HAND BOOK

EUROPEAN HOME MICROCOMPUTER

NASCOM 1

SOFTWARE NOTES

NASCO SALES LTD

LYNX Electronics (LONDON) LTD.

92 BROAD STREET
CHESHAM, BUCKS.

N.A.S. GmbH
Briennerstr. 56
D-8000 München 2
Tel.: 089/5233153/4
Telex: 0522061 nasco d

Teil 16 = Erläuterung der NASBUG-Befehle

DAS BETRIEBSSYSTEM DES N A S C O M 1

----- NASBUG -----

1.
Die Leistungsfähigkeit eines Computers wird maßgeblich durch sein Betriebssystem bestimmt. Das Betriebssystem des NASCOM 1 hat den Namen NASBUG. Das Betriebssystem für diesen auf dem Mikroprozessor Z 80 aufgebauten Computer soll den Benutzer beim Erstellen seines Programmes und bei der Fehlersuche unterstützen. Die Benutzerprogramme werden in Maschinencode geschrieben ("Object code"). Alle Befehle des NASBUG verlangen daher Operanden in hexadezimaler Form. Alle Daten, Befehle und Adressen müssen in hexadezimaler Schreibweise eingegeben werden.

Nachdem die RESET-Taste betätigt wurde (RS) beginnt NASBUG zu arbeiten. Wie das Betriebssystem initialisiert wird, kann man der Programmliste entnehmen, welche parallel zu dieser Beschreibung gelesen werden kann (S.57ff). Bei RESET wird der Bildschirm gelöscht und das "PROMPT"-Zeichen (Eingabeaufforderung) erscheint in der linken äußeren Ecke der untersten Bildschirmzeile. Sobald dieses Zeichen ausgegeben wurde, erwartet der NASBUG ihren Befehl. Wenn das Eingabeanforderungszeichen erscheint, bedeutet dies außerdem, daß die Hardware (d.h. der Rechner) funktioniert und daß die Variablenfelder des Betriebssystemes ins RAM übertragen worden sind (siehe auch Abschnitt 11).
Nachdem das Eingabeanforderungszeichen ausgegeben wurde, kann der Benutzer einen der folgenden Befehle eingeben:

L	Lade vom seriellen Interface
Baaaa	Setze Breakpoint
Maaaa	Zeige Inhalt einer Speicherzelle an und/oder verändere ihn
Taaaa bbbb	Gib Inhalt der Speicherzellen aaaa bis bbbb aus
Daaaa bbbb	Übertrage den Speicherbereich aaaa bis bbbb über Serienschnittstelle

Caaaa bbbb cccc	Blocktransfer. Copiere den Speicherinhalt beginnend bei dem Block mit der Anfangsadresse aaaa in den Block der bei bbbb beginnt. Die Länge des übertragenen Blockes ist cccc.
Eaaaa	Führe das Programm aus, das bei Adresse aaaa beginnt. aaaa kann implizit angegeben sein.
Saaaa	Einzelschrittbetrieb, beginnend bei Adresse aaaa. aaaa kann implizit angegeben sein.

Beachten Sie bitte:

Zwischen einem Befehl und der ersten zugehörigen Adresse darf kein Zwischenraum sein !

Alle Befehle können mit Backspace korrigiert werden.

Die Kleinbuchstaben in den obigen Erläuterungen sind die Argumente der Befehle und zwar sind es immer Adressen. Ein Befehl kann mit der "BS" (Backspace)-Funktion (der Cursor läuft immer einen Schritt rückwärts und löscht dabei ein Zeichen) teilweise oder ganz gelöscht und/oder verändert werden. Ein Befehl wird erst ausgeführt, wenn die NEW LINE-Taste betätigt wird. Dies gilt für alle Befehle.

Bei Adressen und Daten müssen die führenden Nullen nicht mit eingegeben werden. Also gilt:

Für Adressen (Beispiel)	FA	≙	ØØFA
	B	≙	ØØØB
Für Daten (Beispiel)	1	≙	Ø1

In den folgenden Abschnitten werden Zeichen, die vom Benutzer geschrieben wurden stets unterstrichen.

2.

Maaaa

Um den Inhalt des Schreib/Lesespeichers zu überprüfen oder zu verändern, benötigt man das Kommando M auf das unmittelbar eine Adresse folgt.

Der Benutzer kann eine Folge von Datenbytes eingeben, die in aufeinanderfolgende Speicherplätze geschrieben werden. Die Daten müssen immer in der untersten Zeile der Anzeige stehen. D.h. bevor das Zeilenende überschritten wird, muß die NEW LINE-Taste betätigt werden. Jedes Datenbyte muß vom folgenden Byte durch einen Zwischenraum getrennt sein.

Im folgenden Beispiel sind Eingaben des Benutzers unterstrichen.

Die Buchstabenkombination NL soll anzeigen, daß der Benutzer die NEW LINE-Taste gedrückt hat. Das folgende Beispiel zeigt ein Programm, wie es vom Benutzer eingegeben wird. Die Stellen, an denen die NEW LINE-Taste gedrückt wird kann man sich solange nach Belieben aussuchen, wie das Zeilenende nicht überschritten wird. Da NASBUG immer nur die Zeichen verarbeitet, die in der untersten Zeile des Bildschirms rechts vom Eingabeanforderungszeichen stehen, muß ein Zeilenüberlauf verhindert werden.

```
>MCFA NL
0CFA 00>3A 00 0E NL
0CFD 00>3C NL
0CFE 00>32 00 0E NL
0D01 00>CD 3B 01 CD 35 00 NL
0D07 00>C3 FA 0C NL
0D0A 00>. NL
>
```

Die Ausführung des M-Befehls wird durch einen Punkt . gefolgt von NL abgebrochen.

3.

Taaaa

Der Ausgabebefehl T dient dazu, einen Block von Speicherinhalten auf dem Bildschirm auszugeben. Beachten Sie dabei, daß bei einem Block größer als 68H am oberen Ende des Blockes Daten nicht angezeigt werden. (68 ist für den Rechner eine hexadezimale Zahl und wird 68H oder manchmal auch H '68' geschrieben. Die entsprechende Dezimalzahl ist 6x16 + 8= 104D = D '104' =H '68'=68H = 16 ' 68). Wenn ein längerer Block ausgegeben wird, sind nach Abschluß der Operation nur noch die letzten H '68' (104_{dez}) Speicherinhalte auf dem Bildschirm sichtbar.

```
>TCFA D0A NL
0CFA 3A 00 0E 3C 32 00 0E CD
0D02 3B 01 CD 35 00 C3 FA 0C
.
>
```

4.

Daaaa bbbb

Das D-Kommando ("DUMP") benutzt denselben Code im Betriebssystem wie das T-Kommando. Daher ist auch das Datenformat identisch. Was beim T-Kommando nicht offensichtlich ist, ist die Tatsache, daß eine Textzeile nicht so ausgegeben wird, wie sie eigentlich geschrieben wurde. Die Textzeile wird vielmehr so ergänzt wie unten gezeigt, denn in diesem Format wird die Textzeile auf den Kassettenrecorder überspielt oder von einem Fernschreiber ausgegeben. Das D-Kommando bewirkt die Ausgabe im gleichen Format, das auch das T-Kommando verwendet. Allerdings erfolgt die Ausgabe nicht über den Bildschirm sondern auf den Fernschreiber bzw. die serielle Schnittstelle. Bevor die Übertragung anläuft, sollte der Benutzer den Kassettenrecorder in Stellung "Aufnahme" loslaufen lassen. Während der Übertragung leuchtet die rote Leuchtdiode "Peripheral Drive". Am Blockende verhält sich das System genauso wie beim T-Kommando, nur daß die Leuchtdiode abgeschaltet und das Eingabeanforderungszeichen ausgegeben wird.

Im untenstehenden Ausdruck zeigt die gestrichelte Linie an, an welcher Stelle ein 8-Zeichen-Block zu Ende ist. Hinter der Linie stehen die Zeichen, die zum Kassettenrecorder übertragen werden, aber nicht auf dem Bildschirm erscheinen.

Wie lange der Rechner zum Abarbeiten des D-Kommandos benötigt, hängt von der Taktfrequenz des UART ab. Beim Kassettenrecorder beträgt die Übertragungsgeschwindigkeit etwa 25 Zeichen/Sekunde.

```
>DCFA D0A NL
>

0CFA 3A 00 0E 3C 32 00 0E CD | 97 BS BS NL
0D02 3B 01 CD 35 00 C3 FA 0C | 16 BS BS NL
.
>
```

5.

Caaaa bbbb dddd

Das C-Kommando ("COPY") kopiert den Inhalt eines Speicherblockes, der bei der Adresse aaaa beginnt bis zur Adresse aaaa + dddd in einen Speicherblock, der bei Adresse bbbb beginnt. So wird z.B. das letzte Byte von der Adresse aaaa+dddd in die Adresse bbbb + dddd kopiert.

Beachten Sie bitte: Das C-Kommando ist mit Umsicht anzuwenden. Da ein Datenblock augenblicklich übertragen wird, werden Daten in sich überlappenden Bereichen der Speicherblöcke zerstört ! Um fehlerlos zu kopieren, sollten Sie daher immer die Regel beherzigen:

bbbb muß stets größer sein als aaaa+dddd

Umcopieren kann man, indem man einen freien Bereich verwendet, um einen Datenblock temporär zwischenzuspeichern. Solange der freie Bereich mit aaaa+dddd nicht überlappt, kann das C-Kommando verwendet werden, um einen Block zurück zu kopieren. Diese Methode wird im folgenden Beispiel gezeigt. Ein Datenblock wird "geöffnet" und ein Byte eingefügt. Der Benutzer hat nämlich vergessen, ein Byte in die Speicherzelle H 'EØ5 zu schreiben. Die letzte Adresse des Datenblockes ist H 'E37. Mit dem C-Kommando wird der Block H 'EØ5 bis H 'E37 einschließlich zu einem unbenutzten Speicherblock bei H 'FØØ übertragen. Dieser Block wird dann, nach Einfügen des fehlenden Byte 'Ø6 wieder zurück übertragen, beginnend bei der Adresse H 'EØ6.

Man kann das C-Kommando auch verwenden, um ganze Speicherblöcke mit einem Byte zu füllen. Dazu schreibt man einen C-Befehl, bei dem die Adresse des ersten Blockes um eins niedriger ist, als die Adresse des zweiten Blockes. Das dritte Argument gibt an, wieviele Bytes mit dem Wert geladen werden sollen, den der Inhalt der Speicherzelle aaaa angibt.

```
>TEØØ EØF NL
ØEØØ Ø1 Ø2 Ø3 Ø4 Ø5 Ø6 Ø7 Ø8
ØEØ8 Ø9 ØA ØB ØC ØD ØE ØF 1Ø
>
```

```
>TEØØ EØF NL
ØEØØ Ø1 Ø2 Ø3 Ø4 Ø5 Ø6 Ø6 Ø7
ØEØ8 Ø8 Ø9 ØA ØB ØC ØL ØE ØF
>
```

```
>CEØ5 FØØ 32 NL
>CFØØ EØ6 32 NL
>
```

6.

Eaaaa

Das E-Kommando ("EXECUTE") überträgt den Inhalt der "Register Safe Area" in die internen Register des Z 80 und setzt den Programmzähler auf aaaa. Unter "Register Safe Area" versteht man einen Bereich im RAM, in dem NASBUG die Inhalte der Register A,B,C,...H,L sowie PC ablegt. Nur wenn das Programm auf einen Breakpoint gelaufen ist (siehe Abschnitt 8) muß aaaa explizit angegeben werden. Wird aaaa nicht angegeben, so verwendet das Betriebssystem den letzten angegebenen Wert des Programmzählers um das E-Kommando auszuführen. Dies gilt gleichfalls für das S-Kommando (siehe Abschnitt 7).

```
>ME00 NL
0E00 00>20 NL
0E01 00>. NL
>ECFA NL
!"#$%&'()*+,- (ETC.)
```

Beachten Sie, daß das erste Kommando das Einzelschrittprogramm aufruft. Es wird jedoch nicht ausgeführt, sondern das bei aaaa beginnende Programm wird normal abgearbeitet. Der erst ausgeführte Befehl wird jedoch während er ausgeführt wird durch einen nichtmaskierbaren Interrupt (NMI) unterbrochen. Bei allen folgenden Operationen ist dies nicht der Fall. Das bedeutet aber, daß ein HALT-Befehl nicht ausgeführt wird, wenn er in der Speicherzelle steht, auf die PC beim Ausführen des E-Befehles zeigt.

Das untenstehende Beispiel zeigt, wie der Z 80 einen HALT-Befehl (Adresse H 'ØFØ3) ausführt, aber durch den NMI unterbrochen wird, den NASBUG für den Einzelschrittbetrieb erzeugt. Die nächsten beiden Zeilen zeigen, wie der Z 80 angehalten werden kann. Dieser Zustand wird von der HALT-Leuchtdiode angezeigt.

```
>MF00 NL
0F00 00>0 0 0 76 0 C3 86 2 NL
0F08 00>. NL
>EF03 NL
>                (BACK IN MONITOR LOOP)
>EF02 NL
_                (HALTED).
```

7.

Saaaa

Wie beim E-Kommando, so kann auch beim S-Befehl aaaa implizit angegeben werden. Dieses Kommando ist einer der nützlichsten Befehle, die man braucht um ein Programm zu testen. Er erschien so wichtig, daß die Funktion der Taste NEW LINE erweitert wurde, sodaß man, nachdem einmal Saaaa eingegeben wurde, beim Betätigen der Taste NEW LINE immer der nächstfolgende Befehl abgearbeitet wird. So kann die Einzelschrittausführung durch den NMI (d.h. der Einzelschrittbetrieb wird teilweise Hardware-gesteuert) schrittweise durch das ganze Programm führen. Das S-Kommando kann auch benutzt werden um die Arbeitsweise des ROM-Programmes zu untersuchen, d.h. NASBUG läuft schrittweise durch das Betriebsprogramm.

Wie man die impliziten Adressen beim E-Kommando und beim S-Kommando benutzt, zeigt das folgende Beispiel:

```
>B0 NL
>E0 NL
>BD04 NL
>E0 NL
>SCFA NL
1000 0CFD 2042 FFCF FF00 0600
>S NL
1000 0CFE 2120 FFCF FF00 0600
> NL
1000 0D01 2120 FFCF FF00 0600
>ME00 NL
0E00 21>. NL
>S NL
0FFE 013B 2120 FFCF FF00 0600
>E NL
!1000 0D04 2120 FFCF FF00 0600
>E NL
"1000 0D04 2220 FFCF FF00 0600
>E NL
#1000 0D04 2320 FFCF FF00 0600
>B0 NL
>E NL
$%&'()*+,-
                (ETC.)
```

Jedesmal wenn ein Schritt ausgeführt wird, werden die Inhalte der folgenden internen Z 80 Register auf dem Bildschirm ausgegeben:

SP	PC	AF	HL	DE	BC
Stapel-zeiger	Programm-zähler	Register-paar AF Akkumulator und Flags	Register-paar HL	Register-paar DE	Register-paar BC

8. Baaaa

Immer wenn der NASCOM 1 eingeschaltet wird, sollte der Breakpoint auf die Adresse ØØØØ gesetzt werden. Bei RESET wird die Breakpointadresse nicht automatisch auf ØØØØ gesetzt, da sonst diese Adresse immer verloren ginge, wenn man einen RESET benutzt um ins Monitorprogramm zurück zu springen. Da der Prozessor nicht anders als durch einen RESET in den Grundzustand gebracht werden kann, sollte sich der Benutzer angewöhnen, nach dem ersten RESET den Breakpoint auf Adresse Ø zu setzen.

Wie man das B-Kommando einsetzt, wurde bereits im Abschnitt 7 gezeigt. Der Zweck des Breakpoint ist es, ein Programm zu unterbrechen und dem Benutzer die Inhalte der Z 80 Register sowie PC und SP anzuzeigen. Dazu baut das Betriebssystem den Restartbefehl H 'E7 in die Speicherzelle des RAM ein, die aaaa angibt und transferiert das Byte, das in aaaa steht, zur Adresse ØC17. Das Programm wird abgearbeitet bis es auf den Breakpoint trifft. Am Breakpint wird die Programmkontrolle an NASBUG übergeben und zwar zur Breakpointbehandlungsroutine bei Adresse ØØ2Ø. Diese bringt die aktuellen Registerinhalte auf den Bildschirm und zwar im gleichen Format wie beim S-Kommando. Der ursprüngliche Code wird wiederhergestellt wenn man BØ eingibt, d.h. einen neuen Breakpoint bei Adresse ØØØØ setzt.

Achtung: Auf einen Breakpoint-Befehl muß immer ein E-Befehl folgen (Siehe Beispiel unter Abschnitt 7).

9.

L

Das L-Kommando ("LOAD") läßt ein Programm ablaufen, das die Daten in der letzten Zeile des Bildschirms im gleichen Format wie beim D-Kommando interpretiert. Nur mit dem Unterschied, daß die Daten beginnend bei der am Anfang der Zeile angegebenen Adresse in den Speicher übernommen werden. Der Vorgang läuft genau umgekehrt ab wie beim Abspeichern allerdings mit folgender Besonderheit: Wenn das Prüfsummenbyte einen Fehler anzeigt wird die Zeile nicht in den Speicher geladen, sondern auf dem Bildschirm um eine Zeile nach oben geschoben. Somit stehen am Ende des Ladevorganges (der erkannt wird, wenn das "end-of-dump"-Zeichen gelesen wurde) nur noch fehlerhafte Zeilen, die wegen Band-oder Kassettenrecorderfehlern nicht in den Speicher geladen wurden, auf dem Bildschirm. Wenn man das Band zurückspult und neu lädt, können solche Fehler in der Regel korrigiert werden. Wenn man feststellt, daß immer wieder die gleich Zeile nicht geladen werden kann, muß man eine neue Aufnahme anfertigen und/oder die Zeile mithilfe des M-Kommandos neu schreiben.

10.

Der NASBUG

Die Einzelschritt-Hardware beansprucht den NMI-Eingang des Prozessors, sodaß dieser Eingang und die zum NMI gehörigen Befehle ohne Änderungen der Hardware nicht für den Benutzer zugänglich sind. NASBUG wurde so ausgelegt, daß er für den Benutzer von möglichst großem Nutzen ist. Alle Kommandos rufen nämlich Unterprogramme auf, die auch für die Benutzersoftware mit verwendet werden können.

Die Kommando-Tabelle ist so aufgebaut, daß sie erweitert werden kann, d.h. der Anwender kann eigene Kommandos schreiben und sie zusammen mit NASBUG - Befehlen verwenden.

Die Tastaturcodiertabelle ist ebenso angelegt. Sie kann neu geschrieben und im RAM abgelegt werden wobei den einzelnen Tasten neue Codes zugewiesen und/oder die Anzahl der Tasten erhöht werden kann.

Beachten Sie bitte, daß die LICON-Tastatur eine spezielle Beschaltung verwendet. Wenn man zusätzliche Tasten einbauen möchte, benötigt man auch zusätzliche Hardware um diese Tasten lesen zu können.

11.

Folgende Unterprogramme im NASBUG können vom Anwender benutzt werden:

KDEL Adr.	0035	bewirkt 6 ms Verzögerung
MOTFLP	0051	Schaltet "Peripheral Drive Leuchtdiode ein/aus (Flipflopverhalten)
SRLOUT	005D	Überträgt Akkumulatorinhalt an UART
KBD	0069	Wenn ein Zeichen von der Tastatur eingegeben wurde ist nach dem Rücksprung aus KBD das Carryflag gesetzt. Das Zeichen steht in A.

Wichtige Betriebssystemvariablen:

$KTAB Dies ist die im RAM stehende Adresse der neuen Tastaturcodiertabelle.

Beachten Sie bitte, daß sowohl CRT als auch KBD mit NASBUG durch einen Sprung ins RAM übergeben werden. Das bedeutet, daß der Benutzer die Speicherinhalte $CRT und $KBD ändern kann, um den NASBUG diese beiden Subroutinen irgendwo anders (im Benutzer-RAM) suchen zu lassen.

CRT	Ø13B	Das Zeichen in A wird auf den Bildschirm gebracht oder der Cursor vorgerückt (BS oder CR) oder der Bildschirm gelöscht (FF).

Es gibt einen Sprung bei $CTAB der dem NASBUG die Anfangsadresse seiner eigenen Kommandotabelle angibt. Das bedeutet, daß der Benutzer den Befehlssatz erweitern kann, wenn er diesen Sprung ändert. Die vorhandenen Befehle können ebenfalls in diese Tabelle aufgenommen werden.

CHIN	ØØ3E	Dieses Programm ruft $KBD auf und überprüft, ob in das UART von der Serienschnittstelle aus Daten übertragen wurden. Wenn Daten erscheinen, übergibt das Unterprogramm ein Carrybit und das Zeichen in A.

$NMI ist die Adresse, die beim nicht maskierbaren Interrupt verwendet wird. Auch sie kann verändert werden.

12.

Dieses Programmbeispiel, es wurde schon in Abschnitt 7 benutzt, zeigt die Anwendung einiger Unterprogramme des NASBUG. Eine Liste des Assemblerprogrammes sieht so aus:

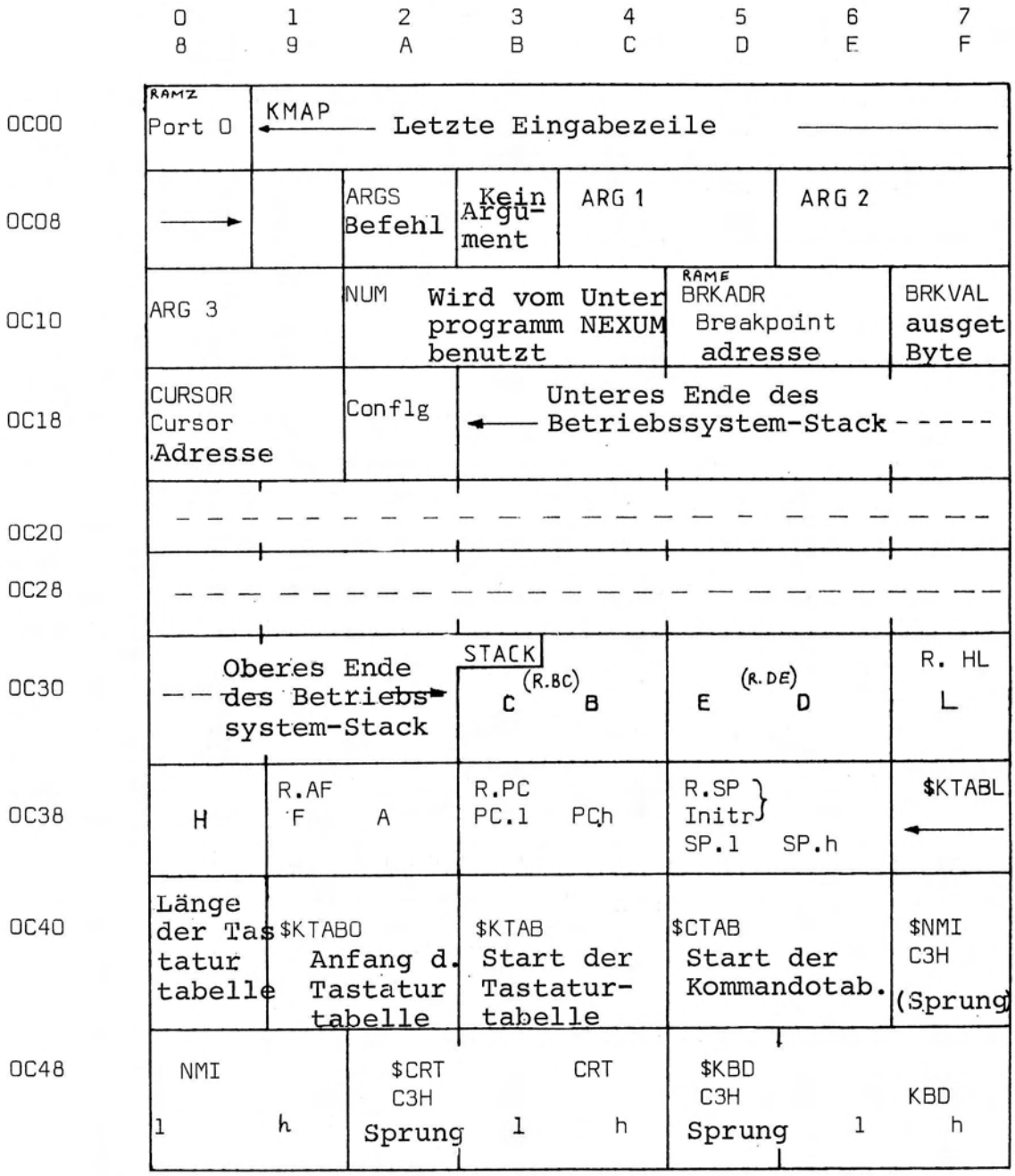
```
        ORG H 'ØCFA'     Programm beginnt bei CFA
CRT     EQU H 'Ø13B'
KDEL    EQU H 'ØØ35'
REG     EQU H 'ØEØØ'
LOOP    LD A,REG
        INC A
        LD REG,A
        CALL CRT         Rufe Unterprogramm bei Adresse 13BH
        CALL KDEL        Rufe Unterprogramm bei Adresse 35H
        JP LOOP          Springe zum ersten Befehl zurück
```

Nur die Befehle ab der Marke LOOP werden tatsächlich in Maschinencode umgesetzt. Der Code wird so eingegeben wie in Abschnitt 2 gezeigt.

13.

Ohne Speichererweiterung stehen dem Benutzer für seine Programme die Speicherplätze ØC5Ø bis ØFEØ zur Verfügung. Die obere Grenze richtet sich nach der Tiefe des Stack.

14: Aufteilung des von NASBUG benutzten RAM-Speicherbereiches



Vom NASBUG verwendeter Speicher

(Um die entsprechende Tabelle auf dem Bildschirm ausgeben zu lassen, schreiben sie folgenden Befehl:

TCØØ C4F NL)

Der erste Speicherplatz, den der Benutzer verwenden kann ist ØC5Ø.

Teil 17: Das NASBUG MONITOR PROGRAMM

A: Bemerkungen zum Z 80 MONITOR

ØØØ START setze Stapelzeiger auf C33
lösche RAMZ bis RAME - 1 (CØØ bis C14)
initialisiere C3D bis C4F
lösche Bildschirm
springe nach:

359 STRTO: entferne Breakpoint
springe nach:
PARSE

---

286 PARSE lies eine Zeile ein
(die zweite Zeile auf dem Bildschirm ist nun die Zeile)
Wenn Zwischenraum erkannt wird, dann wie letzter Befehl
wenn "S" dann setze "S" ein (Einzelschrittbefehl)
Speichere Befehlsbuchstabe in ARGS (CØA)

29E PLOOP: Rufe NEXUM auf, um das Argument des Befehls zu bekommen;
wenn es eines gibt, schreibe es nach ARG1
gehe nach PLOOP:
Schreibe die Anzahl der Argumente nach ARGS+1
Suche in CTAB nach dem Befehlsbuchstaben
Wenn er nicht gefunden wurde, gehe nach PARSE.
Ansonsten lege PARSE auf dem Stapel ab, Springe
zum gewünschten Unterprogramm (CALL-Aufruf).

2DØ EXEC: ↓ Setze Conflg=-1

Oder

2FF STEP: ↓ Setze Conflg =Ø

EXEC1: Reinitialisiere NMI (C48)

(Nur dann, wenn vom Benutzer geändert)

Kehre nicht zu PARSE zurück (RETURN überspringen)

Wenn ein Argument vorhanden ist, schreibe es in R.PC
Hole BC,DE,AF aus dem Stack (AF ist der alte HL-Inhalt)
Hole Benutzer-Stapelzeiger aus dem Stack.
Lege Benutzer-Programmzähler auf den Stack.
Hole HL aus dem Stack.
Sichere Inhalt von AF im Stack während NMI.
RETN zum Benutzerbefehl.

Befehl
↓
NMI
↓
TRAP

305 TRAP Addiere 1 zum Benutzer-Programm auf dem Stack
(Wird später in einem Programmteil der TRAP
und dem Breakpoint gemeinsam ist wieder dekrementiert).
Sichere AF; HL im Stack
Lösche NMI-Flag an Port Ø
Wenn CONFLG nicht = Ø war (d.h. es kam ein "E")
sichere Benutzerbefehl an der Breakpointadresse.
Setze RST 4 in die Breakpointadresse ein.
Hole HL,AF aus dem Stack, dekrementiere PC,
RETN zum Benutzercode.
Im anderen Falle:
Sichere DE ; springe nach
BPTI (326)

Anfangsadresse für
die Breakpoint-
Behandlungsroutine: Ø2Ø

Sichere AF,HL,DE im Stack
dann folgt:

326 BPTI: Sichere BC; Lade SP in HL
Kopiere die Register vom Benutzer-Stack
in die "Register Safe Area" im Monitor RAM

```
                Dekrementiere den Benutzer-Programmzähler, sodaß
                er auf die Breakpointadresse zeigt.
                Gib die Inhalte der Benutzerregister aus.

347 REGSI:

359 STRTO:      Setze die ursprüngliche Instruktion wieder in
                die Breakpointadresse ein und springe zu
                PARSE

Ø69 KBD         Sichere Register
                Lösche Zähler
                Stelle Zeiger auf Tabelle (Tastatur)
                Lies Zeile Ø (Shift)
                in die Speicherzelle KMAP

KSC1:           Inkrementiere Zähler
                Inkrementiere Tabellenzeiger, prüfe ob Zustands-
                wechsel aufgetreten ist.
                Bei Zustandsänderung gehe nach KSC2

KSC1A:          Wiederhole acht mal

KSC8:           Lösche Carryflag, wenn keine Taste gedrückt
                wurde.

KSC9:           Lade Register mit alten Werten, kehre zum
                Hauptprogramm zurück.

KSC2:           Verzögerung (KDEL, Ø35)
                Lies erneut.
                Errechne Spaltennummer des Bit, das sich geändert
                hat, (C)
                Bit-Maske (D) anwenden.
                Überprüfe ob Änderung tatsächlich auftreten ist,
                wenn kein Sprung zu KSC 1A vorlag.
                Bringe Tabelle auf den neusten Stand.
                Bei Freigabe springe nach KSC 1A.
                Errechne Zahl aus SHIFT, Zählerstand und Bitnummer.
                Suche in KTAB nach dieser Zahl.
```

(Wenn sie nicht gefunden wurde, lösche SHIFT-Bit und versuche es nochmal).
Wenn sie immer noch nicht gefunden wurde, gehe zu KSC8.
Errechne den ASCII-Code mithilfe der Adresse in KTAB (ØEA).

ØEØ KSC3: Setze Carryflag (Buchstabe gefunden), Kehre zum Hauptprogramm zurück.

Ø3E CHIN: Frage Tastatur und UART ab, bis ein Zeichen empfangen wird.

Ø5D SRLOUT: Schreibe ein Zeichen zum UART. Warte bis es gesendet worden ist.

Ø5B FLIP: Setze ein Bit in Port Ø.

Ø4A FLPFLP: Setze ein Bit in Port Ø und setze es wieder zurück.

Ø35 KDEL: Verzögerung; Mit PUSH und POP wird die Schleife zeitlich gedehnt.

13B CRT: Ignoriere Zeichen Ø.
Sichere Register.
FF? ja. Schreibe - 1 in die linke obere Ecke,
15mal wiederh. { führe dann 48 Leerbefehle aus (Zwischenraum)
dann 16 mal Ø
dann 48 Zwischenräume }
setze dann - 1 in die rechte untere Ecke.

CRTØ: Setze HL in die linke untere Ecke.

CRT1: SEtze den Cursor auf den Bildschirm, sichere Cursoradresse.

CRT2: Hole alte Registerinhalte zurück, Kehre zum Hptprg zur.
Ersetze Cursor durch Zwischenraum.
BS? → dekrementiere Cursor, lösche Zeichen.
Wenn -1 erreicht, inkrementiere Cursor wieder;
Springe nach CRT1 um den Cursor wieder auf den Bild-

schirm zu bringen. Kehre zum rufenden Prog. zurück.
CR? → CRT3 (scroll)
Normaler Buchstabe: Schreibe ihn auf den Bildschirm
Cursor springt über Bildschirmgrenze.
-1 erreicht? nein → kehre zurück über CRT1

195 CRT3: Scroll; (Bildschirminhalt eine Zeile nach oben schieben).
Lösche unterste Zeile und springe nach CRTØ,
um den Cursor zurück zu setzen.

1DB INLINE: Gib Eingabeanforderungszeichen aus

1DE INLØ Lies Buchstaben; BS? → INL2;
CR? → springe zurück über CRLF

1E9 INL1: Gib Buchstaben aus; → INLØ

1EE INL2: BS?
Prüfe ob Promt-Zeichen (Eingabeanforderungszeichen)
hier steht, wenn ja gehe zu INLØ
anderenfalls gehe zu INL1 um letztes Zeichen zu
löschen.

(RST 5)

Ø28 PRS: Hole Rückkehradresse
Sende Zeichen bis Ø an den Bildschirm.
Nach Ø kehre zum Befehl zurück (Rufendes UP).

---

224 B2HEX: Sichere Zahl; schiebe obere Hex-zahl nach unten
Rufe B2HEX1 (um auszugeben)
Hole alten Wert zurück.

24D Gib untere Hex-zahl aus, Kehre zum rufenden Prog.
zurück.

25A NEXUM: DE zeigt zur aktuellen Zeile auf dem Bildschirm.
NUM = Anzahl der Stellen der Zahl
NUM + 1; NUM + 2 ← Zahl

1AD MODIFY Hole Adresse von ARG1

1BØ MODI Gib Adresse aus

```
                Rufe INLINE, um eine Zeile einzulesen
1BF   MOD2:     Zahl ? ——→ Verändere Speicherinhalt
                           Inkrementiere Adresse ——→ MOD2
1CF             "." ?, Springe zurück
                Irgendwelche Zahlen in der Zeile? nein→inkremen-
                tiere Adresse und  gehe zu MOD1.
```

```
1FC TABCDE:     Adresse = (ARG1)

    TBCD 1:     Adresse = ARG2?
                                Ja——→schreibe "."; Kehre zurück
                Lösche Prüfsummenwort.
                Schreibe Adresse (TBCD3)
216 TBCD1A:     Gib Speicherinhalt aus (TBCD2 errechnet Prüfsumme).
                Inkrementiere Adresse
                Wiederhole acht mal von 1A:
                Schreibe Prüfsumme
                Lösche Prüfsumme mit BS und gehe zu
                TBCD1:
22B TBCD2:      Errechne Prüfsumme, schreibe sie hexadezimal aus.
232 TBCD3:      Schreibe Adresse unter Benutzung von TBCD2.
```

```
3D1 DUMP:       Schalte Motor ein und rufe
                Zeitverzögerung   auf.
                Bringe $CRT zu SRLOUT
                Rufe TABCDE auf.
                Hole $CRT nach CRT zurück
                Kehre über das Programm, das den Motor ausschaltet
                zum Hauptprogramm zurück.
```

37C	LOAD:	Schalte Motor ein
	LOD1:	Auf Anfang der Zeile Zeiger einstellen.
	LOD1B:	Lies Zeichen; Ignoriere BS Kein CR? Springe zum Ausgabeprogramm, dann zurück zu LOD1B
	LOD1A:	Zeiger auf den Zeilenanfang stellen "." ? Gehe zu MOTFLP um Motor auszuschalten und kehre zum rufenden Programm zurück Lies Zahlen in die obere linke Bildschirm- grenze ein. Prüfe Prüfsumme. Wenn fehlerhaft, scroll, gehe dann zu LOD1 Wenn nicht, kopiere in den Speicher ⟶ kehre zurück über LOD1.
3EF	COPY	Blocktransfer von dem Block, der bei ARG1 beginnt, zu dem Block der bei ARG2 beginnt. Anzahl der Bytes in ARG3.

Vorbemerkung zu Schreibweisen und Vereinbarungen.


```
HEXASM V005 ASSEMBLY ON 15-FEB-78 AT 21:33.     PAGE 1      ,LP:<CSDOC
DK:CSDOC.SRC    HEXASM NOTES
```

HEXASM NOTES

Alle Zahlen werden in dezimaler Form angegeben. Werden Zahlen in anderen Zahlensystemen verwendet, gibt man sie in der Form an:

Basis ' Zahl.

'A ist ASCII "A".

```
Numbers are normally in decimal.
Numbers in other bases are specified by:
   base'number      e.g.  16'FF = 2'11111111 = 255

'*' is the 'bit' operator, eg  *7 = 16'80,  *0 = 1

'A gives the ASCII code for A, i.e. 65

Angle brackets  <  >  are used as brackets within expressions

the statement   FRED=3;   equates the symbol 'FRED' with 3

'.' is the location counter,  e.g.
   .=3    sets the location counter to 3  (  ORG 3 )

RAM. specifies the start of RAM
.ROM; .RAM; switch between ROM and RAM

';' is used to separate statements

0000 010203          .WORD  1,2,3;    [ assembles 3 words containing 1, 2 & 3]
0003 FFFF0100        .ADDR  -1,1;     [ assembles as 2 double-words (low order first)
0007                 .BLKW  3;        [ reserves 3 words os RAM
000A                 .BLKA  2;        [ reserves 2 double words

000E 78              A~B;             [ is equivalent to  LD A,B
000F 3E03            A~#3;            [ is equivalent to  LD A,3
0011 3A0700          A~FRED;          [ is equivalent to  LD A,(FRED)

                     load's which have no direct Z-80 equivalent are treated as two instructions;
                     the first loads the accumulator, the seond storin the value.  e.g.

0014 3E053207003E0532 FRED~#5;        [=]            A~#5; FRED~A;

                     other shorthand's are:

001E B7B7            TSTA;    [=]     OR A;
0020 AFAF            CLA;     [=]     XOR A;


                     the following opcodes have different names in the Z-80 manual:

                     CMA;      =      CPL
                     DBNZ LABEL; =    DJNZ LABEL;
                     J LABEL;    =    JP LABEL;
                     BR LABEL;   =    JR LABEL;
```

operations which normally take one argument can be given more than one, with the expected result. e.g.:

Operationen, die normalerweise nur ein Argument haben, können mit mehr als einem Argument ausgestattet werden. Siehe Beispiel in Zeile 0022.

```
0022 F5C5D5F5C5D5     PUSH AF,BC,DE;     [=]  PUSH AF; PUSH BC; PUSH DE;
```

the code: *der Code:*

```
0028 20033A0700               IF Z; A~FRED; FI;
```

is equivalent to: *ist äquivalent zu:*

```
002D 20033A0700               BR NZ lab1; A~FRED;
0032                  lab1:
```

the code: *der Code:*

```
0032 2005                     IF Z;
0034 3A0700                    A~FRED;
0037 1803                     ELSE;
0039 3A0600                    A~JIM;
003C                          FI;
```

is equivalent to *ist äquivalent zu:*

```
003C 2005                     BR NZ lab2;
003E 3A0700                    A~FRED;
0041 1803                      BR lab3;
0043 3A0600           lab2:    A~JIM;
0046                  lab3:
```

in IF FRED Z; the code A~FRED; TSTA; is inserted before the first branch.

```
HEXASM V005 ASSEMBLY ON 15-FEB-78 AT 21:33.      PAGE 3        ,LP:<CSDOC
DK:CSDOC.SRC     HEXASM NOTES

                         JIM=6
                         FRED=7
  0046                   .END
```

```
HEXASM V005 ASSEMBLY ON 15-FEB-78 AT 21:33.      PAGE 4        ,LP:<CSDOC
SYMBOL TABLE

FRED   =0007
JIM    =0006
LAB1   :0032
LAB2   :0043
LAB3   :0046
RAMTOP:0000
RAM.   =0000
ROMTOP:0046
ROM.   =0000
       =0046
```

Symboltabelle

Das im folgenden Abschnitt aufgelistete Programm wird in dieser Form im NASBUG-Monitor-PROM geliefert.

```
HEXASM V005 ASSEMBLY ON 12-APR-78 AT 17:02.     PAGE 1      ,LP<CSMON
DK:CSMON.SRC    Z-80 Monitor

                        RAM.=16'C00

                          initialise stack pointer and RAM   Initialisiert Stapelzeiger und RAM.

4206 0000 31330C        START:  SP_#STACK
     0003 21000C0615            HL_#RAMZ; B_#RAME-RAMZ;
     0008 36002310FB            (HL)_#0; INC HL; DBNZ .;
                                set reflections
     000D 212801113D0C          HL_#INITT; DE_#INITR;
     0013 011300EDB0            BC_#INITE-INITT; LDIR;
                                initialise crt
     0018 3E1ECD3B01            A_#FF; CALL CRT;

     001D C35903                J STRT0;

                         BREAKPOINT RESTART
     0020 F5E5D5C32603          PUSH AF,HL,DE; J BPT1;
     0026 00                    NOP;
     0027 00                    NOP;

                         RST 5 = PRINT FOLLOWING STRING, TERMINATED BY 00   Schreibe den folgenden
                                                                            String, abgeschlossen mit 00.
     0028 E3            PRS:    EX (SP),HL;
     0029 7E23B7        PRS1:   A_(HL); INC HL; TSTA;
     002C 2805CD4A0C18F6        IF NZ; CALL $CRT; BR PRS1; FI;
     0033 E3C9                  EX (SP),HL; RET;

                            keyboard debounce delay routine   Verzögerungsroutine zur Tastaturentprellung.
     0035 AF            KDEL:   CLA;
     0036 F5F1F5F13D20F9C9      PUSH AF; POP AF; PUSH AF; POP AF; DEC A; BR NZ .; RET;

                             read a char from keyboard or uart (first come first served)
     003E CD4D0CD8      CHIN:   CALL $KBD; RET CS;
X    0042 DB 02 17 30 F7        IN A, (02); RLA; JRNC - 07;        Lies Zeichen von der Tastatur oder
     0047 DB 01 09              IN A, (01); RET                    vom UART.

                                set & reset a bit in I/O port 0
     004A F5CD5300F11802FLPFLP: PUSH AF; CALL FLIP; POP AF; BR FLIP;   Setze Bit in Port 0 oder setze es
                                                                       zurück.
                                start or stop motor
     0051 3E10          MOTFLP: A_#*4;
                                flip a bit in port 0
     0053 E521000CAE    FLIP:   PUSH HL; HL_#PORT0; XOR (HL);   Schalte Motor ein und aus.
     0058 D30077E1C9            OUT 0,A; (HL)_A; POP HL; RET;

                            put character out thru UART, and wait till sent
     005D D301          SRLOUT: OUT 1,A;
     005F DB0287F818FA          IN A,2; ADD A; RET M; BR .;     Gib Zeichen ans UART und warte, bis es ge-
                                                                sendet wurde.
     0065 00                    NOP; [padding]

                          NMI VECTOR
     0066 C3470C                J $NMI;    Interruptvektor NMI.
```

Unterprogramm zum Lesen der Tastatur.

Das Carryflag wird gesetzt, wenn ein Zeichen verfügbar ist. Der ASCII-Buchstabe/Zeichen wird in A übergeben. Ausnahmen sind die Zeichen:

```
                        routine to read from keyboard
                        carry is set if a char. is available
                        the standard ASCII code for the char is returned in A
                        EXCEPT FOR the following chars
                                BS=16'1D   backspace
                                CR=16'1E   carriage return (=newline)
                                FF=16'1F   form feed =clear screen
0069 C5D5E5             KBD:    PUSH BC,DE,HL;
006C 3E02CD4A00                 A_#*1; CALL FLPFLP;
0071 21010CDB002F77             HL_#KMAP; IN A,0; CMA; (HL)_A;
0078 0608                       B_#8;
007A 3E01CD4A00         KSC1:   A_#*0; CALL FLPFLP;
007F 23DB002F57AE2007           INC HL; IN A,0; CMA; D_A; XOR (HL); BR NZ KSC2;
0087 10F1               KSC1A:  DBNZ KSC1;
0089 B7                 KSC8:   TSTA;
008A E1D1C1C9           KSC9:   POP HL,DE,BC; RET;
008E CD3500             KSC2:   CALL KDEL;
0091 DB002F5F7AAE               IN A,0; CMA; E_A; A_D; XOR (HL);
0097 0EFF160037                 C_#-1; D_#0; STC;
009C CB120C1F30FA               RL D; INC C; RRA; BR NC .:
00A2 7AA35F                     A_D; AND E; E_A;
00A5 7EA2BB28DD                 A_(HL); AND D; CMP E; BR Z KSC1A;
00AA 7EAA77                     A_(HL); XOR D; (HL)_A;
00AD 7BB728D6                   A_E; TSTA; BR Z KSC1A;
00B1 3A010CE610B0               A_KMAP; AND #*4; OR B;
00B7 878787B1                   ADD A; ADD A; ADD A; OR C;
00BB ED4B3F0C2A430CEDB1         BC_$KTABL; HL_$KTAB; CPIR;
                            check again for unshifted character   prüfe nochmals für Zeichen ohne SHIFT
00C4 280B                       IF NZ;
00C6 2A430CED4B3F0C              HL_$KTAB; BC_$KTABL;
00CD E67FEDB1                    AND #16'7F; CPIR;
00D1                            FI;
00D1 20B6                       BR NZ KSC8;
00D3 ED4B430C37ED42             BC_$KTAB; STC; SBC HL,BC;
00DA ED4B410C097D               BC_$KTAB0; ADD HL,BC; A_L;
00E0 3718A7             KSC3:   STC; BR KSC9;

                            set breakpoint address                setze Breakpointadresse
00E3 2A0C0C22150CC9     BREAK:  HL_ARG1; BRKADR_HL; RET;
```

Die Tabellenadressen stellen die Schlüsselzahlen für jeden ASCII-Code dar. Jede Eingabe erfolgt im Format SRRRRCCC .Dabei zeigt S= 1 an, daß die SHIFT-Taste betätigt wurde.
RRRR=8-Zeilennummer (Zählerinhalt)
Hex FF zeigt, daß kein Zeichen gelesen wurde, das in diese Tabelle gehört.
CCC ist die Spaltennummer (Bit-nummer). Wenn die SHIFT-Taste gedrückt wurde und kein Code für die Taste gefunden werden konnte, untersucht das Program, ob evtl. die SHIFT-Taste oben steht.

```
                        table entries represent key number for each ASCII code
                            appearing in ASCII order starting at code 16'1D
                            Each entry is in the format SRRRRCCC
                            where S=1 implies that shift key must be down
                            RRRR=8-row number (number in counter)
                            CCC=column number (bit number)
                        Setting all ones (16'FF) implies that there is no key
                            for this code
                        If the shift key is down and no code is found,
                            then the table is searched again as if
                            the shift key were up.

00EA 088809             KTAB:   .WORD 16'08,16'88,16'09;                                [   BS, FF, CR
00ED 149C9BA392C2BAB2           .WORD 16'14,16'9C,16'9B,16'A3,16'92,16'C2,16'BA,16'B2; [ SPACE -   '
00F5 AAA298A0290A2119           .WORD 16'AA,16'A2,16'98,16'A0,16'29,16'0A,16'21,16'19; [   (   -   /
00FD 1A1C1B2312423A32           .WORD 16'1A,16'1C,16'1B,16'23,16'12,16'42,16'3A,16'32; [   0   -   7
0105 2A22182[illegible]99       .WORD 16'2A,16'22,16'18,16'20,16'B1,16'8A,16'B9,16'99; [   8   -   ?
010D 0D2C41133B334310           .WORD 16'0D,16'2C,16'41,16'13,16'3B,16'33,16'43,16'10; [   @   -   G
0115 402D383028313925           .WORD 16'40,16'2D,16'38,16'30,16'28,16'31,16'39,16'25; [   H   -   O
011D 1D2415344535112B           .WORD 16'1D,16'24,16'15,16'34,16'45,16'35,16'11,16'2B; [   P   -   W
0125 443D3C                     .WORD 16'44,16'3D,16'3C;                               [   X,  Y,  Z
```

```
                        RAM workspace area      Arbeitsbereich des RAM

0128                    .RAM
0C00                    RAMZ:       [ this part cleared on RESET ]      dieser Teil wird mit RESET gelöscht.
0C00                    PORT0:  .BLKW;  [ copy of output port 0 ]       Kopie der Ausgabe auf PORT 0.
0C01                    KMAP: .BLKW 9; [ keyboard switch state table ]  Tastaturzustandstabelle
                          argument list set up by PARSE ]
0C0A                    ARGS:   .BLKW 2;                                Liste der Argumente, die von PARSE er-
0C0C                    ARG1: .BLKA;                                    zeugt wird.
0C0E                    ARG2: .BLKA;
0C10                    ARG3: .BLKA;
0C12                    NUM: .BLKW 3;

0C15                    RAME: [ end of cleared RAM ]                    Ende des gelöschten RAM-Bereiches
0C15                    BRKADR: .BLKA; BRKVAL: .BLKW;
0C18                    CURSOR: .BLKA;  [CRT CURSOR ADDRESS   Cursoradresse
0C1A                    CONFLG: .BLKW;
0C1B                            .BLKA 12; [Monitor's stack space]       Monitorstack
0C33                    STACK: .BLKA;
0C35                            .BLKA;
0C37                    R.HL:   .BLKA;
0C39                    R.AF:   .BLKA;
0C3B                    R.PC:   .BLKA;
0C3D                    INITR:
0C3D                    R.SP:   .BLKA;
                          reflections
0C3F                    $KTABL: .BLKA; [ reflected length of KTAB]
0C41                    $KTABO: .BLKA; [ reflected offset to first character in KTAB]
0C43                    $KTAB: .BLKA;
0C45                    $CTAB: .BLKA;
0C47                    $NMI: .BLKW 3
0C4A                    $CRT: .BLKW 3;
0C4D                    $KBD: .BLKW 3;

0C50                    .ROM;

                          reflection initialisation table       Tabelle für Initialisierung der Systemvariablen

0128 0010               INITT:  .ADDR 16'1000; [END OF RAM
012A 3E00                       .ADDR 64+3-5; [$KTABL]
012C 1D00                       .ADDR 32-3; [$KTABO]
012E EA006303                   .ADDR KTAB,CTAB;
0132 C30503                     J TRAP;
0135 C33B01                     J CRT
0138 C36900                     J KBD
013B                    INITE:
```

```
                    CRTRAM=16'800; [ CRT ram addr ]          CRT RAM Adresse
                    CUR='_; [cursor character = underline]   Cursor = Strich
Zwischenraumzeichen BL=32; [space]
Neue Zeile          CR=31; [NEWLINE]
                    FF=30; [SHIFT+BS= CLEAR SCREEN]  Shift & BS ≙ Bildschirm löschen
Letztes Zeichen löschen BS=29; [BACKSPACE ]

                    CURLIN=CRTRAM+10+<14*64>; [current CRT line]  Aktuelle CRT-Zeile
                    LINE=CURLIN-64; [start of previous line]      Anfang der vorhergehenden Zeile
Dieses Programm bringt  Routine puts a char on screen
ein Zeichen auf den         margins of screen contain zeroes except for top left
Bildschirm. Die Grenzen     and bottom right which contain -1.
des Bildschirm enthalten    FF  initialses screen and puts cursor on bottom line
0 mit Ausnahme der          BS  backspaces
oberen                      CR  carriage returns and line feeds
 013B B7C8          CRT:    TSTA; RET Z; [ignore nulls]
 013D F5C5D5E5              PUSH AF,BC,DE,HL;
 0141 FE1E                  CMP #FF;
 0143 202F                  IF Z;
                                [initalise screen]        Bildschirm initialisieren
 0145 21090836FF             HL_#CRTRAM+9; (HL)_#-1;
 014A 230630                 INC HL; B_#48;
 014D 36202310FB             (HL)_#BL; INC HL; DBNZ .;
 0152 0610                   B_#16;
 0154 36002310FB             (HL)_#0; INC HL; DBNZ .;
 0159 EB210A08               EX DE,HL; HL_#CRTRAM+10;
 015D 01B003EDB0             BC_#15*64-16; LDIR;
 0162 3EFF32BA0B             CRTRAM+<14*64>+58_#-1
 0167 218A0B        CRT0:   HL_#CURLIN;
 016A 365F22180C    CRT1:   (HL)_#CUR; CURSOR_HL;
 016F E1D1C1F1C9    CRT2:   POP HL,DE,BC,AF; RET;
 0174                       FI;
                            [remove cursor]
 0174 2A180C3620            HL_CURSOR; (HL)_#' ;          Cursor entfernen
 0179 FE1D                  CMP #BS;
 017B 200B                  IF Z;
                              [ backspace (thru margins if necessary)]   Rücklauf, falls nötig auch über
 017D 2B7EB728FB              DEC HL; A_(HL); TSTA; BR Z .;              die Zeilengrenzen hinweg.
 0182 3C20E5                  INC A; BR NZ CRT1;
 0185 2318E2                  INC HL; BR CRT1;
 0188                       FI;
 0188 FE1F2809              CMP #CR; BR Z CRT3;
                              [ put char on screen, scroll if necessary]   Setze Zeichen auf den Bildschirm.
 018C 77                    (HL)_A;                                       Scroll wenn nötig.
 018D 237EB728FB            INC HL; A_(HL); TSTA; BR Z .;
 0192 3C20D5                INC A; BR NZ CRT1;
                              [ scroll]
 0195 110A08214A08  CRT3:   DE_#CRTRAM+10; HL_#CRTRAM+10+64;
 019B 017003EDB0            BC_#14*64-16; LDIR;
 01A0 211000                HL_#16;
 01A3 190630                ADD HL,DE; B_#48;
 01A6 36202310FB            (HL)_#BL; INC HL; DBNZ .;
 01AB 18BA                  BR CRT0;
```

linken und der untersten rechten Speicherzelle, die -1 enthalten. FF initialisiert Bildschirmprogramm und setzt Cursor in die unterste Zeile.

```
                        memory modify, arg1=address   Modifiziere Speicherinhalt; ARG1 = Adresse

01AD 2A0C0C         MODIFY: HL_ARG1;
01B0 CD3202         MOD1:   CALL TBCD3,
01B3 7ECD4402               A_(HL); CALL B2HEX;
01B7 CDDB0111520B0600       CALL INLINE; DE_#LINE+8; B_#0;
                              note that line starts at LINE+8   beachte, die Zeile beginnt bei LINE & 8
01BF E5CD5A02       MOD2:   PUSH HL; CALL NEXNUM;
01C3 7EB72808               A_(HL); TSTA; BR Z MOD3;
01C7 237EE17704             INC HL; A_(HL); POP HL; (HL)_A; INC B;
01CC 2318F0                 INC HL; BR MOD2;
01CF E11AFE2EC8     MOD3:   POP HL; A_(DE); CMP #'.; RET Z;
01D4 78B720012318D5         IF B Z; INC HL; FI; BR MOD1;

                        print system prompt and read a line   Schreibe Eingabeanforderungszeichen und lies
                                                              eine Zeile.
01DB EF3E00         INLINE: RST 5; ">";
01DE CD3E00FE1D2809 INL0:   CALL CHIN; CMP #BS; BR Z INL2;
                              return on CR
01E5 FE1F2857               CMP #CR; BR Z CRLF;
                              put out char and continue       Gib Zeichen aus und fahre fort
01E9 CD4A0C18F0     INL1:   CALL $CRT; BR INL0;
                              handle backspace; dont allow backspace over prompt  Behandlung von BS.
01EE ED5B180C1B1AFE3E INL2: DE_CURSOR; DEC DE; A_(DE); CMP #'>;
01F6 28E63E1D18ED           BR Z INL0; A_#BS; BR INL1;        Rücklauf nicht weiter als bis zum Eingabean-
                                                              forderungszeichen.
                        tabulate code. ARG1=start addr, ARG2=end
                            routine is used by Dump command   Tabuliere Speicherinhalt. ARG1=Startadresse
                                                              ARG2=Endadresse. Programm wird auch vom DUMP-
01FC 2A0C0C         TABCDE: HL_ARG1;                          Befehl verwendet.
01FF ED5B0E0CE5B7ED52 TBCD1: DE_ARG2; PUSH HL; TSTA; SBC HL,DE;
0207 E13805EF2E1F00C9       POP HL; IF CC; RST 5; ".",CR; RET; FI;
020F 0E00CD32020608         C_#0; CALL TBCD3; B_#8;
0216 7ECD2B0223     TBCD1A: A_(HL); CALL TBCD2; INC HL;
021B CD3C0210F6             CALL SPACE; DBNZ TBCD1A;
                        put out checksum and backspace over it so it doesnt show  gib Prüfsumme aus und führe
0220 79CD4402               A_C; CALL B2HEX;                  Backspace aus, so daß die Prüfsumme nicht auf
0224 EF1D1D1F00             RST 5; .WORD BS,BS,CR,0;          den Bildschirm kommt.
0229 18D4                   BR TBCD1;
022B 57814F7AC34402 TBCD2:  D_A; ADD C; C_A; A_D; J B2HEX;
0232 7CCD2B02       TBCD3:  A_H; CALL TBCD2;
0236 7DCD2B021800           A_L; CALL TBCD2; BR SPACE;

023C 3E201817       SPACE:  A_#' ; BR JCRT;
0240 3E1F1813       CRLF:   A_#CR; BR JCRT;

                        print A in hex                        Gib A hexadezimal aus.

0244 F51F1F1F1F     B2HEX:  PUSH AF; RRA; RRA; RRA; RRA;
0249 CD4D02F1               CALL B2HEX1; POP AF;
024D E60FC630       B2HEX1: AND #16'F; ADD #'0;
0251 FE3A3802C607           CMP #'9+1; IF CC; ADD #'A-'0-10; FI;
0257 C34A0C         JCRT:   J $CRT;
```

```
                        read in a hex number, DE being used as pointer to line
                           NUM+1, NUM+2 contain the number
                           NUM set non zero if there is a number there at all

025A 1AFE201328FA1B  NEXNUM: A_(DE); CMP #' ' ; INC DE; BR Z .; DEC DE;
0261 AF21120C                CLA; HL_#NUM;
0265 7723772377              (HL)_A; INC HL; (HL)_A; INC HL; (HL)_A;
026A 1A2B2B          NN1:    A_(DE); DEC HL,HL;
026D D630F8                  SUB #'0; RET M;
0270 FE0A3808                CMP #10; BR CS NN2;
0274 D607                    SUB #'A-'0-10;
0276 FE0AF8                  CMP #10; RET M;
0279 FE10F0                  CMP #16; RET P;
027C 133423ED6F      NN2:    INC DE,(HL),HL; RLD;
0281 23ED6F                  INC HL; RLD;
0284 18E4                    BR NN1;

                        main monitor loop; read a line and obey it

0286 CDDB01          PARSE:  CALL INLINE;
0289 114B0B010A0C1A          DE_#L NE+1; BC_#ARGS; A_(DE);
0290 FE20                    CMP #' ' ; [ CHECK FOR STEP REPEAT]
0292 20050AFE5320ED          JR Z; A_(BC); CMP #'S; BR NZ PARSE; FI;
0299 020313AF02              (BC)_A; INC BC,DE; CLA; (BC)_A;
                        get the arguments
029E 03CD5A02        PLOOP:  INC BC; CALL NEXNUM;
02A2 7EB7280D                A_(HL); TSTA; BR Z PEND;
02A6 237E022303              INC HL; A_(HL); (BC)_A; INC HL,BC;
02AB 7E02                    A_(HL); (BC)_A;
02AD 210B0C3418EB            HL_#ARGS+1; INC (HL); BR PLOOP;
02B3 ED4B0A0C2A450C  PEND:   BC_ARGS; HL_$CTAB;
02BA 7EB728C8        PEND1:  A_(HL); BR A Z PARSE; [no such command]
02BE 23                      INC HL;
02BF B9                      CMP C;
02C0 2805                    IF NZ;
02C2 00                       NOP; [patch]
02C3 232318F3                 INC HL,HL; BR PEND1;
02C7                         FI;
02C7 5E2356                  E_(HL); INC HL; D_(HL);
02CA 218602E5EBE9            HL_#PARSE; PUSH HL; EX DE,HL; J (HL);
```

Lies eine Hexadezimalzahl ein. DE wird als Zeiger auf die Zeile NUM & 1 verwendet; NUM & 2 enthält die Zahl. Wenn eine Zahl vorhanden ist, wird NUM ≠ Ø gesetzt.

Hauptteil der Betriebssystemschleife. Lies eine Zeile und führe den Befehl aus.

Lies die Argumente des Befehls

(Kommando ist unbekannt)

```
                        execute command, if arg supplied then this is start address
' 02D0 3EFF321A0C       EXEC:   CONFLG_#-1;
                           common to E and S, conflg tells which
                           set NMI for end of instr
  02D5 21050322480C     EXEC1:  HL_#TRAP; $NMI+1_HL;
  02DB E1                       POP HL; [RUBBISH]
' 02DC 3A0B0CB72806             IF ARGS+1 NZ;EXEC 2;
  02E2 2A0C0C223B0C              HL_ARG1; R.PC_HL;
  02E8                          FI;
  02E8 C1D1F1F1         EXEC 2: POP BC,DE,AF,AF;
  02EC 2A3D0CF9                 HL_R.SP; SP_HL;
  02F0 2A3B0CE52A370C           HL_R.PC; PUSH HL; HL_R.HL;
' 02F7 F53E08D300               PUSH AF; OUT 0,#*3;
  02FC F1ED45                   POP AF; RETN;

                        step, if arg supplied then this is address
  02FF AF321A0C18D0     STEP:   CLA; CONFLG_A; BR EXEC1;

  0305 E323E3           TRAP:   EX (SP),HL; INC HL; EX (SP),HL;
  0308 F5E5                     PUSH AF,HL;
' 030A 3A000CD300               OUT 0,PORT0;
' 030F 3A1A0CB72810             IF CONFLG NZ;
  0315 2A150C7E32170C            HL_BRKADR; A_(HL); BRKVAL_A;
  031C 36E7                      (HL)_#8'347; [RST 4]
  031E E1F1E32BE3                POP HL,AF; EX (SP),HL; DEC HL; EX (SP),HL;
  0323 ED45                      RETN; FI;
  0325 D5               TRAP 1: PUSH DE;
  0326 C521000039       BPT1:   PUSH BC; HL_#0; ADD HL,SP;
  032B 11330C                   DE_#STACK;
  032E 31330C010800EDB0         SP_#STACK; BC_#8; LDIR;
  0336 5E23                     E_(HL); INC HL;
  0338 56231B                   D_(HL); INC HL; DEC DE;
  033B ED533B0C223D0C           R.PC_DE; R.SP_HL;

                        print out regs  SP PC AF HL DE BC
  0342 213F0C0606               HL_#R.SP+2; B_#6;
  0347 2B7ECD4402       REGS1:  DEC HL; A_(HL); CALL B2HEX;
  034C 2B7ECD4402               DEC HL; A_(HL); CALL B2HEX;
  0351 CD3C02                   CALL SPACE;
  0354 10F1                     DBNZ REGS1;
  0356 CD4002                   CALL CRLF;
  0359 2A150C3A170C77   STRT0:  HL_BRKADR; A_BRKVAL; (HL)_A; [RESTORE BREAKPOINT]
  0360 C38602                   J PARSE;
```

Führe Befehl aus. Wenn ein ein Argument angegeben ist, ist es die Startadresse.
CONFLG ist E und S-Kommando gemeinsam. CONFLG gibt eine Aussage darüber ob NMI angefordert wurde und damit der Befehl abgeschlossen wird.

Führe Einzelschritt aus. Wenn ein Argument angegeben ist, dann ist es Adresse.

Gib die Inhalte der Register SP PC AF HL DE und BC aus.

```
                        command table
                            format:  character, address of subroutine

0363 4DAD01         CTAB:   .WORD 'M; .ADDR MODIFY;
0366 43EF03                 .WORD 'C; .ADDR COPY;
0369 45D002                 .WORD 'E; .ADDR EXEC;
036C 53FF02                 .WORD 'S; .ADDR STEP;
036F 54FC01                 .WORD 'T; .ADDR TABCDE;
0372 42E300                 .WORD 'B; .ADDR BREAK;
0375 4C7C03                 .WORD 'L; .ADDR LOAD;
0378 44D103                 .WORD 'D; .ADDR DUMP;
037B 00             NOP;

                        load command

037C CD5100         LOAD:   CALL MOTFLP; [start motor]
037F 218A0B22180C   LOD1:   HL_#CURLIN; CURSOR_HL;
0385 CD3E00FE1D28F9 LOD1B:  CALL CHIN; CMP #BS; BR Z .;
038C FE1F2805               CMP #CR; BR Z LOD1A;
0390 CD4A0C20F0             CALL $CRT; BR NZ LOD1B;
0395 118A0B0608     LOD1A:  DE_#CURLIN; B_#8;
039A 1AFE2ECA5100           A_(DE); CMP #'.; J Z MOTFLP;
03A0 CD5A022A130C           CALL NEXNUM; HL_NUM+1;
03A6 7D844F                 A_L; ADD H; C_A;
03A9 E5210008E5             PUSH HL; HL_#CRTRAM [TEMP BUFFER IN FIRST 8 WORDS]; PUSH HL;
03AE E5CD5A02237E   LOD2:   PUSH HL; CALL NEXNUM; INC HL; A_(HL);
03B4 E17723814F10F3         POP HL; (HL)_A; INC HL; ADD C; C_A; DBNZ LOD2;
03BB CD5A02237EB9           CALL NEXNUM; INC HL; A_(HL); CMP C;
03C1 E1D1                   POP HL,DE;
03C3 2007                   IF Z;
03C5 010800EDB018B3  BC_#8; LDIR; BR LOD1;
03CC                        FI;
03CC CD400218AE             CALL CRLF; BR LOD1;

                          DUMP,  uses same code as TABULATE

03D1 CD5100         DUMP:   CALL MOTFLP;
03D4 0600                   B_#0;
03D6 CD350010FB             CALL KDEL; DBNZ .;
03DB 2A4B0CE5               HL_$CRT+1; PUSH HL;
03DF 215D00224B0C           HL_#SRLOUT; $CRT+1_HL;
03E5 CDFC01                 CALL TABCDE;
03E8 E1224B0C               POP HL; $CRT+1_HL;
03EC C35100                 J MOTFLP;

                           copy,  arguments:from, to, length

03EF 2A0C0CED5B0E0C COPY:   HL_ARG1; DE_ARG2;
03F6 ED4B100CEDB0           BC_ARG3; LDIR;
03FC C9                     RET;

03FD 000000         .BLKZ 16'400-.; [PAD OUT TO END OF ROM]
0400                .END START
```

Kommandotabelle

Format:
Befehlsbuchstabe; Adresse der Behandlungsroutine

Ladekommando

DUMP benutzt denselben Code wie TABULATE

Copieren; Argumente, von / nach Länge eines Blockes

SYMBOL TABLE *Symboltabelle*

```
ARGS   :0C0A
ARG1   :0C0C
ARG2   :0C0E
ARG3   :0C10
BL     =0020
BPT1   :0326
BREAK  :00E3
BRKADR:0C15
BRKVAL:0C17
BS     =001D
B2HEX  :0244
B2HEX1:024D
CHIN   :003E
CONFLG:0C1A
COPY   :03EF
CR     =001F
CRLF   :0240
CRT    :013B
CRTRAM=0800
CRT0   :0167
CRT1   :016A
CRT2   :016F
CRT3   :0195
CTAB   :0363
CUR    =005F
CURLIN=0B8A
CURSOR:0C18
DUMP   :03D1
EXEC   :02D0
EXEC1  :02D5
FF     =001E
FLIP   :0053
FLPFLP:004A
INITE  :013B
INITR  :0C3D
INITT  :0128
INLINE:01DB
INL0   :01DE
INL1   :01E9
INL2   :01EE
JCRT   :0257
KBD    :0069
KDEL   :0035
KMAP   :0C01
KSC1   :007A
KSC1A  :0087
KSC2   :008E
KSC3   :00E0
KSC8   :0089
KSC9   :008A
KTAB   :00EA
LINE   =0B4A
LOAD   :037C
LOD1   :037F
LOD1A  :0395
LOD1B  :0385
LOD2   :03AE
MODIFY:01AD
MOD1   :01B0
MOD2   :01BF
MOD3   :01CF
MOTFLP:0051
NEXNUM:025A
NN1    :026A
NN2    :027C
NUM    :0C12
PARSE  :0286
PEND   :02B3
PEND1  :02BA
PLOOP  :029E
PORT0  :0C00
PRS    :0028
PRS1   :0029
RAME   :0C15
RAMTOP:0C50
RAMZ   :0C00
RAM.   =0C50
REGS1  :0347
ROMTOP:0400
ROM.   =0128
R.AF   :0C39
R.HL   :0C37
R.PC   :0C3B
R.SP   :0C3D
SPACE  :023C
SRLOUT:005D
STACK  :0C33
START  :0000
STEP   :02FF
STRT0  :0359
TABCDE:01FC
TBCD1  :01FF
TBCD1A:0216
TBCD2  :022B
TBCD3  :0232
TRAP   :0305
$CRT   :0C4A
$CTAB  :0C45
$KBD   :0C4D
$KTAB  :0C43
$KTABL:0C3F
$KTABO:0C41
$NMI   :0C47
.      =0400
```

Teil 18 Bedienung und Programmierung des NASCOM 1

Abschnitt A: Bedienung des NASCOM 1

1. Beachten Sie S. 3 des Handbuches "Wichtige Hinweise für alle NASCOM-Benutzer ". ("Important notes for all NASCOM users".)

2. Stellen Sie sicher, daß alle Betriebsspannungen den richtigen Wert haben, bevor Sie die Stromversorgung an den NASCOM 1 anschließen. (Leihen Sie falls erforderlich ein Voltmeter aus.)

3. Beachten Sie, daß beim Einschalten des Netzgerätes keine der Betriebsspannungen fehlen darf. Einige Bausteine (Zeichengenerator und EPROM) können zerstört werden, falls die - 5 V-Versorgungsspannung kurzzeitig fehlt. Falls die Versorgungsspannungen nicht gleichzeitig eingeschaltet werden können, muß die - 5 V-Versorgungsspannung immer als erste eingeschaltet (bevor eine der positiven Versorgungsspannungen angelegt wird) und als letzte ausgeschaltet werden. Dieses Problem kann umgangen werden, wenn die Betriebsspannungen durch ein geeignet ausgelegtes Netzteil in der erforderlichen Reihenfolge oder gleichzeitig angelegt werden.

4. Wenn das System arbeitet und das Fernsehgerät angeschaltet ist, führen sie mit dem NASBUG einige Beispiele zu den Befehlen durch wie in Teil 16 und Abschnitt E dieses Handbuches erläutert. Dann können Sie eigene Programme eingeben oder Programme anderer Anwender eingeben, die für den NASCOM 1 geschrieben wurden.

5. Wenn sie weitere Peripheriegeräte an ihr System anschließen wollen, beachten Sie bitte den folgenden Abschnitt B und alle wichtigen technischen Handbücher.

6. Wenn sie noch keine Erfahrungen mit Computern haben, lesen Sie bitte Abschnitt C oder ein gutes Grundlagenwerk.

7. Wenn Sie später in höheren Programmiersprachen oder in Assembler mit Ihrem NASCOM arbeiten wollen, lesen Sie bitte Abschnitt D und die neusten NASCOM-Produktankündigungen.

8. Bitte, treten sie dem International Nascom Users' Club bei und lassen sie uns alle Probleme, konstruktive Kritik Hardware- und Softwareideen wissen, oder Ideen zu Produkten, die zur Ergänzung des NASCOM 1 nützlich sein könnten.

Abschnitt B: Periphere Hardware

Sowohl die Entwicklung in der Halbleiterindustrie als auch die des Heimcomputersektors verläuft derart rasant, daß diese Hinweise nur einen Eindruck davon geben können, welche Möglichkeiten dem Experimentierer offenstehen. (Lesen sie unsere "seminar notes" ,um einige Möglichkeiten kennen zu lernen).

So leistungsfähig ein Computer auch sein mag, seine Nützlichkeit wird maßgeblich auch dadurch bestimmt, über welche Schnittstellen er mit dem Benutzer und der Umwelt kommuniziert. Ohne Zusatzgeräte verfügt der NASCOM 1 über ein Interface für einen Videomonitor oder ein handelsübliches Fernsehgerät, das als Datensichtgerät 768 (zehn) darstellen kann. Außerdem ist eine Serienschnittstelle für Daten- oder Programmspeicherung auf einem Kassettenrecorder vorgesehen. Außerdem stehen dem Benutzer 16 Ein/Ausgabeleitungen zur Verfügung sowie eine alphanumerische Tastatur.

Obwohl kein spezieller Kassettenrecorder verwendet werden muß, sollen ein paar Hinweise zur Auswahl eines solchen Gerätes gegeben werden. Das einfache NASCOM 1 Kassetteninterface speichert Daten durch ein- und ausschalten eines bestimmten Tones. Daher können Fehler durch "Dropouts" entstehen. Wir empfehlen daher das kräftigste erhältliche Kassettenband (C60) eines renommierten Herstellers zu verwenden. Eine C 60-Kassette kann etwa 85 kByte Daten aufnehmen, oder 20 kByte Daten, die im normalen DUMP-Format ausgegeben werden. Es ist nicht nötig, eine ultra-low noise oder eine CrO2 hi-fi Kassette zu verwenden. Es sollte darauf geachtet werden, daß der Tonkopf und Antrieb sauber und frei von Oxidteilchen sind. Hi-hi Fachgeschäfte bieten Tonkopfreinigungssätze an.

Jedes Bandgerät, ob Kassettengerät oder Spulentonbandgerät kann verwendet werden. Es wird jedoch empfohlen, ein Gerät zu verwenden, das über einen Bandzähler verfügt. Das erleichtert das Auffinden eines Datensatzes sehr. Das einzige Argument für ein kostspieligeres Hi-fi Kassettengerät ist, daß es in der Regel über einen besseren Antrieb und eine ausgefeilte Mechanik verfügt. Ein Stereogerät sollte im Monobetrieb arbeiten, da die Kanäle eine zu geringer Übersprechdämpfung aufweisen, als daß beide Spuren verwendet werden könnten.

Mit der Serienschnittstelle kann man auch (Jumper LK 2,3,4 umsetzen) eine Standard 20 ma-Linienstromschnittstelle betreiben. Man kann damit einen Fernschreiber, ein externes Datensichtgerät eine Tastatur etc. über Anschluß 16 SK2 betreiben. Indem man die Steuereingänge des UART modifiziert, kann man an der Serienschnittstelle auch einen 5-Kanal Baudotfernschreiber betreiben. (Datenblatt des UART liegt dem Bausatz bei). Wenn ein externes Taktsignal für das UART über LK4 eingespeist wird (16-fache Bitrate) dann können Daten mit einer Übertragungsgeschwindigkeit von 200 kBaud verarbeitet werden. Jede dieser Möglichkeiten bedarf kurzer Treiberprogramme, um den Übertragungsablauf zu steuern.

Wird parallel dazu Kassettenbetrieb gewünscht, so kann man einen Vielfachumschalter, ein Relais oder einen COSMOS-Analogschalter (z.B. CD 4066) anstelle der Jumper einsetzen.

Es gibt wenige Alternativen zum TV-Interface. Es erschienen maximal 48 Zeichen pro Zeile auf einem normalen Heimfernsehgerät als gerade noch gut lesbar. Die Erfahrung hat gezeigt, daß es einige Unterschiede zwischen den verschiedenen Empfängern gibt, je nachdem, welche Zwischenfrequenz sie verwenden. Ein Farbfernsehgerät sollte nicht verwendet werden, da das Farbpunktraster oder Farbstreifenraster zusammen mit evtl. Konvergenzfehlern zu einer weniger zufriedenstellenden Darstellung der Zeichen führt. Trotzdem kann der NASCOM 1 mit allen üblichen Farbfernsehern (außerdem dem veralteten 405 Zeilen-System) verwendet werden, wenn der Tuner die UHF-Bänder 4 und 5 empfangen kann. Auch ein VHF-Empfänger kann verwendet werden, wenn die Trägerfrequenz erniedrigt wird, z.B. indem man eine oder zwei Windungen zu der Spule L1 hinzufügt. Obwohl ursprünglich für den Gebrauch mit dem 625 Zeilen/50 Hz-System gebaut, kann der Nascom auch mit einem 525 Zeilen/60 Hz-Gerät betrieben werden, wie diese in Nord- und Südamerika oder Japan üblich sind. Dazu muß die Bildhöhe reduziert werden; die Zeilenfrequenzen sind nahezu identisch.

Wenn ein Empfänger speziell für Verwendung mit dem NASCOM 1 erworben wird, so ist ein kleiner tragbarer Schwarz/Weißfernseher mit Rechteckbildröhre am besten geeignet. Um optimale Bildqualität zu erreichen ist allerdings ein Videomonitor die ideale Lösung. Wir möchten niemandem, der nicht über hinreichende Erfahrung verfügt, raten, er möge einen Videoeingang in sein Fernsehgerät einbauen. Nur wer die Gefahren kennt, die dabei auftreten können, sollte sich daran wagen. Es ist zu hoffen, daß ein Videoeingang bald zu jedem neuen Fernsehempfänger als Standard hinzugehört.

Die 16 Parallel-Ein/Ausgabeleitungen (und die vier zugehörigen Steuerleitungen) können über eine geeignete Pufferschaltung an jedes Gerät angeschlossen werden, mit dem der Benutzer arbeiten möchte. Verwendet man sie als Eingänge, so kann man Daten in digitaler Form einlesen, z.B. die Zeit, die eine Uhr angibt, ASCII-Code, der von einer separaten Tastatur kommt oder analoge Daten, wie z.B. Temperatur, nach A/D-Umwandlung. Schaltet man sie als Ausgänge, so kann man z.B. einen parallel angeschalteten Drucker für Datenausgabe ansteuern, oder D/A-Umsetzer, Thyristor-gesteuerte Leuchten oder einfach eine 7-Segment LED-Anzeige. Die PIO kann in verschiedenen Betriebsarten arbeiten. So können Daten von CPU z.B. durch polling abgefragt werden. Sie können aber interruptgesteuert übertragen werden. Weitere Einzelheiten enthält das PIO-Handbuch. Sie sollten auch wissen, daß zwei Bits des Port Ø vom Benutzer in beiden Richtungen verwendet werden können.

In Zukunft kann der Benutzer durch das standardisierte Bus-Interface sein System durch RAM, I/O- Karten usw. nahezu beliebig erweitern. So können auch parallele und serielle Schnittstellen oder CTC (Counter/Timer-Controller) nachgerüstet werden.

Abschnitt C: Kommunikation mit dem Rechner

Für sich allein genommen bringt der Mikroprozessor oder sein Äquivalent im Großrechner, die Zentraleinheit, keinen Nutzen. Erst wenn eine Anzahl Schalter und Anzeigen an all' die Adreß- Daten- und Steuerleitungen angeschlossen worden sind, kann der interne Schreib/Lesespeicher (die Register) und der Befehlssatz des Rechners getestet werden. (Voraussetzung dazu ist, daß das System mit einer Taktperiode von einigen Sekunden arbeiten kann).

Auch ein reales System, das nur für einen ganz bestimmten Zweck entwickelt wurde (wie z.B. die heute erhältlichen Ein-Chip-Mikrocomputer) muß ROM (Nur-Lese-Speicher) für Programme und Eingabe/Ausgabeleitungen enthalten, um mit der Umwelt in Verbindung treten zu können. Oftmals braucht der Mikrocomputer auch noch ein kleines RAM wenn die internen Register nicht genügend Speicherplatz bieten. Ein solcher Mikrocomputer kann zum Steuern einer Waschmaschine, für Alarmanlagen, elektronische Taschenrechner oder als Peripheriegerät eines größeren Computers verwendet werden. Er ist allerdings absolut unflexibel, d.h. seine internen Programme können immer nur die gleiche Aufgabe erledigen.

Ein Mikrocomputer wie der NASCOM 1 verdankt seine Flexibilität vor allem dem Schreib/Lesespeicher, der jedes beliebige Programm oder Daten speichern kann. Außerdem sind Schnittstellen für den Benutzer, Vorkehrungen zur RAM-Erweiterung oder Ein/Ausgabeerweiterung praktisch ohne Begrenzung, sowie ein Monitorprogramm ("BUG") in einem ROM vorhanden. Im einfachsten Fall besteht der Monitor aus einem "Bootstrap-Loader", der lediglich die Möglichkeit bietet, weitere Daten über eine der Schnittstellen des Rechners in das RAM zu lesen. In unserem Falle steht uns mit dem NASBUG-Monitor (in EPROM gespeichert) ein Betriebssystem zur Verfügung, das die Tastatur, das Serieninterface, das Video/TV-Display, Programmeingabe und MOdifikation und Fehlersuche steuert. Wir haben somit einen Rechner zur Verfügung, der nicht nur eine einzige der oben erwähnten Funktionen ausführen kann, sondern auch als nützliche Hilfe verwendet werden kann, um Systeme für bestimmte Anwendungsfälle zu entwickeln.

Alle diese Einheiten können mit hexadezimalem Code gesteuert und programmiert werden. Er wird auch Maschinencode genannt und besteht aus digitalen Worten von acht Bit (1 Byte) beim Z 80. Andere Computer verwenden auch Wortlängen von 4,8,12,16, 24,32 oder mehr Bit. Für unsere Zwecke stellen wir diese acht Binärziffern als zwei Hexadezimalzahlen dar. Ein Wort von acht Bit wird also in zwei Gruppen von 4 Bit aufgespalten, wobei je vier Bit ein Hexadezimalzeichen zugeordnet wird. Die Hexadezimalzeichen haben die Werte 0..15 und werden wie folgt dargestellt:

BINÄR Basis 2	DEZIMAL Basis 10	HEXADEZIMAL Basis 16
0000	0	0
0001	1 (= 2^0)	1
0010	2 (= 2^1)	2
0011	3	3
0100	4 (= 2^2)	4
0101	5	5
0110	6	6
0111	7	7
1000	8 (= 2^3	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Hier noch einige Beispiele:

		0001 0000 (1 byte)	16		10
		0001 0001	17		11
		0001 1000	24		18
		0001 1111	31		1F
		0010 0000	32	(= 2^5)	20
		0011 0000	48		30
		0100 0000	64	(= 2^6)	40
		0110 0100	100		64
		1000 0000	128	(= 2^7)	80
		1100 1000	200		C8
		1111 1111	255		FF
0000	0001	0000 0000 (2 bytes)	256	(= 2^8)	100
0000	0001	1111 0100	500		1F4
0000	0010	0000 0000	512	(= 2^9)	200
0000	0011	1110 1000	1000		3E8
0000	0011	1111 1111	1023		3FF
0000	0100	0000 0000	1024	(= 2^{10} or 1K)	400
0000	0111	1101 0000	2000		7D0
0000	1000	0000 0000	2048	(=2" or 2K)	800
0001	0000	0000 0000	4096	(=12^{12} or4K)	1000
0010	0000	0000 0000	8192	(=2^{13} or8K)	2000
0100	0000	0000 0000	16384	(=2^{14} or16K)	4000
1000	0000	0000 0000	32768	(=2^{15}or 32K)	8000
1111	1111	1111 1111	65535		FFFF
ditto plus 1:		(3 bytes)	65536	(=2^{16} or64K)	10000

Tritt eine Hexadezimalzahl in einer Beschreibung auf, so ist sie mit dem Suffix 'H' versehen, um Unklarheiten zu vermeiden. So ist z.B. 16 = 10H. Achten Sie außerdem darauf, daß Datensichtgeräte, Drucker, Tastaturen und Anzeigen normalerweise zwischen O (Buchstabe) und Ø (Ziffern "Ø") unterscheiden.

Fernschreiber weisen diese Eigenschaft in der Regel nicht auf, trotzdem treten in der Regel dadurch keine Fehler auf. Beim Erstellen von Programmen bevorzugen es viele Programmierer, die Null mit einem Schrägstrich zu kennzeichnen.

Eine der wichtigsten Eigenschaften des NASBUG ist, daß er die von der Tastatur eingegebenen Kodebuchstaben für hexadezimale Zeichen in Hexadezimalzeichen umwandelt und in einem Byte abspeichert.Wir haben somit ein System vor uns, das mit hexadezimalem Code programmiert werden kann. Auch Daten werden in dieser Art eingegeben. Wir nennen diesen Code "Objektcode" oder auch etwas ungenauer:" Maschinensprache".

- . - - . - - . - - . -

Abschnitt E: Alternative Programmiersprachen

Auf der untersten Ebene kann jeder Computer nur seine eigene Maschinensprache verstehen. Jede CPU hat ihre eigene Sprache. Es ist jedoch weit vorteilhafter, den Rechner in einer Sprache zu programmieren, die der gewöhnlichen Alltagssprache etwas näher kommt, oder aber auch der Ausdrucksweise, der sich die Mathematiker bedienen (mit Zahlen im Dezimalsystem). Es gibt daher eine Anzahl höherer Programmiersprache wobei jede ihre speziellen Eigenschaften und ihre speziellen Vorteile und Nachteile hat. Zu den am weitesten verbreiteten Programmiersprachen gehören BASIC, FORTRAN, COBOL(für kommerziellen Gebrauch), APL (wissenschaftliche Problemstellungen), PL/1 und einige andere. Eine unangenehme Eigenschaft dieser Sprachen ist, daß jeder Benutzer und jeder Computerhersteller einen klein wenig vom Standard (falls es einen gibt) abweichenden Befehlssatz verwendet. So muß ein Programm, das z.B. in BASIC geschrieben ist,für einige Rechner abgeändert werden, um ablauffähig zu sein da es kleine Abweichungen der Sprache, sog. "Dialekte" gibt. Im allgemeinen sind diese Sprachen jedoch deshalb von Vorteil, weil sie die Möglichkeit bieten, Programme zu erstellen, die theoretisch auf jedem Rechner laufen müßten, der über einen Compiler oder Interpreter verfügt, mit dem die Hochsprache in Maschinencode umgesetzt werden kann.

Die NASCOM 1 - Speichererweiterungskarte ist geeignet, einen 2k "TINY BASIC"-Interpreter aufzunehmen, mit dem der Benutzer die Grundzüge von BASIC kennen lernen kann. Es liegen auch Pläne vor, um einen sehr leistungsfähigen 16k BASIC Interpreter zu implementieren, der in einiger Zeit verfügbar sein wird.

Zwischen der Maschinensprache und den höheren Sprachen liegt der ASSEMBLER. Er ist Maschinen-spezifisch, d.h. jede CPU hat einen eigenen Assembler. Für jeden Maschinenbefehl wurde ein dem Befehl entsprechender "MNEMONIC", d.h. eine "Merkhilfe" geschaffen. Für den Z 80 gibt es z.B. folgende mnemonics:

MNEMONIC	BEDEUTUNG
LD	Lade
EX	Tausche Registerinhalte aus
HALT	Halt. Prozessor wartet auf Interrupt oder RESET.

MNEMONIC	BEDEUTUNG
IN	Daten werden von einer Schnittstelle eingelesen.
JP	Sprung zu einem anderen Teil eines Programmes.

Hinter dem Mnemonic steht der OPERAND. Das kann eine Zahl sein oder auch eine Register- oder Speicherzellenbezeichnung. Um alle Mnemonics sowie die zulässigen Operanden und die Adressierungsarten kennen zu lernen, sollten sie die Programmierhandbücher für den Z 80 und die technischen Handbücher dazu lesen.

Wenn unser Rechner nicht die Möglichkeit bietet in einer höheren Programmiersprache zu arbeiten, oder wenn wir, um den vorhandenen Speicherplatz optimal zu nutzen auf die höhere Sprache verzichten, programmieren wir in ASSEMBLER-SPRACHE. Es gibt auch sehr maschinennahe Probleme (z.B. Ein/ Ausgabe über PIO) bei denen man nur in Maschinensprache bzw. Assembler arbeiten kann, weil die Hochsprache nicht den nötigen Befehlsvorrat hat, um das spezielle Problem zu lösen.

Um ein Programm zu erstellen, geht man so vor, daß zunächst ein Programmlisting in Assemblercode schreiben. Anschließend setzt man mithilfe einer Tabelle die man im Programmierhandbuch findet, die Mnemonics und die zugehörigen Operanden in Maschinensprache um. So erhält man eine Liste, die nur noch hexadezimale Zahlen enthält. Für längere Programme kann diese Prozedur sehr mühsam sein. Wir verwenden dann ein spezielles Programm, das ebenfalls als ASSEMBLER bezeichnet wird. Es gibt verschiedene Ausführungen von Assemblern. Der einfache Interpreter setzt genau einen Mnemonic in den zugehörigen Maschinencode um."Multipass compiler", sehr komfortable Assembler also, berücksichtigen auch Programmmarken und Unterprogramme. Für diejenigen Anwender, die einen leistungsfähigen Assembler benötigen, wird ein solches Programm in Kürze zur Verfügung stehen.

Im nun folgenden Abschnitt wird gezeigt, wie man den NASCOM 1 in seiner Grundausführung in Maschinensprache programmieren kann, wobei die Standardschnittstellen berücksichtigt sind.

Abschnitt E: Programmierung des NASCOM 1

1. Vorbemerkungen

Wenn man den NASCOM 1 einschaltet, befindet er sich in einem undefinierten Zustand, ähnlich wie nach einem fehlerhaften Programm, das außer Kontrolle geraten ist. Der Bildschirm zeigt irgendwelche zufälligen Zeichen, eine oder beide Leuchtdioden sind eingeschaltet und der Rechner reagiert nicht auf Tastendruck (außer RESET). Um einen definierten Grundzustand herzustellen, betätigt man die RESET-Taste. Die CPU beginnt das ab der Adresse ØØØØ stehende Monitorprogramm abzuarbeiten. ØØØØ ist die erste Adresse im NASBUG-Monitor, der im EPROM gespeichert ist.

Eingabe	Resultat
RESET	Die Leuchtdioden werden abgeschaltet. Der Bildschirm wird gelöscht (Evtl. kann ein Zufallszeichen stehen bleiben, das vom Breakpointprogramm gegeschrieben wurde) In der linken unteren Ecke des Bildschirmes schreibt der Rechner das "Prompt"-Zeichen (>) gefolgt vom "Cursor" (_).

Das System arbeitet nun in einer Programmschleife, in der ständig die Tastatur und das UART auf Dateneingabe überprüft werden. Das Tastaturprogramm übernimmt auch das Entprellen der Tastatureingabe und das Umsetzen der eingegebenen Zeichen in Hexadezimalcode. Ein eingegebenes Zeichen wird immer in der jeweils nächsten freien Stelle auf dem Bildschirm ausgegeben.

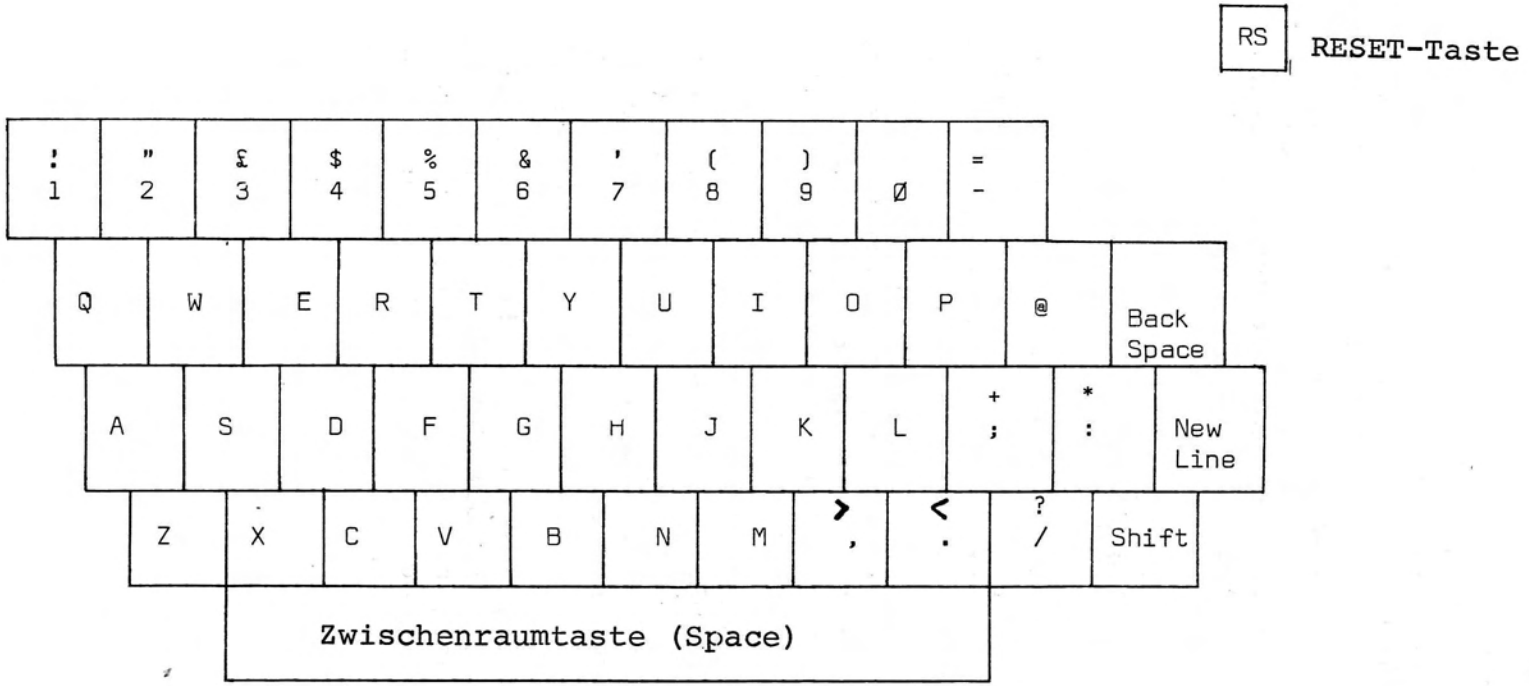
Bevor man irgendetwas eingibt, muß man noch das System initialisieren. Der Breakpoint, der nach dem Einschalten an einer beliebigen Stelle stehen kann, muß so verlegt werden, daß er bei der späteren Programmeingabe nicht stört (d.h. er muß aus dem RAM-Bereich herausgelegt werden.) Man macht dies normalerweise folgendermaßen:

Eingabe	
BØ nl	nl: "newline"
EØ nl	

Nun kann der Benutzer alle verfügbaren Zeichen benutzen und mit den acht Monitorbefehlen arbeiten.

2. Zeichen auf der Tastatur

Mit der Tastatur können die folgenden Zeichen eingegeben werden:



NASCOM 1 - TASTATURBELEGUNG

Es ist auch die Lage der mit "Shift" umgeschalteten Zeichen angegeben.

Wenn die Shift-Taste (Zeichenumschaltung) zusammen mit der Taste eines Buchstabens betätigt wird, für den es kein zugeordnetes Zeichen gibt (z.B. "A"), so wird einfach der Buchstabe dargestellt. Wenn mehrere Tasten gleichzeitig gedrückt werden, treten in der Regel Fehler auf, oder die eingegebenen Zeichen erscheinen nicht in der gewünschten Reihenfolge. Falls man mehr als 48 Zeichen in eine Zeile schreiben will, wird mit dem 49zigsten Zeichen die letzte Eingabezeile um eine Zeile nach oben geschoben. Der Cursor steht wieder in der untersten Zeile, der 49zigste Buchstabe steht in der untersten Zeile links.

Eines oder mehrere Zeichen kann mit BACKSPACE gelöscht werden. Der Cursor läuft um eine Stelle nach links und löscht das dort befindliche Zeichen. Mit Backspace kann der Cursor allerdings nur so weit nach links laufen, bis er auf das PROMPT-Zeichen (>) trifft. Wenn der Cursor nicht in der untersten Zeile steht, sondern z.B. in der dritten Zeile, so kann man auch mehr als eine Zeile mit Backspace löschen, falls zwischendrin kein Prompt-Zeichen auftritt. Der Cursor läuft dann vom linken letzten Buchstaben der letzten Zeile um rechten äußeren Zeichen der darunter liegenden Zeile. Weiter als bis zum linken Ende der untersten Zeile kann der Cursor grundsätzlich nicht laufen.

Der Bildschirm kann (einschließlich der stehenden obersten Zeile) gelöscht werden, wenn man gleichzeitig die Tasten SHIFT und BACKSPACE betätigt. Der Cursor springt in die linke untere Ecke des Bildschirmes, es wird jedoch kein PROMPT ausgegeben. Erst wenn NEWLINE betätigt wurde, erscheint das PROMPT-Zeichen (Eingabeanforderungszeichen).

Durch Betätigen der NEWLINE-Taste werden die unteren fünfzehn Zeilen des Bildschirmes um eine Zeile nach oben geschoben. Der Cursor springt in die linke untere Ecke der letzten Zeile. Die unterste Zeile wird gelöscht. NEWLINE ist eine der wichtigsten Tasten am NASCOM, denn sie schließt jeden Befehl ab. Dabei muß der Befehl unmittelbar hinter dem PROMPT-Zeichen in der linken unteren Ecke des Bildschirmes stehen.

Falls aus irgendeinem Grund (z.B. weil man den Bildschirm als Datensichtgerät benutzen will), die Monitorbefehle nicht ausgeführt werden sollen, muß mindestens ein Zwischenraumzeichen zwischen dem Prompt-Zeichen und dem ersten Buchstaben stehen. (Es kann auch ein kleines Programm zu diesem Zweck geschrieben werden.)

3. Monitorbefehle

M	Gib Speicherzelleninhalt aus; Änderung von Inhalten (Wird zur Programmeingabe mit verwendet.)
T	Gib Inhalt des Speichers in Tabellenform aus. (Speicherblock, der durch zwei Adressen festgelegt ist)
D	Gib Daten aus dem Speicher an das serielle Interface
L	Lade Daten/Programme von der seriellen Schnittstelle (Kann mit "NL", ".", "NL","NL" abgeschlossen werden, wenn die Endmarke auf dem Band fehlt.

C	Kopiere einen Speicherblock der Länge dddd von einer Anfangsadresse bbbb beginnend in einen Block beginnend bei aaaa.
B	Setze Breakpoint im Benutzerprogramm.
E	Führe Benutzerprogramm aus.
S	Einzelschrittbetrieb. Gehe schrittweise durch ein Programm. Wenn man einen anderen Befehl eingibt oder einen "." Punkt eingibt, wird der Einzelschrittbetrieb abgebrochen.

Wenn das Programm auf einen Breakpoint trifft, oder wenn es im Einzelschrittbetrieb arbeitet, werden sechs Registerpaare in folgendem Format ausgegeben:

SP PC AF HL DE BC .

Um die Registerinhalte zu einem beliebigen Zeitpunkt anzuzeigen, gibt man den Befehl S3FF nl. (Es handelt sich um einen Einzelschrittbefehl, mit dem ein "NOP"-Befehl abgearbeitet wird.)

Der S-Befehl erlaubt es Programme, die im RAM oder im ROM (EPROM) gespeichert sind, schrittweise abzuarbeiten. Man kann so auch die Programme des Betriebssystemes schrittweise durchlaufen, allerdings nicht die Tastaturabfrage (dabei würde der Rechner in einen undefinierten Zustand geraten).

Führende Nullen bei Adressen und Daten brauchen nicht angegeben zu werden. Für den Rechner sind also die folgenden beiden Ausdrücke bedeutungsgleich:

TØ 5F = TØØØØ ØØ5F .

Wenn man im Einzelschrittbetrieb ein Programm abarbeiten läßt, aber nicht am Programmanfang, sondern an einer anderen Stelle beginnt, muß man sorgfältig darauf achten, daß der Stackpointer nicht falsch behandelt wird. So sollte man nie einen "RET"-Befehl oder einen "POP"-Befehl abarbeiten, ohne daß vorher der zugehörige "CALL"-Befehl oder "PUSH"-Befehl durchlaufen wurde.

Falls sie Zweifel haben, wo der Breakpoint steht, so können sie die Adresse ausgeben lassen. Dazu müssen die Speicherzellen C15 und C16 ausgegeben werden. Das höherwertige Adreßbyte steht in C16, das niederwertige in C15. Normalerweise enthalten diese Speicherzellen den Wert ØØ, der ihnen bei der Systeminitialisierung zugewiesen wurde.

Falls irgendwelche Schwierigkeiten auftreten, betätigen sie die RESET-Taste. Damit sind sie wieder im Monitorprogramm.

Falls eines ihrer Programme nicht wie gewünscht arbeitet, können sie das Programm entweder im Einzelschrittbetrieb abarbeiten, bis sie die Stelle gefunden haben, an der ein Registerinhalt nicht den gewünschten Wert hat, oder sie können Breakpoints in das Programm setzen, bis sie den Punkt gefunden haben, der die Schwierigkeiten bereitet.

4. Beispiel für eine Programmentwicklung

Ziel — Alle verfügbaren Zeichen (wie auf Seite 33 in den Hardware-Notes gezeigt) sollen in Tabellenform auf dem Bildschirm ausgegeben werden. Es sind dies alle Zeichen, die der Zeichengenerator erzeugen kann.

Methode — Wir steigen sofort in das eigentliche Problem ein. Wem Flußdiagramme zum Verständnis helfen, der möge sich zur Veranschaulichung solche Diagramme erstellen. Falls weitere Information benötigt wird, empfehlen wir, Grundlagenliteratur zu lesen bzw. Fachzeitschriften heranzuziehen.

Verfahrensschritt	Kommentar
(a)	Einige Register werden so vorbesetzt, daß sie später als Zeiger auf Speicherzellen oder den Bildschirm verwendet werden können. Es werden auch die Schleifenzähler vorbesetzt. Bevor die nächsten beiden Schritte durchgeführt sind wissen wir allerdings nicht, mit welchen Werten die Register vorbesetzt werden müssen.
(b)	Das Programm wird hauptsächlich eine Schleife enthalten in der ein Zeichen auf den Bildschirm geschrieben und ein Zeiger auf den nächsten zu beschreibenden Speicherplatz auf dem Bildschirm vorgerückt wird.
(c)	Dann muß eine Variable überprüft werden, um festzustellen, ob bereits eine Zeile zu Ende geschrieben wurde. Wenn dies der Fall ist, darf nicht einfach das nächste Zeichen ausgegeben werden, sondern der Zeiger muß auf die nächste Zeile und dort auf die erste zu beschreibende Speicherzelle gesetzt werden. Erst dann kann das nächste Zeichen ausgegeben werden. Wenn wir die letzte Zeile zu Ende geschrieben haben, muß das Programm in irgendeiner Weise abgeschlossen werden.
(d)	Denken wir noch einmal über Schritt (b) nach. Offensichtlich benötigen wir einen Schreibbefehl,um ein Zeichen auf den Bildschirm zu bringen, falls wir dazu nicht ein im Betriebssystem vorhandenes Unterprogramm verwenden. Wir benutzen den Befehl LD (HL),A. Wir benutzen also das HL-Registerpaar, das auf eine Speicherzelle des Bildschirmes zeigt und laden mit diesem Schreibbefehl den Inhalt des Akkumulators in die Speicherzelle, auf die HL zeigt. A muß den Zeichencode enthalten. Um die Tabelle übersichtlich zu gestalten, füllen wir die obere Hälfe des Bildschirmes mit dem Zeichensatz. Zwischen zwei beschriebenen Zeile ist immer ein Abstand von drei Zeilen.

Damit sieht der Kern unseres Programmes etwa so aus:

```
LD (HL),A     Schreibe Zeichen auf den Bildschirm

INC A         Code für nächstes Zeichen

INC HL  ┐
INC HL  ├     Drei Leerzeilen; setze HL auf nächsten
INC HL  ┘     Bildschirmspeicherplatz
```

(e) Betrachten wir nochmal Schritt (c). DJNZ ist der Befehl "Dekrementiere und springe wenn Inhalt von Register B nicht = Ø". Es wird der unter (d) diskutierten Programmschleife also ein Befehl vorangehen, mit dem die Anzahl der Zeichen pro Zeile festgelegt wird:

```
LINE:   LD B, 1ØH     (1OH = 16dezimal)
```

Nachdem das Programm die Schleife durchlaufen hat, trifft es auf den Befehl:

```
DJNZ   -Ø5H    Wenn der Inhalt von B nicht
               Null ist,springe um fünf
               Speicherzellen zurück (zur
               Marke "WRITE") und führe von
               dort aus das Programm weiter aus.
```

Ein übliches Verfahren, um den Zeiger auf den Anfang der nächsten Zeile zu setzen ist, das DE-Registerpaar mit 1OH zu laden ($16_{dezimal}$) und anschließend HL und DE zu addieren:

```
ADD HL,DE      .
```

Nun müssen wir noch feststellen, wann die Tabelle abgearbeitet ist. Man kann dazu entweder den Zeichencode im A-Register überprüfen oder die Stellung des HL-Zeigers. Wird benutzen hier die erste Methode:

```
CP  8OH        Überprüfe, ob das 129zigste
               Zeichen im Akkumulator steht.
               ( Das erste Zeichen war ØØ. )
```

Wenn der Akkumulator nicht 8OH enthält, springt das Programm zur Marke "LINE" zurück:

```
JRNZ "LINE"    Springe zur Marke "LINE" zurück,
               wenn das Z-Flag nicht gesetzt
               ist. (Wieviele Byte bei JRNZ
               rückwärts gesprungen wird, werden
               wir beim eigentlichen Erstellen
               des Maschinencode berechnen.)
```

Man könnte das Programm durch die verschiedensten Befehle abschließen. Einer davon wäre:

```
HALT
```

wobei die "HALT"-Leuchtdiode aufleuchten würde.

Wenn die CPU den HALT-Befehl interpretiert hat, führt sie laufend "NOP"-Zyklen aus. Diesen Zustand kann sie nur mit einem RESET oder einem Interrupt verlassen (falls ein Interrupt vorbereitet wurde.) Man könnte auch das Tastaturunterprogramm KBD aufrufen, und abfragen, ob eine bestimmte Taste betätigt wurde, um ggfls. zu einem anderen Programmteil zu verzweigen. Ein sehr nützlicher Abschluß ist der Rücksprung in die Abfrageschleife des Monitors ("PARSE") mit dem Befehl:

JP Ø286H

Nachdem das Programm diesen Punkt erreicht hat kann man die acht Monitorbefehle genauso benutzen, wie dies vor Ablauf des Programmes möglich war.

(f) Nun können wir das gesamte Programm einschließlich Maschinencode und Initialisierung angeben. (Die Label und Kommentare sollen nur die Übersichtlichkeit verbessern). Das Programm ist voll "relocatable" (verschieblich), d.h. es kann an jeder beliebigen Stelle des RAM beginnen, da es keine absoluten Sprüngen zwischen Teilen des Programmes enthält.

Programm, das den gesamten Zeichensatz auf dem Bildschirm darstellt

ADRESSE	OBJECT CODE	LABEL	QUELLCODE	KOMMENTAR
ØCEØ	11 1Ø ØØ	START	LD DE,ØØ1ØH	Abstand zweier Zeilen
ØCE3	21 ØB Ø8		LD HL,Ø8ØBH	Zeiger auf Bildschirm
ØCE6	3E ØØ		LD A,ØØH	Erster Zeichencode
ØCE8	Ø6 1Ø	LINE	LD B,1ØH	16 Zeichen pro Zeile
ØCEA	77	WRITE	LD (HL),A	Zeichen auf Bildschirm
ØCEB	3C		INC A	Code für nächstes Zeichn
ØCEC	23		INC HL	Zeiger drei Stellen
ØCED	23		INC HL	nach rechts rücken
ØCEE	23		INC HL	
ØCEF	1Ø F9		DJNZ -Ø5H	Wenn nicht Zeilenende, gehe nach "WRITE"
ØCF1	19		ADD HL,DE	Anfang der nächsten Zeile
ØCF2	FE 8Ø		CP 8ØH	Ende der Tabelle ?
ØCF4	2Ø F2		JRNZ -ØCH	Wenn nicht, dann "LINE"
ØCF6	C3 86 Ø2		JP Ø286H	Sprung zu PARSE in Monitor

(g) Der Maschinencode wird mit dem M-Befehl in den Rechner eingegeben, so wie das in Teil 16 dieses Handbuches beschrieben wurde.

(h) Um Fehler aufzufinden, kann man sich den eingegebenen Code mit dem Befehl

TCEØ CFF $_{nl}$ auf dem Bildschirm

ausgeben lassen.

(i) Dann kann man mit dem S-Befehl das Programm vom Beginn an schrittweise abarbeiten. Dabei kann überprüft werden, ob die Register in der vorgesehenen Weise verändert werden.

(j) Mit dem B-Kommando kann man einen Breakpoint setzen, z.B.:

```
BCEB nl.
```

(k) Das Programm läuft bis zum Breakpoint. Achtung: Auf einen B-Befehl muß immer ein E-Befehl folgen! Der Anfangspunkt des Programmes ist ØCEØ. Wir schreiben daher den Befehl:

```
ECEØ nl
```

direkt nachdem wir den Breakpoint gesetzt haben.

(l) Man kann nun mit dem S-Kommando überprüfen, ob die Sprungbefehle alle richtig errechnet wurden. Man kann dies auch einfach überprüfen, indem man eingibt

```
E nl.
```

Die Schleife wird einmal durchlaufen, dann trifft das Programm auf den Breakpoint. Wie schon oben erklärt,wird beim Breakpoint und beim S-Befehl ein Registerdump durchgeführt, d.h. die sechs wichtigsten Registerinhalte werden angezeigt.

(m) Man kann den Breakpoint entfernen oder an eine andere Stelle setzen (z.B. nach ØCF1; Ende einer Zeichenzeile).

(n) Man kann den Breakpoint entfernen:

```
BØ nl
```

(o) Anschließend wird das Programm normal ausgeführt:

```
ECEØ nl
```

(p) Es könnte zweckmäßig sein, vor Beginn des Programmes den Bildschirm zu löschen (siehe Abschnitt F).

(q) Durch das vom Monitor durchgeführte "scrolling" wird die durch unser Programm erzeugte Zeichentabelle wieder zerstört.

(r) Der Benutzer, der nun gerne selbst programmieren lernen möchte, könnte auf den Gedanken kommen, das vorliegende Programm noch weiter zu verbessern. Eine Möglichkeit wäre, die Tabelle in anderem Format auszugeben, oder an anderer Stelle des Bildschirmes zu plazieren.

5. Ein weiteres Programmbeispiel

Zweck Es soll wie im letzten Beispiel der volle Zeichensatz angezeigt werden, allerdings mit einer programmierbaren Zeitverzögerung zwischen den einzelnen Schreiboperationen.

Aufrufe

ED00 nl ; Ausgabe jede Minute (Beenden mit RESET)

ED08 nl ; Einmalige Ausführung. Anschließend Rückkehr in den Monitor.

Langsame Ausgabe des gesamten Zeichensatzes

Adresse	Object Code	Label	Quellcode	Kommentar
0D00	CD 10 0D	START A	CALL 0D10H	Rufe Unterprogramm auf
0D03	CD 12 0D		CALL 0D12H	Rufe Unterprogr. 2 (Löschen)
0D06	18 F8		JR -6H	Zurück zu START
0D08	CD 10 0D	START B	CALL 0D10H	Rufe Unterprogramm auf
0D0B	C3 86 02		JP 0286H	Rücksprung zum Monitor
0D0E	00 00		NOP, NOP	
0D10	0E 80	SUB ROUTINE	LD C,80H	Erstes Zeichen (plus 80H)
0D12	11 10 00	SUB.2	LD DE,0010H	Abstand der Zeilen
0D15	21 0B 08		LD HL, 080BH	Zeiger HL auf Bildschirm
0D18	06 10	LINE	LD B, 10H	Anzahl Zeichen pro Zeile
0D1A	71	WRITE	LD (HL),C	Schreibe Zeichen auf Bildschirm
0D1B	79		LD A,C	Zwischenraumzeichen ?
0D1C	FE 20		CP 20H	
0D1E	28 01		JR Z,+ 3H	Nächstes Zeichen erzeugen, falls
0D20	0C		INC C	Schirm nicht gelöscht wurde
0D21	23		INC HL	Bildschirmzeiger drei Stellen
0D22	23		INC HL	nach rechts.
0D23	23		INC HL	
0D24	79		LD A,C	Variable Verzögerung, abhängig
0D25	08		EX AF,AF'	vom Inhalt des Registers C
0D26	AF		XOR A	
0D27	3D		DEC A	
0D28	20FD		JRNZ-1H	
0D2A	08		EX AF,AF'	
0D2B	3D		DEC A	
0D2C	20F7		JRNZ-7H	
0D2E	10EA		DJNZ-14H	Wenn Zeilenende nicht erreicht, gehe zur Marke "WRITE"
0D30	19		ADD HL,DE	HL auf nächste Zeile
0D31	7C		LD A,H.	Tabellenende ?
0D32	FE 0A		CP 0AH.	
0D34	20 E2		JRNZ - 1CH	Gehe zu LINE wenn kein Tab.ende
0D36	0E 20		LD C, 20H	Zwischenraumzeichen in C setzen
0D38	C9		RET	Rückkehr zum rufenden Programm.

Wir überlassen es dem Leser, dieses Programm in all' seinen Einzelheiten zu analysieren. Es wurden in diesen Beispiel z.B. andere Register verwenden, um den Akkumulator für andere Zwecke benutzen zu können. Die hier angewandten Methoden der Programmierung brauchen nicht optimal zu sein. Es wurde von der pragmatischen Aufgabenstellung ausgegangen: "Ein Programm schreiben, das funktioniert und keine unerwünschten Nebeneffekte hat"(z.B. reservierte Ram-Bereiche löschen)

Abschnitt F: PROGRAMMIERTIPS

Schon bald werden sie mit Ihrem NASCOM 1 auch sehr komplizierte Problem lösen können, je nachdem wieweit es ihre Zeit, die vorliegenden Erfahrungen, Peripherie und RAM-Kapazität ermöglichen. So können sie Computerspiele, ein persönliches Kontosaldo, ein Buchhaltungssystem oder eine Bildschirmzeitung auf ihrem System implementieren. Bei diesen etwas aufwendigeren Anwendungen wird man natürlich einen Assembler oder eine höhere Programmiersprache einsetzen wollen. Bis diese Programme für Ihren NASCOM 1 verfügbar sind, sollen ein paar Hinweise ihnen helfen, Ihren NASCOM 1 optimal auszunutzen.

Einige NASBUG-Unterprogramme (siehe auch S. 47)

1. CD 69ØØ = CALL ØØ69H (CALL KBD). Dieses Programm fragt die Tastatur ab. Wenn irgendeine Taste betätigt wurde setzt KBD das Carryflag und liefert den Hexadezimalcode, der dem Zeichen entspricht im Akkumulator.
2. CD 3BØ1 = CALL Ø13B (CALL CRT). CRT bringt den Code, der in A steht als ASCII-Zeichen zur Anzeige und zwar an der Stelle, an der der Cursor gerade steht. CRT rückt den Cursor um eine Stelle nach rechts. Wenn im Akkumulator der Code 1E stand, wird der Bildschirm gelöscht.
3. CD 3CØ2 = CALL Ø23CH (CALL SPACE). Rückt den Cursor um eine Stelle nach rechts.
4. CD 4ØØ2 = CALL Ø24ØH (CALL CRLF) Scroll um eine Zeile. (D.h.: alle Zeilen des Bildschirms mit Ausnahme der O.ten Zeile werden um eine Zeile nach oben geschoben. Der Inhalt der obersten Zeile geht verloren).
5. CD 44Ø2H = CALL Ø244 (CALL B2 HEX) Interpretiert das im Akkumulator stehende Byte als zwei Hexadezimalzahlen und gibt sie auf dem Bildschirm aus.
6. CD 35ØØ = CALL ØØ35H (CALL KDEL) Bewirkt eine Zeitverzögerung von 7 1/2 ms und löscht den Akkumulator.
7. C3 86Ø2 = JP Ø286H (JP PARSE). Rücksprung in den Monitor. UART und Tastatur werden abgefragt. Befehle/Daten werden interpretiert.
8. Wenn man die Bildschirmausgaberoutine selbst schreiben will, muß man die Anfangsadresse des eigenen Programmes in die Speicherzellen ØC4B und ØC4c eintragen.
9. Will man die Tastaturabfrage selbst schreiben, so muß man die Adresse in ØC4E, ØC4F ändern.
10. Wenn man die Befehlstabelle ändern will, muß man die Anfangsadresse der neuen Tabelle in die Adressen ØC45 und ØC46 eintragen.
11. Falls man das gleiche mit der Tastaturtabelle machen möchte sind die Adressen ØC43, ØC44 zu ändern.
12. Die Länge der neuen Tastaturtabelle muß in ØC3F, ØC4Ø angegeben werden.

13. Falls der Anfang der neuen Tastaturtabelle anders liegt, muß die Anfangsadresse in ØC41 ØC42 angegeben werden.

14. Die Kommandotabelle kann abgeschaltet werden, indem man eingibt:

    MC45 nl und dann FF. nl

15. Die nicht belegten Ausgabeleitungen des Port Ø kann man setzen und rücksetzen, indem man die zugehörigen Bits in der Speicherzelle ØCØØH setzt oder rücksetzt während das Monitorprogramm läuft.

16. Die erste Adresse, die vom Benutzerprogramm verwendet werden kann, ist ØC5ØH, nicht wie an anderer Stelle möglicherweise angegeben die Adresse ØC6Ø.

Hinweise zur Programmierung der Z 8Ø - CPU

17. Es gibt zwei Möglichkeiten, den Akkumulator zu löschen:

    3E ØØ LD A, ØØ oder AF XOR A
    Vorteil von XOR A: Der Befehl braucht nur ein Byte.
    Vorteil von LD A,ØØ: Die Flags werden nicht beeinflußt.

18. Wenn man, nachdem man ein Byte in den Akkumulator geladen hat, die Flags setzen möchte, z.B. um zu überprüfen, ob man eine ØØ in den Akkumulator geladen hat, so kann man dies mit dem Befehl tun:

    B7 OR A ohne den Inhalt von A zu verändern.

19. Um eine Zufallszahl aus dem Bereich ØØ bis 7FH zu erzeugen, kann man den Befehl

    ED 5F LD A,R verwenden. Wenn er das erste Mal verwendet wird (in einem Programm, das noch nicht gelaufen ist), dann gewinnt man eine echte Zufallszahl. Danach kann man allerdings den Wert von R vorhersagen, der von der Anzahl M1-Zyklen zwischen zwei Aufrufen abhängt.

20. Um das Carryflag in alle Bits von A zu kopieren, kann man den Befehl benutzen:

    SBC A,A (9F)

21. Um das Carryflag in alle Bits von HL zu kopieren, benutzt man die Anweisung:

    ED 62 SBC HL,HL

22. Um die PIO Ports als Ausgänge des NASCOM 1 zu initialisieren, schreibt man die Befehle:

    3E ØF LD A, ØFH (ØF ≙ "Mode Ø"; Ausgabe)
    D3 Ø6 OUT (Ø6),A (Für Port 4) A
    D3 Ø7 OUT (Ø7),A (Für Port 5) B

23. Um die Leitungen der PIO als Eingänge zu programmieren, benutzt man die folgende Befehlsfolge:

```
3E  4F   LD A,4F     (4F ≙ "Mode 1"; Eingabe)
D3  Ø6   OUT (Ø6),A  (Für Port 4)
D3  Ø7   OUT (Ø7),A  (Für Port 5)
```

Weitere Details entnehmen sie bitte dem PIO-Handbuch.

24. Um ihnen das errechnen der Sprungweiten bei relativen Sprüngen etwas zu erleichtern, geben wir hier eine Tabelle an:

Sprungweite dezimal	hexadezimal	Code im Sprungbefehl
-126 (max.)	-7EH	80
- 62	-3EH	C0
- 32	-20H	DE
- 30	-1EH	E0
- 16	-10H	EE
- 14	-0EH	F0
- 12	-0CH	F2
- 10	-0AH	F4
- 9	-09H	F5
- 8	-08H	F6
- 7	-07H	F7
- 6	-06H	F8
- 5	-05H	F9
- 4	-04H	FA
- 3	-03H	FB
- 2	-02H	FC
- 1	-01H	FD
+ 3	+03H	01
+ 4	+04H	02
+ 5	+05H	03
+ 6	+06H	04
+ 7	+07H	05
+ 8	+08H	06
+ 16	+10H	0E
+ 18	+12H	10
+ 32	+20H	1E
+ 64	+40H	3E
+128	+80H	7E
+129 (max).	+81H	7F

Abschnitt G Umsetzen von Z 80 in 8080 Programme und umgekehrt

Der Z 80 Maschinencode ist eine erweiterte Version des 8080-Maschinencode. Er enthält 2 Byte-Opcodes und 4 Byte Befehle. Daher läuft ein 8080-Programm in der Regel auf dem Z 80, nicht aber umgekehrt. Auf den folgenden beiden Seiten geben wir eine Tabelle an, die den Z 80 und den 8080-Maschinencode vergleicht.

Umsetzen von Z 80 in 8080-Maschinencode

8080		Z80	
ACI	n	ADC	A,n.
ADC	r	ADC	A,r.
ADC	M	ADC	A,(HL).
ADD	r	ADD	A,r
ADD	M	ADD	A,(HL).
ADI	n	ADD	A,n.
ANA	r	AND	r
ANA	M	AND	(HL)
ANI	n	AND	n
CALL	nn	CALL	nn
CC	nn	CALL	C,nn.
CM	nn	CALL	M,nn.
CMA		CPL	
CMC		CCF	
CMP	r	CP	r
CMP	M	CP	(HL)
CNC	nn	CALL	NC, nn
CNZ	nn	CALL	NZ,nn
CP	nn	CALL	P,nn.
CPI	n	CP	n
CPE	nn	CALL	PE,nn
CPO	nn	CALL	PO,nn
CZ	nn	CALL	Z,nn
DAA		DAA	
DAD	B	ADD	HL,BC.
DAD	D	ADD	HL,DE.
DAD	H	ADD	HL,HL.
DAD	SP	ADD	HL,SP

8080		Z80	
DCR	r	DEC	r
DCR	M	DEC	(HL).
DCX	B	DEC	BC
DCX	D	DEC	DE
DCX	H	DEC	HL
DCX	SP	DEC	SP
DI		DI	
EI		EI	
HLT		HALT	
IN	n	IN	A, (n)
INR	r	INC	r
INR	M	INC	(HL)
INX	B	INC	BC
INX	D	INC	DE
INX	H	INC	HL
INX	SP	INC	SP
JC	nn	JP	C,nn
JM	nn	JP	M,nn
JMP	nn	JP	nn
JNC	nn	JP	NC,nn
JNZ	nn	JP	NZ,nn
JP	nn	JP	P,nn
JPE	nn	JP	PE,nn
JPO	nn	JP	PO,nn
JZ	nn	JP	Z,nn.
LDA	nn	LD	A,(nn).
LDAX	B	LD	A,(BC).
LDAX	D	LD	A,(DE).
LHLD	nn	LD	HL,(nn).
LXI	B,nn	LD	BC,nn.

8080		Z80	
LXI	D,nn	LD	DE,nn.
LXI	H,nn	LD	HL,nn.
LXI	SP,nn	LD	SP,nn.
MOV	r,r'	LD	r,r'
MOV	M,r	LD	(HL), r.
MOV	r,M	LD	r, (HL)
MVI	r,n	LD	r,n
MVI	M,n	LD	(HL),n
NOP		NOP	
ORA	r	OR	r
ORA	M	OR	(HL)
ORI	n	OR	n
OUT	n	OUT	(n),A
PCHL		JP	(HL)
POP	B	POP	BC
POP	D	POP	DE
POP	H	POP	HL
POP	PSW	POP	AF
PUSH	B	PUSH	BC
PUSH	D	PUSH	DE
PUSH	H	PUSH	HL
PUSH	PSW	PUSH	AF
RAL		RLA	
RAR		RRA	
RC		RET	C
RET		RET	
RIM	(8085)	-	
RLC		RLCA	
RM		RET	M
RNC		RET	NC
RNZ		RET	NZ
RP		RET	P
RPE		RET	PE
RPO		RET	PO
RRC		RRCA	
RST	0	RST	00H
RST	1	RST	08H
RST	2	RST	10H
RST	3	RST	18H
RST	4	RST	20H
RST	5	RST	28H
RST	6	RST	30H
RST	7	RST	38H
RZ		RET	Z
SBB	r	SBC	A,r
SBB	M	SBC	A,(HL)
SBI	n	SBC	A,n
SHLD	nn	LD	(nn),HL.
SIM	(8085)	-	
SPHL		LD	SP,HL.
STA	nn	LD	(nn),A.
STAX	B	LD	(BC), A.
STAX	D	LD	(DE),A.
STC		SCF	
SUB	r	SUB	r
SUB	M	SUB	(HL)
SUI	n	SUB	n
XCHG		EX	DE,HL
XRA	r	XOR	r
XRA	M	XOR	(HL)
XRI	n	XOR	n
XTHL		EX	(SP),HL.

Bezeichnungsweise

r oder r' = Register A,B,C,D,E,H oder L
n = 8 Bit-Zahl oder Portadresse
nn = 16 Bit-Zahl oder Speicheradresse

Achten sie bitte darauf, daß bei folgenden Mnemonics keine Verwechslungen auftreten:

8080		Z80
CP	=	CALL P
CMP,CPI	=	CP
-------		CPI
JP	=	JP P
JMP	=	JP

Um 8080-Kompatibilität zu erreichen, haben einige Z 80 - Befehle den gleichen Maschinencode wie die zugehörigen 8080-Befehle. Z.B.:

ØF	RRCA
CB ØF	RRC A

In der Regel kann man Z 80 - Programme nur mit erhöhtem Aufwand in 8080-Programme umsetzen. Die Programme werden dabei länger. Alle Befehle, über die der Z 80 zusätzlich verfügt, beginnen mit folgenden Codes:

Ø8, 1Ø, 18, 2Ø, 28, 3Ø, 38, CB, D9, DD, ED, oder FD.

(Beim 8085 werden 2Ø und 3Ø für spezielle Zwecke benötigt und sind nicht Z 80-kompatibel.)

Abschnitt H: Schlußbemerkung

Bitte, bleiben sie über den NASCOM USERS CLUB mit uns in Verbindung und lassen sie uns ihre Probleme und Erfolge wissen. Wir würden uns auch freuen, wenn sie uns mitteilen würden,ob und welche Fehler unsere Beschreibungen enthalten bzw. was man den Beschreibungen noch hinzufügen könnte.

Wir freuen uns, das Zeitalter der Mikrocomputerrevolution mit Ihnen zusammen beginnen zu können.