ROMROUTINES & ENTRYPOINTS

KOMMNADOS

Mittels eines kurzen Programmes (abgedruckt unter NP 15) lassen sich die Kommandos des DAI-Rechners auflisten.

Die Liste enthält :

INTERPRETERCODE	die interne Darstellung des Kommandos, die im Textbuffer auftritt
ENTRYPOINT	die aktuelle Ansprungadresse, z.B. hat CALLM #DEB5 dieselbe Wirkung wie NEW
STATE	EXEC bedeutet Command nur im Programmmodus IMM bedeutet Command nicht im Programm-modus zulässig
COMPILIERROUTINE	Adresse des Unterprogrammes zur Übersetzung des Commands bei Eingabe in Memory Bank RST 1 (?)
LISTROUTINE	Adresse des Unterprogramms, das den Command listet in der Bank Ø

FUNKTIONEN

Die zweite Liste enthält die Function-Codes (intern wird dem Function-Code #2Ø vorangestellt; SIN wird also z.B. durch #2Ø,#22 dargestellt).
Bei den Entrypoints ist zu beachten :
alle Funktionen sind im Bereich EØØØ-EFFF, wenn bei Entrypoint eine Zahl 6xxx steht, so behandelt das Unterprogramm den Parameteraufruf selbst, wenn bei Entrypoint Exxx steht, so lädt der Interpreter das (FPT) Argument vor dem Aufruf in den FPT-ACCU.
Die Liste TYP/ARGTYP ist der vollständigkeit halber angegeben.

Autoren : Klaus Hoffmann
Helge Rebhahn

COMMAND	INTER PRETER CODE	ENTRY POINT	STATE	COMPILE ROUTINE	LISTING ROUTINE
NEW	81	DEB0	IMM	E369	ED3A
CONT	82	DED5	IMM	E369	ED3A
STOP	83	DF03	EXEC	E369	ED3A
END	84	DF0C	EXEC	E369	ED3A
RESTORE	85	E401		E369	ED3A
RETURN	86	DF4C	EXEC	E369	ED3A
RUN	87	DF9E	IMM	E295	ED3A
RUN Z	88	DFBA	IMM	E295	ED41
GOTO	89	DF63	EXEC	E36A	ED41
GOSUB	8A	DF2A	EXEC	E36A	ED41
LOAD	8B	D274		E355	ED4D
SAVE	8C	D23D	IMM	E355	ED4D
CHECK	8D	D2C3	IMM	E369	ED3A
OUT	8E	DFC9		E302	ED47
POKE	8F	DFC0		E302	ED47
WAIT	90	DFD5		E259	ED6B
WAIT MEM	91	DFF7		4157	ED6B
WAIT TIME	92	E016	IMM	CB	ED4D
LIST	93	E197		E228	ED3A
LIST Z	94	E1AA		E228	ED41
LIST Z-Z	95	E1B6		E228	ED7A
SOUND	96	E48C		E317	ED84
NOISE	97	E50C		E325	ED9B
ENVELOPE	98	E570		E1F6	EDA4
CURSOR	99	E5B2		E302	ED47
MODE	9A	E5BB		E1D1	EDC1
DOT	9B	E5C1		E28F	ED62
DRAW	9C	E5CE		E28C	ED5C
FILL	9D	E5D7		E28C	ED5C
COLORT	9E	E60E		E30B	ED50
COLORB	9F	E615		E30B	ED50
INPUT	A0	E302	EXEC	E115	EDE0
INPUT ""	A1	E2FC	EXEC	E115	EDD9
DATA	A2	E18F	EXEC	E8A2	ED44
READ	A3	E323	EXEC	E127	EDE0
LET	A4	E45A		E0FE	EDFF
=(LET)	A5	E45A		622A	EDFF
IF THEN	A6	DF20	EXEC	E0BC	EE09
IF GOTO	A7	DF15	EXEC	E0BC	EE1A
IF THEN Z	A8	DF15	EXEC	E0BC	EE25
REM	A9	E18F	EXEC	E366	ED44
FOR	AA	E02B		E05F	EE30
NEXT	AB	E0E5		E0A9	ED3A
NEXT V	AC	E0C5		E0A9	EE4F
PRINT	AD	E2B3		E19F	EE52
ON GOSUB	AE	DF6A	EXEC	E176	EE66
ON GOTO	AF	DF71	EXEC	E176	EE71
DIM	B0	E62F		E166	EDE0
***	B1	DA33	EXEC	E366	ED44
UT	B2	E69E	IMM	E369	ED3A
CALLM	B3	E6A4		E344	EE87
CLEAR	B4	E6B5		E314	ED4D
IMP	B5	E195		622A	ED3A
LIST	B6	E1F5		E228	ED3A
LIST Z	B7	E253		E228	ED41
LIST Z-Z	B8	E25C		E228	ED7A
SAVEA	B9	D81D		E4A9	D89E
LOADA	BA	D85E		E4A9	D89E
TALK	BB	CD64		E314	ED4D
STEP	BC	DEFE	IMM	E369	ED3A
TRON	BD	E6CE		E369	ED3A
TROFF	BE	E6D5		E369	ED3A

EOT

FW 3,1

FUNCTION	CODE	ENTRYPOINT	TYP/ARGTYP			
ABS	Ø	EA5Ø	FPT	FPT		
ALOG	1	EA53	FPT	FPT		
ASC	2	6ACB	INT	STR		
CHR$	3	6AD2	STR	INT		
CURX	4	6AB8	INT			
CURY	5	6ABE	INT			
EXP	6	EA56	FPT	FPT		
FRAC	7	EA59	FPT	FPT		
FRE	8	6B43	INT			
FREQ	9	EB5C	INT	FPT		
GETC	A	6B75	INT			
HEX$	B	6A83	STR	INT		
INP	C	6882	INT	INT		
INT	D	EB8D	FPT	FPT		
LEFT$	E	6AE2	STR	STR	INT	
LEN	F	6AC4	INT	STR		
VARPTR	1Ø	6BA1	INT			
LOG	11	EA5C	FPT	FPT		
LOGT	12	EA5F	FPT	FPT		
XMAX	13	6BA7	INT			
YMAX	14	6BAE	INT			
MID$	15	6BØE	STR	STR	INT	INT
PDL	16	6BC1	INT	INT		
PEEK	17	6C16	INT	INT		
PI	18	6C1D	FPT			
RIGHT$	19	6AFF	STR	STR	INT	
RND	1A	EC27	FPT	FPT		
SCRN	1B	6C9D	INT	INT	INT	
SGN	1C	EC7B	FPT	FPT		
SPC	1D	6A8C	STR	INT		
SQR	1E	EA62	FPT	FPT		
STR$	1F	EA77	STR	FPT		
TAB	2Ø	6AA2	STR	INT		
VAL	21	6B25	FPT	STR		
SIN	22	EA65	FPT	FPT		
COS	23	EA68	FPT	FPT		
TAN	24	EA6B	FPT	FPT		
ASIN	25	EA6E	FPT	FPT		
ACOS	26	EA71	FPT	FPT		
ATN	27	EA74	FPT	FPT		

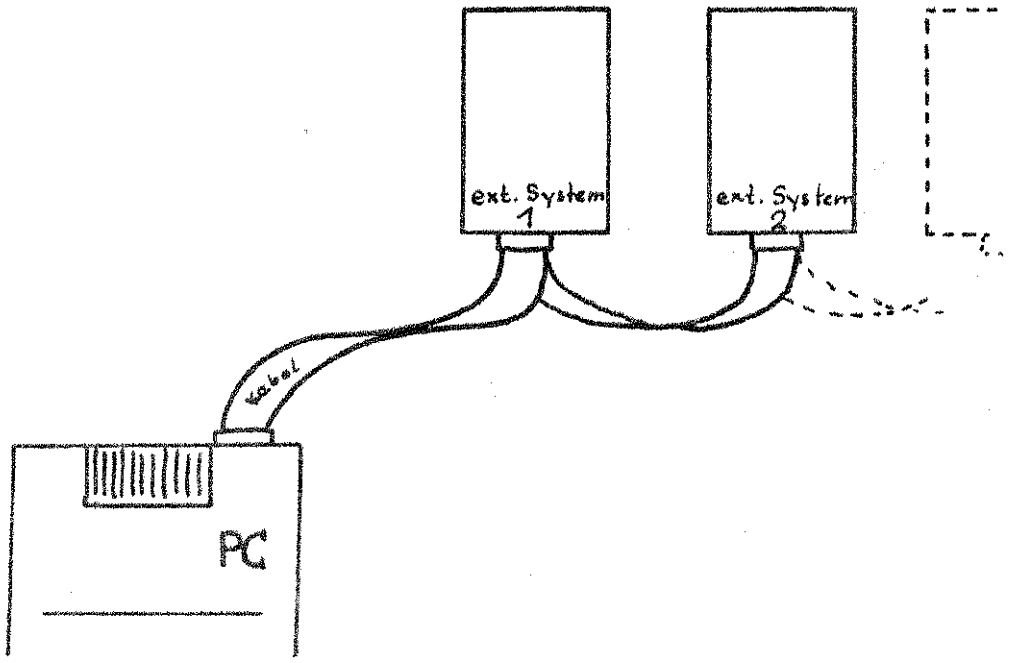
Allgemeine Beschreibung

des DCE-Bus-Systemes

Der DCE-Bus

Allgemeines:

Mit dem DCE-Bus-Stecker besteht fuer den DAI-PC-Anwender die Moeglichkeit, eine Vielzahl von Interface-Karten mit dem Personal-Computer zu verbinden. Bild 1 zeigt als Beispiel den Anschluss von 2 externen Systemen an den DAI-PC. Das DCE-Anschlusskabel verbindet alle ext. Systeme mit dem PC. Daher liegen Signale gleichzeitig an allen Karten an.



Fuer die einwandfreie Datenuebertragung von oder zu einem ext. System muss das ext. System selbstverstaendlich nach dem DCE-Bus-Format ausgelegt sein. Die folgende Beschreibung soll fuer das Arbeiten mit diesem Bus naehere Informationen liefern.

Aufteilung im DCE-BUS-SYSTEM

Datenleitungen:

Fuer die Daten werden wie im PC selbst 8 bit, also 8 Leitungen benoetigt. Ueber diese Leitungen kann der PC sowohl Daten senden,wie auch empfangen.

Es gilt folgende Pin-Belegung:

Datenleitung	DCE-Bus-Stecker	RWC-Karte
0	16	24
1	14	26
2	12	28
3	10	30
4	9	29
5	11	27
6	13	25
7	15	23

Diese Leitungen stellen den Informationspfad dar. Jede Information wird darueber geleitet (in beide Richtungen). Durch die festgelegte Breite von 8 bit koennen Daten nur folgendes Format besitzen:

Zahlen:0-255 Zeichen:8 bit-ASCII-Code Binaer:0-11111111

Umfangreichere bzw. groessere Zeichen/Zahlen erfordern eine Uebertragung in mehreren Einzelschritten.

Beispiel: Der Befehl OUT(x),15 wird auf den Datenleitungen 0-7 das Bitmuster 15= 00001111 *schreiben*.

!!!Wichtig: Alle Befehle aktivieren die Leitungen vom DCE-Bus nur fuer eine kurze Zeit, d.h. sie koennen nicht mit einem normalen Vielfachmessgeraet angezeigt werden.!!!

Adressleitungen:

Um mehrere ext.Systeme ansprechen zu koennen, wird fuer jede Datenuebertragung noch eine Adresse ausgegeben. Ein ext.System ist ueber Hardware (z.B. Schalter oder Loetbruecken) auf eine feste Adresse eingestellt. Jedes ext. System vergleicht die anstehende Adresse auf den Adressleitungen mit seiner eigenen und wird nur bei Uebereinstimmung mit der Festadresse aktiviert. Alle anderen ext.Systeme bleiben abgeschaltet.
Der DCE-Bus besitzt 8 Adressleitungen, die in 2 Gruppen zu je 4 bit organisiert sind. Die 4 hoeherwertigen Bit geben die Kartengrundadresse an, die 4 niederwertigen Bit sind fuer Unteradressen auf einem ext.System bestimmt. Diese Aufteilung ermoeglicht das Adressieren von 16 ext.Systemen (Kartengrundadresse). Jedes ext.System kann zusaetzlich bis zu 16 Unteradressen besitzen.

Beispiel: Ein 8-Kanal-Digital-Analog-Wandler mit der Kartengrundadresse 5: Jeder Wandlerkanal ueber die Unteradresse einzeln ansprechbar (0-7).

!!!Wichtig: Adressen werden nur vom PC ausgegeben.!!!

Beispiel: Der Befehl OUT(#34),x wird auf den Adressleitungen der Kartengrundadresse die Karte Nr.3 ansprechen und auf den Adressleitungen der Unteradresse die 4 ansprechen.

Fuer umfangreiche Systeme mit vielen ext.Systemen ist eine Adressierung mit z.B. 256 Grundadressen moeglich,im Rahmen dieses Artikels wird aber grundsaetzlich die Standartaufteilung angesprochen.

Es gilt folgende Pin-Belegung:

Kartengrundadresse	DCE-Bus-Stecker	RWC-Karte
0	24	9
1	23	11
2	22	13
3	21	15

Unteradresse	DCE-Bus-Stecker	RWC-Karte
0	30	12
1	31	10
2	32	8
3	**25**	**7**

Steuerleitungen

Neben Adressen und Daten werden vom DCE-Bus weitere 8 Leitungen fuer Steuersignale bereitgestellt. Der PC aktiviert nur 3 Leitungen, waehrend das Industriesystem (oder Assemblerprogrammierer) fuer schnelle bzw. komplexe Datenuebertragungen mehr Leitungen benutzen.
Um dem ext.System anzuzeigen,ob Daten gelesen oder geschrieben werden sollen, dienen die Steuersignale Read und Write. Will der Computer von einem ext.System lesen, so wird die Read-Leitung aktiviert (auf 0-Pegel gesetzt), bei einem Schreibvorgang dementsprechend die Write-Leitung (auf 0-Pegel).
Gleichzeitig mit der Adressinformation wird das Signal Bus-Expand (Buserweiterung) gesetzt (auf 1-Pegel). Dieses Signal muss zur vollstaendigen Decodierung der Adresse mit-verwendet werden. Dies ist natuerlich nur fuer Eigenentwicklungen relevant(Auf DAI-Produkten ist dies bereits implementiert)

Es gilt folgende Pin-Belegung:

Leitung	DCE-Bus-Stecker	RWC-Karte
Read-Leitung	28	16
Write-Leitung	27	17
Bus-Expand	26	18

!!!Wichtig: Steuersignale werden nur vom PC ausgegeben.!!!

Zusaetzliche Leitungen:

Um ext.Systeme beim Einschalten in einen definierten Zustand zu bringen,wird das Signal EXRESET bereitgestellt (wird auf 0-Pegel gesetzt).
Soll ein ext.System den Rechner in seiner allg. Arbeit unterbrechen, koennen die Signalleitungen EXINTR (externer Interrupt) oder IN7(Bit 7 des 5101 Multifunktionsbausteines) benutzt werden.
Fuer ext.Systeme,die nur einen geringen Stromverbrauch aufweisen, stehen die Stromversorgungsleitungen mit +5Volt, -5Volt,+12Volt und Masse zur Verfuegung.

Es gilt folgende Pin-Belegung:

Leitung	DCE-Bus-Stecker	RWC-Karte
EXRESET	7	5
IN7	5	3
EXINTR	6	4
+5Volt	1	31
+12Volt	2	2
-5Volt	3	1
Masse	4	6

BASIC und der DCE-Bus

Befehle:

Fuer das Arbeiten mit ext.Systemen existieren im BASIC-Befehlssatz des PC's 2 Grundbefehle (der Befehl WAIT STATUS wird hier nicht beruecksichtigt).
Um Daten zu einem ext.System zu senden,gilt der Befehl

```
OUT(I),X
```

Um Daten von einem ext.System zu lesen der Befehl

```
X=INP(I)
```

Output:

Der Befehl OUT(I),X sendet die Daten X (z.B. ein ASCII-Zeichen oder eine Variable) zu dem ext.System mit der Adresse I. Die Daten sind wie ber. erklaert,8 bit breit, koennen also nur Werte bis 255 annehmen. Die Adresse I besteht aus 2 Adressteilen (siehe auch Kapitel Adressleitungen), der Kartengrundadresse und der Unteradresse (jeweils 4 bit). Besonders vorteilhaft erweist sich bei Adressen die hexadezimale Schreibweise. Um z.B. Karte Nr.3 und davon die Unteradresse 7 anzusprechen,ergibt sich folgende Schreibweise:

```
OUT(55),X    bei dez.Adresse
OUT(#37),X    "  hex.    "
```

Im 2.Fall gibt die erste Ziffer (3) die Kartengrundadresse, die zweite Ziffer (7) die Unteradresse an.

Input:

Der Befehl X=INP(I) weist der Variablen X die Daten zu, die ueber den DCE-Bus von dem ext.System mit der Adresse I gelesen wurden. Fuer Daten und Adressen gelten die in Kapitel Output erklaerten Beziehungen:
Die Variable X kann nur Werte bis 255 erhalten;Die Adresse (I) besteht aus zwei Teilen;die hex.Schreibweise ist vorteilhaft.

Zusammenfassung:

Der PC ist grundsaetzlich der bestimmende Teil in einem kompl.System. Er bestimmt,wann/wie Daten wohin/woher geleitet werden. Wie ein ext.System diese Daten interpretiert, ist voellig offen. Somit stellt der DCE-Bus ausschliesslich einen Datenkanal dar, ueber den beliebige Informationen laufen. Der DCE-Bus ist ein in sich geschlossenes System, das nicht mit dem Mikroprozessor-Bus, oder einem, auf einem ext.System verwendeten internen Bus(RWC-RTC) verwechselt werden darf.

Beispiel:

```
10 X=INP(#31)
20 OUT(#7A),X
```

Dieses kleine Programm liest von dem ext.System mit der Kartengrundadresse 3 und der Unteradresse 1 die Daten in die Variable X und schreibt sie danach an das ext.System mit der Kartengrundadresse 7 und der Unteradresse A.

Zu Zeile 10:

1. Adresse wird ausgegeben (mit BUS-EXPAND)
2. Signal Read wird aktiviert
3. Das ext.System liest die Daten auf den Datenbus
4. Variable X bekommt die Daten vom Datenbus zugewiesen

Zu Zeile 20:

1. Adresse wird ausgegeben (mit BUS-EXPAND)
2. Die Daten der Variablen X werden auf den Datenbus gelegt
3. Signal Write wird aktiviert
4. Das ext.System nimmt die Daten vom Datenbus auf

Beschreibung einer typischen RWC-Karte

Allgemeines:

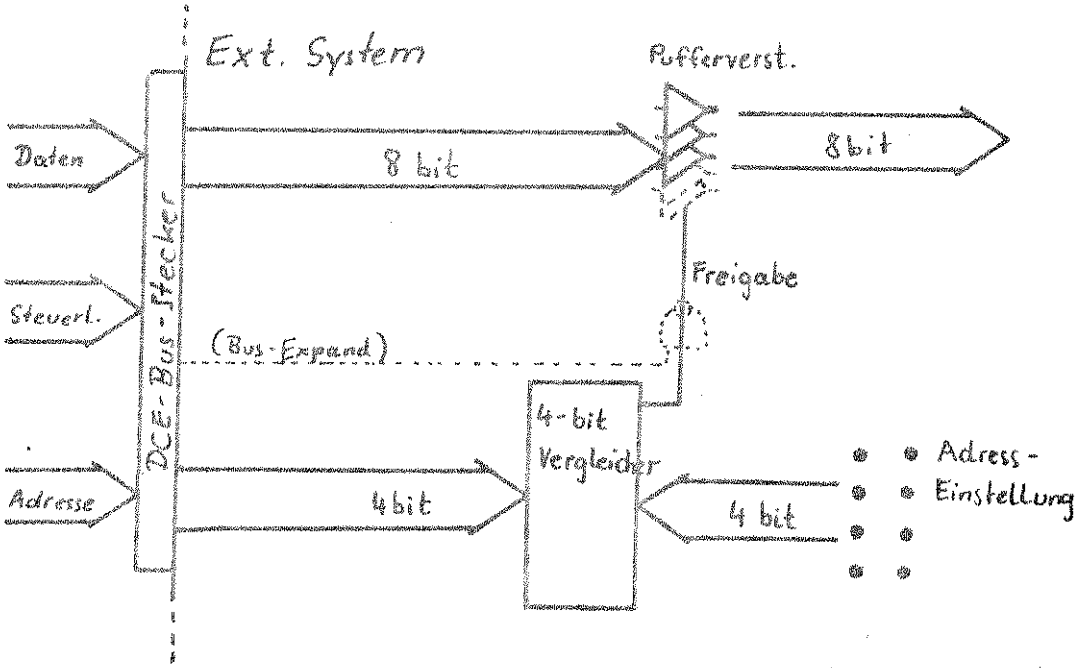
Betrachtet man einen Aufbau aus einem PC und mehreren ext. Systemen, so findet man 3 verschiedene Bus-Systeme,die miteinander korrespondieren.
Im PC ist es der Mikroprozessor-Bus mit seinen 8 Datenleitungen, den 16 Adressleitungen und den Steuerleitungen.
Im ext.System ist ebenfalls ein Bus-System aufgebaut, das fuerjede Karte allerdings anders aufgebaut sein kann (einfache ext.Systemen auch ohne eigenes Bus-System).
Beide Bus-Systeme tauschen ihre Daten ueber den DCE-Bus aus.

Die folgende Ausfuehrung soll den Aufbau einer typischen RWC-Karte (ext.System) naeher beschreiben.

Aufbau

Besitzt ein ext.System nur einfache Funktionen wie z.B. ein Druckerinterface, so ist nur wenig Schaltungsaufwand noetig. Grundsaetzlich vorhanden ist der Adressdecodierer, der die vom PC ausgegebene Adresse mit seiner fest eingestellten Adresse vergleicht. Um den DCE-Bus zu sichern,sind ueblicherweise die Datenleitungen ueber Pufferstufen entkoppelt.

Bild 2
Einfaches ext.System



Fuer komplexere ext.Systeme, die zusaetzlich mehrere Steuersignale benoetigen, wird ein Ein/Ausgabebaustein des Typ's 8255 von Intel verwendet. Die vielseitigen Moeglichkeiten dieses Bausteines entnehmen Sie bitte dem Datenblatt.
Der 8255 erlaubt es, dass Daten,die ueber den DCE-Bus (Datenleitungen) kommen, auf 3 verschiedene 8-bit-Ports seleitet werden koennen. Auch das Lesen dieser 3 Ports ist jederzeit durchfuehrbar. Diese 3 Ports werden allgemein mit A, B und C bezeichnet. Ob diese 3 Ports als Eingang oder als Ausgang bzw. sogar gemischt betrieben werden, ist ueber das sog. Command-Register des 8255 selektierbar.
Port A hat dabei die Unteradresse 0, Port B die Unteradresse 1, Port C die Unteradresse 2 und das Commandregister die Unteradresse 3.

Aus der Sicht des PC's (ueber den DCE-Bus) besteht das ext. System somit aus einer Kartengrundadresse (frei waehlbar) und 4 Unteradressen (3*Ports und 1*Command-register).

In dem ext. System stehen die 3 Ports frei zur Verfuegung. Der 8255 ist also ein Weichen-Baustein, ueber den die Informationen vom DCE-Bus (Datenleitungen) zu oder von 3 Aus/Eingaengen geschaltet werden.

Bild 3
8255 Uebersicht

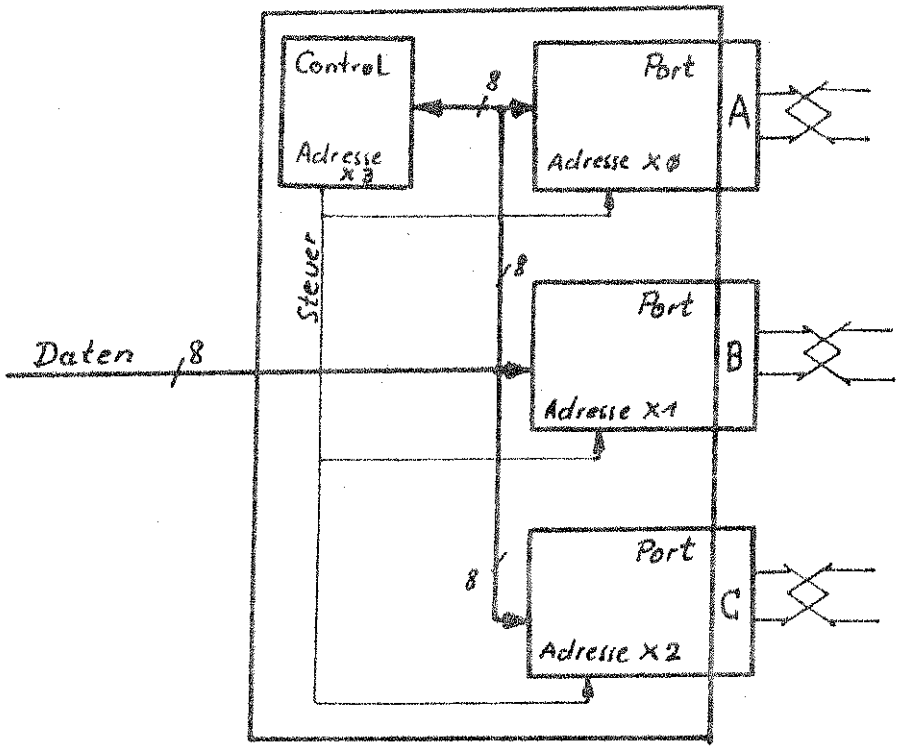
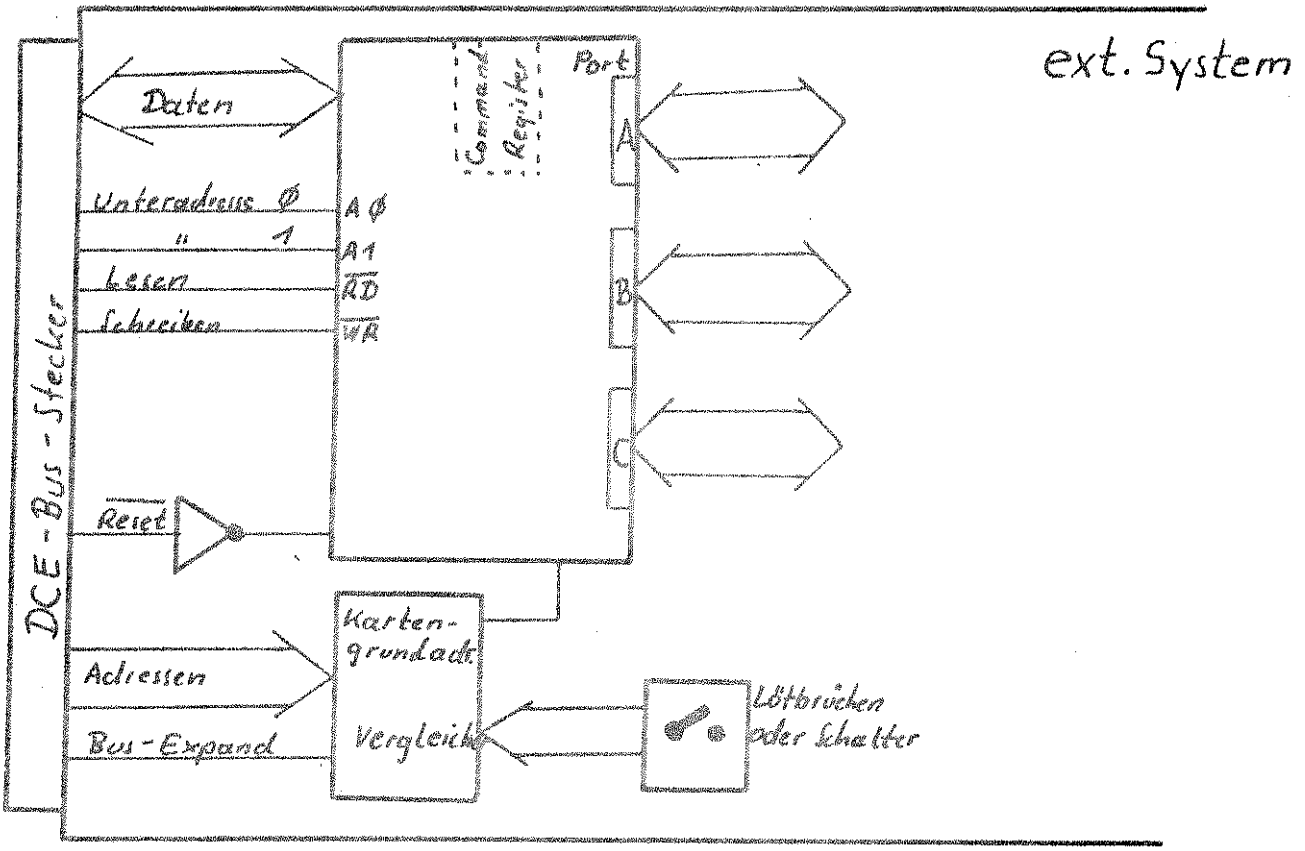


Bild 4
Eingang einer RWC-Karte mit 8255

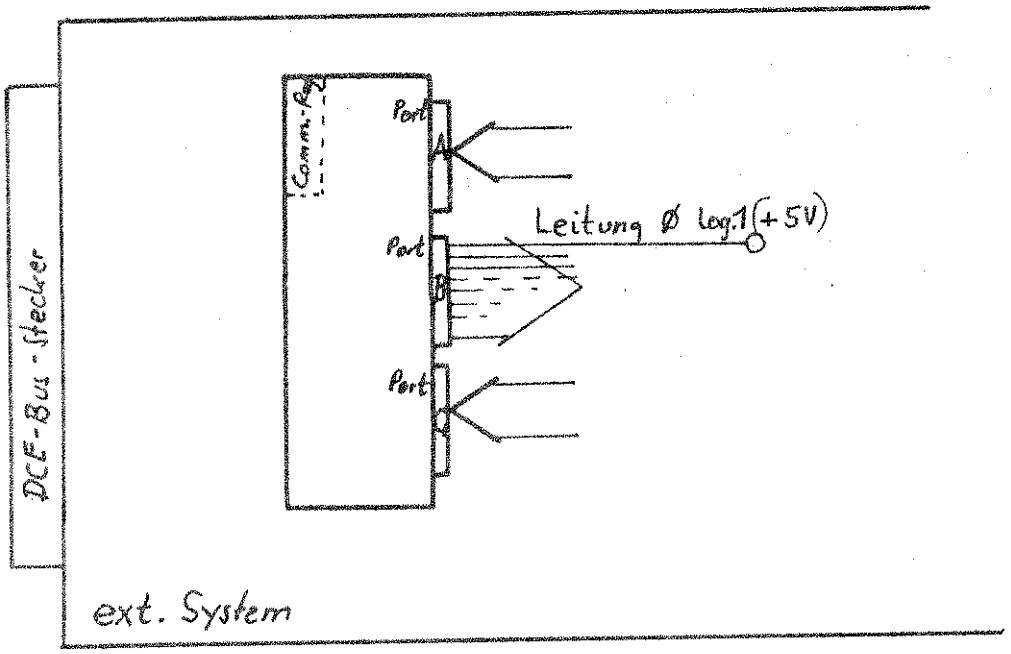


Wie kann nun der PC auf einem ext. System die einzelnen Leitungen ansprechen?
Die folgenden Beispiele sollen dies verdeutlichen.

Beispiel 1:

Aufgabe: Ein ext. System mit der Kartengrundsdresse 3 soll auf Leitung Ø des B Port's High-Potential bekommen. Port A und C sollen als Eingänge geschaltet sein.

Bild 5
Bit Ø von Port B=log. 1 (High)



Ausfuehrung: Zuerst muss der 8255 der RWC=Karte ueber das Command-Register die 3 Ports in die gewuenschte Form bringen (siehe Datenblatt).

```
10 OUT(#33),#82 ------------Command-Word fuer A u. C Eingang
        !!-------------------Unteradresse fuer Command-Regist.
        !--------------------Kartengrundadresse
```

Jetzt kann an Port B die Leitung 0 aktiviert werden.

```
20 OUT(#31),#1--------------Daten fuer Port
        !!-------------------Unteradresse fuer Port B
        !--------------------Kartengrundadresse
```

Beispiel 2:

Aufgabe: Auf ein ext.System mit der Kartengrundadresse 7 soll an Port A die Variable M ausgegeben werden, danach ebenfalls von Port A die inzwischen verarbeiteten Daten wieder gelesen (in Variable MNEU).

Ausfuehrung: Die 3 Ports der RWC-Karte werden in den gew. Mode gebracht:

```
10 OUT(#73),#8B---------Comm.-Word fuer A=Output
        !!                        B u. C=Input
        !!--------------Unteradresse fuer Command-Register
        !---------------Kartengrundadresse
```

Die Variable M wird an Port A gegeben:

```
20 OUT(#70),M----------Variable laut Aufgabenstellung
        !!-------------Unteradresse fuer Port A
        !--------------Kartengrundadresse
```

Um Daten lesen zu koennen, muss Port A ueber das Command-Register auf Eingang gestellt werden:

```
30 OUT(#73),#9B--------Comm.-Word fuer A B u. C=Input
        !!-------------Unteradresse fuer Command-Register
        !--------------Kartengrundadresse
```

Jetzt kann fuer die neue Variable gelesen werden:

```
40 MNEU=INP(#70)
    !        !!--------Unteradresse fuer Port A
    !        !---------Kartengrundadresse
    !------------------Variable mit Wert von Port A
```

Zusammenfassung:

Mit einfachen BASIC-Befehlen kann ueber den DCE=Bus jeder der einzelnen Ports auf einem ext. System erreicht werden. Das Command-Register ist vorher mit einem Command-Word zu laden, das fuer die richtige Modeselektion der einzelnen Ports noetig ist.

Autor: Helmut Siegel

mit Genehmigung entnommen aus "USERS MANUAL für REAL-TIME-CLOCK, RWC-RTC"

➡ DAInaclock

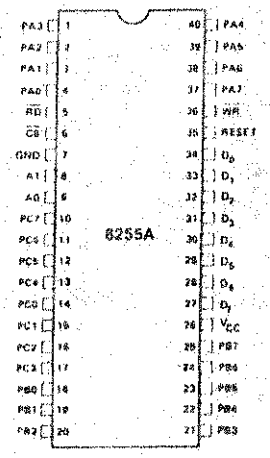
8255A
PROGRAMMABLE PERIPHERAL INTERFACE

- 24 Programmable I/O Pins
- Completely TTL Compatible
- Fully Compatible with Intel Microprocessor Families
- Improved Timing Characteristics
- Direct Bit Set/Reset Capability Easing Control Application Interface
- 40 Pin Dual-In-Line Package
- Reduces System Package Count
- Improved DC Driving Capability

The 8255A is a general purpose programmable I/O device designed for use with microprocessors. It has 24 I/O pins which may be individually programmed in two groups of twelve and used in three major modes of operation. In the first mode (Mode 0), each group of twelve I/O pins may be programmed in sets of 4 to be input or output. In Mode 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining four pins three are used for handshaking and interrupt control signals. The third mode of operation (Mode 2) is a Bi-directional Bus mode which uses 8 lines for a bi-directional bus, and five lines, borrowing one from the other group, for handshaking.

Other features of the 8255A include bit set and reset capability and the ability to source 1 mA of current at 1.5 volts. This allows darlington transistors to be directly driven for applications such as printers and high voltage displays.

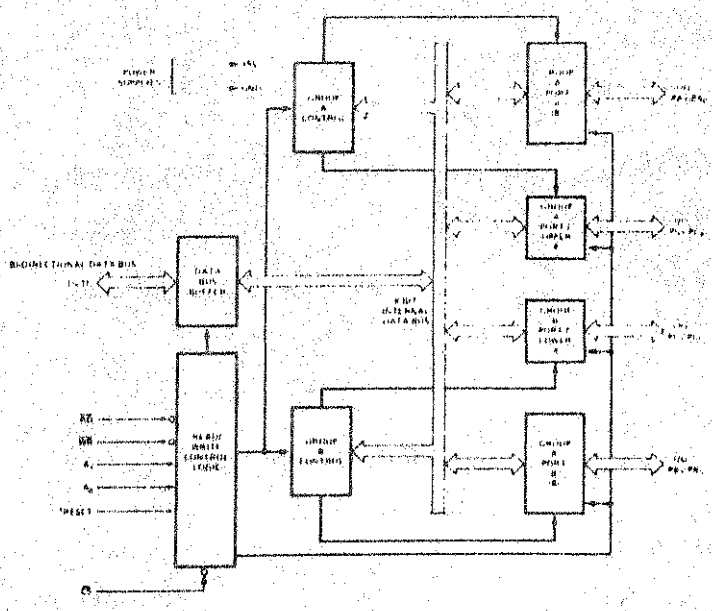
PIN CONFIGURATION

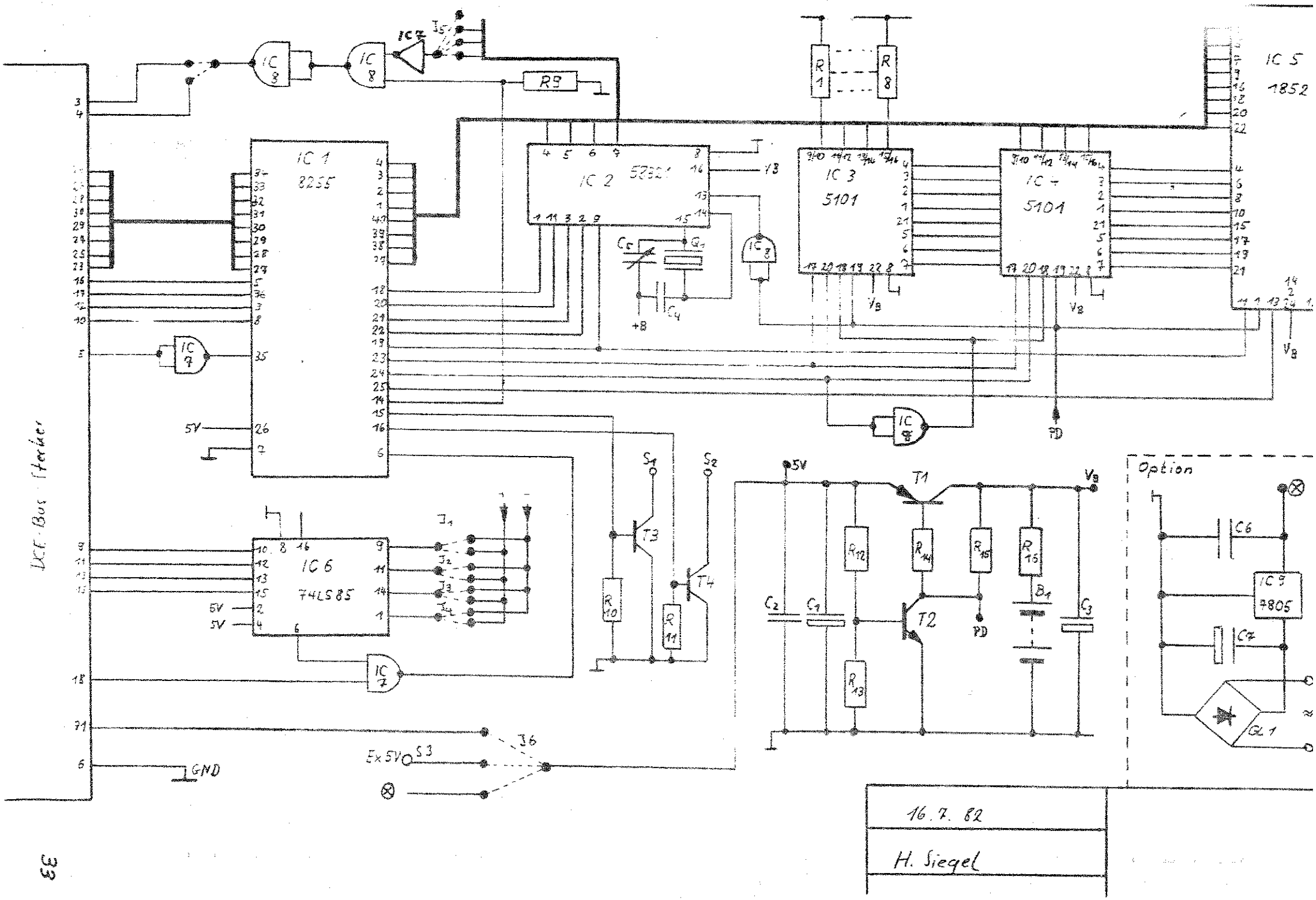


PIN NAMES

D_7-D_0	DATA BUS (BI-DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	0 VOLTS

BLOCK DIAGRAM





IEC-Bus Interface

(RESET)

Reset: A "high" on this input clears all internal registers including the Control Register and all ports (A, B, C) are set to the input mode.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the 8080 CPU "outputs" a control word to the 8255. The control word contains information such as "mode", "bit set", "bit reset" etc. that initializes the functional configuration of the 8255.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A – Port A and Port C upper (C7-C4)
Control Group B – Port B and Port C lower (C3-C0)

The Control Word Register can Only be written into. No Read operation of the Control Word Register is allowed.

Ports A, B, and C

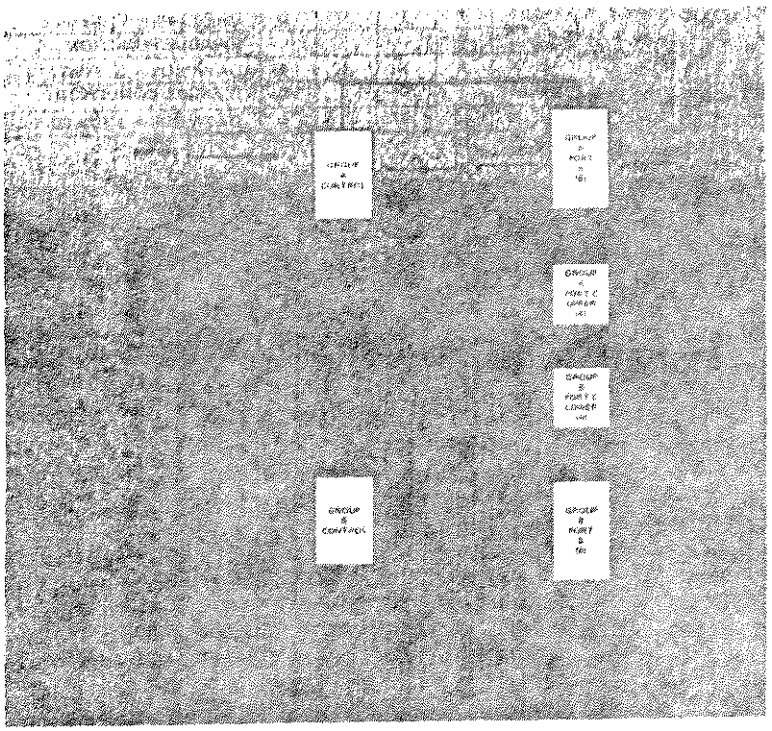
The 8255 contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255.

Port A: One 8-bit data output latch/buffer and one 8-bit data input latch.

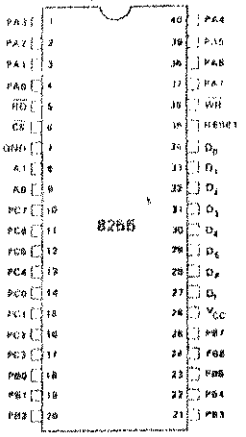
Port B: One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

Port C: One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with Ports A and B.

8255 BLOCK DIAGRAM



PIN CONFIGURATION



PIN NAMES

D_7–D_0	DATA BUS (BI-DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	0 VOLTS

8255 BASIC FUNCTIONAL DESCRIPTION

General

The 8255 is a Programmable Peripheral Interface (PPI) device designed for use in 8080 Microcomputer Systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the 8080 system bus. The functional configuration of the 8255 is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state, bi-directional, eight bit buffer is used to interface the 8255 to the 8080 system data bus. Data is transmitted or received by the buffer upon execution of INput or OUTput instructions by the 8080 CPU. Control Words and Status information are also transferred through the Data Bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the 8080 CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

($\overline{CS}$)

Chip Select: A "low" on this input pin enables the communication between the 8255 and the 8080 CPU.

($\overline{RD}$)

Read: A "low" on this input pin enables the 8255 to send the Data or Status information to the 8080 CPU on the Data Bus, in essence, it allows the 8080 CPU to "read from" the 8255.

($\overline{WR}$)

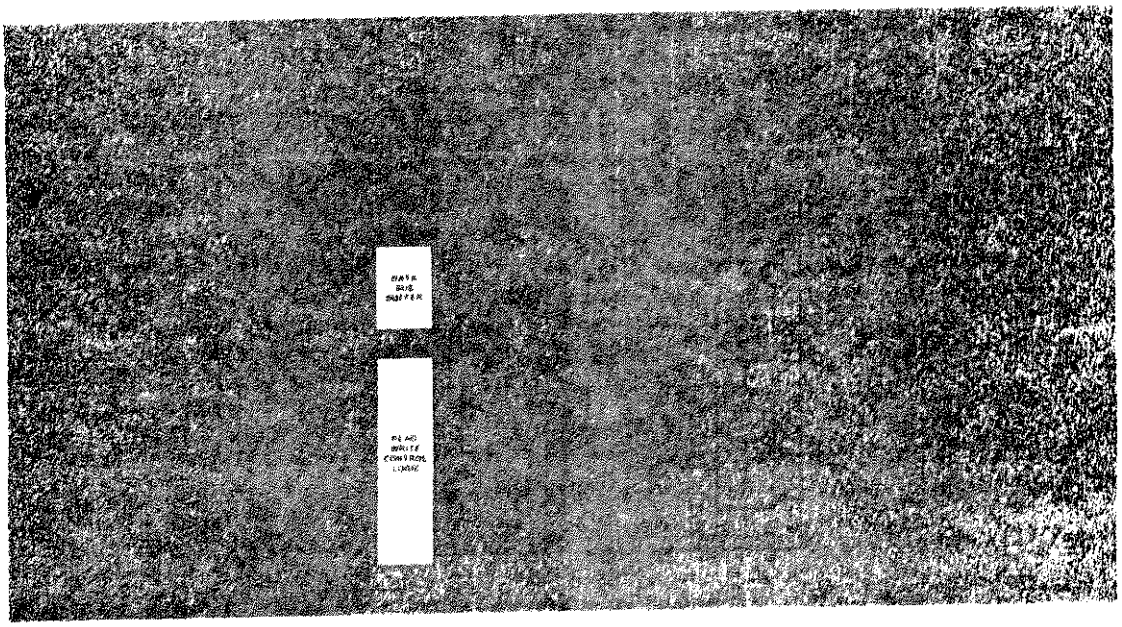
Write: A "low" on this input pin enables the 8080 CPU to write Data or Control words into the 8255.

(A_0 and A_1)

Port Select 0 and Port Select 1: These input signals, in conjunction with the $\overline{RD}$ and $\overline{WR}$ inputs, control the selection of one of the three ports or the Control Word Register. They are normally connected to the least significant bits of the Address Bus (A_0 and A_1).

8255 BASIC OPERATION

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	INPUT OPERATION (READ)
0	0	0	1	0	PORT A → DATA BUS
0	1	0	1	0	PORT B → DATA BUS
1	0	0	1	0	PORT C → DATA BUS
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	DATA BUS → PORT A
0	1	1	0	0	DATA BUS → PORT B
1	0	1	0	0	DATA BUS → PORT C
1	1	1	0	0	DATA BUS → CONTROL
					DISABLE FUNCTION
X	X	X	X	1	DATA BUS → 3-STATE
1	1	0	1	0	ILLEGAL CONDITION
X	X	1	1	0	DATA BUS → 3-STATE



8255 Block Diagram

8255 DETAILED OPERATIONAL DESCRIPTION

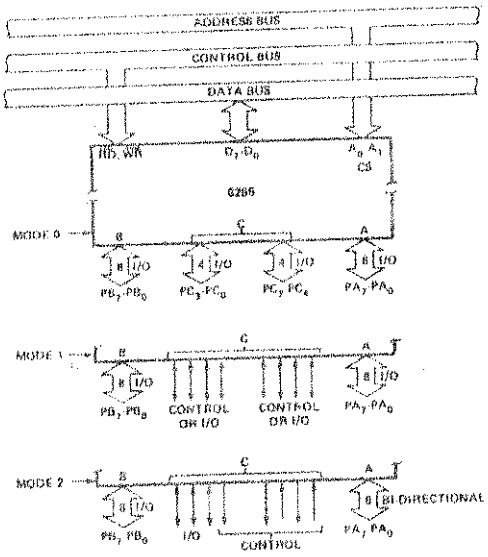
Mode Selection

There are three basic modes of operation that can be selected by the system software:

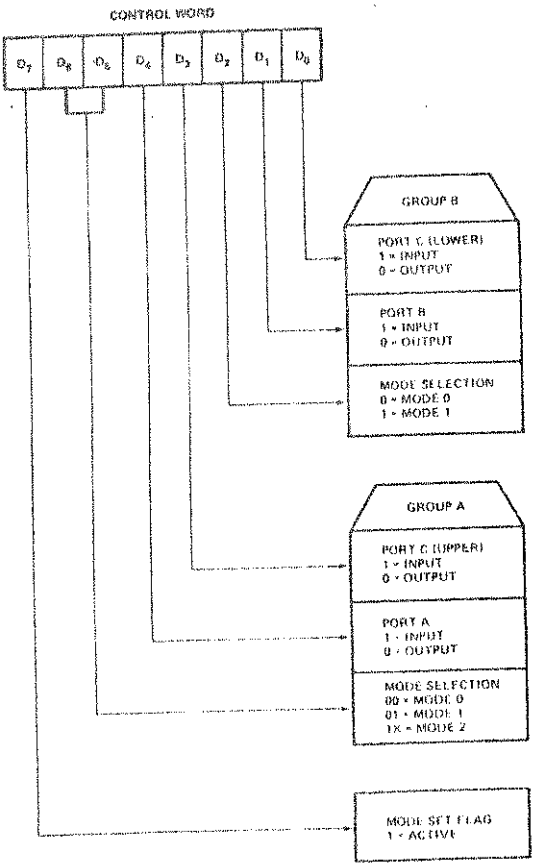
Mode 0 — Basic Input/Output
Mode 1 — Strobed Input/Output
Mode 2 — Bi-Directional Bus

When the RESET input goes "high" all ports will be set to the input mode (i.e., all 24 lines will be in the high impedance state). After the RESET is removed the 8255 can remain in the input mode with no additional initialization required. During the execution of the system program any of the other modes may be selected using a single OUTput instruction. This allows a single 8255 to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.



Basic Mode Definitions and Bus Interface

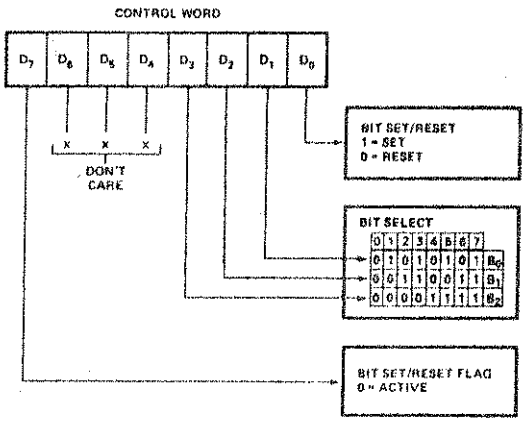


Mode Definition Format

The Mode definitions and possible Mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 8255 has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTput instruction. This feature reduces software requirements in Control-based applications.



Bit Set/Reset Format

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.

Interrupt Control Functions

When the 8255 is programmed to operate in Mode 1 or Mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from Port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the Bit set/reset function of Port C.

This function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without affecting any other device in the interrupt structure.

INTE flip-flop definition:

(BIT-SET) — INTE is SET — Interrupt enable
(BIT-RESET) — INTE is RESET — Interrupt disable

Note: All Mask flip-flops are automatically reset during mode selection and device Reset.

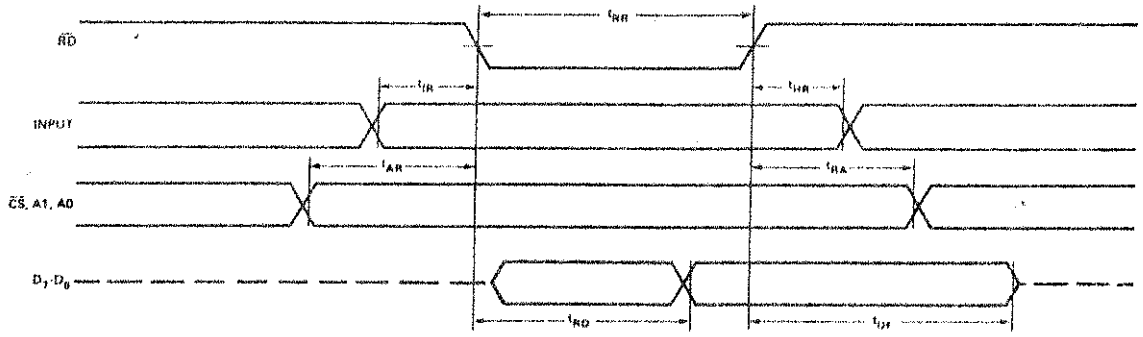
Operating Modes

Mode 0 (Basic Input/Output)

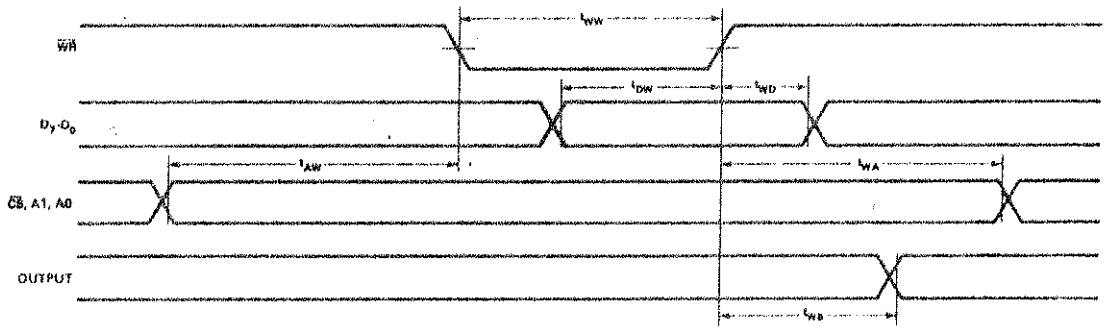
This functional configuration provides simple Input and Output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.



Mode 0 (Basic Input)



Mode 0 (Basic Output)

++++++++++++++++ CODE-TABELLE ++++++++++++++++

Im Bereich E8C5-E934 der ROM-Bank 3 liegt die Code-Tabelle, die die Zuordnung der Zeichencodes zu den einzelnen Tasten bestimmt:

E8C5	30-'0'	31-'1'	32-'2'					
E8C8	33-'3'	34-'4'	35-'5'	36-'6'	37-'7'	38-'8'	39-'9'	3A-':'
E8D0	3B-';'	2C-','	2D-'-'	2E-'.'	2F-'/'	0D-'RET'	41-'A'	42-'B'
E8D8	43-'C'	44-'D'	45-'E'	46-'F'	47-'G'	48-'H'	49-'I'	4A-'J'
E8E0	4B-'K'	4C-'L'	4D-'M'	4E-'N'	4F-'O'	50-'P'	51-'Q'	52-'R'
E8E8	53-'S'	54-'T'	55-'U'	56-'V'	57-'W'	58-'X'	59-'Y'	5A-'Z'
E8F0	5B-'['	5E-'^'	20-'SPC'	00-'RPT'	08-'DEL'	10-'CUP'	11-'CDN'	12-'CLT'
E8F8	13-'CRT'	09-'TAB'	80-'CTRL'	ab hier	SHIFT	30-'0'	21-'!'	22-'"'
E900	23-'#'	24-'$'	25-'%'	26-'&'	27-'''	28-'('	29-')'	2A-'*'
E908	2B-'+'	3C-'<'	3D-'='	3E-'>'	3F-'?'	0D-'RET'	61-'a'	62-'b'
E910	63-'c'	64-'d'	65-'e'	66-'f'	67-'g'	68-'h'	69-'i'	6A-'j'
E918	6B-'k'	6C-'l'	6D-'m'	6E-'n'	6F-'o'	70-'p'	71-'q'	72-'r'
E920	73-'s'	74-'t'	75-'u'	76-'v'	77-'w'	78-'x'	79-'y'	7A-'z'
E928	5D-']'	7E-'~'	20-'SPC'	00-'RPT'	08-'DEL'	14-'SUP'	15-'SDN'	16-'SLT'
E930	17-'SRT'	09-'TAB'	80-'CTRL'	00	00-'SHIFT'			

RET = RETURN
SPC = SPACE
CUP = CURSOR UP
CDN = CURSOR DOWN
CLT = CURSOR LEFT
CRT = CURSOR RIGHT
SUP = CURSOR UP + SHIFT
SDN = CURSOR DOWN + SHIFT
SLT = CURSOR LEFT + SHIFT
SRT = CURSOR RIGHT + SHIFT

Man kann die Zuordung der einzelnen Codes aendern, indem man die Tabelle in den RAM verschiebt: UT,ME8C5 E934 68C5,B und den Zeiger auf den Beginn der Tabelle entsprechend aendert: POKE #2A7,#C5:POKE #2A8,#68

Im Prinzip kann jetzt jede Taste mit jedem beliebigen Zeichen belegt werden. Um auf keine bestehenden Belegungen verzichten zu muessen, empfiehlt es sich, die neuen Zeichen auf die nicht verwendeten Tastenkombinationen TAB, SHIFT+TAB, SHIFT+RETURN, SHIFT+CTRL, SHIFT+0, SHIFT+SPACE, SHIFT+DEL zu legen.

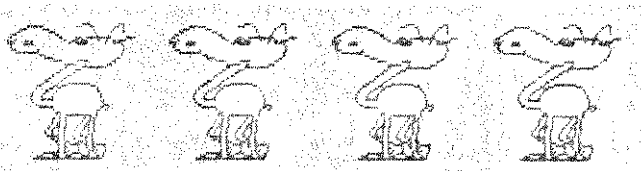
Achtung: Die Zeichen der ASCII-Steuercodes 00-1F koennen in BASIC nur durch die Codes>128 sichtbar gemacht werden. Diese Zeichen sind im Editor leider nicht sichtbar.
Im Editor muessen die Codes<128 verwendet werden. Wenn das Programm dann gelistet wird, werden allerdings die Steuerfunktionen ausgefuehrt. Schwierigkeiten ergeben sich natuerlich auch, falls das Programm auf einem Drucker ausgegeben werden soll.

Beispiele:

```
POKE #68F9,#7B:POKE #690D,#88:POKE #692A,#98
POKE #692C,#40:POKE #6931,#7D:POKE #6932,#7C
```
Es sind 6 weitere ASCII-Zeichen ueber die Tastatur erreichbar.

```
POKE #692D,#5E:POKE #692E,#8C:POKE #692F,#5F:POKE #6930,#89
```
Bei SHIFT+Cursor erscheinen Pfeile.

```
FOR I=#690E TO #6927:POKE I,I-#690F+#80:NEXT
```
Anstelle der Kleinbuchstaben erscheinen Graphikzeichen.



MODE 5,6 / 5A,6A

1) BILDSCHIRMORGANISATION

i) *** MODE 5,6 ***
Der Bildschirm ist eingeteilt in 256 Zeilen und 336 Spalten. Jeweils zwei Bytes beinhalten die Information für 8 Punkte. Dementsprechend benötigt eine Zeile (336/8)*2=84 Bytes für die Darstellung der 336 Punkte.

	0..7	8..15	...	328...335
255	EFCD/EFCA	BFE9/BFE8		BF99/BF98
254	BF91/BF90	BF8F/BF8E		BF3F/BF3E
253	BF37/BF36	BF35/BF34		BEE5/BEE4
.				
.				
.				
2	66F9/66F8	66F7/66F6		66A7/66A6
1	669F/669E	669D/669C		664D/664C
0	6645/6644	6643/6642		65F3/65F2

ii) *** MODE 5A,6A ***
Der Bildschirm zeigt nur noch die Zeilen 0 bis 211.

	0..7	8..15	...	328...335
211	BFEB/BFEA	BFE9/BFE8		BF99/BF98
210	BF91/BF90	BF8F/BF8E		BF3F/BF3E
.				
.				
.				
2	7671/7670	766F/766E		761F/761E
1	7617/7616	7615/7614		75C5/75C4
0	75BD/75BC	75BB/75BA		756B/756A

iii) *** MODE 5,6 / 5A,6A ***
Rechts und links von den oben angegebenen Bytes befinden sich jeweils zwei weitere Bytes (also 8 Punkte), die von BASIC aus nicht erreichbar sind.
Daher sieht die vollständige Zeile 0 in MODE 5,6 so aus:

```
    2 Kontrollbytes  : 6649 , 6648
    2 Bytes für Rand : 6647 , 6646
   84 Bytes der Zeile: 6645 bis 65F2
    2 Bytes für Rand : 65F1 , 65F0
```

Der Abstand zwischen zwei Zeilen beträgt 90 dezimal.
Die Adressen, in denen ein Punkt (x,y) kodiert ist, lassen sich leicht ermitteln (hier für MODE 5,6):

```
adr:=#6645-INT(X/8) *2+90*y
```

Der gesucht Punkt ist in adr und adr-1 kodiert.

2) AUFBAU DER BEIDEN BYTES

i) *** MODE 6,6A ***

Ich erläutere die Bytebelegung anhand eines Beispiels.
Dazu betrachte man die Bytes #6644 und #6645, die in MODE 6
die Punkte 0 bis 7 der Zeile 0 beinhalten.
Nach der Eingabe: COLOR0 0 : 15 :
DOT 0,0 0:DOT 1,0 0:DOT 2,0 15:DOT 3,0 0:
ergibt sich folgende Belegung der Bytes:

7 6 5 4 3 2 1 0 -Bit Wertigkeit
0 1 2 3 4 5 6 7 -Dot Position
adr.: #6644 := #82 = 1 0 0 0 0 0 1 0
adr.: #6645 := #12 = 0 0 0 1 0 0 1 0
0 8 E-Punkt 4 b -Farbinformation

Ein in einem Byte gesetztes Bit bewirkt, daß an der entsprechenden Stelle auf dem Bildschirm ein Punkt gesetzt wird. In Abhängigkeit von dem Byte, in dem das Bit gesetzt wurde, wird die Farbe gewählt.
Die COLOR0-Anweisung sei so: COLOR 0:C0 C1 C2 C3 C4.
Folgende Möglichkeiten ergeben sich für Bits gleicher Wertigkeit:
eine '0' in beiden Bytes entspricht C0 = Hintergrund
eine '1' in beiden Bytes entspricht C4
eine '1' nur in der low address entspricht C2
eine '1' nur in der high address entspricht C3

ii) *** MODE 5,5A ***

Als Beispiel wähle ich MODE 5A, Zeile 0, Punkte 0 bis 7.
In einem 4-Farben-Mode beinhalten beide Bytes die Orts- und Farbinformation. In einem 16-Farben-Mode bekommt jedes Byte eine eigene Aufgabe. Das low-address-Byte beinhaltet dann die Farbinformation, während das high-address-Byte die zu zeichnenden Punkte kodiert.
Folgendes Beispiel: COLOR0 8 : 0 0:MODE5A
DOT 0,0 0:DOT 1,0 0:DOT 2,0 0:DOT 3,0 0
Grau (8) ist als Hintergrundfarbe gewählt. Alle gezeichneten Punkte erscheinen in schwarz (0).
Die Belegung von #75BC und #75BD ist die folgende:

7 6 5 4 3 2 1 0 -Bit Wertigkeit
adr.: #75BC := #00 = 0 0
adr.: #75BD := #92 = 1 0 0 1 0 0 1 0
0 1 2 3 4 5 6 7 -Dot Position

Alle gesetzten Bits innerhalb des high-address-Byte bestimmen als zu zeichnende Farbe diejenige, die durch die oberen vier Bits des low-address-Byte festgelegt ist. Die Farbe, die durch die unteren vier Bits festgelegt wird, ist an den Punkten zu finden, die im high-address-Byte durch eine Null kodiert sind.
Aus der obenstehenden Überlegung ist es leicht ersichtlich, daß innerhalb von acht Punkten nur zwei Farben auftreten dürfen. Zu beachten ist außerdem noch, daß in obigem Beispiel die beiden Bytes auch so belegt sein dürfen:
adr.: #75BC := #80 = 1 0
adr.: #75BD := #6D = 0 1 1 0 1 1 0 1
Wie man erkennen kann, hätte dies die gleiche Ausgabe wie die obige Belegung.

Stefan Goller, Meckenheim

D A I P A D D L E

H.E.Goller 03/01/83

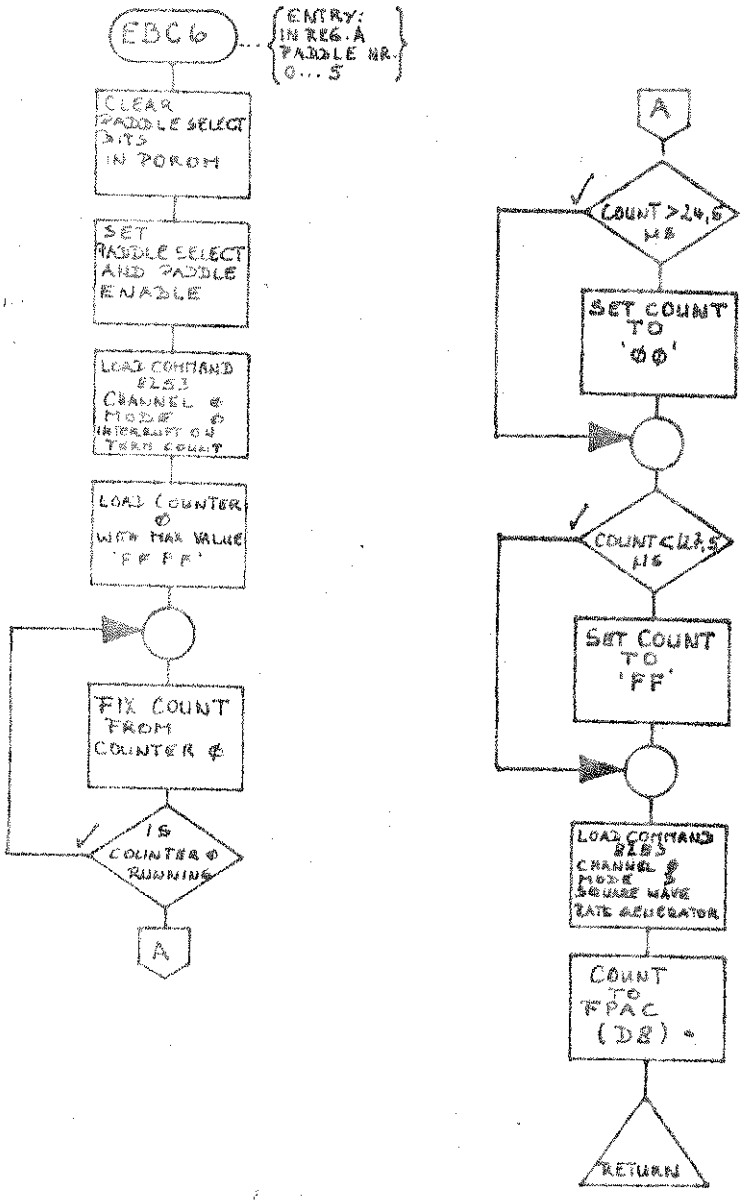
1 Basic

A=PDL(I)

Der Variable A wird ein Wert von 0 bis 255 zugewiesen, dieser gibt die Position eines der 6 Potentiometer an, die durch den Wert I adressiert werden. I=(0-5)

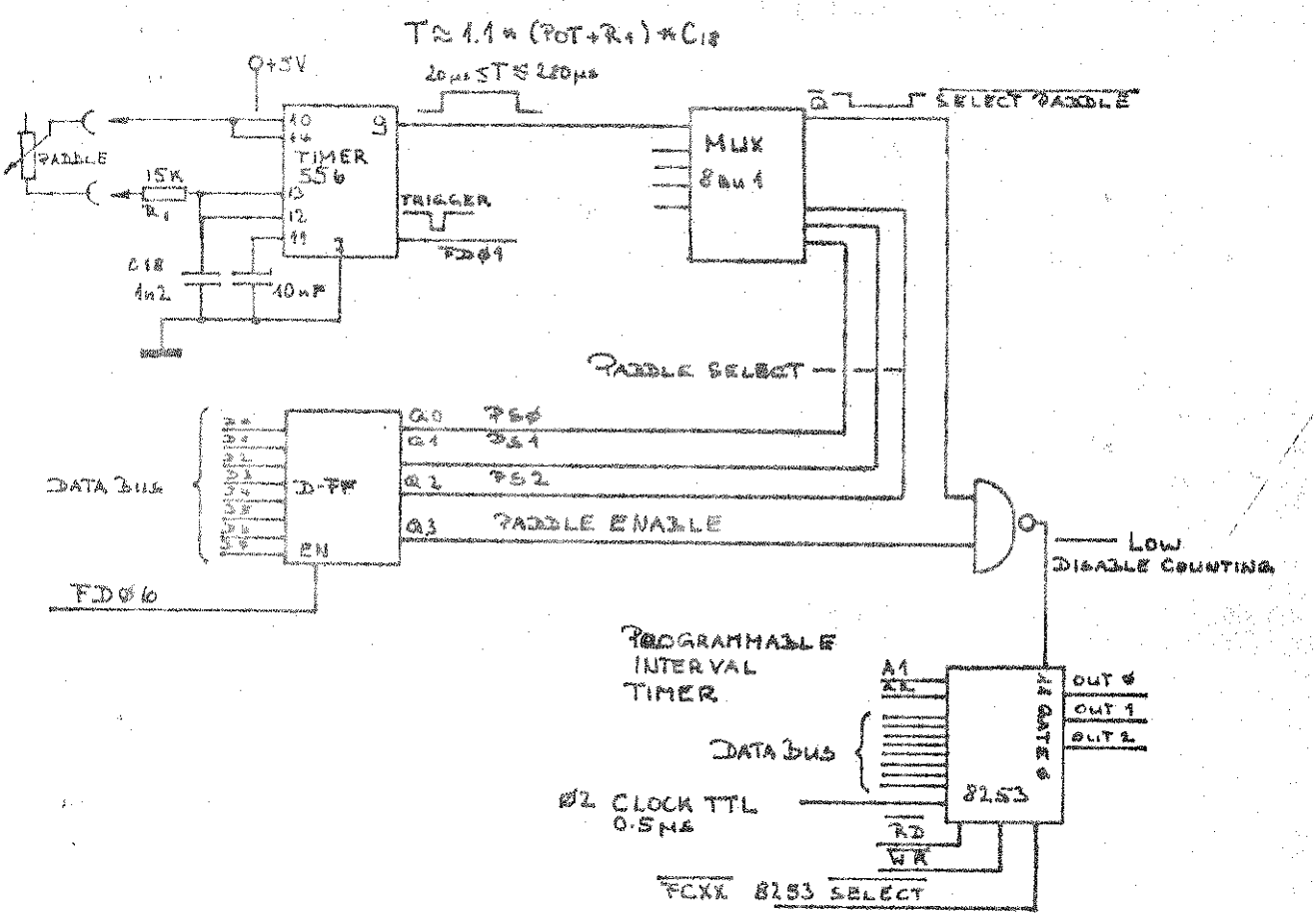
2 Maschinensprache

```
MVI   A,x       x=paddle nr. 0....5
CALL  :EBC6
LDA   :D8       Wert 0...255 vom Software Accu
```



FW 7

3 Hardware



Nachdem der Timer 556 über einen Triggerimpuls angestoßen wurde, erzeugt er ein Ausgangssignal, dessen Länge direkt abhängig von der Stellung des Potentiometers ist. Der Triggerimpuls wird durch lesen der I/O Device Adresse FD01 erzeugt. Die Länge des Ausgangsignals errechnet sich nach folgender Formel $T \approx 1.1 \times (POT + R1) \times C18$. Daraus ergibt sich eine Zeitdauer von 19.8µs wenn POT=0, in der max. Stellung von 200 KΩ eine Dauer von 283.8µs .

Das Ausgangssignal wird einen 8 zu 1 MUX zugeführt und durch die eingestellten Paddle select Signale durchgeschaltet. Diese Paddle select Signale entstehen durch einschreiben der drei wertniedrigsten Bits auf die Adresse FD06.

Der $\overline{q}$ Ausgang des MUX ist auf ein NAND Gate geschaltet. Der andere Eingang des Gates erhält das Paddle enable Signal, welches durch einschreiben des 4.Bits (xxxx 1xxx) auf die Output Device Adresse FD06 ermöglicht wird. Für die Dauer des $\overline{\text{select paddle}}$ Signals ist das Nand Gate nicht erfüllt, dies ermöglicht nun den 8253, da dieses Signal zum Gate 0 geführt ist, seinen Counter herabzuzählen, bis $\overline{\text{select paddle}}$ inaktiv wird und ein weiteres Zählen blockiert wird.

Verwendung der BETRIEBSSYSTEM - GRAFIK - ROUTINEN in ASSEMBLERPROGRAMMEN

Der Aufruf sämtlicher Grafikroutinen erfolgt prinzipiell nach folgendem

Schema :
```
RST 5
DATA xx
```

Welches DATA für welche Routine notwendig ist und welche Register mit welchen Angaben geladen sein müssen, sei im Folgenden beschrieben :

DATA ØØ : MODE Ø Initialisierung

Eingang: HL: obere Schirmadresse
DE: Zeigt auf die Liste mit den Init Parametern.

Ausgang: Alle Register zerstört

DATA Ø3 : Zeichenausgabe

Eingang: A: Zeichen

Ausgang: ABCDEHL gerettet
CARRY=1: Zeichen ignoriert

DATA Ø6 : COLORT

Eingang: HL: Zeigt auf das erste von 4 Bytes mit den 4 Farben (obere 4 bits werden ignoriert).

Ausgang: Alle Register gerettet.

DATA Ø9 : CURSOR

Eingang: HL: y,x Position

Ausgang: BCDEHL gerettet
CARRY=1 falls OFF SCREEN

DATA ØC : CURX, CURY

Eingang: keine Bedingungen

Ausgang: HL: Y,X des Cursors
DE: max. Y, X Werte des Textes

DATA ØF : Cursor erneuern

Eingang: HL: Zeigt auf neue Cursor Info

Ausgang: Alle Register gerettet

DATA 12 : Cursor blinken

Eingang: keine Bedingungen

Ausgang: Alle Register gerettet.

DATA 15 : GETC

Eingang: C: Position in der Zeile, von wo das Zeichen gelesen werden soll.

Ausgang: A: Zeichen

DATA 18 :	MODE x	Eingang: A:Mode Ausgang: ABCDEHL gerettet CARRY=1: OUT OF MEMORY
DATA 1B :	COLORG	Eingang: HL: Zeigt zum 1. der 4 Farbbytes. Ausgang: Alle Register gerettet
DATA 1E :	DOT	Eingang: C,HL: Y,X Koord. des Punktes A: Farbe Ausgang: CARRY=1: COLOUR NOT AVAILABLE
DATA 21 :	DRAW	Eingang: B,DE: Y,X 1.Punkt C,HL: Y,X 2.Punkt A: Farbe Ausgang: CARRY=1: wie DOT
TA 24 :	FILL	Eingang: B,DE; C,HL: Y,X der beiden Punkte A: Farbe Ausgang: CARRY=1: wie DOT
DATA 27 :	SCRN, XMAX, YMAX	Eingang: C,HL: Y,X des Punktes Ausgang: A: Farbe B,DE: YMAX, XMAX CHL gerettet CARRY=1: Fehler
DATA 2A :	Init EDITOR	Eingang: Keine Bedingungen Ausgang: HL: Cursor Position AF: zerstört BCDE gerettet
TA 2D :	EDIT	Eingang: A: Zeichen Ausgang: CARRY=1: fertig ABCDEHL gerettet

Marco van Meegen
Tempelhofer Str.42
5090 Leverkusen 1

Der DAI - Editor

Der DAI ist im Gegensatz zu vielen anderen Computern dieser Preisklasse mit einem sehr leistungsfaehigen Full-Screen-Editor ausgestattet. Dieser ermoeglicht bei trickreicher Anwendung eine Anzahl von sehr hilfreichen Operationen. Ich will hier die genaue Funktionsweise des Editors erklaeren,wie man ihn fuer MERGE,KILL und fuer eigene Programme einsetzen kann.

Was geschieht beim Eintippen des Befehls EDIT 10-100 ?

1) Der Bildschirm wird auf MODE 0 gesetzt.
2) Alle Variablen und Felder werden geloescht.
3) Der Speicherbereich fuer den EDIT-BUFFER wird initialisiert.
   Die SPEICHERAUFTEILUNG im DAI ist folgendermassen organisiert :
   (alle Angaben in Hexadezimalschreibweise)
   Von 0 - 2EB : Bereich des Betriebssystems
   2EC - (29B,C) : Bereich fuer Maschinenprogramme etc.
   (nicht von Basic benutzt)
   (29B,C) - (29F,2A0) : Bereich der HEAP,hier werden alle
   Strings und Arrays gespeichert
   (29F,2A0) - (2A1,2) : Hier wird das eigentliche Basic-programm
   in einem internen Zwischencode gespeichert
   (2A1,2) - (2A3,4) : Bereich der SYMBOL-TABLE,alle Variablennamen
   werden hier gespeichert,sowie alle Variablenwerte
   von INT- und FPT- variablen.
   Bei Arraynamen und Strings ist ein Pointer
   auf die HEAP gespeichert,wo sich das Array
   oder der String befindet.
   (2A3,4) - (2A5,6) : Dieser Bereich ist FREI,wird aber beim
   Editieren sowie beim Laden oder Abspeichern
   von Stringarrays als Zwischenspeicher benutzt.
   (2A5,6) - BFFF : Bildschirmspeicher

Beim Initialisieren des Editors wird nun das Basic-programm sowie die Symboltabelle an das Ende des FREIen Bereiches geschoben und die Heap geloescht. Daher ist nun der Bereich von (29B,C) bis (29F,2A0) frei und kann zur Abspeicherung des editierten Textes benutzt werden. Der Pointer zum Anfang des Edit-buffers (A2,A3) wird auf (29B,C)+2 gesetzt und der Pointer zum Ende des Edit-buffers (A6,A7) wird auf (29F,2A0)-2 gesetzt.
Der Pointer (A4,A5) zeigt auf das Ende des vom Editor benutzten Bereiches,dieser ist gleich (A6,A7),da bis jetzt noch kein Text im Edit-buffer gespeichert ist.

4) Nun wird der Ausgabe-switch (131) auf 2 gesetzt. Damit wird jede Ausgabe anstatt auf den Bildschirm in den Edit-buffer gebracht.

5) Jetzt wird der Befehl LIST 10-100 ausgefuehrt. Da aber der Ausgabe-switch auf den Edit-buffer zeigt,wird in diesen gelistet.
Die Ausgabe kann wie beim normalen LIST durch Druecken einer Taste gestoppt werden,dann muss man Space druecken,um fortzufahren.
Wenn BREAK gedrueckt wird,wird die Ausgabe abgebrochen und in den EDIT-modus gesprungen. Aber ! ACHTUNG ! ,jetzt enthaelt der Edit-buffer nur die bis zum Break gelisteten Zeilen,die anderen sind verloren,wenn der Editor nicht durch BREAK/BREAK verlassen wird.

Beim Listen wird der Pointer (A4,A5) hinter das Ende des Listings gesetzt. Der Edit-buffer enthaelt also das Listing der Zeilen 10-100 im ASCII-Format. Das erste Zeichen ist bei (A2,A3),das letzte bei (A4,A5)-2,da hinter dem Text noch eine 0 als Endemarkierung steht und der Pointer hinter das Ende des Edit-buffers zeigt.

6) Der Ausgabe-switch wird auf 0 zurueckgesetzt (Ausgabe auf SCREEN/RS232).
7) Nun wird die erste Seite des Textes im Buffer auf den Bildschirm gelistet. Dieses geschieht mit Hilfe der Routine,die durch RST 5 data 2A angesprungen werden kann.

8) Nun beginnt der eigentliche Editier-vorgang. Die Tastatur wird abgefragt (CALL D6BE) ,und entsprechend wird der Edit-buffer und das Textfenster veraendert. Dieses geschieht durch RST 5 data 2D, wobei der Code der gedrueckten Taste im Akku stehen muss.

9) Der Edit-modus wird durch BREAK verlassen.
10) Jetzt gibt es zwei Moeglichkeiten :
a) Es wird noch einmal BREAK gedrueckt.
b) Eine beliebige andere Taste wird gedrueckt.

a) Wenn ein zweites Mal BREAK gedrueckt wird,werden die HEAP sowie Programm und Symboltable auf ihren urspruenglichen Platz zurueckgeschoben. Dabei wird der Edit-buffer durch das Programm ueberschrieben, wenn die HEAP nicht gross genug dimensioniert war,sonst bleibt der editierte Text in der HEAP stehen und kann durch POKE 135,2 abgerufen werden.

b) Wenn man eine andere Taste als BREAK drueckt,wird der Eingabe-switch (135) auf 2 gesetzt und die alten editierten Zeilen im Programm geloescht,so dass Platz fuer die verbesserten Zeilen ist.
Nun wird das Programm direkt hinter den benutzten Bereich des Edit-buffers geschoben und zum BASIC-EINGABE-MODUS zurueckgesprungen.
Da der Eingabe-switch auf 2 steht,wird die Eingabe nicht von der Tastatur sondern von dem Edit-buffer geholt.Wenn das Ende des benutzten Edit-buffers erreicht ist,werden die HEAP sowie das Programm und die Symboltable auf ihren urspruenglichen Platz geschoben.

TRICKS MIT DEM EDIT-BUFFER

MERGE : CLEAR groesser als das Programm im ASCII-Code
EDIT
BREAK/BREAK
LOAD
POKE 135,2

KOPIEREN VON ZEILEN : CLEAR groesser als zu kopierende Zeilen
EDIT nr-nr
Zeilennummern veraendern
BREAK/BREAK
POKE 135,2

LOESCHEN VON ZEILEN : CLEAR groesser als zu loeschende Zeilen
EDIT nr-nr und direkt zweimal RETURN (dadurch wird das Listen in den EDIT-buffer unterbrochen)
BREAK (das bis dahin gelistete Programm erscheint auf dem Bildschirm, es sollte nur eine Zeile sein)
Die Zeilennr. loeschen
BREAK/SPACE

DIE EDITOR-STORY

Dieser Artikel beschreibt den Ablauf des BASIC-Kommandos 'EDIT'. Für weitergehende Einzelheiten sollte das 'DAI pC FIRMWARE MANUAL' zu Rate gezogen werden.

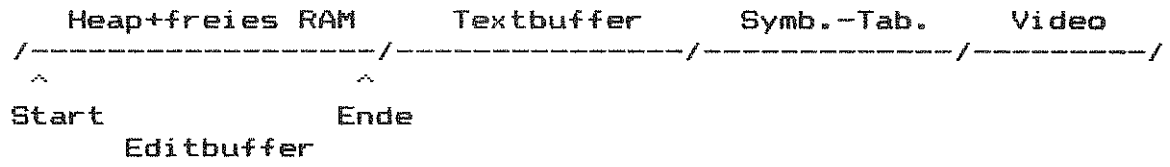
Wenn das BASIC-Kommando 'EDIT' eingegeben wird (nur im Direkt-Modus), springt das Betriebssystem auf Adresse #0E1F5 (ROM-Bank 0).

- Initialisieren des Editbuffers:

E1F5 E265 Der EDIT-Buffer wird bereitgestellt. Der Bildschirm wird auf MODE 0 gesetzt. Falls nicht genügend freies RAM vorhanden ist, wird über #D86D eine Fehlermeldung ausgegeben. Ist genügend Speicher vorhanden, werden Heap und Symbol-Tabelle gesäubert (alle Variablen werden auf Null gesetzt).

E26C Danach wird die Menge des freien RAM berechnet und die Heap-Größe dazuaddiert. Dieser Wert gibt den maximal nutzbaren RAM-Bereich für den Editbuffer an.

E275 Nun werden Textbuffer und Symbol-Tabelle (also das BASIC-Programm) an das Ende des freien RAM verschoben. Somit entsteht ein großer, zusammenhängender RAM-Bereich zwischen 'start of heap' und 'start textbuffer':



E27D Die Startadresse des Heap +2 ist die Startadresse des Editbuffers; das Ende des Editbuffers wird an die Startadresse des Textbuffers gesetzt.

E28E Der Output-Schalter #131 wird auf den Editbuffer gesetzt.

E1F8 E19F Jetzt wird das Programm in den Editbuffer gelistet. Dies geschieht genauso wie beim Listen auf den Bildschirm, sodaß der Editbuffer ein komplettes Listing in 'reinem BASIC-Text' enthält.

E1FC Nun wird der Output-Schalter #131 auf den Bildschirm zurückgesetzt. Die Tastatur wird wieder komplett abgefragt, und ein '0'-Byte wird ans Ende des Listings angefügt, um das Ende des Inhalts anzuzeigen (alle anderen Bytes sind <> 0).
Die Tabulatoren-Tabelle wird durch Null-Setzen des Pointers #00B4 abgeschaltet, deshalb sind TAB's im Editbuffer normalerweise nicht benutzbar.

E20B Hier wird der Bildschirm-Editor mittels RST 5; DATA #2A initialisiert:

2EBF4 Wieder wird der Schirm auf MODE 0 gesetzt und geleert, und alle Zeiger für den Cursor und das Fenster (der sichtbare Teil des Editbuffers auf dem Schirm) werden 'genullt'.

2EC17 Auf diesen lerren Schirm wird eine Seite (24 Zeilen), beginnend beim Textanfang, geschrieben.

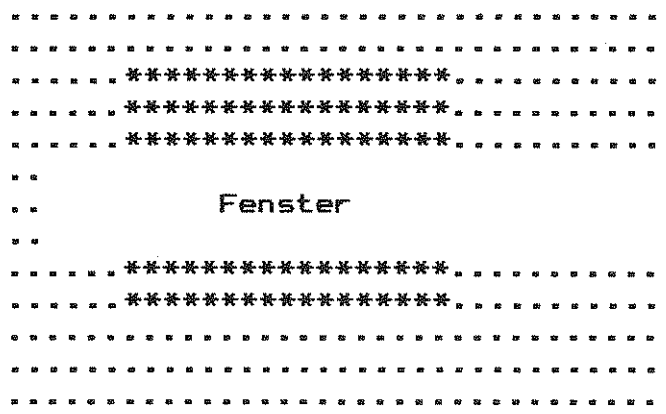
- Eingaben in den Editbuffer:

E20D Nun läuft das Programm in eine Schleife, die Eingaben erwartet. Abhängig von der gedrückten Taste werden verschiedene Aktionen vom Editor vollzogen. Durch Drücken der BREAK-Taste kann die Eingabe beendet werden.

E216 Die Eingaben werden durch RST 5; DATA #2D ausgewertet.

2EC1E Mittels der Cursortasten kann der Cursor über den Bildschirm bewegt werden. Wenn die Ränder des Schirms erreicht werden, 'rollt' dieser automatisch. D.h., daß das Fenster über den Text im Editbuffer bewegt wird.

Dieses 'Fenster' kann man sich als ein rechteckiges Vergrößerungsglas vorstellen, das über ein Blatt Papier bewegt wird. Nur der durch das Glas gesehene Teil ist sichtbar:



Durch gleichzeitiges Drücken von Cursor- und SHIFT-Taste kann das Fenster auch bewegt werden. Diesmal bleibt der Cursor auf demselben Zeichen (solange der Schirmrand nicht erreicht wird).

2EF4B Jedes andere eingetippte Zeichen wird an der Cursorposition in den Textbuffer eingefügt. Um Platz für das neue Zeichen zu machen, wird der Rest des Textes bewegt.

2EFCC Falls das eingegebene Zeichen 'CHAR DEL' ist, wird das Zeichen an der momentanen Cursorposition gelöscht und der Rest des Textes um eine Stelle bewegt.

E218 Wenn ein eingegebenes Zeichen ausgewertet und evtl. in den Editbuffer eingefügt wurde, wird das nächste Zeichen erwartet und behandelt. Dieser Prozess wird fortgesetzt, bis die BREAK-Taste gedrückt wird.

- Rückkehr von EDIT:

E21B Nach dem 'BREAK' wird der Schirm geleert und der nächste Tastendruck erwartet.

E222 Falls die zweite Taste wieder 'BREAK' ist, werden alle ge-
E24D machten Textkorrekturen ignoriert. Mit anderen Worten: Das alte Programm wird wiederhergestellt.
Die Zeiger für Heap, Textbuffer und Symboltabelle werden auf ihre alten Werte gesetzt und das Programm an seinen vorherigen Platz verschoben.

Die Zeiger für den Editbuffer und der Inhalt des Editbuffers werden nicht gelöscht. Weil aber der Editbuffer vom Start des freien RAMs beginnend (incl.Heap) aufwärts beschrieben wurde, wird er vom alten Programm überschrieben.

E225 Falls nach dem ersten Break die zweite Taste irgendeine andere ist, werden die Änderungen, die im Editbuffer gemacht wurden, nun in das alte Programm eingefügt.

- Füge editierte Programmzeilen ein:

E22A Der Eingabe-Schalter #135 wird auf den Editbuffer gesetzt. D.h.: der Editbuffer ist die Quelle für zu kodierende Zeilen.

E231-E238 Weil in den Zeigern #0119 und #011B der Start und das Ende des editierten Programmteiles gespeichert wurden, kann nun dieser Teil aus dem alten Programm (das sich immer noch am Ende des freien RAM befindet) entfernt werden.

E23B-E247 Nun wird die Länge des belegten Bereiches des Editbuffers berechnet: vom Anfang des Heap (=Start Editbuffer) bis zum Eingabe-Zeiger #00A4 für den Editbuffer.

E248 Jetzt wird das alte Programm (ohne die gelöschten Zeilen) ans Ende des benutzten Editbuffers verschoben:

```
   Editbuffer      Textbuffer     Symb.-Tab.     freies RAM
/-------------/-/--------------/--------------/------------/
^
Start Heap
```

E24B Nun ist die BASIC-Routine 'EDIT' beendet und das Programm kehrt an die normale BASIC-Monitor-Routine bei #C818 zurück.

C84B Weil der Input-Schalter #135 auf den Editbuffer gesetzt ist,
D879 werden Textzeilen jetzt aus diesem Buffer gelesen.

- Eingaben vom Editbuffer:

E291 Die Startadresse des Editbuffers wird im Input-Zeiger #0132 gespeichert. Dieser Zeiger zeigt auf den Anfang der zu kodierenden Zeile.

E298 Ein Zeichen wird aus dem Editbuffer gelesen. Wenn dieses Zeichen nicht '0' ist (am Ende des Buffers), wird der Rest der
E2A0 Zeile bis zum 'CR' gelesen, das das Ende der Textzeile anzeigt.

E2A3 Nach einem 'CR' wird die Startadresse des Editbuffers auf den Anfang der nächsten Zeile gesetzt (dabei wird der Editbuffer jedesmal kleiner) und das Programm kehrt zum BASIC-Monitor zurück.

C84E Hier wird die aus dem Editbuffer gelesene Zeile kodiert (in
C867 den Textbuffer-Code übersetzt und in diesen Buffer an der richtigen Stelle eingefügt).
Über #C818 wird die nächste Zeile aus dem Editbuffer geholt, bis das Ende des Editbuffers (eine '0') gefunden wird.

E2AA Wenn alle Zeilen aus dem Editbuffer kodiert und in den Textbuffer eingefügt sind, wird der Input-Schalter #0135 auf die Tastatur zurückgesetzt.

E2AD Nun verschwindet der Editbuffer: der Textbuffer und die Symboltabelle werden genau hinter das Ende des Heap verschoben und alle Zeiger werden richtig gesetzt.

Jetzt können weitere Eingaben wieder von der Tastatur gemacht werden: das Programm kehrt zum BASIC-Monitor bei #C818 - #C956 zurück.

übersetzt: Bernd Preusing, September '83

Der BASIC Monitor
1. Teil: Allgemeine Beschreibung

Autor: Jan Boerrigter

Dieser Artikel beschreibt den prinzipiellen Aufbau des BASIC Monitors. In einem weiteren Artikel werden dann einige Teile des Monitors genauer erklärt. Als Referenz zu diesem Artikel wird das 'DAI FIRMWARE MANUAL' verwendet.

Der BASIC Monitor kann als das 'Herz' des DAI-BASIC-Betriebssystems angesehen werden. Unter der Kontrolle dieses Monitors werden die Programmzeilen in den Textbuffer eingefügt, direkte Kommandos wie RUN, PRINT (d.h. ohne Zeilennummer) werden durchgeführt und während eines Programmlaufs wird z.B. auf Programmunterbrechung durch BREAK abgetestet.

Der Monitor wird beim DAI nach einem Reset oder nach dem Einschalten über die Adresse #C818 aufgerufen. Andere Einsprungadressen (#C80C, #C814 und #C823) werden nach Beendigung bestimmter Routinen benutzt. Dies wird jedoch im Teil 2 beschrieben.

Initialisierung (#C818-#C843):

Der Monitor muß initialisiert werden. Das bedeutet: es muß eine definierte Startposition hergestellt werden. Der Stackpointer wird gesetzt, alle Flags, die einen Programmlauf anzeigen (z.B. Bearbeitung von INPUT, GOSUBs, FOR Schleifen usw.) werden gelöscht, d.h. auf ihre Anfangswerte gesetzt. Außerdem werden noch die Tastatur- und Uhr Interrupts erlaubt, da sonst keine Ein- und Ausgaben gemacht werden könnten.

Eingaben aus dem Edit Buffer (#C846-#C84E):

Zuerst wird der Eingabe Schalter (Adresse #135) überprüft. Dieser Schalter bestimmt woher der Eingabetext zum Encodieren genommen werden soll (z.B. ein POKE #135,2 bewirkt, daß der Eingabetext nun aus dem Editor gelesen werden soll - nützlich um zwei Programme aneinander zu binden). Encodieren bedeutet: Den Eingabetext (also lesbares BASIC) in einen Code zu überführen, den der DAI Halb-Compiler verstehen und ausführen kann. In diesem Code wird ein Programm auch im Textbuffer abgespeichert.

Wenn jetzt der der Eingabetext aus dem Editbuffer genommen werden soll, dann wird die Steuerung durch das Programm ab #D879 durchgeführt und die Encodierung wird ab #C867 gestartet. Nachdem eine Textzeile encodiert wurde startet der Monitor das Verfahren wieder am Anfang, solange bis der Editbuffer leer ist. (Hierzu siehe auch die 'Editor Story' aus einer früheren Ausgabe.

Eingaben von der Tastatur (#C851-#C864)

Wenn der Eingabetext nicht aus dem Editbuffer, sondern von der Tastatur gelesen werden soll, dann wird der Prompt '*' auf dem Bildschirm angezeigt und die Eingabe einer BASIC Textzeile (Adresse #DD1A) wird abgewartet. Im Klartext: der DAI wartet auf eine direkt auszuführende Zeile oder eine Zeile, die im Textbuffer abgespeichert werden soll. Diese Eingabesequenz kann nur durch 'Return' oder 'Break' beendet werden. Alle eingegebenen Zeichen werden auf dem Bildschirm angezeigt - dies bedeutet insbesondere, daß dieser Text nur im Bildschirmspeicher vorhanden ist, also nicht etwa in einem besonderen Eingabespeicher.

Besondere Programm (Befehls-) Beendigungen (#C8AA-#C8B5)

Falls ein Token gefunden wurde und die entsprechende Bearbeitungsroutine (z.B. Setzen eines Punktes beim DOT) ist beendet, so wird nun überprüft, ob diese Routine normal oder mit bestimmten Angaben verlassen wurde (z.B. Fehler oä) Dieses spezielle Ende einer Routine wird durch einen Code im Akkumulator und ein gesetztes Carry-Flag angezeigt. Hierzu als Beispiel der Befehl STOP: CY (Carry)=1, A=3 Vier Codes sind hier möglich:

0: #C908-#C915: Nach 'LOAD': Falls der LOAD Befehl vom Programm durchgeführt wird, so wird das eingelesene Programm sofort gestartet.
1: #C818: Can't continue: Nach Kommandos, die Programmzeiger oder den Inhalt des Textbuffers ändern, wird wieder der Monitor initialisiert.
2: #C8C0-#C8C8: Nach einen 'Soft-Break': Dieser spezielle Break muß behandelt werden.
3: #C8B8-#C8BD: Nach einem STOP Kommando in einer Programmzeile. Die Position, wo das Programm unterbrochen wurde, muß gemerkt werden und die Meldung 'STOPPED IN LINE' wird ausgegeben. Dannach wird wie beim Soft-Break fortgefahren.

Behandlung eines 'Soft-Breaks' (#C8CB-#C8E2):

Wird während eines Programmlaufs die Break Taste gedrückt, dann wird die Meldung 'BREAK IN LINE ...' ausgegeben. Alle Pointer (Der sogenannte FRAME (#100-#115)), die relevant für den gegenwärtigen Programmstatus sind, werden auf dem Stack abgespeichert und das Flag #126, welches den Abbruch, bzw. die Unterbrechung eines Programms anzeigt, wird gesetzt, bevor der Monitor neu gestartet wird. Falls Break während der Bearbeitung einer direkt auszuführenden Programmzeile gedrückt wird, so erscheint nur ein CR als Ausgabe. Der Programmstatus wird in diesem Fall natürlich nicht abgespeichert.

Zusammenfassung der Hauptschleifen des Monitors:

Nachdem der Monitor initialisiert ist und ein Programm eingegeben wurde, so passiert nach Eingabe eines 'RUN' folgendes:

- 'RUN' ist ein direkt Kommando, welches in den EBUF encodiert wird. Über den Token (#87) des RUN Befehls wird die Ausführungsroutine ab Adresse #DF9E gefunden. Hier wird neben einigen anderen Dingen auch noch das Registerpaar BC auf die Startadresse des EBUFs gesetzt und der Monitor wird wieder bei Punkt e) angesprungen.
- Über die Hauptschleife 1 wird das erste Zeichen des Textbuffers genommen. Dies ist das Längenbyte der ersten Programm Zeile. Falls kein TRACE oder STEP Flag gesetzt ist kehrt der Monitor mit den dem erhöhten Registerpaar BC am Punkt b) zurück.
- Über die Hauptschleife 2 arbeitet der Monitor weiter. Falls ein Token hinter der Zeilennummer gefunden wurde, so wird das entsprechende Kommando ausgeführt und der Monitor macht danach ab Punkt b) weiter
- Dies wird solange durchgeführt, bis das Ende des Textbuffers - eine '0' - gefunden wird. In diesem Fall ruft sich der Monitor selbst auf (Initialisierung usw.)

BASIC Monitor Teil 2

Autor: J.Boerrighter

In der letzten Ausgabe der Clubzeitung wurde der Teil 1 dieser Beschreibung veröffentlicht. Der letzte Artikel beschrieb die Arbeitsweise des Monitors auf allgemeiner Ebene. Dieser zweite Beitrag soll einige Funktionen des Monitors genauer erklären.

Initialisierung (#C80C-#C843):

Der Monitor wird über unterschiedliche Einsprungadressen aufgerufen. Der normale Einsprung ist die Adresse #C818. Diese Adresse wird nach Reset, dem Encodieren einer BASIC Programmzeile und nach dem END Kommando in einem Programm angesprungen.

Andere Einsprungadressen von #C818 sind #C80C und #C814. Beides sind Adressen zu denen der Monitor nach der Bearbeitung spezieller Ereignisse springt.

#C80C: Rücksprung nach einem 'Hard-Break': Die Meldung '*** BREAK) wird ausgegeben bevor der Monitor wieder gestartet wird.

#C814: Rücksprung nach einem 'Run-Time-Error' - Die Tastatur wird wieder als Eingabegerät definiert

#C818: Da hier keine besonderen Handlungen vorangegangen waren, wird der Monitor komplett neu initialisiert: Der Stackpointer wird auf #F900 (oberste Stapel Adresse) gesetzt und dieser Wert wird im Zeiger #127 abgespeichert. Dieser Zeiger ist z.B. für ein 'CONT' nach einem Soft-Break wichtig. Das Flag, um ein unterbrochenes Programm anzuzeigen, wird gelöscht, da zu diesem Zeitpunkt (noch) keine Programmunterbrechungen existieren können.

Eine weitere Einsprungadresse ist #C823. Diese wird vom Monitor selbst benutzt nach Beendigung der Ausführung eines Direkt-Kommandos, nach einem Soft-Break, STOP usw.

#C823 Der Stackpointer wird auf den in #127 gespeicherten Wert gesetzt. Beim Aufruf von #C818 oder vorher hat dies keinen Effekt, aber wenn der Lauf eines BASIC Programms durch einen Soft-Break abgebrochen wird, dann ist dieser abgespeicherte Stapelzeiger wichtig, um die spätere Fortsetzung des Programms zu ermöglichen.

#C827: Alle Zeiger, die irgentwelche Aktivitäten signalisieren werden gelöscht: (BASIC V1.0)

#100: Aktuelle Zeilennummer. Dieses ist die Zeilennummer eines BASIC Programms dessen Abarbeitung der Monitor gerade bearbeitet.

#104: Zeiger auf die gegenwärtige Schleifenvariable. Während der Bearbeitung einer FOR-NEXT Schleife wird hiermit auf die aktuelle Schleifenvariable gezeigt.

#113: Wert des Stacks vor dem Aufruf des letzten GOSUB Befehls. Wird benutzt, um nach der Abarbeitung den Zustand vor dem Aufruf des GOSUBs wiederherzustellen.

#122: Flag zum Encodieren einer Zeile. Dieses Flag zeigt an, daß der Monitor z.Z. mit dem Encodieren beschäftigt ist.

#117: Flag zum Anzeigen, daß gerade eine INPUT Anweisung bearbeitet wird.

#118: Flag, welches während der Abarbeitung eines BASIC Programms gesetzt wird.

#3E731 (abhängig von #135). Über #C024 wird die Zeilennummer (im ASCII Code) in den MACC (=Mathematik-Akkumlator) in Binärer Form eingelesen. Falls die Zeilennummer ungleich Null und kleiner #FFFF ist, dann wird die Nummer in die ersten zwei Bytes des EBUFs gebracht und HL wird um 2 erhöht. Falls die Zeilennummer nicht korrekt war, so wird die Fehlermeldung 'NUMBER OUT OF RANGE' ausgegeben, das Encodieren wird abgebrochen und der Monitor neu gestartet.

#C91D: Falls in der Eingabezeile die Zeilennummer nur noch von einem CR gefolgt ist, dann wird die entsprechende Zeilennummer im Textbuffer über die Routine #C9A2 gelöscht und das Encodieren ist beendet. Nach der Zeilennummer folgen noch Anweisungen und diese werden wie folgt encodiert.

#C929 Das D-Register wird mit der Maske #40 geladen. Diese Maske wird benutzt, um nur noch Kommandos zuzulassen, die auch tatsächlich in einem Programm auftauchen können (so sind z.B. EDIT oder UT nur als direkt Kommandos zulässig). Diese Maske lautet für Kommandos, die nur direkt ausgeführt werden können #80 (siehe hierzu #C86D) Die Tabelle mit den Befehlen befindet sich übrigens in den Adressen ab #CBBF

#C92B: Nun wird die Programmzeile durch einen CALL nach

#C93C: encodiert. Als erstes wird der aktuelle Stackpointer in #11D abgespeichert. Dies ist wichtig, um zum Punkt zurückkehren zu können bei dem das Encodieren begann, falls beim Encodieren der Zeile ein Fehler auftreten sollte. Dannach wird das Flag #122 gesetzt, um anzuzeigen, daß der Monitor z.Z. mit dem Encodieren der Zeile beschäftigt ist und das Encodieren wird nun über einen RST1/00 durchgeführt: Ein Aufruf der Adresse

#3E024 ROM Bank 3, Adresse #E024. Nachdem nun die Adresse, nach der nach dem Beendigen des Encodierens gesprungen werden soll, auf dem Stack abgelegt wurde, sucht diese Rouine in der

#3E031 Tabelle ab #CBBF nach einem BASIC Commando das dem in der Eingabezeile entspricht. Wenn es gefunden ist wird der Code des Zeichens

#3E035 mit der Maske aus dem D-Register verglichen um herauszufinden, ob es sich um ein zulässiges Commando handelt. Ansonsten wird 'COMMAND INVALID' ausgegeben.

#3E040 Der Code wird nun in das Token überführt

4 — FARBMODUS — ASCII

```
1      REM .  Farbige Zeichen im 4-Farbmodus
5      REM .  Farbregister mit Farben belegen
9      COLORT 8 Ø 3 14
1Ø     REM
11     REM . Hintergrundfarbe Vordergrundfarbe HF VF
12     REM .         1                 2        3  4
13     REM
18     REM . 2 Zeilen auf 4-Farbmodus setzen
19     REM . Zeile 2
2Ø     POKE #B4F1,#7A
21     REM . Zeile 3
22     POKE #B577,#7A
23     REM
24     REM
25     REM . Zeile 2 mit gewaehlter Farbe und Zeichen
26     REM . belegen
27     REM
28     REM . Schleife fuer jede Position der Zeile 2
29     FOR X=#B4FØ-132.Ø TO #B4FØ-2.Ø STEP 2.Ø
3Ø     REM
31     REM . Hintergrundfarbe belegen
32     POKE X,#FF
33     REM
34     REM . Zeichen setzen, #41 = A
35     POKE X+1,#41
36     REM
37     REM . naechste Position der Zeile
38     NEXT X
39     REM
4Ø     REM . fuer Zeile 3 der selbe Vorgang
41     REM
42     FOR X=#B576-132.Ø TO #B576-2.Ø STEP 2.Ø
43     POKE X,#FF
44     POKE X+1,#42
45     NEXT X
46     REM
47     REM . Zeile 4 mit Farben 1 und 2 belegen
48     REM
49     POKE #B5FD,#7A
5Ø     FOR X=#B5FC-132.Ø TO #B5FC-2.Ø STEP 2.Ø
51     POKE X,Ø
52     POKE X+1,#43
53     NEXT X
*
```

ZUSAETZL.FARBE IM 4FM-ASCII

```
1      REM .  zusaetzliche Farbe im 4-Farbmodus
2      REM .  Farbregister mit Farben belegen
3      REM .  waehlen der 4 Farben A, B, C, D
4      COLORT 8 Ø 3 14
5      REM
6      REM . Hintergrundfarbe Vordergrundfarbe HF VF
7      REM .         1                2        3  4
8      REM
9      REM . Zeile auf 4-Farbmodus setzen
1Ø     REM . Zeile 3
11     POKE #B577,#7A
12     FOR X=#B576-132.Ø TO #B576-2.Ø STEP 2.Ø
13     POKE X,#FF
14     POKE X+1,#2A
15     NEXT X
19     REM . Zeile 2
2Ø     POKE #B4F1,#7A
21     REM . Farbe D = # F
22     POKE #B4FØ,#FF
23     REM
24     REM
25     REM . Zeile 2 mit gewaehlter Farbe und Zeichen
26     REM . belegen
27     REM
28     REM . Schleife fuer jede Position der Zeile 2
29     FOR X=#B4FØ-132.Ø TO #B4FØ-2.Ø STEP 2.Ø
3Ø     REM
31     REM . Hintergrundfarbe belegen
32     POKE X,#FF
33     REM
34     REM . Zeichen setzen, #41 = A
35     POKE X+1,#41
36     REM
37     REM . naechste Position der Zeile
38     NEXT X
39     REM
4Ø     REM . Zeile 1 auf 4-Farbmodus und alte Farbe D=14
41     REM . (=#D) einsetzen
42     REM
43     POKE #B46B,#7A
44     REM
45     REM . alte Farbe
46     POKE #B46A,#FD
47     REM
48     REM . wie Anweisungszeile 28
5Ø     FOR X=#B46A-132.Ø TO #B46A-2.Ø STEP 2.Ø
51     POKE X,#FF
52     POKE X+1,#42
53     NEXT X
54     REM
55     REM . Zeile Ø mit Farben A und B belegen
56     REM
57     POKE #B3E5,#7A
58     FOR X=#B3E4-132.Ø TO #B3E4-2.Ø STEP 2.Ø
59     POKE X,Ø
6Ø     POKE X+1,#43
61     NEXT X
*
```

IB2

MISCHFARBEN IN H-GRAFIK

```
1       REM . Mischfarben in H-Grafik
2       REM . Vordergrundfarbe/Hintergrundfarbe
3       REM
4       A=#Ø
5       REM
6       PRINT CHR$(12)
8       REM
9       REM . setze 15 Zeilen auf H-Grafik
1Ø      REM
11      FOR Z=#B3E5 TO #BC45 STEP 134
12      POKE Z,#BA
13      REM
14      REM . jede Position der Zeile abwechselnd V/H-Farbe
15      REM . (#BB = Binaer 1Ø111Ø11)
16      FOR X=Z-133 TO Z-3 STEP 2
17      IF A>#FF THEN A=#1Ø:RESTORE
18      POKE X,A
19      POKE X+1,#BB
2Ø      NEXT X
21      REM
22      REM . erhoehe Hintergrundfarbe
23      A=A+1
24      REM
25      REM . naechste Zeile
26      NEXT Z
27      REM
28      REM . waehle Vordergrundfarbe
29      IF A>#F2 THEN RESTORE
3Ø      READ A
31      REM . neuer Ablauf
32      REM
33      REM . Auswertung Anzeige Vordergrundfarbe
34      PO=A/#1Ø+48:IF PO>57 THEN PO=PO+7
35      REM
36      REM . Poke Cursor
37      POKE #75,PO:GOTO 9
38      REM
39      DATA #1Ø,#2Ø,#3Ø,#4Ø,#5Ø,#6Ø,#7Ø,#8Ø,#9Ø
4Ø      DATA #AØ,#BØ,#CØ,#DØ,#EØ,#FØ
*
```

528*256 BILDPUNKTE

```
1       REM . Hoechste Aufloesung mit 528*256 Bildpunkten
2       REM . schreibe in 256 Zeilen die noetigen Startbe-
3       REM . dingungen
4       PRINT CHR$(12):CURSOR Ø,Ø
5       FOR Z=#BFEF TO #BFEF-(256*134) STEP -134
6       POKE Z,#AØ
7       POKE Z-1,64
8       REM . schreibe in jede Position einer Zeile Farbe
9       REM . und Bildbyte
1Ø      FOR X=Z-133 TO Z-3 STEP 2
11      POKE X,#AØ
12      POKE X+1,#FF
13      REM
14      REM . naechste Position
15      NEXT X
16      REM
17      REM . naechste Zeile
18      NEXT Z
19      REM
```

KLEINE CHARAKTER

```
10    REM SMALL CHARACTERS  / F.H. DRUIJFF 2/82
20    MODE 0:CLEAR 2000:DIM K(4,1),A(6,2),X(6,2):PRINT CHR$(12)
25    FOR I=0 TO 24:POKE #BFEF-I*#86,#7F:NEXT
30    PRINT "KLEINE CHARAKTER IM TEXTMODE."
40    PRINT "============================="
50    PRINT :PRINT "DIES SIND DIE NORMALEN CHARAKTER FUER DAI."
60    PRINT "AAAAAAAAAAAAXXXXXXXXXXXXAAAAAAAAAAAAXXXXXXXXXXXXAAAAAAAAAAAA"
70    CURSOR 0,9:PRINT "UND DIESES SIND KLEINERE CHARAKTER":CURSOR 0,1
80    FOR I=#BFEF-20*#86 TO I-4*#86 STEP -86:POKE I,#7F:NEXT
90    FOR I=#BFEF-6*#86 TO I-6*#86 STEP -#86:POKE I,#30:NEXT
100   FOR J=0 TO 2:FOR I=0 TO 6:READ A(I,J):NEXT:NEXT
110   FOR J=0 TO 2:FOR I=0 TO 6:READ X(I,J):NEXT:NEXT
120   GOSUB 200:GOSUB 300:GOSUB 200:GOSUB 300:GOSUB 200
130   FOR I=#BFEF-15*#86 TO I-4*#86 STEP -#86:POKE I,#30:NEXT
140   POKE #BFEF-13*#86,#7F
150   FOR J=0 TO 1:FOR I=0 TO 4:READ K(I,J):NEXT:NEXT
160   T=0:FOR L=0 TO 1:FOR J=0 TO 1:GOSUB 400:NEXT:NEXT:J=0:GOSUB 400
190   END
200   FOR K=0 TO 3:FOR J=0 TO 2:T=T+2:FOR I=0 TO 6
210   B=#BFEF-(6+I)*#86-6-T:POKE B,A(I,J):NEXT:NEXT:NEXT:RETURN
300   FOR K=0 TO 3:FOR J=0 TO 2:T=T+2:FOR I=0 TO 6
310   B=#BFEF-(6+I)*#86-6-T:POKE B,X(I,J):NEXT:NEXT:NEXT:RETURN
400   FOR K=0 TO 11:T=T+2:FOR I=0 TO 4
410   B=#BFEF-(15+I)*#86-6-T:POKE B,K(I,J):NEXT:NEXT:RETURN
500   DATA #20,#51,#8A,#8A,#FB,#8A,#8A
510   DATA #82,#45,#28,#28,#EF,#28,#28
520   DATA #08,#14,#A2,#A2,#BE,#A2,#A2
530   DATA #8A,#8A,#51,#20,#51,#8A,#8A
540   DATA #28,#28,#45,#82,#45,#28,#28
550   DATA #A2,#A2,#14,#08,#14,#A2,#A2
560   DATA #44,#AA,#AA,#EE,#AA
570   DATA #AA,#AA,#44,#AA,#AA
*
```

CURSORKONTROLLE

```
10    REM CURSORCONTROL / F.H. DRUIJFF 1/82
15    COLORT 8 0 14 0:POKE #75,#FF:POKE #74,0
20    PRINT CHR$(12)
30    PRINT "CURSOR KONTROLLE"
40    PRINT "================":PRINT
50    PRINT "DIESES PROGRAMM ZEIGT DIE GUTE CURSORKONTROLLE"
60    PRINT "DES DAI PERSONAL COMPUTERS."
70    PRINT "DER BILDSCHIRM WIRD MIT ZUFALLERZEUGTEN ZEICHEN"
80    PRINT "AN BELIEBIGEN CURSORSTELLEN GEFUELLT."
90    PRINT :PRINT "TYPE SPACE, WENN GELESEN ."
100   IF GETC<>32 GOTO 100
110   PRINT CHR$(12)
120   FOR I=0 TO 100:CURSOR RND(60),RND(24)
125   PRINT CHR$(RND(256)):NEXT
130   T$="HAT DER DAI EINE GUTE CURSORKONTROLLE ?"
140   S$="SIND SIE UEBERZEUGT":PRINT CHR$(12)
150   G=LEN(T$)-1:FOR L=18 TO 6 STEP -2:FOR I=G TO 0 STEP -1
160   CURSOR I,L:PRINT MID$(T$,I,1);:NEXT
170   FOR I=0 TO G:CURSOR I,L-1:PRINT MID$(T$,I,1);:NEXT
180   NEXT:G=LEN(S$)-1
190   FOR L=45 TO 57 STEP 2:FOR I=G TO 0 STEP -1
200   CURSOR L,I:PRINT MID$(S$,G-I,1);:WAIT TIME 1:NEXT
210   FOR I=0 TO G:CURSOR L+1,I:PRINT MID$(S$,G-I,1);
215   WAIT TIME 1:NEXT
220   NEXT:CURSOR 0,2
225   PRINT "TYPE SPACE fuer Wiederholung,  STOP MIT 'BREAK'.          "
230   IF GETC<>32 GOTO 230:GOTO 110
*
```

```
1       REM Ein Farben und Sound Demo von k.frerichs
2       ENVELOPE 0 15
3       GOSUB 100:GOSUB 200:GOSUB 10:GOSUB 19:GOSUB 23
4       GOSUB 26:GOSUB 29:GOSUB 33:GOSUB 39:GOSUB 45:GOSUB 51
8       GOTO 70
9       REM ERST DER KREIS *******
10      MX%=XMAX/2:MY%=YMAX/2:R=100.0:X0%=MX%:Y0%=MY%+100
11      FOR I=0.0 TO PI+PI STEP PI/15.0:X%=R*SIN(I):Y%=R*COS(I
        ).
12      SOUND 0 0 15 0 FREQ(2000.0):SOUND 1 0 15 0 FREQ(2001.0
        ):SOUND 2 0 15 0 FREQ(2002.0)
13      SOUND 0 0 15 2 FREQ(600.0):SOUND 1 0 15 2 FREQ(601.0):
        SOUND 2 0 15 2 FREQ(602.0)
14      DRAW X0%,Y0% MX%+X%,MY%+Y% F1%:DRAW X0%-1,Y0%-1 MX%+X%
        ,MY%+Y% F%
15      DRAW X0%+1,Y0%+1 MX%+X%,MY%+Y% F1%:DRAW X0%,Y0% MX%+X%
        +1,MY%+Y%+1 F%
16      DRAW X0%,Y0% MX%+X%-1,MY%+Y%-1 F1%:DRAW X0%-2,Y0%+2 MX
        %+X%-2,MY%+Y%+2 F%
17      X0%=MX%+X%:Y0%=MY%+Y%:NEXT:SOUND OFF :RETURN
18      REM JETZT DER INHALT ********
19      GOSUB 90:FOR I%=100 TO 235 STEP 2
20      FOR MU%=1 TO 20:FOR MU1%=4 TO 0 STEP -2:POKE #FC00+MU1
        %,MU%:NEXT:NEXT:SOUND OFF
22      DRAW I%,80 I%,180 F%:NEXT:RETURN
23      GOSUB 90:FOR I%=85 TO 175 STEP 3
24      FOR MU%=1 TO 4:FOR MU1%=0 TO 25:MU2%=MU1% MOD 6+2:POKE
         #FC00+MU2%,MU1%+MU2%:NEXT:NEXT:SOUND OFF
25      DRAW 105,I% 230,I% F1%:NEXT:RETURN
26      GOSUB 90:FOR I%=110 TO 225 STEP 3
27      FOR MU%=9 TO 0 STEP -1:FOR MU1%=25 TO 0 STEP -1:POKE #
        FC00,MU%+MU1%:NEXT:NEXT:SOUND OFF
28      DRAW I%,90 I%,170 F2%:NEXT:RETURN
29      GOSUB 90:FOR I%=95 TO 165 STEP 2
30      FOR MU%=0 TO 15:FOR MU1%=0 TO 22:POKE #FC00,MU%+MU1%:N
        EXT:NEXT:SOUND OFF
31      DRAW 115,I% 220,I% F%:NEXT
32      WAIT TIME 20:RETURN
33      GOTO 35:REM FARBWECHSEL 1 *******
34      COLORG 0 5 7 10:F%=7:F1%=10:F2%=5
35      FOR I%=1 TO 150:T0%=30:FOR MU%=#FC00 TO #FC20:T0%=T0%+
        6:POKE MU%,T0%:NEXT:SOUND OFF
36      F%=F%-1:IF F%<0 THEN F%=9
37      COLORG F% F% F% F2%
38      NEXT:RETURN
39      GOTO 41:REM FARBWECHSEL 2 *********
40      COLORG 0 5 7 10:F%=10:F1%=5:F2%=7
41      FOR I%=1 TO 50:NOISE 0 15:SOUND 0 0 15 0 FREQ(870.0):S
        OUND 1 0 15 3 FREQ(840.0):WAIT TIME 2:SOUND OFF
42      F1%=F1%-2:IF F1%<0 THEN F1%=14
43      COLORG F% F% F1% F%
44      NEXT:RETURN
45      GOTO 47:REM FARBWECHSEL 3 *********
46      COLORG 0 5 7 10:F%=0:F1%=10:F2%=5
47      FOR I%=1 TO 30:FOR MU%=1 TO 3:FOR MU1%=0 TO 12:MU2%=MU
```

```
      1% MOD 6+2:POKE #FC00+MU2%,MU%+MU1%:NEXT:NEXT:SOUND OF
      F
48    F2%=F2%-1:IF F2%<1 THEN F2%=13
49    COLORG F% F% F2% F%
50    NEXT:RETURN
51    GOTO 53:REM FARBWECHSEL 4 ********
52    REM COLORG 0 5 7 10:F=5:F1=7:F2=10
53    FOR I%=1 TO 30:FOR MU%=20 TO 0 STEP -2:FOR MU1%=20 TO
      0 STEP -2:POKE #FC00,MU%+MU1%:NEXT:NEXT:SOUND OFF
54    GOSUB 90
63    COLORG F1% F% F% F2%
64    WAIT TIME 6:NEXT:RETURN
65    GOSUB 200
70    GOSUB 90:TA%=GETC:REM Wiederholung der zeichnung und f
      arbwechsel
71    IF TA%=49 THEN T%=1:GOTO 80
72    IF TA%=50 THEN T%=2:GOTO 80
73    IF TA%=51 THEN T%=3:GOTO 80
74    IF TA%=52 THEN T%=4:GOTO 80
75    IF TA%=53 THEN T%=5:GOTO 80
76    IF TA%=54 THEN T%=6:GOTO 80
77    IF TA%=55 THEN T%=7:GOTO 80
78    IF TA%=56 THEN T%=8:GOTO 80
79    IF TA%<49 OR TA%>56 THEN 70
80    ON T% GOSUB 19,23,26,29,34,40,46,52
81    GOTO 70
90    F%=F%+1
91    IF F%<0 THEN F%=F1%+5
92    IF F%>15 THEN F%=13
93    F1%=F1%-1
94    IF F1%<0 THEN F1%=F1%+3
95    IF F1%>15 THEN F1%=12
96    F2%=F2%-1
97    IF F2%<0 THEN F2%=F1%
98    IF F2%>15 THEN F2%=13
99    COLORG F0% F% F1% F2%:RETURN
100   MODE 0:PRINT CHR$(12)
110   PRINT :PRINT :PRINT "      Nach dem letzten Farbwechsel
      ,wenn das Bild steht,"
120   PRINT " kann durch druecken der Tasten 1 - 8 jeder Dur
      chgang  "
130   PRINT " einzeln gestartet werden.  "
140   PRINT " Man kann auch mit < RUN 65 > starten ."
150   PRINT "        Dann  Taste          1  -  8  "
160   PRINT "   Viel Spass"
170   IF GETC=0.0 THEN 170
180   RETURN
200   MODE 6:COLORG 0 5 7 10:F0%=0:F%=5:F1%=7:F2%=10:RETURN
```

DAInamic Assembler Printersteuerung

Das hier beschriebene Programm bezieht sich auf den vom belgischen Club vertriebenen Assembler.
Es erfullt drei Aufgaben:
- es ermöglicht die Ausgabe einer Hardcopy über den DCE-Bus
- nach 'H.' wird das Promptzeichen 'J' weiterhin eingerückt
- die Kommandozeilen wie 'J#L.','J#S.','JH.' u.ä. gelangen nicht mehr auf den Drucker.

E I N G A'B E:
1) Einlesen des Assemblers
2) Utility nicht verlassen
   >M2CA8 2CBE 2CA9
   >S2CA8 B7-FE B7-3 CA- BB-BC
   >S2C5E CD-0 95-0
3) >20
   >G1100 Starten des Assemblers
4) Eingabe des source codes
5) J#P. Assemblieren
6) J#U. zurück in Utility
7) >W107E 2FFF ml Assembler 107E-2FFF
   abspeichern des geänderten Assemblers

A N D E R U N G S M Ö G L I C H K E I T E N
- Der Printer hat die card address '2' (Zeile 26)
- Die Routine PRINT2 (Zeile 64) benutzt Port A für die Daten. Sie arbeitet mit dem Drucker im Handshake Verfahren. Wenn PRINT2 geändert würde, könnte die RS-232 angesprochen oder eine Warteschleife implementiert werden.
- ST.TAB (Zeile 83) enthält einige Initialisierungen für den Drucker. In dieser Form sind sie für den ITOH 8510 gedacht.
- Wenn das Programm geändert wird, ist darauf zu achten, daß es weiterhin bei #10FF endet. Dazu wäre dann die ORG-Anweisung (Zeile 16) zu ändern.

****

Ich habe noch eine Frage, und würde mich freuen, wenn sie mir jemand beantworten könnte.
Nach #L. wird ein seitenweise getrenntes Listing ausgegeben. Ich würde nun gerne die Anzahl der Leer- und der Textzeilen pro Seite verändern. Leider ist mir dies noch nicht gelungen Falls jemand helfen kann, wende er sich bitte an:

Stefan Goller
Buschweg 14
5309 Meckenheim-Merl

oder an die Redaktion.

```
002                     *
003                     * ************************************************ *
004                     *          von Stefan Goller Meckenheim            *
005                     *        Idee aus DAInamic 11 Seite 222/223        *
006                     *                Dezember 1982                     *
007                     * ************************************************ *
008                     *
009                             ORG   :115F
010 115F CD8010                 CALL  INIT      Output ueber DCE-Bus
011                     *
012                             ORG   :2C76
013 2C76 CDF510                 CALL  CHANGE    aendere Ausgabemedium
014                                             (Drucker - Bildschirm)
015                     *
016                             ORG   :107E
017                     *
018 107E                MEM     RES   1,1
019 107F                CHAR    RES   1,0
020                     *
021 1080 E5             INIT    PUSH  H
022 1081 2103FE                 LXI   H,:FE03
023 1084 36A0                   MVI   M,:A0     inits DCE-Bus
024 1086 2B                     DCX   H
025 1087 2B                     DCX   H         HL=:FE01  Port B
026 1088 3602                   MVI   M,:2      card address Drucker
027 108A 21DD02                 LXI   H,:2DD
028 108D 36C3                   MVI   M,:C3
029 108F 219710                 LXI   H,CHECK
030 1092 22DE02                 SHLD  :2DE      inits DOUTC
031 1095 E1                     POP   H
032 1096 C9                     RET
033                     *  prueft ob das auszugebende Zeichen
034                     *  zum Kommando gehoert
035 1097 E5             CHECK   PUSH  H
036 1098 F5                     PUSH  PSW
037 1099 6F                     MOV   L,A
038 109A 3A7F10                 LDA   CHAR
039 109D FE20                   CPI   ' '       ist das Zeichen ein Blank
040 109F C2A810                 JNE   LAB.PR
041 10A2 7D                     MOV   A,L
042 10A3 FE5D                   CPI   :5D
043 10A5 CAAF10                 JEQ   NO.PR
044 10A8 3A7E10         LAB.PR  LDA   MEM       gehoert Zeichen zum
045 10AB A7                     ANA   A         Kommando
046 10AC CAC310                 JZ    PRINT
047 10AF 7D             NO.PR   MOV   A,L
048 10B0 217E10                 LXI   H,MEM
049 10B3 3601                   MVI   M,1
050 10B5 FE2E                   CPI   '.'
051 10B7 C2BB10                 JNE   EXIT
052 10BA 35                     DCR   M
053 10BB 327F10         EXIT    STA   CHAR
054 10BE F1                     POP   PSW
```

```
055 10BF E1                 POP   H
056 10C0 EF                 RST   5
057 10C1 03                 DATA  3
058 10C2 C9                 RET
059                 *  Druckerausgabe ueber DCE-Bus
060 10C3 7D         PRINT   MOV   A,L
061 10C4 CDC910             CALL  PRINT2
062 10C7 7D                 MOV   A,L
063 10C8 C3BB10             JMP   EXIT
064 10CB 3200FE     PRINT2  STA   :FE00
065 10CE 3A02FE     PRLOOP  LDA   :FE02
066 10D1 A7                 ANA   A
067 10D2 F2CE10             JP    PRLOOP
068 10D5 C9                 RET
069                 * Initialisierung des Druckers
070 10D6 E5         INITPR  PUSH  H
071 10D7 F5                 PUSH  PSW
072 10D8 21EA10             LXI   H,ST.TAB
073 10DB 7E         LOOPIN  MOV   A,M
074 10DC A7                 ANA   A
075 10DD CAE710             JZ    EXINIT
076 10E0 CDCB10             CALL  PRINT2
077 10E3 23                 INX   H
078 10E4 C3DB10             JMP   LOOPIN
079 10E7 F1         EXINIT  POP   PSW
080 10E8 E1                 POP   H
081 10E9 C9                 RET
082                 * fuer ITOH 8510
083 10EA 1B41       ST.TAB  DATA  :1B,'A'   1/6 Inch Zeilenvorschub
084 10EC 1B45               DATA  :1B,'E'   Elite pitch 94 char./line
085 10EE 0F                 DATA  15        normale Zeichenbreite
086 10EF 1B                 DATA  :1B
087 10F0 4C303035           ASC   'L005'    linker Rand
088 10F4 00                 DATA  0         Ende der Daten
089                 *
090 10F5 1F         CHANGE  RAR
091 10F6 1F                 RAR
092 10F7 3F                 CMC
093 10F8 17                 RAL
094 10F9 17                 RAL
095 10FA FE03               CPI   3
096 10FC CAD610             JEQ   INITPR
097 10FF C9                 RET
098 1100                    END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************

CHANGE 10F5   CHAR   107F   CHECK  1097   EXINIT 10E7
EXIT   10BB   INIT   1000   INITPR 10D6   LAB.PR 10A8
LOOPIN 10DB   MEM    107E   NO.PR  10AF   PRINT  10C3
PRINT2 10CB   PRLOOP 10CE   ST.TAB 10EA
```

Verbessertes Renumber vom Toolkit 1

```
65070 REM *3* GET NEW FIRST LINENUMBER AND STEP
65071 CURSOR 0,10:GOSUB 65208:PRINT Q$(1.0);
65072 CURSOR 26,10:COLORT 8 0 15 0:INPUT LNB:COLORT 8 0 8 0:REM LINENUMBER NEW BEGIN
65073 REM LJB / LJA =LINE JUST BEFOR / JUST AFTER TARGET TO RENUMBER TO
65074 ADR=TXTBGN:LN=0
65075 LJB=LN:GOSUB 65192:IF LN<LNB THEN ADR=ADR+L+1:GOTO 65075
65076 IF LN<LNX AND LJB>=LPR THEN GOSUB 65192:ADR=ADR+L+1:GOTO 65076
65078 LJA=LN:IF LNB<=LJB THEN EI=2:GOSUB 65196:GOTO 65070
65080 GOSUB 65204:CURSOR 42,10:COLORT 8 0 15 0:INPUT LST:COLORT 8 0 8 0:REM LINENUMBER STEP
65082 IF LNB+(LC-1)*LST>=LJA OR LST<1 THEN EI=2:GOSUB 65196:GOTO 65070
65084 CURSOR 0,10:GOSUB 65208:PRINT Q$(1.0);:CURSOR 25,10:PRINT LNB;:CURSOR 41,10:PRINT LST

65102 CURSOR 23,8:IF IM=DM THEN PRINT "TOO MUCH, RENUMBERING NOT POSSIBLE":GOTO 65166

65124 CURSOR 23,8:PRINT IM$;SPC(27):PRINT SPC(60)

65163 IF LPR<=LJB AND LNX>=LJA THEN 65128:PRINT SPC(30);"EDIT";
65164 IF LNX<LJA THEN PRINT LNX;"-";LJA:GOTO 65128
   65 PRINT LJA;"-";LNX:GOTO 65128
```

Wenn man die obenstehenden Zeilen des Renumber-Programms vom Toolkit 1 des belgischen Clubs wie beschrieben abändert, so erhält man ein Renumber mit erweiterten Möglichkeiten. Ein Beispiel:

```
10 ...
20 ...
30 ...
40 ...
50 ...
```

Ich möchte nun den Zeilen 30 und 40 neue Nummern zuweisen. Ich muß darauf achten, daß das umnummerierte Programmstück immer noch zwischen den alten Grenzzahlen 20 und 50 liegt. Durch die obige Änderung erhält man die Möglichkeit, den durch die Zeilen 30 und 40 gegebenen Progammteil aus dem Bereich zwischen 20 und 50 herauszulösen. Man kann sich nun andere Grenzen suchen. In diesem Fall hat man folgende Möglichkeiten: Umnummerieren der Zeilen 30 und 40 in den Bereich zwischen 0 und 10, zwischen 10 und 20, zwischen 20 und 50 und zwischen 50 und 65000. Dabei sind die angegebenen Grenzen als neue Zeilennummern nicht zulässig. Wenn man nicht den Bereich zwischen 20 und 50 wählt, so erscheint, nachdem die Neunummerierung vorgenommen wurde, die Anweisung EDIT x-z auf dem Bildschirm. Dies ist auszuführen und mit 'BREAK' und 'SPACE' abzuschließen.

Stefan Goller, Meckenheim

W A S T U N ? ? ? ?

..... wenn der DAI ständig abstürzt?

Viele Besitzer besonders der alten DAI-Version kennen dieses Problem zur Genüge, und man kann sich schon manchmal schwarz ärgern, wenn der gnädige PC mitten beim Arbeiten an einem langen Programm geruht abzustürzen.
Aus diesem Grund habe ich hier einmal einen kleine Maßnamenkatalog zusammengestellt, der diesem Übel abhelfen soll.

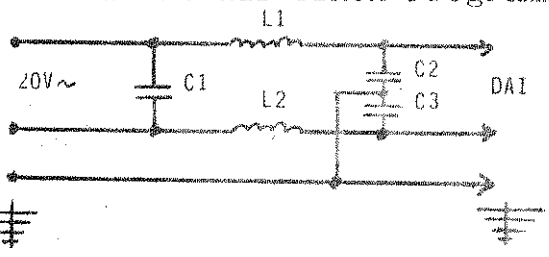
1. Verbinden der metallenen Gehäuseteile mit Masse:
Wie schon vor einiger Zeit unter HW 6 beschrieben wurde, ist es zweckmäßig die metallene Tastaturumrandung durch Anlöten eines Massedrahtes der mit TP6 (der Steckkontakt rechts oben neben der Tastatur) verbunden wird, zu erden. Es hat sich jedoch herausgestellt, daß auch ein Aufladen des metallenen Lüftungsgitters zum Absturz führen kann. Daher empfiehlt es sich mit diesem ebenso zu verfahren.

2. Isolieren der Tastaturumrandung:
Trotz Erdung kann es bei einer sehr starken statischen Entladung an der Tastaturumrandung noch zum Abstürzen des DAI kommen. Deshalb kann es nützlich sein, wenn man ihn sowieso schon offen hat, die Tastaturumrandung zu isolieren. Dazu besorgt man sich ein Stück (klares) DC-Fix. Dieses klebt man dann bei abgenommener Abdeckung so über die Tastaturumrandung, daß es noch ein gutes Stück über das Metall hinausragt. Dann schneidet man das Loch für die Tastatur frei und schlägt die losgeschnittenen Streifen nach innen um. Das Loch für den RESET-Knopf nicht vergessen. Die Folie ist,wenn sie sauber aufgeklebt wird, kaum zu sehen.
Auf eine Isolierung des Lüftungsgitters sollte wegen der nötigen Wärmeableitung verzichtet werden.

3. Erhöhen der Luftfeuchtigkeit:
Es ist natürlich am besten, wenn erst überhaupt keine statische Elektrizität entsteht. Darum sollte, wer die Möglichkeit dazu hat, zumindest während der Heizperiode einen Luftbefeuchter aufstellen. Der DAI mag's nicht gerne staubtrocken.

4. Entstören der Netzleitung:
Beim Einschalten leistungsstarker Haushaltsgeräte (Wäschetrockner, Durchlauferhitzer usw.) kann es, da das Netzteil des DAI anscheinend Störimpulse aus dem Stromnetz nicht ausreichend aussiebt,zu einem Absturz kommen. In diesem Fall kann die unten abgebildete Schaltung,wenn sie in die Netzleitung des DAI eingesetzt wird,oder noch besser,in eine Steckerleiste eingebaut wird,an der sämtliche Geräte vom Drucker bis zum Monitor hängen, sicher helfen.

Ich habe die genannten Maßnamen an meinem DAI vorgenommen und seitdem nicht einen einzigen Absturz mehr gehabt, der nicht auf einen Programmfehler zurückzuführen gewesen wäre.



R. Kietzmann

C1=0,082µF , C2=C3=0,068µF , 1000V⎓ ,400V~
L1=L2: Ringkern-Entstördrossel

Reinhard Kietzmann

D 1000 Berlin 20,den 26.3.1984
Tharsanderweg 34
Tel.:(030) 361 59 26

DAInamic PC User-Club
Werner Schmitz
Jakob-Krebs-Str. 124

4156 Willich 2

Hier ein kurzer Beitrag für die IB-Rubrik unserer Clubzeitung:

ERGÄNZUNG ZU IB 8 (Anti-Absturz-Tips)

Bei dem Entwerfen der Reset-Elektronik des DAI hatten seine Konstrukteure wiedereinmal eine Sternstunde. Beim Studieren der Schaltplähne fiel mir folgendes auf:
Beim 'Reset' wird die Reset-Leitung an Pin 2 von IC 94 über den Taster an Masse gelegt. Im normalen Betrieb jedoch ist diese Leitung völlig ohne Anschluß, was bedeutet, daß sie ganz wunderbar als Antenne für Stöhrungen aller Art wirkt. Solche Stöhrungen sind z.B. statische Entladungen am DAI und in dessen Nähe, sowie kurze Impulse aus dem Stromnetz. Abhilfe kann man durch die folgende Maßnahme erreichen, die als Punkt 0 in die Maßnahmenliste aufgenommen werden sollte:

0. Festlegen des Reset-Leitungs-Potentials:
   Um die Reset-Leitung auf ein definiertes Potential zu legen, lötet man auf der Unterseite der Platine an den Mittelkontakt des Reset-Tasters (alle anderen Lötpunkte liegen an Masse) einen Widerstand von 240 Ohm und verbindet diesen dann mit einer +5V Leiterbahn, die man am leichtesten an einem der roten Kondensatoren finden kann, weil diese auf der Platinenoberseite mit + und - markiert sind. Vorsicht, daß es auf der Platinenunterseite dabei keine Kurzschlüsse gibt.

Diese Maßnahme ist derart effektvoll, daß sie wahrscheinlich in der Mehrzahl aller Fälle zum gewünschten Erfolg der absoluten Absturzsicherheit führt, und die restlichen Maßnahmen überflüssig macht.

In Zukunft hoffendlich noch mehr Spaß mit dem DAI wünscht

R. Kietzmann

Hilfen bei der Programmierung und Adressen des Bildschirmspeichers bei 4-Farb-Grafiken

ALLE ZAHLENANGABEN SIND HEXADEZIMAL !!!!

An dieser Stelle nun einige Hinweise für das Arbeiten mit 4-Farb-Grafiken.

(1) Der COLORG - Befehl

Die Farbinformationen, die durch COLORG f1,f2,f3,f4 bestimmt werden, werden in folgenden Speicherstellen abgelegt:

1. Farbe (f1) . . . . . BFFE (#80 + f1)
2. Farbe (f2) . . . . . BFFA (#90 + f2)
3. Farbe (f3) . . . . . BFF6 (#A0 + f3)
4. Farbe (f4) . . . . . BFF2 (#B0 + f4)

Weiterhin werden f1-f4 in den Farbregistern 20-23 abgelegt. Dies kann dann z.B. vom BASIC aus direkt aufgerufen werden:

DRAW x1,y1 x2,y2 21 zeichnet eine Linie in der Farbe f2. Es wird also in Farbregister 20 die Hintergrundfarbe f1, in Farbregister 21 die Farbe f2, in Farbregister 22 die Farbe f3 und schliesslich in Farbregister 23 die Farbe f4 gesetzt.

Diese Vorteile sollten vor allem bei der BASIC-Programmierung von 4-Farb-Grafiken ausschliesslich benutzt werden, d. h. es findet sich nur noch am Anfang der Befehl: COLORG f1,f2,f3,f4 und dann im Programm nur noch die Farbregister 20, 21, 22 und 23. Dies erleichtert es, Programme auf andere Farben umzusetzen (besonders für RGB-Monitore ein absolutes Muss). Versucht einmal, konsequent damit zu arbeiten.

(2) Struktur und Anordnung der Farbbits

Die Anordnung der Farbe in den Modes 2, 2A, 4, 4A, 6 und 6A geschieht folgendermassen:
Die Farbe eines Punktes ergibt sich aus zwei Bits:

0 0 (kein Bit gesetzt) . . . . . 1. Farbe (f1, Register 20)
0 1 (rechtes Bit gesetzt). . . . . 2. Farbe (f2, Register 21)
1 0 (linkes Bit gesetzt) . . . . . 3. Farbe (f3, Register 22)
1 1 (beide Bits gesetzt) . . . . . 4. Farbe (f4, Register 23)

Diese zwei Bits sind parallel in zwei Bytes angeordnet, d. h. Sie finden hier folgende Struktur:

	Bit 0.	1.	2.	3.	4.	5.	6.	7.
High Byte (rechtes Byte)	X0	X1	X2	X3	X4	X5	X6	X7
Low Bytes (linkes Byte)	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
	0. Bit	1.	2.	3.	4.	5.	6.	7.

Es bestimmen nun jeweils Xn und Yn die Farbe eines Punktes. Man kann also feststellen: Jeweils zwei Bytes bestimmen 8 Punkte.

Vor jeder Zeile stehen zwei Kontroll-Bytes, dann folgen acht Punkte, die vom BASIC aus nicht beschrieben werden können; es folgen 42 x 8 = 336 Punkte (vom BASIC) und erneut acht Punkte, die nicht durch das BASIC erreicht werden.

Die Farbinformationen stehen übrigens auch noch einmal am Anfang des Speicherbereichs, sind aber nicht wirksam.

Im A-Modus werden die oberen Bildschirmzeilen im Speicher nach unten (hinter die neuen Textzeilen) verschoben, die anderen Zeilen entsprechend nach oben (siehe Tabellen).

Speicherbelegung bei den 4-Farb-Grafik-Modes

MODE 6: Speicherbereich von 65E0 bis BFFF

Y-Pkt	Kontroll-bytes	8 Punkte vor BASIC	42x8=336 Punkte für BASIC 42x2=84 Bytes	8 Pkt nach BASIC
255	BFEF BFEE	BFED BFEC	BFEB BFEA . . . BF99 BF98	BF97 BF96
254	BF95 BF94	BF93 BF92	BF91 BF90 . . . BF3F BF3E	BF3D BF3C
.	. . .	. . .	. . .	. . .
.	. . .	. . .	. . .	. . .
0	6649 6648	6647 6646	6645 6644 . . . 65F3 65F2	65F1 65F0
Berechnung der 1. Kontrollbytes: KONTROLLBYTE = #BFEF - 90 * (255 - ZEILENNUMMER)				

MODE 6A: Speicherbereich von 63B8 bis BFFF

BFFF - BFF0	Farbinformationen für die Grafik
BFEF - BF96 75C1 - 7568	Zeile 211 bis (Kontrollbytes und Punkte) Zeile 0
7567 - 7558	Farbinformation für den Text
7557 - 7402 74D1 - 744B - 73C5 - 7340	4. Textzeile 3. Textzeile 2. Textzeile 1. Textzeile
733F - 7330	(Farbinformation)
732F - 72D6 6411 - 63B8	Zeile 255 bis Zeile 212

MODE 4: Speicherbereich von A884 bis BFFF

Y-Pkt	Kontroll-bytes	8 Punkte vor BASIC	20x8=160 Punkte für BASIC 20x2=40 Bytes	8 Pkt nach BASIC
129	BFEF BFEE	BFED BFEC	BFEB BFEA . . . BFC5 BFC4	BFC3 BFC2
128	BFC1 BFC0	BFBF BFBE	BFBD BFBC . . . BF97 BF96	BF95 BF94
.	. . .	. . .	. . .	. . .
.	. . .	. . .	. . .	. . .
0	A8C1 A8C0	A8BF A8BE	A8BD A8BC . . . A897 A896	A895 A894

Berechnung der 1. Kontrollbytes:
KONTROLLBYTE = #BFEF - 46 * (129 - ZEILENNUMMER)

MODE 4A: Speicherbereich von A65C bis BFFF

BFFF - BFF0	Farbinformationen für die Grafik
BFEF - BFC2 AD11 - ACE4	Zeile 105 bis (Kontrollbytes und Punkte) Zeile 0
ACE3 - ACD4	Farbinformation für den Text
ACD3 - AC4E AC4D - ABC8 ABC7 - AB41 - AABC	4. Textzeile 3. Textzeile 2. Textzeile 1. Textzeile
AABB - AAAC	(Farbinformation)
AAAB - AA7E A689 - A65C	Zeile 129 bis Zeile 106

MODE 2: Speicherbereich von B968 bis BFFF

Y-Pkt	Kontroll-bytes	8 Punkte vor BASIC	9x8=72 Punkte für BASIC 9x2=18 Bytes	8 Pkt nach BASIC
64	BFEF BFEE	BFED BFEC	BFEB BFEA . . . BFDB BFDA	BFD9 BFD8
63	BFD7 BFD6	BFD5 BFD4	BFD3 BFD2 . . . BFC3 BFC2	BFC1 BFC0
.	. . .	. . .	. . .	. . .
.	. . .	. . .	. . .	. . .
0	B9EF B9EE	B9ED B9EC	B9EB B9EA . . . B9DB B9DA	B9D9 B9D8

Berechnung der 1. Kontrollbytes:
KONTROLLBYTE = #BFEF - 24 * (64 - ZEILENNUMMER)

MODE 2A: Speicherbereich von B7A0 bis BFFF

BFFF - BFF0	Farbinformationen für die Grafik
BFEF - BFD8 BB0F - BAF8	Zeile 52 bis (Kontrollbytes und Punkte) Zeile 0
BAF7 - BAE8	Farbinformation für den Text
BAE7 - BA62 BA61 - B9D8 - B955 - B8D0	4. Textzeile 3. Textzeile 2. Textzeile 1. Textzeile
B8CF - B8C0	(Farbinformation)
B8BF - B8A8 B7B7 - B7A0	Zeile 64 bis Zeile 53

```
**************************************************************************
**************************************************************************
***   Im MODE 0 sind am linken  Bildschirmrand drei Zeichen    ***
***    setzbar, die nicht scrollen und vom BASIC aus nicht     ***
***     direkt ansprechbar (teilweise auch nicht sichtbar,     ***
***        abhängig vom Fernseher oder Monitor) sind.          ***
*** Sie werden durch POKE adresse,zeichen:POKE adresse-3,255   ***
***  benutzt. Hierbei wird die Farbe von den letzten beiden    ***
***  Registern des COLORT-Befehls (COLORT . . a b) bestimmt,   ***
***  wobei a die Farbe des Buchstabens und b die des Hinter-   ***
***    grundes angibt. Bei LIST, MODE 0 und PRINT CHR$(12)     ***
***  werden diese Zeichen gelöscht. Bei POKE adresse,zeichen   ***
*** ist für zeichen der ASCII-Wert einzusetzen. Die jeweilige ***
***     adresse kann folgender Tabelle entnommen werden:       ***
**************************************************************************
**************************************************************************
```

	Zeichen Farbe		Zeichen Farbe		Zeichen Farbe	
Zeile 23	#BFED	#BFEA	#BFEB	#BFE8	#BFE9	#BFE6
Zeile 22	#BF67	#BF64	#BF65	#BF62	#BF63	#BF60
Zeile 21	#BEE1	#BEDE	#BEDF	#BEDC	#BEDD	#BEDA
Zeile 20	#BE5B	#BE58	#BE59	#BE56	#BE57	#BE54
Zeile 19	#BDD5	#BDD2	#BDD3	#BDD0	#BDD1	#BDCE
Zeile 18	#BD4F	#BD4C	#BD4D	#BD4A	#BD4B	#BD48
Zeile 17	#BCC9	#BCC6	#BCC7	#BCC4	#BCC5	#BCC2
Zeile 16	#BC43	#BC40	#BC41	#BC3E	#BC3F	#BC3C
Zeile 15	#BBBD	#BBBA	#BBBB	#BBB8	#BBB9	#BBB6
Zeile 14	#BB37	#BB34	#BB35	#BB32	#BB33	#BB30
Zeile 13	#BAB1	#BAAE	#BAAF	#BAAC	#BAAD	#BAAA
Zeile 12	#BA2B	#BA28	#BA29	#BA26	#BA27	#BA24
Zeile 11	#B9A5	#B9A2	#B9A3	#B9A0	#B9A1	#B99E
Zeile 10	#B91F	#B91C	#B91D	#B91A	#B91B	#B918
Zeile 9	#B899	#B896	#B897	#B894	#B895	#B892
Zeile 8	#B813	#B810	#B811	#B80E	#B80F	#B80C
Zeile 7	#B78D	#B78A	#B78B	#B788	#B789	#B786
Zeile 6	#B707	#B704	#B705	#B702	#B703	#B700
Zeile 5	#B681	#B67E	#B67F	#B67C	#B67D	#B67A
Zeile 4	#B5FB	#B5F8	#B5F9	#B5F6	#B5F7	#B5F4
Zeile 3	#B575	#B572	#B573	#B570	#B571	#B56E
Zeile 2	#B4EF	#B4EC	#B4ED	#B4EA	#B4EB	#B4E8
Zeile 1	#B469	#B466	#B467	#B464	#B465	#B462
Zeile 0	#B3E3	#B3E0	#B3E1	#B3DE	#B3DF	#B3DC
	adr	adr-3	adr-2	adr-5	adr-4	adr-7

Allgemeine Formel: adr = #BFED - (23 - Zeilennummer) * #86

D A T A

BP110983

Beim Koppeln oder MERGEn von Basic-Unterprogrammen, die beide DATA-Zeilen enthalten, kommt es anscheinend immer wieder zu Schwierigkeiten beim korrekten Einlesen der Daten.
Hier möchte ich nun eine Methode vorschlagen, die auch das Bearbeiten von DATA-Tabellen erlaubt. Sie hat in etwa die gleiche Wirkung wie ein RESTORE-Zeilennr., ist aber weitaus flexibler und 'lesbarer'.

Ist z.B. eine Hilfsroutine, die ein Maschinenprogramm ins RAM POKEt, mit einem Hauptprogramm ge'merge'd, das eigene DATAs enthält, läßt sich die im Unterprogramm angegebene Einleseroutine natürlich nicht unverändert verwenden, da sie evtl. die falschen Daten lesen würde.

Dieses Problem kann man umgehen, indem man vor jeden zusammengehörigen DATA-Block eine 'Marke' oder einen Namen setzt, der den Inhalt dieses Blocks kennzeichnet.

Beispiel:

```
10000 DATA MASCHINENPROGRAMM
10010 DATA #F5,#E5 .......
  .
  .
20000 DATA NAMEN
20010 DATA MÜLLER,MEIER,SCHULZE,....
  .
  .
30000 DATA TABELLE1
30010 DATA 0.23,67.4,2.3,.....
  .
  .
40000 DATA DEUTSCHES BUCHSTABIEREN
40010 DATA ANTON,BERTA,CAESAR,...
  .
  .
40100 DATA INTERNATIONALES BUCHSTABIEREN
40110 DATA ALPHA,BRAVO,CHARLY,DELTA,......
  .
  .
```

Ein allgemeines Unterprogramm, das den DATA-Zeiger auf einen bestimmten DATA-Block stellt, wäre z.B.:

```
2000 REM 'RESTORE BLOCK'     (BENUTZT HILFSVARIABLE A$)
2010 REM EINGANG: BLOCK$ ENTHÄLT NAMEN DES BLOCKS
2020 REM AUSGANG: DATA-ZEIGER STEHT AUF 1.WERT VON BLOCK
2030 RESTORE
2040 READ A$:IF A$<>BLOCK$ THEN 2040
2050 RETURN
```

Häufiger einzulesende Blocks sollten aus Geschwindigkeitsgründen natürlich immer möglichst am Anfang stehen!

Vor der Benutzung des Editors muß ein sicherer freier RAM-Bereich geschaffen werden. Dies geschieht z.B. durch Dimansionierung eines genügend großen Arrays oder Verschieben des Heap-Anfangs.
Danach muß der Zeiger #A2,#A3 auf den Anfang des Bereiches und der Zeiger #A6,#A7 hinter das Ende des Bereiches gesetzt werden.

Will man etwas in den Editbuffer 'eindrucken' (über POKE #131,2), so muß der Zeiger #A4,#A5 zu Beginn auch auf den Anfang gestellt werden. Er zeigt nachher jeweils auf die Position, auf die das nächste Zeichen kommt, also hinter das momentan letzte Zeichen.
Falls danach editiert werden soll, muß zum Schluß ein PRINT CHR$(); erfolgen, da die EDIT-Routine zur Ende-Erkennung immer eine Null benötigt.
Für das Editieren eines leeren Buffers kann man zu Beginn einfach ein CHR$(0); eindrucken.

Das eigentliche Editieren kann man vom BASIC aus mit Hilfe zweier kleiner Maschinenprogramme machen:

Das Erste initilalisiert den Bildschirm und muß vor dem Editieren einmal aufgerufen werden: CALLM INIT%

```
INIT    EF    RST  5
        2A    DATA #2A
        C9    RET
```

Das Zweite bringt jeweils ein Zeichen in den Buffer und auf den Schirm und wird durch CALLM CHARIN%,ZEICHEN% aufgerufen, wobei ZEICHEN% z.B. durch GETC belegt werden kann, aber nicht Null sein darf!
Zeichen > #80 werden zwar in den Buffer eingefügt, aber nicht auf dem Bildschirm dargestellt und können auch mittels CHAR DEL nicht wieder herausgelöscht werden!

```
CHARIN  F5    PUSH PSW
        23    INX  H
        23    INX  H
        23    INX  H      HL JETZT AUF ZEICHEN
        7E    MOV  A,M    ZEICHEN IN AKKU
        EF    RST  5
        2D    DATA #2D    IN BUFFER
        F1    POP  PSW
        C9    RET
```

Mit Hilfe dieser kleinen Programme kann man also nach Herzenslust editieren. Wenn man nun ins BASIC-Programm noch eine Abfrage einbaut, ob z.B. der Buffer ausgedruckt werden soll, kann man mit der folgenden Routine den Text recht schnell auswerfen:

```
1000  POKE #131,DRUCKERAN%:REM =0 FÜR SERIELL
1010  FOR I%=PEEK(#A2)+256*PEEK(#A3) TO PEEK(#A4)+256*PEEK(#A5)-2:
C           PRINT CHR$(PEEK(I%));:NEXT
1020  POKE #131,1
1030  RETURN
```

Wenn 'mal eben' ein kleiner Text geschrieben werden soll und kein wichtiges Programm im Speicher ist, hat sich die folgende Methode schon recht gut bewährt:

```
NEW
CLEAR 10000
EDIT
...schreiben... bis BREAK
Zeile 1010 direkt eintippen
```

A S S E M B L E R

Eine Einführung in Fortsetzungen von K.Plachetta

Assembler - ein Zauberwort. Bei den in der Clubzeitschriften veröffentlichten Programmen sind die besonderen (oder auch die besonders nützlichen) meistens in Assembler geschrieben und in Maschinensprache übersetzt.
Ich will versuchen, in einigen Beiträgen für die Clubzeitung eine Einführung in die Assemblerprogrammierung zu geben.
Allerdings komme ich nicht von der fachlichen Seite her, sondern arbeite mich gerade selbst in dieses Gebiet ein. Deshalb bitte ich fachkundige Leser, auf evtl. auftretende Fehler hinzuweisen bzw. diese richtigzustellen.

Warum scheint mir eine solche Artikelserie berechtigt?
Aus persönlichen Gesprächen bzw. brieflichem Kontakt weiß ich, daß nicht alle Clubmitglieder die Zeit oder die Möglichkeit haben, sich in die Assemblerprogrammierung und das Umgehen mit Maschinenprogrammen einzuarbeiten.
Besonderen Wert möchte ich möglichst bald auf kleine Anwendungen für den DAI legen, den sonst könnte man ja auch ein Lehrbuch verwenden.
Apropos Bücher: Ich verwende z.Zt. folgende zwei Bücher:

KÄFERFIBEL V (Kapitel 1-8) von Rony,Larsen,Titus und Scherrer im AT Verlag Stuttgart, ca.DM 30.-
Dies ist eine Art Heft, das aus einer Zeitschriftenserie hervorgegangen ist. Es enthält die Erklärung der einfacheren Befehle des Mikroprozessors und übungen für ein kleines Mikrocomputersystem und kurze Programmbeispiele.

8080,8085 Programmieren in Assembler von Lance A. Leventhal im te-wi Verlag München, Preis DM 59.-
Das Buch enthält auch viele Programmbeispiele.

Zunächst zur Begriffserklärung:
1. Was ist Assembler?
Mit Assembler bezeichnet man ein (übersetzungs-)Programm, das spezielle Kurzworte (Mnemonics) in den Hexadezimalcode übersetzt, den der Mikroprozessor verarbeiten kann. Diese Kurzworte sind nicht festgelegt, aber eine Normung ist sinnvoll und man verwendet meistens die Kurzworte der Herstellerfirma, beim DAI ist es ja der 8080A von Intel.
Das Ziel der Maschinenprogrammierung sind also eigentlich die Hexadezimalcodes. Diese sind aber so schlecht zu handhaben, daß man in Mnemonics (z.B. LDA,STA,MOV) programmiert und den Assembler übersetzen läßt.

2.Wie kann ein Assembler übersetzen?
Eine einfache Möglichkeit ist im Programm "Assembler System" von Helge Rebhahn (NP 11) realisiert:
In den Zeilen 414 bis 471 wird durch Vergleichen mit der Liste aller Befehle (abgelegt im Array A$(1)) geprüft, ob eine korrekte Eingabe vorliegt. Die Mnemonics (es sind insgesamt 244) sind im Array als auch in der DATA-Liste (Zeile 10 bis 195) so geordnet,daß ihre Nummer gleichzeitig den zugehörigen Hexadezimal-Befehl für den 8080A darstellt. Somit ist es ein leichtes, sie zu übersetzen und in Speicherplätze zu poken (Zeilen 480, 500 oder 520).

3.Welche Befehle gibt es für den 8080A?
Sie sind im deutschen Handbuch im 3.Teil enthalten und natürlich in den oben erwähnten Büchern. Keine Angst- von den 244 Befehlen sind nicht alle gleich wichtig.

4.über den Umgang mit Maschinenprogrammen
Um richtig in Assembler programmieren zu können, muß man Bescheid wissen über die Speicheraufteilung des verwendeten Computers. Ich werde also zunächst auf diese Fragen eingehen.
Bis zur Adresse 2EB (=747dez) sind Speicher für das Betriebssystem und Basic-Variable (siehe FW 2). Von 2EC bis 3EB (=1003dez) reicht der HEAP (siehe FW 1) nach dem Einschalten bzw. einem RESET. Lädt man ein BASIC-Programm, so speichert der Rechner es ab der Adresse 3EC. Hinter dem BASIC-Programm kommt noch ein Speicherbereich (Tabelle) für die im Programm verwendeten Variablen.

Das Maschinenprogramm muß vor dem Zugriff durch Basic (Programmspeicherbereich, Symboltabelle und Grafikbereich) geschützt werden. Hierzu wird meistens der Bereich vor dem Heap (und damit auch vor dem Programmtext) genommen. Dieser Platz muß jedoch erst geschaffen werden, da der Rechner den Heap in 2EC direkt an die Basic-Variablen anschließt.
Die Anfangsadresse des Basic-Programms (Textbuffer) steht in 29F,2A0. 29F enthält normalerweise EC (niederwertiges Byte) und 2A0 03 (höherwertiges Byte).
Die Endadresse des (listbaren) Basicprogramms (=Textbuffer) wird nicht direkt gespeichert, sondern ist um 1 kleiner als die Anfangsadresse der Variablentabelle (Symboltabelle) in 2A1 und 2A2. Diese Tabelle wird bei SAVE mit dem Textbuffer zusammen abgespeichert. 2A3 und 2A4 "zeigen" auf die erste Speicherstelle hinter der Symboltabelle. Daher werden solche Speicherstellen auch Zeiger (Pointer) genannt.

Es sollen nun 2 Fälle unterschieden werden, bei denen ein Basic-Programm auf ein Maschinenprogramm zugreift:
1.Fall: ml-Programm und Basic-Programm werden getrennt geladen.
Vor dem Laden des ml-Programms (File Typ 1,laden mit R) wird der Heap-Anfang in 29B,29C im Direkteingabemodus neu eingepoked:
POKE#29B,#0:POKE#29C,#5:NEW
Damit sind der Heapanfang auf 500 und der Programmanfang auf 500 + Größe des Heap festgelegt. Das ml-Programm kann nun in den Bereich von 2EC bis 4FF geladen werden. Wenn das nicht reibungslos geht,muß vor dem Laden des ml-Programms im UT-Mode noch Z3 eingegeben werden. Dadurch werden bestimmte Vektoren auf Standardwerte gesetzt.
2.Fall: Das ml-Programm soll vom Basic-Programm durch Poken erzeugt werden.
Hierzu muß sich das Basic-Programm samt Symboltabelle selbst verschieben. Dabei kann der Befehl CLEAR verwendet werden,weil bei einem CLEAR der Heap vergrößert werden kann und das dahinterstehende Programm ja auch verschoben werden muß.
Also: 10POKE#29B,#0:POKE#29C,#5:CLEAR1000
Dabei kann das System aussteigen. Mit der Zeile: 20 MODE 0 hört die Abarbeitung des Programms bei mir auf, es kommt ein Prompt und das Programm läuft erst bei einem erneuten RUN richtig.

Weitere Möglichkeiten:
Basic-Programm samt ml-Programm werden als File Typ 1 mit W abgespeichert, wobei das Ende der Symboltabelle aus 2A3 (low byte) und 2A4 (high byte) entnommen werden kann. Das gesamte Programm kann dann mit R wieder geladen werden.
Die Basic-Pointer (Heap-Anfang,Programmanfang,-ende und Symboltabellenanfang und -ende werden so erweitert, daß ein ml-Programm zusammen mit einem kurzen Basicprogramm (von dem es durch CALLM aufgerufen wird)als Basic-Programm abgespeichert wird.
Mit Hilfe eines Bootstrap-Loaders können Basic und ml mit einem LOAD geladen werden. (Siehe Spielekassette G4)

Ein kleines Maschinenprogramm

In MODE 5A wird der Bildschirm blau ausgefüllt, wobei der Grafikschirm zum Schluß "aussteigt", weil im Programm keine Abbruchbedingung enthalten ist. Zur Erklärung von MODE 5A sei auf FW 6 verwiesen. Der Grafikspeicher ist nach dem Einschalten mit FF (binär 11111111) gefüllt, so daß nur das Farbbyte gesetzt werden muß.
Das Farbbyte mit der niedrigsten Adresse in MODE 5A ist in 756A.

Stefan Goller, Meckenheim Februar 1984

Was sind indexverkettete Dateien?

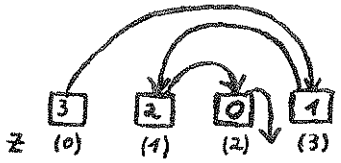
Ich möchte diese Frage anhand eines Beispieles beantworten.
Die Aufgabenstellung sei wie folgt:
Eine Menge von Personen mit den Daten Vor-, Nachname, Straße und Telefon soll nach den Nachnamen in alphabetisch aufsteigender Reihenfolge geordnet werden. Die Personen werden in ungeordneter Folge eingegeben. FD(n,5) sei ein Feld mit n*5 Elementen. Dabei ist n die maximale Anzahl der zu verwaltenden Personen. Die Fünf kommt von den fünf Daten, die es zu jeder Person geben soll.

Die Daten der ersten Person, die eingegeben wird, werden also in FD(1,1) bis FD(1,5) abgelegt. Bei konventioneller Arrayverwaltung tritt das Problem auf, daß, falls der Nachname der zweiten eingegebenen Person "kleiner" ist als der der ersten, man den Inhalt von von FD(1,1) bis FD(1,5) nach FD(2,1) bis FD(2,5) umspeichern muß. Die Daten der zweiten eingegebenen Person werden dann in FD(1,1) bis FD(1,5) abgelegt. Der Zeitaufwand für das Umspeichern wächst natürlich proportional mit der Anzahl der Datensätze, die verschoben werden sollen.

Ziel soll es sein, den Zeitaufwand zum Einfügen eines neuen Elementes konstant zu halten.Er soll also unabhängig sein von Zahl der Elemente, die bereits abgespeichert sind und von der Position, an der der neue Datensatz einzufügen ist.
Dazu wird ein zweites Feld Z(n,1) definiert. Dieses soll Zeiger auf Elemente beinhalten. Die Datensätze werden jetzt in der Reihenfolge, in der sie eintreffen, in das Feld FD geschrieben. Also in nicht sortierter Folge; die erste Person in FD(1,1..5) die Zweite in FD(2,1..5) etc. Die Reihenfolge der Elemente wird jetzt durch die Zeiger im Array Z markiert.
Bsp.: Startbelegung Z(0):=0

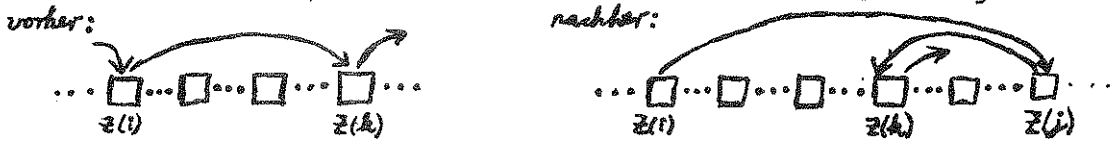
- Eingabe des ersten Datensatzes (DC). Abgespeichert wird in FD(1,1..5).
  Z(0):=1 ; Z(1):=0
- Eingabe des zweiten Datensatzes (DG). Abgespeichert wird in FD(2,1..5).
  Unter der Annahme, daß der zweite Nachname "größer" ist als der erste, folgt für die Belegung von Z: Z(1):=2 ; Z(2):=0
  Z erfüllt also folgende Auflagen:
  - Z(0) zeigt auf das kleinste Element der Liste.
  - Z(i) zeigt auf das nächstgrößere Element der Liste.
  - wenn Z(i)=0, so ist i das letzte besetzte Element.
  - wenn Z(0)=0, so ist die Liste leer.
- Eingabe des dritten Datensatzes (DB), der "kleiner" ist als DA. Abgespeichert wird in FD(3,1..5) und Z(0):=3 ; Z(3):=1.

Graphisch sieht das dann so aus:



Z enthält die Werte: Z(0)=3 ; Z(3):=1 ; Z(1)=2 ; Z(2)=0

Für jedes Element, das neu aufgenommen wird, müssen also nur noch maximal zwei Werte geändert werden.
Wenn ein Element j zwischen die Elemente i und k eingefügt werden soll, so müssen die beiden Operationen Z(j):=Z(i)=k und Z(i):=j durchgeführt werden.



Das Umspeichern von Datensätzen fällt also ganz weg.
Die Zeit, die man braucht, um einen Datensatz abzuspeichern, ist demnach im wesentlichen nur noch abhängig von der Zeit, die man braucht, um den Index des Elementes zu finden, hinter dem der neue Satz gespeichert werden soll.

BAUDRATE IN DER UTILITY

Besitzen Sie einen Drucker, der eine andere Baudrate als die Standard-Rate 9600 Baud des DAI benötigt? Haben Sie jemals versucht, eine LOOK-Sequenz auszudrucken?
Es funktionierte nicht, trotzdem Sie sicher waren, das Baudraten-Register #FFF5 des TIC mit dem korrekten Wert geladen zu haben?
Dieser Artikel eklärt, warum es nicht funktionierte und wie man dieses Problem umgeht.

Bevor Sie im UTILITY-Monitor ein 'GO'- oder 'LOOK'-Kommando starten, ist es anzuraten, vorher ein 'Z2' oder 'Z3' einzugeben.
Ohne dieses Z-Kommando gäbe es u. U. ein großes Chaos auf dem Bildschirm.

Unter Anderem initialisiert der Z2/Z3-Befehl den Timer-Interrupt-Controller (TIC-5501) und den Parallel-Interface-Chip 8255 (GIC).
Genaueres siehe ROM-Bank 3, Adresse #ECBA!

Mittels des Z-Befehls wird das Baudraten-Register des TIC auf 9600 Baud gesetzt, indem der entsprechende Wert in der RAM-Adresse #60 abgelegt wird.
Sobald ein LOOK- oder GO-Kommando gegeben wird, wird dieser Wert aus Adresse #60 genommen und in das TIC-Register #FFF5 gespeichert. (siehe #3ED9C).

Wenn Sie eine andere Baudrate wollen, müssen Sie nur den Inhalt von Adresse #60 ändern. Jedesmal, wenn ein GO- oder LOOK-Befehl läuft, wird dieser Wert genommen und im Baudraten-Register #FFF5 abgespeichert.

Es gibt folgende Möglichkeiten:

9600 Baud: #FC	300 Baud: #BC
4800 Baud: #EC	150 Baud: #AC
2400 Baud: #DC	110 Baud: #9C
1200 Baud: #CC	

Um die Standard-Baudrate zu ändern, nur einmal nach dem Einschalten oder nach RESET folgendes eingeben:

```
*UT
>Z3
>S0060 FC-**
```

wobei ** einer der oben genannten Werte ist.

(C) - Jan Boerrigter - Nov. 1983.

Übersetzt von Bernd Preusing DnDt2/84

DAI und CP/M

Vor gut einem Jahr besorgte ich mir eine billige Z80 - Rechnerkarte um damit ein wenig "herumzuspielen". Es war ein nettes, kleines Platinchen mit einer 4 MHZ CPU, 64KB RAM, 8KB ROM, CRT-Teil, Floppyteil mit Shugart kompatibler Schnittstelle, einem 8251 und einem 8255 wie im DAI. Da sich das Ganze auf nur einer Europakarte mit ECB-Bus befand, war es recht kompakt aufgebaut.

Ich schloss ein Netzteil und einen Monitor an, nahm die serielle Tastatur mit der ich bis dato meinen DAI bediente, und die Kiste lief. Als ich wieder Geld hatte, kaufte ich mir noch eines dieser neuen 1MB 5.25" Half-Height Laufwerke und konnte auch dieses (wie möglicherweise noch 3 von dieser Sorte) über ein Flachbandkabel direkt anschließen. Die hohe Arbeitsgeschwindigkeit überrachte mich als nicht gerade verwöhnten DAI-Floppy Besitzer nicht eben schlecht. Ich überlegte mir, ob es nicht sinnvoll wäre den neuen Rechner dem DAI anstelle der Floppy untertan zu machen. Da auch das DAI-DOS nicht gerade meinen Erwartungen entsprach, war ich nahe daran die Rechner zu koppeln und ein neues DOS zu schreiben.

Dann kam die Meldung von KEN-DOS und das Ganze schien mir plötzlich sinnlos. Ich hatte keine Lust " das Rad zweimal zu erfinden ". Als ich für die neue Kiste CP/M bekam und mich von den Vorzügen von Wordstar und Pascal MT+ verwöhnen ließ, wanderte mein DAI endgültig ins Eck. Nun jedoch ist der Punkt erreicht, an dem ich mich wieder nach den einmaligen Sound- und Graphikfähigkeiten meines DAI zurücksehne. Nach dem Motto : Ein gebrauchter DAI den man schon hat ist immernoch billiger als eine Farbgraphikkarte, ein Programmierbares Soundboard, ein Paddleanschluss und eine Tastatur, bin ich auf die Idee gekommen den DAI als intellegentes Farbgraphikterminal an den CP/M Rechner anzuschließen. Das Ganze stelle ich mir so vor :

- Die Verbindung der Rechner erfolgt genau gleich wie die zwischen DAI und Floppy
- Alle Konsolenausgaben die das BIOS bekommt werden direkt an den DAI geschickt und dort angezeigt oder verarbeitet
- Alle Konsoleneingaben werden von der DAI-Tastatur oder von der seriellen Schnittstelle geholt

Die Kommunikation mit dem DAI kann von CP/M aus in allen Programmiersprachen erfolgen, da sie ausschließlich über PRINT/INPUT bzw. READ/WRITE in PASCAL o.ä. abgewickelt wird. Um die besonderen Möglichkeiten (Graphik/Sound/..) des DAI nutzen zu können, wird in die PRINT-Anweisung einfach eine Steuersequenz eingefügt, die vom DAI passend interpretiert werden muß. So zum Beispiel :

```
PRINT "§DRAW ",X+1,Y+1,X1,Y1,14
```

Das § Zeichen leitet die Steuersequenz ein (man könnte auch ein beliebiges anderes Zeichen hierzu verwenden). DRAW wird als DRAW-Befehl interpretiert und die nächsten 5 Zahlen als Parameter. Die Zahlen werden folglich im ASCII-Format übertragen. Getrennt sind die Parameter durch die Blanks oder Tabs, die die PRINT-Routine normalerweise einfügt.

Vorteil :
- einfache Handhabung ohne PEEK und POKE oder andere undurchsichtige Tricks
- Ausgabeanweisungen gibt es in allen Programmiersprachen
- gleichzeitiges Arbeiten beider Rechner möglich : DAI zeichnet - der "Andere" rechnet schon die nächsten Eckpunkte aus
- Syntax der Steueranweisungen ist einfach und orientiert sich an den entsprechenden DAI-Äquivalenten
- neue Graphikprimitiven lassen sich leicht integrieren, zB. Fläche füllen

Nachteil :
- einfache Befehle sind effektiv langsamer als auf DAI-Basic, da das übertragungsprotokoll einen höheren Aufwand erfordert
- da Steuerzeichen durch ein bestimmtes Zeichen (§) eingeleitet werden, darf dieses im normalen Text nicht verwendet werden

Alle Sonderfunktionen des DAI können so leicht genutzt werden und lassen sich durch neue ergänzen.

Im Moment jedoch beschreibt dieser Artikel nur eine Möglichkeit. Bei breitem Interesse würde ich versuchen daraus ein laufendes System zu machen.
Die Kosten für ein solches System wären kaum höher als die einer guten Floppystation. Hier eine ganz grobe Kalkulation :

- Rechnerkarte im Europaformat, 4 MHZ Z80, 64KB RAM 8KB ROM, CRT-Teil 80/25, Floppyteil mit 1793, max. 4 Laufwerke sd/dd, 8251 für serielle Schnittstelle nach RS-232, 8255 für Parallel und Rechnerkoppelung. — 1000,- DM
- Gehäuse und Kleinteile — 150,- DM
- Netzteil 5V 5A, 12V 2A, -12 100mA — 150,- DM
- Laufwerk 80 Track DD/DS — 800,- DM
  (zB. TEAC) anschließbar ist im Prinzip alles von 3,5" bis 8", nur muß die Software dann daran angepasst werden
- Lizenz für CP/M — 400 - 700,- DM
- angepasstes BIOS, Urlader und für DAI nötige Soft- — < 100,- DM
  (hängt von der Arbeitszeit ab)

Die Preise sind grobe Schätzpreise für die einzelnen Komponenten. Jeder kann sie in Einzelstückzahl selbst für diesen Preis erwerben oder selber bauen (Netzteil, Rechnerkarte als Bausatz). Viele dürften auch schon ein Laufwerk oder CP/M haben oder günstig bekommen. Bei gemeinsamem Einkauf sind sicher noch Mengenrabatte herauszuschlagen. Ein Komplettzusammenbau ließe sich sicher auch organisieren, wäre aber einiges teurer.
Die gleiche Hardware läßt sich dann natürlich auch als intellegente Floppy für den DAI verwenden. Das hierzu nötige DOS zu schreiben bedeutet allerdings einen nicht unerheblichen Aufwand.
Wenn Sie Interesse daran haben, daß dieses Projekt in die Tat umgesetzt wird, dann schreiben Sie mir bitte. Legen Sie Ihre guten Ideen und Vorschläge am besten gleich bei - 10 Köpfe wissen in der Regel mehr als einer !
Da ich nur selten daheim bin (ich studiere noch), ist vom "telephonischen Weg" abzuraten.
Karlheinz Peter, Neubriach 13, 7982 Baienfurt
0751 / 51712

Feb. '84

```
**************************************************
*                                                *
*   Vorstellung des  D A I - S T A R - SYSTEMS   *  .
*                                                *
**************************************************
```

Durch die Unzulaenglichkeiten des DAI-Floppysystems seit langem genervt und von Ken-Dos-System nicht voellig ueberzeugt, kamen wir vor einiger Zeit auf die Idee, ein sinnvolles Erweiterungssystem fuer den DAI-PC zu entwerfen. Wir, das sind Karlheinz Peter, Informatik-Student in Karlsruhe, und Richard Schillinger, Physik-Student in Freiburg. Die Moeglichkeiten und Anwendungsgebiete,die sich dem Benutzer dieses Erweiterungssystems bieten, wollen wir in diesem Artikel vorstellen. Da wir selbst die Faehigkeiten des (DAI+Erweiterung)-Systems ganz ueberragend finden, haben wir es DAI-STAR-SYSTEM getauft.

I.) Motivation zum Entwurf des DAI-STAR-Systems

Es ist fuer jeden PC-Benutzer ein altbekanntes Uebel, dass er fuer die schnelle und elegante Speicherung seiner Programme und Daten fast genausoviel bezahlen muss, wie fuer den PC selbst. Ein Floppy-System ist nun aber hauptsaechlich deshalb so teuer, weil es viel Intelligenz enthaelt, die nur in den Augenblicken genutzt wird, in denen Diskettenoperationen ablaufen.
Es gibt nun 2 Moeglichkeiten ein Floppysystem zu optimieren: Entweder man verringert die so selten benutzte Intelligenz, um ein preiswerteres Geraet zu erlangen, oder aber man erweitert die im Floppy-System steckende Intelligenz und sorgt dafuer, dass sie zusaetzliche Aufgaben (ausser Diskettenhandling) uebernimmt und damit nicht stundenlang ungenutzt vor sich hinschlaeft. Da die erste beschriebene Optimierungsmoeglichkeit meist zu leistungsschwachen Systemen fuehrt, haben wir uns fuer den zweiten Weg entschieden. Unser Erweiterungsgeraet fuer den DAI sollte also nicht nur als Diskettenkontroller arbeiten, sondern noch ganz anders genutzt werden koennen: Als C P / M - R e c h n e r naemlich ! Damit kann man die Moeglichkeiten des DAI ganz unwahrscheinlich erweitern, ohne viel mehr Geld als in eine "gewoehnliche" Diskettenstation zu investieren. Zusaetzlich zur DAI-Software steht dann dem Benutzer die riesige Palette der CP/M-Software zur Verfuegung. Mit weit mehr als Tausend CP/M-Programmen aus allen Bereichen (Textverarbeitung, Kalkulation, Datenbanken, Programmiersprachen, Branchenloesungen und Werkzeugen zur Softwareentwicklung) duerfte sich fuer jeden das Richtige finden. Zur Schaffung dieser Faehigkeiten fuer den DAI wollten wir einen Grossteil der sonst brachliegenden Diskettenhandlingsintelligenz einspannen.

II.) Entwicklung des DAI-STAR-Systems

Aufgrund der obigen Ueberlegungen schlossen wir also keinen "normalen" Controllerbaustein sondern einen beinahe kompletten CP/M-Rechner mit Z80-CPU (4 MHz) und 64 kByte RAM-Speicher an den DCE-Bus des DAI an. Diese Single-Board-Rechnerplatte im Europaformat sollte je nach Betriebsart entweder "nur" das Diskettenhandling fuer den DAI uebernehmen, oder aber selbst die Hauptrolle uebernehmen, als speicherstarker CP/M-Rechner arbeiten, der den DAI-PC als Farbgraphik-Terminal, Tongenerator und Input/Output-System benutzt.

Was das DAI-STAR-System in dieser Betriebsart von einem handelsueblichen CP/M--Rechner unterscheidet, sind die Graphik- und Soundmoeglichkeiten des DAI, die in jedem CP/M-Programm verwendet werden koennen. Speziell kann man in Basic, Pascal, Fortran, C und vielen andern Programmiersprachen die ueblichen DAI-Graphik- und Soundbefehle mitbenutzen.
Die Aufgaben des DAI in dieser Betriebsart:
Der DAI dient jetzt als Sounderzeuger,Graphik-Terminal und IN/OUTPUT-System. Da die 48 kB RAM des DAI nur als Bildschirmspeicher benutzt werden, hat man bei der Arbeit mit der Rechnerkarte die ganzen 64 kB als Programm- und Datenspeicher zur Verfuegung. Kein Bit dieses Speicherpotentials hat Bildschirmbetreuungen zu uebernehmen. Man kann getrost die hochaufloesendste Graphikstufe des DAI benutzen ohne ein zu knappes Programmspeicherreservoir fuerchten zu muessen.

Da die Betriebssoftware fuer diese Betriebsart schon fertiggestellt ist und laeuft, hier einige detailliertere Informationen zu den Moeglichkeiten in dieser Betriebsart:
Zu den vorhandenen Graphik-, Sound- und I/O-Befehlen des DAI wurden neue hinzugefuegt, die an die Moeglichkeiten des DAI angepasst sind und im CP/M-Betrieb benutzt werden koennen.Folgende Befehle wurden in Betriebsart II installiert und koennen damit in jeder Programmiersprache verwendet werden.

MODE,0-8: 0-6A sind die bekannten DAI-Modes, 7,7A,8,8A sind die entsprechenden Konfigurationen fuer die hoechste Aufloesungsstufe (512*256 Punkte)

COLORG
COLORT
COLSET : Schaltet im Textmodus auf einen zweiten Farbsatz um. Alle Zeichen die dem COLSET-Befehl folgen werden mit neu definierter Vorder- und Hintergrundfarbe auf den Schirm ausgegeben, solange bis ein erneuter COLSET-Befehl die Farben auf den ersten Farbsatz zuruecksetzt.

DOT
DRAW
FILL
CIRCLE : zeichnet Kreise und Kreisboegen.
POLY : zeichnet einen Polygonzug.
AREA : fuellt eine beliebig geformte Flaeche
TEXT : in seiner Funktion dem FGT aehnlich.

SOUND	CURSOR	POKE
ENVELOPE	CURX	PEEK
NOISE	CURY	PDL
TALK	SCREEN	

SAVE : Speichert einen beliebigen Teil des DAI-Ram (Maschinenprogramme oder Bildschirminformation) wahlweise in das Ram der Rechnerkarte oder auf ein CP/M-Diskettenfile ab.
LOAD : Umkehrung zu "SAVE"

SLINE : Schaltet Statuszeile an oder ab
FORMAT : Definiert die Groesse des virtuellen Bildschirms.
WINDOW : Positioniert den realen Schirm (window) an einer bestimmten Stelle des virtuellen Schirms.

Besonderheiten des Terminalbetriebs (Betriebsart II):

1.) Tastaturabfrage

Eine neue Tastaturabfrageroutine macht die Tastatur komfortabler und sicherer. Zum einen koennen bei ausgelagerten Tastaturen laengere Zuleitungskabel verwendet werden,ohne dass Stoerungen auftreten.Eine Belegung der Tastatur auf 3 Ebenen

Das DAI-STAR-System ist fuer Leute gedacht, die einerseits ihren DAI weiterhin so verwenden wollen wie bisher, gleichzeitig aber den Wunsch haben, mit einem CP/M-Rechner arbeiten zu koennen, um zusaetzlich zur DAI-Software auf das weltweite und umfangreiche Softwareangebot des CP/M-Betriebssystems zuzugreifen. DAI-STAR-System bietet die Moeglichkeit, beides gleichzeitig zu haben: den DAI mit den fuer ihn erstellten und gekauften Programmen zusammen mit einer leistungsfaehigen Floppystation weiterhin zu verwenden und zugleich einen leistungsfaehigen CP/M-Rechner zu besitzen, der obendrein alle Graphikmoeglichkeiten des DAI voll aus schoepfen kann. Wer also jetzt oder in absehbarer Zukunft vermutet, die Betriebs art 2 des DAI-STAR-Systems verwenden zu koennen, fuer den duerfte DAI-STAR-System eine optimale Loesung sein. Fuer Leute, die mit dem Gedanken spielen, vom DAI auf ein anderes Geraet umzusteigen, ist DAI-STAR-System eine Loesung, die in Punkto Leistung keinen Vergleich mit andern PCs zu scheuen braucht und dennoch wesentlich preisguenstiger ist.

V.) Beschreibung der Systemkomponenten von DAI-STAR

A.) Die HARDWARE

1.) Rechnerkarte im Europakartenformat (16*10 cm) mit Z80 CPU (4 MHz Taktfrequenz, optional 6 MHz), mit 64 kB RAM (ohne Waitzyklen), Floppycontroller, 3 Parallelports, 8kB Eprom, wahlweise ECB-Bus.

2.) Schaltnetzteil zur Versorgung der Rechnerkarte und der Laufwerke.

3.) solides Metallgehaeuse mit Ausschnitten fuer 2 Laufwerke

4.) Flachbandkabel mit DCE-Bus-Steckern

5.) 1 MByte Laufwerke

a.) 1 Laufwerk STANDARDVERSION
b.) 2 Laufwerke
c.) Sonderanfertigungen (1-2 2MByte Laufwerke, RAM-Erweiterungskarte (64kB - 1MB), 6 MHz-CPU)

B.) Die SOFTWARE von DAI-STAR

1.) Eine Diskette mit dem Bios fuer die Zusammenarbeit von DAI und Floppysystem sowie dem DOS zum Betrieb der Erweiterung als Diskettenstation.

2.) Eine Diskette, die das Bios zur Verwendung des DAI als Terminal (inclusive aller Routinen zur Nutzung der DAI-Graphik im CP/M) sowie das lizensierte, komplette CP/M-Betriebssystem (CP/M 2.2, inclusive Formatierprogramm, Kopierprogramm (PIP), Editor (ED) und Assembler (ASM)) enthaelt.

HALLO INPUT-FANS !

MICH HAT ES GEAERGERT, DASS IN EINER INPUT-ANWEISUNG DER PROMT-TEXT NUR ALS KONSTANTE VORGEGEBEN WERDEN KANN.

ALSO : 10 INPUT"TEXT";WERT

UM INNERHALB EINER SCHLEIFE DEN TEXT ZU AENDERN, HABE ICH FOLGENDE "KONSTRUKTION" GEFUNDEN:

BEISPIEL:

```
10 FOR X=1 TO 9
20 A$="TEXT"+STR$(X)
30 INPUT""+A$;WERT
40 NEXT
```

DIE ADDITION VON NULL-STRING UND VARIABLER A$ ERGIBT DIE GEWUENSCHTE FLEXIBILITAET.

D.Sachtleben

Hier möchte ich nun noch einmal genau erklären, wie ein Array im Speicher liegt und was bei seiner Verwendung zu beachten ist.

Jeder benutzte Array muß mit dem DIM-Befehl definiert werden. So dimensioniert z.B. die BASIC-Zeile

```
100 DIM A%(100),B!(1,2,3),C$(255,1)
```

drei verschiedene Arrays im Heap (der zuvor mit dem CLEAR-Befehl groß genug gemacht werden muß) und setzt alle darin enthaltenen Variablen auf 0 bzw. den Leerstring.

Dabei sind die Zahlen in den Klammern die Indizes, die Werte von 0 bis 255 annehmen dürfen; die Anzahl der durch Kommata getrennten Indizes bestimmt die Dimension des Arrays, dies dürfen maximal 15 Werte (Ausdrücke, Konstanten, Variablen) sein.

Bei jeder Dimensionierung errechnet der DAI die Anzahl der Bytes, die der Array benötigt, reserviert den entsprechenden Platz im Heap und setzt in der Symboltabelle einen Zeiger auf den Anfang des Arrays im Heap. Dieser Zeiger zeigt auf das erste Byte, das die Anzahl der Dimensionen (1 bis 15) enthält. Die nachfolgenden Bytes stellen die Werte dar, die auch im DIM-Befehl benutzt wurden, also die Maximalwerte der einzelnen Indizes. Diese Werte benötigt der DAI, um auf jedes einzelne Element des Arrays zugreifen zu können. Danach folgen direkt die Variablen. Hierbei wird der jeweils letzte Index am schnellsten durchlaufen, d.h. auf B!(1,1,2) folgt B!(1,1,3) und dann B!(1,2,0).

Sehen wir uns nun jeden einzelnen Array aus unserem Beispiel etwas genauer an:

```
DIM A%(100)
```

Hier wird ein eindimensionales Feld aus 101 Integer-Zahlen deklariert. 101 deshalb, weil das Element 0 *immer* mitzählt! Für jede Zahl werden vier Byte reserviert, weil Integers im DAI eben genau vier Byte benötigen. Der Zeiger des Arrays zeigt dann auf Folgendes:
#01 (eindimensional) #64 (max. Index=100), darauf folgen 404 Null-Bytes (101*4 Byte).

```
DIM B!(1,2,3)
```

Dieses Feld ist dreidimensional und enthält 24 FPT-Zahlen ((1+1)*(2+1)*(3+1)). Zwischen INT- und FPT-Arrays besteht in der Speicherbelegung keinerlei Unterschied. Hier zeigt der Pointer auf:
#03 (dreidimensional) #01 #02 #03 (max.Indizes) und 4*24 Null-Bytes

```
DIM C$(255,1)
```

Das ist ein zweidimensionaler String-Array, der 256*2=512 Strings beinhaltet. Im Unterschied zu den Zahlen-Feldern wird hier nur sozusagen ein 'Pointer-Array' im Heap angelegt, d.h. im Array selbst befinden sich nicht die Strings, sondern nur ihre Zeiger. Das sieht dann in diesem Fall so aus: #02 (2-dim) #FF #01 und 512*2 Null-Bytes (weil ein Zeiger zwei Byte benötigt).
Übrigens ist ein String nicht deshalb leer, weil sein Pointer=0000 ist, sondern weil im Normalfall auf Adresse #0000 immer #00 steht. Spielfreunde sollten einmal POKE 0,16 eingeben und dann z.B. LIST oder A$=""+"":PRINT A$! (bei LIST wegen (LET))

Soll nun einem Feldelement ein Wert zugewiesen werden, so wird bei einem Zahlen-Array einfach die Speicheradresse im Array errechnet und die Zahl mit ihren 4 Byte dort abgelegt. Bei String-Arrays sieht die Sache allerdings ganz anders aus: zuerst wird im Heap ein freier Bereich gesucht, der groß genug ist, diesen String aufzunehmen, dann wird er dort abgelegt und im 'Pointer-Array' auf dem entsprechenden Platz die Adresse dieses Strings eingetragen. Dabei belegt jeder String zwei Byte + seine Länge; das erste Byte dient der Heap-Verwaltung und ist immer #00, das zweite enthält die Länge des Strings (hierauf zeigt der Pointer), dann folgen die Zeichen.
Der String-Array aus dem obigen Beispiel benötigt also, selbst wenn alle Elemente Leerstrings sind, schon 256*2*2+4=1028 Bytes im Heap. Nehmen wir an, jeder String hat im Schnitt die Länge 20, dann kommen noch einmal 512*(20+2)=11264 Bytes dazu! Und da wundern sich einige Leute, die noch höher dimensionieren, warum sie mit String-Arrays nicht zurechtkommen!

Der grundlegende Unterschied zwischen Zahlen- und String-Arrays ist also, daß Zahlen-Arrays ihre Information in einem zusammenhängenden Block gespeichert haben, String-Arrays dagegen haben ihre Inhalte kreuz und quer und eher zufällig im Heap verstreut.

Kommen wir nun zu LOADA und SAVEA:

Diese Befehle dienen dazu, Arrays abzuspeichern und wieder zu laden. Dazu brauchen die Befehle als Parameter den Array-Namen ohne Klammern und Indizes und evtl. einen File-Namen unter dem das Feld abgespeichert wird.
Beim Abspeichern werden nach dem Namen zwei Blöcke auf das Speichermedium geschrieben. Der erste Block besteht nur aus einem Byte und ist durch den Array-Typ bestimmt (#00=FPT, #10=INT, #20=STR), der zweite Block enthält die reine Information des Arrays ohne die Anzahl und Maximalwerte der Dimensionen.

Für Zahlen-Felder ist die Sache einfach: hier werden nur die Zahlen, also 4*(Anzahl Elemente) Bytes weggespeichert. Das ist deshalb so einfach, weil ja die Zahlen in ihrem Array schön beieinander liegen. Bei LOADA werden die Daten auch wieder direkt in den Heap gelesen.
String-Arrays dagegen machen es dem DAI nicht leicht: die Strings liegen im Heap nicht zusammen, sondern weit verstreut und können aus diesem Grunde nicht so einfach abgespeichert werden, vom Laden gar nicht zu reden! Hier behilft sich der Rechner mit einem Trick, der manchem Benutzer das Leben schwer macht: die Strings werden vor dem Abspeichern hinter dem Ende des BASIC-Programms zu einem Block zusammengeschoben und dann zusammenhängend gespeichert. Beim Laden wird erst der Block hinter das Ende des BASIC-Programms eingelesen und danach wird jeder String einzeln in den Heap kopiert und der Pointer im 'Pointer-Array' (der ja vor dem Laden schon dimensioniert worden sein muß) auf diesen String gesetzt. Deshalb dauert es nach dem eigentlichen Laden bei größeren Arrays recht lange, bis das BASIC sich wieder meldet. Der Block selbst ist folgendermaßen aufgebaut: die ersten zwei Bytes enthalten die Länge des Pointer-Arrays (also die Anzahl der Strings * 2) in der Reihenfolge High-Byte, Low-Byte, danach folgen die Strings: Len-Byte,String,Len-Byte,String,...
Beim Laden und Speichern von String-Arrays benötigt man also hinter dem BASIC-Programm noch einmal fast denselben Platz, den der Array im Heap verbraucht (sonst erfolgt die beliebte Fehlermeldung 'OUT OF MEMORY'). Nach dem Laden kann selbst nach einiger Zeit noch die Meldung 'OUT OF STRING SPACE' erfolgen, wenn der Rechner für den gerade einzusortierenden String keinen Platz im Heap mehr findet.
Daraus folgt: Finger weg von großen String-Arrays, lieber die Verwaltung selbst übernehmen und einen INT-Array als Platzhalter benutzen!