO-Port wird gelesen und alle Bits außer Bit-1 maskiert (ausgeblendet).
- Zeile 30: Diese Funktion wechselt den Zustand der LED
- Zeile 33: Der neue LED-Zustand (Bit-0: 0 o. 1) wird auf dem Port ausgegeben/gesetzt. Die nicht benutzten Ausgänge erhalten den Wert 0.
- Zeile 104: Hauptschleife, Abbruch mit der ON/BREAK-Taste

Die folgende Abbildung zeigt exemplarisch den Versuchsaufbau und die Ausgaben auf dem Display des PC-G850V.



PIC-Modus

Der PIC-Modus des PC-G850V(S) dient der Übertragung eines assemblierten PIC-Programms (vgl. Kapitel 12) auf einen PIC-Mikrocontroller und wird ausschließlich durch den PIC-Loader im Assembler-Menue aktiviert. Der PC-G850V(S) unterstützt dabei das serielle ICSP (In-Circuit Serial Programming) Protokoll der PIC16F8x-Familie und kompatibler Modelle.

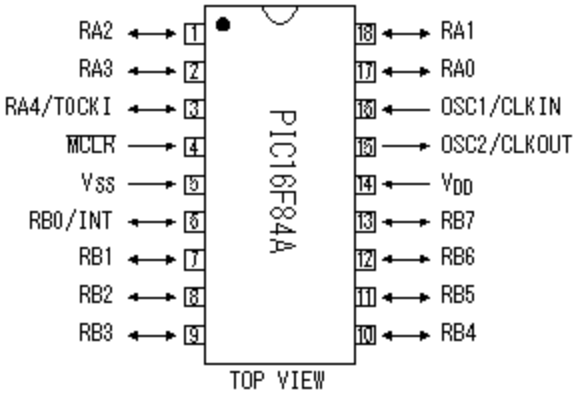
Die Signale des 11-Pin Interfaces im PIC-Modus haben folgende Funktion:

Pin #	Signal	Richtung	Funktion
4	CP	Out	Dieses Signal steuert den Programmiermodus des PIC. Während das Signal HIGH ist, muss die Brennspannung (12-14V) am MCLR# Eingang des PIC angelegt werden. Ist dieses Signal LOW, so muss MCLR# auf GND oder VDD (+5V) liegen.
5	CLK#	Out	Dieses Signal ist der Taktgeber (Clock) für die serielle Kommunikation zwischen dem PC-G850V(S) und dem zu programmierenden PIC im ICSP-Modus. Es muss invertiert (d.h. als CLK) am RB6 Eingang des PIC anliegen. Dieser übernimmt Daten-Bits an der fallenden Flanke des CLK-Signals.
6	DATAIN	In	Dieser Eingang muss mit dem RB7-Pin des PIC verbunden werden. Über diese Leitung werden Daten vom PIC zur Verfikation der Programmierung seriell gelesen.
7	DATAOUT	Out	Dieser Ausgang dient der seriellen Kommando- und Daten-Übertragung zum PIC im ICSP-Modus. Er muss ebenfalls mit dem RB7-Pin des PIC verbunden werden.
8	LOWBATT#	In	Dieses Signal kann an eine Spannungsüberwachungs-Schaltung angeschlossen werden (insb. bei externer Programmierspannung sinnvoll). LOW wird als zu schwache Spannungsversorgung interpretiert.

Gemäß Spezifikation der PIC16F8x Familie wechselt der PIC in den ICSP-Programmiermodus, sobald folgende Bedingungen gleichzeitig erfüllt sind:

- VDD = +5V, VSS = GND
- MCLR# = 12-14V
- RB6 (CLK) = LOW.
- RB7 (DATA) = LOW.

Folgende Abbildung zeigt stellvertretend die Pin-Belegung des PIC16F84A:

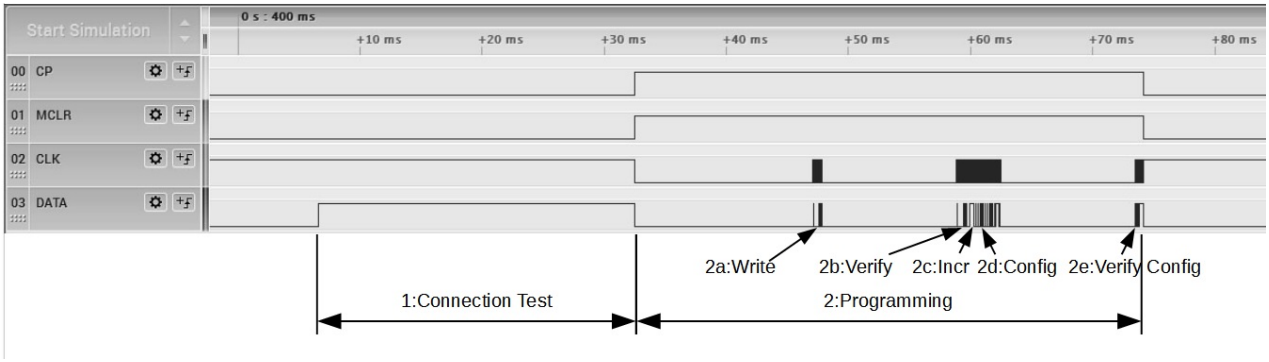


Der PIC-Loader des PC-G850V(S) unterstützt wie gesagt das ICSP-Protokoll, aber diesem geht noch eine Verbindungsprüfung voraus. Schlägt sie fehl, kommt es zu der Fehlermeldung `Connection error!`

Zur Vertiefung der dynamischen Sicht des Programmiervorganges wollen wir ein minimales Programm für den PIC16F84A betrachten. Es besteht lediglich aus dem Konfigurationswort und einer Endlosschleife ohne Rumpf:

```
10 #include ”p16f84a.inc”
20 __config 0x3ff6
30loop goto loop
```

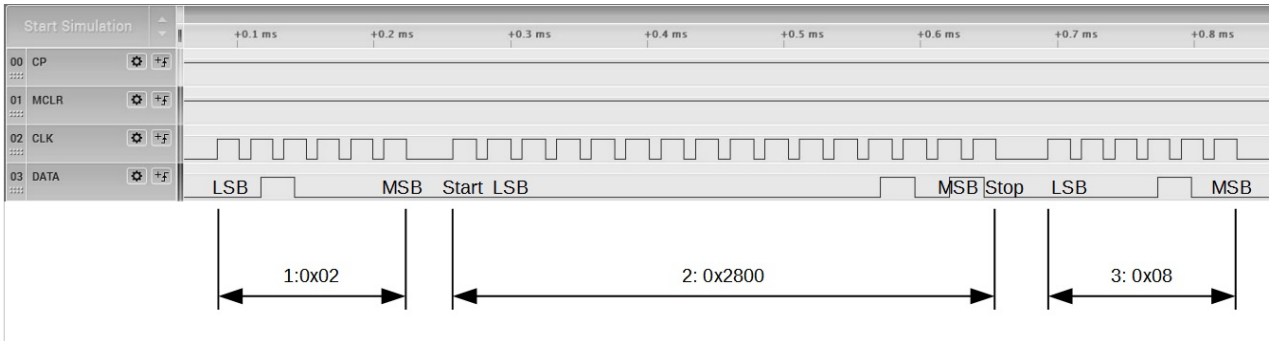
Der PIC-Assembler übersetzt dieses Quell-Programm in ein Maschinenprogramm, welches genau ein PIC-Wort (= 14-Bit) groß ist. Ein erfolgreicher Brennvorgang mit Hilfe des PIC-Loaders des PC-G850V(S) besteht dann aus folgenden Phasen:



CLK wurde hier am RB6-Eingang des PIC abgegriffen (d.h. das bereits invertierte CLK# Signal). Der DATA-Abgriff ist der RB7-Pin. CP steuert direkt die Brennspannung MCLR#.

1. Verbindungstests: Der PC-G850V setzt DATAOUT auf HIGH und prüft, ob DATAIN entsprechend auch auf HIGH wechselt. Die beiden Signale müssen also verbunden sein, ansonsten wird das ICSP-Protokoll nicht aktiviert!
2. Dies ist die ICSP-Phase. Sie wird eingeleitet durch MCLR# = 12,5V, CLK = LOW, DATA = LOW.
    a. Schreiben des assemblierten 14-Bit Wortes (PIC-Maschinenbefehl)
    b. Verifikation (Lesen) des geschriebenen PIC-Maschinenbefehls
    c. Inkrementierung des PIC-Programmspeicher-Zeigers (würde das binäre PIC-Programm aus mehr als einem Wort bestehen, wiederholten sich die Schritte 2a/b/c für jedes weitere Wort)
    d. Schreiben des Konfigurationswortes
    e. Verifikation des Konfigurationswortes

Eine Ausschnittvergrößerung der Phase 2a zeigt das folgende Diagramm:



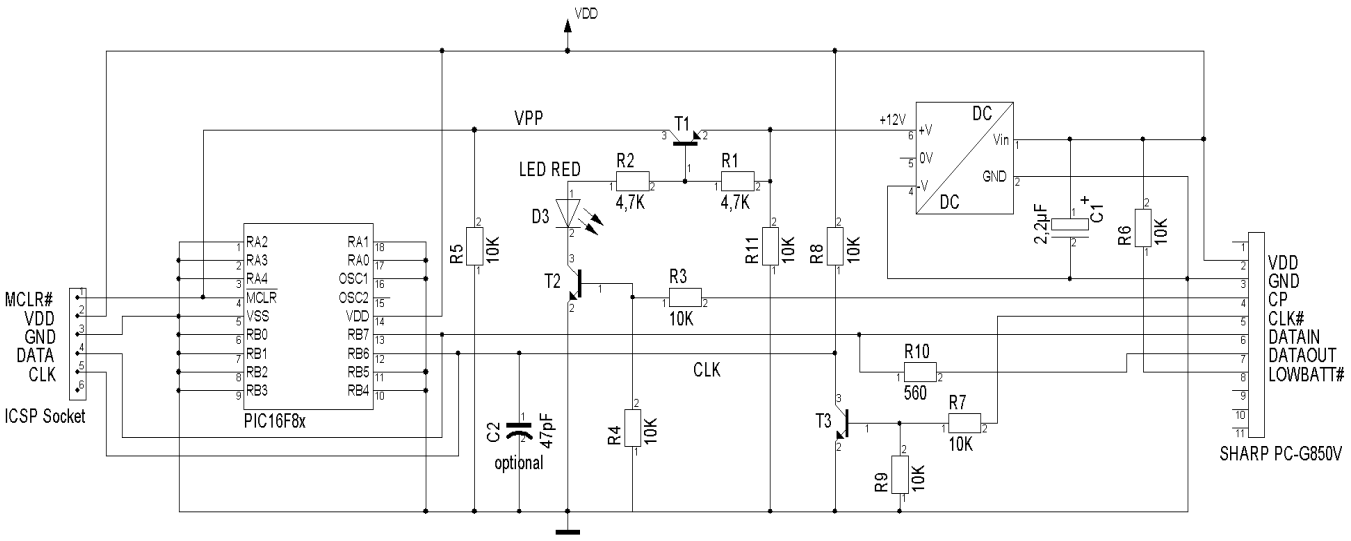
PIC-ICSP Kommandos umfassen 6-Bit und sind der Spezifikation des PIC zu entnehmen. Einem Kommando kann ein Datenwort folgen – lesend oder schreibend. Datenworte sind 14-Bit breit, werden aber von einem Start- und einem Stop-Bit umrahmt, so dass 16-Bit pro Wort übertragen werden. Die serielle Übertragung erfolgt prinzipiell mit dem niederwertigsten Bit zuerst (LSB-first). Die Bits werden, wie beschrieben, an der fallenden Flanke des CLK-Signals gültig (übernommen bzw. ausgegeben).

1. PIC-ICSP Kommando „Load Data for Program Memory“ (0x02).
2. Datenübertragung für das „Load Data for Program Memory“-Kommando. Das vom PIC-Assembler in diesem Beispiel erzeugte 14-Bit Wort lautet also 0x2800.
3. PIC-ICSP Kommando „Beginn Programming Cycle“ (0x08). Dieses Kommando besitzt keinen Datenparameter und startet den Brennvorgang für das zuletzt übertragene (gepufferte) 14-Bit Wort.

Es wird nun eine geeignete PIC-Brenner-Schaltung für das 11-Pin Interface des PC-G850V(S) benötigt, die den integrierten PIC-Loader nutzt und die PIC16F8x-Familie unterstützt. Eine solche Schaltung muss u.a. folgende Kriterien erfüllen:

1. Das CP-Signal muss die Brennspannung am PIC steuern.
2. DATAIN und DATAOUT müssen bereits während der Verbindungsprüfung elektrisch verbunden sein.
3. Das CLK# Signal muss invertiert am RB6-Pin des PIC anliegen.
4. Das CLK# Signal ist sehr empfindlich gegen Störungen (Crosstalk) – insb. durch DATOUT – und muss ggf. geeignet abgeschirmt und/oder entstört werden. Außerdem wird ein Pulldown-Widerstand für einen definierten LOW-Pegel benötigt.
5. Der LOWBATT# Eingang sollte entweder an eine Überwachungs-Schaltung für die Programmierspannung angeschlossen werden, oder sie muss konstant auf HIGH liegen.

Folgende Schaltung realisiert diese Anforderungen. Eine Besonerheit ist, dass sie keine externe Spannungsquelle für die Brennspannung benötigt, sondern diese mittels eines DC/DC-Konverters aus der Versorgungsspannung (+5V) erzeugt:



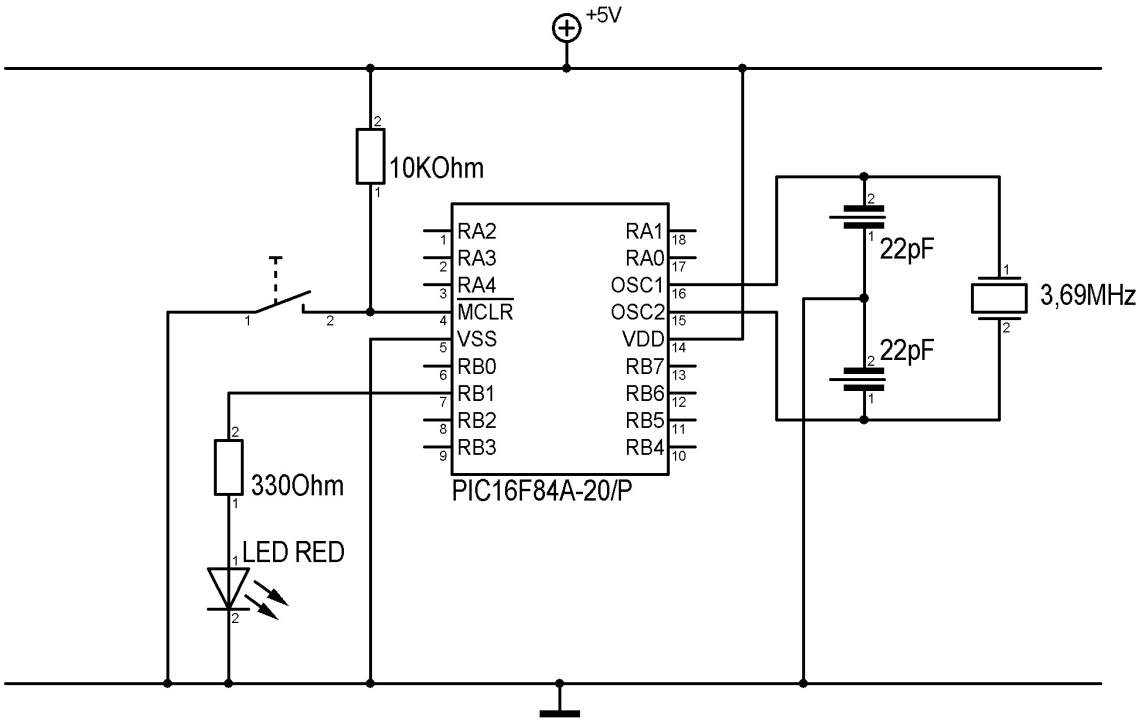
Ein +5V auf +12V DC/DC-Konverter (z.B. TMA0512C oder ~D) kann zur Erzeugung der Brennspannung verwendet werden. Das CP-Signal steuert diese als VPP am MCLR#-Pin über die Transistorstrecke T1, T2. Die LED dient als Indikator für den ICSP-Modus. Über T3 und R8 wird das CLK#-Signal invertiert und liegt als CLK am PIC an. C2 ist optional und dient als Tiefpassfilter der Entstörung des CLK Signals sofern nötig. Die Spannungsüberwachung schlägt hier nur an, falls die Versorgungsspannung während des Brennvorganges unter den Schwellwert für den LOW-Pegel (= logisch 0) fällt.

Zum Test des gesamten PIC Programmierzyklus mit dem PC-G850V(S) eignet sich ein einfaches Programm, welches eine am Pin RB1 des PIC angeschlossene LED blinken lässt:

```
10 #include "p16f84a.inc"
20 __config 0x3ff1 ;CP_OFF & PWRT_ON & WDT_OFF & XT_OSC
30DELAY1 equ 0x08 ;delay counter 1
40DELAY2 equ 0x09 ;delay counter 2
50 org 0
99
100start
110 bsf STATUS,RP0 ;change to bank 1
120 bcf TRISB,1 ;enable RB1 for output
130 bcf STATUS,RP0 ;back to bank 0
140loop
150 bsf PORTB,1 ;RB1=1,LED=on
160 call delay
170 bcf PORTB,1 ;RB1=0,LED=off
180 call delay
190 goto loop
299
300delay
310 movlw 255
320 movwf DELAY1
330 movwf DELAY2
340dloop
350 decfsz DELAY1,f
360 goto dloop
370 decfsz DELAY2,f
380 goto dloop
390 return
```

Geben Sie das Programm im TEXT-Modus ein und übersetzen Sie es mit dem integrierten PIC-Assembler. Schließen Sie nun einen PIC16F84A mit der oben angegebenen (oder einer äquivalenten) PIC-Brenner-Schaltung an das 11-Pin Interface des PC-G850V(S) an. Aktivieren Sie anschließend den PIC-Loader im Assembler-Menue (vgl. Kapitel 12).

Lösen Sie nach erfolgreichem Programmiervorgang den PIC vom Brenner und installieren Sie ihn in folgender Testschaltung:



Diese Testschaltung verwendet einen externen Quarz < 4MHz als Taktgeber. Entsprechend ist XT_OSC im Konfigurationswort (0x3FF1) des Beispielprogramms gesetzt (s. Spezifikation d. PIC16F84A).

Sofern der PIC korrekt mit dem Beispielprogramm "gebrannt" wurde, beginnt die LED zu blinken, sobald eine Spannungsquelle (+5V) an diese Testschaltung angeschlossen wird. Der Taster ist optional und führt bei Betätigung den PIC in den RESET-Zustand (MCLR# = LOW). Die Programmausführung wird dadurch gestoppt und die LED erlischt. Die Blinkfrequenz wird Hardware-seitig durch die Quarz-Frequenz und Software-seitig durch die Vorgabe der Iterationen der äußeren Delay-Schleife (Zeile 310, Wertebereich = 1 .. 255) beeinflusst.

Anhang B: Tastenfunktion

Taste	Beschreibung
BREAK ON	• Zum Einschalten der Stromversorgung, auch wenn das Gerät durch die AUTO OFF-Funktion abgeschaltet wurde, • Beim Drücken dieser Taste während der Ausführung von einem Programm entspricht sie der BREAK- Taste und führt zur Unterbrechung der Ausführung. • Beim Drücken dieser Taste während einer direkten Eingabe wird die Ausführung von Befehlen wie LOAD oder BSAVE unterbrochen. • In der TEXT- und der C-Betriebsart geht der Computer wieder auf das Hauptmenü oder das Menü zurück.
OFF	• Schaltet das Gerät aus
BASIC	• Schaltet/wechselt in den BASIC-RUN- oder BASIC-PRO-Modus
SHIFT + ASMBL	• Schaltet in den Assembler-, CASL- oder PIC-Modus
TEXT	• Schaltet in das Hauptmenu des TEXT-Modus
SHIFT + C TEXT	• Wechselt in das Menu des C-Compilers
SHIFT + コントラスト	• Einstellung des Bildschirm-Kontrastes
SHIFT	• Eine der gelben Tasten, die mit "SHIFT" bzw. "2ndF" • markiert sind, muss gedrückt (und im Falle von "SHIFT" • gehalten) werden, um die zweite Funktion einer Taste zu aktivieren(angezeigt direkt über der Taste).
小文字 CAPS	• Zum Ein- und Ausschalten der Betriebsart für Großbuchstaben. Das Symbol CAPS erscheint auf der Anzeige. • Wenn das CAPS-Symbol angezeigt wird, werden alle Buchstaben als Großbuchstaben eingegeben. Wenn CAPS gedrückt wird, verschwindet das CAPS-Symbol wieder und Kleinbuchstabenwerden eingegeben. • Standardmäßig ist CAPS nach dem Einschalten des Geräts eingeschaltet. • Bei aktiviertem Kanji-Modus wird zwischen den großen und den kleinen Zeichen gewechselt.
カナ	Ein- und Ausschalten des Kanji-Modus
TAB	• Zum Bewegen des Cursors auf die Tabulatorposition. In den Betriebsarten BASIC-RUN und PRO wird der Cursor in Abständen von sieben Stellen bewegt. In der TEXT-EDITOR-Betriebsart wird der Cursor beim ersten Drücken um acht Stellen, beim zweiten Drücken um sechs Stellen und bei jedem weiteren Druck um sieben Stellen weiterbewegt.
►	• Bewegt den Cursor nach rechts. • Führt Wiedergabe-Anweisungen aus. • Zur Darstellung des Cursors, wenn er nicht angezeigt wird, während ein Inhalt angezeigt wird, • Löschen eines Fehlers bei der direkten Eingabe.
◄	• Bewegt den Cursor nach links. Ansonsten gleich der Taste ►
ANS	• Zum Abrufen des letzten Ergebnisses.
CONST	• Zur Eingabe einer Konstanten und von Operatoren für Berechnungen mit Konstanten (die Anzeige "CONST" erscheint auf dem Display). Durch

	Drücken von 2ndF+CONST ((SHIFT+CONST) wird die gegenwärtig gespeicherte Konstante angezeigt.
INS	• Schaltet die Eingabe in den Einfüge-Modus. Nach dem Einschalten ist standardmäßig der Einfüge-Modus abgeschaltet.
SHIFT + DEL	Löscht das Zeichen, auf dem sich der Cursor gerade befindet.
BS	• Löscht das Zeichen direkt links vom Cursor.
2nd F	• "2ndF" muss gedrückt werden um die zweite Funktion einer Taste zu aktivieren (angezeigt direkt über der Taste).
CLS	• Löschen des Bildschirmeingabebereichs • Löschen eines angezeigten Fehlers •
SHIFT + CA	• Zum Löschen der Anzeige auf dem Display und gleichfalls zum Zurückstellen des Computers auf den Anfangsstatus. • Rückstellung der WAIT-Zeiteinstellung. • Rückstellen des Anzeigeformats (USING-Format) • Rückstellen des TRON-Status (TROFF). • Rückstellen der Fehlerbedingung
↵	• Eingabe einer Programmzeile in den Computer beim Schreiben von Programmen. • Fragt nach manueller Berechnung oder direkter Ausführung einer Befehlsanweisung. • Zum Wiederaufnehmen eines Programms, welches mit dem INPUT-Befehl kurzfristig unterbrochen wurde.
SHIFT + P↔NP	• Zur Einstellung der Betriebsart für Drucken oder kein Drucken, wenn ein zusätzlicher Drucker angeschlossen ist.

Die beiden Tasten  haben die entsprechenden Funktionen, abhängig von der Betriebsart und dem Status des Computers, die in der folgenden Tabelle aufgelistet werden.

Betriebsart	Status		
RUN	Programm wird ausgeführt	Keine Funktion	
	Unterbrochen durch den STOP-Befehl oder die BREAK-Taste.	Ausführung der folgenden Zeile und Unterbrechen.	Gedrückt halten, um die auszuführende bzw. bereits ausgeführte Programmzeile zur Anzeige zu bringen.
	Fehlerbedingung während der Ausführung eines Programms.	Keine Funktion	Gedrückt halten, um eine fehlerhafte Zeile anzuzeigen.
	Trace-Modus ON.	Ausführen eines Programms im Trace-Modus.	Gedrückt halten, um die auszuführende bzw. ausgeführte Programmzeile zur Anzeige zu bringen.
PRO (Bei Änderung in die PRO-Betriebsart und fehlender Darstellung der Programmzeilen.)			
	Das Programm wird zeitweise unterbrochen.	Anzeige der unterbrochenen Zeile.	Wie links.
	Fehlerbedingung	Anzeige der fehlerhaften Zeile	Wie links.
	Andere Bedingung.	Anzeige der ersten Zeile	Anzeige der letzten Zeile
PRO (wenn Programmzeilen angezeigt werden)			
		Anzeige der folgenden Programmzeile.	Anzeige der vorherigen Prcgrarmmzeile.

Anhang C: Rechenbereiche

Numerische Berechnungen:
Für Rechenoperationen mit x muß x in einem der unten aufgeführten Bereiche liegen.

$-1 \times 10^{100} < x \leq -1 \times 10^{-99}$ für negatives x
$10^{-99} \leq x < 10^{100}$ für positives x
x=0
Der angezeigte Wert für x wird von der Anzahl der möglichen Stellen auf dem Display begrenzt.

Funktioen:

Befehl	Bereich von x	Funktion
SIN x COS x TAN x	DEG: $\|x\| < 1 \times 10^{10}$ RAD: $\|x\| < \frac{\pi}{180} * 10^{10}$ GRAD: $\|x\| < \frac{\pi}{9} * 10^{10}$ Ebenso nur für tan x: (n=ganzzahlig) DEG: $\|x\| \neq 90 (2n — 1)$ RAD: $\|x\| \neq \frac{\pi}{9} (2n — 1)$ GRAD: $\|x\| \neq 100 (2n — 1)$	sin x cos x tan x
ASN x ACS x	$-1 \leq x \leq 1$	$\sin^{-1} x$ $\cos^{-1} x$
ATN x	$\|x\| < 1 \times 10^{100}$	$\tan^{-1} x$
HSN x HCS x HTN x	$-227.9559242 \leq x \leq 230,2585092$	sinh x cosh x tanh x
AHS x	$\|x\| < 1 \times 10^{50}$	$\sinh^{-1} x$
AHC x	$1 \leq x < 1 \times 10^{50}$	$\cosh^{-1} x$
AHT x	$\|x\| < 1$	$\tanh^{-1} x$
LN x LOG x	$1 \times 10^{-99} \leq x < 1 \times 10^{100}$	$\ln x = \log_e x$
EXP x	$-1 \times 10^{100} < x \leq 230.2585092$	e^x e≈2.718281828
TEN x	$-1 \times 10^{100} < x < 100$	10^x
RCP x SQU x CUB X SQR x CUR x	$\|x\| < 1 \times 10^{100}$ $x \neq 0$ $\|x\| < 1\ 10^{50}$ $\|x\| < 2.154434690 \times 10^{33}$ $0 \leq x < 1 \times 10^{100}$ $\|x\| < 1 \times 10^{100}$	$\frac{1}{x}$ x^2 x^3 $\sqrt{x}$ $\sqrt[3]{x}$
y^x	wenn y > 0, $-1 \times 10^{100} < x \log y < 100$ wenn y = 0, x > 0	$y^x = 10^{x \times \log y}$

	wenn y < 0,x = ganzzahlig oder wenn 1/x = ungeradzahlig (x≠0) und -1 x 10^{100} < x log |y| < 100	
&H x	0 ≤ x ≤ 2540BE3FF (x in hexadezimal) FDABF41C01 ≤ x ≤ FFFFFFFFFF	
POL (x,y) $(x, y \rightarrow r, \theta)$	$(x^2+y^2) < 1 \times 10^{100}$ $\frac{x}{y} < 1x10^{100}$	$r = \sqrt{x^2 + y^2}$ $\theta = tan^{-1}\frac{y}{x}$
REC (r, θ) $(r, \theta \rightarrow x, y)$	$r < 1\times10^{100}$ $\|r \sin\theta\| < 1\times10^{100}$ $\|r \cos\theta\| < 1\times10^{100}$	$x = r\,cos\theta$ $y = r\,sin\theta$
NPR (n,r)	$\frac{n!}{(n-r)!} < 1\times10^{100}$ $0 \le r \le n \le 9999999999$ n,r ganzzahlig	${}_nP_r$
NCR (n,r)	$\frac{n!}{(n-r)!r!} < 1\times10^{100}$ $0 \le r \le n \le 9999999999$ n,r ganzzahlig wenn n - r < r, n – r ≤ 69 wenn n - r ≥ r, r ≤ 69	${}_nC_r$
FACT x	0 ≤ x ≤ 69	n!
DEG x	|x| < 1 x 10^4	DMS->DEG
DMS x	|x| < 1 x 10^4	DEG->DMS

Statistische Berechnungen

|x| < 1 x 10^{50}	1 ≤ n < 1 x 10^{100}	|y| < 1 x 10^{50}
$\sum x$	$\sum x^2$	$\sum y$	$\sum y^2$
$\bar{x} = \frac{\sum x}{n}$		$\bar{y} = \frac{\sum y}{n}$	
$sx = \sqrt{\sum \frac{x^2 - n\bar{x}^2}{n-1}}$		$sy = \sqrt{\sum \frac{y^2 - n\bar{y}^2}{n-1}}$	
$\sigma x = \sqrt{\sum \frac{x^2 - n\bar{x}^2}{n}}$		$\sigma y = \sqrt{\sum \frac{y^2 - n\bar{y}^2}{n}}$	
$a = \bar{y} - b\bar{x}$		$b = \frac{Sxy}{Sxx}$	
$r = \frac{Sxy}{\sqrt{Sxx \times Syy}}$		$Sxx = \sum x^2 - \frac{(\sum x)^2}{n}$	
$x' = \frac{y-a}{b}$		$Syy = \sum y^2 - \frac{(\sum y)^2}{n}$	
$y' = a + bx$		$Sxy = \sum xy - \frac{\sum x \times \sum y}{n}$	

Anhang D: Technische Daten

Gerät:	PC-G850V(S)
Prozessor:	8-Bit CMOS CPU (entspricht Z80)
Speicherkapazität:	Systembereich: 2.3KB Bereich der festen Variablen 208 Byte Bereich für Programme/Daten 30179 Byte
Stapelspeicher:	Funktionsstapel: 16 Puffer Datenstapel: 8 Puffer Subroutinenstapel: 10 Puffer Stapelspeicher für Basic: insgesamt 90 Byte: • REPEAT~UNTIL : 4 Bytes pro Instanz • WHILE-WEND : 5 Bytes pro Instanz • SWITCH-CASE : 6 Bytes pro Instanz • FOR-NEXT : 18 Bytes pro Instanz (Bei SWITCH-CASE ist nur eine Instanz erlaubt)
Operatoren:	Addition, Subtraktion, Multiplikation und Division Funktionen: Trigonometrische Funktionen und invers trigonometrische Funktionen, logarithmische und Exponentialfunktionen, Winkel Umwandlung, Quadrat und Quadratwurzel, Potenz, Vorzeichen, Integer, absolute Zahlen, ganze Zahlen, Zeichen-Funktion, pi, Koordinaten-Umwandlung und andere
Numerische Präzision:	10 Stellen (Mantisse) + 2 Stellen (Exponent)
Berechnungsmethode:	Prioritätserkennung in Funktionen
Editierfunktionen:	Cursor nach rechts und links, Zeile nach oben und unten, Einfügen von Zeichen, Löschen von Zeichen TEXT-Editor, Monitor für die Maschinensprache des Z80
Interface-Möglichkeiten:	Sharp-11Pin-Interface: CE-126P (Drucker), CE-T800 (PC-Datenübertragungskabel), EA-129C (Verbindungskabel zwischen 2 Sharp-Rechnern)
Display:	6 Zeilen, 24 Zeichen, Zeichenformat 5x7 Grafik: 48x144 Bildpunkten
Betriebstemperatur:	0°C — 40°C

Stromversorgung:	Vier mal Typ AAA-Trockenbatterien (R03), ca 90 Betriebsstunden. 6V Gleichstrom 0.2W über externe Stromversorgung. (z.B. EA-23E)
Stromverbrauch:	0.2W
Abmessungen:	196 (B) x 95 (T) x 20 (H) mm
Gewicht:	270g (G850VS: 260g)
Mitgeliefertes Zubehör:	

Anhang E: Rücksetzen des Computers

Wenn es ein Problem mit dem Computer gibt, z.B. durch fehlerhafte Programme, kann ein Rücksetzen des Computers helfen.

1. Die ON-Taste drücken und dann den Reset-Taster unter der SHIFT-Taste mit einem Kugelschreiber oder einem ähnlichen Gerät eindrücken. Den Reset-Taster dann wieder loslassen.

2. Direkt nach dem Drücken des RESET-Tasters zeigt der PC-G850V die folgende Anzeige. Falls irgendeine andere Anzeige erscheint, muss der obige Vorgang wiederholt werden.
Der PC-G850V fragt nach der Bestätigung zum Löschen des Speichers:

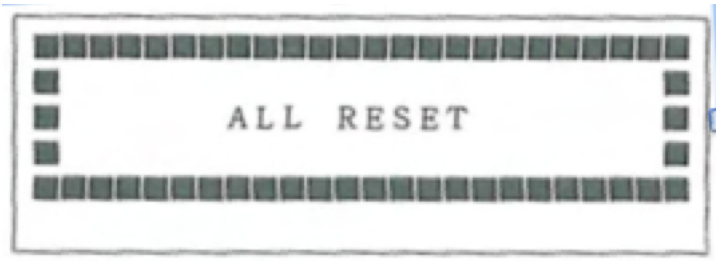
```
MEMORY CLEAR O. K. ?  (Y/N)
```

3. Wenn Sie die Daten behalten wollen drücken sie die Taste N

```
RUN MODE
>
```

Falls danach der Computer immer noch nicht einwandfrei arbeitet, kann man den Computer mit dem Löschen aller Daten auf die Werkseinstellung zurücksetzen. Dazu die Punkte 1 und 2 wiederholen. Danach mit dem folgenden Punkt 4 weiter fortfahren:

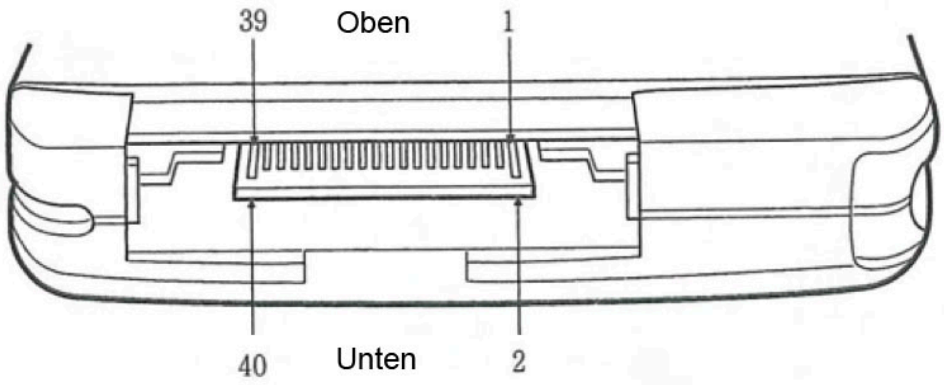
4. Die Taste Y drücken. Die folgende Meldung blinkt auf und zeigt damit an, daß der Computer initialisiert wurde und alle Speicherinhalte gelöscht sind.



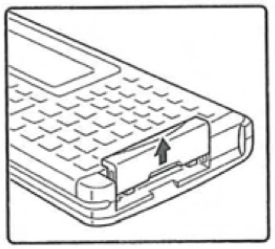
3. Eine beliebige Taste drücken. Folgende Anzeige erscheint:

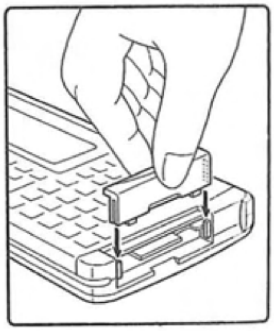
Anhang F: Systembus

Rechte Seite des Computers



Oben		Unten	
Signalbezeichnung	Pin	Pin	Signalbezeichnung
Vcc	1	2	Vcc
$\overline{M1}$	3	4	$\overline{MREQ}$
$\overline{IORQ}$	5	6	IORESET
$\overline{WAIT}$	7	8	$\overline{INT1}$
$\overline{WR}$	9	10	$\overline{RD}$
BNK1	11	12	BNK0
$\overline{CEROM2}$	13	14	CERAM2
D7	15	16	D6
D5	17	18	D4
D3	19	20	D2
D1	21	22	D0
A15	23	24	A14
A13	25	26	A12
A11	27	28	A10
A9	29	30	A8
A7	31	32	A6
A5	33	34	A4
A3	35	36	A2
A1	37	38	A0
GND	39	40	GND





Hinweis: Je nach Akkuleistung wird die Spannung bei Vcc zwischen 4-6V liegen. Da der Rechner aus CMOS-Bauelementen besteht müssen die CMOS-üblichen Pegel eingehalten werden.

Anhang G: Kanji-Umwandlungstabelle

	A		I		U		E		O	
ア行	A	ア	I	イ	U	ウ	E	エ	O	オ
							YE	イェ		
カ行	KA	カ	KI	キ	KU	ク	KE	ケ	KO	コ
	CA	カ			CU	ク			CO	コ
	QA	クァ	QI	クィ	QU	ク	QE	クェ	QO	クォ
	KYA	キャ	KYI	キィ	KYU	キュ	KYE	キェ	KYO	キョ
サ行	SA	サ	SI	シ	SU	ス	SE	セ	SO	ソ
	SHA	シャ	SHI	シ	SHU	シュ	SHE	シェ	SHO	ショ
	SYA	シャ	SYI	シィ	SYU	シュ	SYE	シェ	SYO	ショ
タ行	TA	タ	TI	チ	TU	ツ	TE	テ	TO	ト
	TSA	ツァ	TSI	ツィ	TSU	ツ	TSE	ツェ	TSO	ツォ
	CHA	チャ	CHI	チ	CHU	チュ	CHE	チェ	CHO	チョ
	TYA	チャ	TYI	チィ	TYU	チュ	TYE	チェ	TYO	チョ
	CYA	チャ	CYI	チィ	CYU	チュ	CYE	チェ	CYO	チョ
ナ行	NA	ナ	NI	ニ	NU	ヌ	NE	ネ	NO	ノ
	NYA	ニャ	NYI	ニィ	NYU	ニュ	NYE	ニェ	NYO	ニョ
ハ行	HA	ハ	HI	ヒ	HU	フ	HE	ヘ	HO	ホ
	FA	ファ	FI	フィ	FU	フ	FE	フェ	FO	フォ
	HYA	ヒャ	HYI	ヒィ	HYU	ヒュ	HYE	ヒェ	HYO	ヒョ
マ行	MA	マ	MI	ミ	MU	ム	ME	メ	MO	モ
	MYA	ミャ	MYI	ミィ	MYU	ミュ	MYE	ミェ	MYO	ミョ
ヤ行	YA	ヤ	YI	イ	YU	ユ			YO	ヨ
ラ行	RA	ラ	RI	リ	RU	ル	RE	レ	RO	ロ
	LA	ラ	LI	リ	LU	ル	LE	レ	LO	ロ
	RYA	リャ	RYI	リィ	RYU	リュ	RYE	リェ	RYO	リョ
	LYA	リャ	LYI	リィ	LYU	リュ	LYE	リェ	LYO	リョ
ワ行	WA	ワ							WO	ヲ
ン	N		N SHIFT + Ü		M					

	A		I		U		E		O	
ア行	VA	ヴァ	VI	ヴィ	VU	ヴ	VE	ヴェ	VO	ヴォ
ガ行	GA	ガ	GI	ギ	GU	グ	GE	ゲ	GO	ゴ
	GYA	ギャ	GYI	ギィ	GYU	ギュ	GYE	ギェ	GYO	ギョ
ザ行	ZA	ザ	ZI	ジ	ZU	ズ	ZE	ゼ	ZO	ゾ
	JA	ジャ	JI	ジ	JU	ジュ	JE	ジェ	JO	ジョ
	JYA	ジャ	JYI	ジィ	JYU	ジュ	JYE	ジェ	JYO	ジョ
	ZYA	ジャ	ZYI	ジィ	ZYU	ジュ	ZYE	ジェ	ZYO	ジョ
ダ行	DA	ダ	DI	ヂ	DU	ヅ	DE	デ	DO	ド
	DHA	デャ	DHI	ディ	DHU	デュ	DHE	デェ	DHO	デョ
	DYA	ヂャ	DYI	ヂィ	DYU	ヂュ	DYE	ヂェ	DYO	ヂョ
バ行	BA	バ	BI	ビ	BU	ブ	BE	ベ	BO	ボ
	BYA	ビャ	BYI	ビィ	BYU	ビュ	BYE	ビェ	BYO	ビョ
パ行	PA	パ	PI	ピ	PU	プ	PE	ペ	PO	ポ
	PYA	ピャ	PYI	ピィ	PYU	ピュ	PYE	ピェ	PYO	ピョ

Anhang H: Tabelle der Zeichencodes

		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	ヌル		スペース	0	@	P	'	p		⊥		ー	タ	ミ	═	×
1	1			!	1	A	Q	a	q	▁	⊤	。	ア	チ	ム	⊧	円
2	2			"	2	B	R	b	r	▂	⊣	⌐	イ	ツ	メ	⊭	年
3	3			#	3	C	S	c	s	▃	⊢	⌙	ウ	テ	モ	⊰	月
4	4			$	4	D	T	d	t	▄	¯	、	エ	ト	ヤ	◢	日
5	5			%	5	E	U	e	u	▅	―	・	オ	ナ	ユ	◣	時
6	6			&	6	F	V	f	v	▆	｜	ヲ	カ	ニ	ヨ	◥	分
7	7			'	7	G	W	g	w	▇	｜	ァ	キ	ヌ	ラ	◤	秒
8	8			(	8	H	X	h	x	▏	⌐	ィ	ク	ネ	リ	♠	"
9	9			)	9	I	Y	i	y	▎	¬	ゥ	ケ	ノ	ル	♥	
10	A			*	:	J	Z	j	z	▍	ㄴ	ェ	コ	ハ	レ	◆	
11	B			+	;	K	[	k	{	▋	⌟	ォ	サ	ヒ	ロ	♣	
12	C			,	<	L	¥	l	¦	▉	⌜	ャ	シ	フ	ワ	●	
13	D			−	=	M	]	m	}	▉	¬	ュ	ス	ヘ	ン	○	
14	E			.	>	N	^	n	~	▉	ㄴ	ョ	セ	ホ	゛	／	
15	F			／	?	O	_	o		+	⌟	ッ	ソ	マ	°	＼	

Diese Tabelle der Zeichencodes zeigt die Zeichen und ihre Zeichencodes, die mit den Befehlen CHR$ und ASC verwendet werden. Jeder Zeichencode besteht aus 2 hexadezimalen Zeichen (oder 8 binären Bit).

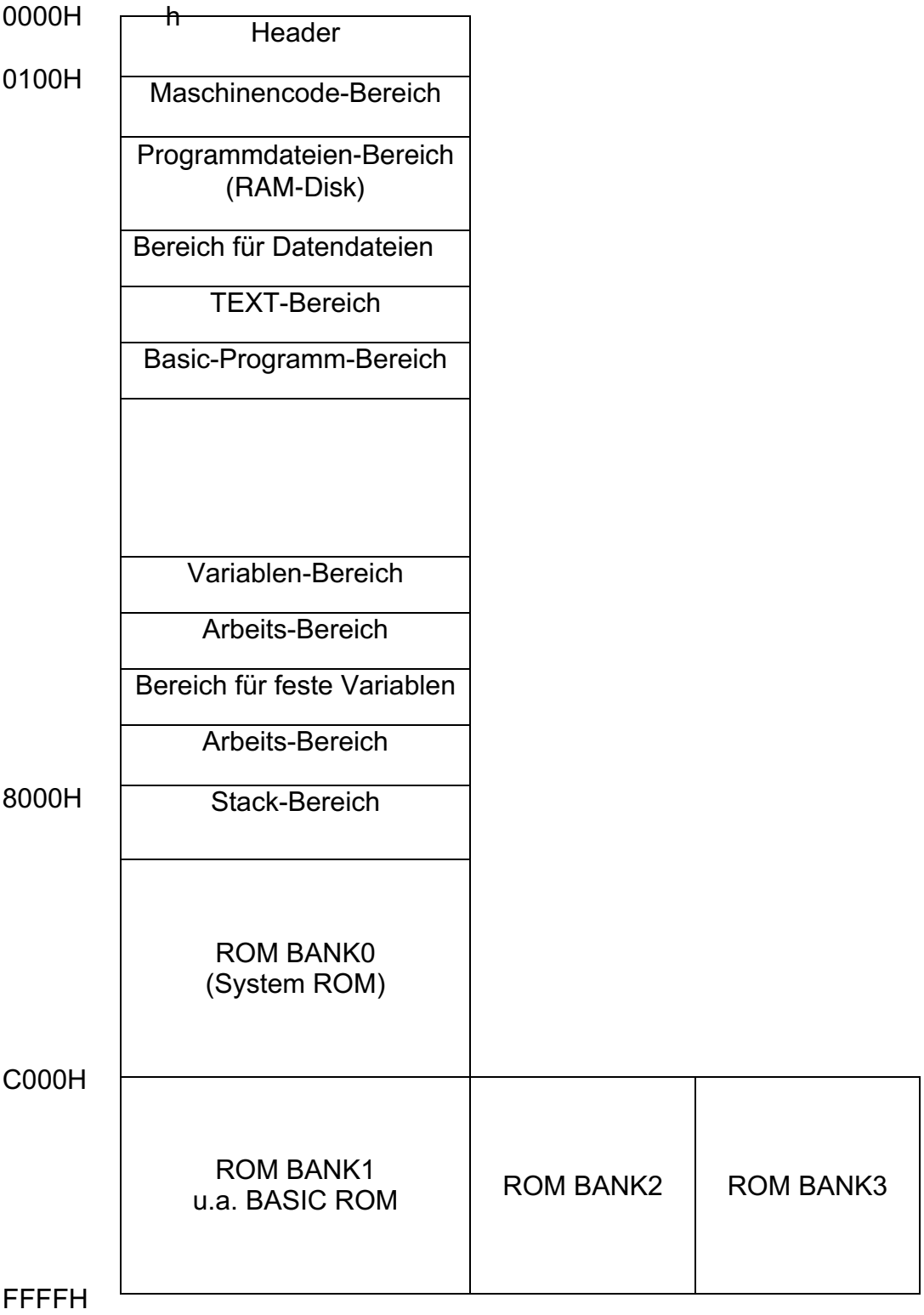
Das Zeichen "A" ist z.B. hexadezimal "41", dezimal "65" und binär "01000001". Das Zeichen "P" ist dezimal "80", hexadezimal "50" und binär "01010000".
Die Zeichencodes werden folgendermaßen dargestellt:

Beispiele:
Code für x
Hexadezimal &H2A Dezimal 42 (32 + l0)

Code für P Hexadezimal & H50 Dezimal 80
Hinweis:
Beim Drucken mit dem CE-126P werden die Zeichen mit den Codes 129(& H81)-159(&H9F), 224(&HE0)-231(&HE7), 236(&HEC)-240(&HF0), 245(&HF5)-248(&HF8) als Leerstellen gedruckt.

Anhang I: AUFTEILUNG DES SPEICHERBEREICHES

Speicherbereich:



Rambereich	Adress-Bereich
1) Header (reserved)	0 - 00FF
2) Machine language area	0100 - (7FFE,7FFF)-1
3) Ram file area	(7FFE,7FFF) -
4) Text Area	(7973,7974) - (7975,7976)
5) Basic program area	(79E1,79E2) - (79E3,79E4)
6) Basic stack	(79FC,79FD) - 77DF
7) System area	77E0 - 7FFF

Adressen in Klammern stellen die Adresse dar in der die jeweilige aktuelle Speicherposition hinterlegt ist.

Anhang J: ROM-ROUTINEN, I/O-Ports und ADRESSEN

ROM-Routinen:

Adresse	Bezeichnung	Beschreibung
84F7		Bildschirm eine Zeile hochrollen
871A		Default-Initialisierung des seriellen Interfaces (11-Pin)
9249		Springt zu einer Adresse in einer bestimmten Bank, die der CALL-Anweisung folgt
BCBE	STAT	Aufruf des STAT-Modus
BCDF		Liest ein Byte vom aktiven seriellen Interface nach A (wartet nur eine kurze von der Baudrate abhängige Zeit auf das Startbit)
BCE2		Liest ein Byte vom seriellen Interface nach A (wartet unendlich lange auf das Startbit)
BCE5		Liest ein Byte vom aktiven seriellen Interface nach A (wartet unendlich lange auf das Startbit)
BCE8		Öffnet (OPEN) das serielle Interface (11-Pin)
BCEB		Schließt (CLOSE) das serielle Interface (11-Pin)
BCEE		Fügt CR/LF an die Datei an, auf die Register HL zeigt
BCFI	CLRBAS	Startet die Routine für „BASIC DELETE OK?“
BCF7	CLRTXT	Startet die Routine für „TEXT DELETE OK?“
BCFD (88C1)	GETCHR	Liest ein Zeichen von der Tastatur in das Register A
BD00	LDPSTR	liest Pixelstring ab x-y-Position(in DE)der Länge B zur Adresse ab HL (x-Position in E, y-Position in D) x = 0-5 und y=0-23 1 Byte kodiert 7 übereinanderliegende Pixel und 5 Bytes ein Zeichen.
BD03	REGOUT	zeigt Werte aller Prozessorregister an und wartet auf Tastendruck
BD09	AOUT	zeigt Wert des A-Register und kann dann eingegeben werden
BD0F	HLOUT	zeigt Wert des HL-Register und kann dann eingegeben werden
BD15		Liest ein ASCII-String vom seriellen Interface nach HL bis EOT,EOL oder ein Fehler (SCF) erkannt wird.
BD2D	OFF	Power off (Schaltet den Rechner aus)
BE53 (89BE)	INKEY	Testet ob eine Taste gedrückt wurde und schreibt die Taste nach A (Inkey-Funktion). A=0 Keine Taste gedrückt A=52 mehrere Tasten wurden gleichzeitig gedrückt (carry-Flag wird gesetzt wenn eine Taste gedrückt wurde)
BE62 (8440)	PUTCHR	Gibt das Zeichen in Register A aus. DE definiert die x-y-Position (x-Position in E, y-Position in D)
BE65	INSLN	Legt eine leere Zeile auf der x-y-Position (in DE) an (x-Position in E, y-Position in D)

BFAF		Schreibt den Inhalt des Registers A zum seriellen Interface
BFB2		Schreibt einen String von HL zum seriellen Interface. Die Übertragung wir beendet wenn auf das Zeichen NULL gestoßen wird.
BFCD		Liest ein Zeichen von der Tastatur in das Register A. (wartet bis eine Taste gedrückt wird)
BFD0		Schreibt einen Pixelstring dessen Adresse in HL steht mit der Länge B. Die Ausgabe beginnt ab der x-y-Positionen in DE (x-Position in E, y-Position in D) x = 0-5 und y=0-23 1 Byte kodiert 7 übereinanderliegende Pixel und 5 Bytes ein Zeichen. Es erfolgt im Gegensatz zu den Routinen BFEE und BFF1 kein Zeilenumbruch
BFEE		Gibt das Zeichen in A ab x-y-Position in DE B-mal nacheinander aus. x = 0-5 und y=0-23
BFF1		Ausgabe des String der Länge B ab Adresse HL mit x-y-Position in DE. Der String wird gegebenenfalls am Zeilenende umgebrochen und am Displayende wird das LCD hochgescrollt (das gleiche Verhalten auch bei BFEE)
BFF4		Aufruf des RUN-Modus
C110		Power Off

BASIC-Routinen (nicht verifiziert):

Adresse	Bezeichnung	Beschreibung
C065		RAM (0000-003F) initialisieren
C0FD		Fragt nach ob der Speicher gelöscht werden soll.
D7C3		HL Zeigt auf das Basic-Byte. Der Token-String wird in DE übergeben
F9BD		Konvertiert den Inhalt von Register A zu 2 Hexzahlen auf die HL zeigt
FFF7		Dekodiert Basic Byte in B. Übergibt in A die Länge und in DE die Adresse des Strings

Adressen (nicht komplett verifiziert):

0000	Jump to BFFA
0030	Jump to BD03
0038	RET
0066	RETN
USER-Bereich+1A	Beginn der Ramdisk (in MONITOR ist USER änderbar (default USER=FF) Die Filelänge (immer 8+8 Bytes) in der Ramdisk steht in den beiden Bytes hinter dem Filenamen
779C	Kontrast. Eine Änderung bewirkt nicht gleich eine Veränderung der Anzeige.Beispiel: `10 PRINT "now:";PEEK(&H779C) 20 INPUT "change(0-31):";A 30 POKE &H779C,A 40 OUT &H40,&H80+A 50 GOTO 10`
77E0	Start des System-RAM-Bereichs
7800-78CF	Variablenbereich A-Z (7800=Z, je 7 Bytes
78E7-78E8	Startadresse des IO-Buffers
78EC	SIO Transmission mode bits: Bit 7: is received char EOT ? Bit 6: EOL matches (is complete) Bit 5: previous was CR ? Bit 4: check for EOL ?
78ED	Baudrate: 0x1=300, 0x2=600, 0x4=1200, 0x8=2400, 0x10=4800, 0x20=9600 highest bit starting from bit 5 is relevant. all bits 0 ==> 300 Baud
78EE	Parameter der seriellen Schnittstelle: Bit 2 add --> CR, else if BIT 0 add --> LF else add --> CR LF Bit 1: (set for CR LF), Bit 3: unused, Bit 4: 1+Bit4 Stopbits, Bit 5: 0=Odd 1=Even parity if parity enabled Bit 6: 0=no parity 1=parity check/generation enabled Bit 7: 7+Bit7 Databits
78EF	Byte zur Kennzeichnung des Übertragungsendes (EOT)
78F0	Auto-Power-Off-Pointer
7900	current bank-id mapped to C000-FFFF
7901-7904	Maske für Displaysymbole 7901: 00000111 | | + --- 1 immer | + ---- CAPS + ----- Cana
790D	VRAM Display Startposition. first LCD row offset (0-7) [enables simple scrolling]
790E	Nummer des zuletzt angewählten Files in der Ramdisk
7912-7913	Anfang Eintrag erstes File in der Ramdisk (Name)
7921	Aktuelle Cursorzeile (0-3)
7922	Aktuelle Cursorspalte (0-23)

7932	current interrupt mask at port 17H
7966	INKEY1, Abfrage des Tastaturcodes, siehe Key Matrix
7973-7974	Start Textbereich
7975-7976	Ende Textbereich
79B3-79B4	Basic-Pointer
79B5-79B6	Basic-Zeilenbereich, der gerade abgearbeitet wird
79B9	current Basic byte code
79C0-79C7	Password
79E1-79E2	Anfang ausführbares Basic-Programm
79E3-79E4	Ende ausführbares Basic-Programm
79FC-79FD	Unteres Ende Basic-Variablenbereich RAMTOP
79FE-79FF	Start ausführbares Basic-Programm
7A60-7A77	letzte Zeile CAL Rechenergebnis
7A80-7A98	letztes CAL Rechenergebnis (auf 11 Stellen genau)
7AA0-7AA1	Programmpointer
7AA2-7AA3	Programmpointer
7AA6-7AA7	gerade benutzte Variable (?)
7AB6-7AB7	FOR-Pointer (?)
7AB8-7AB9	Variablenzeiger
7AC8-7AC9	FOR-Pointer
7ACA-7ACB	Variablenzeiger
7ADC-7ADD	Variablenzeiger
7B00-7B5F	Zeichen im (Monitor-)Display
7BB0-7BC7	Displayzeile CAL
7C00-7CFF	Eingabepuffer, ausgewertet
7E00-7ED5	Basic-Stringpuffer 7E00: INKEY2, ASCII-Wert wie INKEY$
7EE8-	Eingabezeile
7F40-7F4B	LCD line scratch data (12 characters)
7FFD-	TOP of Stack (is decreased by PUSH) at most 178 bytes !!!
7FFE-7FFF	address of first non USER range, i.e. here USER+1 is stored

Display-Control-Ports 40h, 41h:
Die Low-Level LCD Ansteuerung erfolgt über zwei Ports: 40h = Control-Port, 41h = Data-Port. Die Low-Level Grafik-Cursor Positionierung hat eine Auflösung von 144x6 (144 Spalten und 6 Reihen). Die Reihen haben also Text-Auflösung, die Spalten Grafik-Auflösung.
Die erste Position (0,0) ist (links, oben). An eine gesetzte Cursor-Position kann dann via Port 41h ein vertikales Bitmuster (1 Byte) im GPRINT-Stil geschrieben werden (bit0 = unterstes Pixel, bit7 = oberstes Pixel).
Die Cursor-Position wird dabei (nach der Ausgabe) automatisch um eine Spalte nach rechts versetzt!!!

Dies ist die mit Abstand schnellste Möglichkeit auf das LCD zuzugreifen - es wird dabei (wie beim PC-1600) direkt der
Hardware-LCD-Treiber angesteuert, ohne Umweg über ein VRAM. Letzteres läßt sich damit jedoch einfach implementieren.

Darüber hinaus kann der Control-Port auch gelesen werden. Bit7 gibt dabei an, ob die LCD-Hardware noch BUSY ist.
In diesem Fall muss mit dem nächsten OUT-Befehl gewartet werden.

Folgende Werte können in den Port 40h geschrieben werden:

Wert (hex)	Funktion	Beschreibung
0n	Setzt das niederwertige Halbbyte (Low-Nibble) der X-Achse)	Wertebereich: 0<=n<=F
1n	Setzt das höherwertige Halbbyte (High-Nibble) der X-Achse	Wertebereich: 0<=n<=8
2n	n=4 LCD aus n=5 LCD an	
3n	n=Cursorblinkrate	30-3F,von schnell zu langsam, wobei 3F immer noch deutlich schneller ist als der Standard.
40-7F	VRAM Display Startposition	Das LCD hat 144x48 Punkte aber es befinden sich 144x64 Punkten im Speicher. 16 vertikalen Punkte sind immer versteckt. Beispiel für Scrolling: FOR A=0 TO 63 :OUT &H40,&H40+A : NEXT
80-9F	LCD-Kontrast	80 ist am hellsten, 9F ist am dunkelsten. Nutzbare Werte von 80-8F, danach ist kein Unterschied mehr zu erkennen
An	n=0 Mirror-Modus aus n=1 Mirror-Modus an n=4 alle Pixel(Maske) aus n=5 alle Pixel(Maske) an n=6 Inverse aus	

	n=7 Inverse an n=8 Spannung an n=9 Spannung verringern n=E alle aktiven Pixel aus n=F alle aktiven Pixel an	
Bn	Setzt die y-Achse	Wertebereich: 0<=n<=5
Cn	Partielles einschalten des Displays n=0 normale Anzeige n=1 linke 16 Punkte n=2 rechte 10 Punkte, einschließlich Modus n=3 linke 32 Punkte n=4 entspricht 1 + 2 n=5 rechts 42 Punkte + Modus	Beim Einschalten des Displays erscheint je nach dem links und rechts eine zur Mitte hin aufsteigende Linie. Wird Bit 8 mitgesetzt ist es eine zur Mitte abfallende Linie
Dn	Keine Funktion	
En	ungeklärt	
Fn	Keine Funktion	

Beispiel in Assembler für das Setzen der Cursor-Position und das Schreiben des 8-Bit-Pattern:

```
DI
LD A,0<colLow>
OUT (40H),A
LD A,1<colHigh>
OUT (40H),A
LD A,B<row>
OUT (40H),A
LD A,<8bit-pattern>
OUT (41H),A
EI
```

Die gleiche Programmfunktion wie oben in Basic:

```
10 GCURSOR(<colHigh>*16+<colLow>,7+<row>*8)
20 GPRINT "<8bit-pattern>"
```

Beispiel für das Invertieren des Displays in Basic:

```
OUT &H40,&HA7
```

Key Matrix:

Ausgang 11h
Eingang 10h

Ausgangs-Bit → Eingangs-Bit ↓	7	6	5	4	3	2	1	0
7	)	R · CM	M +	Enter	↑	,	K	U
6	/	*	-	+	↓	M	J	Y
5	9	6	3	=	SPACE	N	H	T
4	8	5	2	.	TAB	B	G	R
3	7	4	1	0	Cana	V	F	E
2	π	INS	CON	ON	CAPS	C	D	W
1	BS	O	;	→	TXT	X	S	Q
0	P	I	L	←	BASIC	Z	A	OF

Ausgang 12h
Eingang n 10h

Ausgangs-Bit → Eingangs-Bit ↓	7	6
S7	CLS	MDF
6	F ← → E	1 / x
5	tan	(
4	log	yx ^
3	in	x2
2	cos	√
1	sin	→ DEG
0	2ndF	nPr

Beispiel:

```
10 CLS
20 LINE(7,16)-(18,7),B
30 A=1
40 GCURSOR(8,15)
50 FOR B=1 TO 7
60 OUT &H11,A
70 A=A*2
80 GPRINT INP &H10;
90 NEXT
100 OUT &H11,0
110 FOR B=1 TO 2
120 OUT &H12,B
130 GPRINT INP &H10;
140 NEXT
150 OUT &H12,0
160 GOTO 30
```

BIOS Tastenwerte

obere 4 Bits → untere 4 Bits ↓	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		OFF	Q	W	E	R	T	Y	U	A	S	D	F	G	H	J
1	K	Z	X	C	V	B	N	M	,	BASIC	TEXT	CAPS	カナ	TAB	SPACE	↓
2	↑	←	→	ANS	0	.	=	+	RETURN	L	;	CONST	1	2	3	-
3	M+	I	O	INS	4	5	6	*	R-CM	P	BS	π	7	8	9	/
4	)	nPr	→ DEG	√	x2	yx^	(	1/x	MDF	2ndF	sin	cos	ln	log	tan	F⇔E
5	CLS	ON														

+80h beim Drücken der Shift-Taste
52h Wenn zwei oder mehr Tasten gedrückt wurden

Umrechnung der Werte des BEEP-Befehls zu Tönen:

Format:
BEEP repeat [, level][,length]

repeat
Anzahl Beep-Töne. 0 bis 65535

level
Frequenz des Summers. 230Hz ~ 8kHz (0 ~ 255). Optional.

length
Dauer des Tones. 0 bis 65535, optional.

length kann durch die folgende Gleichung mit der Frequenz berechnet werden.
1300000 / (166 +22 *level)Hz

Tabelle

C	C+	D	D+	E	F	F+	G	G+	A	A+	B
7											
21		18		15	14		12		10		8
49	46	43	40	37	35		30		26		22
105	99	93	87	82	77	72	68	64	60	56	52
219	200	194	182	172	162	152	143	135	127	119	112
										246	232

Beispiel:

```
10 DATA 105,93,82,77,68,60,52,49
20 FOR A=1 TO 8
30 READ B
40 C=650000/(166+22*B)
50 BEEP 1,B,C
60 NEXT
```

Inspection-Modus

```
OUT &H69,6
```

Wenn Sie diesen Befehl ausführen, wird das folgende Menü angezeigt:

```
* PC-G850V V1.03 CHECK *
1:TOTAL          2:RAM
3:ROM            4:11PIN
5:LCD      6:KEY
7:SHOCK          8:AGING
9:L.B,ESD        0:CURRENT
```

Mit diesem Menu kann der Pocket-Computer getestet werden.

ACHTUNG: Viele Funktionen löschen den gesamten Speicher.

Basic-Codes:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																MON
1	RUN	NEW	CONT	PASS	LIST	LLIST	CLOAD	RENUM	LOAD			DELETE	FILES			LCOPY
2	CSAVE	OPEN	CLOSE	SAVE		RANDOMIZE	DEGREE	RADIAN	GRAD	BEEP	WAIT	GOTO	TRON	TROFF	CLEAR	USING
3	DIM	CALL	POKE	GPRINT	PSET	PRESET					ERASE	LFILES	KILL			
4						OUT			PIOSET	POIPUT	SPOUT	SPINP	HDCOPY	ENDIF	REPEAT	UNTIL
5	CLS	LOCATE	TO	STEP	THEN	ON	IF	FOR	LET	REM	END	NEXT	STOP	READ	DATA	
6	PRINT	INPUT	GOSUB	LNINPUT	LPRINT	RETURN	RESTORE		GCURSOR	LINE						CIRCLE
7	PAINT	OUTPUT	APPEND	AS			ELSE				WHILE	WEND	SWITCH	CASE	DEFAULT	ENDSWITCH
8	MDF	REC	POL				TEN	RCP	SQU	CUR	HSN	HCS	HTN	AHS	AHC	AHT
9	FACT	LN	LOG	EXP	SQR	SIN	COS	TAN	INT	ABS	SGN	DEG	DMS	ASN	ACS	ATN
A	RND	AND	OR	NOT	PEEK	XOR	INP		PIOGET					POINT	PI	FRE
B	EOF		LOF				NCR	NPR								CUB
C							MOD	FIX								
D	ASC	VAL	LEN	VDEG												
E										INKEY$	MID$	LEFT$	RIGHT$			
F	CHR$	STR$	HEX$	DMS$												

Anhang K: BASIC-FEHLERMELDUNGEN

Wenn ein Fehler auftritt, wird einer der unten aufgeführten Codes auf der Anzeige dargestellt. Bei Fehlern, die während der Ausführung von Programmen auftreten, wird ebenfalls die Zeilennummer angezeigt, in welcher der Fehler aufgetreten ist.

Fehler-code	Bedeutung
10	Verwendung eines nicht zulässigen Ausdrucks oder einer nicht zulässigen Anweisung.
12	Es wurde versucht, einen Befehl auszuführen, der bei direkten Eingabevorgängen oder bei der Programmausführung nicht zulässig ist. Die Betriebsart für PRO oder RUN wurde falsch gewählt.
13	Die CONT-Anweisung wurde unzulässigerweise ausgeführt,
14	Es wurde versucht, ein Kennwort einem Programm zuzuweisen, welches nicht existiert.
15	Falsche Adresseingabe bei BSAVE M
20	Das Berechnungsergebnis übersteigt die Berechnungskapazität 10^{100}
21	Es wurde versucht, durch Null zu dividieren.
22	Ein nicht zulässiger Vorgang wurde angestrebt. (z.B. LOG -3 oder ASN 2)
30	Es wurde versucht, den Namen einer Feldvariablen zu vergeben, der bereits vergeben war.
31	Der Name einer Feldvariablen wurde ohne die DIM-Anweisung bestimmt.
32	Das Feld wurde unzulässigerweise adressiert (der Feldindex überschreitet die Größe, die dem Feld mit der DIM-Anweisung zugewiesen worden ist).
33	Der spezifizierte Wert überschreitet den zulässigen Bereich.
40	Die spezifizierte Zeilennummer oder die Benennung existieren nicht.
41	Die Zeilennummer wurde unzulässigerweise spezifiziert. (1-65279)
43	Unzulässige RENUM- oder LCOPY-Anweisung. (Eine nicht bestehende Zeilennummer wurde spezifiziert oder die Reihenfolge der Zeilenausführung wurde falsch angegeben.)
44	In einer Anweisung, z.B. LLIST oder DELETE, ist die Nummer der Endzeile geringer als die Nummer der Anfangszeile.
50	Das Niveau der Verschachtelung der Anweisungen GOSUB, FOR, REPEAT, WHILE oder SWITCH überschreitet den zulässigen Bereich.
51	Es wurde versucht, eine RETURN-Anweisung ohne den Aufruf einer Subroutine auszuführen.
52	Die FOR-Anweisung fehlt für eine NEXT-Anweisung,
53	Die DATA-Anweisung fehlt für eine READ-Anweisung.
54	Die zulässige Anzahl für Datenpuffer (8) oder Funktionspuffer (16) wurde überschritten.
55	Die Länge der eingegebenen Zeichenfolge überschreitet 255 Byte. Die Zeile überschreitet 255 Byte.
60	Die Größe des Programms oder der Variablen überschreitet die Speicherkapazität.
61	Es wurde versucht, eine IF oder ELSE-Anweisung im Blockformat auszuführen, ohne eine ENDIF-Anweisung zu geben

62	UNTIL wurde ohne REPEAT verwendet.
63	WHILE wurde ohne WEND verwendet.
64	Zur WHILE-Schleifenanweisung wurde kein WEND gefunden.
66	Nach einer DEFAULT-Anweisung in einem SWITCH-Befehl wurde eine weitere CASE- bzw. DEFAULT-Anweisung verwendet.
68	SWITCH, CASE bzw. DEFAULT wurde ohne ENDSWITCH verwendet.
69	CASE, DEFAULT bzw. ENDSWITCH wurde ohne SWITCH ausgeführt.
70	Die Zeichen können nicht in dem Format ausgegeben werden, das mit der USING-Anweisung bestimmt wurde.
71	Das Format, das mit der USING-Anweisung bestimmt wurde, ist nicht zulässig.
72	Fehler des I/O-Gerätes.
77	Die Größe der zu schreibenden Datei wurde überschritten
80	Prüfsummenfehler beim Lesen vom seriellen Interface.
81	Beim seriellen Interface wurde die Wartezeit während Programm- oder Daten-Ein-/Ausgabe überschritten).
82	Bei der Überprüfung des Inhalts durch die BLOAD ?-Anweisungen wurde ein Missverhältnis in den Inhalten festgestellt.
83	Die Art der spezifizierten Variablen im INPUT#-Befehl stimmt nicht mit der Art der gelesenen Daten überein.
84	Fehlfunktion des Druckers.
85	Ein Gerät oder eine Datei wurde nicht vor einem Ausgabebefehl (z.B. PRINT, INPUT) mit dem OPEN-Befehl geöffnet.
86	Ein Gerät oder eine Dateinummer wurde zu öffnen versucht, obwohl dieses Gerät bzw. die Dateinummer bereits geöffnet ist.
87	Es wurde versucht Daten zu lesen obwohl das Ende der Datei erreicht wurde.
90	Ein Versuch der Zuweisung von Zeichen zu einer numerischen Variablen oder von Zahlen zu einer Zeichenfolge-Variable wurde vorgenommen oder eine Zeichenfolge-Variable wurde in einer Funktion spezifiziert, die nur Zahlen als Variable haben kann, z.B. SIN A$.
91	Eine feste Variable, der Zahlen zugewiesen wurden, wurde als eine Zeichenfolge-Variable verwendet oder die Variable, der Buchstaben zugewiesen wurden, wurde als numerische Variable verwendet.
92	Das Kennwort stimmt nicht überein.
93	Ein Versuch zur manuellen Ausführung des MON, PEEK, POKE oder RENUM-Befehls wurde bei Vorhandensein eines Kennwortes vorgenommen.
94	Die spezifizierte Datei wurde nicht gefunden.
95	Unzulässiger Dateienname.
96	Ein Versuch zum Lesen einer TEXT-Datei in der BASIC-Betriebsart oder einer BASIC-Datei in der TEXT-Betriebsart wurde vorgenommen.
97	Die Anzahl der Dateien überschreitet 255.

Anhang L: Kurzanleitung zur Programmierung im Z80-Maschinencode

Dieses Kapitel ersetzt kein Handbuch für die Z80-Programmierung. Es soll nur als einfaches Dokument zum nachschlagen dienen.

Z80-Register und Flags:

Der Z80 verfügt über verschieden 8-Bit, sowie 16-Bit-Register. Wobei sich dabei einige 8-Bit-Register zu 16-Bit-Register zusammenfügen (gut zu erkennen an der Namensgebung der Register).

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

Der Z80-Prozessor stellt die 8-Bit-Register doppelt zur Verfügung so dass man zwei Registerpaare zur Verfügung hat. Mit einem Befehl kann man die aktuelle Register-Belegung mit einer anderen austauschen.

Der Z80 verfügt auch über eine Reihe von Flags. Flags sind 1-bit-Register in denen aktuelle Zustände angezeigt werden können.

S	Vorzeichenflag (Signed-Flag, 1 wenn negativ)
Z	Nullflag (Zero-Flag, 1 wenn Ergebnis 0)
AC	Hilfsübertragflag (auch Half-carry-Flag genannt)
P	Paritätsflag (1 bei Überlauf)
N	Subtraktionsflag (1 wenn Akkumulator als letztes eine Subtraktion ausgeführt hat)
C	Übertragsflag (Carryflag oder auch CY, 1 wenn Overflow)

Die Flags werden im sognannten F-Register gehalten (von Bit7-0):
S,Z,(5.Bit des letzten 8Bit-Befehls der ein Flag verändert hat),H(AC), (3.Bit des letzten 8Bit-Befehls der ein Flag verändert hat),P,N,C

Befehle des Z80

Abkürzungen:

r,r'	- Einfachregister A,B,C,D,E,H,L
dd	- Doppelregister BC, DE, HL, SP
qq	- Doppelregister AF, BC, DE, HL
pp	- Doppelregister BC, DE, SP
n	- 8-Bit-Konstante
nn	- 16-Bit-Konstante, Adresse
d	- Verschiebung bei Adressierung über Indexregister, im Bereich -128 ≤ d ≤ 127
b	- Bit, das in den Einzelbitbefehlen behandelt werden soll 0 ≤ b ≤ 7
m,M	- Inhalt der 8-Bit-Speicherstelle, die durch HL adressiert wird (L enthält Bits 0-7 ; H Bits 8-15)
p	- einer der Werte 00h, 08h, 10h, 18h, 20h, 28h, 30h, 38h
CY	- Carry-Flag
T	- Anzahl der Taktzyklen des Befehls

8-Bit-Ladebefehle

Die Ladebefehle transportieren 8-Bit-Daten zwischen Registern oder zwischen Registern und dem Speicher. Dabei steht im Operandenfeld als erstes der Zielspeicherplatz und als zweiter Operand der Quellenspeicherplatz. Der Inhalt des Quellenspeicherplatzes wird bei diesen Befehlen nicht verändert.

Mnemonik	T	**Wirkungsweise des Befehls**	**SZHPNC**
LD r,r'	4	der Inhalt des Registers r' wird in das Register r geladen	------
LD r,n	7	die 8-Bit-Konstante n wird im Register r gespeichert	------
LD r,m	7	der Inhalt des durch HL adressierten Speicherplatzes wird in das Register r geladen	------
LD r,(IX+d)	19	der Inhalt des durch IX plus Verschiebung d adressierten Speicherplatzes wird in das Register r geladen	------
LD r,(IY+d)	19	der Inhalt des durch IY plus Verschiebung d adressierten Speicherplatzes wird in das Register r geladen	------
LD m,r	7	der Inhalt des Registers r wird in den durch HL adressierten Speicherplatz geladen	------
LD (IX+d),r	19	der Inhalt des Registers r wird in den durch IX plus Verschiebung d adressierten Speicherplatz geladen	------
LD (IY+d),r	19	der Inhalt des Registers r wird in den durch IY plus Verschiebung d adressierten Speicherplatz geladen	------
LD m,n	10	die Konstante n wird in den durch HL adressierten Speicherplatz geladen	------
LD (IX+d),n	19	Die Konstante n wird in den durch IX plus Verschiebung d adressierten Speicherplatz geladen	------
LD (IY+d),n	19	Die Konstante n wird in den durch IY plus Verschiebung d adressierten Speicherplatz geladen	------
LD A,(BC)	7	der Inhalt des durch BC adressierten Speicherplatzes wird in den Akkumulator (A-Register) geladen	------
LD A,(DE)	7	der Inhalt des durch DE adressierten Speicherplatzes wird in den Akkumulator (A-Register) geladen	------
LD A,(nn)	13	der Inhalt des Speicherplatzes nn wird in den Akkumulator (A-Register) geladen	------
LD (BC),A	7	der Inhalt des Akkumulators (A-Register) wird in den durch BC adressierten Speicherplatz geladen	------
LD (DE),A	7	der Inhalt des Akkumulators (A-Register) wird in den durch DE adressierten Speicherplatz geladen	------
LD (nn),A	13	der Inhalt des Akkumulators (A-Register) wird an die Stelle des Speicherplatzes nn geladen	------
LD A,I	9	der Inhalt des Interruptregisters I wird in den Akkumulator (A-Register) geladen	**0F0-
LD A,R	9	der Inhalt des Refreshregisters R wird in den Akkumulator (A-Register) geladen	**0F0-
LD I,A	9	der Inhalt des Akkumulators (A-Register) wird in das Interruptregister I geladen	------
LD R,A	9	der Inhalt des Akkumulators (A-Register) wird in das Refreshregister R geladen	------

16-Bit-Ladebefehle

Diese Befehle transportieren 16-Bit-Daten zwischen Registern oder zwischen Registern und dem Speicher. Dabei steht im Operandenfeld als erstes der Zielspeicherplatz und als zweiter Operand der Quellenspeicherplatz. Der Inhalt des Quellenspeicherplatzes wird bei diesen Befehlen nicht verändert.

Spezielle 16-Bit-befehle sind die PUSH- und POP-Befehle. Mit ihnen werden 16-Bit-Daten aus Doppelregistern in den Kellerspeicher (Stack) gebracht bzw. zurück in die Doppelregister geholt.

Alle 16-Bit-Daten werden grundsätzlich in der Intel-Order (niederwertiges Byte zuerst) gespeichert.

Mnemonik	T	**Wirkungsweise des Befehls**	**SZHPNC**
LD dd,nn	10	die Konstante nn wird in das Doppelregister geladen	------
LD IX,nn	14	die Konstante nn wird in das Indexregister IX geladen	------
LD IY,nn	14	die Konstante nn wird in das Indexregister IY geladen	------
LD HL,(nn)	16	der Inhalt der Speicherplätze nn und nn+1 wird in das Doppelregister HL geladen (nn->L, nn+1->H)	------
LD pp,(nn)	20	der Inhalt der Speicherplätze nn und nn+1 wird in das Doppelregister pp geladen (nn->L, nn+1->H)	------
LD IX,(nn)	20	der Inhalt der Speicherplätze nn und nn+1 wird in das Doppelregister IX geladen (nn->X, nn+1->I)	------
LD IY,(nn)	20	der Inhalt der Speicherplätze nn und nn+1 wird in das Doppelregister IY geladen (nn->Y, nn+1->I)	------
LD (nn),HL	16	der Inhalt des Doppelregisters HL wird an die Adressen nn und nn+1 geladen (L->nn, H->nn+1)	------
LD (nn),pp	20	der Inhalt des Doppelregisters pp wird an die Adressen nn und nn+1 geladen (L->nn, H->nn+1)	------
LD (nn),IX	20	der Inhalt des Doppelregisters IX wird an die Adressen nn und nn+1 geladen (X->nn, I->nn+1)	------
LD (nn),IY	20	der Inhalt des Doppelregisters IY wird an die Adressen nn und nn+1 geladen (Y->nn, I->nn+1)	------
LD SP,HL	6	der Inhalt des Doppelregisters HL wird im Stackpointer SP gespeichert	------
LD SP,IX	10	der Inhalt des Doppelregisters IX wird im Stackpointer SP gespeichert	------
LD SP,IY	10	der Inhalt des Doppelregisters IY wird im Stackpointer SP gespeichert	------
PUSH qq	11	der Inhalt des Doppelregisters qq wird im Stack gespeichert DEC SP; LD (SP),H; DEC SP; LD (SP),L	------
PUSH IX	15	der Inhalt des Doppelregisters IX wird im Stack gespeichert DEC SP; LD (SP),I; DEC SP; LD (SP),X	------
PUSH IY	15	der Inhalt des Doppelregisters IY wird im Stack gespeichert DEC SP; LD (SP),I; DEC SP; LD (SP),Y	------
POP qq	10	der letzte Wert im Stack wird in das Doppelregister qq geladen LD L,(SP); INC SP; LD H,(SP); INC SP	------
POP IX	14	der letzte Wert im Stack wird in das Doppelregister IX geladen LD X,(SP); INC SP; LD I,(SP); INC SP	------
POP IY	14	der letzte Wert im Stack wird in das Doppelregister IY geladen LD Y,(SP); INC SP; LD I,(SP); INC SP	------

8-Bit-Arithmetik und Logikbefehle

Diese Befehle arbeiten mit Daten, die sich im Akkumulator A und Daten die sich in anderen Registern oder auf Speicherplätzen befinden. Das Ergebnis dieser Befehle wird im Akkumulator (A-Register) abgelegt.

Mnemonik	T	**Wirkungsweise des Befehls**	**SZHPNC**
ADD r	4	der Inhalt des Registers r wird zum Akkumulatorinhalt addiert	***V0*
ADD m	7	der Inhalt des durch das Register HL adressierten Speicherplatzes m wird zum Akkumulator Inhalt addiert	***V0*
ADD n	7	die Konstante n wird zum Akkumulatorinhalt addiert	***V0*
ADD (IX+d)	19	der Inhalt des durch das Register IX plus Verschiebung adressierten Speicherplatzes wird zum Akkumulatorinhalt addiert	***V0*
ADD (IY+d)	19	der Inhalt des durch das Register IX plus Verschiebung adressierten Speicherplatzes wird zum Akkumulatorinhalt addiert	***V0*
ADC r	4	der Inhalt des Registers r plus Carry-Flag wird zum Akkumulatorinhalt addiert	***V0*
ADC m	7	der Inhalt des durch das Register HL adressierten Speicherplatzes m plus Carry-Flag wird zum Akkumulator Inhalt addiert	***V0*
ADC n	7	die Konstante n plus Carry-Flag wird zum Akkumulatorinhalt addiert	***V0*
ADC (IX+d)	19	der Inhalt des durch das Register IX plus Verschiebung plus Carry-Flag adressierten Speicherplatzes wird zum Akkumulatorinhalt addiert	***V0*
ADC (IY+d)	19	der Inhalt des durch das Register IY plus Verschiebung plus Carry-Flag adressierten Speicherplatzes wird zum Akkumulatorinhalt addiert	***V0*
SUB r	4	der Inhalt des Registers r wird vom Akkumulatorinhalt subtrahiert	***V1*
SUB m	7	der Inhalt des durch das Register HL adressierten Speicherplatzes m wird vom Akkumulatorinhalt subtrahiert	***V1*
SUB n	7	die Konstante n wird vom Akkumulatorinhalt subtrahiert	***V1*
SUB (IX+d)	19	der Inhalt des durch das Register IX plus Verschiebung adressierten Speicherplatzes wird vom Akkumulatorinhalt subtrahiert	***V1*
SUB (IY+d)	19	der Inhalt des durch das Register IY plus Verschiebung adressierten Speicherplatzes wird vom Akkumulatorinhalt subtrahiert	***V1*
SBC r	4	der Inhalt des Registers r plus Carry-Flag wird vom Akkumulatorinhalt subtrahiert	***V1*
SBC m	7	der Inhalt des durch das Register HL adressierten Speicherplatzes m plus Carry-Flag wird vom Akkumulatorinhalt subtrahiert	***V1*
SBC n	7	die Konstante n plus Carry-Flag wird vom Akkumulatorinhalt subtrahiert	***V1*
SBC (IX+d)	19	der Inhalt des durch das Register IX plus Verschiebung plus Carry-Flag adressierten Speicherplatzes wird vom Akkumulatorinhalt subtrahiert	***V1*
SBC (IY+d)	19	der Inhalt des durch das Register IY plus Verschiebung plus Carry-Flag adressierten Speicherplatzes wird vom Akkumulatorinhalt subtrahiert	***V1*
AND r	4	logische UND-Verknüpfung mit dem Inhalt eines Registers und dem Akkumulatorinhalt	**1P00
AND m	7	logische UND-Verknüpfung mit dem durch das Register HL adressierten Speicherplatzes m und dem Akkumulatorinhalt	**1P00
AND n	7	logische UND-Verknüpfung mit der Konstanten und dem Akkumulatorinhalt	**1P00
AND (IX+d)	19	logische UND-Verknüpfung mit dem Inhalt des durch das Register IX plus Verschiebung adressierten Speicherplatzes und dem Akkumulatorinhalt	**1P00
AND (IY+d)	19	logische UND-Verknüpfung mit dem Inhalt des durch das Register IY plus Verschiebung adressierten Speicherplatzes und dem	**1P00

		Akkumulatorinhalt	
OR r	4	logische ODER-Verknüpfung mit dem Inhalt eines Registers und dem Akkumulatorinhalt	**0P00
OR m	7	logische ODER-Verknüpfung mit dem durch das Register HL adressierten Speicherplatzes m und dem Akkumulatorinhalt	**0P00
OR n	7	logische ODER-Verknüpfung mit der Konstanten und dem Akkumulatorinhalt	**0P00
OR (IX+d)	19	logische ODER-Verknüpfung mit dem Inhalt des durch das Register IX plus Verschiebung adressierten Speicherplatzes und dem Akkumulatorinhalt	**0P00
OR (IY+d)	19	logische ODER-Verknüpfung mit dem Inhalt des durch das Register IY plus Verschiebung adressierten Speicherplatzes und dem Akkumulatorinhalt	**0P00
XOR r	4	logische Exklusive-ODER-Verknüpfung mit dem Inhalt eines Registers und dem Akkumulatorinhalt	**0P00
XOR m	7	logische Exklusive-ODER-Verknüpfung mit dem durch das Register HL adressierten Speicherplatzes m und dem Akkumulatorinhalt	**0P00
XOR n	7	logische Exklusive-ODER-Verknüpfung mit der Konstanten und dem Akkumulatorinhalt	**0P00
XOR (IX+d)	19	logische Exklusive-ODER-Verknüpfung mit dem Inhalt des durch das Register IX plus Verschiebung adressierten Speicherplatzes und dem Akkumulatorinhalt	**0P00
XOR (IY+d)	19	logische Exklusive-ODER-Verknüpfung mit dem Inhalt des durch das Register IY plus Verschiebung adressierten Speicherplatzes und dem Akkumulatorinhalt	**0P00
CP r	4	Vergleich des Inhalts eines Registers mit dem Akkumulatorinhalt Zero-Flag: 1 -> beide Inhalte sind gleich; 0 -> Inhalte sind unterschiedlich Carry-Flag: 1 -> Akkumulatorinhalt ist kleiner zweitem Operanden; 0 -> Akkumulator ist größer oder gleich zweiten Operanden	***V1*
CP m	7	Vergleich des Inhalts dem durch das Register HL adressierten Speicherplatzes m mit dem Akkumulatorinhalt	***V1*
CP n	7	Vergleich der Konstante mit dem Akkumulatorinhalt	***V1*
CP (IX+d)	19	Vergleich des Inhalts des durch das Register IX plus Verschiebung adressierten Speicherplatzes mit dem Akkumulatorinhalt	***V1*
CP (IY+d)	19	Vergleich des Inhalts des durch das Register IY plus Verschiebung adressierten Speicherplatzes mit dem Akkumulatorinhalt	***V1*
INC r	4	der Inhalt des Registers r wird um eins erhöht	***V0-
INC m	11	er Inhalt des durch HL adressierten Speicherplatzes m wird um eins erhöht	***V0-
INC (IX+d)	23	der Inhalt des durch das Register IX plus Verschiebung adressierten Speicherplatzes wird um eins erhöht	***V0-
INC (IY+d)	23	der Inhalt des durch das Register IY plus Verschiebung adressierten Speicherplatzes wird um eins erhöht	***V0-
DEC r	4	der Inhalt des Registers r wird um eins vermindert	***V1-
DEC m	11	er Inhalt des durch HL adressierten Speicherplatzes m wird um eins vermindert	***V1-
DEC (IX+d)	23	der Inhalt des durch das Register IX plus Verschiebung adressierten Speicherplatzes wird um eins vermindert	***V1-
DEC (IY+d)	23	der Inhalt des durch das Register IY plus Verschiebung adressierten Speicherplatzes wird um eins vermindert	***V1-
DAA	4	korrigiert nach Addition / Subtraktion zweier gepackter BCD-Zahlen den Akkumulatorinhalt so, dass im Akkumulator wieder die gepackte BCD-Darstellung erreicht wird	***P-*
CPL	4	bitweises Negieren (Komplementieren) des Akkumulatorinhalts (1er-Komplement)	--1-1-

NEG	8	subtrahieren des Akkumulatorinhalts von Null (2er-Komplement, bitweises Negieren aller Bits, dann um 1 erhöhen)	***V1*
CCF	4	Komplementieren des Carry-Flags	--X-0*
SCF	4	Setzen des Carry-Flags auf 1	--0-01

16-Bit-Arithmetikbefehle

Diese Befehle arbeiten ähnlich den 8-Bit-Arithmetikbefehlen, jedoch mit Doppelregistern. Als "Akkumulator" wird eines der Doppelregister HL, IX oder IY benutzt.

Mnemonik	T	**Wirkungsweise des Befehls**	**SZHPNC**
ADD HL,dd	11	der Inhalt des Registerspaares dd wird zum Inhalt des Registerpaares HL addiert	--x-0*
ADD IX,IX	15	der Inhalt des Registerspaares IX wird mit sich selbst addiert (Verdopplung)	--x-0*
ADD IY,IY	15	der Inhalt des Registerspaares IY wird mit sich selbst addiert (Verdopplung)	--x-0*
ADD IX,pp	15	der Inhalt von pp wird zum Inhalt des Registerpaares IX addiert	--x-0*
ADD IY,pp	15	der Inhalt von pp wird zum Inhalt des Registerpaares IY addiert	--x-0*
ADC HL,dd	15	der Inhalt des Registerspaares dd plus Carry-Flag wird zum Inhalt des Registerpaares HL addiert	**xV0*
SBC HL,dd	15	der Inhalt des Registerspaares dd plus Carry-Flag wird vom Inhalt des Registerpaares HL subtrahiert	**xV1*
INC dd	6	der Inhalt des Doppelregisters dd wird um eins erhöht	------
INC IX	10	der Inhalt des Doppelregisters IX wird um eins erhöht	------
INC IY	10	der Inhalt des Doppelregisters IY wird um eins erhöht	------
DEC dd	6	der Inhalt des Doppelregisters dd wird um eins vermindert	------
DEC IX	10	der Inhalt des Doppelregisters IX wird um eins vermindert	------
DEC IY	10	der Inhalt des Doppelregisters IY wird um eins vermindert	------

Registeraustauschbefehle

Diese Befehle dienen dem schnellen Austausch von Doppelregisterinhalten und erschließen dem Programmierer die Hintergrundregister.

Mnemonik	T	**Wirkungsweise des Befehls**	**SZHPNC**
EX DE,HL	4	Austausch des Inhalts der Doppelregister DE und HL	------
EX AF,AF'	4	Austausch des Inhalts der Doppelregister AF und AF'	------
EXX	4	Austausch des Inhalts der Doppelregister BC ⇄ BC' DE ⇄ DE' HL ⇄ HL'	------
EX (SP),HL	19	Austausch des Inhalts des Doppelregisters mit dem letzten Wert im Stack (SP+1) ⇄ H (SP) ⇄ L	------
EX (SP),IX	23	Austausch des Inhalts des Doppelregisters mit dem letzten Wert im Stack (SP+1) ⇄ I (SP) ⇄ X	------
EX (SP),IY	23	Austausch des Inhalts des Doppelregisters mit dem letzten Wert im Stack (SP+1) ⇄ I (SP) ⇄ y	------

Programmverzweigungsbefehle

Es gibt unbedingte und bedingte Sprüngen. Weiterhin sind absolute und relative Sprünge möglich. Die relativen Sprünge sind nur in einer Umgebung von -126 bis +129 Byte möglich. Bei bedingten Sprüngen müssen die Flag-Bedingungen als Operanden angegeben werden und es werden die entsprechenden Flag-Bits getestet. In Abhängigkeit von diesem Test wird dann der Sprung ausgeführt oder ignoriert.

Mnemonik	T	**Wirkungsweise des Befehls**	**SZHPNC**
JP nn	10	unbedingter Sprung nach Adresse nn	------
JP NZ,nn	10	Sprung nach Adresse nn, wenn das Zero-Flag gleich 0 ist	------
JP Z,nn	10	Sprung nach Adresse nn, wenn das Zero-Flag gleich 1 ist	------
JP NC,nn	10	Sprung nach Adresse nn, wenn das Carry-Flag gleich 0 ist	------
JP C,nn	10	Sprung nach Adresse nn, wenn das Carry-Flag gleich 1 ist	------
JP PO,nn	10	Sprung nach Adresse nn, wenn das P-Flag gleich 0 ist	------
JP PE,nn	10	Sprung nach Adresse nn, wenn das P-Flag gleich 1 ist	------
JP P,nn	10	Sprung nach Adresse nn, wenn das Sign-Flag gleich 0 ist	------
JP M,nn	10	Sprung nach Adresse nn, wenn das Sign-Flag gleich 1 ist	------
JR nn	10	Relativer unbedingter Sprung nach Adresse nn	------
JR NZ,nn	12	Relativer Sprung nach Adresse nn, wenn das Zero-Flag gleich 0 ist	------
JR Z,nn	12/7	Relativer Sprung nach Adresse nn, wenn das Zero-Flag gleich 1 ist	------
JR NC,nn	12/7	Relativer Sprung nach Adresse nn, wenn das Carry-Flag gleich 0 ist	------
JR C,nn	12/7	Relativer Sprung nach Adresse nn, wenn das Carry-Flag gleich 1 ist	------
JP m	4	unbedingter Sprung zu der Adresse, die im Doppelregister HL steht	------
JP (IX)	8	unbedingter Sprung zu der Adresse, die im Doppelregister IX steht	------
JP (IY)	8	unbedingter Sprung zu der Adresse, die im Doppelregister IY steht	------
DJNZ nn	13/8	der Inhalt des Registers B wird um eine vermindert, relative bedingter Sprung zur Adresse nn, wenn B <> 0	------

Unterprogrammbefehle

Es gibt wie bei den Sprungbefehlen bedingte und unbedingte Befehle. Der Unterprogrammaufruf erfolgt mit der Speicherung der dem CALL-Befehl folgenden Adresse (Rückkehradresse) auf dem Stack. Wird das Unterprogramm mit dem RET-Befehl abgeschlossen, so wird das Programm ab der Rückkehradresse weiter abgearbeitet, in dem die Adresse vom Stack zurückgeladen wird.

Mnemonik	T	**Wirkungsweise des Befehls**	**SZHPNC**
CALL nn	17	unbedingter Unterprogrammaufruf zur Adresse nn	------
CALL NZ,nn	17/10	Unterprogrammaufruf zur Adresse nn, wenn das Zero-Flag gleich 0 ist	------
CALL Z,nn	17/10	Unterprogrammaufruf zur Adresse nn, wenn das Zero-Flag gleich 1 ist	------
CALL NC,nn	17/10	Unterprogrammaufruf zur Adresse nn, wenn das Carry-Flag gleich 0 ist	------
CALL C,nn	17/10	Unterprogrammaufruf zur Adresse nn, wenn das Carry-Flag gleich 1 ist	------
CALL PO,nn	17/10	Unterprogrammaufruf zur Adresse nn, wenn das P-Flag gleich 0 ist	------
CALL PE,nn	17/10	Unterprogrammaufruf zur Adresse nn, wenn das P-Flag gleich 1 ist	------
CAPP P,nn	17/10	Unterprogrammaufruf zur Adresse nn, wenn das Sign-Flag gleich 0	------

		ist	
CALL M,nn	17/10	Unterprogrammaufruf zur Adresse nn, wenn das Sign-Flag gleich 1 ist	------
RST p	11	der RST-Befehl ist ein spezieller Unterprogrammaufruf, es sind 8 folgende Restart-Adressen zugelassen: p = 00h,08h,10h,18h,20h,28h,30h,38h der höherwertige Adressteil ist stets Null, ansonsten entspricht der RST-Befehl einem unbedingten Unterprogrammaufruf	------
RET	10	unbedingter Rücksprung aus einem Unterprogramm die Ausführung erfolgt, in dem die Rückkehradresse wie bei einem POP-Befehl aus dem Stack geholt wird und von dieser Adresse weiterbearbeitet wird	------
RET NZ	11/5	bedingter Rücksprung, wenn das Zero-Flag gleich 0 ist	------
RET Z	11/5	bedingter Rücksprung, wenn das Zero-Flag gleich 1 ist	------
RET NC	11/5	bedingter Rücksprung, wenn das Carry-Flag gleich 0 ist	------
RET C	11/5	bedingter Rücksprung, wenn das Carry-Flag gleich 1 ist	------
RET PO	11/5	bedingter Rücksprung, wenn das P-Flag gleich 0 ist	------
RET PE	11/5	bedingter Rücksprung, wenn das P-Flag gleich 1 ist	------
RET P	11/5	bedingter Rücksprung, wenn das Sign-Flag gleich 0 ist	------
RET M	11/5	bedingter Rücksprung, wenn das Sign-Flag gleich 1 ist	------
RETI	14	Rücksprung aus Interruptbehandlungsroutine (ISR) eines maskierbaren Interrupts	------
RETN	14	Rücksprung aus Interruptbehandlungsroutine (ISR) eines nicht maskierbaren Interrupts	------

Rotations- und Verschiebebefehle

Durch diese Befehle im Akkumulator (A-Register), in einem anderen Register oder in einem Speicherplatz Daten zyklisch (bitweise) verschoben. Das aus dem Byte herausgeschobene Bit wird im Carry-Flag abgelegt.

Mnemonik	T	**Wirkungsweise des Befehls**	**SZHPNC**
RLCA	4	Linksrotation des Akkumulatorinhalts um eine Bitposition nach links, Bit 7 wird zum Inhalt von Bit 0	--0-0*
RRCA	4	Rechtsrotation des Akkumulatorinhalts um eine Bitposition nach rechts, Bit 0 wird zum Inhalt von Bit 7	--0-0*
RLA	4	Linksrotation des Akkumulatorinhalts um eine Bitposition nach links durch das Carry-Flag, Bit 7 wird zum Inhalt des Carry-Flags und dessen alter Inhalt zum Bit 0	--0-0*
RRA	4	Rechtsrotation des Akkumulatorinhalts um eine Bitposition nach rechts durch das Carry-Flag, Bit 0 wird zum Inhalt des Carry-Flags und dessen alter Inhalt zum Bit 7	--0-0*
RLC r	8	Linksrotation des Registers um eine Bitposition nach links, Bit 7 wird zum Inhalt von Bit 0	**0P0*
RLC m	15	Linksrotation des mit dem durch das Register HL adressierten Speicherplatzes um eine Bitposition nach links, Bit 7 wird zum Inhalt von Bit 0	**0P0*
RLC (IX+d)	23	Linksrotation des Inhalts des durch das Register IX plus Verschiebung adressierten Speicherplatzes um eine Bitposition nach links, Bit 7 wird zum Inhalt von Bit 0	**0P0*
RLC (IY+d)	23	Linksrotation des Inhalts des durch das Register IY plus Verschiebung adressierten Speicherplatzes um eine Bitposition nach links, Bit 7 wird zum Inhalt von Bit 0	**0P0*
RRC r	8	Rechtsrotation des Registers um eine Bitposition nach rechts, Bit 0	**0P0*

		wird zum Inhalt von Bit 7	
RRC m	15	Rechtsrotation des mit dem durch das Register HL adressierten Speicherplatzes um eine Bitposition nach rechts, Bit 0 wird zum Inhalt von Bit 7	**0P0*
RRC (IX+d)	23	Rechtsrotation des Inhalts des durch das Register IX plus Verschiebung adressierten Speicherplatzes um eine Bitposition nach rechts, Bit 0 wird zum Inhalt von Bit 7	**0P0*
RRC (IY+d)	23	Rechtsrotation des Inhalts des durch das Register IY plus Verschiebung adressierten Speicherplatzes um eine Bitposition nach rechts, Bit 0 wird zum Inhalt von Bit 7	**0P0*
RL r	8	Linksrotation des Registers um eine Bitposition nach links durch das Carry-Flag, Bit 7 wird zum Inhalt des Carry-Flags und dessen alter Inhalt zum Bit 0	**0P0*
RL m	15	Linksrotation des mit dem durch das Register HL adressierten Speicherplatzes um eine Bitposition nach links durch das Carry-Flag, Bit 7 wird zum Inhalt des Carry-Flags und dessen alter Inhalt zum Bit 0	**0P0*
RL (IX+d)	23	Linksrotation des Inhalts des durch das Register IX plus Verschiebung adressierten Speicherplatzes um eine Bitposition nach links durch das Carry-Flag, Bit 7 wird zum Inhalt des Carry-Flags und dessen alter Inhalt zum Bit 0	**0P0*
RL (IY+d)	23	Linksrotation des Inhalts des durch das Register IY plus Verschiebung adressierten Speicherplatzes um eine Bitposition nach links durch das Carry-Flag, Bit 7 wird zum Inhalt des Carry-Flags und dessen alter Inhalt zum Bit 0	**0P0*
RR r	8	Rechtsrotation des Registers um eine Bitposition nach rechts durch das Carry-Flag, Bit 0 wird zum Inhalt des Carry-Flags und dessen alter Inhalt zum Bit 7	**0P0*
RR m	15	Rechtsrotation des mit dem durch das Register HL adressierten Speicherplatzes um eine Bitposition nach rechts durch das Carry-Flag, Bit 0 wird zum Inhalt des Carry-Flags und dessen alter Inhalt zum Bit 7	**0P0*
RR (IX+d)	23	Rechtsrotation des Inhalts des durch das Register IX plus Verschiebung adressierten Speicherplatzes um eine Bitposition nach rechts durch das Carry-Flag, Bit 0 wird zum Inhalt des Carry-Flags und dessen alter Inhalt zum Bit 7	**0P0*
RR (IY+d)	23	Rechtsrotation des Inhalts des durch das Register IY plus Verschiebung adressierten Speicherplatzes um eine Bitposition nach rechts durch das Carry-Flag, Bit 0 wird zum Inhalt des Carry-Flags und dessen alter Inhalt zum Bit 7	**0P0*
SLA r	8	Linksverschiebung eines Registers um ein Bit durch das Carry-Flag. Das Bit 0 wird 0.	**0P0*
SLA m	15	Linksverschiebung des mit dem durch das Register HL adressierten Speicherplatzes um ein Bit durch das Carry-Flag. Das Bit 0 wird 0.	**0P0*
SLA (IX+d)	23	Linksverschiebung des Inhalts des durch das Register IX plus Verschiebung adressierten Speicherplatzes um ein Bit durch das Carry-Flag. Das Bit 0 wird 0.	**0P0*
SLA (IY+d)	23	Linksverschiebung des Inhalts des durch das Register IY plus Verschiebung adressierten Speicherplatzes um ein Bit durch das Carry-Flag. Das Bit 0 wird 0.	**0P0*
SRA r	8	Rechtsverschiebung eines Registers um ein Bit durch das Carry-Flag. Der Inhalt von Bit 7 bleibt erhalten	**0P0*
SRA m	15	Rechtsverschiebung des mit dem durch das Register HL adressierten Speicherplatzes um ein Bit durch das Carry-Flag. Der Inhalt von Bit 7 bleibt erhalten	**0P0*
SRA (IX+d)	23	Rechtsverschiebung des Inhalts des durch das Register IX plus	**0P0*

		Verschiebung adressierten Speicherplatzes um ein Bit durch das Carry-Flag. Der Inhalt von Bit 7 bleibt erhalten.	
SRA (IY+d)	23	Rechtsverschiebung des Inhalts des durch das Register IY plus Verschiebung adressierten Speicherplatzes um ein Bit durch das Carry-Flag. Der Inhalt von Bit 7 bleibt erhalten	**0P0*
RLD	18	zyklische Verschiebung nach links zwischen dem Akkumulator und dem Inhalt des durch HL adressiertem Speicherplatzes (m). Die unteren 4 Bit von m werden in die oberen 4 Bitstellen übertragen und diese ihrerseits in die unteren 4 Bits des Akkumulators. Die unteren 4 Bits des Akkumulators werden in die unteren 4 Bitstellen von m transportiert. Die oberen 4 Bits des Akkumulators bleiben unverändert.	**0P0*
RRD	18	zyklische Verschiebung nach rechts zwischen dem Akkumulator und dem Inhalt des durch HL adressiertem Speicherplatzes (m). Die unteren 4 Bit von m werden in die unteren 4 Bitstellen des Akkumulators übertragen und diese ihrerseits in die oberen 4 Bits von m. Die oberen 4 Bits von m werden in die unteren transportiert. Die oberen 4 Bits des Akkumulators bleiben unverändert.	**0P0*

Einzelbitbefehle

Diese Befehle erlauben es einzelne Bits in Registern oder auf Speicherplätzen zu testen, zu setzen oder zu löschen.

Mnemonik	T	**Wirkungsweise des Befehls**	**SZHPNC**
BIT b,r	8	die durch b gekennzeichnete Bitposition wird in dem Register r getestet. Das Komplement steht im Zero-Flag.	X*1x0–
BIT b,m	12	die durch b gekennzeichnete Bitposition wird in der Speicherstelle m getestet. Das Komplement steht im Zero- Flag.	X*1x0–
BIT b,(IX+d)	20	die durch b gekennzeichnete Bitposition wird in der Speicherstelle getestet. Das Komplement steht im Zero- Flag.	X*1x0–
BIT b,(IY+d)	20	die durch b gekennzeichnete Bitposition wird in der Speicherstelle getestet. Das Komplement steht im Zero- Flag.	X*1x0–
SET b,r	8	die durch b gekennzeichnete Bitposition wird in dem Register r auf 1 gesetzt	------
SET b,m	12	die durch b gekennzeichnete Bitposition wird in der Speicherstelle m auf 1 gesetzt.	------
SET b,(IX+d)	20	die durch b gekennzeichnete Bitposition wird in der Speicherstelle auf 1 gesetzt.	------
SET b,(IY+d)	20	die durch b gekennzeichnete Bitposition wird in der Speicherstelle auf 1 gesetzt.	------
RES b,r	8	die durch b gekennzeichnete Bitposition wird in dem Register r auf 0 zurückgesetzt.	------
RES b,m	12	die durch b gekennzeichnete Bitposition wird in der Speicherstelle m auf 0 zurückgesetzt.	------
RES b,(IX+d)	20	die durch b gekennzeichnete Bitposition wird in der Speicherstelle auf 0 zurückgesetzt.	------
RES b,(IY+d)	20	die durch b gekennzeichnete Bitposition wird in der Speicherstelle auf 0 zurückgesetzt	------

CPU-Steuerbefehle

Diese Befehle dienen der Steuerung des Interruptsystems der CPU.

Mnemonik	T	Wirkungsweise des Befehls	SZHPNC
NOP	4	die CPU führt keine Operation aus, es werden aber Refreshzyklen erzeugt	------
HALT	4	die CPU führt so lang eine Folge von NOP-Befehlen aus, bis ein Interrupt oder RESET an der CPU aktiv wird.	------
DI	4	der maskierbare Interrupt wird durch Rücksetzen der Interruptfreigabe-Flipflops IFF1 und IFF2 gesperrt. Nichtmaskierbare Interrupts werden anerkannt.	------
EI	4	der maskierbare Interrupt wird durch Setzen der Interruptfreigabe-Flipflops IFF1 und IFF2 freigegeben. Während der Ausführung dieses Befehls akzeptiert die CPU keine Interruptanforderungen.	------
IM 0	8	CPU in Interruptmodus 0 bringen	------
IM 1	8	CPU in Interruptmodus 1 bringen	------
IM 2	8	CPU in Interruptmodus 2 bringen	------

Blocktransfer- und -suchbefehle

Mit einem einzigen Befehl können beliebig große Datenmengen im Speicher kopiert werden bzw. es kann in einem Speicherbereich nach einem Datenbyte gesucht werden. Die Suche wird beendet, wenn das Byte gefunden oder das Ende des Speicherbereichs erreicht wurde.

Mnemonik	T	Wirkungsweise des Befehls	SZHPNC
LDI	16	Kopiert ein Datenbyte von der Speicherstelle, die durch HL adressiert wird, an die Speicherstelle, die durch DE adressiert wird. Die Register DE und HL werden um eins erhöht, das Register BC um eins vermindert. BC = 0 -> P = 0 BC <> 0 -> P = 1	--0*0-
LDIR	21	kopiert mehrere Datenbytes durch Ausführung des Befehls LDI Wiederholung des Befehls, bis BC = 0 ist	--000-
LDD	16	Kopiert ein Datenbyte von der Speicherstelle, die durch HL adressiert wird, an die Speicherstelle, die durch DE adressiert wird. Die Register DE, HL und BC werden um eins vermindert. BC = 0 -> P = 0 BC <> 0 -> P = 1	--0*0-
LDDR	21	kopiert mehrere Datenbytes durch Ausführung des Befehls LDD Wiederholung des Befehls, bis BC = 0 ist	--000-
CPI	16	Vergleich des Inhalts des durch HL adressierten Speicherplatzes mit dem Inhalt des Akkumulators (A-Register) A = (HL) -> Z = 1 A <> (HL) ->Z = 0 anschließend wird das Register HL um eins erhöht und das Register BC um eins vermindert BC = 0 -> P = 0 BC <> 0 -> P = 1	****1-
CPIR	21	vergleicht mehrere Datenbytes durch Ausführung des Befehls CPI Wiederholung des Befehls, bis BC = 0 oder A = (HL) ist	****1-
CPD	16	Vergleich des Inhalts des durch HL adressierten Speicherplatzes	****1-

		mit dem Inhalt des Akkumulators (A-Register) A = (HL) -> Z = 1 A <> (HL) -> Z = 0 anschließend wird das Register HL und BC um eins vermindert BC = 0 -> P = 0 BC <> 0 -> P = 1	
CPDR	21	vergleicht mehrere Datenbytes durch Ausführung des Befehls CPD Wiederholung des Befehls, bis BC = 0 oder A = (HL) ist	****1–

Ein- und Ausgabebefehle

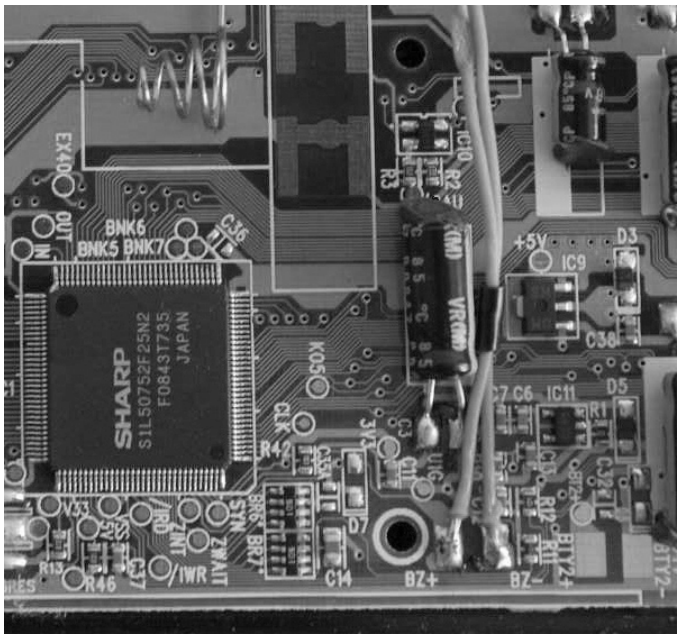
Mit diesen Befehlen können Datenbytes zwischen Registern oder Speicheradressen und externen Bausteinen ausgetauscht werden. Der externe Baustein wird dabei über eine sogenannte Portadresse (8-Bit-Wert) angesprochen. Diese Portadresse wird je nach Befehl entweder direkt angegeben (als Konstante) oder muss im Register C zur Verfügung stehen. Ähnlich den Blocktransferbefehlen existieren auch hier Befehle für die Datenein- und -ausgabe ganzer Speicherbereiche.
Wird für die Adressierung das Register C benutzt, liegt der Inhalt des Registers B an den höherwertigen 8 Bits des Adressbusses an.

Mnemonik	T	**Wirkungsweise des Befehls**	**SZHPNC**
IN A,(n)	11	die Eingabekanaladresse wird mit der Konstante n eingestellt; Zielregister ist der Akkumulator (n) -> A	------
IN r,(C)	12	die Eingabekanaladresse wird indirekt mit dem Register C eingestellt; Zielregister ist r (C) -> r	**0P0–
INI	16	die Eingabekanaladresse wird mit der Konstante n eingestellt; Zielregister ist der durch HL adressierte Speicherplatz B wird um eins vermindert und HL um eins erhöht (C) -> (HL) B-1 -> B B=0? -> Z=1 sonst Z=0 HL+1 -> HL	x*xx1–
INIR	21	wiederholte Ausführung des Befehls INI, bis das Register B gleich Null ist	x1xx1–
IND	16	die Eingabekanaladresse wird mit der Konstante n eingestellt; Zielregister ist der durch HL adressierte Speicherplatz B und HL wird um eins vermindert (C) -> (HL) B-1 -> B B=0? -> Z=1 sonst Z=0 HL-1 -> HL	x*xx1–
INDR	21	wiederholte Ausführung des Befehls IND, bis das Register B gleich Null ist	x1xx1–
OUT (n),A	11	die Ausgabekanaladresse wird mit der Konstante n eingestellt; Quellregister ist der Akkumulator A -> (n)	------
OUT (C),r	12	die Ausgabekanaladresse wird indirekt mit dem Register C eingestellt; Quellregister ist r r -> (C)	------
OUTI	16	ie Ausgabekanaladresse wird mit der Konstante n eingestellt; Quellregister ist der durch HL adressierte Speicherplatz; B wird um eins vermindert und HL um eins erhöht (HL) -> (C) B-1 -> B B=0? -> Z=1 sonst Z=0 HL+1 -> HL	x*xx1–
OTIR	21	wiederholte Ausführung des Befehls OUTI, bis das Register B gleich Null ist	x1xx1–
OUTD	16	die Ausgabekanaladresse wird mit der Konstante n eingestellt;	x*xx1–

		Quellregister ist der durch HL adressierte Speicherplatz; B und HL wird um eins vermindert (HL) -> (C) B-1 -> B B=0? -> Z=1 sonst Z=0 HL-1 -> HL	
OTDR	21	wiederholte Ausführung des Befehls OUTD, bis das Register B gleich Null ist	x1xx1–

Anhang M: Nachrüsten eines Lautsprechers/Piezos

Der PC-G850V(S) verfügt über einen Anschluss zum Nachrüsten eines Lautsprechers. Diese Anschlüsse sind mit BZ+ und BZ- gekennzeichnet. Hier wird der Piezo angelötet und z.B. mit beidseitigem Klebeband am Gehäuse oder der Schutzfolie befestigt.



Hinweis: Die Vorgängermodelle besitzen nicht diese beiden Anschlüsse, hier müssen die Kabel am 11Pin-Interface direkt angelötet werden. (Pin 3 (FL3) und Pin 7 (FL7).