Assembler ist keine Alchimie

(Teil 6)

In der vorangegangenen Ausgabe haben wir die relative und die Zeropage-Adressierung kennengelernt. Heute kommt die indizierte Adressierung dran und natürlich sehen wir uns wieder einige neue Assembler-Befehle an. Wir werden uns einige Gedanken machen über die sogenannten Fließkommazahlen und den Basic-Befehl USR. Auch die Speicherorganisation unseres Computers soll uns nochmal beschäftigen.

Zunächst die indizierte Adressierung. Indizieren heißt, etwas mit einem Index, also einem Zeichen oder einer Nummer, zu versehen. Beispielsweise bezeichnet man in der Mathematik die beiden Lösungen einer quadratischen Gleichung häufig als X1 und X2. Dabei ist dann die Ziffer (1 oder 2) der Index und X ist eine indizierte Größe. Man geht also aus von einer festgelegten Grundmenge (Lösungsmenge X) und trifft durch den Index eine weitere Unterscheidung.

So ähnlich können wir uns auch die Funktion der indizierten Adressierung bei der Assembler-Programmierung vorstellen. Nehmen wir als Beispiel den Befehl

LDA 1500,X

Man spricht hier von einer absolut-X-indizierten Adressierung. Das Assemblerwort LDA ist uns bekannt: Lade den Akku. Woher soll der für den Akku bestimmte Inhalt geholt werden? Aus der Speicherzelle, die sich durch 1500 plus Inhalt des X-Registers ergibt. Steht also im X-Register zum Zeitpunkt des Befehlsaufrufes eine 5, dann wird der Akku aus Speicherzelle 1500+5, also 1505, geladen. Das X-Register kann Werte von 0 bis $FF (dez. 255) enthalten. Die Ähnlichkeit sieht also so aus:

Aus einer Gesamtmenge von 256 Adressen, die durch die Anfangsadresse (bei unserem Beispiel 1500) und die möglichen 256 Belegungen des X-Registers festgelegt sind (die Grundmenge), werden je nach X-Registerinhalt einzelne Adressen unterschieden und adressiert. Das X-Register fungiert dabei als ein Index, weswegen man auch oft die Bezeichnung »Index-Register X« in der Literatur findet.

Ebenfalls als Index-Register kann das Y-Register dienen, was zum Beispiel zum Befehl

LDX 1500,Y

führen kann. Dies ist dann eine absolut-Y-indizierte Adressierung.

Befehl	Indizierte Adressierung			
	absolut		Null-Seite-absolut	
	X	Y	X	Y
Folge 2				
LDA	+	+	+	—
LDX	—	+	—	+
LDY	+	—	+	—
STA	+	+	+	—
STX	—	—	—	+
STY	—	—	+	—
RTS	/	/	/	/
Folge 3				
INX	/	/	/	/
INY	/	/	/	/
INC	+	—	+	—
DEX	/	/	/	/
DEY	/	/	/	/
DEC	+	—	+	—
SED	/	/	/	/
CLD	/	/	/	/
BNE	/	/	/	/
Folge 4				
ADC	+	+	+	—
CLC	/	/	/	/
SBC	+	+	+	—
SEC	/	/	/	/
BEQ	/	/	/	/
BCC	/	/	/	/
BCS	/	/	/	/
BMI	/	/	/	/
BPL	/	/	/	/
BVC	/	/	/	/
BVS	/	/	/	/
Folge 5				
CMP	+	+	+	—
CPX	—	—	—	—
CPY	—	—	—	—
Folge 6				
BIT	—	—	—	—
CLV	/	/	/	/
NOP	/	/	/	/
TAX	/	/	/	/
TAY	/	/	/	/
TXA	/	/	/	/
TYA	/	/	/	/
JMP	—	—	—	—
JSR	—	—	—	—

\+ anwendbar
— nicht erlaubt
/ weder absolute noch Zeropage-Adressierung möglich

Tabelle 1. Anwendbarkeit der indizierten Adressierungsarten auf die bisher gelernten Assembler-Befehle.

Genauso wie man die normale absolute Adresse (also zum Beispiel 1500) als Basis der Indizierung durch das X- oder das Y-Register verwenden kann, ist das auch mit eine Zeropage-Adresse möglich. So gibt es zum Beispiel die Befehle

LDY 2B,X

oder

STX 19,Y

Man nennt diese Art der Adressierung dann Zeropage-absolut-X-indiziert beziehungsweise -Y-indiziert.

Weil die Zeropage aber nur 256 Adressen umfaßt, andererseits jedoch die Indexregister auch 256 Werte annehmen können, kann es geschehen (wenn man nicht aufpaßt), daß die Summe aus der Basisadresse (zum Beispiel $2B) und dem Indexregisterinhalt größer als 256 wird. Wenn zum Beispiel in dem Befehl

LDA FE,X

der X-Registerinhalt 2 beträgt, ergäbe sich $FE+$02=$0100. In diesem Fall wird aber nicht der Inhalt von $0100 in den Akku geladen, sondern der Befehl spricht die Speicherstelle $00 an. Der Grund dafür liegt in der Tatsache, daß unser Prozessor den Befehl als 2-Byte-Befehl interpretiert — das 2. Byte ist die Zeropageadresse, die sich als Summe ergibt — und deswegen nur das LSB der Adresse beachtet. Von $0100 ist das LSB aber $00. Mit anderen Worten: Die Zero-page-absolut-indizierten Befehle lassen einen Zugriff nur auf die Zeropage selbst zu. Dieses Verhalten muß man beim Programmieren beachten.

chen. In der Tabelle 3 sind die KERNAL-Adressen und ihre Funktion aufgeführt. Manche davon können ohne jede Vorbereitung benutzt werden, andere brauchen bestimmte Routinen oder Angaben, um sinnvoll zu arbeiten.

Die Absicht von Commodore ist es, daß jeder Aufruf von zum Beispiel FFD2 die Ausgabe eines Zeichens bewirkt, und zwar unabhängig davon, welchen Computer in welcher Version wir benutzen. Das Programm, welches diese Zeichenausgabe letztendlich ausführt, kann sich ändern, kann in ganz andere Speicherbereiche gelegt werden. An der Stelle $FFD2 wird aber immer ein JMP mit der Einsprungadresse stehen. Leider ist diese Sprungtabelle viel zu knapp gehalten. Es gibt so viele interessante ROM-Routinen, die wir alle ohne diese schöne Sicherheit anspringen müssen.

Adresse HEX	Adresse dezimal	Name	Funktion
FF81	65409	CINT	Prüfen der TV-Norm, Berechnung der Taktfrequenz
FF84	65412	IOINIT	Ein/Ausgabe-Reset
FF87	65415	RAMTAS	Prüfen auf freien Basic-RAM
FF8A	65418	RESTOR	Initialisieren der I/O-Vektoren
FF8D	65412	VECTOR	Lesen und Setzen der I/O-Vektoren
FF90	65424	SETMSG	Setzen des Ausgabe-Modus
FF93	65427	SECOND	Ausgeben der Sekundäradresse nach LISTEN
FF96	65430	TKSA	Ausgabe der Sekundäradresse nach TALK
FF99	65433	MEMTOP	Lesen/Setzen des Speicherendes
FF9C	65436	MEMBOT	Lesen/Setzen des Speicheranfangs
FF9F	65439	SCNKEY	Abfragen der Tastatur
FFA2	65442	SETTMO	Setzen der Time-Out-Flagge
FFA5	65445	ACPTR	Zeichen vom seriellen Port in Akku lesen
FFA8	65448	CIOUT	Zeichen vom Akku auf seriellen Port ausgeben
FFAB	65451	UNTLK	Sendet UNTALK an seriellen Bus
FFAE	65454	UNLSN	Sendet UNLISTEN an seriellen Bus
FFB1	65457	LISTEN	Sendet LISTEN an Geräte per seriellen Bus
FFB4	65460	TALK	Sendet TALK an Geräte per seriellen Bus
FFB7	65463	READST	Liest I/O-Status in den Akku
FFBA	65466	SETLFS	Festlegung der Parameter für OPEN
FFBD	65469	SETNAM	Festlegung des Filenamens
FFC0	65472	OPEN	Öffnet spezifizierten File
FFC3	65475	CLOSE	Schließt spezifizierten File
FFC6	65478	CHKIN	Öffnet einen Eingabekanal
FFC9	65481	CHKOUT	Öffnet einen Ausgabekanal
FFCC	65484	CLRCHN	Schließt Ein- und Ausgabekanäle
FFCF	65487	CHRIN	Holt vom aktiven Eingabekanal ein Zeichen in den Akku
FFD2	65490	CHROUT	Sendet Akku-Inhalt auf aktiven Ausgabekanal
FFD5	65493	LOAD	LOAD und VERIFY von Programmen
FFD8	65496	SAVE	Speichern von Programmen
FFDB	65499	SETTIM	Uhrzeit setzen
FFDE	65502	RDTIM	Uhrzeit lesen
FFE1	65505	STOP	STOP-Taste abfragen
FFE4	65508	GETIN	Zeichen aus dem Tastaturpuffer in den Akku lesen
FFE7	65511	CLALL	Schließen aller Kanäle und Files
FFEA	65514	UDTIM	Uhr um 1/60 Sekunde weiterzählen
FFED	65517	SCREEN	Lesen des Bildschirmformates
FFF0	65520	PLOT	Lesen/Setzen der Cursor-Position
FFF3	65523	IOBASE	Lesen der Startadresse der Ein- und Ausgabebausteine

Tabelle 3. Kernal-Routinen

Die Urzelle eines Programmprojektes

Wir sind jetzt soweit, daß wir die Urzelle eines Programmprojektes, welches uns eine lange Zeit begleiten wird, aufbauen können. Wir wollen etwas unter den Teppich kehren. Der Teppich, das sind die uns bislang nicht zugängigen RAM-Bereiche unter den ROMs. Haben Sie das nicht auch schon mal erlebt, daß Sie während einer Programmarbeit plötzlich feststellen, Sie benötigen zum Beispiel für eine Zwischenrechnung ein weiteres Programm, oder Sie wälzen Listen und denken sich, ein kleiner Hilfsbildschirm wäre jetzt von Nutzen, oder....

Mit diesem heute zu startenden Programm wäre all das und noch viel mehr realisierbar. Es soll auf einfache Weise beliebige Speicherbereiche unters ROM schieben und sie wieder hervorholen können.

Natürlich braucht die Entwicklung dieses Projektes einige Zeit, zumal wir noch vieles lernen müssen. Deswegen sind wir in dieser ersten Urzelle noch sehr eingeschränkt: Wir verschieben zuerst einmal nur eine Bildschirm-Kopfzeile unter den oberen ROM-Bereich. Auch in dieser einfachsten Version gibt es noch einige Programmteile, die Sie erst nach der nächsten Ausgabe verstehen werden. Aber irgendwann müssen wir ja mal anfangen, Nägel mit Köpfen zu machen.

Unser Maschinenprogramm soll durch die USR-Funktion aufgerufen werden. Wie wir es in dieser Ausgabe gelernt haben, muß deshalb vor dem ersten Aufruf eine Initialisierung durch Belegen des USR-Vektors mit unserer Startadresse stattfinden. Die Startadresse soll $02B6 (dez. 694) sein, denn dort gibt es einen freien RAM-Bereich bis inklusive $02FF (dez. 767), der weder andere Programme noch Kassettenoperationen stört. Das MSB $02 ist dezimal auch 2 und wird nach 786 gePOKEt:

POKE786,2

Das LSB $B6 ist dezimal 182 und soll in 785 geschrieben werden:

POKE785,182

Damit ist der USR-Vektor gestellt und wir brauchen uns nicht mehr weiter darum zu kümmern: Jeder USR-Aufruf wird nun den Start des Programmes bewirken. Nun zum Programm selbst. In Bild 1 finden Sie ein Flußdiagramm dazu.

Zunächst konstruieren wir den Teil, der die erste Bildschirmzeile nach $E000 und folgende Speicherstellen schiebt. Das X-Register verwenden wir als Index und laden es mit dez.40 = $27.

Schalten Sie also den SMON ein und starten Sie den Assembler mit:

A 02B6

Dann geben Sie ein:

```
02B6    LDX #27
```

Nun packen wir das letzte Zeichen der obersten Bildschirmzeile in den Akku:

```
02B8    LDA 0400,X
```

In das Y-Register legen wir die dazugehörige Farbe aus dem Bildschirmfarbspeicher:

```
02BB    LDY D800,X
```

Den Akkuinhalt — also die Bildschirminformation — legen wir nach $E000+$27:

```
02BE    STA E000,X
```

Dasselbe tun wir mit dem Farbcode, der ab $E028+$27 abwärts gespeichert wird. Leider kann man STY nicht X-indiziert-absolut adressieren (siehe Tabelle 1). Deshalb schieben wir zuerst den Y-Registerinhalt in den Akku:

```
02C1    TYA
02C2    STA E028,X
```

Damit ist das letzte Zeichen der Kopfzeile verschoben. Wir zählen das X-Register um 1 herunter:

```
02C5    DEX
```

Der X-Index weist nun auf das vorletzte Zeichen, mit dem sich alles ab $02A9 wiederholt. Wenn das X-Register bis 0 heruntergezählt ist, weist es auf das erste Zeichen der Kopfzeile. Die Schleife muß dann noch einmal durchlaufen werden und ein weiteres Herabzählen des X-Registers erzeugt $FF, was zum Setzen der N-Flagge führt. Das ist dann unser Signal, daß die gesamte Kopfzeile übertragen wurde. Die N-Flagge wird durch den BPL-Befehl getestet:

```
02C6    BPL 02B8
```

So weit, so gut. Wir hätten natürlich auch das X-Register von 0 an hochzählen können. Zum Beenden der Schleife wäre dann aber ein CPX-Befehl erforderlich gewesen, der jedesmal den X-Registerinhalt mit der Zahl $27 vergleicht.

MERKE: Indexregister in Schleifen abwärts zu zählen, kann Rechenzeit einsparen!

Ab $02CE soll der umgekehrte Vorgang, also das Zurückschieben der vorher gespeicherten Kopfzeile in den Bildschirmspeicher geschehen. Das einfachste wäre es sicherlich, diesen Programmteil mit einem weiteren USR-Kommando zu starten. Das sähe dann so aus:

1.USR-Befehl -- schiebt Kopfzeile unter oberes ROM
2.USR-Befehl -- holt Kopfzeile zurück in Bildschirmspeicher
3.USR-Befehl -- schiebt wieder Kopfzeile unter ROM
4.USR-Befehl -- holt sie wieder zurück und so weiter.

Weil aber das Umstellen des USR-Vektors durch POKEs vom Basic aus lästig ist, tun wir das einfach immer am Ende des betreffenden Maschinenprogrammabschnittes. Wir schreiben also das LSB der Programmfortführung ($CE) nach $311. Das MSB bleibt unverändert $02.

```
02C8    LDA #CE
02CA    STA 0311
02CD    RTS
```

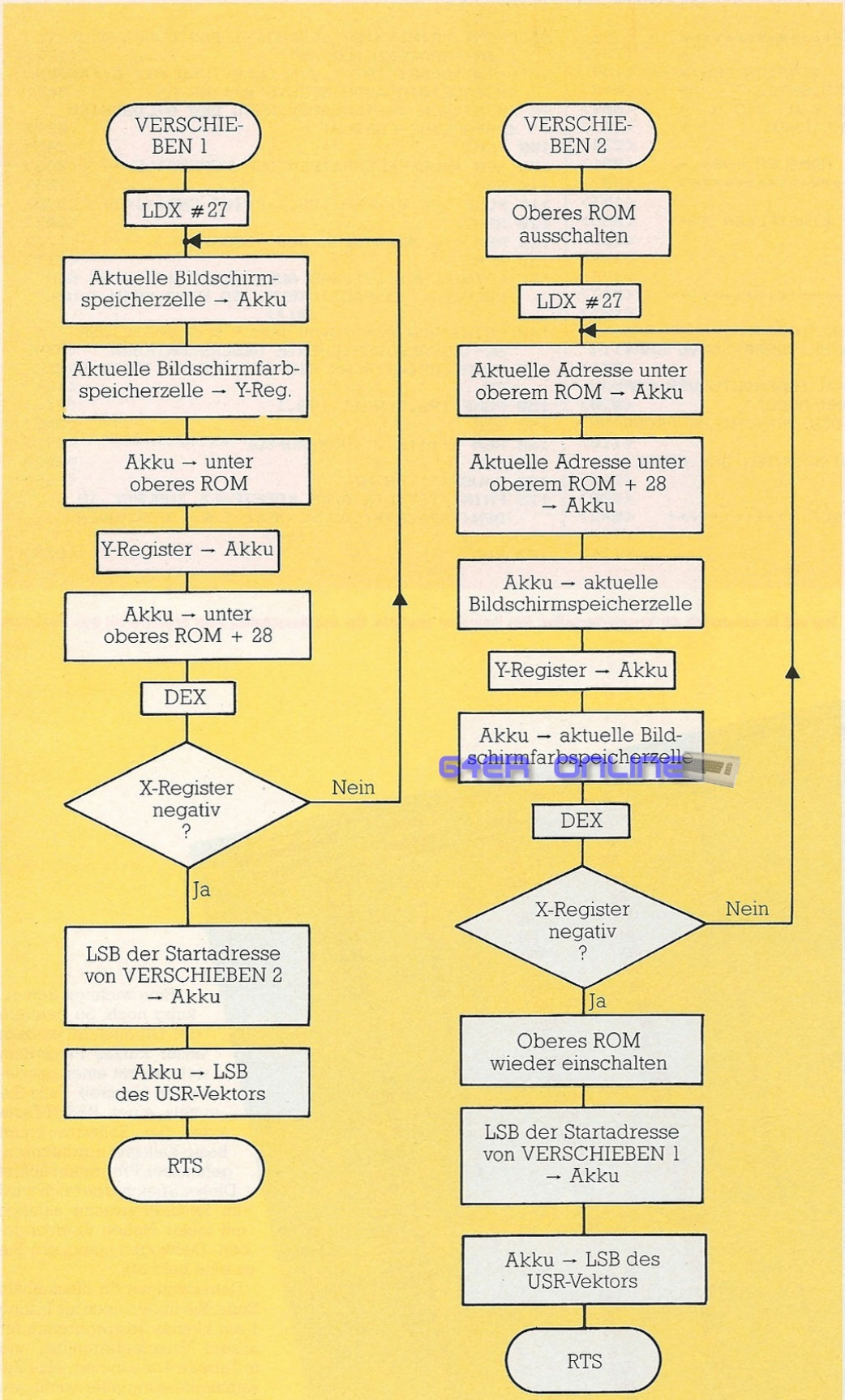

Bild 1. Das Flußdiagramm zu dem im Text erklärten Programm.

Mit dem RTS sind wir wieder im Basic-Programm gelandet, welches nun normal weiterverarbeitet wird. Erst ein neues USR-Kommando — im Programm oder im Direktmodus — startet den zweiten Teil unseres Maschinenprogrammes (weil in $0311, — der Einsprungpunkt des USR-Befehls — die Startadresse der auszuführenden Routine steht).

Einfache Befehle mit großer Wirkung

In diesem 2. Teil müssen wir erst einige Befehle geben, die Sie jetzt vielleicht noch nicht verstehen. Das hängt damit zusammen, daß zum Herauslesen des RAM unter dem ROM das ROM ausgeschaltet werden muß (entspricht POKE 1,53):

```
02CE    LDA 01
02D0    PHA
02D1    LDA #35
02D3    STA 01
```

(Der PHA-Befehl dient hier zur Zwischenspeicherung des Akku-Inhaltes). Das ist hiermit geschehen und wir kommen wieder in bekannte Gefilde mit der Ausleseschleife:

```
02D5    LDX #27
02D7    LDY E000,X
02DA    LDY E028,X
02DD    STA 0400,X
02E0    TYA
02E1    STA D800,X
02E4    DEX
02E5    BPL 02D7
```

Damit ist die gesamte gespeicherte Kopfzeile wieder zurückgeholt und wir können das ROM wieder einschalten:

```
02E7    PLA
02E8    STA 01
```

Falls nun wieder ein USR-Kommando auftaucht, soll die Kopfzeile mit dem 1. Programmteil unter das obere ROM gelegt werden wie am Anfang. Wir müssen deshalb den USR-Vektor auf $02B6 zurückschreiben:

```
02EA    LDA #B6
02EC    STA 0311
02EF    RTS
```

Das wärs! Wenn nun im Programm oder im Direktmodus wieder ein USR-Befehl auftritt, kann das Ganze von vorne beginnen. In dieser Version wird jedesmal eine neue Kopfzeile hin- und wieder zurückgeschoben. Wenn Sie eine einmal festgelegte Kopfzeile immer wieder benutzen möchten, dann stellen Sie den USR-Vektor einfach nicht mehr zurück: Lassen Sie also die Befehle bei 02EA und 02EC weg. Das Programm endet in dem Fall mit:

```
02EA    RTS
```

```
1 REM ********************************** <250>
2 REM *                                * <229>
3 REM *  TEST FUER DIE 1. VERSION DES  * <139>
4 REM *       PROGRAMM-PROJEKTES       * <048>
5 REM * V E R S C H I E B E N  V O N   * <009>
6 REM *       SPEICHERBEREICHEN        * <193>
7 REM *                                * <234>
8 REM * HEIMO PONNATH    HAMBURG 1984 * <081>
9 REM ********************************** <002>
10 REM                                   <153>
15 REM ++++++ USR-VEKTOR EINSTELLEN ++++ <065>
20 REM                                   <163>
25 POKE 785,182:POKE 786,2               <239>
30 REM                                   <173>
35 REM ++++++ KOPFZEILE ++++++++++++++++ <013>
40 REM                                   <183>
45 PRINT CHR$(147)CHR$(18)"TEST
   : BILD $0400=1024, FARBE $D800=55296"CHR$(14
   6)                                    <071>
50 PRINT:PRINT:PRINT"DURCH IRGENDEIN USR-KOMMAN
   DO WIRD NUN IM PROGRAMM-MODUS"        <020>
55 PRINT"DER ERSTE TEIL DES VERSCHIEBE-PROGRAMM
   ES AUFGERUFEN"                        <110>
60 PRINT"DIE KOPFZEILE WIRD UNTER DAS OBERE
    ROM{2SPACE}KOPIERT."                 <215>
65 REM                                   <208>
70 REM ++++++ 1. USR-AUFRUF ++++++++++++ <042>
75 REM                                   <218>
80 A=USR(1)                              <124>
85 PRINT:PRINT"HIER GESCHIEHT DAS DURCH A=USR(1
   ) IN{4SPACE}ZEILE 65"                 <132>
90 PRINT"DABEI IST 1 EIN DUMMY UND MIT A FANGEN
   {2SPACE}WIR AUCH NICHTS WEITER AN."   <063>
95 PRINT"AUF TASTENDRUCK WIRD DER BILDSCHIRM
   {2SPACE}GE-LOESCHT"                   <029>
100 REM                                  <243>
105 REM ++UEBERSCHREIBEN DER KOPFZEILE ++ <046>
110 REM                                  <253>
115 POKE 198,0:WAIT 198,1:PRINT CHR$(147) <090>
120 REM                                  <007>
125 REM +++ NEUBEGINN DES PROGRAMMES ++++ <173>
130 REM                                  <017>
135 PRINT CHR$(19)"WAS AUCH IMMER JETZT IN DER
    KOPFZEILE{3SPACE}STEHT, ES WIRD BEIM 2.USR"
                                         <017>
140 PRINT"VON DEM ZUVOR DURCH DAS ERSTE USR
    GE-{3SPACE}SPEICHERTE UEBERSCHRIEBEN" <078>
145 PRINT:PRINT"WENN SIE JETZT EINE TASTE DRUEC
   KEN..."                               <104>
150 POKE 198,0:WAIT 198,1                <246>
155 REM                                  <042>
160 REM ++++++ 2. USR-AUFRUF ++++++++++++ <090>
165 REM                                  <052>
170 A=USR(1):PRINT                       <169>
175 PRINT"IST DIE ALTE KOPFZEILE ZURUECK IN
    DEN{3SPACE}BILDSCHIRMSPEICHER GESCHOBEN."
                                         <164>
180 END                                  <052>
```

Listing 1. Test und Demonstration der Verschieberoutine. Das Programm zeigt das Ein- und Ausschalten einer Kopfzeile auf dem Bildschirm

Befehlswort	Adressierung	Bytezahl	Code Hex	Code Dez	Taktzyklen	Beeinflussung von Flaggen
LDA	absolut,X	3	BD	189	4	N,Z
	0-page-abs,X	2	B5	181	4	N,Z
LDX	absolut,Y	3	B9	185	4	N,Z
	absolut,Y	3	BE	190	4	N,Z
LDY	0-page-abs,Y	2	B6	182	4	N,Z
	absolut,X	3	BC	188	4	N,Z
STA	0-page-abs,X	2	B4	180	4	N,Z
	absolut,X	3	9D	157	5	N,Z
	absolut,Y	3	99	153	5	/
STX	0-page-abs,X	2	95	149	4	/
STY	0-page-abs,Y	2	96	150	4	/
INC	0-page-abs,X	2	94	148	4	/
	absolut,X	3	FE	254	7	/
DEC	0-page-abs,X	2	F6	246	6	N,Z
	absolut,X	3	DE	222	7	N,Z
ADC	0-page-abs,X	2	D6	214	6	N,Z
	absolut,X	3	7D	125	4	N,Z
	absolut,Y	3	79	121	4	N,V,Z,C
SBC	0-page-abs,X	2	75	117	4	N,V,Z,C
	absolut,X	3	FD	253	4	N,V,Z,C
	absolut,Y	3	F9	249	4	N,V,Z,C
CMP	0-page-abs,X	2	F5	245	4	N,V,Z,C
	absolut,X	3	DD	221	4	N,V,Z,C
	absolut,Y	3	D9	217	4	N,Z,C
BIT	0-page-abs,X	2	D5	213	4	N,Z,C
	absolut	3	2C	44	4	N,Z,C
CLV	0-page-abs.	2	24	36	4	N,V,Z
NOP	implizit	1	B8	184	3	N,V,Z
TAX	implizit	1	EA	234	2	V
TAY	implizit	1	AA	170	2	/
TXA	implizit	1	A8	168	2	N,Z
TYA	implizit	1	8A	138	2	N,Z
JMP	absolut	3	98	152	2	N,Z
	indirekt	3	4C	76	3	N,Z
JSR	absolut	3	6C	108	5	/
			20	32	6	/

Tabelle 4. Zusammenfassung aller wichtigen Daten der neuen Befehle

Eine wichtige Bemerkung noch: So bequem der Ort auch ist, an dem unser kurzes Programm steht, er hat einen gravierenden Nachteil: Falls Sie mittels einer RESET-Taste oder per Software einen Basic-Kaltstart durchführen, geht unser Programm flöten! Dieser Speicherbereich wird im Reset-Programm nämlich mit lauter Nullen überschrieben. Deswegen speichern Sie es bitte bald ab.

Damit sind wir für diesmal am Ende. Sie finden noch als Listing 1 ein kleines Testprogramm für unsere Verschieberoutine, und in Tabelle 4 wie immer, eine Zusammenfassung aller wichtigen Daten der neuen Befehle. In der nächsten Folge greifen wir noch einmal das Thema Fließkomma auf, werden die einfachsten und kürzesten Kurzspeicher-Befehle kennenlernen und beginnen mit den leistungsfähigsten Befehlen des 6502, den indirekt-indizierten-Befehlen.

(H. Ponnath/gk)