Handbuch Teil 2

R O B O T R O N

Mikrorechnerbausatz Z 1 0 1 3

Handbuch Teil II

Inhaltsverzeichnis

Bestandteile des Handbuches:

Handbuch Teil I Handbuch Teil II Anlagenteil

5. Software des MRB Z1013

5.1 Monitor

5.1.1. Leistungen des Monitors

Nach dem Einschalten benötigt der Mikroprozessor eine definierte Befehlsfolge für seine

Arbeitsfähigkeit. Ohne ein Betriebsprogramm, dem sogenannten Monitorprogramm, das in einem nichtflüchtigen Speicher stehen muss, sind keinerlei Aktionen der CPU wie Tastaturabfrage oder Ausgaben möglich. Der Monitor des MRB Z1013 umfasst eine Größe von 2K Byte und belegt den Adressbereich von F000H bis F700H. Mit diesem Monitor kann der Benutzer Speicherbereiche ansehen, eigene Programme von Hand oder von Magnetband eingeben, diese eingegebenen Programme austesten sowie auf Magnetband abspeichern. Weiterhin ist der Start eigener, ausgetesteter Programme möglich.

Auch zum Austesten eigener Hardwareerweiterungen kann der Monitor verwendet werden. Um diese Möglichkeiten umfassend nutzen zu können, wurden bestimmte Kommandos festgelegt, die bereits im Abschnitt 1.3. vorgestellt wurden.

Außer dem unmittelbaren Aufruf bestimmter Monitorleistungen durch Kommandos, auf die hier nicht mehr eingegangen werden soll, enthält der Monitor noch eine Reihe weiterer, häufig verwendeter Programmteile.

Im folgenden werden diese Monitorfunktionen beschrieben. Dazu wird außerdem die Adresse, unter der dieses Unterprogramm aufrufbar ist, und ein Datenbyte (DB), über dessen Verwendung weiter unten noch etwas gesagt wird, angegeben. In verschiedenen Fällen müssen dem Programm bestimmte Parameter zur Verfügung gestellt werden.

Das ist einmal über die CPU-Register und zum anderen über bestimmte Arbeitszellen des Monitors, die sich im RAM-Bereich befinden, möglich. In die letztgenannten Zellen werden die erforderlichen Parameter mit dem M-Kommando geschrieben.

Die Adressen einiger ausgewählter Arbeitszellen des Monitors sind:

```
Name  Adresse Anzahl der Bedeutung
              Byte
SOIL  0016     2       Anfangsadresse der Eingabe-
                       zeile (Eingabepuffer)
ARG1  001B     2       1. Parameter eines Kommandos
ARG2  001D     2       2. Parameter
AR03  0023     2       3. Parameter
CURSR 002B     2       Kursoradresse
```

Nachfolgend die Monitorfunktionen:

OUTCH (OUT CHARACTER)
Adr. F21BH DB 00H

Ausgabe des im A-Register stehenden Zeichens über den Videotreiber. 4 Steuerzeichen werden vom Z1013-Video-Treiber speziell verarbeitet:

08H - Kursor links
09H - Kursor rechts
OCH - Bildschirm löschen
ODH - neue Zeile; bei Erreichen des unteren Bildschirmrandes wird gerollt

INCH (IN CHARACTER)
Adr. F20CH DB 01H

Mit dieser Routine wird die Eingabe eines Zeichens von der Tastatur in das A- Register realisiert. Dabei wird die Routine INKEY genutzt. Die Register BC, DE und HL werden gerettet. Der Rücksprung aus INCH erfolgt nur bei (A) ungleich 0. Ansonsten befindet sich der Monitor in einer Eingabewarteschleife.

PRST7 (PRINT STRING mit dem 7. Bit als Endzeichen)
Adr. F2A5H DB 02H

Die der Datenbytedefinition (DB 2) folgende Bytekette wird ausgegeben, bis das 7. Bit gesetzt ist. Graphikzeichen (80H..FFH) sind also mit dieser Routine nicht ausgebbar. Für diesen Zweck ist ein Unterprogramm mit OUTCH aufzubauen.

INHEX (Konvertierung einer max. 4-steillgen hexadezimalen Zeichenkette in das interne Format)
Adr. F2F4H DB 03H

Die Routine realisiert die Konvertierung einer max. 4-stelligen hexadezimalen Zeichenkette in das Format eines Doppelregisterinhaltes. Die Anfangsadresse muss im DE-Register übergeben werden. Führende Leerzeichen werden überlesen. Das einer max. 4-stelligen hexadezimalen Zeichenkette folgende Leerzeichen bzw. jedes andere Zeichen, welches verschieden von den Hexa-Zeichen ist, wird als Trennzeichen interpretiert. Der konvertierte Wert steht im HL-Register. Bei längeren Zeichenketten erfolgt keine Fehlerausschrift, sondern im HL-Register befindet sich der gewandelte Wert der letzten 4 Hexa-Zeichen.

INKEY (IN KEYBOARD/Tastatureingabe)
Adr. F130H DB 04H

Tabelle 1:

```
Shift-Ebene 0:

Zei-
len-   Spalten-Nr.
Nr.     0  1  2  3  4  5  6  7
------------------------------
0       @  A  B  C  D  E  F  G
   HEX 40 41 42 43 44 45 46 47

1       H  I  J  K  L  M  N  O
   HEX 48 49 4A 4B 4C 4D 4E 4F

2       P  Q  R  S  T  U  V  W
   HEX 50 51 52 53 54 55 56 57

3      S1 S2 S3 S4 <- SP -> Enter
   HEX             08 20 09 0D
------------------------------

Shift-Ebene 1:

        0  1  2  3  4  5  6  7
------------------------------
```

```
0       X  Y  Z  [  \  ]  ^  _
   HEX 58 59 5A 5B 5C 5D 5E 5F

1       0  1  2  3  4  5  6  7
   HEX 30 31 32 33 34 35 36 37

2       8  9  :  ;  <  =  >  ?
   HEX 38 39 3A 3B 3C 3D 3E 3F
------------------------------

Shift-Ebene 2:

        0  1  2  3  4  5  6  7
------------------------------
0       x  y  z  {  |  }  ~  _
   HEX 78 79 7A 7B 7C 7D 7E 7F

1      SP  !  "  #  $  %  &  '
   HEX 20 21 22 23 24 25 26 27

2       (  )  *  +  ,  -  .  /
   HEX 28 29 2A 2B 2C 2D 2E 2F
------------------------------

Shift-Ebene 3:

        0  1  2  3  4  5  6  7
------------------------------
0       '  a  b  c  d  e  f  g
   HEX 60 61 62 63 64 65 66 67

1       h  i  j  k  l  m  n  o
   HEX 68 69 6A 6B 6C 6D 6E 6F

2       p  q  r  s  t  u  v  w
  HEX  70 71 72 73 74 75 76 77
------------------------------

SHift-Ebene 4:
        0  1  2  3  4  5  6  7
------------------------------
0         UA                UG
   HEX 10 11 12 13 14 15 16 17

1              STOP
   HEX 00 01 02 03 04 05 06 07

2     C-L C-R     CLS ENT
   HEX 08 09 0A 0B 0C 0D 0E 0F
------------------------------
```

Mit Shift 4 und der Taste G (17H) wird im INKEY auf Graphik umgeschaltet, d. h. es wird zum ermittelten Hexa-Kode eine 80H aufaddiert. Die Umschaltung in die A-Ebene erfolgt mit Shift 4 und der Taste A (91H).

```
C-L  : Kursor links
C-R  : Kursor rechts
CLS  : Bildschirm löschen
ENT  : ENTER
STOP : Programmabarbeitungen stoppen
```

INLIN (Eingabe einer Zeile mit führendem Promptsymbol)
Adr. F2B3H DB 05H

Ausgabe eines Promptsymbols und anschließende Eingabe einer Zeichenkette bis Enter (0DH). die Startadresse für die Eingabezeile wird in SOIL zwischengespeichert und kann nach Rückkehr für die Auswertung, z. B. für INHEX, verwendet werden. Da jedes eingegebene Zeichen durch die Folge INCH/OUTCH bis 0DH sofort auf dem Bildschirm ausgegeben wird, erfolgt beim Rollen am unteren Bildschimrand automatisch eine Korrektur von SOIL um -20H.

OUTHX (OUT A-Register hexadezimal)
Adr. F301H DB 06H

Ausgabe des A-Registers hexadezimal. Es werden pro Byte zwei Zeichen ausgegeben.

OUTHL (OUT A-Register hexadezimal)
Adr. F31AH DB 07H

Ausgabe des HL-Registers hexadezimal. Es werden 4 Zeichen ausgegeben.

CSAVE (Save to Cassette)
Adr. F369H DB 08H

Entspricht dem S-Kommando des Monitors, wobei die Anfangsadresse und die Endadresse vorher in ARG1 und ARG2 einzutragen sind.

CLOAD (Load from Cassette)
Adr. F3F8H DB 09H

Die Routine entspricht dem L-Kommando des Monitors. Anfangs- und Endadresse müssen vorher in ARG1 und ARG2 eingetragen werden.

MEM (Modify Memory)
Adr. F325H DB 0AH

Die Routine entspricht dem M-Kommando des Monitors. Vor Ansprung über RST 20H muss die Anfangsadresse in ARG1 eingetragen werden.

WIND (Rollfenster für Bildschirmbereich)
Adr. F6D1H DB 0BH

Entspricht dem W-Kommando des Monitors. In ARG1 und ARG2 sind Anfangsadresse und Endeadresse+1 für das Rollfenster einzutragen.

OTHLS (Ausgabe von 2 Byte hexadezimal entspr. der Adresse im HL-Register)
Adr. F5C7H DB 0CH

Entsprechend der Adresse im HL-Hegister werden 2 Byte = 4 Zeichen (erst High- Teil, dann Low-Teil) und anschließend ein Leerzeichen ausgegeben.

OUTDP (Ausgabe eines Doppelpunktes (:) und weiter wie OTHLS)
Adr. F5C4H DB 0DH

siehe OTHLS

OUTSP (Ausgabe eines Leerzeichens)
Adr. F5CFH DB 0EH

Ausgabe eines Leerzeichens

TRANS (Transfer)
Adr. F51DH DB 0FH

Die Routine entspricht dem T-Kommando des Monitors. In ARG1, ARG2 und ARG3 sind vorher die Werte für „von Adresse“, „auf Adresse“ und Byteanzahl einzutragen.

INSTR (Eingabe einer Zeichenkette)
Adr. F2B9H DB 10H

Es wird die Eingabe einer Zeichenkette abgefordert, die mit Enter abzuschließen ist. Wie in INLIN steht in SOIL die Anfangsadresse der Zeichenkette für eine anschließende Auswertung. Im INSTR wird kein führendes Promptsymbol ausgegeben.

KILL (Füllen eines Speicherbereichen mit einem Byte)
Adr. F50DH DB 11H

Die Routine entspricht dem K-Kommando des Monitors. ARG1, ARG2 und ARG3 sind vorher mit „von Adresse“, „bis Adresse“ und dem zu füllenden Byte zu laden.

HEXUM (Hexa-Umschaltung)
Adr. F6B8H DB 12H

Die Routine entspricht dem H-Kommando des Monitors. Umschaltung der Tastatur auf die Zeichen 0..7, 8..? in die Shift-Ebene Null. Diese Umschaltung ist z. B. vor Zifferneingabe sehr sinnvoll.

ALFA (Alpha-Umschaltung)
Adr. F6C5H DB 13H

Umschaltung der Tastencodetabelle auf die Zeichen H…W in der Shift-Ebene Null. Dieser RST 20H entspricht dem A-Kommando des Monitors.

Ihre Anwendung in eigenen, selbst gefertigten Programmen erleichtert die Programmierung sehr wesentlich und verkürzt den sonst erforderlichen Programmumfang. Der Anwender muss nur wissen, wie diese Programmteile aufgerufen werden und auf welche Art die Übermittlung der Parameter vorgenommen wird.

Diese aufgeführten Monitorfunktionen können ohne Einschränkungen aus beliebigen Nutzerprogrammen aufgerufen werden.

Ihr Aufruf kann erfolgen in der Art: 'CALL adr', wobei als Adresse die Aufrufadresse der Monitorroutine angegeben wird. Alle Routinen werden mit dem Befehl 'RET' beendet, so dass die Programmfortsetzung im aufrufenden Programm gewährleistet ist.

In Anlage 4 sind wichtige Arbeitszellen des Monitors angegeben.

Allerdings muss bei einer eventuellen Änderung des Monitors, die zwangsläufig zu neuen Adressen der Rufe führt, in allen Nutzerprogrammen, die diese Aufrufe benutzen, diese neuen Adressen in den 'CALL'-Befehlen geändert werden.

Deshalb wurde noch eine andere Art des Aufrufs bereitgestellt, die unabhängig von Änderungen im Monitor stets den richtigen Anschluss garantiert Dazu wird anstelle des Aufrufes mit dem 'CALL'-Befehl eine der Restart-Adressen verwendet. Beim Abarbeiten des Restart-Befehls 'RST 20' (E7H) wird ein 'CALL'-Befehl zur Adresse 0020H ausgeführt, die Rücksprungadresse wurde vorher in den Keller gerettet. Auf der Adresse 20H steht ein Sprungbefehl in den Monitor, der beim Programmstart oder nach Reset automatisch dort eingetragen wird.

Im Monitor befindet sich dann eine Auswertelogik, die anhand der gekellerten Rücksprungadresse die konkret geforderte Monitorroutine ermittelt und deren Realisierung einleitet. Die benötigte Auswahlinformation steht in einem dem RST-Befehl folgenden Byte, die Rücksprungadresse kann also direkt als Zeiger auf dieses Byte verwendet werden, muss dann aber zur Programmfortsetzung auf den dem Byte folgenden Befehl gestellt werden.

Ein einfaches Beispiel soll das verdeutlichen:

In einem Programmstück so zuerst der Bildschirm gelöscht und anschließend eine beliebige Anzahl von Zeichen eingegeben werden. Diese Zeichen sollen sofort wieder auf dem Bildschirm erscheinen, die Betätigung der Entertaste beendet das Programm und kehrt in den Monitor zurück.

Zum Vergleich wurde dieses Beispiel einmal unter Verwendung von 'CALL'- und einmal unter Verwendung von 'RST'-Befehlen programmiert.

```
Befehls-       Maschinen-
zahler         code

               ;
               ; BEISPIEL MIT CALL-BEFEHLEN
               ;
               OUTCH: EQU 0F21B  ; AUSGABE ZEICHEN
               INCH:  EQU 0F20C  ; EINGABE ZEICHEN
1000 3E 0C     BSP:   LD  A,0CH  ; LOESCHEN BILD-
                                   SCHIRM
1002 CD 1B F2         CALL OUTCH ; AUSGABE
1005 CD 0C F2  M1:    CALL INCH  ; EINGABE
1008 CD 1B F2         CALL OUTCH ; AUSGABE
100B FE 0D            CMP  ODH   ; ENTER ?
100D C2 05 10         JPNZ M1    ; NEIN
1010 C9               RET        ; JA-->MONITOR

               ;BEISPIEL MIT RST-BEFEHLEN
               ;
```

```
               OUTCH:  EQU 0     ; AUSGABE ZEICHEN
               INCH:   EQU 1     ; EINGABE ZEICHEN
1000 3E 0C     BSP2:   LD  A,0CH ; LOESCHEN BILD
                                   SCHIRM
1002 E7                RST 20H   ; RUF
1003 00                DB OUTCH  ; AUSGABE
1004 E7        Ml:     RST 20H   ; RUF
1005 01                DB INCH   ; EINGABE
1006 E7                RST 20H   ; RUF
1007 00                DB OUTCH  ; AUSGABE
1008 FE 0D             CMP 0DH   ; ENTER ?
100A C2 04 10          JPNZ M1   ; NEIN
100D C9                RET       ; JA -> MONITOR
```

Falls die Funktionen des Monitors durch eigene Programmteile realisiert werden sollen, muss der Sprungbefehl auf der Adresse 20H, der auf drei Auswerteroutinen im Monitor zeigt, durch einen Sprungbefehl in eine eigene Auswerteroutine ersetzt werden. Bleibt die Zuordnung der Auswahlinformationen erhalten, so sind auch in diesem Fall keinerlei Änderungen in den eigenen Programmen notwendig die den Aufruf über den 'RST'-Befehl verwenden.

Damit ist es z. B. möglich, eigene Ein- und Ausgaberoutinen, mit denen die Bedienung anderer Geräte realisiert wird, zu verwenden und damit die Ein- und Ausgaben über andere Geräte zu erreichen.

Abschließend sei noch bemerkt, dass ein komplettes Assemblerlisting des Monitors mit Hilfe des im Anhang abgedruckten Reassemblers durch Reassemblieren des ROM-Inhaltes erzeugt werden kann.

5.1.2. Erweiterungen des Monitors

Falls der Kommandoumfang und die Leistungen des Monitors nicht ausreichen, ist die Nutzung eigener Programmteile zur Erweiterung möglich. Damit diese auch aus der Kommandoschleife des Monitors nutzbar sind, müssen diese Kommandos mit dem Zeichen '@' eingeleitet werden. Statt des nun üblichen Leerzeichens steht nun ein ASCII-Zeichen, das das aufzurufende Programm spezifiziert. Der Anschluss zur Kommandoschleife wird in folgender Weise hergestellt.

Ab der Adresse 00B0H werden die hexadezimale Verschlüsselung des ASCII-Zeichens sowie die Anfangsadresse des zugehörigen Programmteiles eingetragen. Als Beispiel sollen durch die zusätzlichen Kommandos '@B' der BASIC-Interpreter (siehe Abschnitt 5.2.) ab der Adresse 0100H gestartet werden bzw. durch '@C' ein Wiederstart ab der Adresse 0103H erfolgen.

Dazu wird ab Adresse 0B0H eingetragen:

```
00B0   42    Zeichen "B"
00B1   00    niederwertiger Teil der Startadresse
00B2   01    höherwertiger Teil der Startadresse
00B3   43    Zeichen "C"
00B4   03    niederwertiger Teil der Startadresse
00B5   01    höherwertiger Teil der Startadresse
```

Die Eingabe zusätzlicher Programmteile kann z. B. im Adressbereich zwischen 0100H und 03FFH erfolgen, da die meisten Programme nicht unterhalb von 0400H arbeiten und die Arbeitszellen des

Monitors unterhalb von 0100H liegen. Diese zusätzlichen Programmteile müssen allerdings immer wieder in den Speicher gebracht werden, da sie mit Ausschalten des Gerätes verlorengehen. In dem nachfolgenden Abschnitt wird die Eingabe, Speicherung und Testung selbsterstellter Programme beschrieben.

5.2. Hinweise zum Aufbau einer Programmbibliothek

Der Monitor allein als nutzbares Programm wird bald in all seinen Möglichkeiten ausgeschöpft sein, so dass der Wunsch nach weiteren Programmen geweckt wird.

Da aber nur der Monitor in einem nichtflüchtigen Speicher steht, gehen andere, meist sehr mühsam eingegebene Programmteile nach dem Ausschalten wieder verloren. Aus diesem Grund ist es notwendig, sich eine Programmbibliothek auf einem externen Datenträger anzulegen, in die jedes neue Programm aufgenommen wird, um es für spätere Nutzung immer wieder zur Verfügung zu haben. Im Monitor sind zu diesem Zweck die Kommandos „L“ (load from cassette) und „S“ (Save to cassette) enthalten. Mit diesen Kommandos ist es möglich, Programme aus dem Speicher auf ein Magnetband, auszulagern und wieder in den Speicher zu laden.

Die Eingabe der Programme erfolgt beim ersten mal meist mittels der Tastatur. Im allgemeinen wird die hexadezimale Ziffernfolge, die in einem vorgegebenen Speicherbereich eingetastet werden soll, einer Liste entnommen (z. B. die in diesem Buch enthaltenen Testbeispiele).

Diese Listen entstanden entweder als Resultat eines Übersetzungslaufes auf einer EDV-Anlage oder wurden durch manuelle Übersetzung selbst gefertigt. In jedem Fall wird es sich um ein in sich abgeschlossenes Programmteil handeln. Eine Programmänderung oder -erweiterung während der Eingabe wird kaum zu brauchbaren Ergebnissen führen.

Einige Programmteile, die als Liste geliefert werden, sind nur ein Speicherabzug in der Art, wie er auch mit den D-Kommando auf dem Bildschirm zu sehen ist.

Damit können recht einfach Liste und tatsächlicher Speicherinhalt miteinander verglichen werden. Zur Eingabe wird das M-Kommando des Monitors verwendet.

Mit dem Parameter wird die Anfangsadresse des Programmes an gegeben. Es ist zweckmäßig, vorher die Tastatur mit dem H-Kommando in die hexadezimale Eingabe zu schalten. Die Ziffern sind jetzt ohne Benutzung der Shift-Taste direkt erreichbar. Anschließend kann ein großer Bereich der Liste eingegeben werden.

Es empfiehlt sich, nach einer bestimmten Anzahl von hexadezimalen Ziffern die Eingabe mit dem Semikolon „;“ zu beenden und mit dem Kommando 'D :' den bis dahin eingetasteten Abschnitt zu kontrollieren. Die zeilenweise mitausgegebenen Prüfsummen (CS) können mit dem Listenausdruck verglichen werden. Die weiteren Blöcke werden dann wieder mit dem M-Kommando ab der Abbruchadresse eingegeben.

Nach endlicher Zeit steht das Programm im gewünschten Speicherbereich und entspricht dem Original auf der Liste. Es wäre nun grundfalsch, sofort mit der Abarbeitung zu beginnen, vielmehr sollte das Programm zuerst einmal auf Magnetband abgespeichert und somit gesichert werden. Das erspart wiederholtes Eingeben von Programmteilen über Tastatur und erleichtert die Arbeit wesentlich.

Meist wird man diese Abspeicherung nicht gleich auf dem Magnetband vornehmen, auf dem sich die

anzulegende Programmbibliothek befindet, sondern definiert ein anderes als „Arbeitsband“.

Auf jeden Fall sollte man sich sehr genau die Bandstellen zu Beginn und Ende sowie die im S-Kommando verwendeten Parameter notieren. Das erleichtert ein späteres Wiederauffinden der gespeicherten Informationen und gewährleistet das ordnungsgemäße Einlesen in den Speicher.

Möchte man das wiederholte Einlesen vom Band, das ja auch einen gewissen Aufwand erfordert, so gering wie möglich halten und ist noch genügend Speicherplatz frei, empfiehlt sich auch folgende Methode:

Mit dem T-Kommando (Transfer) wird eine gleichartige Kopie des Programms in einem anderen, nicht genutzten Speicherbereich erzeugt. Falls jetzt das Programm beim Testen zerstört, wird ist es möglich, wieder mit dem T-Kommando die Kopie auf den ursprünglichen Speicherbereich zurückzuspeichern und damit das Programm wieder herzustellen.

Im allgemeinen werden Programme nach dem Eingeben nicht sofort richtig und fehlerfrei arbeiten, dazu sind zu viele Fehlermöglichkeiten vorhanden. Sie reichen von einfachen Eingabefehlern über falsche Anschlussstellen bis hin zu Fehlern im Programmentwurf (insbesondere bei manuell assemblierten Programmen).

Damit ergibt sich die Notwendigkeit, diese Programme auszutesten. Der Monitor stellt dazu einige Hilfen zur Verfügung, von denen Haltepunkt und Schrittbetrieb bereits erwähnt wurden.

Zum Testen von Programmen unter den für diese Verhältnisse typischen Bedingungen ergibt sich etwa nachfolgende Herangehensweise, die mit fortschreitendem Erfahrungsstand individuell modifiziert werden kann:

- Zu Beginn der Programmtestung wird mit dem I-Kommando der Registerrettebereich gelöscht und damit ein definiertes Laden der CPU-Register gewährleistet.
- Der Haltepunkt wird mit dem B-Kommando auf den ersten zu testenden Befehl des Programmes gesetzt und mit dem E-Kommando gestartet.
- Mit Erreichen des Haltepunktes wird mit dem N-Kommando eine schrittweise Abarbeitung vorgenommen. Gegebenenfalls ist die Ausgabe der Registerinhalte auf Ausführung der Befehle zu kontrollieren.
- Auftretende Programmschleifen sowie häufig aufgerufene Routinen (Unterprogramme) sind mindestens einmal im Schrittbetrieb abzuarbeiten. Sofern man danach von der Richtigkeit dieser Programmteile überzeugt ist, wird bei wiederholtem Aufruf die Haltepunktadresse auf den dem Aufruf oder dem Schleifenausgang folgenden Befehl gelegt und damit diese Programmteile direkt abgearbeitet.
- Alle im Schrittbetrieb abgearbeiteten Befehle werden bei richtiger Funktion in der Liste abgehakt, damit ist jederzeit ein Überblick über das zu testende Programm vorhanden.
- Falls die Testung des Programmes nicht auf einmal durchgängig erfolgen kann, sei es durch zu großen Programmumfang oder durch Programmabsturz (undefiniertes Verhalten), kann mittels Aktivieren des letzten erfolgreich erreichten Haltepunktes oder der in Schrittbetrieb erreichten Adresse das zu testende Programm jederzeit wieder neu gestartet werden. Der weitere Ablauf ist dann genauso wie bereits beschrieben.
- Falls durch den Monitor nicht bereits die Protokollierung der Monitorbefehle bei Monitorrufen

verhindert wird, sollte durch Setzen der Haltpunktadresse auf den dem Monitorruf folgenden Befehl diese Protokollierung ausgeschlossen worden.

Bei Beachtung dieser Hinweise ist die Austestung aller Programme möglich. Sollten während der Testung in Programm Fehler festgestellt werden, die eine Veränderung der Programmstruktur erforderlich machen, ist das bei den im Maschinenkode vorliegenden Programmen besonders schwierig. Entweder man bemüht sich um eine Änderung mit anschließender Neuübersetzung oder es wird an der zu ändernden Stellen eine Maschinenkodeeinfügung vorgenommen.

Dazu wird an Stelle von drei oder mehr Befehlsbyte ein 'CALL'-Befehl eingefügt, dessen Adresse auf das erste freie Byte am Programmende oder in einen anderen freien Speicherbereich zeigt. Die dabei verdrängten Befehlsbyte müssen vollständige Befehle sein, erforderlichenfalls sind mehr als drei Byte zu ersetzen und die nicht benötigten durch 'NOP'-Befehle aufzufüllen. Keinesfalls darf ein Relativsprungbefehl ersetzt werden, da dessen Sprungdistanz von seiner Position im Programm abhängig ist.

An der durch den 'CALL'-Befehl adressierten Stelle stehen als erstes die durch die Einfügung verdrängten Befehlsbyte sowie die geplante Erweiterung. Mit einem 'RET'-Befehl wird die Abarbeitung wieder an der ursprünglichen Stelle im Programm fortgesetzt.

Diese Art der Maschinenkodeerweiterung ist zu Testzwecken durchaus möglich, sollte aber zu einem späteren Zeitpunkt durch Änderung im Programm und Neuübersetzung Berücksichtigung finden.

Wenn die Programme vollständig ausgetestet sind, werden sie in die Programmbibliothek aufgenommen. Auf diesem Magnetband befinden sich eine Reihe von bereits ausgetesteten Programmen, die bei Bedarf in dcn Speicher geladen werden können.

Allergrößten Wert ist auf ein genaue Inhaltsverzeichnis zu legen, damit diese Programmteile wiedergefunden und später problemlos geladen werden können.

Dieses Laden erfolgt mit dem L-Kommando des Monitors, nachdem zuvor die benötigten Parameter dem Inhaltsverzeichnis entnommen wurden. Von besonders wichtigen Programmteilen sollte man sich noch eine weitere Kopie anlegen, um bei einer eventuellen Zerstörung der Programmbibliothek diese jederzeit wieder herstellen zu können.

5.3. BASIC

5.3.1. Programmiersprache BASIC

BASIC ist eine sogenannte höhere Programmiersprache, eine Sprache also, die sich nicht direkt auf die Maschinensprache des Rechners bezieht. Diese Sprache wurde um 1965 von John G. Kemeny und Thomas E. Kurtz im Dartmouth College in den USA entwickelt. Sie hatten dabei die Entwicklung einer Computersprache vor Augen, die einerseits vom Anfänger leicht zu erlernen ist und andererseits viele Möglichkeiten bietet. Man sollte mit dieser Sprache leicht numerische (Zahlen-) Probleme angehen können, aber ebenso sollten Verwaltungsaufgaben, bei denen die Textverarbeitung eine große Rolle spielt, in der Sprache zu behandeln sein. Deshalb ersann man für diese Sprache die Bezeichnung BASIC, ein Wort, das konstruiert wurde aus den Anfangsbuchstaben von: Beginner's All purpose Symboic Instruction Code (etwa: Anfänger-Allzweck-Symbolik-Instruktions-Code).

Neben den Wörtern „Beginner's“ und „All purpose“ (Allzweck), deren Bedeutung ohne weiteres

einleuchtet, können die Wörter „Symbolic Instruction Code“ vielleicht ein wenig verwirren. Damit wird nur gesagt, dass man mit einem sogenannten symbolischen Befehlskode arbeitet. Das sind Schlüsselwörter, mit denen bestimmte Verarbeitungsbefehle mit Hilfe von Symbolen angegeben werden, z. B. das Addieren oder Subtrahieren. Also nur wenige Wörter, deren Bedeutung bei der weiteren Beschäftigung mit der Sprache ohne weiteres einleuchten wird.

Kemeny und Kurtz haben mit ihrem einfachen Aufbau genau ins Schwarze getroffen. Die Erwartungen haben sich erfüllt: BASIC wurde eine sehr populäre Programmiersprache. Sie hat sich im Unterricht bewährt, und auch im Laboratorien und in Betrieben gibt es zahllose Computer, die den Befehlen in BASIC gehorchen.

5.3.2. Der BASIC-Interpreter

Die eingegebenen Programmzeilen müssen ihrem Inhalt entsprechend bestimmte Verarbeitungsleistungen aufrufen. Diese Aufgabe übernimmt der BASIC-Interpreter. Dieser Interpreter arbeitet Anweisung für Anweisung interpretativ ab, d. h. jedem entschlüsselten Befehl oder Kommando wird ein entsprechendes Maschinenprogramm zugeordnet und mit den in der BASIC-Anweisung angegebenen Zahlenwerten abgearbeitet. Diese interpretative Abarbeitung nutzt ein Maschinenprogramm für alle im gesamten Programm vorkommenden gleichen Kommandos bzw. Befehle. Damit sind genau so viele verschiedene Maschinenprogramme notwendig, wie es verschiedene Kommandos und Befehle gibt. Damit ist die Länge des BASIC-Interpreter festgelegt, unabhängig von der Länge der abzuarbeitenden Programme. Als Nachteil ist die geringere Rechengeschwindigkeit gegenüber übersetzten Programmen anzusehen.

Ein Vorteil ist die einfache Dialogfähigkeit, BASIC-Zeilen erscheinen so wieder auf dem Bildschirm, wie sie eingegeben wurden. Außerdem können einzelne Anweisungen sofort ausgeführt werden. Auf diese Weise kann der Computer an Stelle eines Tischrechners verwendet werden. Auf diese Betriebsart wird noch genauer eingegangen.

An dieser Stelle sei auch auf einen Nachteil des Interpreters hingewiesen. Wie spätere Beispiele zeigen werden, kann ein Teil des BASIC-Programmes häufiger als nur einmal ausgeführt werden, z. B. bei der Unterprogrammarbeit oder in Programmschleifen. Vom Interpreter wird aber jedes mal, wenn der entsprechende Abschnitt an der Reihe ist, Anweisung um Anweisung neu interpretiert. Das hat zur Folge, dass solche Programme längere Verarbeitungszeiten gegenüber gleichen Programmlösungen in Maschinensprache erfordern. Im Extremfall kann das dazu führen, dass besonders zeitkritische Probleme nur in Maschinensprache lösbar sein werden. Diese Maschinenprogramme können dann innerhalb einer BASIC- Anweisung aufgerufen werden.

5.3.3. Laden des BASIC-Interpreters

Da der BASIC-Interpreter selbst ein Maschinenprogramm darstellt, benötigt er im Speicher des Z1013 einen Speicherplatz von ca. 2.75 KByte. Dazu wird noch weiterer Speicherraum zum Ablegen der BASIC-Programme gebraucht.

Im Anhang ist eine Liste des Maschinenkodes des BASIC-Interpreter-Programmes enthalten. Diese Liste von Maschinenbefehlen muss beim ersten Mal per Hand in den Rechner eingetippt werden, wobei bei Adresse 0100H zu beginnen ist. Diese mühsame Arbeit ist aber nur beim erstem Mal notwendig. Deshalb soll an dieser Stelle darauf verwiesen werden, dass dabei entsprechende Sorgfalt von Nöten ist. Ein einziger Fehler kann das Interpreterprogramm funktionsunfähig machen.

Das Eintippen erfolgt mit dem in 1.3. kennengelernten M-Kommando. Es wird immer ein kleiner Abschnitt eingegeben und anschließend mit dem D-Kommando wieder angezeigt. Durch Vergleich mit den Prüfsummen je Zeile auf dem Bildschirm und in der Liste wird die Fehlerfreiheit festgestellt. Muss die Arbeit unterbrochen werden oder ist der Interpreter vollständig eingegeben, kann er wie jedes andere Maschinenkode-Programm auf Magnetband gespeichert werden und steht ab jetzt immer zur Verfügung. Soll jetzt mit dem MRB Z1013 ein BASIC-Programm bearbeitet werden, so muss vorher nur noch der BASIC-Interpreter von Magnetband in den Adressbereich 0100H bis 0BFFH geladen und mit dem J-Kommando ab Adresse 0100H gestartet werden.

5.3.4. Arbeit mit dem BASIC-Interpreter

Nach dem Start des BASIC-Interpreters erscheint auf dem vorher gelöschten Bildschirm die Meldung 'robotron Z 1013 BASIC 3.01', in der nächsten Zeile 'READY' und am Beginn der folgenden Zeile das Zeichen '>' (größer als) als Promptsymbol.

Immer wenn das Zeichen '>' auf dem Bildschirm auftaucht, befindet sich der Interpreter in einer Eingabeschleife, d. h., dass der Interpreter zur Eingabe von Befehlen, Kommandos oder Programmzeilen bereit ist. Eine solche Programmzeile hat folgenden Aufbau: [Zlnr] anweisung 1 [; anweisung 2, ...]. Die eckige Klammer bedeutet, dass diese Eingaben nicht unbedingt getätigt werden müssen.

Die Länge einer Programmzeile darf 64 Zeichen nicht überschreiten. Jede Programmzeile und jede Kommandoeingabe ist mit der Enter-Taste abzuschließen

Beginnt die eingegebene Zeile mit einer Zeilennummer (Zlnr), so wird diese Zeile als Programmzeile interpretiert und abgespeichert. Diese Zeilennummern sind ganze Zahlen im Bereich zwischen 1 und 32767. Bei der Nummerierung der Programmzeilen geht man sinnvollerweise in Zehnerschritten vor. Dadurch ergibt sich die Möglichkeit, noch genügend Einfügungen in bereits bestehende Programme vorzunehmen. Die Abarbeitung des BASIC-Programmes erfolgt in der Reihenfolge der Zeilennummern.

Alle anderen Eingaben ohne Zeilennummern werden als Befehl zur sofortigen Ausführung angesehen. Sind sie zulässig werden sie ausgeführt, sonst erfolgt eine Fehlermeldung. Danach kann die nächste Eingabe erfolgen.

Die Arbeit ohne Zeilennummer nennt man auch Tischrechnermodus.

```
>A=66-20; PRINT A
   46
READY
```

Innerhalb einer einzugebenden Zeile kann beliebig oft mit den Kursortasten „Kursor links '←'“ oder „Kursor rechts '→'“ korrigiert werden. Der Kursor wird unter das fehlerhafte Zeichen bewegt und durch Eingabe des richtigen Zeichens der Fehler behoben. Danach muss der Kursor wieder an das Zeilenende gebracht werden (auf die erste freie Zeichenstelle) und die Zeile kann mit der Enter- Taste abgeschlossen werden.

Soll eine bereits im Programmspeicher abgespeicherte Programmzeile geändert werden, wird diese einfach neu eingegeben (mit der gleichen Zeilennummer). Soll eine Zeile gelöscht werden, so muss nur ihre Zeilennummer angegeben werden.

Alle Befehle und Kommandos können mit einem Punkt nach dem ersten oder nach weiteren

Buchstaben abgekürzt werden. In der im Anhang befindlichen Befehlsliste sind die möglichen Abkürzungen zu finden. Es sind auch kürzere Varianten der Befehle möglich, aber dadurch kommt es im Interpreter unter Umständen zu Verwechslungen mit anderen Befehlen und zu fehlerhaften Abarbeitungen.

Auch Leerzeichen zwischen den einzelnen Elementen der Programmzeilen können weggelassen werden. Diese beiden Möglichkeiten der Programmverkürzung erlauben es, den Programmspeicher besser auszunutzen, um mehr Programmzeilen unterzubringen. Dadurch geht aber die Übersichtlichkeit der Programme verloren.

Es dürfen mehrere Anweisungen in einer Programmzeile untergebracht werden. Diese Anweisungen sind durch ein Semikolon voneinander zu trennen.

Der BASIC-Interpreter realisiert eine einfache Ganzzahlarithmetik in den vier Grundrechenarten Addition, Subtraktion, Multiplikation und Division (+ - * /) im Zahlenbereich von -32768 bis +32767. Aus diesem Grund besitzt er nur einen Datentyp. Zu beachten ist, dass auch die Division nur ganzzahlig ausgeführt wird, d. h., dass z. B. 9/4=2 ist. Der Teil des Resultates hinter dem Komma wird einfach weggelassen.

Den meisten der Schlüsselworte folgt ein weiterer Ausdruck. Das können Zahlen, Variable oder arithmetische Konstruktionen mit diesen sein. Erforderlichenfalls sind derartige Konstruktionen mit Klammern aufzubauen, um diese Konstruktionen mathematisch eindeutig zu machen. Der Interpreter arbeitet nach

der bekannten Rechenregel: Punktrechnung geht vor Strichrechnung. Mehrere Klammern können dabei beliebig geschachtelt werden. Innerhalb eines Befehls können weitere Befehle oder Funktionen verwendet werden.

Als Variable sind alle Buchstaben des Alphabets von A bis Z erlaubt. Eine Variable darf aber nur aus einem einzelnen Buchstaben bestehen. Es können nur Großbuchstaben verwendet werden.

Mit dem Symbol '@' ist die Nutzung eines eindimensionalen Feldes (Vektor) möglich. Die Teilanweisung @(A) stellt dabei das A-te Element des Feldes dar. Anstelle von A kann sowohl ein anderer Buchstabe des Alphabetes als auch eine Zahl oder eine arithmetische Konstruktion stehen, d. h. ein Ausdruck wie oben bereits beschrieben.

Werden bei der Arbeit mit dem BASIC-Interpreter Syntaxfehler gemacht, so werden diese Fehler erkannt und angezeigt. Logische Fehler im Programm kann der Interpreter nicht finden, das bleibt dem Geschick des Programmierers überlassen. Der BASIC-Interpreter kennt drei verschiedene Fehlermeldungen:

```
WHAT? - Das Schlüsselwort bzw. der Ausdruck sind nicht er-
        laubt bzw. fehlerhaft, d. h. der Interpreter versteht
        die Anweisung nicht.

HOW?  - Die Ausführung der Anweisung ist im Rahmen der Mög-
        lichkeiten dieses Interpreters nicht möglich (z. B.
        bei einer Zahlenbereichsüberschreitung).

SORRY - Die Ausführung der Anweisung ist zwar möglich, aber
        nicht unter den aktuellen Voraussetzungen( z. B. der
        Programmspeicher ist erschöpft).
```

Tritt eine Fehlermeldung bei der Abarbeitung eines Programmes auf, so wird zur Fehlermeldung auch die Zeile ausgegeben, in der der Fehler aufgetreten ist. An der fehlerhaften Stelle wird vom BASIC-Interpreter ein Fragezeichen eingefügt. Das erleichtert die Fehlersuche.

5.3.5. Kommandos des BASIC-Interpreters

Im nachfolgenden sind alle Befehle und Kommandos, die der BASIC-Interpreter verstehen und ausführen kann, aufgeführt und erläutert. Zusammen mit dem im vorhergehenden Abschnitt gesagten ergeben sich daraus alle Möglichkeiten der im MRB Z 1013 realisierten Programmiersprache BASIC. Die im Anhang befindliche Befehlsliste beinhaltet auch die möglichen Kurzformen.

LIST

Dieses Kommando bewirkt das Auflisten des im Speicher stehen den BASIC- Programmes auf dem Bildschirm. Dadurch lassen sich Programme leicht kontrollieren.

```
>LIST     Auflisten des gesamten Programmes in aufsteigender
          Reihenfolge der Zeilennummern.

>LIST 10  Auflisten des BASIC-Programmes ab der Zeile 10
          Es werden genau 20 Zeilen aufgelistet.
```

RUN

Mit dem Kommando 'RUN' wird ein BASIC-Programm gestartet. Bei der Abarbeitung wird mit der niedrigsten Zeilennummer begonnen. Ist die letzte Zeile abgearbeitet oder eine 'STOP'- Anweisung erreicht, kehrt der Interpreter in die Eingabeschleife zurück und meldet sich mit 'READY' und auf der nächsten Zeile mit den Aufforderungszeichen '>'. Er erwartet jetzt weitere Kommando- oder Befehlseingaben.

NEW

Das im Speicher vorhandene BASIC-Programm wird scheinbar gelöscht. Tatsächlich ist es noch im Programmspeicher enthalten, wird aber bei Neueingabe eines Programmes überschrieben.

BYE

Mit 'BYE' wird die Arbeit mit dem BASIC-Interpreter beendet und ins Monitorprogramm zurückgekehrt. Das zuletzt eingegebene Programm befindet sich noch im Speicher. Wird an dessen Inhalt nichts geändert, kann mit dem Monitorkommande 'J 0103' wieder der BASIC-Interpreter aktiviert werden (Restart). Jetzt erscheint nur ein 'READY' und in der nächsten Zeile das Aufforderungszeichen '>' auf dem Bildschirm. Die Arbeit mit dem BASIC- Interpreter kann fortgesetzt und das vorher eingegebene Programm wieder gestartet werden.

END

Dieser Befehl wird zum Vergrößern des vom Interpreter genutzten Programmspeichers verwendet. Mit 'END' kann also das Programmspeicherende neu gesetzt werden, z. B. bei Speichererweiterung. Lässt der augenblickliche Ausstattungsgrad des MRB Z1013 diesen Speicherbedarf nicht zu, weil er real nicht vorhanden ist, wird die Fehlermeldung 'SORRY' ausgegeben. Dabei ist zu beachten, dass ein Bereich von 140 Byte hinter den Programmspeicher frei bleibt, der vom BASIC-Interpreter intern

verwaltet wird.

Beispiel:

```
>END HEX(3FFF)-140
```

Die oben aufgeführten Kommandos dürfen in keinen Programm auftauchen, sie dienen nur zur Arbeit mit dem Interpreter und werden prinzipiell sofort ausgeführt.

CSAVE

Mit dem Kommando CSAVE „name“ wird ein BASIC-Programm unter dem angegebenen Namen auf Magnetband abgespeichert. Dabei kann der Name maximal 16 Zeichen umfassen und muss von Anführungszeichen eingeschlossen sein.

CLOAD

Mit diesem Kommando wird ein durch CSAVE abgespeichertes BASIC-Programm wieder von Magnetband eingegeben. Zur Kontrolle wird der im CSAVE-Kommando verwendete Name auf dem Bildschirm ausgegeben.

5.3.6. Programmierbare Befehle bzw. Anweisungen

LET

Definiert den Anfang einer Ergibtanweisung, d. h. einer Wertzuweisung. 'LET' muss nicht vorangestellt werden, es dient lediglich der besseren Übersichtlichkeit.

Die im folgenden angegebenen Beispiele stellen nicht in jedem Fall Programme dar, sondern können auch nur Varianten eines Anweisungstypes verdeutlichen.

```
>10  LET A=1
>20  A=50;B=30
>30  LET C=A-B+3
>40  LET X=3+A+(B-3)/C
>50  LET @(3)=24
```

IF

Mit der 'IF'-Anweisung werden Bedingungen für die Ausführung einer der Anweisung folgenden Anweisung festgelegt. Im Zusammenhang mit der GOTO-Anweisung lassen sich damit Programmverzweigungen realisieren.

```
>100 IF B=10 GOTO 200
>108 IF C=0 PRINT 'FERTIG'
>115 IF A+B<100 A=A+1;GOTO 100
```

Die 'IF'-Anweisung selbst darf keine Folgeanweisung in einer Programmzeile sein, muss also immer am Zeilenanfang stehen. Die 'IF'-Anweisung ist damit einer Vergleichsoperation gleichzusetzen.

Nachfolgend sind alle erlaubten Vergleichsoperatoren aufgeführt:

```
>= größer gleich
#  ungleich
>  größer
=  gleich
<  kleiner
<= kleiner gleich
```

Ist die in der IF-Anweisung angegebene Vergleichsoperation wahr, wird die in der gleichen Programmzeile folgende Anweisung ausgeführt, sonst die nachfolgende Programmzeile abgearbeitet.

GOTO

Unbedingter Sprung zu einer Zeile, deren Zeilennummer direkt angegeben wird, oder sich aus den angegebenen Ausdruck berechnet. 'GOTO zlnr' als Direktanweisung (ohne vorangestellte Programmzeilennummer) startet das Programm ab der angegebenen Zeilennummer. In Verbindung mit 'IF' kann 'GOTO' zur Konstruktion von bedingten Sprunganweisungen verwendet werden (siehe auch Beispiele der 'IF'-Anweisung).

```
>100 GOTO 120
>GOTO 120
>110 GOTO 120+B
>120 GOTO A
>120 IF A>0 GOTO 100
```

FOR...TO...STEP...NEXT

Damit lassen sich leicht Programmschleifen aufbauen, die mit einer frei bestimmbaren Anzahl von Schleifendurchläufen abgearbeitet werden sollen. Jede mit 'FOR' eröffnete Schleife muss mit einer NEXT-Anweisung, die die gleiche Zählvariable wie die FOR-Anweisung beinhaltet, abgeschlossen werden.

```
>100 N=10
>110 FOR I=0 to N
>120 LET a=I*10;b=I*I
>121 PRINT A,B, A*B

>150 NEXT I
>160 ...
```

Der Programmabschnitt von Zeile 110 bis Zeile 150 wird N-mal durchlaufen, wobei '1' mit dem Wert '0' beginnend in jedem Durchlauf um '1' erhöht wird, bis der Wert N überschritten wurde. Danach wird die Schleife verlassen.

Eine Schrittweite wird mit dem Schlüsselwort STEP gekennzeichnet. Wird STEP weggelassen, wird der Standardwert 1 genommen.

In der 'FOR..NEXT'-Anweisung können die Anfangs- und Endwerte sowie die Schrittweite der Schleifendurchläufe für die Zählvariable (im Beispiel das 'I') beliebige arithmetische Konstruktionen bzw. Ausdrücke sein. Bei jeden Schleifendurchlauf wird die Zählvariable um den Wert der Schrittweite verändert, bis der Endwert überschritten wird. Bei negativer Schrittweite ist auf die richtige Angabe der Anfangs- und Endwerte zu achten.

```
>110 FOR X=A TO N+B STEP C
>120 ...
...
>150 NEXT X
```

Eine 'FOR-NEXT'-Schleife kann zu jedem beliebigen Zeitpunkt durch eine 'IF..GOTO'-Anweisung verlassen werden. Es darf aber nicht in eine 'FOR…NEXT'- Schleife von außerhalb der Schleife hineingesprungen werden.

GOSUB…RETURN

Mit GOSUB erfolgt der Aufruf eines in BASIC geschriebenen Unterprogramms, welches mit 'RETURN' beendet werden muss. Nach dem Befehl 'RETURN' wird mit dem, dem Unterprogrammruf folgenden Befehl im BASIC-Programm fortgesetzt. Unterprogramme sind dort sinnvoll, wo gleiche Programmteile an mehreren Stellen benötigt werden. Dadurch wird Speicherplatz eingespart. (ähnliche Problematik der MC-Programmierung, siehe Abschnitt 4.) Innerhalb eines Unterprogrammes sind die Variablen des rufenden Programmes ebenfalls gültig und werden zur Parameter- und Ergebnisübermittlung genutzt.

```
>120 C=25;GOSUB 180;PRINT 'ZEIT',
>125 C=60;GOSUB 180;PRINT 'SCHLEIFE'
...
>170 STOP
>180 REM ZEITPROGRAMM
>190 IF C#0 C=C-1; GOTO 190
>210 RETURN
```

REM

Dadurch werden in einen BASIC-Programm; Kommentarzeilen gekennzeichnet. Sie dienen der besseren Übersichtlichkeit der Programme und werden bei der Abarbeitung durch den Interpreter überlesen. Die Programmzeile belegt aber entsprechend ihrer Länge Speicherplatz in RAM-Bereich des Rechners.

>110 REM LET C=1024 >180 REM ZEITPROGRAMM

Diese Zeilen werden nicht mit abgearbeitet.

INPUT

Dadurch wird die Eingabe von numerischen Werten mittels der Tastatur ermöglicht. Ein eingegebener Wert wird dabei einer Variablen zugeordnet. Alle eingegebenen Zeichen werden wieder auf dem Bildschirm ausgegeben. Korrekturen der Eingabe sind mit der Taste 'Kursor links' bzw. 'Kursor rechts' möglich. Die gesamte

Eingabe ist mit der Enter-Taste abzuschließen (nach Korrekturen muss der Kursor auf die erste freie Position in der Eingabezeile gestellt werden!).

Nach der 'INPUT'-Anweisung kann ein in Hochkomma eingeschlossener Text angegeben werden, welcher bei der Ausführung der Anweisung auf dem Bildschirm mit ausgegeben wird. Mittels einer 'INPUT'-Anweisung können mehrere Eingaben ausgeführt werden. Anstelle einer Zahl kann auch ein Ausdruck eingegeben werden.

```
>10 INPUT X
>20 INPUT 'SPRUNGWEITE' S
>30 INPUT 'WERTEPAAR A'A,B
>RUN
X: eingabe
SPRUNGWEITE: eingabe
WERTEPAAR A: eingabe  B: eingabe
```

Das Wort 'eingabe' erscheint nicht auf dem Bildschirm, es wurde hier nur verwendet, um deutlich zu machen, dass an dieser Stelle eine Eingabe mittels der Tastatur erfolgt.

```
>20  INPUT ' 'S
>RUN
: eingabe
```

PRINT

Damit wird die Ausgabe von numerischen Werten und von Textketten ermöglicht. Texte sind dabei in Hochkomma einzuschließen. Mehrere Ausgabeparaneter innerhalb einer 'PRINT'-Anweisung sind mit Komma voneinander zu trennen. Zahlen werden bei fehlender Formatangabe sechsstellig mit unterdrückten Vornullen rechtsbündig ausgegeben (d. h.: eine einstellige Ziffer beansprucht sechs Zeichenpositionen auf den Bildschirm, wobei die ersten 5 leer bleiben und die auszugebende Ziffer die sechste Position einnimmt).

Durch ein Doppelkreuz, gefolgt von einer Zahl 1...6), kann diese Formatisierung geändert werden. Die Zahl gibt die maximal auszugebende Stellenzahl an. Die Formatisierung bleibt bis zur nächsten 'PRINT'-Ausgabe bestehen. Wird die 'PRINT' Anweisung mit einem Komma beendet, so beginnt die Ausgabe der nächsten 'PRINT'-Anweisung in der gleichen Zeile nach der vorangegangenen Ausgabe.

```
>10  X=5;Y=50;Z=500
>20  PRINT 'ZAHL X=',X
>30  PRINT 'ZAHL X=',#2,X
>40  PRINT X,Y,
>50  PRINT Z
>RUN
ZAHL X=5
ZAHL X=5
     5  50  500
```

STOP

Diese Anweisung beendet die Abarbeitung des BASIC-Programmes. Der Interpreter kehrt in die Eingabeschleife zurück. Eine 'STOP'-Anweisung muss nicht unbedingt als letzte Anweisung eines BASIC-Programmes stehen. Gegebenenfalls kann diese Anweisung bei der Abarbeitung durch bedingte Sprünge ('IF...GOTO') oder Unterprogrammrufe übersprungen werden, ehe sie dann im weiteren Programmlauf erreicht wird.

CALL

Mit 'CALL' und einer nachfolgenden, als Hexadezimalausdruck gekennzeichneten MC- Adresse wird ein in Maschinensprache geschriebenes Unterprogramm vom BASIC- Interpreter aus gestartet. Soll

nach Abarbeitung des MC-Unterprogrammes die Arbeit des Interpreters mit der folgenden BASIC-Anweisung fortgesetzt werden, so ist das MC-Unterprogramm mit einen 'RETURN'-Befehl (bedingt oder unbedingt, z. b. 0C9H, siehe auch Abschnitt 4, Unterprogrammtechnik) abzuschließen. Zur Übermittlung der Parameter werden die PEEK- und POKE-Befehle verwendet.

Beispiel:

```
>200 CALL (HEX (3000))
...
MC-Unterprogramne:
3000 3A FF 31  LD A, (31FFH)
3003 3C        INC  A
3004 27        DAA
3005 32 FF 31  LD (31FFH), A
3008 C9
```

Das durch die CALL-Anweisung in Zeile 200 aufgerufene MC-Unterprogramm (von Adresse 3000H bis 3009H) zählt den Inhalt des Speicherplatzes 31FFH dezimal um '1' weiter. Auf diesem Speicherplatz könnte dann mit einer PEEK-Anweisung zugegriffen werden.

Mit diesem Befehl können also auch die Routinen des Monitors aufgerufen werden. Das wird z. B. im Anwendungsprogramm 'Telefonverzeichnis' beim Retten von Dateien praktiziert.

PEEK

Die PEEK-Anweisung ermöglicht den direkten Speicherzugriff zum RAM bzw. ROM des Rechners. Die Anweisung enthält eine Adresse als Parameter. Vom Speicherplatz, der durch die Adresse ausgewählt wurde, wird ein Byte als Wert einer Variablen dezimal zugewiesen.

```
>10  X=PöK (1023)
Speicherplatz 1023 wird adressiert (dezimale Adresse)
>20  X=PöK (HEX (3FF))
Speicherplatz 1023 wird adressiert (hexadezimale Adresse)
```

POKE

Mit Hilfe von POKE kann ein Speicherplatz beschrieben werden (wobei der Speicherplatz im RAM-Bereich liegen muss). Der erste Parameter bestimmt die Adresse des Speicherplatzes, der zweite gibt den Datenwert an, der abgespeichert werden soll.

```
>10  POKE HEX (ED08),65
```

Der Wert 65 (dezimal) wird in den Speicherplatz mit der Adresse ED08H (BWS) als 41H abgespeichert. 41H entspricht dem ASCII-Kode für den Großbuchstaben A.

```
>10  FOR I=0 TO HEX (3FF)
>20  POKE HEX (3000) + I, PEEK (HEX (F000) + I) ; NEXT I
ODER
>10  A=HEX (3000); B=HEX (F000)
>20  FOR I=0 TO 1023; POKE A + I, PEEK (B + I); NEXT I
```

Es wird der Monitor ab der Adresse F000 in der Länge von 1K Byte nach Adresse 3000H umgespeichert.

Dieses Programm entspricht dem Monitorkommando: T F000 3000 3FF. Man beachte die unterschiedlichen Ausführungszeiten.

BYTE

Damit wird der Wert des nachfolgenden Ausdrucks als Hexadezimalausdruck auf dem Bildschirm ausgegeben. Es erfolgt nur die Ausgabe von dezimalen Werten bis 255 als zweistellige Hexadezimal zahl.

>BYTE (16)

```
  10
```

WORD

Diese Anweisung wirkt ähnlich wie Byte. Es werden aber hier 4 Stellen hexadezimal ausgegeben.

```
>10 N=1023
>20 WORD (N)
>RUN
  03FF
```

HEX

Mittels der HEX-Anweisung wird eine angegebene Hexadezimalzahl in eine Dezimalzahl umgewandelt.

```
>10  X=HEX (1000)
>20  PRINT X
>RUN
  4096
```

' ' (QUOTE)

Der durch ' '(Hochkomma) dargestellte Befehl QUOTE realisiert die Einzelzeichenumwandlung eines ASCII-Zeichens in einen Dezimalwert. Das ASCII- Zeichen ist zwischen dem Hochkomma anzugeben.

```
>10  X='B'
>20  PRINT X
>RUN
    66
```

OUTCHAR

Der der OUTCHAR-Anweisung folgende dezimale Ausdruck wird als entsprechendes ASCII-Zeichen auf dem Bildschirm ausgegeben. Bestimmte Sonderzeichen werden sofort ausgeführt (siehe OUTCH-Funktion des Monitors).

```
>10 OUTCHAR 65
>20 X=66
```

```
>30 OUTCHAR X
>40 X='C'
>50 OUTCHAR X
>60 PRINT X
>RUN
ABC
    67
```

OUTCHAR 12 löscht z. B. den gesamten Bildschirm.

INCHAR

Die INCHAR-Anweisung ermöglicht die Eingabe eines einzelnen Zeichens mittels der Tastatur. Bei der Eingabe dieses Zeichens wird es nicht auf dem Bildschirm ausgegeben. Die Enter-Taste muss nach dar Zeicheneingabe nicht betätigt werden. Der Wert des ASCII-Zeichens wird der in der Anweisung mit angegebenen Variablen zugewiesen.

```
>10 PRINT INCHAR; GOTO 10
oder
>10 X=INCHAR; PRINT X; GOTO 10
```

Mit dieser Programmzeile kann die gesamte Tastatur getestet werden. Es werden alle Tasten mit allen Shift-Tasten kombiniert betätigt und dadurch der dezimale Zahlenwert auf dem Bildschirm angezeigt. Ein Verlassen dieser Programmschleife ist nur mit S4/K (STOP) möglich.

OUT

Der in der OUT-Anweisung angegebene Wert des Ausdruckes wird an die in der Anweisung zugewiesene E/A-Adresse des MRB Z1013 ausgegeben. Der Wert darf 255 nicht überschreiten (255 ist bekanntlich die größte Dezimalzahl, die mit 8 Bit darstellbar ist.).

```
>10 OUT (0)=10
```

Das Bitmuster für 10 (00001010B) wird an die E/A-Adresse 0 ausgegeben. In der Grundvariante des MRB Zl013 ist 0H die mögliche E/A-Adresse des PIO-Port's A, die die freie Verwendung durch den Anwender ermöglicht.

Bei Verwendung der PIO als Port darf die entsprechende Initialisierung nicht vergessen werden, z. B.:

```
>10 OUT (1)=HEX (CF)
>20 OUT (1)=0   (Ausgabe)
bzw.
>20 OUT (1)=255 (Eingabe)
```

IN

Diese Anweisung ermöglicht die Eingabe von Werten von einer E/A-Adresse des MRB. (Adresse der Grundvariante ist 0H.) Der Wert, der an der E/A-Adresse anliegt, wird einer Variablen zugeordnet.

```
>10 X=IN (0)
```

I$

```
>10 I$ (TOP)
```

Eingabe einer Zeichenkette über Tastatur auf den ersten freien Speicherplatz nach einem BASIC-Programm.

O$

```
>20 O$ (TOP)
```

Ausgabe einer Zeichenkette, die ab dem ersten freien Speicherplatz nach dem BASIC-Programm gespeichert ist, auf dem Bildschirm.

LEN

Stellt die Länge der zuletzt mit einer I$-Anweisung eingegebenen Zeichenkette zur Verfügung.

```
>10  I$ (TOP)
>20  FOR I=0 TO LEN
>30  IF PEEK (TOP + I) = 'A' PRINT (TOP + I)
>40  NEXT I
```

Folgende Funktionen werden außerdem durch den BASIC-Interpreter realisiert:

RND

Die RND-Funktion weist einer Variablen einen zufälligen Wert zwischen 1 und dem in der Anweisung festgelegten Endwert zu.

```
>10  X=RND (2000)
>20  PRINT X
>RUN
  1576
```

ABS

Es wird der Absolutbetrag des folgenden Ausdrucks gebildet und einer Variablen zugewiesen.

```
>10  A=-120
>20  A=ABS (A)
>30  PRINT A
>RUN
  120
```

TAB

Die TAB-Funktion stellt eine sogenannte Tabulatoranweisung dar. Die sich aus dem nachfolgenden Ausdruck ergebende Anzahl von Leerzeichen wird auf dem Bildschirm ausgegeben. Die nachfolgende Ausgabe von Zeichen beginnt nach diesen Leerzeichen.

```
 >10  PRINT 'ANWEISUNG:',
 >20  X=5
 >30  TAB (X)
 >40  PRINT 'TAB',
 >50  TAB (6)
 >60  PRINT '(',#1,'X,')'

 ANWEISUNG  TAB    (5)
```

Durch die TAB-Punktion wird die Darstellung von Kurven möglich.

```
>10  FOR I=-5 TO 5; TAB (I*I); PRINT '*'; NEXT I
```

TOP

Die TOP-Funktion ermittelt den zum Zeitpunkt der TOP-Funktion aktuellen ersten freien Speicherplatz hinter dem soeben eingegebenen BASIC-Programm (dezimal).

```
>PRINT TOP
  3561
```

SIZE

Damit wird der aktuelle freie RAM-Speicherbereich ermittelt, der für ein BASIC-Programm noch zur Verfügung steht.

```
>10 PRINT SIZE, 'FREIE BYTE'
>RUN
  260 FREIE BYTE
```

Die ermittelte Anzahl freier Speicherplätze wird SIZE zugewiesen und kann im BASIC-Programm weiterverwendet werden.

5.4. Hinweise für die Erarbeitung Anwenderprogrammen

5.4.1. Allgemeine Hinweise

In diesem Abschnitt soll dem Ungeübten die Möglichkeit gegeben werden, sich anhand von einfachen Beispielen mit grundsätzlichen Problemen der Programmierung zu befassen. Bevor wir beginnen unserem Mikrorechner eine Leistung abzuverlangen, müssen wir uns darüber im klaren sein, dass er unsere Probleme erst dann lösen hilft, wenn wir ihm eine eindeutige Rechenvorschrift in Form eines Maschinenkode- oder BASIC-Programm geliefert haben.

Natürlich können wir uns diese Denkarbeit ersparen und den Rechner mit fertigen Programmen „füttern“, die sich ein anderer für uns erdacht hat. Das ist bei der Lösung von häufig wiederkehrenden Problemen zweckmäßig. Hier sei als Beispiel nur das mitgelieferte BASIC-Interpreter-Programm genannt, in dem eine Fülle geistiger Arbeit steckt und das einem breiten Anwenderkreis eine vereinfachte Programmierung auf der Basis von einheitlichen BASIC-Befehlen ermöglicht. Wenn es aber gilt, eigene spezifische Probleme rechentechnisch zu lösen, kommen wir um die Aufbereitung

nicht herum. Diese notwendige Vorarbeit kann man im wesentlichen in drei Schritte einteilen:

1. Problemanalyse
2. Erarbeitung der Rechenanweisung (Algorithmus)
3. Programmierung

5.4.2. Problemanalyse

Um eine gestellte Aufgabe mit Hilfe unseres Mikrorechners lösen zu können, gibt es ider Regel eine Fülle von Möglichkeiten. Bei der Präzisierung unseres Problems und der Anpassung an die rechentechnischen Gegebenheiten müssen folgende Gesichtspunkte in Einklang gebracht werden:

- Speicherplatzbedarf
- Verarbeitungszeit
- übersichtlichkeit
- Art der Datenein- und -ausgabe
- Bedienungskomfort.

Im folgenden soll das anhand eines Beispiels verdeutlicht werden:

Wir wollen den Inhalt unseres 2K-Zeichengenerators auf dem Bildschirm übersichtlich darstellen. Die grafische Darstellung soll so erfolgen, dass der Kode aller Zeichen ersichtlich ist (siehe auch Anlage 7). Da für die Darstellung eines Zeichens 8 Byte notwendig sind (8×8 Bildpunkte), können prinzipiell 2048/8 - 256 verschiedene Zeichen dargestellt werden.

Das Bildschirmformat unseres Rechners bietet uns die Möglichkeit, 32 Zeilen mit jeweils 32 Zeichen darzustellen, d. h. ein Bild kann aus max. 1024 Zeichen zusammengesetzt werden (siehe Anlage 8). Wir könnten nun mit einem einfachen, kurzen Programm den Zeichengeneratorinhalt mit ansteigender Kode-Zahl „hintereinanderweg“ abbilden, das ergibt 256/32 - 8 Zeilen. Darunter würde aber die Übersichtlichkeit leiden, da in dieser gedrängten Darstellung insbesondere die Grafikzeichen, die das 8×8-Bildpunkt-Format ausnutzen, Zum Teil ohne Übergang ineinanderfließen. Da wir mit dieser Methode auch nur ein Viertel des Bildschirmes ausnutzen, bietet es sich an, die Zeichendarstellung durch das Einfügen von Leerzeichen und Leerzeilen aufzuspreizen. Wir erhalten damit 16 beschriebene Zeilen mit jeweils 16 Zeichen. Diese Darstellungsart deckt sich auch gut mit der hexadezimalen Kodierung: In der ersten Zeile stehen die Zeichen 0H.. FH, in der zweiten (beschriebenen) Zeile die Zeichen 10h…1FH usw., so dass wir auf eine Beschriftung der Zeichen, für die ohnehin der Raum nicht ausreichen würde, verzichten können.

Auf Probleme der Verarbeitungszeit werden wir bei der Programmierung noch zusprechen kommen. Prinzipiell wäre es möglich, den Bedienungskomfort zu steigern, z. B. eine Anwahl bestimmter Zeichen oder Zeichengruppen über die Tastatur zu ermöglichen und parallel dazu den Zeichenkode sowohl hexadezimal als auch dezimal darzustellen. Das würde aber unseren Forderungen nach Einfachheit und Übersichtlichkeit (Erfassung aller Zeichen auf einen Blick) widersprechen.

5.4.3. Erarbeitung der Rechenanweisung (Algorithmus)

Nachdem wir unsere jeweilige Aufgabenstellung entsprechend den Empfehlungen des

vorhergehenden Abschnittes analysiert und präzisiert haben, messen wir für unseren Rechner eine exakte Rechenvorschrift aufstellen.

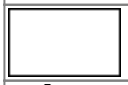
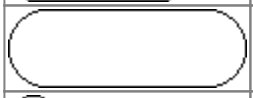
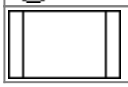
Für unser gewähltes Beispiel könnte man den Algorithmus folgendermaßen beschreiben:

1. Setze auf die Anfangsadresse des Bildschirms das Zeichen mit der Kode-Zahl 0!
2. Setze auf die nachfolgende Adresse ein Leerzeichen!
3. Wiederhole die beiden vorhergehenden Operationen mit wachsender Adresse und ansteigender Kode-Zahl, bis eine Bildschirmzeile voll ist, d. h. 16 mal!
4. Schreibe eine Leerzeile, d. h. setze auf die nächsten 32 Adressen jeweils 1 Leerzeichen!
5. Beschreibe auf die gleiche Art die folgenden 30 Zeilen!

Dieser mit Worten beschriebene Algorithmus stellt eine Liste dar, die von oben nach unten abgearbeitet wird. Dadurch wird es schwierig, sich wiederholende Programmabschnitte (Zyklen) und Programmverzweigungen (Sprünge) exakt zu beschreiben.

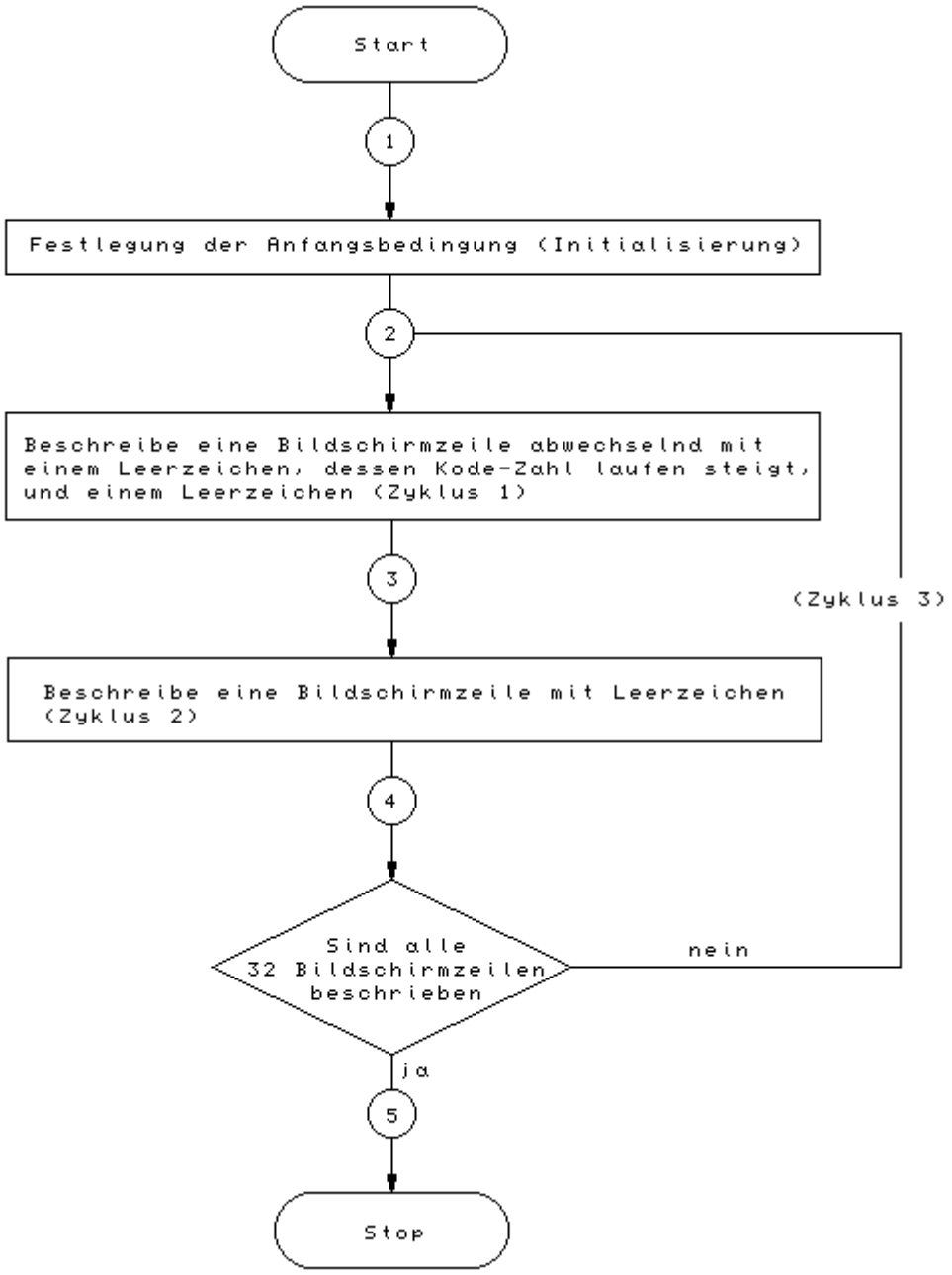
Hierfür erweist sich der Programmablaufplan (PAP) als ein günstiges Hilfsmittel bei der Erarbeitung eines Programms. Durch die Möglichkeit der zweidimensionalen Darstellung wer den zu lösende Probleme anschaulicher und überschaubarer.

Für die Gestaltung eines PAP werden folgende Elemente benötigt:

Bild	**Bedeutung**	
	Anweisung	einzelne oder mehrere Operationen (Folge)
	Verzweigung	Variation des Programmablaufes in Abhängigkeit einer oder mehrerer Bedingungen, dadurch Aufbau von Zyklen möglich
	Eingabe/Ausgabe	beliebige E/A-Operationen
	Grenzstelle	Anfang (Start), Unterbrechung (Zwischenstop) oder Ende (Stop) eines Programmes
	Verbindungsstelle	Verbindung von Programmteilen
	Unterprogramm	Programmteil, das an dieser Stelle aufgerufen wird

Um bei einem Programmentwurf durch die Vielfalt von Programmteilen und Gestaltungselementen nicht die Übersicht zu verlieren, empfiehlt es sich, schrittweise den Abstraktionsgrad zu verfeinern.

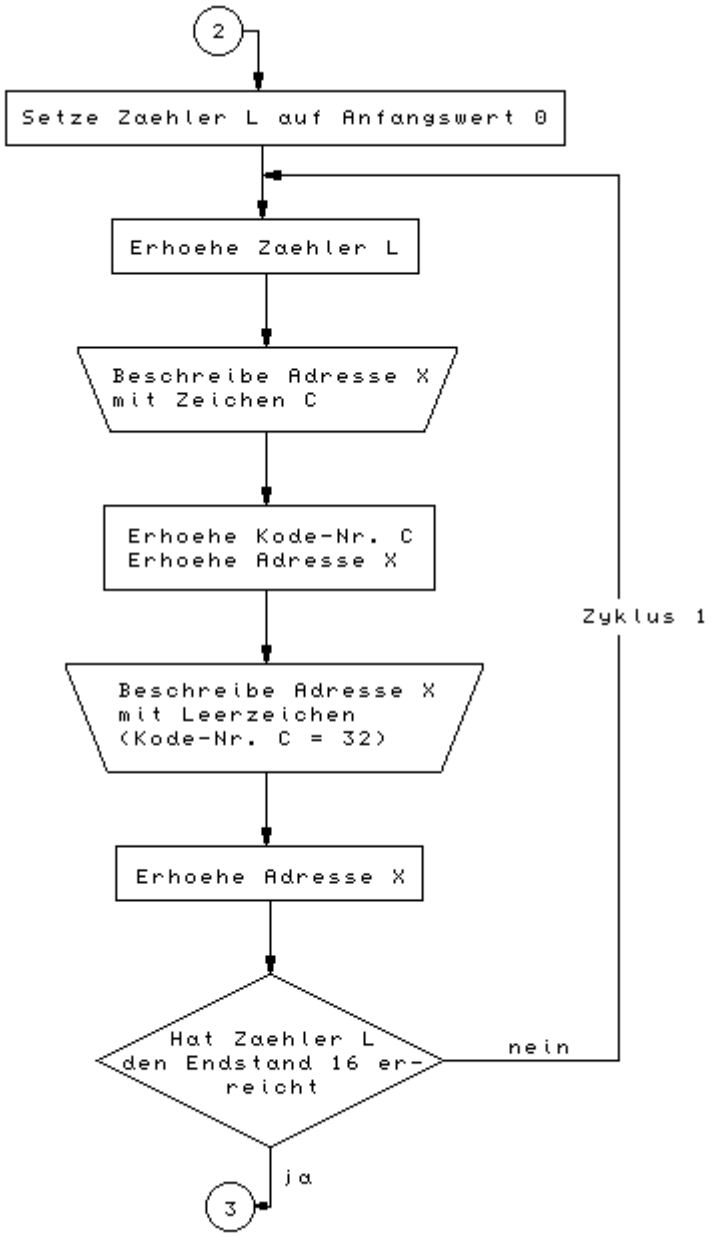
Diese Methode soll an unserem gewählten Beispiel demonstriert werden. Den vorhin angeführten verbalen Algorithmus könnte man in einer groben Abstraktion folgendermaßen als PAP darstellen:



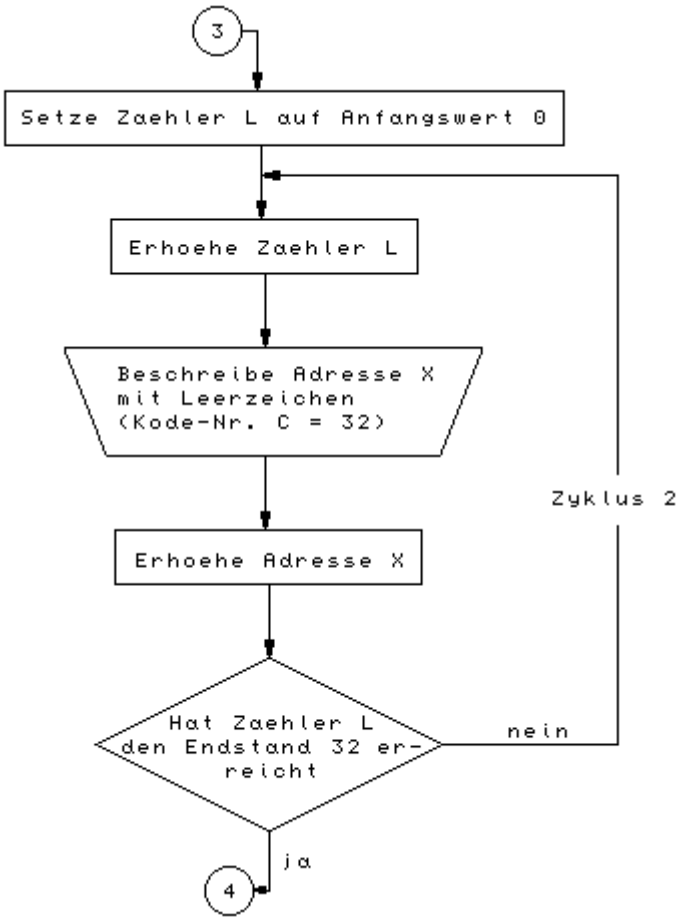
Dieser PAP enthält viele Vereinfachungen, die noch nicht geeignet sind für eine Übersetzung in die BASIC-Sprache. Insbesondere die Zyklen 1 und 2 sind soweit abstrahiert, dass ihr zyklischer Inhalt gar nicht erkennbar ist.

Wir wollen deshalb zunächst Zyklus 1 präzisieren. Um eine Bildschirmzeile entsprechend unseren Wünschen zu füllen, muss die Anweisungsfolge (Zeichen schreiben/Leerzeichen schreiben) mit wachsender Kode-Zahl C jeweils 16 mal wiederholt werden. Für die fortlaufende Adressierung des Bildschirms wird die Variable X benutzt, für die Anzahl der durchlaufenen Zyklen die Variable L.

Der Zyklus 1 stellt eine in sich abgeschlossene funktionelle Einheit dar, die auch als Modul bezeichnet wird. Hier die Darstellung von Zyklus 1 als Teil-PAP:

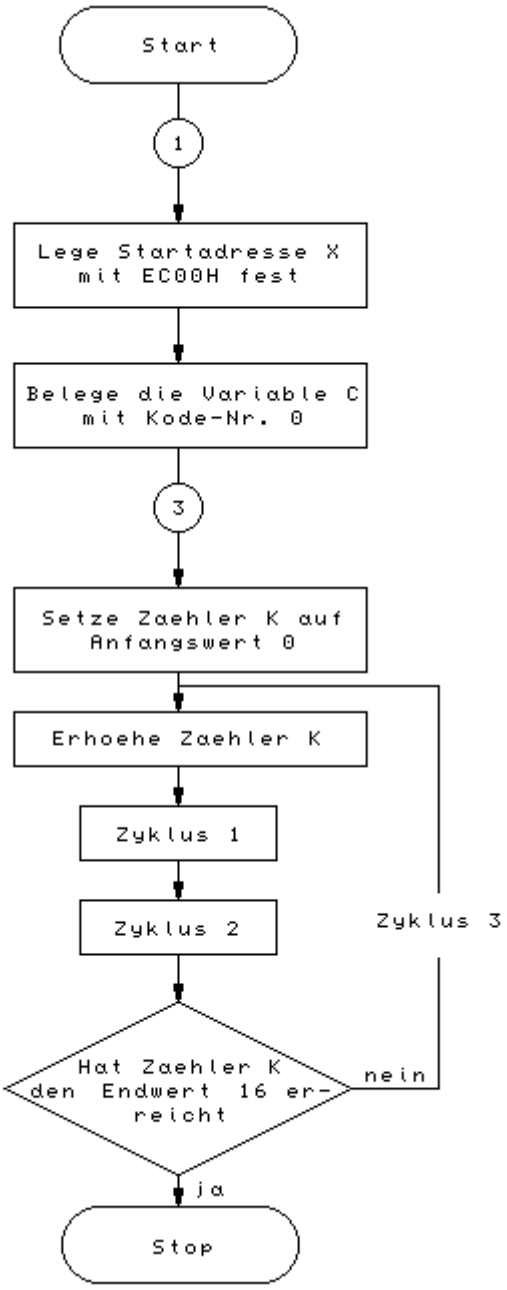


Zyklus 2 besteht aus der 32fachen zyklischen Wiederholung des Schreibens eines Leerzeichens bei ebenfalls fortlaufender Erhöhung der Adresse X und der Kode- Zahl C. Da der Zyklus 2 an den abgeschlossenen Zyklus 1 anschließt, kann die Variable L, nachdem ihr ein neuer Anfangswert zugeordnet wurde, erneut für das Zählen der Zyklen-Anzahl benutzt werden:



Mit der einmaligen Realisierung der Folge (Zyklus 1/Zyklus 2) haben wir erst das Programm für 2 Bildschirmzeilen abgearbeitet. Um das Gesamtprogramm zu realisieren, müssen wir die genannte Folge 16mal abarbeiten (Zyklus 3). Als Zähler verwenden wir jetzt die Variable K. Vorher sind noch die Anfangsbedingungen festzulegen. Als erste Bildschirmadresse wird entsprechend Anlage 8 die Adresse EC00H festgelegt, als erste Kode-Zahl wird C = 0 gewählt.

Der komplette PAP soll im folgenden aus Platzgründen unter Verwendung der bereits aufgeführten Programm-Module Zyklus 1 und Zyklus 2 dargestellt werden:



Damit ist unser PAP soweit aufbereitet, dass die Umsetzung in die Programmsprache erfolgen kann. Daraus resultiert auch die wichtige Schlussfolgerung, dass ein Programmablaufplan auf die verwendete Programmsprache und ihre Möglichkeiten „maßgeschneidert“ werden muss, d. h. die Erarbeitung des PAP setzt die gründliche Kenntnis der Elemente der beabsichtigten Programmiersprache voraus.

5.4.4. Programmierung

Bei der eigentlichen Programmierung können wir die Vorteile nutzen, die uns das BASIC durch seine Dialogfähigkeit bietet. Das bedeutet für uns, dass wir unser Programm schrittweise aufbauen und testen können.

Wir wollen das wieder an unserem Programmbeispiel üben. Sicherheitshalber werden bei diesem Beispiel noch einmal die wichtigsten Bedienungshinweise mit angegeben.

Nachdem wir den Anfangszustand hergestellt haben (in Abschn. 1.2.3. ausführlich beschrieben),

erwartet der Rechner, der sich jetzt im Betriebsprogramm befindet, eine Bedienereingabe. Es signalisiert das durch Ausgabe des Zeichens „#“ als Promptsymbol, gefolgt von einem Leerzeichen und dem Kursor „_“.

Gehen wir davon aus, dass wir unseren BASIC-Interpreter entsprechend den Hinweisen von Abschn. 5.3.3. auf Magnetband gespeichert vorliegen haben. Durch Eintippen der Anweisung

```
# L 100 BFF
```

geben wir dem Rechner zu verstehen, dass er in den Speicheradressraum von 100H bis BFFH Daten einlagern soll. Nach dem Positionieren des Magnetbandes (optimal ist hier ein Bandlängenzählwerk) und Einschalten der Wiedergabe erwarten wir den Kennton. Sobald er ertönt, aktivieren wir durch Drücken der ENTER-Taste unsere vorbereitete Bedienereingabe. Der Rechner liest den BASIC-Interpreter ein. Während des Einlesens befindet sich der Kursor am Zeilenanfang. Nach fehlerfreiem Einlesen des BASIC-Interpreters erscheint unter unserer Bedienereingabe wieder das Promptsymbol „#“.

Sollte das nicht auf Anhieb klappen, z. B. durch zu spätes Drücken der ENTER- Taste, erkennen wir das an der Bildschirmausschrift.

Es ist z. B. möglich, dass durch Staubkörnchen zwischen Band und Tonkopf oder durch Kontaktfehler der Übertragungsleitung einzelne „Bits“ verlorengehen. Das erkennt unser Rechner durch Kontrolle der Übereinstimmung der Summe aller Daten in einer Zeile mit den ebenfalls im Datensatz übermittelten Checksummen.

Durch Ausschrift z. B. von

```
CS < 0300
```

zeigt der Rechner uns an, dass im übertragenen Datenblock < Adresse 0300H ein Checksummenfehler aufgetreten ist. Es kann auch vorkommen, dass der Rechner „steckenbleibt“, d. h. der Kursor bleibt am Bildanfang stehen, es erscheint kein Promptsymbol. Da hilft uns nichts weiter, als nach Abstellen der Fehlerursachen und erneutem RESET von vorn anzufangen.

Prinzipiell ist es auch möglich, beim Auftreten von einzelnen Checksummenfehlern mit dem D-Kommando den angegebenen Datenblock durch Vergleich mit der BASIC- Interpreter-Liste nach Fehlern abzusuchen und diese mit dem N-Kommando zu korrigieren.

Im allgemeinen ist es aber weniger umständlich, das Einlesen neu zu starten, vorausgesetzt, das auf Band befindliche Maschinenkode-Programm ist fehlerfrei abgespeichert.

Haben wir diesen Schritt erfolgreich abgeschlossen, starten wir den BASIC- Interpreter ab der Anfangsadresse 0100H, indem wir eingeben:

```
J 100.
```

Nach Ausführen von ENTER meldet sich der BASIC-Interpreter, wie in 5.3.4. beschrieben und fordert mit dem neuen Promptsymbol „>“ eine Bedieneranweisung, jetzt in Form von gültigen BASIC-Kommandos oder -Programmzeilen.

Beginnen wir jetzt, unseren PAP in die BASIC-Sprache umzusetzen. Als erstes legen wir die Startadresse auf unserem Bildschirm fest. Sie ist uns als EC00H bekannt. Der Interpreter erwartet

aber von uns eine dezimale Adressenangabe. Wir können die umständliche Umrechnung dem Rechner überlassen, indem wir den HEX-Befehl benutzen. Unsere erste Programmzeile, der wir entsprechend der Vorschrift aus Abschn. 5.3.4. die Zeilennummer 10 zu ordnen, lautet:

```
10 X=HEX (EC00).
```

Erst durch Betätigen von ENTER wird die Zeile als Programmzeile abgespeichert. Vorher ist es noch möglich, eventuell notwendige Korrekturen durch Verschieben des Kursors mit „←-“ und „–>“, durch überschreiben mit anderen Zeichen oder durch Löschen mit der Leerzeichentaste „ „anzubringen. Wichtig ist es, dass nur die vor dem Kursor befindlichen Zeichen bei Abschluss mit ENTER in die Programmzeile übernommen werden.

Also:

Vor dem ENTER noch einmal auf den Bildschirm schauen!

Die nächste Anweisung aus dem PAP ergibt eine neue Programmzeile:

```
20 C=0
```

Damit wird der Variablen C die Zahl 0 zugeordnet. Das ist unsere Anfangsbedingung, die sichert, dass als erstes Zeichen das mit der Kode-Zahl 0 aus dem Zeichengenerator geholt wird.

Um anfänglichen Unsicherheiten bei fehlerhafter Bedienerführung von vornherein entgegenzutreten, ist an dieser Stelle noch einmal der Hinweis angebracht, die Abschnitte 5.3.4. und 5.3.5. besonders gründlich zu studieren und anzuwenden. Sollte z. B. durch mehrfache Fehlerausschriften, Änderungen usw. die Programmdarstellung auf dem Bildschirm an Übersicht verlieren, kann man jederzeit durch Eingabe des LIST-Kommandos den aktuellen Stand aller geordneten Programmzeilen erfahren.

Wenden wir uns nun zunächst der Programmierung von Zyklus 1 zu (vgl. PAP zu Zyklus 1).

Der Variablen L wird der Anfangswert 0 zugeordnet:

```
50 L=0
```

Da die Variable L als Zähler dient, muss sie in jeder Programmschleife um den Wert 1 erhöht werden:

```
60 L=L+1
```

Zum Beschreiben der Bildschirmdresse X mit dem Zeichen, Kode-Zahl C, wird der POKE-Befehl angewendet:

```
70 POKE X,C
```

Die Kode-Nr. und die Bildschirmadresse werden anschließend für die nächste Programmschleife um 1 erhöht:

```
 80 C=C+1
 90 X=X+1
```

Auf diese neue Adresse schreiben wir ein Leerzeichen (Kode-Nr. 32) und erhöhen anschließend wieder

die Adresse:

```
100 POKE X,32
110 X=X+1
```

Damit diese Befehlsfolge nach einmaligem Durchlauf beendet wird, verwenden wir den bedingten Sprung IF..GOTO Als Abbruchbedingung wird der Stand unseres Zählers L kontrolliert:

```
120 IF L<16 GOTO 60
```

Sind wir hier erfolgreich angelangt (zweckmäßig ist hier noch einmal die Kontrolle des bisherigen Programms mit LIST), können wir das Teilprogramm bereits durch Eingabe des Kommandos RUN starten und damit das Beschreiben der ersten Bildschirmzeile auslösen.

Sollten wir aber vorher bereits durch unser Programmieren die unterste Bildschirmzeile erreicht haben, wird durch das infolge der Fertigmeldung mit READY und Erscheinen des Promptsymbols um 2 Zeilen weiterrollende Bild unser „Programmiererfolg“ wieder verschwinden. Um das von vornherein zu verhindern, wenden wir einen Trick an, indem wir den Rechner „endlos“ beschäftigen. Dazu können wir z. B. eine Programmschleife konstruieren, die den Rechner zwingt, solange einen bedingten Sprung auszuführen, bis durch unseren Eingriff ein Abbruch erfolgt:

```
200 GOTO 200
```

Befindet sich der Rechner in einer solchen Schleife, reagiert er nicht mehr auf normale Eingabebefehle. Einen Abbruch erreichen wir Jetzt nur durch gleichzeitiges Betätigen von 54 und K. Dann erscheint auch wieder das Promptsymbol als Zeichen der Bereitschaft.

Wünschen wir über Bedienereingabe einen leeren Bildschirm, betätigen wir gleichzeitig S4 und T mit anschließendem ENTER.

Kommen wir nun zum Zyklus 2. Mit unseren Erfahrungen vom Zyklus 1 können wir ihn jetzt sofort umsetzen:

```
130 L=0
140 L=L+1
150 POKE X,HEX(20)
160 X=X+1
170 IF L<32 GOTO 140
```

Wer sich nicht damit abfinden kann, dass die zweite Zeile mit ihren 32 Leerzeichen „unsichtbar“ bleibt, kann z. B. anstelle der Leerzeichen das Zeichen „*“ (Kode-Nr. 2AH) verwenden:

```
150 POKE X,HEX(2A)
```

Dieser Befehl lässt sich später auf Wunsch wieder zurück wandeln. Durch LIST können wir uns überzeugen, dass der neue Programmteil sich entsprechend der Zeilennummerierung in das Gesamtprogramm eingefügt hat.

Bei erneutem Starten des bisherigen Programms durch RUN werden jetzt die ersten beiden Bildschirmzeilen überschrieben. (Nicht vergessen: vor dem Weiterarbeiten S4/K drücken!)

Um unser Gesamtprogramm zu verwirklichen, müssen wir noch den Zyklus 3 umsetzen:

```
30 K=0
40 K=K+1
180 IF K<16 GOTO 40
```

Wenn wir alles erfolgreich bewältigt haben, erscheint jetzt nach RUN der komplette Zeichengenerator in angemessenen Tempo auf dem Bildschirm. Damit ist unser kleines Programm fertig. Es soll aber gezeigt werden, dass es auch andere Möglichkeiten gibt, zum gleichen Ziel zu gelangen. Wir können z. B. vor dem Zyklus 1 den gesamten Bildschirm löschen. Damit entfällt der Zyklus 2.

Benutzen wir für das Bildschirmlöschen die entsprechende Monitorroutine, indem wir über den OUTCHAR-Befehl das Steuerzeichen 0CH = 12D abrufen (s.a. Abschnitt 5.1.1.), so haben wir den Vorteil, dass das Löschen in wesentlich kürzerer Zeit realisiert wird. Zum üben ist es zu empfehlen, hierfür den PAP abzuändern. Wenn wir zur Realisierung der Programmschleifen gleich noch die elegantere FOR…TO…NEXT-Anweisung benutzen, können wir durch Löschen und überschreiben einiger Programmzeilen folgende Variante erhalten:

```
 10 X=HEX (EC00)
 20 C=0
 30 OUTCHAR 12
 40 FOR K=1 TO 16
 60 FOR L=1 TO 16
 70 POKE X,C
 80 C=C+1
 90 X=X+2
120 NEXT L
140 X=X+32
180 NEXP K
200 GOTO 200
```

Zeile 30 bewirkt das schlagartige Löschen des gesamten Bildschirmes. Zeile 60..120 realisiert Zyklus 1, Zeile 40..180 stellt den Zyklus 3 dar, Zeile 90 rückt die Bildschirmadresse um jeweils 2 Plätze weiter, Zeile 140 bewirkt das überspringen einer Zeile.

Einen interessanten Aspekt bietet die Gegenüberstellung zu einem Maschinenprogramm, das nach demselben Algorithmus arbeitet wie unser Ausgangsprogramm. Es soll im folgenden komplett wiedergegeben werden (man beachte die Ausführungszeit im Vergleich zum BASIC-Programm):

```
3000  3E 00    LD A , 00
3002  21 00 30 LD HL, EC00
3005  06 10    LD B , 10
3007  C5       PUSH BC
3008  06 10    LD B , 10
300A  77       LD M , A
300B  23       INC HL
3000  36 20    LD M , 20
3003  23       INC HL
30Q?  30       INC A
3010  10 F8    DJNZ 300A - #
3012  06 20    LD B , 20
```

```
3014  36 20    LD B , 20
3016  23       INC HL
3017  10 FB    DJNZ 3014 - #
3019  Cl       POP BC
301A  10 EB    DJNZ 3007 - #
3010  76       HALT
```

6. Erweiterungen des MRB Z1013

6.1. Allgemeine Hinweise

Die Grundausbaustufe des MRB Z10l3 stellt das Kernstück eines Mikrorechnersystems dar, welches durch zusätzliche Baugruppen immer weiter komplettiert werden kann.

Diese Erweiterungsbaugruppen sollen ebenfalls industriell gefertigt und im Handel angeboten werden. Dazu gehört ein Baugruppenträger mit Rückverdrahtung, die außer den Bauelementen zur Verstärkung der Systemsignale eine Anzahl von Steckverbinderplätzen enthält.

Versierte Bastler, die neben der Beschäftigung mit dem MRB Z1013 auch Freude am Selbstbau elektronischer Baugruppen haben, werden sich ihre Erweiterungen vielleicht selbst bauen. Dieser Selbstbau ist aber nur erfahrenen Elektronikern anzuraten, da tiefgründiges Wissen und große Erfahrung vorausgesetzt werden. Weiterhin ist zu beachten, dass aus Kostengründen die Grundausbaustufe nur die unbedingt notwendigen Bauelemente enthält, die Systemsignale an den Steckverbinderanschlüssen meist unverstärkt und direkt von der CPU kommen und damit Fehler in Erweiterungsschaltungen die Grundausbaustufe zerstören können.

Deshalb sollte bei umfangreichen Erweiterungen im Selbstbau die Verstärkung und Entkopplung der Systemsignale sowie ein leistungsfähigeres Netzteil an erster Stelle stehen.

Aus den angeführten Gründen werden zu Hardwareerweiterungen nur einige grundsätzliche Hinweise gegeben, die für erfahrene Elektroniker ausreichen dürften. Auf die Wiedergabe von Schaltungen wird bewusst verzichtet.

ACHTUNG! Beachten Sie unbedingt den folgenden Hinweis. Es entfallen bei der Durchführung von Lötarbeiten auf der Z1013-Leiterplatte, ausgenommen das Anlöten der Tastaturkabel, im Garantiezeitraum alle Garantieansprüche.

6.2. Speichererweiterungen

Die Grundausbaustufe des MRB Z1013 besitzt standardmäßig einen 2K Byte ROM mit dem Monitorprogramm im Adressbereich F000H bis F7FFH und einen 1K Byte umfassenden statischen RAM im Bereich EC00H bis EFFFH als BWS. Zusätzlich ist er abhängig von der Bestückungsvariante mit 16K Byte dynamischen RAM (Z1013.01) oder mit 1K Byte statischen RAM (Z1013.12) ab Adresse 0000H ausgerüstet.

Die letztgenannte Variante erfordert für die Nutzung des BASIC-Interpreters einen größeren Arbeitsspeicher.

Achtung! Bei allen Erweiterungen ist zu beachten, dass die Belastbarkeit der Signalleitungen und der Stromversorgung nur für die Grundausbaustufe ausgelegt ist.

6.3. Anschluss von Steuereinheiten

Für kleine Steuer- und Regelungsaufgaben steht als E/A-Port eine 8 Bit- Schnittstelle des PIO's zur Verfügung (PIO-Port A). Dieser Peripherieanschluss hat die E/A-Grundadresse 0. Die erforderlichen Anschlüsse stehen am Steckverbinder X4 zur Verfügung (siehe Anlage 6). Je nach Umfang der zu steuernden Aufgabe kann das bereits ausreichend sein. Das Fort wird im Modus bitgesteuerte E/A betrieben und kann sowohl Eingabesignale entgegennehmen als auch den Prozess beeinflussende Signale abgeben.

Sollte bei gewachsenen Anforderungen dieser 8 Bit breite Fort nicht ausreichen, kann zusätzlich mit den Spaltenauswahlleitungen der Tastatur eine Auswahl der Signalquellen sowie der zu steuernden Einheiten vorgenommen werden. Es können damit 10 Jeweils 8 Bit breite E/A-Ports ausgewählt werden, ohne dass zusätzlicher Dekodieraufwand notwendig ist. Beispiel: Es wird eine Information auf dem PIO-Fort A ausgegeben. Die auszugebenden Daten werden durch den PIO am Steckverbinder X4 zur Verfügung gestellt. Durch die Ausgabe einer Spaltennummer zwischen 0 bis 9 auf die E/A-Adresse 8 wird die entsprechende Spaltenleitung aktiviert (Low-aktiv) und gewährleistet die Datenübernahme in das ausgewählte Port. Da durch den Spaltendekoder 10 Spaltenleitungen aktiviert werden können, ist der Anschluss von 10 verschiedenen E/A-Ports am Steckverbinder X4 möglich. Damit stehen einem Anwender ohne großen zusätzlichen Aufwand 80 Kommandoleitungen zur Verfügung, die nach eigenem Ermessen in Ein- und Ausgabeleitungen eingeteilt werden können.