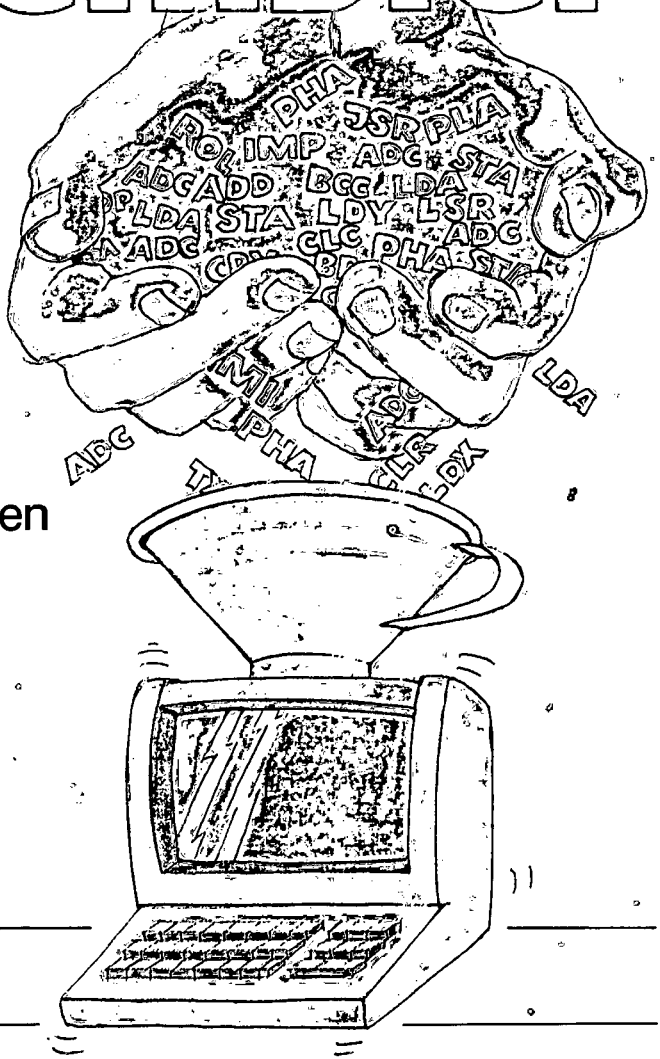
Appie Assembler

Tips und Tricks



Mit ausführlichen Programmbeispielen

Ulrich Stiehl

2. Auflage

Hüthig

ASCII-Tabelle

§	00	00000000	000	§	40	01000000	064	§	80	10000000	128	§	C0	11000000	192
A	01	00000001	001	A	41	01000001	065	A	81	10000001	129	A	C1	11000001	193
B	02	00000010	002	B	42	01000010	066	B	82	10000010	130	B	C2	11000010	194
C	03	00000011	003	C	43	01000011	067	C	83	10000011	131	C	C3	11000011	195
D	04	00000100	004	D	44	01000100	068	D	84	10000100	132	D	C4	11000100	196
E	05	00000101	005	E	45	01000101	069	E	85	10000101	133	E	C5	11000101	197
F	06	00000110	006	F	46	01000110	070	F	86	10000110	134	F	C6	11000110	198
G	07	00000111	007	G	47	01000111	071	G	87	10000111	135	G	C7	11000111	199
H	08	00001000	008	H	48	01001000	072	H	88	10001000	136	H	C8	11001000	200
I	09	00001001	009	I	49	01001001	073	I	89	10001001	137	I	C9	11001001	201
J	0A	00001010	010	J	4A	01001010	074	J	8A	10001010	138	J	CA	11001010	202
K	0B	00001011	011	K	4B	01001011	075	K	8B	10001011	139	K	CB	11001011	203
L	0C	00001100	012	L	4C	01001100	076	L	8C	10001100	140	L	CC	11001100	204
M	0D	00001101	013	M	4D	01001101	077	M	8D	10001101	141	M	CD	11001101	205
N	0E	00001110	014	N	4E	01001110	078	N	8E	10001110	142	N	CE	11001110	206
O	0F	00001111	015	O	4F	01001111	079	O	8F	10001111	143	O	CF	11001111	207
P	10	00010000	016	P	50	01010000	080	P	90	10010000	144	P	D0	11010000	208
Q	11	00010001	017	Q	51	01010001	081	Q	91	10010001	145	Q	D1	11010001	209
R	12	00010010	018	R	52	01010010	082	R	92	10010010	146	R	D2	11010010	210
S	13	00010011	019	S	53	01010011	083	S	93	10010011	147	S	D3	11010011	211
T	14	00010100	020	T	54	01010100	084	T	94	10010100	148	T	D4	11010100	212
U	15	00010101	021	U	55	01010101	085	U	95	10010101	149	U	D5	11010101	213
V	16	00010110	022	V	56	01010110	086	V	96	10010110	150	V	D6	11010110	214
W	17	00010111	023	W	57	01010111	087	W	97	10010111	151	W	D7	11010111	215
X	18	00011000	024	X	58	01011000	088	X	98	10011000	152	X	D8	11011000	216
Y	19	00011001	025	Y	59	01011001	089	Y	99	10011001	153	Y	D9	11011001	217
Z	1A	00011010	026	Z	5A	01011010	090	Z	9A	10011010	154	Z	DA	11011010	218
Ä	1B	00011011	027	Ä	5B	01011011	091	Ä	9B	10011011	155	Ä	DB	11011011	219
Ö	1C	00011100	028	Ö	5C	01011100	092	Ö	9C	10011100	156	Ö	DC	11011100	220
Ü	1D	00011101	029	Ü	5D	01011101	093	Ü	9D	10011101	157	Ü	DD	11011101	221
↑	1E	00011110	030	↑	5E	01011110	094	↑	9E	10011110	158	↑	DE	11011110	222
_	1F	00011111	031	_	5F	01011111	095	_	9F	10011111	159	_	DF	11011111	223
	20	00100000	032	'	60	01100000	096		A0	10100000	160	'	E0	11100000	224
!	21	00100001	033	a	61	01100001	097	!	A1	10100001	161	a	E1	11100001	225
"	22	00100010	034	b	62	01100010	098	"	A2	10100010	162	b	E2	11100010	226
#	23	00100011	035	c	63	01100011	099	#	A3	10100011	163	c	E3	11100011	227
$	24	00100100	036	d	64	01100100	100	$	A4	10100100	164	d	E4	11100100	228
%	25	00100101	037	e	65	01100101	101	%	A5	10100101	165	e	E5	11100101	229
&	26	00100110	038	f	66	01100110	102	&	A6	10100110	166	f	E6	11100110	230
.	27	00100111	039	g	67	01100111	103	.	A7	10100111	167	g	E7	11100111	231
(	28	00101000	040	h	68	01101000	104	(	A8	10101000	168	h	E8	11101000	232
)	29	00101001	041	i	69	01101001	105	)	A9	10101001	169	i	E9	11101001	233
*	2A	00101010	042	j	6A	01101010	106	*	AA	10101010	170	j	EA	11101010	234
+	2B	00101011	043	k	6B	01101011	107	+	AB	10101011	171	k	EB	11101011	235
,	2C	00101100	044	l	6C	01101100	108	,	AC	10101100	172	l	EC	11101100	236
-	2D	00101101	045	m	6D	01101101	109	-	AD	10101101	173	m	ED	11101101	237
.	2E	00101110	046	n	6E	01101110	110	.	AE	10101110	174	n	EE	11101110	238
/	2F	00101111	047	o	6F	01101111	111	/	AF	10101111	175	o	EF	11101111	239
0	30	00110000	048	p	70	01110000	112	0	B0	10110000	176	p	F0	11110000	240
1	31	00110001	049	q	71	01110001	113	1	B1	10110001	177	q	F1	11110001	241
2	32	00110010	050	r	72	01110010	114	2	B2	10110010	178	r	F2	11110010	242
3	33	00110011	051	s	73	01110011	115	3	B3	10110011	179	s	F3	11110011	243
4	34	00110100	052	t	74	01110100	116	4	B4	10110100	180	t	F4	11110100	244
5	35	00110101	053	u	75	01110101	117	5	B5	10110101	181	u	F5	11110101	245
6	36	00110110	054	v	76	01110110	118	6	B6	10110110	182	v	F6	11110110	246
7	37	00110111	055	w	77	01110111	119	7	B7	10110111	183	w	F7	11110111	247
8	38	00111000	056	x	78	01111000	120	8	B8	10111000	184	x	F8	11111000	248
9	39	00111001	057	y	79	01111001	121	9	B9	10111001	185	y	F9	11111001	249
:	3A	00111010	058	z	7A	01111010	122	:	BA	10111010	186	z	FA	11111010	250
;	3B	00111011	059	ä	7B	01111011	123	;	BB	10111011	187	ä	FB	11111011	251
<	3C	00111100	060	ö	7C	01111100	124	<	BC	10111100	188	ö	FC	11111100	252
=	3D	00111101	061	ü	7D	01111101	125	=	BD	10111101	189	ü	FD	11111101	253
>	3E	00111110	062	ß	7E	01111110	126	>	BE	10111110	190	ß	FE	11111110	254
?	3F	00111111	063	←	7F	01111111	127	?	BF	10111111	191	←	FF	11111111	255

Ulrich Stiehl · Apple Assembler

Ulrich Stiehl

Apple Assembler

Tips und Tricks

Mit ausführlichen Programmbeispielen

2. Auflage

Dr. Alfred Hüthig Verlag Heidelberg

Als Ergänzung zu diesem Buch ist gesondert lieferbar:
„Begleitdiskette zu Apple Assembler"
ISBN 3-7785-1048-7

CIP-Kurztitelaufnahme der Deutschen Bibliothek
Stiehl, Ulrich:
Apple Assembler: Tips u. Tricks; mit ausführl. Programmbeispielen/Ulrich Stiehl.
– 2. Aufl. –
– Heidelberg: Hüthig, 1985
ISBN 3-7785-1047-9

Analytisches Inhaltsverzeichnis

(Mit Kurzbeschreibung aller Programme und Angabe der jeweils benutzten ROM-Routinen)

0. Zum Aufbau dieses Buches

Dieses Buch wendet sich nicht an Apple-Neulinge, die gerade ihr neues Gerät aus dem Verpackungskarton herausgenommen haben, sondern an alle, die sich bereits in Applesoft auskennen und zudem mit Hilfe eines 6502-Anfängerbuches ihre ersten Programme in Assembler erstellt haben, aber nunmehr ein Nachschlagewerk suchen, das eine Vielzahl nützlicher ROM-Routinen und sonstiger Assembler-Utilities in einer systematischen Form zusammenstellt.

– Applesoftprogrammierer mit sehr geringen oder gar keinen Assemblerkenntnissen können die in diesem Buch enthaltenen etwa 40 Routinen, die zur Wahrung des Unterroutine-Charakters im allgemeinen relativ kurz sind, in Form des Objekt-Codes im Monitor eingeben.
– Assemblerprogrammierer können demgegenüber den Source-Code in ihren Assembler eingeben, womit die Möglichkeit der Änderung sowie der Assemblierung für andere Speicherbereiche besteht. (Die Assemblerlistings wurden mit dem Big Mac bzw. Merlin erstellt, doch ist eine Konvertierung für Lisa, SC-Assembler usw. leicht möglich, da keine seltenen Opcodes und gar keine Macros verwendet wurden.)

Es sei an dieser Stelle bereits darauf hingewiesen, daß sich die zahlreichen ROM-Adressen sowie sinngemäß die Assemblerroutinen sowohl auf den Apple II Plus wie auch auf den Apple IIe beziehen. Darüber hinaus enthält dieses Buch zahlreiche Adressen und Routinen, die ausschließlich für den Apple IIe gedacht sind, z.B. doppelte Lores- und Hires-Grafik. Einerseits wurde auf eine Darstellung spezifischer Apple-II-ROM-Routinen verzichtet, weil dieser alte Apple schon seit mehreren Jahren nicht mehr im Handel ist. Andererseits wurde aber auch von einer Schilderung von ROM-Routinen für den ganz neuen Apple IIc abgesehen, weil bei diesem nicht nur der Applesoft-Interpreter ($D000-$F7FF), sondern der ganze F8-Monitor ($F800-$FFFF) umgeschrieben

wurde. Insgesamt wurden beim Apple IIc gegenüber dem Apple IIe im Bereich $D000-$FFFF über 1100 Speicherstellen geändert, von massiven Änderungen des $C100-$CFFF-Bereichs ganz zu schweigen. Damit ist der Apple IIc in vieler Hinsicht nicht mehr mit dem Apple II Plus und IIe kompatibel. Diese Politik der Firma Apple ist weitgehend unverständlich, erschien doch der Apple IIc bereits ein gutes Jahr nach dem Apple IIe. Wer sich also wundert, warum der Applewriter IIe und andere Programme nicht mehr auf dem Apple IIc laufen, sollte sich fragen, ob er nicht das falsche Modell erworben hat. Sollte dieses Gerät eine größere Verbreitung erfahren, werde ich einen gesonderten Assemblerband herausbringen, zumal der Apple IIc insbesondere im Bereich $C100-$CFFF die neuen 65C02-Opcodes verwendet.

Apple-spezifische Einführungen in 6502-Assemblerprogrammierung gibt es wie Sand am Meer, z.B. „Wagner, Assembly Lines: The Book" (sehr gut), „Hyde, Using 6502 Assembly Language" (sehr gut, wenn auch nicht als Erstbuch geeignet) oder „Inman/Inman, Apple Maschinensprache" (recht gut, doch wird nicht auf Assembler selbst eingegangen). Neben diesen Anfängerbüchern gibt es nur noch geräteneutrale Prozessorbücher wie „Leventhal, Assembly Language Programming" (sehr gut) usw. Will man sich jedoch einen Überblick über wichtige ROM-Adressen verschaffen oder sucht man eine Sammlung apple-spezifischer Assembler-Unterprogramme, dann stellt man fest, daß geeignete Literatur fehlt. Im allgemeinen muß man jetzt alte Hefte von Zeitschriften wie „Nibble" usw. durchblättern, bis man – meist per Zufall – etwas Geeignetes findet.

Dieses Buch ist keine „wirre" Anhäufung von unausgetesteten Peeks, Pokes und Calls, denn es galt der Grundsatz: Die Angabe einer Adresse ist nur dann von Nutzen, wenn sie erstens stimmt (eigentlich selbstverständlich) und wenn zweitens klar definiert wird, welche Werte vor dem Sprung in die Routine initialisiert werden müssen. Bedauerlicherweise haben viele Autoren von ihren Vorgängern „abgekupfert", ohne das Abgeschriebene überprüft zu haben. So schreibt z.B. Firma Apple in den als Loseblattwerk erhältlichen „APPLE TECH NOTES" auf Seite „2600.033.07" vom 28.6.82, daß die Routine GIVAYF A und Y in FAC unwandelt, wobei auf Seite „2600.033.01" A als Low Byte und Y als High Byte definiert werden. Stimmt dies? Antwort: Nein! Diese Routine ist von Firma Apple „richtig" aus einem Aufsatz von J. Crossley in „Apple Orchard" abgeschrieben worden, wo bereits A und Y falsch bzw. vertauscht waren. Noch schlimmer wird es, wenn man größere Zusammenstellungen von Adressen unter die Lupe nimmt, wie z.B. die in der Zeitschrift „mc" 2/1983 erschienene Sammlung von „ROM-Routinen des Applesoft-Basic-Inter-

preters". Hier wird man bei genauerem Hinsehen permanent Fehler finden, so daß der Nutzen dieser Sammlung mehr als fragwürdig ist. Ein Beispiel: Zur Adresse $E2AD heißt es: „Unterprogramm für Feldbehandlung: multipliziert Inhalt von $64-$65 mit Inhalt von $AD-$AE. Ergebnis in Y und X". Mein Kommentar hierzu: Erstens kann man nicht in die Stelle $E2AD hineinspringen, sondern muß den Entry $E2B8 nehmen (denn bei $E2AD wird der Speicherinhalt, auf den LOWTR zeigt, in $64-$65 kopiert). Zweitens findet man nach JSR $E2B8 nicht das Produkt in Y und X, sondern in X (Low Byte) und A (High Byte).

Angesichts dieser mißlichen Ausgangssituation enthält dieses Buch zu den „meisten – aus Platzgründen leider nicht zu allen – aufgeführten Routinen entsprechende Anwendungsprogramme mit Utility-Charakter, damit es dem Benutzer nicht so ergeht wie mir, als ich die oben erwähnte Multiplikationsroutine laut „mc" austesten wollte. Darüber hinaus findet man zahlreiche weitere selbstentwickelte Unterroutinen, die nicht auf ROM-Routinen basieren. Insgesamt enthält dieses Buch über 40 Assemblerprogramme, die – von wenigen Ausnahmen abgesehen – zur Einbindung in eigene Programme gedacht sind. Das Material basiert auf einem Fundus von etwa 100 Programmen, von denen die 40 wichtigsten für diese Sammlung ausgewählt wurden. Und von diesen 40 Programmen wurde wiederum etwa die Hälfte speziell für dieses Buch umgeschrieben, um den allgemeinen Nutzwert zu erhöhen. Zahlreiche Programme sind neuartig, d.h. sie wurden in dieser oder ähnlicher Form bislang meines Wissens noch nie publiziert, z.B. die Programme BILDSCHIRMMASKE, DOUBLEPLOT, DOUBLE.LORES u.a.

Das vorliegende Buch wurde in folgende Teile gegliedert:

Teil 1: 6502-Repetitorium: Dieser Teil, der nicht für Assembler-Neulinge bestimmt ist, enthält ein Repetitorium der wichtigsten Befehle, Adressen und sonstigen Besonderheiten des 6502 sowie Angaben zu den apple-spezifischen Zahlenformaten (Integerzahlen, Fließkommazahlen).

Teil 2: Monitor-ROM: Hier werden neben einer Kurzwiederholung der Monitor-Befehle praktisch alle Routinen und Adressen zusammengestellt, die für Assemblerprogrammierer von Nutzen sein könnten. Wichtige ROM-Routinen werden als Utility-Programme vorgestellt. Darüber hinaus findet der Leser Unterroutinen für hexadezimale Addition, Subtraktion, Multiplikation und Division mit mehrfacher Genauigkeit.

Teil 3: Speicherverwaltung: In diesem Teil werden verschiedene Lösch-, Move- und Testroutinen für die Language Card sowie die 64K-Karte vorgestellt, wobei besonders ausführlich auf die Softswitches eingegangen wird.

Teil 4: Applesoft-ROM: Dieser Hauptabschnitt beschreibt die interne Struktur von Applesoft-Programmen, Methoden der Parameterübergabe mittels CALL, USR, &, PEEK und POKE sowie eine größere Anzahl von Interpreter-Adressen. Bei den Utility-Programmen liegt das Schwergewicht auf Fließkommamathematik einschließlich Print Using usw.

Teil 5: Text- und Grafikspeicher: Dieser Teil befaßt sich mit der Erstellung von Bildschirmmasken für den Apple II Plus und mit ROM-unabhängigen 80-Z/Z-Routinen sowie doppelter Lores- und Hires-Grafik für den Apple IIe, wobei auch hier wie bei der Speicherverwaltung besonderes Augenmerk auf die Handhabung von Softswitches gelegt wird.

Wie ersichtlich, enthält dieses Buch keine DOS-Assemblerroutinen, da diese bereits in meinen anderen Apple-Büchern („Apple DOS 3.3" und „Apple ProDOS") ausführlich dargestellt wurden. Ferner konnte eine Reihe von Assembler-Utilities, insbesondere zur Hires-Grafik und zur String-Verarbeitung, aus Platzgründen nicht aufgenommen werden, so daß auch auf die Nennung entsprechender Interpreter-Adressen verzichtet wurde. Bei Bedarf werden diese in Form eines gesonderten Bandes veröffentlicht.

Abschließend noch ein Wort zum Titel „Tips und Tricks", mit dem leider viel Schindluder getrieben wird. So wird z.B. das Data-Becker-Buch „Apple II Tips & Tricks" in der Werbung angepriesen mit „...nützliche Peeks und Pokes, Unterprogramme in Maschinensprache..." usw. Das besagte Buch enthält exakt 9 Call-Adressen (z.B. Call -151), ca. 10 Peeks/Pokes (z.B. Poke -16368, 0) und ein einziges „komplettes" Assemblerprogramm (9 Bytes lang und zudem noch aus einem fremden Anfängerbuch von Mottola zitiert), was laut Werbung als „ungeheure Fülle an Informationen" angepriesen wird. Das genannte Buch hat zweifelsohne seine Existenzberechtigung, denn es ist eine Applesoft-Einführung für „blutige Laien". Von „Tips und Tricks" kann man meines Erachtens jedoch nur dann sprechen, wenn ein Buch für Fortgeschrittene gedacht ist und wenn es neben „Call -151" auch noch andere Dinge bringt, die weniger oder gar nicht bekannt sind.

1. 6502-Repetitorium

Nachfolgend werden in Form eines Repetitoriums terminologische Grundfragen geklärt, die 6502-Befehle und deren Adressierungsarten wiederholend zusammengefaßt sowie vermischte Tips und Tricks für die Assembler-Programmierung gegeben. (Dies ist mithin keine Einführung für Assembler-Anfänger. Wer eine solche sucht, sei auf meine Aufsatz-Serie „6502 leicht gemacht" verwiesen, die seit Heft 1/1983 in der Zeitschrift „CAL – Computer-Anwendung im Labor" läuft.)

1.1. Binär- und Hexzahlen

Ein Bit kann 1 (on = ein; set = gesetzt) oder 0 (off = aus; reset, clear = zurückgesetzt) sein. Die 8 Bits eines Bytes werden von rechts nach links und von 0 bis 7 (rückwärts) numeriert:

```
  76543210     Bit-Position
%11110000      $F0 (Beispiel)
```

Man sagt z.B.: Bit 7 ist off, Bit 0 ist on usw.

Eine 2stellige Hexzahl (= Byte) besteht aus 2 Hex-Ziffern oder 2 Hex-Nibbles (= Halbbytes): High Nibble H und Low Nibble L, z. B.:

$F0: $F = High Nibble H, $0 = Low Nibble L

Eine 4stellige Hexzahl besteht aus 2 Hexbytes (Low Byte LL und High Byte HH) und wird als Adresse (bzw. 16-Bit-Zero-Page-Pointer) in der Form

$HHLL geschrieben, jedoch intern in der Form $LLHH (Low Byte first) abgelegt, z.B.:

```
0300:    20  00  10    JSR    $1000
         OP LL HH
```

Eine 6stellige Hexzahl kann bei Multiplikationen usw. vorkommen, nicht dagegen bei Adressierungsarten, da der 6502 nur 64K adressieren kann. Wir schreiben dann $HHMMLL und speichern ggf. intern $LLMMHH. MM steht für das Middle Byte.

Dezimalzahlen erhalten kein Präfix, Binärzahlen erhalten das Präfix % und Hexzahlen erhalten das Präfix $, z.B.:

```
 255          = Dezimalzahl 255
%11111111     = Binärzahl (Dualzahl) für dezimal 255
$FF           = Hexzahl (hexadezimale Zahl) für dezimal 255
```

(Daneben gibt es BCD-Zahlen, z.B. $11 = dezimal 11, die aber beim Apple kaum benutzt werden.)

Wenn bei einem Assembler eines der Register A, X, Y mit einer Zahl im Dezimalbereich 0-255 geladen werden soll, wird das Nummernkreuz vorangestellt (= umittelbare Adressierung), z.B.:

```
LDA  #255
LDA  #%11111111
LDA  #$FF
```

Man beachte, daß Bit-Muster bei LDA usw. stets 8stellig geschrieben werden müssen. Wird eine Stelle vergessen, z.B. LDA #%0101010 (nur 7stellig!), dann berechnet der Assembler in der Regel einen falschen Wert.

Wenn der Hexwert eines ASCII-Buchstabens geladen werden soll, verwenden wir folgende Gänsefüßchen-Konvention:

```
LDA  #„A„      ;entspricht $C1 = %11000001 Bit 7 gesetzt
LDA  #‚A‘      ;entspricht $41 = %01000001 Bit 7 zurückgesetzt
```

Wenn eine als Label definierte Adresse geladen werden soll, verwenden wir folgende Kleiner-Größer-Konvention:

```
PTRL    EQU   $00CE      ;Pointer Low
PTRH    EQU   $00CF      ;Pointer High
ADRS    EQU   $1000
        LDA   #<ADRS     ;entspricht LL, d.h. $00
        STA   PTRL       ;Low Byte
        LDA   #>ADRS     ;entspricht HH, d.h. $10
        STA   PTRH       ;High Byte
        RTS
```

Bei 2stelligen Hexzahlen kann man das Bit 7 als Vorzeichen-Bit ansehen. Es gilt dann der Zahlenbereich:

$00-$7F = %00000000 – %01111111 = +0 bis +127 dezimal; BPL
$80-$FF = %10000000 – %11111111 = -1 bis -128 dezimal; BMI

2stellige Hexzahlen mit Vorzeichen kommen 6502-intern bei relativen Vorwärts- und Rückwärts-Verzweigungen vor. Ferner kann man mit BPL und BMI prüfen, in welchem der beiden Bereiche sich ein Registerwert befindet, unabhängig davon, ob man eine Vorzeichenzahl im Sinn hat oder nicht, z.B. Tastaturabfrage:

```
KEY     LDA   $C000      ;Keyboard
        BPL   KEY        ;noch keine Taste gedrückt!
        BIT   $C010      ;Keyboard-Strobe
        RTS              ;gedrückte Taste im Bereich $80-$FF jetzt im
                         A-Register
```

Ähnliches gilt für 4stellige Hexzahlen mit Vorzeichen (= Integerzahlen), wobei Bit 15 das Vorzeichen-Bit ist:

$0000-$7FFF = +0 bis +32767
$8000-$FFFF = -1 bis -32768

Für Applesoft-Integerzahlen gelten als Besonderheiten: Weder -32768 noch +32768 ist erlaubt! Im Speicher werden sie in der Form HH LL abgelegt!

Die Applesoft-Fließkommazahlen sind im Prinzip „linksbündig ausgeschlossene 40stellige" Binärzahlen (Potenzen von 2) mit einer 32-Bit-Mantisse und einem 8-Bit-Exponenten, wobei man jedoch wegen der „verschrobenen" teils gepackten 5-Byte- (= Memory-Format) und teils nicht-gepackten 6-Byte-Verschlüsselung (FAC-Format) sowie der Position der Binärkommastelle und der Linksverschiebung der Mantisse „Durchblicker-Störungen" hat. Daher nachfolgend eine starke Vereinfachung als Beispiel.

Mit dem FAC-Programm (Kap. 4.2.10) erzeugen wir mit

& 536870911 das folgende FAC-Bit-Muster

```
10011101  11111111  11111111  11111111  11111000
Exponent  Mantisse  Mantisse  Mantisse  Mantisse
```

Nun schieben wir die 4 Mantissen nach rechts

```
00011111  11111111  11111111  11111111
76543210  76543210  76543210  76543210
  |         |         |         |       |
  28        23        15        7       0
```

und rechnen manuell mit folgendem Applesoft-Programm nach

```
10 FOR X = 0 TO 28
20 F = F + 2 ↑ X
30 NEXT : PRINT F
```

Wir erhalten wieder die Zahl 536870911 durch Addition von 1 + 2 + 4 ...67108864 + 134217728 + 268435456.

Bei dem Exponenten %10011101 denken wir uns Bit 7 weg und erhalten %00011101, d.h. dezimal 29. Wenn wir rechnen 2 ↑ 29 - 1, kommen wir wieder zu 536870911. Soviel als vereinfachtes Anschauungsbeispiel zu den Binärzahlen.

1.2. Datenregister und Programmzähler

Der 6502 hat bekanntlich 3 Datenregister, die jeweils nur genau 1 Byte = 8 Bits umfassen können. Deshalb spricht man auch vom 8-Bit-Datenbus. Von der

Breite des Datenbusses hängt die Geschwindigkeit des Datentransfers ab. EDV-Anlagen haben meist einen etliche Bytes breiten Datenbus.
Die Register heißen beim 6502:

A = Akkumulator oder A-Register
X = Indexregister X oder X-Register
Y = Indexregister Y oder Y-Register

Ferner hat der 6502 einen 16-Bit-Programmzähler = Program Counter = Adreßregister PCL-PCH. Damit können exakt $10000 = 65536 Speicherstellen, numeriert von $0000 bis $FFFF, adressiert werden. Der Programmzähler als Adreßregister erhält während des Programmablaufs jeweils die Adresse des nächsten zu bearbeitenden Befehls, wobei JMP, JSR, bedingte Verzweigungen usw. den Programmzähler entsprechend abändern. Da der Programmzähler 16 Bit breit ist, spricht man auch vom 16-Bit-Adreßbus. Die Breite des Adreßbusses bestimmt den Adressierungsbereich. Ohne Bank-Switching können beim Apple nur 64K (1K = 1024 Bytes) adressiert werden. Mit Bank-Switching können zwar auch nur 64K adressiert werden, doch sind dann diese 64K eine Teilmenge der Gesamt-Kilobyte-Zahl. Beispiel: Wenn ein Apple IIe eine 64K-Karte hat, kann man mit demselben, einzigen 6502-Prozessor von den „unteren" 64K auf die „oberen" 64K springen. Man hat dann – hardwaremäßig gesehen – „andere" 64K als vorher, doch nach wie vor nur einen Adreßbereich von insgesamt 64K.

1.3. Adressierungsarten

Eine Adresse kann in einem Assemblerlisting folgendermaßen dargestellt werden:

```
        JMP   769       ;Dezimaladresse

        JMP   $0300     ;Hexadezimaladresse

LABEL   EQU   $0300     ;Labeldefinition
        JMP   LABEL     ;Labeladresse
```

Ein 6502-Befehl umfaßt 1-3 Bytes: ein Operationscode-Byte (Opcode, Op) sowie ggf. 1-2 Adreßbytes. Das Opcode-Byte wird vom Assembler als 3stelliges Buchstaben-Mnemonic (Kürzel) dargestellt. Beispiele:

```
20  00  10      JSR   $1000      ;3-Byte-Befehl
OP  LL  HH      MNE   Adresse

85  CE          STA   $00CE      ;2-Byte-Befehl
OP  LL          MNE   Adresse

EA              NOP               ;1-Byte-Befehl
OP              MNE
```

Die Abarbeitung eines Befehls erfordert 2-7 Takte. Bei dem normalen 1-MHz-6502-Prozessor entspricht 1 Takt = 1 Millionstel Sekunde. Geht man davon aus, daß im Mittel 4 Takte pro Befehl benötigt werden, dann können in 1 Sekunde 1000.000 : 4 = 250.000 Befehle abgearbeitet werden. Die Taktzahl wird durch die Art des Befehls und die Adressierungsart bestimmt. Es gibt insgesamt 13 Adressierungsarten:

Akkumulator-Adressierung (eigentlich gar keine Adressierung), z.B.

```
ASL
ROL
```

Implizite Adressierung (eigentlich gar keine Adressierung), z.B.

```
CLC
TXA
INY
```

Unmittelbare Adressierung (lädt Register mit demjenigen Byte, das im Speicher unmittelbar auf den Opcode LDA usw. folgt), z.B.

```
LDA   #$AA
LDA   #<LABEL
LDA   #„A“
```

Einfache Zero-Page-Adressierung (bei der das High Byte der Adresse stets $00 ist), z.B.

```
LDA   $00CE       ;LDA $HHLL, wobei HH = 00
LDA   $CE         ;LDA $LL, wobei HH = 00 automatisch hinzukommt
```

Einfache, absolute Adressierung, z.B.

```
LDA   $FFFF
```

Indizierte, absolute Adressierung (mit X und Y), z.B.

```
LDA   $1000,X
LDA   $1000,Y
```

Mit der indizierten, absoluten Adressierung kann man einen Speicherbereich von 256 Bytes „bearbeiten". So verschiebt etwa das nachfolgende Programm den Bereich $1000-$10FF nach $1100-$11FF:

```
        LDX   #0
LOOP    LDA   $1000,X
        STA   $1100,X
        INX
        BNE   LOOP
        RTS
```

Relative Adressierung bei Verzweigungen, z.B.

```
BEQ   LABEL1
BCC   LABEL2
```

Man beachte, daß man bei relativer Adressierung nur 128 Speicherstellen vorwärts (plus $00-$7F) oder rückwärts (minus $80-$FF) verzweigen kann. Bei größeren Sprüngen muß man die Vergleichslogik umdrehen. Beispiel:

```
        ORG   $1000
        LDA   $00
        BEQ   $2000       ;Dies würde nicht gehen
        RTS

        ORG   $1000
        LDA   $00
        BNE   WEITER      ;Wenn ungleich 0, dann weiter
        JMP   $2000       ;Wenn 0, dann nach $2000
WEITER  RTS
```

Indirekte, indizierte Zero-Page-Adressierung (nur mit Y-Register), z.B.

```
LDA   ($CE),Y
```

Da diese wichtige Adressierungsform oft nicht richtig verstanden wird, hier nochmals ein Beispiel zur Erläuterung. Angenommen, die Speicherstelle $00CE-$00CF enthalte 00CE: 00 10 LL HH. Ferner sei Y = $FF und die Speicherstelle $10FF enthalte 10FF: AA. Dann bewirkt LDA ($CE), Y folgendes: Zuerst wird der Prozessor mit der in $00CE-$00CF enthaltenen Adresse $1000 geladen. Dann wird $1000 um den Y-Registerwert $FF auf $10FF erhöht. Und schließlich wird der Wert $AA der Speicherstelle $10FF in das A-Register geladen. Mit dieser indirekten, Y-indizierten Adressierungsform kann man einen Speicherbereich von 64K „bearbeiten". Beispiel:

```
IND1A   EQU  $00
IND2A   EQU  $02
ANF1    EQU  $2000
END1    EQU  $4000
ANF2    EQU  $6000
* $0000: 00 20
        LDA  #<ANF1   ;#$00
        STA  IND1A    ;$0000
        LDA  #>ANF1   ;#$20
        STA  IND1A+1  ;$0001
* $0002: 00 60
        LDA  #<ANF2   ;#$00
        STA  IND2A    ;$0002
        LDA  #>ANF2   ;#$60
        STA  IND2A+1  ;$0003
* $2000-$3FFF nach $6000-$7FFF moven
        LDY  #0
LOOP    LDA  (IND1A), Y
        STA  (IND2A), Y
        INY
        BNE  LOOP
        INC  IND1A+1  ;$0001
        INC  IND2A+1  ;$0003
        LDA  IND1A+1  ;$0003
        CMP  #>END1   ;#$40
        BNE  LOOP
        RTS
```

Einfache, indirekte Adressierung nur bei JMP, z.B.

```
JMP    ($FFFC)
```

Ferner gibt es:
(a) Indizierte Zero-Page-Adressierung (z.B. LDA $00,X oder LDX $00,Y)
(b) Indirekte Zero-Page-Adressierung (nur mit X-Register, z.B. LDA ($00,X))
Diese Adressierungsarten können Sie vergessen, da sie praktisch nie vorkommen.

1.4. P-Register

Der 6502 hat neben den drei 8-Bit-Datenregistern A, X, Y und dem 16-Bit-Adreßregister (Program Counter) außerdem noch einen 8-Bit-Stackpointer (ergänzt um ein neuntes Bit) sowie ein 8-Bit-Prozessor-Status-Register (P-Register), das folgende Flags umfaßt:

```
7 6 5 4 3 2 1 0
N V - B D I Z C
```

Ein Flag wird gesetzt, wenn eine Bedingung zutrifft, und zurückgesetzt, wenn sie nicht zutrifft.

N-Flag (Negativ-Flag; Bit-7-Flag): BMI, BPL

N = 1, wenn z.B. A mit einem Wert mit Bit 7 on geladen wird.

```
LDA   #%11111111
BMI   JA          N = 1

LDA   #%01111111
BPL   JA          N = 0
```

V-Flag (Überlauf-Flag, Bit-6-Flag): BVS, BVC

V = 1, wenn z.B. beim BIT-Befehl die getestete Speicherstelle Bit 6 on hat.

```
LDA   #%01000000
STA   $1000
```

```
...
BIT   $1000
BVS   JA          V = 1

LDA   #%10111111
STA   $1000
...
BIT   $1000
BVC   JA          V = 0
```

Der BIT-Befehl verändert nicht den Akkumulator-Inhalt und auch nicht die getestete Speicherstelle, sondern setzt lediglich die entsprechenden Flags.

B-Flag (= Break-Flag)
I-Flag (= Interrupt-Flag)
(Zu beiden siehe Ende von Kap. 1.5)

D-Flag (= Dezimal-Flag)

Die Apple-ROM-Routinen erwarten in der Regel, daß das Dezimal-Flag – mit z.B. CLD – zurückgesetzt ist. Andernfalls werden z.B. die Bildschirmadressen usw. falsch, d.h. im BCD-Modus, ausgerechnet. Normalerweise braucht man sich nicht um das D-Flag zu kümmern, da es durch Software-Reset bei NEW-MON ($FA81) auf Null gesetzt wird. Im Zweifelfall muß man es selbst mit CLD zurücksetzen, da sonst „wundersame Dinge" geschehen.

Z-Flag (Zero-Flag): BNE, BEQ

Z = 1, wenn z.B. nach einem Ladevorgang der Registerwert $00 ist oder wenn zwei gleiche Werte verglichen werden.

```
LDA   #$00
BEQ   JA          Z = 1

LDA   #$FF
BNE   JA          Z = 0

LDA   #$10
CMP   $1000
BEQ   JA          ;wenn 1000:10
BNE   NEIN        ;wenn 1000: < > 10
```

C-Flag (Carry-Flag, Übertragsflag): BCC, BCS

Das Carry-Flag ist das wichtigste Flag für Vergleichsoperationen und soll daher hier besonders behandelt werden. Wenn man A (oder ersatzweise X bzw. Y) mit einer Speicherstelle (Memory M) vergleicht, ergeben sich zunächst nur 2 Vergleiche:

Wenn A > = M, dann BCS, also C-Flag = 1
Wenn A < M, dann BCC, also C-Flag = 0

Um die Vergleiche A > M und A < = M zu implementieren, sind Doppeltverzweigungen erforderlich. Ferner beachte man, daß bei BCC-BCS-Vergleichen die A- und M-Werte stets als Hexbytes ohne Vorzeichen angesehen werden. Konkrete Beispiele:

```
A > = M     LDA   #$F0
            CMP   MEMORY
            BCS   JA          ;wenn A > = M, d.h. M $00-$F0
WEITER      ...
JA          ...

A < M       LDA   #$F0
            CMP   MEMORY
            BCC   JA          ;wenn A < M, d.h. M $F1-$FF
WEITER      ...
JA          ...

A > M       LDA   #$F0        ;Doppelvergleich erforderlich
            CMP   MEMORY
            BEQ   WEITER      ;wenn A = M, d.h. M $F0
            BCS   JA          ;wenn A > M, d.h. M $00-$EF
WEITER      ...
JA          ...


A < = M     LDA   #$F0        ;Doppelvergleich erforderlich
            CMP   MEMORY
            BEQ   JA          ;wenn A = M, d.h. M $F0
            BCC   JA          ;wenn A < M, d.h. M $F1-FF
WEITER      ...
JA          ...
```

1.5. Stackpointer

Der Stack oder Stapel („Keller") ist der feste Speicherbereich $0100-$01FF, der von oben, also von $01FF, mit Rücksprungadressen aufgrund von JSR gefüllt wird. Der Stackpointer SP ist ein spezielles 6502-Register, das auf die momentan verfügbare Stelle des Stacks zeigt. Als Beispiel nehmen wir an, der SP zeige auf $01FF und es würde der nachfolgende Befehl ausgeführt:

```
1000:   20 00 40      JSR   $4000
1003:   ...
4000:   60            RTS
```

Zunächst decodiert der Prozessor bei $1000 den Opcode $20 als JSR. Da er sich immer nur eine einzige Adresse in seinem internen Adreßregister merken kann, schiebt er die Adresse des *zweiten* Bytes nach JSR, also die Stelle von HH = $40 = $1002 auf den Stack in der HH-LL-Form (von „oben" gesehen):

```
01FF: 10 HH  auf Stack schieben (Push), dann SP = SP -1 (Decrement)
01FE: 02 LL  auf Stack schieben (Push), dann SP = SP -1 (Decrement)
01FD: ??     SP = $FD nach JSR und vor RTS
```

Jetzt lädt der Programmzähler die Adresse $4000 und führt im Anschluß daran die dortige Routine aus, die bei dem Beispiel nur aus einem nackten RTS besteht. Dieses RTS wird vom Prozessor decodiert und bewirkt folgendes:

```
SP = SP + 1 (Increment), dann $02 LL vom Stack holen (Pull)
SP = SP + 1 (Increment), dann $10 HH vom Stack holen (Pull)
SP jetzt wieder $FF
```

Dann wird Rücksprungadresse $1002 *um 1* auf $1003 erhöht und dort der nächste Befehl decodiert.

Merke: Erst Push, dann Decrement!
Erst Increment, dann Pull!

Der Stackpointer wird völlig initalisiert mit

```
LDX   #$FF        (LDX #$DF)
TXS
```

Meist empfiehlt sich jedoch ein etwas niedrigerer Wert, z.B. $DF, um ein Durchdrehen nach „oben" zu vermeiden.

Mit PHA kann man den A-Inhalt auf den Stack schieben und mit PLA wieder vom Stack ziehen. Da der Stack nur 256 Bytes umfaßt (gegenüber etwa dem Z-80-Prozessor), ist es beim 6502 ein schlechter Programmstil, permanent Daten auf den Stack zu schieben. Alle ROM-Routinen benutzen ihn nämlich bereits sehr intensiv, weil sie keine anderen Zwischenspeicherungsmöglichkeiten haben. Ferner benutzt Applesoft den unteren Stackbereich $0100-$0110 für die Umwandlung von FAC-Zahlen zwecks Ausgabe über COUT usw. Jeder aktive JSR benötigt zwar nur 2 Stellen auf dem Stack, dagegen bereits jeder aktive Applesoft-FOR-NEXT-Loop über 15 Bytes, von verschachtelten GO-SUBs und mathematischen Zwischenergebnissen bei FRMEVAL sowie von DOS ganz zu schweigen. Da kann man sich leicht ausrechnen, wann ein Stapelüberlauf auftreten wird. Fazit: PHA und PLA nur spärlich verwenden.

Ähnliches gilt für die Zero-Page. Wenn man statt z.B. STA $0300 (4 Takte) einen Wert mit STA $00 (3 Takte) „wegdrückt", spart man 1 Takt. Ist das Programm damit „25% schneller"? Wohl nur dann, wenn diese Zero-Page-Zwischenspeicherung tausend und aber tausendmal vorkommt. Denn 1 tausendmal wären ja nur 1 tausendstel Sekunde! Fazit: Zero-Page in erster Linie für indirekte Adressierung und erst in zweiter Linie für Datenspeicherung benutzen.

Das P-Register kann man mit PHP auf den Stack retten und mit PLP wieder vom Stack holen. Will man das P-Register insgesamt initialisieren, dann verfahre man folgendermaßen:

```
LDA  #0
PHA
PLP
```

Wenn der 6502 auf den BRK-Befehl (Opcode $00) stößt, schiebt er die BRK-Adresse + 2 sowie das P-Register auf den Stack, setzt das Break-Flag auf 1 und springt dann indirekt zur IRQ-Adresse, die bei $FFFE-$FFFF abgelegt ist (in der Regel $FA40). Auch das soll durch ein Beispiel verdeutlicht werden. Nehmen wir an, der SP sei auf $FF initialisiert, und bei Speicherstelle $031C, wo sich der 6502 gerade programmzählermäßig befände, wäre $00. Dann würde der Stack nach Abarbeitung des BRK-Befehls so aussehen:

```
01FF:    03 HH $031C + 2 = $031E
01FE:    1E LL
01FD:    30 = %00110000 P-Register
                76543210
                   B
```

Ähnliches gilt für Reset, wobei allerdings der Reset-Vektor aus der Speicherstelle $FFFC-$FFFD geholt wird (beim Apple II Plus und IIe $FA62).

1.6. Befehlsarten

LDA, LDX, LDY; STA, STX, STY

Diese Lade- und Speicherbefehle werden am häufigsten benutzt. Die Ladebefehle berühren nur das N- und Z-Flag, die Speicherbefehle gar keine Flags.

TAX, TXA; TAY, TYA

Diese Transferbefehle sind gelegentlich ganz nützlich, werden jedoch insgesamt nicht sehr häufig benutzt. Befehle in der Art TXY oder TYX fehlen.

TSX, TXS; PHA, PLA; PHP, PLP

Der Stackpointer SP kann nur über das X-Register erreicht und verändert werden. Man beachte ferner, daß es keine Möglichkeit gibt, den Programmzähler direkt zu ermitteln. Bei relokativen Programmen, die überhaupt keine JMPs und nur JSRs zu bekannten ROM-Routinen enthalten und die deshalb in beliebige Speicherbereiche geBLOADet werden können (siehe z.B. „IIE.64K?“ in Kap. 4.2.4), muß man oft ermitteln, wo sich die Startadresse des eigenen Programms befindet. Ein Beispiel soll dies erläutern. Zu diesem Zweck nehmen wir an, daß das nachfolgende Programm nach $0300 geladen worden sei, was jedoch zunächst unbekannt ist.

```
0300:    JSR   $FF58      ;Monitor-Stelle, wo ein RTS steht
0303:    TSX
0304:    DEX              ;SP = SP - 1
0305:    SEC
0306:    LDA   $100,X     ;Low Byte
0309:    SBC   #2         ;$0302 - $02 = $0300
```

```
030B:    STA   $FE       ;LL
030D:    INX              ;SP = SP + 1
030E:    LDA   $100,X    ;High Byte
0311:    SBC   #0
0313:    STA   $FF       ;HH
0314:    ...
```

Das Programm springt zunächst zu einer bekannten RTS-Stelle, wodurch $0300 + $02 = $0302 in der Form HHLL auf den Stack geschoben werden, d.h. erst HH, dann LL. Nach RTS zeigt der SP auf HH, so daß wir zwecks Subtraktion von 2 das X-Register zunächst dekrementieren und später wieder erhöhen müssen. Später findet sich die Startadresse $0300 in $00FE-$00FF LLHH.

CLC, SEC, CLI, SEI, CLD, SED, CLV

Diese Befehle dienen zum Setzen bzw. Zurücksetzen verschiedener, wenngleich nicht aller Flags. Das fehlende SEV (Set Overflow Flag) könnte man durch BIT $FF58 (bekanntes RTS) simulieren, da der Opcode für RTS $60 = %01100000 ist, bei dem Bit 6 gesetzt ist. Die fehlenden CLN (Clear Negative Flag), SEN (Set Negative Flag), CLZ (Clear Zero Flag) und SEZ (Set Zero Flag) könnte man ebenfalls simulieren, doch besteht hierfür kaum Bedarf.

JSR, JMP; RTS, RTI

Der indirekte JMP ($HHLL) wird gelegentlich benutzt, wenn man eine Adressentabelle für indirekte JMPs ($LLHH) im Speicher angelegt hat, was z.B. bei PRODOS häufiger vorkommt. Eine andere, etwas skurrile Methode ist die Stack-Sprungtabelle, die man z.B. im Applesoft-Interpreter ab $D000-$D0B1 für alle Basic-Befehle findet. Betrachten wir zu diesem Zweck den dort für Ampersand aufgeführten Sprung:

```
D05E: F4 03   = $03F4 = $03F5 - $01
```

Die eigentliche Ampersand-Routine beginnt bei $03F5, während in der Sprungtabelle $03F4 steht. Der Sprung wird über den Stack wie folgt realisiert:

```
D834:    LDA   $D001, Y   ;High Byte $03
D837:    PHA               ;auf Stack pushen
D838:    LDA   $D000, Y   ;Low Byte $F4
D83B:    PHA               ;auf Stack pushen
```

```
D83C:   JMP   CHRGET  ;RTS-Exit via CHRGET
```

Da RTS – wie bereits erwähnt – die „Rücksprungadresse" *um 1* erhöht, erfolgt nach CHRGET der Sprung zu $03F5.

RTI kommt normalerweise nur am Ende von Interrupt-Handler-Routinen vor (außer bei ProDOS). RTI holt P-Register, LL und HH (in dieser Reihenfolge) vom Stack, erhöht jedoch danach *nicht* die Rücksprungadresse $HHLL *um 1*, wie dies bei RTS üblich ist.

CMP, CPX, CPY; BIT; BMI, BPL, BEQ, BNE, BCS, BCC, BVS, BVC

Die Vergleichsbefehle wurden bereits oben erwähnt. Man beachte, daß mit CMP, CPX und CPY nur die Flags N, Z und C (und *nicht* V) gesetzt/zurückgesetzt werden. Bei BIT sind nur nicht-indizierte Zero-Page- und Absolutadressierung möglich.

INC, INX, INY; DEC, DEX, DEY

Die Inkrementierungs- und Dekrementierungsbefehle lassen nur die X-indizierte Adressierung zu, wie überhaupt bei vielen Befehlen die Adressierungsarten nicht systematisch ausgebaut sind.

ADC und SBC

Die hexadezimale Addition muß stets durch CLC eingeleitet werden. Ein Überlauf fand nicht statt, wenn nach der Subtraktion C = 0 ist.
Die hexadezimale Subtraktion muß stets durch SEC eingeleitet werden. Ein Überlauf (besser „Unterlauf") fand nicht statt, wenn nach der Subtraktion C = 1 ist.

Beispiele:

```
CLC
LDA   #$10
ADC   #$10      ;= $20
BCS   ERROR     ;trifft hier nicht zu!
BCC   OKAY
```

```
CLC
LDA   #$80
ADC   #$80        ;$0100
BCS   ERROR       ;trifft hier zu!
BCC   OKAY

SEC
LDA   #$80
SBC   #$80        ;= $00
BCC   ERROR       ;trifft hier nicht zu!
BCS   OKAY

SEC
LDA   #$10
SBC   #$11        ;= $FF („minus 1“)
BCC   ERROR       ;trifft hier zu!
BCS   OKAY
```

Man beachte, daß die Increment- und Decrement-Operationen (INC, INX, INY; DEC, DEX, DEY) das Carry-Flag nicht berühren, so daß man bei mehrfach-genauer Addition und Subtraktion entsprechende BEQ-BNE-Schleifen aufbauen kann.

AND, ORA, EOR

Dies sind die „Booleschen“ Bit-Maskierungsbefehle.

```
        LDA   #%11100001 „a“ mit Bit 7 on
        AND   #%01111111 Logische Und-Maske
ergibt  A =   #%01100001 „a“ mit Bit 7 off

        LDA   #%01100001 „a“ mit Bit 7 off
        ORA   #%10000000 Logische Oder-Maske
ergibt  A =   #%11100001 „a“ mit Bit 7 on
```

AND dient zum Zurücksetzen unerwünschter Bits, ORA zum Setzen gewünschter Bits.

```
        LDA   #%10101010
        EOR   #%11111111 Logische Entweder-Oder-Maske
ergibt  A =   #%01010101
```

EOR mit $FF dient zum Invertieren von Bit-Mustern, EOR mit anderen Werten als $FF zum Chiffrieren oder um bei trickreichen mathematischen Operationen hier und dort ein Byte einzusparen (siehe z.B. die PLOT1-Unterroutine im F8-Monitor ab $F80E).

Man beachte, daß AND, ORA und EOR immer nur den Akkumulator und nicht eine Speicherstelle direkt ändern.

ASL, LSR; ROL, ROR

Mit diesen Befehlen kann man Bit-Muster nach links und rechts verschieben, und zwar nicht nur beim A-Register, sondern auch direkt bei Speicherstellen. In den Assemblerlistings dieses Buches wird statt
ASL A (MOS-Syntax)
lediglich
ASL (heutige Syntax)
als Mnemonic benutzt.

ASL verdoppelt den A-Wert, LSR teilt ihn durch 2:

```
            LDA   #%00001111= dezimal 15
            ASL               Linksverschiebung
danach      A =   #%00011110= dezimal 30

            LDA   #%11110000= dezimal 240
            LSR               Rechtsverschiebung
danach      A =   #%01111000= dezimal 120
```

Viermaliges ASL/LSR wird oft benutzt, um das Low Nibble/High Nibble nach links/rechts zu schieben. Ein ASL überträgt Bit 7 in das Carry-Flag und ein LSR überträgt Bit 0 in das Carry-Flag. Im übrigen fallen die mit ASL oder LSR geshifteten Bits „unter den Tisch".

Anders bei ROL und ROR. ROL shiftet wie ASL nach links, übernimmt Bit 7 als neues C-Flag und überträgt das alte C-Flag in Bit 0. Nach achtmal ROL wäre man wieder beim Ausgangswert angelangt. ROR ist das Gegenstück zu ROL. ROL/ROR werden namentlich bei Multiplikation/Division benötigt, insbesondere in den Kombinationen ASL-LL ROL-HH und LSR-HH ROR-LL.

Vereinfachtes Multiplikationsbeispiel:

```
        LDX   #4          ;2 ↑ 4 * 240 = 3840
LOOP    ASL   LLBYTE      ;angenommen LL vorher %11110000 = 240
        ROL   HHBYTE      ;angenommen HH vorher %00000000 = 0
        DEX
        BNE   LOOP
        RTS               ;HHLL nachher %00001111 00000000 = 3840
```

NOP, BRK

NOP dient erstens zum „Lahmlegen" von unerwünschten Maschinenprogrammstellen sowie zweitens zur „Feinabstimmung" von Warteschleifen (1 NOP = 2 Takte). BRK wird zum Debuggen verwendet (Breakpoint-Setzen).

1.7. Hardware-Bugs

Abschließend sei auf 2 Prozessor-Bugs hingewiesen:

Wenn der Programmzähler bei einem z.B. noch nicht ausgetesteten Programm auf ein Byte stößt, daß kein Opcode ist, z.B. $02, dann „hängt" der Prozessor meist. Beispiel:

```
CALL-151
300:2
300G
```

Wenn beim indirekten Sprung die Adresse JMP ($??FF) beträgt, wird die Page verwechselt („??" als HH beliebig, $FF als LL in dieser Form).

Diese Bugs sind bei den neueren Versionen 65C02 usw. behoben worden, bei denen übrigens einige weitere Befehle hinzugekommen sind. Der neue Apple IIc hat z.B. den neuen 65C02-Prozessor, und in dessen ROM-Routinen werden die neuen Befehle bereits benutzt. Dies ist insofern unerheblich, als ROM + Prozessor quasi zusammengehören. Ansonsten werden in diesem Buch die neuen Befehle nicht benutzt, um Kompatibilität mit allen Geräten sicherzustellen.

1.8. Hinweis zu den Listings

Die nachfolgenden Assemblerlistings wurden alle mit dem BIG-MAC- bzw. MERLIN-Assembler erstellt. Neben den Standard-MOS-Mnemonics wurde nur ganz wenige Pseudo-Opcodes verwendet, damit sich die Konvertierung in einen anderen Assembler problemlos gestaltet:

ORG = Startadresse für das Programm (Origin)
EQU = Labeldefinition (LABEL equates soundso)
HEX = Definition von Hexzahl(en) (kein $ vorangestellt)
DFB = Definition von Datenbyte(s) (bei Hexzahl mit $)
ASC = ASCII-String-Definition („abc" Bit 7 on; 'abc' Bit 7 off)
DA = Definition einer Adresse $LLHH
DS = Speicherreservierung (Define Storage)

Vollständige Kommentarzeilen beginnen mit einem Sternchen, Kommentare am Ende einer Befehlszeile mit einem Semikolon.

LDA #<ADRS (oder LDA #ADRS) lädt LL der Adresse, LDA #>ADRS lädt HH der Adresse.
Gelegentlich findet man (siehe z.B. BRUTAL.CLEAR in Kap. 3.2.6):
LDA #ADRS2-ADRS1-1
Wenn ADRS2 = $0347 und ADRS1 = $032E, dann ADRS2 - ADRS1 = $0019 und #ADRS2-ADRS1-1 = $18 = LL
LDA: $0000 (man beachte den Doppelpunkt nach LDA) bedeutet erzwungene Nicht-Zero-Page-Adressierung, d.h. absolute Adressierung.

Von den bekannten F8-ROM-Adressen wurden deren Label-Namen übernommen. Zwei Ausnahmen: PRBYTE nenne ich oft HEXOUT und COUT nenne ich oft PRINT. Bei Applesoft-ROM-Routinen wurde die „CALL-A.P.P.L.E"-Terminologie verwendet.

2. Monitor-ROM

Unter Monitor verstehen wir erstens eine Sammlung von Assemblerroutinen, die sich im ROM $F800-$FFFF befinden, sowie zweitens eine Teilmenge dieser Routinen, die als Befehle (L, G usw.) nach CALL -151 zur Verfügung stehen. Im einzelnen sind zu unterscheiden:

a) System Monitor des Apple II

Dieser enthielt ab $FB60 eine Multiplikationsroutine, ab $FB81 eine Divisionsroutine und ferner als Debugger Step/Trace ab $FEC4/$FEC2. Der Reset-Vektor war auf eine Stelle kurz vor dem Monitor-Entry ($FF59) gelegt.

b) Autostart Monitor des Apple II Plus

Die für den System Monitor genannten charakteristischen Routinen fehlen hier. Dafür wurde freie Cursorbewegung implementiert. Der Reset-Vektor wurde jetzt auf $FA62 gelegt, wo beim Einschalten des Apple II Plus über $FAA6 die Disk-Boot-Routine auslöst wird (PWRUP = Power up).

c) Apple IIe Monitor

Der Apple IIe Monitor unterscheidet sich vom Autostart Monitor ganz wesentlich dadurch, daß neben dem Bereich $F800-$FFFF (F8-Monitor) auch der interne ROM-Bereich $C100-$CFFF (INTCXROM-Monitor) benutzt wird, und zwar auch dann, wenn keine 80-Zeichenkarte im Slot 3 steckt. Bei den Routinen (Y-Register enthält Befehlsnummer)

0 = CLREOP
1 = HOME
2 = SCROLL
3 = CLREOL

4 = CLREOLZ
5 = INIT und RESET
6 = KEYIN
7 = FIX ESCAPE CHAR
8 = SETWND

springt der F8-Monitor über $FBB4 nach $C100 zum INTCXROM-Monitor und kommt von dort wieder bei $FD29 bzw $FD2C zum F8-Monitor zurück.

d) Apple IIc Monitor

Gegenüber dem Apple IIe sind $D000-$FFFF sowie $C100-$CFFF stark umgeschrieben worden, so daß viele Detailadressen nicht mehr stimmen; daher in diesem Buch nicht berücksichtigt.

Wenn man von Applesoft-Basic aus CALL -151 eingibt, kommt man in den Befehlsmodus des Monitors (erkennbar * Cursor).

MONZ = CALL -151 = $FF69 (normaler Monitor-Entry)
MON = CALL -155 = $FF65 (CLD Dezimalmodus, dann BELL, dann MONZ)

Den Monitor kann man wieder verlassen mit

Ctrl-C	entspricht E003G Basic-Warmstart
Ctrl-B	entspricht E000G Basic-Kaltstart

Normalerweise empfiehlt sich Ctrl-C

Im Monitor hat man folgende Befehle zur Verfügung, die jeweils zur Wiederholung mit kurzen Beispielen erläutert werden.

a) Hexdump (Peeken)

0300 Return	zeigt Inhalt von $0300 als Hexzahl an (Einzeldump)
300 Return	dasselbe
300.3CF Return	Hexdump des Bereichs $0300-$03CF (Bereichdump)
300.301 Return	Bereichdump $0300-$0301

300 301 Return	Einzeldumps von $0300 und $0301

Einzeldumps im Softswitch-Bereich $C000-$C0FF sollten nicht unbedacht vorgenommen werden und Bereichdumps überhaupt nicht. Beispiele:

C030 Return	klickt Lautsprecher
C081 C081 Return	macht LC schreibfähig (ungefährlich)
C083 C083 Return	macht LC schreib- und lesefähig (gefährlich!)

(Vergleiche hierzu HEXDUMP-Programm in Kap. 2.2.9.)

b) Disassembler (L)

300L Return	disassembliert 20 Zeilen ab Speicherstelle $0300
1000LLLLLLLLL Return	disassembliert 10 mal 20 Zeilen ab $1000

Auch hier gilt, daß die Disassemblierung im Softswitchbereich gefährlich ist.

(Vergleiche hierzu DISASSEMBLER-Programm in Kap. 2.2.10.)

c) Speicheränderung (Poken)

300: 20 58 FC 60 Return	überschreibt $0300-$0303 mit den Bytes
300: 1 2 3 4 Return	dasselbe (Hex-Nibbles werden als Low-Nibbles interpretiert)

Es können ca. 128 Hexbytes auf einmal eingegeben werden.

d) Programmstart (G)

300G Return	startet Maschinenprogramm ab $0300

Wenn das Maschinenprogramm mit RTS endet und der Stackpointer nicht manipuliert wird, gelangt man später wieder in den Monitor zurück. Dies ist z.B. bei E003G nicht der Fall.

e) Move (M)

1600<C600.C6FFM Return	verschiebt $C600-$C6FF nach $1600

Die Monitor-Move-Routine ist ein Vorwärts-Move, der bei Bereichsüberlappungen nicht funktioniert. Dies kann man ausnutzen, um z.B. einen Speicherbereich mit einem Hex-Muster zu füllen:

800:0
801<800.95FEM Return löscht Bereich $0800-$95FF

Man beachte, daß wegen der Überlappung hier als Endadresse $95FE statt $95FF angegeben wurde.

(Vergleiche hierzu BACKMOVE-Programm in Kap. 2.2.3. sowie das Vorwärts-Move-Programm EINZELPOKE in Kap. 4.2.1.)

f) Verify (V)

1600<C600.C6FFV Return vergleicht $C600-$C6FF mit Bereich ab $1600.

g) Slot-Aktivierung

1 Ctrl-P Return aktiviert z.B. Drucker in Slot 1
0 Ctrl-P Return stellt Drucker wieder ab

h) Delimiter I und N

Der Befehl I bewirkt inverse und der Befehl N wieder normale Bildschirmdarstellung. Dies ist an sich wenig nützlich. Deshalb kann man diese Zeichen N und I als Delimiter (neben G, L usw.) benutzen, um Mehrfach-Pokebefehle in einer einzigen Zeile einzugeben.

300: 20 ED FD 60 N 1000:20 ED FD 60 N 300L 1000L 300G 1000G Return

(Vergleiche hierzu das LAMPOKE-Demo in Kap. 4.2.1.)

i) Monitor-Register-Dump

Ctrl-E Return zeigt Register in der Form A, X, Y, P, S an

Unmittelbar danach kann man mit z.B.

:0 0 0 0 0 Return

die Register auf Null setzen.

Ctrl-E N :C1 N 300:4C ED FD N 300G Return würde „A“ anzeigen

Man beachte, daß die angezeigten Register die vom Monitor in der Zero-Page zwischengespeicherten Werte und nicht die tatsächlichen momentanen Registerwerte sind. Der G-Befehl übernimmt immer diese Monitor-Registerwerte, so daß sie zum Debuggen kurzer Routinen verwendet werden können.

j) 8-Bit-Addition und -Subtraktion

```
3A+3B Return
FF-A0 Return
FF+FF Return     (Überlauf)
10-20 Return     („Unterlauf“)
```

(Vergleiche hierzu ADD.SUB-Programm in Kap. 2.2.4.)

k) Ctrl-Y-Befehl

Nach Ctrl-Y Return erfolgt ein Sprung zur Adresse $03F8. Dort kann man einen JMP zu einer selbstgeschriebenen Assemblerroutine implementieren. (Vergleiche hierzu Programm RAMSUCHER in Kap. 2.2.12.)

2.1. Interne Monitor-Routinen

2.1.1. Zero-Page-Adressen

$0020	WNDLFT	(0-39)	linker Rand (normal 0)
$0021	WNDWDTH	(1-40)	Fensterbreite (normal 40 oder 80 bei 80 Z/Z)
$022	WNDTOP	(0-22)	oberer Rand (normal 0)
$0023	WNDBTM	(1-24)	unterer Rand + 1 (normal 24)

Durch Pokes kann man das Bildschirmfenster begrenzen. Bevor man jedoch das Fenster definiert, muß man den Cursor innerhalb desselben positionieren. Bei illegalen Werten dreht die COUT1-Routine durch mit der Folge, daß außerhalb des Bildschirmspeichers $0400-$07FF Zeichen gespeichert werden.
Bei der Apple IIe-80-Z/Z-Karte muß WNDWDTH geradzahlig sein, also z.B. 20 statt 21. Andernfalls wird automatisch abgerundet.

$0024	CH	Horizontale Cursorposition (0-39 oder 0-79)
$0025	CV	Vertikale Cursorposition (0-23)
$0026	GBASL	Lores-Grafik-Bildschirm-Basisadresse
$0027	GBASH	
$0028	BASL	Bildschirm-Basisadresse
$0029	BASH	
$002A	BAS2L	Scroll-Basisadresse (BASL + 1 oder - 1)
$002B	BAS2H	

Die Basisadressen sind die Speicherstellen von HTAB 1, VTAB V (V = 1-24), also die Stellen des linken Randes. In Applesoft werden die Zeilen von 1-24 und im Monitor von 0-23 = $00-$17 numeriert:

00: $0400-$0427	08: $0428-$044F	10: $0450-$0477	$0478-$047F
01: $0480-$04A7	09: $04A8-$04CF	11: $04D0-$04F7	$04F8-$04FF
02: $0500-$0527	0A: $0528-$054F	12: $0550-$0577	$0578-$057F
03: $0580-$05A7	0B: $05A8-$05CF	13: $05D0-$05F7	$05F8-$05FF
04: $0600-$0627	0C: $0628-$064F	14: $0650-$0677	$0678-$067F
05: $0680-$06A7	0D: $06A8-$06CF	15: $06D0-$06F7	$06F8-$06FF
06: $0700-$0727	0E: $0728-$074F	16: $0750-$0777	$0778-$077F
07: $0780-$07A7	0F: $07A8-$07CF	17: $07D0-$07F7	$07F8-$07FF
			„Screenholes“
			Slot 0-7

Wie ersichtlich, sind die Zeilen $00, $08 und $10 speichermäßig ein einziger Bereich. Es gibt insgesamt 8 Makrozeilen zu je 3 Bildschirmzeilen. Am Ende der 8 Makrozeilen befinden sich die Scratchpad-Stellen für Interface-Karten in den Slots 0 bis 7.

Ein Zeichen wird vom Bildschirm mit LDA (BASL), Y „gepeekt“ und mit STA (BASL), Y „gepokt“, wobei Y den CH-Wert 0-39 enthält. Scrollen nach oben geschieht dadurch, daß die Zeile 1 zur Zeile 0, dann Zeile 2 zur Zeile 1 usw. bis Zeile 23 zur Zeile 22 kopiert wird. Am Schluß wird Zeile 23 „ausgeblankt“. Beim Scrollen nach unten wird das Verfahren umgekehrt angewendet. BAS2 wird nur beim Scrollen für die jeweilige Kopierzeile benutzt.

Der Lores-Grafik-Bildschirm ist wie der 40-Z/Z-Bildschirm organisiert mit einem Unterschied: Der für 1 Zeichen des Text-Bildschirms vorgesehene Raum kann nunmehr 2 aufeinandergesetzte Lores-Kästchen aufnehmen.

$002C	H2	(0-39) X2-Endpunkt beim Plotten (siehe Kap. 5.1.6.)
$002D	V2	(0-23) Y2-Endpunkt beim Plotten
$002F	LENGTH	(0-2) des Befehls bei INSTDSP
$0032	INVFLG	Flag: $FF = NORMAL, $3F = INVERSE, $7F = FLASH
$0033	PROMPT	Enthält das bei GETLN benutzte Prompt (*, Ü usw.)
$0036	CSWL	Ausgabe-Vektor (Character Switch; normal $FDF0 COUT1)
$0037	CSWH	
$0038	KSWL	Eingabe-Vektor (Keyboard Switch; normal $FD1B KEY-IN)
$0039	KSWH	

CSWL enthält die nach PR#S und KSWL die nach IN#S gültige Vektoradresse. (Näheres hierüber siehe in „Apple DOS 3.3" und „ProDOS für Aufsteiger".)

$003A	PCL	Enthält Programmzähler beim L-Befehl
$003B	PCH	
$003C	A1L	Source-Startadresse
$003D	A1H	
$003E	A2L	Source-Endadresse
$003F	A2H	
$0042	A4L	Destination-Startadresse
$0043	A4H	

Diese Adressen werden bei Move und Verify sowie bei unseren entsprechenden Programmen benutzt.

$0045	ACC	A
$0046	XREG	X
$0047	YREG	Y
$0048	STATUS	P
$0049	SPNT	S

Zwischenspeicherstellen für die Prozessorregister. Werte können zu Debug-Zwecken gepokt werden.

2.1.2. Monitor-Routinen

Es wird jeweils der Name der Routine, die entsprechende hexadezimale Adresse, der Zweck der Routine sowie – am wichtigsten – die Prozedur (abgekürzt „vorher") genannt. Darunter ist zu verstehen, welche Werte initialisiert werden müssen, bevor die Routine mit JSR aufgerufen werden kann. Wenn keine vorangehende Initialisierung erforderlich ist, so ist dies an dem reinen JSR-Vermerk erkenntlich. Nach „vorher"-Initialisierung erfolgt stets ein JSR, so daß dieser JSR nur dann aufgeführt wird, wenn „nachher" noch ein weiterer Befehl erforderlich ist.
Da die meisten Routinen eines oder mehrere der X-, Y- oder A-Register ändern, sollte man bei Bedarf vorher die Register retten und nachher wieder laden mit:

```
JSR   IOSAVE
JSR   ROUTINE  XYZ
JSR   IOREST
```

IOSAVE (JSR $FF4A): Speichert A, X, Y, P und S in $0045-$0049.

IOREST (JSR $FF3F): Stellt Registerwerte wieder her (außer S!).

Man beachte, daß Interface-Karten oft selbst auf diese Routinen zugreifen, so daß man dann die ursprünglich gespeicherten Werte nicht mehr vorfindet. Deshalb füge man besser eine eigene Save-Restore-Routine in sein Programm ein. Unsere Programme MULTIPLIKATION (Kap. 2.2.6) oder AUXMOVER (Kap. 3.2.8) sind Beispiele hierfür.

2.1.2.1. Lores-Routinen

SETGR (JSR $FB40): Stellt auf Mixed-Screen Lores-Grafik ein.

Für Full-Screen-Lores gibt es keine Monitor-Routine; daher so simulieren:

```
LDA   $C050     GRAFIK
LDA   $C052     NOMIX
LDA   $C056     LORES
LDA   $C054     PAGE1
```

SETTXT (JSR $FB39): Stellt nach SETGR wieder auf Text-Bildschirm zurück. Funktioniert nicht nach HGR2.

INIT (JSR $FB2F): Entspricht dem Applesoft-TEXT-Befehl, d.h. schaltet auch HGR-Grafik ab. Fällt in SETTXT und danach in SETWND.

SETWND (JSR $FB4B): Stellt auf Full-Screen ein, d.h. normalisiert Zero-Page $0020-$0023 (Window-Werte).

PLOT ($F800): Plottet ein Lores-Kästchen bei Schnittpunkt x, y.
(siehe hierzu Kap. 5.1.6. sowie Abbildung S. 200)

vorher: LDY x-Wert(0-39)
LDA y-Wert(0-47)

HLINE ($F819): Plottet einen horizontalen Lores-Balken von x1-x2 in y-Zeile.

vorher: x1 in Y
x2 in H2
y in A

VLINE ($F828): Plottet einen vertikalen Lores-Balken von y1-y2 in x-Spalte.

vorher: x in Y
y1 in A
y2 in V2

CLRSCR (JSR $F832): Löscht gesamten Lores-Bildschirm.

CLRTOP (JSR $F836): Löscht Lores-Zeilen 0-39 (Mixed Screen).

NXTCOL (JSR $F85F): Erhöht Farbnummer um 3. Nach 15 geht es wieder bei 0 los (Wraparound).

SETCOL ($F864): Farbnummer festlegen.

vorher: LDA 0-15 (Farbnummer)

SCRN ($F871): Farbnummer von Schnittpunkt x, y ermitteln.

vorher: LDY x-Wert
LDA y-Wert

GBASCALC ($F847): Berechnet GBASL-Basisadresse.

vorher: LDA 0-23 (Bildschirmzeile)

2.1.2.2. Bildschirm-Routinen

TABV ($FB5B): Vertikaltabulierung; VTAB V

vorher: LDA V (0-23)

VTAB ($FC22): Vertikaltabulierung (etwas umständlicher als TABV).

vorher: LDA V (0-23)
STA CV(=$0025)

HTAB muß man selbst simulieren mit

LDA H (0-39)
STA CH (= $0024)

Bei IIe-80-Zeichenkarte:

LDA H (0-79)
STA $057B

BASCALC ($FBC1): Berechnet Text-Basisadresse.

vorher: LDA 0-23 (Bildschirmzeile)

Man beachte: BASCALC kalkuliert die absolute Basisadresse unabhängig vom möglichen Bildschirmfenster, während TABV und VTAB den ggf. gesetzten linken Bildschirmrand berücksichtigen.

HOME (JSR $FC58): Löscht Bildschirm (oder genauer gesagt Bildschirmfenster) und setzt CV und CH auf 0 (VTAB 1, HTAB 1).

SCROLL (JSR $FC70): Scrollt eine Zeile nach oben.

SCROLLDN ($CCAA): Scrollt eine Zeile nach unten; nur IIe mit 80-Z/Z-Karte!

```
vorher:     STA   $C007     (Enable INTCXROM)
            JSR   $CCAA
nachher:    STA   $C006     (Disable INTCXROM)
```

CLREOP (JSR $FC42): Clear to end of page, d.h. Bildschirm ab Cursorposition bis zum Ende löschen (= mit $A0 füllen). Alte Cursorposition bleibt erhalten.

CLREOL (JSR $FC9C): Clear to end of line, d.h. Zeilenrest ab Cursorposition löschen. Alte Cursorposition bleibt erhalten.

CLREOLZ ($FC9E): Bildschirmzeile ab Y-Wert löschen

```
vorher:   LDY   0-39        (= CH = $0024)
```

ADVANCE (JSR $FBF4): CH um 1 erhöhen. Falls CH bereits 39 (79), dann CR ausführen.

LF (JSR $FC66): Bildschirm-Linefeed, d.h. Cursor 1 Zeile tiefer, wobei CH erhalten bleibt.

CR (JSR $FC62): Bildschirm-Carriage-Return, d.h. 1 Zeile tiefer und CH auf 0 setzen (= auf linken Rand).

BS (JSR $FC10): Bildschirm-Backspace, d.h. CH um 1 vermindern. Falls CH bereits 0, dann 1 Zeile höher und CH auf 39 (79) setzen.

UP (JSR $FC1A): Bildschirm-Reverse-Linefeed, d.h. Cursor 1 Zeile höher, wobei CH erhalten bleibt.

COUT1 ($FDF0): Reine ASCII-Bildschirmausgabe. Funktioniert nur bei 40 Z/Z!

```
vorher:   LDA   Buchstabe   ($80-$FF, also Bit 7 on!)
```

STORADV ($FBF0): Speichert (STORE) beliebigen A-Inhalt, also auch Ctrl-Buchstaben im Bildschirm bei momentaner Cursor-Position und erhöht sie danach um eins (ADVANCE). Funktioniert nur bei 40 Z/Z.

vorher: LDA 0-255

SETINV (JSR $FE80): Inverse Bildschirmdarstellung.

SETNORM (JSR $FE84): Normale Bildschirmdarstellung.

FLASH muß man simulieren mit

```
LDA  #$7F
STA  INVFLG   ($0032)
```

Man beachte, daß nicht nur die Lores-Routinen, sondern auch die obigen Bildschirm-Routinen nicht über COUT gehen, d.h. Peripheriegeräte wie Drukker, Diskettenlaufwerk usw. unberührt lassen.

2.1.2.3. Ausgabe- und Eingabe-Routinen

SETKBD (JSR $FE89): Entspricht IN#0, d.h. KSWL wird auf $FD1B (= KEYIN) eingestellt.

INPORT ($FE8B): KSWL auf Input-Slot einstellen; IN#A

vorher: LDA 0-7 (= Slot 0-7, wobei 0 SETKBD entspricht)

SETVID (JSR $FE93): Entspricht PR#0, d.h. CSWL wird auf $FDF0 (= COUT1) eingestellt. Funktioniert bei keiner 80-Zeichenkarte!

OUTPORT ($FE95): CSWL auf Output-Slot einstellen; PR#A

vorher: LDA 0-7 (= Slot 0-7, wobei 0 SETVID entspricht)

COUT ($FDED): Standard-ASCII-Ausgabe über den CSWL-Vektor (40-Z/Z-Bildschirm, Drucker, 80-Z/Z-Karte usw.). COUT (= „character output") rettet alle drei Register X, Y und A! COUT heißt in unseren Assemblerlistings oft PRINT.

vorher: LDA Buchstabe (in der Regel $80-$FF; bisweilen bei bestimmten Interface-Karten auch $00-$7F).

COUTZ ($FDF6): Wie COUT, doch wird das INVFLG ignoriert.

CROUT (JSR $FD8E): Gibt $8D = Return über COUT aus.

CROUT1 (JSR $FD8B): Erst CLREOL, dann CROUT ausführen.

PRERR (JSR $FF2D): Das Wort „ERR" und dann Piepston ausgeben.

BELL (JSR $FF3A): Piepston ausgeben.

BELL1A (JSR $FBDD): Piepston ausgeben.

BELL2 ($FBE4): Modifizierter Piepston.

vorher: LDY 0-255 (0 = leise, 255 = schrill)

WAIT ($FCA8): Warteschleife (modifiziert nur A-Register). Vergleiche WAIT-Beispiel in Kap. 2.2.5.

vorher: LDA 0-255

PRBLNK (JSR $F948): 3 Leertasten $A0 über COUT ausgeben.

PRBLK2 ($F94A): 1-255 Leertasten über COUT ausgeben.

vorher: LDX 1-255

PRBYTE ($FDDA): Standard-Hexbyte-Ausgabe über den CSWL-Vektor. Läßt X und Y unverändert, während A modifiziert wird. PRBYTE wird von uns oft HEXOUT genannt.

vorher: LDA Hexzahl ($00-$FF)

PRHEX ($FDE3): Ausgabe des Low Nibble als einzelne Hexziffer über PRBYTE.

vorher: LDA #$0L (d.h. High Nibble = $0, Low Nibble beliebig).

Wenn man das High Nibble nicht auf 0 setzt, wird es automatisch durch PRHEX eliminiert.

PRNTYX ($F940): Y und X (in dieser Reihenfolge) als Hexzahlen ausgeben.

vorher: LDY Hexzahl ($00-$FF)
LDX Hexzahl

PRNTAX ($F941): A und X (in dieser Reihenfolge) als Hexzahlen ausgeben.

vorher: LDA Hexzahl ($00-$FF)
LDX Hexzahl

RDKEY (JSR $FD0C): Standard-Eingabe-Routine über KSWL. Zeigt erst einen blinkenden Cursor an, bevor RDKEY1 erfolgt.

nachher: Tastenwert, d.h. gedrückte Taste (oder bei Interface-Input entsprechender Input-Wert) in A. Bei Taste stets mit Bit 7 on, d.h. im Bereich $80-$FF.

RDKEY1 (JSR $FD18): Standard-Eingabe-Routine über KSWL ohne vorherigen Cursor. Erstens für Interface-Input und zweitens für IIe empfohlen, da letzterer ohnehin bei Tastatureingabe über nachfolgendes KEYIN den blinkenden Cursor durch seinen Kästchen-Cursor ersetzt.

nachher: (Tasten)wert in A wie bei RDKEY

KEYIN (JSR $FD1B): Reine Tastatureingabe, also nicht für Interface-Input. Funktioniert nicht bei IIe-80-Zeichenkarte!

nachher: Taste ($80-$FF) in A

KEYIN ohne Cursor muß man selbst simulieren

```
KEYWAIT   LDA   $C000      ;Tastenregister abfragen (KEY-
                           BOARD)
          BPL   KEYWAIT    ;noch keine Taste gedrückt
          BIT   $C010      ;Tastenregister freimachen (KEY-
                           BOARD STROBE)
          RTS
```

nachher: Taste in A ($80-$FF)

RDCHAR (JSR $FD35): Wie KEYIN, aber danach, falls ESC gedrückt wurde, entsprechende Cursorbewegung ausführen (nicht zu empfehlen). Beim Apple IIe befindet man sich bei 80 Z/Z infolge mieser ROM-Programmierung durch „Our Hero Rick A" (zitiert nach „Monitor ROM Listing For Apple IIe", S.12, Zeile 192) sozusagen ständig im RDCHAR-Modus mit der unerquicklichen Folge, daß der Anwender mit ESC Ctrl-L den IIe zum „Absaufen" – einer der zahlreichen Bugs – bringen kann.

GETLN (JSR $FD6A): Standard-Input über KSWL von bis zu 255 ASCII-Zeichen und Return. Gibt vorher Prompt ($33) aus. Meist für Tastatur-Input benutzt, jedoch auch bei DOS 3.3 – nicht jedoch bei ProDOS – für Textfile-Input verwendbar.

vorher: (nur bei Bedarf) LDA Prompt-Zeichen ($A0-$FF)
STA PROMPT ($0033)

nachher: Eingegebene Zeichen mit Bit 7 on im Eingabepuffer ab $0200.
X-Register (= Inputlänge) zeigt auf $8D = Return, z.B.:

```
0200: C1 C2 C3 8D          X = 3 = Länge (ohne Return)
      A  B  C
```

Der Applesoft-Input gestattet gegenüber GETLN nur die Eingabe von maximal 239 Zeichen. Ferner wird Bit 7 bei allen eingegebenen Zeichen auf 0 gesetzt.

GETLNZ (JSR $FD67): Wie GETLN, doch wird vor dem Prompt noch ein Return ausgegeben.

nachher: Eingabezeile ab $0200 mit X = Länge

GETLN1 (JSR $FD6F): Wie GETLN/GETLNZ, doch wird vorher weder Return noch Prompt ausgegeben.

nachher: Eingabezeile ab $0200 mit X = Länge

GETNUM ($FFA7): Diese Monitor-GETNUM-Routine – nicht zu verwechseln mit der Applesoft-GETNUM-Routine – verwandelt einen 2-4stelligen Hexzahl-String in eine 1-2-Byte-Hexzahl in A2L-A2H.

vorher: 2 oder 4 Hexziffern mit Bit 7 on in Eingabepuffer ab $0200, abgeschlossen durch Endmarker $00.

```
LDY  #0        (Y auf 0 setzen!)
JSR  GETNUM
```

nachher: LL Hexzahl in A2L ($003E)
HH Hexzahl in A2H ($003F); bei 1-Byte-Hexzahl auf 0 gesetzt

Vergleiche hierzu das Programm DEZHEX in Kap. 4.2.2.

PREAD ($FB1E): Paddle Read; liest eines der 4 Paddle-Drehregler.

vorher: LDX Paddle-Nummer (0-3)
JSR PREAD
nachher: Y enthält Analogwert (0-255)

2.1.2.4. Simulierte Monitorbefehle

INSTDSP ($F8D0): Instruction Display (über COUT), d.h. Disassemblierung einer Zeile (= eines 1-3 Bytes langen Befehls). Entspricht Monitorbefehl L für 1 statt 20 Zeilen. Vergleiche hierzu Programm DISASSEMBLER in Kap. 2.2.10.

vorher: LL der Adresse in PCL ($003A)
HH der Adresse in PCH ($003B)
nachher: Länge des Befehls (0-2) in $002F (= LENGTH)

Hierzu wie auch bei den nachfolgenden Befehlen ist ein kleines Programm erforderlich, da die Werte nicht vom Monitor aus gepokt werden können. Beispiel:

```
LDA  #$00      ;Adresse $1000
STA  $3A
LDA  #$10
STA  $3B
JSR  $F8D0
RTS
```

LIST ($FE5E): Entspricht Monitorbefehl L und disassembliert 20 Zeilen.

vorher: LL der Startadresse in PCL ($003A)
HH der Startadresse in PCH ($003B)
LDX #0 (X auf 0 setzen!)

LIST2 ($FE63): Wie LIST, jedoch werden 1-255 Zeilen disassembliert.

vorher: LL der Startadresse in PCL ($003A)
HH der Startadresse in PCH ($003B)
LDA Zeilenanzahl (1-255)

REGDSP (JSR $FAD7): Register Display, d.h. Registeranzeige über COUT in der Reihenfolge A, X, Y, P, S. Vor der Anzeige wird ein Return ausgegeben. Entspricht Monitor-Ctrl-E-Befehl.

RGDSP1 (JSR $FADA): Wie REGDSP, doch vor Anzeige kein Return.

XAM ($FDB3): Hexdump eines Speicherbereichs (Examine). Entspricht Monitor-Befehl SSSS.EEEE, z.B. 1000.2000. Vergleiche hierzu Programm HEXDUMP in Kap. 2.2.9, das jedoch die XAM-Routine nicht benutzt, um 16 statt 8 Hexzahlen pro Zeile ausdrucken zu können.

vorher:	LL	Startadresse	in A1L	($003C)
	HH	Startadresse	in A1H	($003D)
	LL	Endadresse	in A2L	($003E)
	HH	Endadresse	in A2H	($003F)

MOVE ($FE2C): Entspricht Monitor-Move-Befehl, z.B. 1000<2000.2FFFM.
VFY ($FE36): Entspricht Monitor-Verify-Befehl, z.B. 1000<2000.2FFFV.

MOVE und VFY setzen dieselbe Parameter-Initialisierung voraus.

vorher:	LL	Startadresse	in A1L	($003C)
	HH	Startadresse	in A1H	($003D)
	LL	Endadresse	in A2L	($003E)
	HH	Endadresse	in A2H	($003F)
	LL	Zieladresse	in A4L	($0042)
	HH	Zieladresse	in A4H	($0043)
	LDY	#0 (muß auf 0 gesetzt werden!)		
	JSR	MOVE oder JSR VFY		

2.1.2.5. Reset und andere Interrupts

Die nachfolgenden Monitoradressen sind in der Regel nicht als Routinen gedacht, die mit JSR angesprungen werden können.

NMI-Vektor: $FFFA: FB 03, d.h. $03FB (Nicht maskierbarer Interrupt). Könnte mit JMP ($FFFA) angesprungen werden.

RESET-Vektor: $FFFC: 62 FA, d.h. $FA62 (Apple II Plus/IIe Software-Reset). Könnte mit JMP ($FFFC) angesprungen werden.

IRQ-Vektor: $FFFE: 40 FA, d.h. $FA40 (Interrupt Request). Nicht anspringen!

OLDRST: $FF59: Initialisiert Textmodus und CSWL/KSWL (PR#0, IN#0) und springt dann in den Monitor. Könnte mit JMP $FF59 angesprungen werden. (Apple II Software-Reset).

IRQ: $FA40: Prüft, ob Break-Flag (= Bit 4 des P-Registers) gesetzt ist. Wenn ja, Sprung zu BREAK, wenn nein, JMP ($03FE), d.h. zum Interrupt-Request-Handler (Hardware-Uhr usw.). Nicht anspringen!

BREAK: $FA4C: Rettet die Register in $0045-$0049, holt Break-Adresse vom Stack und springt indirekt mit JMP ($03F0) zum Page-3-Break-Vektor. Man beachte, daß die Break-Adresse um 2 hätte vermindert werden müssen (siehe Kap. 1.5). Nicht anspringen!

OLDBRK: $FA59: Zeigt Break-Adresse + 2 sowie Register an und springt dann in den Monitor $FF65 (MON). Nicht anspringen!

RESET: $FA62: Initialisiert Textmodus und CSWL/KSWL (PR#0, IN#0) und prüft dann, ob „Funny Complement" stimmt. Wenn ja, mit JMP ($03F2) indirekter Sprung zum Page-3-Reset-Vektor. Wenn nein, Sprung zu PWRUP. Könnte mit JMP $FA62 angesprungen werden.

NEWMON: $FA81: Entspricht Reset, doch wird weder Textmodus noch CSWL/KSWL initialisiert. Könnte mit JMP $FA81 angesprungen werden.

PWRUP: $FAA6: Initialisiert Page-3-Vektoren und „startet voll durch", d.h. es wird vom Slot mit der höchsten Slot-Nummer, die in MSLOT = $07FB abgelegt wird, neu gebootet. Existiert kein DOS-Controller, so erfolgt Sprung zu $E000 (Applesoft-Kaltstart). Kann mit JMP $FAA6 angesprungen werden.

SETPWRC: $FB6F: Bewirkt EOR von $03F3 (= HH des Page-3-Reset-Vektors) mit Hexwert $A5 (= „Funny Complement"). Kann mit JSR angesprungen werden.

Angenommen, man wollte den Page-3-Reset-Vektor auf Applesoft-RUN = $D566 einstellen, damit nach Ctrl-Reset das Applesoft-Programm neu gestartet wird:

```
LDA   #$66      ;$D566
STA   $03F2
LDA   #$D5
STA   $03F3
JSR   $FB6F     ;SETPWRC
RTS
```

VERSION: $FBB3: Diese Speicherstelle enthält $06 bei Apple IIe (und bei IIc, obgleich anderer F8-Monitor!), $EA bei Apple II Plus und $38 bei Apple II:

```
LDA   $FBB3
CMP   #$06
BEQ   APPLEIIE
CMP   #$EA
BEQ   APPLEIIPLUS
CMP   #$38
BEQ   APPLEII
BNE   BOSKOP oder GOLDPARMÄNE usw.
```

TITLE: $FB09: Enthält Wort „Apple ÜÄ" beim Apple IIe sowie „APPLE ÜÄ" beim Apple II Plus, während der Apple II hier alte Monitor-Routinen hat.

```
LDA   $FB0A     ;1 Byte nach $FB09
CMP   #$F0      ;„p"
BEQ   APPLEIIE
CMP   #$D0      ;„P"
BEQ   APPLEIIPLUS
CMP   #$A2
BEQ   APPLEII
BNE   EPROM-Schwarzbrenner
```

2.1.3. INTCXROM-Routinen des Apple IIe

Die INTCXROM-Routinen sind von einem unqualifizierten Programmierer der Firma Apple produziert worden, der außer seinem Namen wenig Rühmliches im

ROM verewigt hat. C. Bongers hat dies in „CALL A.P.P.L.E", Heft 5/1984 unmißverständlich zum Ausdruck gebracht.

Die 80-Zeichenroutinen sind aus verschiedenen Gründen unzureichend:
1. Sie sind ca. 40% langsamer, als sie sein könnten. Vergleiche hierzu die PRINT80-Routine in Kap. 5.1.4.
2. Sie funktionieren nicht, wenn gleichzeitig ein Drucker aktiv ist.
3. Sie enthalten mehrere Bugs (z.B. ESC-Ctrl-L-Bug: $C92A müßte $10 enthalten. Inverse-Bug: nach NORMAL wird nicht immer auf NORMAL umgeschaltet. Ctrl-L-Bug: führt zu SYNTAX ERROR. „Poke 33, CH"-Bug u.a.)

Die INTCXROM-Routinen nehmen den Bereich $C100-$CFFF ein, wobei sich ab $C300 die eigentliche 80-Z/Z-Routine mit Input-Vektor bei $C305 und Output-Vektor bei $C307 sowie ab $C401-$C7FF die Hardware-Testroutine befindet. Der F8-Monitor springt stets bei $C100 auf das INTCXROM, dessen Routinen auch dann präsent sind, wenn eine 80-Zeichenkarte fehlt. Wegen der genannten Bugs erwähnen wir hier nur einige wenige Routinen.

INTCXROM.AKTIV: Wer keine 80-Zeichenkarte besitzt, könnte trotzdem die INTCXROM-Software aktivieren mit

```
STA   $C00A     ;Enable INTC3ROM (nicht STA $C007!)
JSR   $C300     ;danach ALTCHAR, d.h. alternativer Zeichensatz
```

MOVE ($C311 oder $C363): Verschiebt einen (Teil)bereich der „unteren" $0200-$BFFF in die „oberen" $0200-$BFFF und umgekehrt. Setzt 64K-Karte voraus.

```
vorher:   LL   Startadresse                in A1L   ($003C)
          HH   Startadresse                in A1H   ($003D)
          LL   Endadresse                  in A2L   ($003E)
          HH   Endadresse                  in A2H   ($003F)
          LL   Zieladresse                 in A4L   ($0042)
          HH   Zieladresse                 in A4H   ($0043)
          SEC  (für von „unten" nach „oben")
          CLC  (für von „oben" nach „unten")
          STA  $C007 (SETINTCXROM)
          JSR  $C311 (oder $C363)
nachher:  STA  $C006 (SETSLOTCXROM)
```

XFER ($C314 oder $C3B0): Sprung auf Karte. Ohne entsprechende Utilities wie z.B. meine MMU-Programme nutzlos.

COPYROM ($CF78): Kopiert F8-ROM in LC $F800-$FFFF. Modifiziert CSWH!

```
vorher:   LDA   $37        ;CSWH
          PHA
          STA   $C007
          JSR   $CF78
nachher:  STA   $C006
          PLA
          STA   $37
```

TESTCARD ($CB24): Prüft, ob 80-Z/Z (nicht ob Aux. 64K) vorhanden ist.

```
vorher:   STA   $C007
          JSR   $CB24
nachher:  STA   $C006
          BEQ   KARTEDA
          BNE   KARTEFEHLT
```

QUID ($CDAA): Schaltet 80 Z/Z ab. Hängt DOS ab!

```
vorher:   STA   $C007
          JSR   $CDAA
          STA   $C006
          JSR   $03EA      ;DOS 3.3 Connect oder
          JSR   $9A17      ;ProDOS Connect
```

```
                        1           ORG  $300
                        2   *
                        3   * HEXOUT
                        4   * ======
                        5   *
                        6   * Dieses Demo zeigt Hexzahl
                        7   * in Htab H, Vtab V am Bild-
                        8   * schirm an (40 Z/Z)
                        9   * Es verwendet die im Monitor
                       10   * enthaltenen Routinen VTAB,
                       11   * PRBYTE und BASCALC
                       12   *
0300: 60                13   HEXBYTE  HEX  60          ;0-255
0301: 0A                14   VTABPOS  HEX  0A          ;1-24
0302: 0F                15   HTABPOS  HEX  0F          ;1-40
                       16   *
                       17   * Cursor horizontal und vertikal
                       18   *
                       19   CH       EQU  $24
                       20   CV       EQU  $25
                       21   *
                       22   * Basis-Adresse der Stelle von
                       23   * Vtab V (1-24), Htab 1, d.h.
                       24   * Beginn der jeweiligen Zeile
                       25   *
                       26   BASL     EQU  $28
                       27   BASH     EQU  $29
                       28   *
0303: AD 01 03          29   VTABHTAB LDA  VTABPOS
0306: 20 2F 03          30            JSR  VTAB
0309: AD 02 03          31            LDA  HTABPOS
030C: 20 3C 03          32            JSR  HTAB
030F: A4 24             33            LDY  CH          ;Htab
                       34   *
0311: AD 00 03          35   HEXPRNT  LDA  HEXBYTE
0314: 20 18 03          36            JSR  HEXOUT
0317: 60                37            RTS
                       38   *
                       39   * Hexadezimale Anzeige des
                       40   * Bytes im Akkumulator.
                       41   * Erst wird linkes Nibble,
                       42   * dann rechtes Nibble als
                       43   * ASCII-Zeichen angezeigt.
                       44   *
0318: 48                45   HEXOUT   PHA              ;linkes
0319: 4A                46            LSR
031A: 4A                47            LSR
031B: 4A                48            LSR
031C: 4A                49            LSR
031D: 20 23 03          50            JSR  HEXOUT3
                       51   *
0320: 68                52            PLA              ;rechtes
0321: 29 0F             53   HEXOUT2  AND  #%00001111 ;$0F
0323: 09 B0             54   HEXOUT3  ORA  #„0"
```

```
0325: C9 BA      55             CMP  #„:"        ;nach 9
0327: 90 02      56             BCC  HEXOUT4
0329: 69 06      57             ADC  #$06
032B: 91 28      58   HEXOUT4   STA  (BASL),Y
032D: C8         59             INY              ;Htab+1
032E: 60         60             RTS
                 61   *
032F: 38         62   VTAB      SEC              ;24-1
0330: E9 01      63             SBC  #1
0332: 85 25      64             STA  CV
0334: 20 42 03   65             JSR  BASCALC
0337: 65 20      66             ADC  $20         ;linker
0339: 85 28      67             STA  BASL        ;Rand
033B: 60         68             RTS
                 69   *
033C: 38         70   HTAB      SEC              ;40-1
033D: E9 01      71             SBC  #1
033F: 85 24      72             STA  CH
0341: 60         73             RTS
                 74   *
                 75   * Diese Routine muß man mit
                 76   * Bleistift und Papier nach-
                 77   * vollziehen, wenn man sie
                 78   * verstehen will
                 79   *
0342: 48         80   BASCALC   PHA
0343: 4A         81             LSR
0344: 29 03      82             AND  #%00000011  ;$03
0346: 09 04      83             ORA  #%00000100  ;$04
0348: 85 29      84             STA  BASH
034A: 68         85             PLA
034B: 29 18      86             AND  #%00011000  ;$18
034D: 90 02      87             BCC  BSCLC2
034F: 69 7F      88             ADC  #$7F
0351: 85 28      89   BSCLC2    STA  BASL
0353: 0A         90             ASL
0354: 0A         91             ASL
0355: 05 28      92             ORA  BASL
0357: 85 28      93             STA  BASL
0359: 60         94             RTS

--End assembly-

90 bytes

Errors: 0
```

```
                    1              ORG  $0803
                    2    *
                    3    * BINAER
                    4    * ======
                    5    *
                    6    * Dieses Programm erzeugt eine
                    7    * Ascii-Hex-Binär-Dez-Tabelle
                    8    * Es liegt folgender Algorithmus
                    9    * zugrunde:
                   10    *
                   11    * 10 FOR X = 1 TO 64
                   12    * 20 FOR Y = 0 TO 3
                   13    * 30 H = Y * 64 + X - 1
                   14    * 40 PRINT H;: REM usw.
                   15    * 50 NEXT Y
                   16    * 60 PRINT
                   17    * 70 NEXT X
                   18    *
                   19    * Dadurch entsteht eine 4spaltige
                   20    * Tabelle, wobei allerdings die
                   21    * Zeile 40 (PRINT H) durch die
                   22    * Ascii-Hex-Binär-Dez-Anzeige
                   23    * ersetzt wird.
                   24    *
                   25    HEXOUT   EQU  $FDDA
                   26    LINPRT   EQU  $ED24
                   27    PRINT    EQU  $FDED
                   28    DOSWARM  EQU  $03D0
                   29    *
                   30    * FOR X = 1 TO 64
                   31    *
0803: A2 00        32    XLOOP1   LDX  #0          ;WRAP
0805: 8E 45 08     33             STX  X
                   34    *
                   35    * NEXT X
                   36    *
0808: A9 8D        37    XLOOP2   LDA  #$8D
080A: 20 ED FD     38             JSR  PRINT
                   39    *
080D: EE 45 08     40             INC  X           ;X=X+1
0810: AE 45 08     41             LDX  X
0813: E0 41        42             CPX  #65
0815: 90 03        43             BCC  YLOOP1
0817: 4C D0 03     44             JMP  DOSWARM     ;EXIT
                   45    *
                   46    * FOR Y = 0 TO 3
                   47    *
081A: A0 FF        48    YLOOP1   LDY  #$FF        ;WRAP
081C: 8C 46 08     49             STY  Y
                   50    *
                   51    * NEXT Y
                   52    *
081F: EE 46 08     53    YLOOP2   INC  Y           ;Y=Y+1
```

```
0822: AC 46 08  54            LDY  Y
0825: C0 04     55            CPY  #4
0827: B0 DF     56            BCS  XL00P2
                57  *
                58  * HEX = Y * 64 + X - 1
                59  *
0829: C8        60            INY
082A: A9 00     61            LDA  #0          ;Y*64
082C: 88        62  YL00P3    DEY
082D: F0 06     63            BEQ  YL00P4
082F: 18        64            CLC
0830: 69 40     65            ADC  #64
0832: 4C 2C 08  66            JMP  YL00P3
0835: 18        67  YL00P4    CLC
0836: 6D 45 08  68            ADC  X           ;+X
0839: 38        69            SEC
083A: E9 01     70            SBC  #1          ;-1
083C: 8D 47 08  71            STA  HEX
                72  *
083F: 20 48 08  73            JSR  BINAER
0842: 4C 1F 08  74            JMP  YL00P2
                75  *
0845: 00        76  X         HEX  00
0846: 00        77  Y         HEX  00
0847: 00        78  HEX       HEX  00
                79  *
0848: AD 47 08  80  BINAER    LDA  HEX
084B: C9 40     81            CMP  #64
084D: 90 0F     82            BCC  ASCPRINT
084F: A9 A0     83  ABSTAND   LDA  #$A0
0851: 20 ED FD  84            JSR  PRINT
0854: A9 A1     85            LDA  #„!"
0856: 20 ED FD  86            JSR  PRINT
0859: A9 A0     87            LDA  #$A0
085B: 20 ED FD  88            JSR  PRINT
                89  *
                90  * ASCII
                91  *
085E: AD 47 08  92  ASCPRINT LDA  HEX
0861: 09 80     93           ORA  #%10000000
0863: C9 A0     94           CMP  #$A0
0865: B0 03     95           BCS  ASCII3
                96  *
                97  * Ctrl in Nicht-Ctrl sowie
                98  * Del in Space verwandeln
                99  *
0867: 18        100 ASCII2   CLC
0868: 69 40     101          ADC  #$40
086A: C9 FF     102 ASCII3   CMP  #$FF
086C: D0 02     103          BNE  ASCII4
086E: A9 A0     104          LDA  #$A0        ;DEL
0870: 20 ED FD  105 ASCII4   JSR  PRINT
0873: AD DD 08  106          LDA  SPACE
0876: 20 ED FD  107          JSR  PRINT
```

```
                    108  *
                    109  * HEXADECIMAL
                    110  *
0879: AD 47 08      111  HEXPRINT LDA  HEX
                    112  *
                    113  * HEXOUT zeigt A als Hexzahl an
                    114  *
087C: 20 DA FD      115           JSR  HEXOUT
087F: AD DD 08      116           LDA  SPACE
0882: 20 ED FD      117           JSR  PRINT
                    118  *
                    119  * BINÄR
                    120  *
0885: AD 47 08      121  BINPRINT LDA  HEX
0888: A8            122           TAY
0889: A2 07         123           LDX  #7
088B: 98            124  LOOP     TYA
088C: 2A            125           ROL
088D: 90 08         126           BCC  ZERO
088F: A9 B1         127  ONE      LDA  #$B1
0891: 9D D4 08      128           STA  STRING,X
0894: 4C 9C 08      129           JMP  DECREASE
0897: A9 B0         130  ZERO     LDA  #$B0
0899: 9D D4 08      131           STA  STRING,X
089C: 98            132  DECREASE TYA
089D: 2A            133           ROL
089E: A8            134           TAY
089F: CA            135           DEX
08A0: 10 E9         136           BPL  LOOP
08A2: A2 07         137           LDX  #7
08A4: BD D4 08      138  STRPRT   LDA  STRING,X
08A7: 20 ED FD      139           JSR  PRINT
08AA: CA            140           DEX
08AB: 10 F7         141           BPL  STRPRT
08AD: AD DD 08      142           LDA  SPACE
08B0: 20 ED FD      143           JSR  PRINT
                    144  *
                    145  * DECIMAL
                    146  *
08B3: AD 47 08      147  DECPRINT LDA  HEX
08B6: C9 64         148           CMP  #100
08B8: B0 11         149           BCS  DEC1
08BA: A9 B0         150           LDA  #„0"
08BC: 20 ED FD      151           JSR  PRINT
08BF: AD 47 08      152           LDA  HEX
08C2: C9 0A         153           CMP  #10
08C4: B0 05         154           BCS  DEC1
08C6: A9 B0         155           LDA  #„0"
08C8: 20 ED FD      156           JSR  PRINT
                    157  *
                    158  * LINPRT zeigt X=LL A=HH
                    159  * als Dezimalzahl an.
                    160  *
08CB: A9 00         161  DEC1     LDA  #0          ;HIGH
```

```
08CD: AE 47 08  162           LDX  HEX        ;LOW
08D0: 20 24 ED  163           JSR  LINPRT
08D3: 60        164           RTS
08D4: B0 B0 B0  165  STRING   ASC  „00001111"
08D7: B0 B1 B1 B1 B1
08DC: 00        166           HEX  00
08DD: A0        167  SPACE    HEX  A0


--End assembly-

219 bytes

Errors: 0
```

2.2.3. BACKMOVE

```
                1             ORG  $300
                2   *
                3   * BACKMOVE
                4   * ========
                5   *
                6   * Verschiebe Startbereich-Ende
                7   * bis Startbereich-Anfang nach
                8   * Zielbereich-Ende, z.B.
                9   * $5FFF-$4000 nach $3FFF(-$2000)
               10   *
               11   SB_END   EQU  $FA        ;LLHH
               12   SB_ANF   EQU  $FC        ;LLHH
               13   ZB_END   EQU  $FE        ;LLHH
               14   *
0300: A0 00    15             LDY  #0
0302: B1 FA    16   MOVE1    LDA  (SB_END),Y
0304: 91 FE    17             STA  (ZB_END),Y
0306: A5 FE    18             LDA  ZB_END
0308: D0 02    19             BNE  MOVE2
030A: C6 FF    20             DEC  ZB_END+1
030C: C6 FE    21   MOVE2    DEC  ZB_END
030E: A5 FC    22             LDA  SB_ANF
0310: C5 FA    23             CMP  SB_END
0312: A5 FD    24             LDA  SB_ANF+1
0314: E5 FB    25             SBC  SB_END+1
0316: A5 FA    26             LDA  SB_END
0318: D0 02    27             BNE  MOVE3
031A: C6 FB    28             DEC  SB_END+1
031C: C6 FA    29   MOVE3    DEC  SB_END
031E: 90 E2    30             BCC  MOVE1
0320: 60       31             RTS
```

```
                        1               ORG  $0300
                        2   *
                        3   * ADD.SUB
                        4   * ═══════
                        5   *
                        6   * Addition 1-3 Stellen
                        7   * ═══════════
                        8   *
                        9   *   HHMMLL Summand1
                       10   * + HHMMLL Summand2
                       11   * ----------------
                       12   * = HHMMLL Summe
                       13   *
0300: 4C 11 03         14   SUM1     JMP  SUM2
                       15   *
                       16   * Summand1
                       17   *
0303: 00               18   SUMD1L   HEX  00          ;LL
0304: 00               19   SUMD1M   HEX  00          ;MM
0305: 00               20   SUMD1H   HEX  00          ;HH
                       21   *
                       22   * Summand2
                       23   *
0306: 00               24   SUMD2L   HEX  00          ;LL
0307: 00               25   SUMD2M   HEX  00          ;MM
0308: 00               26   SUMD2H   HEX  00          ;HH
                       27   *
                       28   * Summe
                       29   *
0309: 00               30   SUMMEL   HEX  00          ;LL
030A: 00               31   SUMMEM   HEX  00          ;MM
030B: 00               32   SUMMEH   HEX  00          ;HH
                       33   *
                       34   * Flag: 0 = okay; 1 = Überlauf
                       35   *
030C: 00               36   SUMFLAG  HEX  00          ;0=okay
                       37   *
                       38   * Anzahl der Stellen 1-3
                       39   *
030D: 03               40   SUMSTELL HEX  03          ;1-3
                       41   *
030E: 00               42   SUMAREG  HEX  00          ;A
030F: 00               43   SUMXREG  HEX  00          ;X
0310: 00               44   SUMYREG  HEX  00          ;Y
                       45   *
0311: 8D 0E 03         46   SUM2     STA  SUMAREG
0314: 8E 0F 03         47            STX  SUMXREG
0317: 8C 10 03         48            STY  SUMYREG
031A: A2 00            49            LDX  #0
031C: 8E 0B 03         50            STX  SUMMEH
031F: 8E 0A 03         51            STX  SUMMEM
0322: 8E 0C 03         52            STX  SUMFLAG
0325: AC 0D 03         53            LDY  SUMSTELL
                       55   *
0328: 18               56            CLC             ;Add.
0329: BD 03 03         57   SUM3     LDA  SUMD1L,X
```

```
032C: 7D 06 03  58              ADC  SUMD2L,X
032F: 9D 09 03  59              STA  SUMMEL,X
0332: E8        60              INX
0333: 88        61              DEY
0334: D0 F3     62              BNE  SUM3
                63   *
0336: 90 03     64              BCC  SUM4       ;okay
0338: EE 0C 03  65              INC  SUMFLAG    ;BCS!
                66   *
033B: AE 0F 03  67   SUM4        LDX  SUMXREG
033E: AC 10 03  68              LDY  SUMYREG
0341: AD 0E 03  69              LDA  SUMAREG
0344: 60        70              RTS
                71   *
                72   * Subtraktion 1-3 Stellen
                73   * ======================
                74   *
                75   *   HHMMLL Minuend
                76   * - HHMMLL Subtrahend
                77   * ------------------
                78   * = HHMMLL Differenz
                79   *
0345: 4C 56 03  80   SUB1        JMP  SUB2
                81   *
                82   * Minuend
                83   *
0348: 00        84   MINUENDL HEX  00          ;LL
0349: 00        85   MINUENDM HEX  00          ;MM
034A: 00        86   MINUENDH HEX  00          ;HH
                87   *
                88   * Subtrahend
                89   *
034B: 00        90   SUBTRAHL HEX  00          ;LL
034C: 00        91   SUBTRAHM HEX  00          ;MM
034D: 00        92   SUBTRAHH HEX  00          ;HH
                93   *
                94   * Differenz
                95   *
034E: 00        96   DIFFERL  HEX  00          ;LL
034F: 00        97   DIFFERM  HEX  00          ;MM
0350: 00        98   DIFFERH  HEX  00          ;HH
                99   *
                100  * Flag: 0 = okay; 1 = Überlauf
                101  *
0351: 00        102  SUBFLAG  HEX  00          ;0=okay
                103  *
                104  * Anzahl der Stellen 1-3
                105  *
0352: 03        106  SUBSTELL HEX  03          ;1-3
                107  *
0353: 00        109  SUBAREG  HEX  00          ;A
0354: 00        110  SUBXREG  HEX  00          ;X
0355: 00        111  SUBYREG  HEX  00          ;Y
                112  *
```

```
0356: 8D 53 03  113  SUB2     STA  SUBAREG
0359: 8E 54 03  114           STX  SUBXREG
035C: 8C 55 03  115           STY  SUBYREG
035F: A2 00     116           LDX  #0
0361: 8E 50 03  117           STX  DIFFERH
0364: 8E 4F 03  118           STX  DIFFERM
0367: 8E 51 03  119           STX  SUBFLAG
036A: AC 52 03  120           LDY  SUBSTELL
                121  *
036D: 38        122           SEC              ;Subt.
036E: BD 48 03  123  SUB3     LDA  MINUENDL,X
0371: FD 4B 03  124           SBC  SUBTRAHL,X
0374: 9D 4E 03  125           STA  DIFFERL,X
0377: E8        126           INX
0378: 88        127           DEY
0379: D0 F3     128           BNE  SUB3
                129  *
037B: B0 03     130           BCS  SUB4        ;okay
037D: EE 51 03  131           INC  SUBFLAG     ;BCC!
                132  *
0380: AE 54 03  133  SUB4     LDX  SUBXREG
0383: AC 10 03  134           LDY  SUMYREG
0386: AD 0E 03  135           LDA  SUMAREG
0389: 60        136           RTS
                137  *

--End assembly-

138 bytes

Errors: 0
```

2.2.5. WAIT

```
                1             ORG  $300
                2    *
                3    * WAIT
                4    * ====
                5    *
                6    WAIT     EQU  $FCA8
                7    *
                8    * A =   4  ca. 0.000109s
                9    * A =  18  ca. 0.001090s
                10   * A =  60  ca. 0.010048s
                11   * A = 196  ca. 0.100969s
                12   *
                13   * Wait-Beispiel für 1 MHz 6502
                14   * 200 * 0.1s = ca. 20 Sekunden
                15   *
0300: A2 C8     16            LDX  #200
0302: A9 C4     17   WAITER   LDA  #196        ;0,1s
0304: 20 A8 FC  18            JSR  WAIT
0307: CA        19            DEX
0308: D0 F8     20            BNE  WAITER
030A: 60        21            RTS
```

```
                        1           ORG  $300
                        2   *
                        3   * MULTIPLIKATION
                        4   * ============
                        5   *
                        6   * Positiver 2-Byte-Multiplikand
                        7   * (= Faktor1) mal positiven
                        8   * 2-Byte-Multiplikator (= Faktor2)
                        9   * ergibt positives 4-Byte-Produkt
                       10   *
0300: 4C 0E 03         11   MULT0    JMP  MULT1
                       12   *
                       13   * Vorher poken: Faktor1 + Faktor2
                       14   *
0303: 00               15   FAKTOR1L HEX  00          ;LOW
0304: 00               16   FAKTOR1H HEX  00          ;HIGH
                       17   *
0305: 00               18   FAKTOR2L HEX  00          ;LOW
0306: 00               19   FAKTOR2H HEX  00          ;HIGH
                       20   *
                       21   * Nachher peeken: Produkt
                       22   *
0307: 00               23   PRODUKT1 HEX  00          ;LOW
0308: 00               24   PRODUKT2 HEX  00          ;MIDA
0309: 00               25   PRODUKT3 HEX  00          ;MIDB
030A: 00               26   PRODUKT4 HEX  00          ;HIGH
                       27   *
030B: 00               28   MULTREGA HEX  00
030C: 00               29   MULTREGX HEX  00
030D: 00               30   MULTREGY HEX  00
                       31   *
030E: 8D 0B 03         32   MULT1    STA  MULTREGA
0311: 8E 0C 03         33            STX  MULTREGX
0314: 8C 0D 03         34            STY  MULTREGY
                       35   *
0317: A9 00            36            LDA  #0          ;NULL?
0319: A2 03            37            LDX  #3
031B: 9D 07 03         38   MULT2    STA  PRODUKT1,X
031E: CA               39            DEX
031F: 10 FA            40            BPL  MULT2
0321: 18               41            CLC
0322: AD 03 03         42            LDA  FAKTOR1L
0325: 6D 04 03         43            ADC  FAKTOR1H
0328: F0 41            44            BEQ  MULT5
032A: 18               45            CLC
032B: AD 05 03         46            LDA  FAKTOR2L
032E: 6D 06 03         47            ADC  FAKTOR2H
0331: F0 38            48            BEQ  MULT5
                       49   *
0333: AD 05 03         50            LDA  FAKTOR2L
0336: 8D 07 03         51            STA  PRODUKT1
0339: AD 06 03         52            LDA  FAKTOR2H
033C: 8D 08 03         53            STA  PRODUKT2
```

```
                    54   *
033F: 4E 08 03      55        LSR  PRODUKT2
0342: 6E 07 03      56        ROR  PRODUKT1
0345: AD 0A 03      57        LDA  PRODUKT4
0348: A2 10         58        LDX  #16          :16 BITS
034A: 90 0F         59   MULT3  BCC  MULT4
034C: A8            60        TAY
034D: 18            61        CLC
034E: AD 09 03      62        LDA  PRODUKT3
0351: 6D 03 03      63        ADC  FAKTOR1L
0354: 8D 09 03      64        STA  PRODUKT3
0357: 98            65        TYA
0358: 6D 04 03      66        ADC  FAKTOR1H
035B: 6A            67   MULT4  ROR
035C: 6E 09 03      68        ROR  PRODUKT3
035F: 6E 08 03      69        ROR  PRODUKT2
0362: 6E 07 03      70        ROR  PRODUKT1
0365: CA            71        DEX
0366: D0 E2         72        BNE  MULT3
0368: 8D 0A 03      73        STA  PRODUKT4
                    74   *
036B: AD 0B 03      75   MULT5  LDA  MULTREGA
036E: AE 0C 03      76        LDX  MULTREGX
0371: AC 0D 03      77        LDY  MULTREGY
0374: 60            78        RTS

--End assembly-

117 bytes

Errors: 0
```

```
100  HOME
110  PRINT  CHR$ (4)„BLOAD MULTIPLIKATION"
120  PRINT „MULTIPLIKATION.DEMO"
130  PRINT „═══════════════════"
140  PRINT : PRINT „FAKTOR:  $00000000 - $0000FFFF"
150  PRINT „           0         -  65535"
160  PRINT : PRINT „PRODUKT: $00000000 - $FFFFFFFF"
170  PRINT „           0         -  4.311.810.304"
180  PRINT „                        (CA. 43118103E+09)": PRINT
190  HTAB 1: VTAB 11: CALL  - 958: INPUT „FAKTOR1:    ";F1:
     F1 =  INT (F1)
200  INPUT „FAKTOR2:    ";F2:F2 =  INT (F2)
210 H =  INT (F1 / 256):L = F1 - H * 256: POKE 771,L: POKE 772,H
220 H =  INT (F2 / 256):L = F2 - H * 256: POKE 773,L: POKE 774,H
230  CALL 768
240  PRINT „APPLESOFT: ";F1 * F2
250 L =  PEEK (775):M1 =  PEEK (776):M2 =  PEEK (777):
     H =  PEEK (778)
260  PRINT „ASSEMBLER: ";L + M1 * 256 + M2 * 65536 + 16777216
     * H
270  GET X$: GOTO 190
```

```
                        1               ORG  $300
                        2    *
                        3    * DIVISION
                        4    * ========
                        5    *
                        6    * Positiver 2-Byte-Hexzahl-Dividend
                        7    * geteilt durch positiven
                        8    * 2-Byte-Hexzahl-Divisor ergibt
                        9    * 2-Byte-Hexzahl-Quotient
                        10   * plus 2-Byte-Hexzahl-Rest
                        11   *
0300: 4C 0E 03          12   DIV0     JMP  DIV1        ;768
                        13   *
                        14   * Vorher poken: Dividend + Divisor
                        15   *
0303: 00                16   DIVDENDL HEX  00          ;771
0304: 00                17   DIVDENDH HEX  00          ;772
0305: 00                18   DIVISORL HEX  00          ;773
0306: 00                19   DIVISORH HEX  00          ;774
                        20   *
                        21   * Nachher peeken: Quotient + Rest
                        22   *
0307: 00                23   DIVQUOTL HEX  00          ;775
0308: 00                24   DIVQUOTH HEX  00          ;776
0309: 00                25   DIVRESTL HEX  00          ;777
030A: 00                26   DIVRESTH HEX  00          ;778
                        27   *
030B: 00                28   AREG     HEX  00
030C: 00                29   XREG     HEX  00
030D: 00                30   YREG     HEX  00
                        31   *
030E: 8D 0B 03          32   DIV1     STA  AREG
0311: 8E 0C 03          33            STX  XREG
0314: 8C 0D 03          34            STY  YREG
                        35   *
0317: A9 00             36            LDA  #0
0319: 8D 09 03          37            STA  DIVRESTL
031C: 8D 0A 03          38            STA  DIVRESTH
                        39   *
031F: AD 05 03          40            LDA  DIVISORL
0322: D0 0D             41            BNE  DIV2
0324: AD 06 03          42            LDA  DIVISORH
0327: D0 08             43            BNE  DIV2
                        44   *
                        45   * Division durch Null verboten
                        46   *
0329: 8D 07 03          47            STA  DIVQUOTL
032C: 8D 08 03          48            STA  DIVQUOTH
032F: F0 36             49            BEQ  DIV5
                        50   *
0331: AD 03 03          51   DIV2     LDA  DIVDENDL
0334: 8D 07 03          52            STA  DIVQUOTL
0337: AD 04 03          53            LDA  DIVDENDH
```

```
033A: 8D 08 03  54              STA  DIVQUOTH
                55   *
033D: A0 10     56              LDY  #$10       ;16BITS
033F: 0E 07 03  57   DIV3       ASL  DIVQUOTL
0342: 2E 08 03  58              ROL  DIVQUOTH
0345: 2E 09 03  59              ROL  DIVRESTL
0348: 2E 0A 03  60              ROL  DIVRESTH
034B: 38        61              SEC
034C: AD 09 03  62              LDA  DIVRESTL
034F: ED 05 03  63              SBC  DIVISORL
0352: AA        64              TAX
0353: AD 0A 03  65              LDA  DIVRESTH
0356: ED 06 03  66              SBC  DIVISORH
0359: 90 09     67              BCC  DIV4
035B: 8E 09 03  68              STX  DIVRESTL
035E: 8D 0A 03  69              STA  DIVRESTH
0361: EE 07 03  70              INC  DIVQUOTL
                71   *
0364: 88        72   DIV4       DEY
0365: D0 D8     73              BNE  DIV3
                74   *
0367: AD 0B 03  75   DIV5       LDA  AREG
036A: AE 0C 03  76              LDX  XREG
036D: AC 0D 03  77              LDY  YREG
0370: 60        78              RTS

--End assembly-

113 bytes

Errors: 0
```

```
100  PRINT  CHR$ (4)„BLOAD DIVISION"
110  TEXT : HOME : INVERSE : PRINT „DIVISION-DEMO": NORMAL :
   PRINT
120  PRINT : INPUT „DIVIDEND (0-65535): ";DD
130  IF DD = 0 THEN  POKE 771,0: POKE 772,0: GOTO 150
140 H =  INT (DD / 256): POKE 772,H:L = DD - H * 256: POKE 771,L
150  INPUT „DIVISOR  (0-65535): ";DR
160  IF DR = 0 THEN  POKE 773,0: POKE 774,0: GOTO 180
170 H =  INT (DR / 256): POKE 774,H:L = DR - H * 256: POKE 773,L
180  CALL 768
190  PRINT : PRINT „ASSEMBLER:"
200  PRINT „QUOTIENT: "; PEEK (775) +  PEEK (776) * 256;
   „ - REST: "; PEEK (777) +  PEEK (778) * 256
210  PRINT : PRINT „APPLESOFT:"
220  IF DR = 0 THEN Q = 0:R = 0: GOTO 240
230 Q =  INT (DD / DR):R = DD - Q * DR
240  PRINT „QUOTIENT: ";Q„ - REST: ";R
250  GOTO 120
```

```
                    1              ORG  $300
                    2    *
                    3    * MULT16
                    4    * ======
                    5    *
                    6    MULT16    EQU  $E2B8
                    7    *
                    8    * Diese weitgehend ungekannte
                    9    * Applesoft-Multiplikations-
                   10    * routine multipliziert
                   11    * $64/$65 mal $AD/$AE und legt
                   12    * Ergebnis in X- und A-Register
                   13    * ab.
                   14    *
0300: 4C 09 03     15              JMP  MULT16A
                   16    *
                   19    *
                   20    FAKZ1L    EQU  $64
                   21    FAKZ1H    EQU  $65
                   22    *
                   23    FAKZ2L    EQU  $AD
                   24    FAKZ2H    EQU  $AE
                   25    *
                   26    * Hier Ausgangs-Faktoren poken
                   27    *
0303: 00           28    FAKM1L    HEX  00          ;LL
0304: 00           29    FAKM1H    HEX  00          ;HH
0305: 00           30    FAKM2L    HEX  00          ;LL
0306: 00           31    FAKM2H    HEX  00          ;HH
                   32    *
                   33    * Resultat darf $FFFF nicht
                   34    * überschreiten, sonst erfolgt
                   35    * Out of Memory Fehlermeldung!
                   36    *
0307: 00           37    XLLRES    HEX  00          ;X=LL
0308: 00           38    AHHRES    HEX  00          ;A=HH
                   39    *
0309: AD 03 03     40    MULT16A   LDA  FAKM1L
030C: 85 64        41              STA  FAKZ1L
030E: AD 04 03     42              LDA  FAKM1H
0311: 85 65        43              STA  FAKZ1H
                   44    *
0313: AD 05 03     45              LDA  FAKM2L
0316: 85 AD        46              STA  FAKZ2L
0318: AD 06 03     47              LDA  FAKM2H
031B: 85 AE        48              STA  FAKZ2H
                   49    *
031D: 20 B8 E2     50              JSR  MULT16
0320: 8E 07 03     51              STX  XLLRES      ;LL
0323: 8D 08 03     52              STA  AHHRES      ;HH
0326: 60           53              RTS
```

```
                        1               ORG   $300
                        2    *
                        3    * HEXDUMP
                        4    * ═══════
                        5    *
                        6    IND        EQU   $CE          ;-$CF
                        7    HEXOUT     EQU   $FDDA
                        8    PRINT      EQU   $FDED
                        9    *
0300: 4C 08 03          10   INIT1      JMP   INIT2
                       11    *
                       12    * Anzahl der Bytes/Zeile sowie
                       13    * Anfangs- und Endadresse poken,
                       14    * dann Hexdump aufrufen
                       15    *
0303: 10               16    ANZAHL     DFB   16           ;oder 8
0304: 00 10            17    ANFADR     DA    $1000        ;LL-HH
0306: FF 1F            18    ENDADR     DA    $1FFF        ;LL-HH
                       19    *
0308: AD 04 03         20    INIT2      LDA   ANFADR
030B: 85 CE            21               STA   IND
030D: AD 05 03         22               LDA   ANFADR+1
0310: 85 CF            23               STA   IND+1
                       24    *
0312: 20 33 03         25    DUMP1      JSR   DUMP2
                       26    *
                       27    * Adresse um Anzahl erhöhen
                       28    *
0315: 18               29               CLC
0316: AD 03 03         30               LDA   ANZAHL
0319: 65 CE            31               ADC   IND
031B: 85 CE            32               STA   IND
031D: A5 CF            33               LDA   IND+1
031F: 69 00            34               ADC   #0
0321: 85 CF            35               STA   IND+1
0323: B0 0E            36               BCS   DUMP2        ;Wrap!
                       37    *
                       38    * und mit Endadresse vergleichen
                       39    *
0325: 38               40               SEC
0326: AD 06 03         41               LDA   ENDADR
0329: E5 CE            42               SBC   IND
032B: AD 07 03         43               LDA   ENDADR+1
032E: E5 CF            44               SBC   IND+1
0330: B0 E0            45               BCS   DUMP1
0332: 60               46               RTS                ;Exit
                       47    *
                       48    * $ + Adresse anzeigen
                       49    *
0333: A9 A4            50    DUMP2      LDA   #"$"
0335: 20 ED FD         51               JSR   PRINT
0338: A5 CF            52               LDA   IND+1
033A: 20 DA FD         53               JSR   HEXOUT
```

```
033D: A5 CE     54            LDA  IND
033F: 20 DA FD  55            JSR  HEXOUT
0342: A9 A0     56            LDA  #$A0       ;Space
0344: 20 ED FD  57            JSR  PRINT
                58   *
                59   * Hex-Bytes anzeigen
                60   *
0347: A0 00     61            LDY  #0
0349: B1 CE     62   HEXPRT   LDA  (IND),Y
034B: 20 DA FD  63            JSR  HEXOUT
034E: A9 A0     64            LDA  #$A0
0350: 20 ED FD  65            JSR  PRINT
0353: C8        66            INY
0354: CC 03 03  67            CPY  ANZAHL
0357: 90 F0     68            BCC  HEXPRT
                69   *
0359: A9 A0     70            LDA  #$A0       ;Space
035B: 20 ED FD  71            JSR  PRINT
                72   *
                73   * Ascii-Buchstaben anzeigen
                74   *
035E: A0 00     75            LDY  #0
0360: B1 CE     76   ASCPRT   LDA  (IND),Y
                77   *
                78   * Ctrl-Buchstaben ignorieren
                79   *
0362: 09 80     80            ORA  #%10000000
0364: C9 A0     81            CMP  #$A0
0366: B0 02     82            BCS  ASCPRT1    ;>=$A0
0368: A9 AE     83            LDA  #"."       ;Ctrl="."
036A: C9 FF     84   ASCPRT1  CMP  #$FF       ;DEL="."
036C: D0 02     85            BNE  ASCPRT2
036E: A9 AE     86            LDA  #"."
0370: 20 ED FD  87   ASCPRT2  JSR  PRINT
0373: C8        88            INY
0374: CC 03 03  89            CPY  ANZAHL
0377: 90 E7     90            BCC  ASCPRT
                91   *
0379: A9 8D     92            LDA  #$8D       ;Return
037B: 20 ED FD  93            JSR  PRINT
037E: 60        94            RTS

--End assembly-

127 bytes

Errors: 0
```

```
                    1              ORG  $300
                    2   *
                    3   * DISASSEMBER
                    4   * ==========
                    5   *
                    6   LENGTH     EQU  $2F
                    7   PCL        EQU  $3A
                    8   PCH        EQU  $3B
                    9   INSTDSP    EQU  $F8D0
                   10   COUT       EQU  $FDED
                   11   *
0300: 4C 07 03      12   DISASS1    JMP  DISASS2
                   13   *
                   14   * Anfangs- und Endadresse poken,
                   15   * dann Disassembler aufrufen
                   16   *
0303: 00 F8        17   ANFADR     DA   $F800       ;LL-HH
0305: FF FF        18   ENDADR     DA   $FFFF       ;LL-HH
                   19   *
0307: AD 03 03     20   DISASS2    LDA  ANFADR
030A: 85 3A        21              STA  PCL
030C: AD 04 03     22              LDA  ANFADR+1
030F: 85 3B        23              STA  PCH
                   24   *
                   25   * Disassembliert 1-3 Opcodes
                   26   *
0311: 20 31 03     27   INSTDSP1   JSR  INSTDSP2
                   28   *
                   29   * Length = Anzahl der Opcodes - 1
                   30   *
0314: A6 2F        31              LDX  LENGTH      ;0-2
0316: E8           32              INX              ;1-3
0317: 8A           33              TXA
                   34   *
                   35   * Program-Counter um Length
                   36   * erhöhen...
                   37   *
0318: 18           38              CLC
0319: 65 3A        39              ADC  PCL
031B: 85 3A        40              STA  PCL
031D: A5 3B        41              LDA  PCH
031F: 69 00        42              ADC  #0
0321: 85 3B        43              STA  PCH
0323: B0 0C        44              BCS  INSTDSP2    ;Wrap!
                   45   *
                   46   * ... und dann mit Endadresse
                   47   * vergleichen.
                   48   *
0325: CD 06 03     49              CMP  ENDADR+1
0328: 90 E7        50              BCC  INSTDSP1
032A: A5 3A        51              LDA  PCL
032C: CD 05 03     52              CMP  ENDADR
```

```
032F: 90 E0      53             BCC  INSTDSP1
                 54   *
                 55   * Adresse + Befehl anzeigen
                 56   *
0331: 20 D0 F8   57   INSTDSP2 JSR  INSTDSP
                 58   *
                 59   * Poke 36,36 = Htab 37
                 60   *
0334: A9 24      61             LDA  #36
0336: 85 24      62             STA  36
                 63   *
                 64   * 1-3 Bytes in Ascii anzeigen
                 65   *
0338: A0 00      66             LDY  #0
033A: B1 3A      67   ASCII1    LDA  (PCL),Y
                 68   *
                 69   * Ctrl => Nicht-Ctrl
                 70   *
033C: 09 80      71             ORA  #$80
033E: C9 A0      72             CMP  #$A0
0340: B0 02      73             BCS  ASCII2
0342: 69 40      74             ADC  #$40
0344: 20 ED FD   75   ASCII2    JSR  COUT
0347: C8         76             INY
0348: C4 2F      77             CPY  LENGTH
034A: F0 EE      78             BEQ  ASCII1
034C: 90 EC      79             BCC  ASCII1
034E: 60         80             RTS

--End assembly-

79 bytes

Errors: 0
```

2.2.11. SCROLL DOWN

```
100  HOME : HTAB 8: PRINT „Scrolldown für Apple IIe"
110 X$= „300:8D 07 C0 20 AA CC 8D 06 C0 60 ND7D2G":
    FOR X = 1 TO  LEN (X$): POKE 511 + X, ASC ( MID$ (X$,X,1)) +
    128: NEXT : POKE 72,0:  CALL  - 1/.4
120 FOR Y = 1 TO 10: FOR X = 1 TO 23. CALL 768: NEXT X:
    VTAB 24: FOR X = 1 TO 23: PRINT : NEXT X: NEXT Y
```

```
                        1               ORG  $2F0
                        2    *
                        3    * RAMSUCHER
                        4    * ==========
                        5    *
                        6    * Mit BRUN starten, dann vom
                        7    * Monitor aus mit
                        8    * xxyyzz...Ctrl-Y Return
                        9    * Suchstring eingeben, der
                        10   * bis zu 10 Hex-Zahlen
                        11   * umfassen kann.
                        12   *
                        13   * Durchsucht wird:
                        14   * $0800-$BFFF
                        15   * $D000-FFFF Bank1 und
                        16   * $D000-DFFF Bank2.
                        17   *
                        18   * Achtung: Bei Apple IIe bewirkt
                        19   * Ctrl-Y, wenn 80-Z-Karte einge-
                        20   * schaltet ist, zusätzlich den
                        21   * Home-Befehl.
                        22   *
                        23   * U.Stiehl/19.5.84
                        24   *
                        25   IND      EQU  $CE          ;-$CF
                        26   LEN      EQU  $FD
                        27   FLAG     EQU  $FE
                        28   TEMP     EQU  $FF
                        29   PUFANF   EQU  $0200
                        30   PUFMID   EQU  $0280
                        31   BELL     EQU  $FF3A
                        32   HEXOUT   EQU  $FDDA
                        33   PRINT    EQU  $FDED
                        34   *
02F0: A9 4C             35   CTRLY1   LDA  #$4C
02F2: 8D F8 03          36            STA  $3F8
02F5: A9 00             37            LDA  #<CTRLY2
02F7: 8D F9 03          38            STA  $3F9
02FA: A9 03             39            LDA  #>CTRLY2
02FC: 8D FA 03          40            STA  $3FA
02FF: 60                41            RTS
                        42   *
0300: A2 00             43   CTRLY2   LDX  #0
0302: A0 00             44            LDY  #0
0304: 84 FE             45            STY  FLAG
                        46   *
0306: BD 00 02          47   HEX1     LDA  PUFANF,X
0309: C9 99             48            CMP  #$99         ;Ctrl-Y
030B: F0 38             49            BEQ  HEX8
                        50   *
030D: C9 B0             51            CMP  #„0"
030F: 90 31             52            BCC  HEX7         ;Error
0311: C9 BA             53            CMP  #„:"
```

```
0313: B0 04      54              BCS  HEX2
0315: 29 0F      55              AND  #%00001111
0317: 10 0D      56              BPL  HEX3        ;stets
                 57   *
0319: C9 C1      58   HEX2       CMP  #„A"
031B: 90 25      59              BCC  HEX7        ;Error
031D: C9 C7      60              CMP  #„G"
031F: B0 21      61              BCS  HEX7        ;Error
0321: 29 0F      62              AND  #%00001111
0323: 18         63              CLC
0324: 69 09      64              ADC  #$09
                 65   *
0326: 24 FE      66   HEX3       BIT  FLAG
0328: 30 0A      67              BMI  HEX5
                 68   *
032A: 0A         69   HEX4       ASL              ;links
032B: 0A         70              ASL
032C: 0A         71              ASL
032D: 0A         72              ASL
032E: 85 FF      73              STA  TEMP
0330: C6 FE      74              DEC  FLAG        ;$FF
0332: 30 09      75              BMI  HEX6        ;stets
                 76   *
0334: 18         77   HEX5       CLC              ;rechts
0335: 65 FF      78              ADC  TEMP
0337: 99 80 02   79              STA  PUFMID,Y
033A: C8         80              INY
033B: E6 FE      81              INC  FLAG        ;$00
                 82   *
033D: E8         83   HEX6       INX
033E: E0 15      84              CPX  #21         ;10 Hex.
0340: 90 C4      85              BCC  HEX1
                 86   *
0342: 4C 3A FF   87   HEX7       JMP  BELL
                 88   *
0345: E0 02      89   HEX8       CPX  #2
0347: 90 F9      90              BCC  HEX7        ;Error
                 91   *
                 92   * 2, 4, 6, 8, A, C, E, 10, 12, 14
                 93   *
0349: 8A         94              TXA
034A: 4A         95              LSR
034B: B0 F5      96              BCS  HEX7
034D: 84 FD      97              STY  LEN
                 98   *
                 99   * Hex-String suchen ab $800
                100   * ========================
                101   *
                102   * Zunächst Bank2: $D000-DFFF
                103   *
034F: A2 83     104              LDX  #$83        ;C083
0351: A0 00     105              LDY  #$00        ;Y=0
0353: 84 CE     106              STY  IND         ;LOW
```

```
0355: A9 08      107           LDA  #$08
0357: 85 CF      108           STA  IND+1       ;HIGH
0359: D0 11      109           BNE  LOOP1
                 110  *
                 111  * C083 (Bank 2) oder C08B (Bank 1)
                 112  *
035B: 20 87 03   113  LOOP0     JSR  ROMRD
035E: A9 8D      114           LDA  #$8D        ;Return
0360: 20 ED FD   115           JSR  PRINT
0363: A9 C0      116           LDA  #$C0        ;High
0365: 20 DA FD   117           JSR  HEXOUT
0368: 8A         118           TXA              ;Low
0369: 20 BC 03   119           JSR  DISPLAY1
                 120  *
036C: B1 CE      121  LOOP1     LDA  (IND),Y
036E: CD 80 02   122           CMP  PUFMID      ;1.CHAR
0371: F0 29      123           BEQ  TEST1
                 124  *
0373: E6 CE      125  LOOP2     INC  IND
0375: D0 F5      126           BNE  LOOP1
0377: E6 CF      127           INC  IND+1
0379: A5 CF      128           LDA  IND+1
037B: C9 C0      129           CMP  #$C0        ;C000
037D: F0 0F      130           BEQ  BANKER
037F: C9 E0      131           CMP  #$E0        ;E000
0381: F0 11      132           BEQ  BANK1
0383: C9 00      133           CMP  #$00        ;FFFF+1
0385: D0 E5      134           BNE  LOOP1
                 135  *
0387: AD 81 C0   136  ROMRD     LDA  $C081
038A: AD 81 C0   137           LDA  $C081
038D: 60         138           RTS
                 139  *
038E: A9 D0      140  BANKER    LDA  #$D0        ;D000
0390: 85 CF      141           STA  IND+1
0392: D0 C7      142           BNE  LOOP0
                 143  *
                 144  * Bereits Bank1?
                 145  *
0394: E0 8B      146  BANK1     CPX  #$8B
0396: F0 D4      147           BEQ  LOOP1
0398: A2 8B      148           LDX  #$8B
039A: D0 F2      149           BNE  BANKER
                 150  *
                 151  * Pufmid-String = Match?
                 152  *
039C: C8         153  TEST1     INY
039D: C4 FD      154           CPY  LEN
039F: B0 0B      155           BCS  DISPLAY
```

```
03A1: B9 80 02  156           LDA  PUFMID,Y
03A4: D1 CE     157           CMP  (IND),Y
03A6: F0 F4     158           BEQ  TEST1
03A8: A0 00     159  TEST2    LDY  #0
03AA: F0 C7     160           BEQ  LOOP2
                161  *
                162  * Anzeige
                163  *
03AC: 20 87 03  164  DISPLAY  JSR  ROMRD
03AF: A5 CF     165           LDA  IND+1       ;HIGH
03B1: 20 DA FD  166           JSR  HEXOUT
03B4: A5 CE     167           LDA  IND         ;LOW
03B6: 20 BC 03  168           JSR  DISPLAY1
03B9: 4C A8 03  169           JMP  TEST2
                170  *
03BC: 20 DA FD  171  DISPLAY1 JSR  HEXOUT
03BF: A9 A0     172           LDA  #$A0
03C1: 20 ED FD  173           JSR  PRINT
03C4: 20 C8 03  174           JSR  RAMRD
03C7: 60        175           RTS
                176  *
03C8: BD 00 C0  177  RAMRD    LDA  $C000,X
03CB: BD 00 C0  178           LDA  $C000,X
03CE: 60        179           RTS
```

2.2.13 BELL

```
                1             ORG  $0300
                2  *
                3  * BELL
                4  * ====
                5  *
                6  WAIT       EQU  $FCA8       ;Warteschleife
                7  SPKR       EQU  $C030       ;Lautsprecher
                8  *
0300: 8D 1E 03  9  BELL1      STA  ASAVE
0303: 8C 1F 03  10            STY  YSAVE
0306: AC 1C 03  11            LDY  WERT1
0309: AD 1D 03  12 BELL2      LDA  WERT2
030C: 20 A8 FC  13            JSR  WAIT
030F: AD 30 C0  14            LDA  SPKR
0312: 88        15            DEY
0313: D0 F4     16            BNE  BELL2
0315: AD 1E 03  17            LDA  ASAVE
0318: AC 1F 03  18            LDY  YSAVE
031B: 60        19            RTS
031C: 40        20 WERT1      HEX  40          ;änder-
031D: 20        21 WERT2      HEX  20          ;bar
031E: 00        22 ASAVE      HEX  00
031F: 00        23 YSAVE      HEX  00
```

```
                    1               ORG  $300
                    2   *
                    3   * RESET.TEST
                    4   * ==========
                    5   *
                    6   * Dieses Testprogramm zeigt,
                    7   * daß mit jedem Hardware-Reset
                    8   * durch Ctrl-Reset-Tasten
                    9   * der Stackpointer 3 Stellen
                   10   * herabgesetzt wird.
                   11   *
                   12   HOME      EQU  $FC58
                   13   STACK     EQU  $0100
                   14   DOSCOLD   EQU  $03D3
                   15   *
0300: 20 58 FC     16   START     JSR  HOME
                   17   *
                   18   * Stack löschen und Stack-
                   19   * pointer heraufsetzen.
                   20   *
0303: A2 00        21             LDX  #0
0305: 8A           22             TXA
0306: 9D 00 01     23   CLR       STA  STACK,X
0309: E8           24             INX
030A: D0 FA        25             BNE  CLR
                   26   *
030C: A2 FF        27             LDX  #$FF
030E: 9A           28             TXS              ;S=FF
                   29   *
                   30   * Alten Reset-Vektor retten
                   31   * und neuen Vektor setzen.
                   32   *
030F: AD F2 03     33             LDA  $3F2
0312: 8D AC 03     34             STA  RESETSAV
0315: A9 3B        35             LDA  #<RESET
0317: 8D F2 03     36             STA  $3F2
                   37   *
031A: AD F3 03     38             LDA  $3F3
031D: 8D AD 03     39             STA  RESETSAV+1
0320: A9 03        40             LDA  #>RESET
0322: 8D F3 03     41             STA  $3F3
                   42   *
0325: AD F4 03     43             LDA  $3F4
0328: 8D AE 03     44             STA  RESETSAV+2
                   45   *
                   46   * Reset-Highbyte EOR mit $A5
                   47   *
032B: AD F3 03     48             LDA  $3F3
032E: 49 A5        49             EOR  #$A5
0330: 8D F4 03     50             STA  $3F4
                   51   *
                   52   * Man kann 6mal Ctrl-Reset
                   53   * drücken. Danach Exit.
```

```
                        54   *
0333: A9 07             55            LDA  #7          ;6mal
0335: 8D AB 03          56            STA  COUNT
                        57   *
                        58   * Return-Adresse auf Stack
                        59   * schieben durch JSR
                        60   *
0338: 20 8C 03          61            JSR  RETURN1
                        62   *
                        63   * Hier Reset-Vektor
                        64   *
033B: BA                65   RESET    TSX
033C: 8E AA 03          66            STX  STACKPTR
033F: A9 00             67            LDA  #$00
0341: 85 28             68            STA  $28
0343: A9 04             69            LDA  #$04
0345: 85 29             70            STA  $29
0347: A0 00             71            LDY  #$00
                        72   *
                        73   * Stack $01ED-$01FF anzeigen
                        74   *
0349: A2 ED             75            LDX  #$ED
                        76   *
034B: BD 00 01          77   DISPLAY  LDA  STACK,X
034E: 8D A9 03          78            STA  HEXBYTE
0351: 4A                79            LSR
0352: 4A                80            LSR
0353: 4A                81            LSR
0354: 4A                82            LSR
0355: 09 B0             83            ORA  #$B0
0357: C9 BA             84            CMP  #$BA
0359: 90 02             85            BCC  LEFT
035B: 69 06             86            ADC  #$06
035D: EC AA 03          87   LEFT     CPX  STACKPTR
0360: D0 02             88            BNE  LEFTA
0362: 29 7F             89            AND  #$7F
0364: 91 28             90   LEFTA    STA  ($28),Y     ;Screen
0366: C8                91            INY
                        92   *
0367: AD A9 03          93            LDA  HEXBYTE
036A: 29 0F             94            AND  #$0F
036C: 09 B0             95            ORA  #$B0
036E: C9 BA             96            CMP  #$BA
0370: 90 02             97            BCC  RIGHT
0372: 69 06             98            ADC  #$06
0374: EC AA 03          99   RIGHT    CPX  STACKPTR
0377: D0 02            100            BNE  RIGHTA
0379: 29 7F            101            AND  #$7F
037B: 91 28            102   RIGHTA   STA  ($28),Y
037D: C8               103            INY
                       104   *
037E: E8               105            INX
037F: D0 CA            106            BNE  DISPLAY
0381: CE AB 03         107            DEC  COUNT
```

```
0384: AD AB 03  108           LDA  COUNT
0387: F0 08     109           BEQ  EXIT
                110  *
                111  * Hier wird auf Ctrl-Reset
                112  * gewartet
                113  *
0389: 4C 89 03  114  ENDLOS   JMP  ENDLOS
038C: 20 90 03  115  RETURN1  JSR  RETURN2
038F: 60        116           RTS
0390: 60        117  RETURN2  RTS
                118  *
0391: AD AC 03  119  EXIT     LDA  RESETSAV
0394: 8D F2 03  120           STA  $3F2
0397: AD AD 03  121           LDA  RESETSAV+1
039A: 8D F3 03  122           STA  $3F3
039D: AD AE 03  123           LDA  RESETSAV+2
03A0: 8D F4 03  124           STA  $3F4
                125  *
                126  * Doscold initialisert Stack
                127  *
03A3: A2 FF     128           LDX  #$FF
03A5: 9A        129           TXS
03A6: 4C D3 03  130           JMP  DOSCOLD
                131  *
03A9: 00        132  HEXBYTE  HEX  00
03AA: 00        133  STACKPTR HEX  00
03AB: 00        134  COUNT    HEX  00
03AC: 00 00 00  135  RESETSAV HEX  000000

--End assembly-

175 bytes

Errors: 0
```

```
                        1               ORG  $300
                        2    *
                        3    * RESET.NORMAL
                        4    * ============
                        5    *
                        6    VEKPAGE    EQU  $3F0
                        7    *
0300: A2 00             8               LDX  #0
0302: BD 1C 03          9    LOOP       LDA  VEKTOREN,X
0305: 9D F0 03          10              STA  VEKPAGE,X
0308: E8                11              INX
0309: E0 10             12              CPX  #16
030B: D0 F5             13              BNE  LOOP
                        14   *
                        15   * Language Card abstellen
                        16   *
030D: AD 81 C0          17              LDA  $C081      ;RDROM
0310: AD 81 C0          18              LDA  $C081      ;WRBK2
                        19   *
                        20   * Screen und Keyboard einstellen
                        21   *
0313: 20 89 FE          22              JSR  $FE89      ;SETKBD
0316: 20 93 FE          23              JSR  $FE93      ;SETVID
                        24   *
                        25   * Applesoft-Kaltstart
                        26   *
0319: 4C 00 E0          27              JMP  $E000      ;BASIC
                        28   *
031C: 59 FA             29   VEKTOREN  HEX  59FA        ;BRK
031E: 03 E0 45          30              HEX  03E045     ;RESET
0321: 4C 58 FF          31              HEX  4C58FF     ;&
0324: 4C 65 FF          32              HEX  4C65FF     ;CTRL-Y
0327: 4C 65 FF          33              HEX  4C65FF     ;NMI
032A: 65 FF             34              HEX  65FF       ;IRQ
                        35   *
                        36   * Reset-Varianten:
                        37   *
                        38   * 3F2: 03 E0 45: Applesoft
                        39   * 3F2: 59 FF 5A: Monitor
                        40   * 3F2: BF 9D 38: DOS 3.3
                        41   * 3F2: 00 BE 1B: ProDOS

--End assembly-

44 bytes

Errors: 0
```

Softswitches des Apple IIe

```
LDA $C000 READ KEYBOARD
STA $C000 DISABLE 80COL STORE
STA $C001 ENABLE 80COL STORE
STA $C002 READ MAIN47.5
STA $C003 READ AUX47.5
STA $C004 WRITE MAIN47.5
STA $C005 WRITE AUX47.5
STA $C006 ENABLE SLOT C100-CFFF
STA $C007 ENABLE INTERNAL C100-CFFF
STA $C008 READ/WRITE MAIN ZP+BANK
STA $C009 READ/WRITE AUX. ZP+BANK
STA $C00A ENABLE INTERNAL C300-C3FF
STA $C00B ENABLE SLOT C300-C3FF
STA $C00C DISPLAY 40COL
STA $C00D DISPLAY 80COL
STA $C00E DISABLE ALTCHARSET
STA $C00F ENABLE ALTCHARSET
LDA $C010 KEYBOARD STROBE
LDA $C011 BPL BANK1 SELECTED
          BMI BANK2 SELECTED
LDA $C012 BPL READ ROM12
          BMI READ BANK
LDA $C013 BPL MAIN48 READ ENABLED
          BMI AUX47.5 READ ENABLED
LDA $C014 BPL MAIN48 WRITE ENABLED
          BMI AUX47.5 WRITE ENABLED
LDA $C015 BPL SLOT C100-CFFF ACTIVE
          BMI INTERNAL C100-CFFF ACTIVE
LDA $C016 BPL MAIN ZP+BANK ACTIVE
          BMI AUX. ZP+BANK ACTIVE
LDA $C017 BPL SLOT C300-C3FF ACTIVE
          BMI INTERNAL C300-C3FF ACTIVE
LDA $C018 BPL 80COL STORE OFF
          BMI 80COL STORE ON
LDA $C019 VERTICAL BLANKING STATUS
LDA $C01A BPL GRAPHICS MODE ACTIVE
          BMI TEXT MODE ACTIVE
LDA $C01B BPL MIXED MODE OFF
          BMI MIXED MODE ON
LDA $C01C BPL PAGE 1 ACTIVE
          BMI PAGE 2 ACTIVE
LDA $C01D BPL LO-RES MODE ON
          BMI HI-RES MODE ON
LDA $C01E BPL ALTCHARSET OFF
          BMI ALTCHARSET ON
LDA $C01F BPL 40COL DISPLAY ACTIVE
          BMI 80COL DISPLAY ACTIVE
LDA $C020 CASSETTE OUT
LDA $C030 CLICK SPEAKER
LDA $C040 UTILITY STROBE
LDA $C050 ENABLE GRAPHICS MODE
LDA $C051 ENABLE TEXT MODE
LDA $C052 DISABLE MIXED MODE
LDA $C053 ENABLE MIXED MODE
LDA $C054 ENABLE PAGE 1
LDA $C055 ENABLE PAGE 2
LDA $C056 ENABLE LO-RES MODE
LDA $C057 ENABLE HI-RES MODE
LDA $C058 ANNUNCIATOR 0 OFF
LDA $C059 ANNUNCIATOR 0 ON
LDA $C05A ANNUNCIATOR 1 OFF
LDA $C05B ANNUNCIATOR 1 ON
LDA $C05C ANNUNCIATOR 2 OFF
LDA $C05D ANNUNCIATOR 2 ON
LDA $C05E ANNUNCIATOR 3 OFF
LDA $C05F ANNUNCIATOR 3 ON
LDA $C060 CASSETTE IN
LDA $C061 SWITCH INPUT 0 = OPEN APPLE
LDA $C062 SWITCH INPUT 1 = SOLID APPLE
LDA $C063 SWITCH INPUT 2
LDA $C064 ANALOG INPUT 0
LDA $C065 ANALOG INPUT 1
LDA $C066 ANALOG INPUT 2
LDA $C067 ANALOG INPUT 3
LDA $C070 ANALOG INPUT STROBE
LDA $C080 READ BANK2
LDA $C081 (TWICE) WRITE BANK2.
          READ ROM 12
LDA $C082 READ ROM12
LDA $C083 (TWICE) WRITE/READ BANK2
          ($C084-$C087 DUPLICATES
          $C080-$C083)
LDA $C088 READ BANK1
LDA $C089 (TWICE) WRITE BANK1.
          READ ROM12
LDA $C08A READ ROM12
LDA $C08B (TWICE) WRITE/READ BANK1
          ($C08C-$C08F DUPLICATES
          $C088-$C08B)
```

3. Speicherverwaltung

Bezüglich der Bildschirmspeicher-Softswitches sei auf Kap. 5 verweisen.

3.1. ROM, RAM und Softswitches

Der Speicher des Apple II Plus mit LC und des Apple IIe (mit 64K-Karte) läßt sich in drei Gruppen einteilen:

3.1.1. Normales ROM

$D000-$F7FF:	Applesoft-ROM
$F800-$FFFF:	F8-Monitor-ROM

ferner zusätzlich beim Apple IIe

$C100-$CFFF:	Internes INTCXROM (Erweiterung des F8-ROMs)
$C300-$C3FF:	Internes INTC3ROM (identisch mit $C300-$3CFF von INTCXROM)

INTCXROM-Routinen sollten stets so aufgerufen werden:

```
STA   $C007     ;Read-Enable Internal $C100-$CFFF
JSR   ROUTINE
STA   $C006     ;Read-Enable Slot $C100-$CFFF
```

Wenn man das INTCXROM nicht mehr mit STA $C006 desaktiviert, kann auf Slot-Interface-Karten nicht mehr zugegriffen werden, weil dann deren RAM nicht mehr lesefähig ist. Ein Drucker-Interface ist dann gewissermaßen „tot". Umgekehrt ist beim neuen Apple IIc das INTCXROM ständig aktiv, weil periphere Slots im bisherigen Sinne gar nicht mehr existieren.

Der Apple IIe hat zwei Slots 3, einen normalen, hinteren Slot 3 für beliebige Interface-Karten sowie einen mitten auf der Platine plazierten, sogenannten internen Slot 3, der nur für die 64K-Karte bzw. Apple-80-Zeichenkarte bestimmt ist. Beide Slots 3 können nicht gleichzeitig belegt sein. Das INTC3ROM wird mit STA $C00A aktiviert und mit STA $C00B desaktiviert. Nach STA $C00B denken viele Programme und Betriebssysteme (Pascal usw.), daß eine 80-Zeichenkarte fehlt, wenngleich diese im internen Slot 3 steckt, da sie im normalen, äußeren Slot 3 kein RAM vorfinden. Mit

```
STA $C00B
JMP $FAA6 ;Bootadresse
```

kann man dann z.B. für Pascal, Quickfile usw. 40-Zeichendarstellung erzwingen.

3.1.2. Reiner Softswitch-Bereich („anomales RAM")

$C000-$C07F:	Diverse Softswitches
$C080-$C08F:	LC-Softswitches
$C090-$C09F:	Slot 1 Softswitches
$C0A0-$C0AF:	Slot 2 Softswitches
$C0B0-$C0BF:	Slot 3 Softswitches
$C0C0-$C0CF:	Slot 4 Softswitches
$C0D0-$C0DF:	Slot 5 Softswitches
$C0E0-$C0EF:	Slot 6 Softswitches
$C0F0-$C0FF:	Slot 7 Softswitches

Ein Softswitch ist eine „anomale" Speicherstelle, die in der Regel nicht voll decodiert, d.h. gelesen oder beschrieben wird. Wenn man z.B. mit LDA $C012 von diesem Softswitch liest, erhält man in der Regel entweder nur den Wert $0D (Bit 7 off) oder $8D (Bit 7 on), d.h. signifikant gelesen wird eigentlich nur Bit 7. In einen Softswitch kann man keinen beliebigen Wert $00-$FF speichern, der dann von dort gelesen werden könnte. Bei Softswitches ist nicht der Wert als solcher wichtig, sondern daß der Softswitch überhaupt adreßmäßig angesprochen wird: LDA #$FF STA $C009 hat dieselbe Funktion wie etwa LDA #$00 STA $C009. Der einzige voll decodierte Softswitch ist $C000 für die Tastatur, der echte Tastaturwerte – indessen nur im Bereich $80-$FF – enthalten kann, je nachdem, welche Taste gerade gedrückt wurde. Der Apple IIe ist insofern softswitchmäßig intelligenter geworden, als nunmehr nicht nur wie bisher beim Apple II Plus Softswitches existieren, mit deren Hilfe bestimmte Speicherzustände usw. herbeigeführt werden können, sondern auch Softswitches imple-

mentiert wurden, die man lesen kann, um zu ermitteln, welche Speicherzustände (z.B. LC lesefähig?), Videozustände (z.B. HGR aktiv?) im Moment vorliegen.

Die Softswitches $C000-$C08F sind auf Seite 82 vollständig aufgeführt. Man beachte die dortige LDA-STA-Konvention, die unbedingt eingehalten werden sollte. Beispiele: LDA $C000 liest die Tastatur, während STA $C000 „Disable 80 Column Store" bewirkt. LDA $C030 klickt den Lautsprecher einmal, während STA $C030 ihn zweimal zum Klicken bringen würde. STA $C00F bedeutet „Enable Alternate Character Set", während LDA $C01E prüft, ob der Zweitzeichensatz aktiv ist, usw. Anstelle von STA (LDA) ist selbstverständlich auch STX und STY (LDX und LDY) möglich. Für LDA $C010 (Keyboard Strobe) ist auch BIT $C010 üblich und möglich.

3.1.3. Slot-Bereich (teils ROM, teils RAM, teils Softswitches)

$C100-$C1FF:	Slot 1 Bereich (meist für Drucker-Karte verwendet)
$C200-$C2FF:	Slot 2 Bereich
$C300-$C3FF:	Slot 3 Bereich (meist für 80-Zeichenkarte verwendet)
$C300-$C3FF:	bei IIe Auxiliary Slot 3 Bereich; nur für 80-Zeichenkarte
$C400-$C4FF:	Slot 4 Bereich (oft für Z80-Karte verwendet)
$C500-$C5FF:	Slot 5 Bereich
$C600-$C6FF:	Slot 6 Bereich (meist für DOS-Controller verwendet)
$C700-$C7FF:	Slot 7 Bereich (meist für RGB-Karte verwendet)

Beispiel für reines ROM

DOS-Controller enthalten reines ROM. Wenn z.B. ein Controller in Slot 6 steckt, dann ist $C600-$C6FF reines ROM. Die für den Diskettenzugriff erforderlichen Softswitches sind dann im Bereich $C0E0-$C0EF, z.B.:

```
LDA   $C0E8    schaltet Motor aus
LDA   $C0E9    schaltet Motor an

LDA   $C0EA    aktiviert Drive 1
LDA   $C0EB    aktiviert Drive 2
```

Konkretes Beispiel (Write-Protect-Prüfung bei Slot 6, Drive 1):

```
WP?     LDA   $C0EA    ;aktiviert Drive 1
        LDA   $C0E9    ;schaltet Motor an
        LDX   #10      ;10 mal 0,1 Sekunde
WAIT    LDA   #196     ;0,1 Sekunde
        JSR   $FCA8    ;Warteschleife
        DEX
        BNE   WAIT
        LDA   $C0ED    ;„Latch-Load"
        LDA   $C0EE    ;„Latch-Input"
        STA   $FF      ;A zwischenspeichern
        LDA   $C0E8    ;schaltet Motor aus
        LDA   $FF      ;Bit 7 on?
        BMI   WP       ;Ja, dann Schreibschutz
        RTS            ;bei Bit 7 off kein Schreibschutz
WP      JMP   $FF3A    ;Piepston bei Schreibschutz ausgeben
```

Beispiel für reines RAM

Die RAM-Disk-Karten der Firma IBS, Bielefeld, zumindest die mir bekannte AP20, haben 256 Bytes reines Interface-RAM, in das man einen eigenen RAM-Disk-Driver BLOADen könnte.

Beispiel für ROM + Softswitches

Meine Drucker-Interface-Karte enthält z.B. sowohl teils reines ROM als auch Softswitches im $C100-$C1FF-Bereich, wenn die Karte in Slot 1 steckt. Ähnliches gilt z.B. für meine Z80-Karte, die in Slot 4 steckt. Wenn man z.B. mit C400:00 einen Wert in diese Speicherstelle pokt, wird der Z80-Prozessor aktiviert.

3.1.4. Sloterweiterungs-ROM

$C800-$CFFF

Jede Interface-Karte in den Slots 1-7 könnte theoretisch ein Zusatz-ROM haben, das adreßmäßig dem Bereich $C800-$CFFF entspricht. Allerdings kann zu einem gegebenen Zeitpunkt immer nur eines dieser Erweiterungs-ROMs

aktiv sein. Die anderen ROMs werden mit LDA $CFFF – ebenfalls eine Art Softswitch – desaktiviert. Grafikdump-Routinen von Drucker-Interface-Karten benutzen meist diesen C8-Bereich für ihre Dump-Routine.

3.1.5. Bildschirm-RAM

Das Bildschirm-RAM ist einerseits normaler RAM-Bereich und andererseits doch wiederum „nicht ganz normal", weil alles, was in diesen Bereich gepokt wird, als Text, Lores oder Hires am Bildschirm erscheint, je nachdem welche Softswitches zuvor aktiviert wurden.

$0400-$07FF	40 Z/Z Page 1 (Text und Lores-Grafik)
$0800-$0BFF	40 Z/Z Page 2 (Text und Lores-Grafik)
$2000-$3FFF	HGR Page 1
$4000-$5FFF	HGR Page 2

Für Apple IIe mit 80 Z/Z-Karte gilt zusätzlich:

a) Doppeltes Text-RAM (= normale 80-Zeichendarstellung)

$0400-$07FF:	„oben" auf 64K-Karte gerade Spalten (0, 2, 4, 6...78)
$0400-$07FF:	„unten" auf Motherboard ungerade Spalten (1, 3, 5, 7...79)

b) Doppeltes Lores-RAM

$0400-$07FF:	„oben" gerade Spalten (0, 2, 4, 6...78)
$0400-$07FF:	„unten" ungerade Spalten (1, 3, 5, 7...79)

Ferner für Apple IIe mit 64K-Karte

c) Doppeltes Hires-RAM

$2000-$3FFF:	„oben" gerade Spalten (0, 2, 4, 6...558)
$2000-$3FFF:	„unten" ungerade Spalten (1, 3, 5, 7...559)

3.1.6. Normales RAM

$0000-$00FF:	Zero-Page
$0100-$01FF:	Stack
$0200-$02FF:	Eingabepuffer
$0300-$03CF:	freier Page 3 Bereich
$03D0-$03FF:	Page 3 Vektoren
$0C00-$1FFF:	frei
$4000-$BFFF:	frei
$D000-$DFFF:	reine Bank 1 LC
$D000-$DFFF:	reine Bank 2 LC
$E000-$FFFF:	gemeinsame Bank 1 und Bank 2 LC

Diese Einteilung ist zu abstrakt, da sie alle Bildschirmspeicherbereiche ausklammert. Für die Praxis gilt folgende „normale" Einteilung:

$0000-$00FF: Zero-Page

Davon frei für Assembler-Programme bei Apple IIe unter Monitor, Applesoft, DOS 3.3/ProDOS:

$0006-$0007
$0009
$0019-$001E
$00CE-$0CF
$00D7
$00E3
$00EB-$00EF
$00FA-$00FE

Bei der Zero-Page muß man unterscheiden zwischen Stellen, die vorübergehend Werte enthalten, die nachher nicht mehr gebraucht werden (Scratchpad-Stellen), sowie Stellen, die zwar teils veränderliche Werte enthalten, welche jedoch ständig gebraucht werden („lebenswichtige Stellen"). So wird z.B. die Stelle $00FF gelegentlich von Applesoft für STR$ benötigt, ist jedoch ansonsten frei. Das heißt jedoch zugleich, daß ein dort von einem Assemblerprogramm gepokter Wert nicht „ewig" erhalten bleibt, so daß ein Assemblerprogramm diese Stelle ebenfalls nur als Scratchpad benutzen kann. Umgekehrt findet sich z.B. in

$0032 das permanent benötigte INVFLG, so daß diese Zero-Page-Adresse niemals als Scratchpad für irgendein anderes Programm verwendet werden kann.
Die obengenannten Zero-Page-Stellen sind sozusagen „ewig" frei. Sie werden großenteils nicht einmal durch Neubooten zerstört.

$0100-$01FF: Stack

Der Bereich $0100-$0110 kann vorübergehend (!) als Scratch-Area benutzt werden, z.B. für Softswitching für 64K-Karte. Man beachte, daß $0100-$0110 ebenfalls die Scratch-Area für die Applesoft-FIN-Routine ist.

$0200-$02FF: Eingabepuffer

Bereich $0200-$02FF kann vorübergehend (!) als Scratch-Area benutzt werden. Unter ProDOS können jedoch Programme nicht in den Eingabepuffer geladen werden, da ProDOS ebenfalls $0200-$02FF als Scratchpad benutzt.

$0300-$03CF: stets frei für kurze Assemblerprogramme

$03D0-$03FF: Vektoren-Tabelle

$0400-$07FF: Meist nicht frei wegen Bildschirm

Die versteckten Bildschirm-Scratchpad-Stellen sind nur für Interface-Karten gedacht und sollten von Assemblerprogrammen nicht als Datenspeicher benutzt werden.

$0800-$0802: Wegen Applesoft sollten Maschinenprogramme nicht hier beginnen, da sonst der LIST-Befehl „durchdreht".

$0803-$95FF: Meist frei, wenn keine Hires-Grafik benutzt wird.

$9600-$BFFF: Wegen DOS 3.3 oder ProDOS in der Regel nicht frei.

LC-Einteilung für DOS 3.3:

$D000-$DFFF: Bank 1 der LC (meist frei)
$D000-$FFFF: Bank 2 der LC (kann in LC gemovtes DOS enthalten)

LC-Einteilung für ProDOS:

$D000-$FFFF:	Bank 1 der LC (PRODOS)
$D000-$DFFF:	Bank 2 der LC (Dieser Bereich kann unter ProDOS in der Regel benutzt werden, da sich hier nur das ProDOS-Reboot-Programm für geschützte Programme usw. befindet, das vom Bank-1-PRODOS normalerweise nicht aufgerufen wird.)

3.1.7. Language Card Softswitches

Switch	Read	Write	C011-BK1?	C012-RDROM?
2 X LDA $C08B	Bank 1	Bank 1	BPL-ja	BMI-nein
1 X LDA $C08A	ROM 1	ROM 1	BPL-ja	BPL-ja
1 X LDA $C088	Bank 1	ROM 1	BPL-ja	BMI-nein
2 X LDA $C089	ROM 1	Bank 1	BPL-ja	BPL-ja
2 X LDA $C083	Bank 2	Bank 2	BMI-nein	BMI-nein
1 X LDA $C082	ROM 2	ROM 2	BMI-nein	BPL-ja
1 X LDA $C080	Bank 2	ROM 2	BMI-nein	BMI-nein
2 X LDA $C081	ROM 2	Bank 2	BMI-nein	BPL-ja

Die Language Card ist bekanntlich in 2 Bänke (2 mal $D000-$FFFF) unterteilt, die sich bezüglich des Bereichs $D000-$DFFF (2 mal 4K) unterscheiden und bezüglich des Bereichs $E000-$FFFF (1 mal 12K) überlappen, d.h. der Bereich $D000-$DFFF existiert physisch zweimal und der übrige Bereich $E000-$FFFF physisch nur einmal. Der entsprechende ROM-Bereich $D000-$FFFF existiert ebenfalls physisch nur einmal, wird jedoch softswitchmäßig so behandelt, als wäre er zweimal vorhanden. Die Softswitches, die ROM-Write „ermöglichen", sind sinnleer, da man in Read Only Memory per definitionem natürlich nichts schreiben kann. Die entsprechenden Softswitches veranlassen den Prozessor, es trotzdem zu versuchen, wie unser Test zeigt.

Der Apple IIe verfügt neben den Read-Write Enable-Disable Softswitches noch über weitere Softswitches, die den R/W-Status lesen können, und zwar bei der LC:

```
LDA   $C011      (A enthält danach $0D oder $8D)
BPL   Bank 1 aktiv
BMI   Bank 2 aktiv

LDA   $C012      (A enthält danach $0D oder $8D)
BPL   ROM read-enabled
BMI   Bank read-enabled
```

Versuchen Sie bei unserem nachfolgenden, nur für den IIe bestimmten Test nachzuvollziehen, welche $0D/$8D-BPL/BMI-Werte sich am Ende des Programms in STATSAVE, BK1SAVE und BK2SAVE befinden. Dann beherrschen Sie die LC-Softswitches aus dem Effeff!

```
                        1               ORG  $1000
                        2   *
                        3   * Language Card Test
                        4   *
1000: AD 89 C0          5               LDA  $C089      ;WRBK1
1003: AD 89 C0          6               LDA  $C089      ;RDROM
1006: A2 00             7               LDX  #0
1008: A9 FF             8               LDA  #$FF
100A: 9D 00 D0          9   BLANK1      STA  $D000,X
100D: E8                10              INX
100E: E0 20             11              CPX  #32
1010: D0 F8             12              BNE  BLANK1
                        13  *
1012: AD 81 C0          14              LDA  $C081      ;WRBK2
1015: AD 81 C0          15              LDA  $C081      ;RDROM
1018: A2 00             16              LDX  #0
101A: A9 FF             17              LDA  #$FF
101C: 9D 00 D0          18  BLANK2      STA  $D000,X
101F: E8                19              INX
1020: E0 20             20              CPX  #32
1022: D0 F8             21              BNE  BLANK2
                        22  *
                        23  *-----------------------------
                        24  *
1024: A2 FF             25              LDX  #$FF
                        26  *
1026: AD 8B C0          27              LDA  $C08B      ;WRBK1
1029: AD 8B C0          28              LDA  $C08B      ;RDBK1
102C: 20 08 11          29              JSR  STATUS
102F: AD 8A C0          30              LDA  $C08A      ;ROM1
1032: 20 08 11          31              JSR  STATUS
1035: AD 8B C0          32              LDA  $C08B      ;WRBK1
1038: AD 8B C0          33              LDA  $C08B      ;RDBK1
103B: 20 08 11          34              JSR  STATUS
103E: AD 82 C0          35              LDA  $C082      ;ROM2
1041: 20 08 11          36              JSR  STATUS
                        37  *
1044: AD 83 C0          38              LDA  $C083      ;WRBK2
1047: AD 83 C0          39              LDA  $C083      ;RDBK2
104A: 20 08 11          40              JSR  STATUS
104D: AD 82 C0          41              LDA  $C082      ;ROM2
1050: 20 08 11          42              JSR  STATUS
1053: AD 83 C0          43              LDA  $C083      ;WRBK2
1056: AD 83 C0          44              LDA  $C083      ;RDBK2
1059: 20 08 11          45              JSR  STATUS
105C: AD 8A C0          46              LDA  $C08A      ;ROM1
105F: 20 08 11          47              JSR  STATUS
                        48  *
                        49  *-----------------------------
                        50  *
1062: AD 8B C0          51              LDA  $C08B      ;WRBK1
1065: AD 8B C0          52              LDA  $C08B      ;RDBK1
1068: 20 08 11          53              JSR  STATUS
```

```
106B: AD 88 C0  54              LDA  $C088       ;RDBK1
106E: 20 08 11  55              JSR  STATUS
1071: AD 8B C0  56              LDA  $C08B       ;WRBK1
1074: AD 8B C0  57              LDA  $C08B       ;RDBK1
1077: 20 08 11  58              JSR  STATUS
107A: AD 80 C0  59              LDA  $C080       ;RDBK2
107D: 20 08 11  60              JSR  STATUS
                61  *
1080: AD 83 C0  62              LDA  $C083       ;WRBK2
1083: AD 83 C0  63              LDA  $C083       ;RDBK2
1086: 20 08 11  64              JSR  STATUS
1089: AD 80 C0  65              LDA  $C080       ;RDBK2
108C: 20 08 11  66              JSR  STATUS
108F: AD 83 C0  67              LDA  $C083       ;WRBK2
1092: AD 83 C0  68              LDA  $C083       ;RDBK2
1095: 20 08 11  69              JSR  STATUS
1098: AD 88 C0  70              LDA  $C088       ;RDBK1
109B: 20 08 11  71              JSR  STATUS
                72  *
                73  *-----------------------------
                74  *
109E: AD 88 C0  75              LDA  $C088       ;RDBK1
10A1: A2 1F     76              LDX  #31
10A3: BD 00 D0  77  MOVE1       LDA  $D000,X
10A6: 9D 40 11  78              STA  BK1SAVE,X
10A9: CA        79              DEX
10AA: 10 F7     80              BPL  MOVE1
                81  *
10AC: AD 80 C0  82              LDA  $C080       ;RDBK2
10AF: A2 1F     83              LDX  #31
10B1: BD 00 D0  84  MOVE2       LDA  $D000,X
10B4: 9D 60 11  85              STA  BK2SAVE,X
10B7: CA        86              DEX
10B8: 10 F7     87              BPL  MOVE2
                88  *
10BA: AD 81 C0  89              LDA  $C081       ;WRBK2
10BD: AD 81 C0  90              LDA  $C081       ;RDROM
10C0: A2 00     91              LDX  #0
10C2: BD 20 11  92  DISP0       LDA  STATSAVE,X
10C5: 4A        93              LSR
10C6: 4A        94              LSR
10C7: 4A        95              LSR
10C8: 4A        96              LSR
10C9: 20 E3 FD  97              JSR  $FDE3
10CC: A9 A0     98              LDA  #$A0
10CE: 20 ED FD  99              JSR  $FDED
10D1: E8        100             INX
10D2: 8A        101             TXA
10D3: 0A        102             ASL
10D4: 0A        103             ASL
10D5: 0A        104             ASL
10D6: D0 05     105             BNE  DISP1
10D8: A9 8D     106             LDA  #$8D
```

```
10DA: 20 ED FD  107             JSR  $FDED
10DD: E0 20     108  DISP1     CPX  #32
10DF: D0 E1     109             BNE  DISP0
10E1: A9 8D     110             LDA  #$8D
10E3: 20 ED FD  111             JSR  $FDED
                112  *
10E6: A2 00     113             LDX  #0
10E8: BD 40 11  114  DISP2     LDA  BK1SAVE,X
10EB: 4A        115             LSR
10EC: 4A        116             LSR
10ED: 4A        117             LSR
10EE: 4A        118             LSR
10EF: 20 E3 FD  119             JSR  $FDE3
10F2: A9 A0     120             LDA  #$A0
10F4: 20 ED FD  121             JSR  $FDED
10F7: E8        122             INX
10F8: 8A        123             TXA
10F9: 0A        124             ASL
10FA: 0A        125             ASL
10FB: 0A        126             ASL
10FC: D0 05     127             BNE  DISP3
10FE: A9 8D     128             LDA  #$8D
1100: 20 ED FD  129             JSR  $FDED
1103: E0 40     130  DISP3     CPX  #64
1105: D0 E1     131             BNE  DISP2
1107: 60        132             RTS
                133  *
1108: E8        134  STATUS    INX
1109: AD 11 C0  135             LDA  $C011      ;BK1?
110C: 9D 20 11  136             STA  STATSAVE,X
110F: 9D 00 D0  137             STA  $D000,X
1112: E8        138             INX
1113: AD 12 C0  139             LDA  $C012      ;ROM?
1116: 9D 20 11  140             STA  STATSAVE,X
1119: 9D 00 D0  141             STA  $D000,X
111C: 60        142             RTS
111D: AE AE AE  143             ASC  "..."
                144  *
                145  STATSAVE DS   32
                146  BK1SAVE  DS   32
                147  BK2SAVE  DS   32
1180: 00        148             HEX  00

--End assembly-

385 bytes

Errors: 0
```

3.1.8. 64K-Karte-Softswitches

Der mit einer 64K-Karte ausgestattete Apple IIe verfügt über die „unteren" Main 64K (= Motherboard) und die „oberen" Auxiliary 64K (Karte selbst), die sich in folgende Softswitch-Bereiche gliedern:

a) ZP-LC-Bereich: $0000-$01FF (ZP) + $D000-$FFFF (LC)

Dies ist die Zero-Page und der Stack sowie zweimal $D000-$DFFF und einmal $E000-$FFFF. Die entsprechenden Softswitches bewirken sowohl Lesen wie auch Schreiben, und zwar entweder der entsprechenden „unteren" oder „oberen" Bereiche.

b) Mittlerer 47.5K-Bereich: $0200-$BFFF (47.5K)

Dies ist der mittlere Hauptbereich. Die entsprechenden Softswitches bewirken entweder das Lesen oder das Schreiben, und zwar sowohl „unten" wie auch „oben", d.h. Mischung möglich.

```
STA   $C008:    Read-Write Main ZP („unten")
STA   $C009:    Read-Write Aux. ZP („oben")

LDA   $C016
BPL   Main ZP R/W enabled
BMI   Aux. ZP R/W enabled

STA   $C002:    Read Main 47.5K („unten")
STA   $C003:    Read Aux. 47.5K („oben")
STA   $C004:    Write Main 47.5K („unten")
STA   $C005:    Write Aux. 47.5K („oben")

LDA   $C013
BPL   Main 47.5K read-enabled
BMI   Aux. 47.5K read-enabled

LDA   $C014
BPL   Main 47.5K write-enabled
BMI   Aux. 47.5K write-enabled
```

In Verbindung mit der 64K-Karte-Speicherverwaltung sind noch zwei weitere Softswitches zu beachten:

```
STA   $C000:    Disable 80 Column Store
STA   $C001:    Enable 80 Column Store
```

Nach STA $C001 ist der Bildschirmbereich $0400-$07FF der 64K-Karte für die 47.5K-Softswitches „tabu". Will man z.B. Daten von den unteren 64K in den Bereich $0400-$07FF der oberen 64K übertragen, dann sind zuvor zwei Softswitches erforderlich:

```
STA   $C000     ;Bereich $0400-$07FF zugänglich machen
STA   $C004     ;Gesamtbeeich $0200-$BFFF schreibfähig machen
```

3.1.9. Grundregel für Memory Management Utilities

Speicherverwaltungsprogramme dienen der Übertragung von Speicherinhalten aus einem Softswitchbereich (= Ausgangsbereich) in einen anderen (= Zielbereich) mit Hilfe eines MMU-Programms (= Memory Management Utility), das sich im Ausgangsbereich, Zielbereich oder in einem dritten Softswitchbereich befinden kann (bzw. muß). Probleme ergeben sich im wesentlichen dadurch, daß Ausgangs- und Zielbereich in denselben Adreßbereich „gemappt" sind.

Grundregel: Eine MMU muß sich immer selbst lesen können, weil der 6502 sie sonst nicht als Programm abarbeiten kann. Daraus folgt:

a) Bei gleichem Adreßbereich darf sich die MMU im Ausgangsbereich befinden, wenn der Zielbereich schreibfähig und der Ausgangsbereich gleichzeitig lesefähig gemacht werden können. Dies gilt z.B. für den 47.5K-Bereich. Sind z.B. Daten von den „unteren" 47.5K in die „oberen" 47.5K zu übertragen, so kann sich die MMU in den „unteren" 47.5K befinden. Ferner kann sich die MMU in den „oberen" 47.5K befinden, wenn Daten von „oben" nach „unten" zu übertragen sind.
b) Bei gleichem Adreßbereich darf sich die MMU niemals im Zielbereich befinden, auch wenn getrenntes Read/Write für Ausgangs- und Zielbereich möglich ist. Wenn die MMU aus programmtechnischen Gründen nicht im Ausgangsbereich sein kann, dann muß sie sich in einem dritten Softswitchbereich befinden. Nehmen wir an, die MMU befände sich in den „unteren" 47.5K

und wir wollten Daten von „oben" nach „unten" übertragen. Dazu müssen wir die „oberen" 47.5K lesefähig machen, womit sich unsere MMU in den „unteren" 47.5K als Programm selbst nicht mehr lesen kann! Fazit: Eine MMU, die Daten von den „oberen" 47.5K in die „unteren" 47.5K übertragen soll, muß sich entweder in den „oberen" 47.5K oder in dem ZP-LC-Bereich $0000-$01FF bzw. $D000-$FFFF befinden.

LC-Beispiele:

Wenn wir den Inhalt des F8-Monitors $F800-$FFFF vom ROM in die Bank 2 $F800-$FFFF kopieren wollen, darf sich die MMU weder außerhalb von $F800-$FFFF in der Bank 2 noch im Bereich $D000-$DFFF der Bank 1 befinden, da sich die MMU in beiden Fällen, sobald das ROM read-enabled ist, selbst nicht mehr lesen könnte.

Demgegenüber wäre es möglich, Daten vom Bereich $D000-$DFFF Bank 1 in denselben Adreßbereich $D000-$DFFF der Bank 2 mit Hilfe einer MMU zu verschieben, die sich im Bereich $E000-$FFFF der LC befindet. Allerdings wäre für jedes zu übertragende Byte ein Bank-Switching erforderlich: LDA $C08B LDA $C08B, um das Byte von Bank 1 zu lesen, und LDA $C083 LDA $C083, um das Byte auf Bank 2 zu schreiben. In beiden Fällen wäre der Bereich $E000-$FFFF stets lesefähig.

Wenn ein Programm, das sich in einer beliebigen Bank der LC befindet, F8-ROM-Routinen benutzen will, muß sie dies tun, indem sie vorübergehend in den unteren $0200-$BFFF Main 47.5 K-Bereich oder theoretisch auch in den oberen Stackbereich $0100-$0110 springt.

LC-Beispielprogramm für Bank 2

```
COUT    EQU   $FDED

D000:   LDX   #<COUT   ;$ED
        STX   $0306+1  ;LL nach JSR bei $0307
        LDX   #>COUT   ;$FD
        STX   $0306+2  ;HH nach JSR bei $0308
        LDA   #„A"     ;Buchstabe „A" anzeigen
        JSR   $0300    ;nach „unten" springen
        ...
```

```
0300:   LDX   $C081      ;ROM lesefähig machen
        LDX   $C081
0306:   JSR   $FDED      ;COUT LLHH wird gepokt!
        LDX   $C083      ;LC Bank 2 lesefähig machen
        LDX   $C083
        RTS   ;zurück zur LC
```

Dieses hier vorgestellte, sich selbst pokende JSR-Verfahren hat den Vorteil, daß in den unteren 48K kein kostbarer Platz verschwendet wird. Zum Vergleich wurde z.B. in „Nibble", Heft 2/1984 bei dem „Applesoft Expander"-Programm in den unteren 48K ein fast 500 Bytes langes ROM-JMP-Programm installiert, obgleich in der Bank 1 der LC noch genügend Platz war.

```
                        1               ORG  $300
                        2   *
                        3   * LC.LOESCHER
                        4   * ==========
                        5   *
                        6   IND         EQU  $CE
                        7   LC          EQU  $D000
                        8   *
                        9   * Bank 1 lesen und schreiben
                       10   *
0300: AD 8B C0         11   BANK1       LDA  $C08B       ;RDBK1
0303: AD 8B C0         12               LDA  $C08B       ;WRBK1
0306: 20 19 03         13               JSR  LOOP1
                       14   *
                       15   * Bank 2 lesen und schreiben
                       16   *
0309: AD 83 C0         17   BANK2       LDA  $C083       ;RDBK2
030C: AD 83 C0         18               LDA  $C083       ;WRBK2
030F: 20 19 03         19               JSR  LOOP1
                       20   *
                       21   * ROM lesen, Bank 2 schreiben
                       22   *
0312: AD 81 C0         23   RDROM       LDA  $C081       ;RDROM
0315: AD 81 C0         24               LDA  $C081       ;WRBK2
0318: 60               25               RTS
                       26   *
                       27   * $D000-$FFFF löschen
                       28   *
0319: A9 00            29   LOOP1       LDA  #<LC
031B: 85 CE            30               STA  IND
031D: A9 D0            31               LDA  #>LC
031F: 85 CF            32               STA  IND+1
0321: A0 00            33               LDY  #0
0323: A8               34               TAY
0324: 91 CE            35   LOOP2       STA  (IND),Y
0326: C8               36               INY
0327: D0 FB            37               BNE  LOOP2
0329: E6 CF            38               INC  IND+1
032B: D0 F7            39               BNE  LOOP2       ;FFFF
032D: 60               40               RTS
                       41   *
                       42   * Wichtige LC-Softswitches
                       43   * ------------------------
                       44   *
                       45   * Je zweimal mit LDA Adresse
                       46   * aktivieren
                       47   *
                       48   * LDA $C081 LDA $C081 RDROM WRBK2
                       49   * LDA $C083 LDA $C083 RDBK2 WRBK2
                       50   * LDA $C089 LDA $C089 RDROM WRBK1
                       51   * LDA $C08B LDA $C08B RDBK1 WRBK1
                       52   *
```

```
                        1               ORG   $300
                        2    *
                        3    * LC.READER
                        4    * ==========
                        5    *
                        6    * Bank 1 nach $1000-$1FFF
                        7    * Bank 2 nach $2000-$4FFF
                        8    *
                        9    LC         EQU   $CE
                       10    RAM        EQU   $FE
                       11    *
                       12    HOME       EQU   $FC58
                       13    COUT       EQU   $FDED
                       14    DOSWARM    EQU   $03D0
                       15    *
                       16    * CALL 768
                       17    *
0300: 20 58 FC         18    LCREADER   JSR   HOME
0303: A0 00            19               LDY   #0
0305: B9 55 03         20    DISPLAY    LDA   STRING,Y
0308: F0 06            21               BEQ   MOVER
030A: 20 ED FD         22               JSR   COUT
030D: C8               23               INY
030E: D0 F5            24               BNE   DISPLAY
                       25    *
0310: A2 00            26    MOVER      LDX   #$00
0312: 86 FE            27               STX   RAM
0314: A9 10            28               LDA   #$10          ;$1000
0316: 85 FF            29               STA   RAM+1
0318: 86 CE            30               STX   LC
031A: A2 D0            31               LDX   #$D0          ;$D000
031C: 86 CF            32               STX   LC+1
                       33    *
031E: AD 8B C0         34               LDA   $C08B         ;RDBK1
0321: AD 8B C0         35               LDA   $C08B         ;WRBK1
0324: A0 00            36               LDY   #0
0326: B1 CE            37    LOOP1      LDA   (LC),Y
0328: 91 FE            38               STA   (RAM),Y
032A: C8               39               INY
032B: D0 F9            40               BNE   LOOP1
032D: E6 FF            41               INC   RAM+1
032F: E6 CF            42               INC   LC+1
0331: A5 CF            43               LDA   LC+1
0333: C9 E0            44               CMP   #$E0
0335: D0 EF            45               BNE   LOOP1         ;>$DFFF
                       46    *
0337: 86 CF            47               STX   LC+1
0339: AD 83 C0         48               LDA   $C083         ;RDBK2
033C: AD 83 C0         49               LDA   $C083         ;WRBK2
033F: B1 CE            50    LOOP2      LDA   (LC),Y
0341: 91 FE            51               STA   (RAM),Y
0343: C8               52               INY
0344: D0 F9            53               BNE   LOOP2
```

```
0346: E6 FF      54              INC  RAM+1
0348: E6 CF      55              INC  LC+1
034A: D0 F3      56              BNE  LOOP2       ;>$FFFF
                 57   *
034C: AD 81 C0   58              LDA  $C081       ;RDROM
034F: AD 81 C0   59              LDA  $C081       ;WRBK2
0352: 4C D0 03   60              JMP  DOSWARM
                 61   *
0355: C4 B0 B0   62   STRING     ASC  „D000-DFFF BK1:"
0358: B0 AD C4 C6 C6 C6 A0 C2
0360: CB B1 BA
0363: B1 B0 B0   63              ASC  „1000-1FFF"
0366: B0 AD B1 C6 C6 C6
036C: 8D         64              HEX  8D
036D: C4 B0 B0   65              ASC  „D000-FFFF BK2:"
0370: B0 AD C6 C6 C6 C6 A0 C2
0378: CB B2 BA
037B: B2 B0 B0   66              ASC  „2000-4FFF"
037E: B0 AD B4 C6 C6 C6
0384: 8D         67              HEX  8D
0385: 00         68              HEX  00
                 69   *
                 70   *
```

3.2.3. ROM.KOPIE

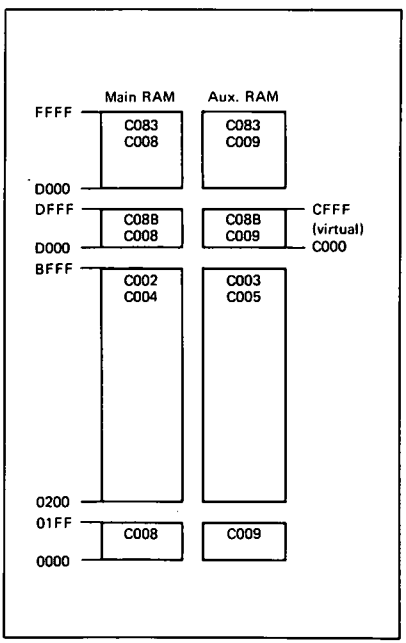
```
                 72   * ROM.KOPIE
                 73   * ═══════
                 74   *
                 75   * Diese Utility kopiert den
                 76   * ROM-Bereich $D000-$FFFF in
                 77   * die Language Card Bank 2
                 78   * Danach kann man Applesoft
                 79   * und Monitor im RAM patchen.
                 80   *
                 81   IND       EQU  $CE         ;-$CF
                 82   *
                 83   * CALL 902
                 84   *
0386: AD 81 C0   85   ROMCOPY1  LDA  $C081       ;RDROM
0389: AD 81 C0   86              LDA  $C081       ;WRRAM
                 87   *
038C: A0 00      88              LDY  #0
038E: 84 CE      89              STY  IND
0390: A9 D0      90              LDA  #$D0        ;$D000
0392: 85 CF      91              STA  IND+1
0394: B1 CE      92   ROMCOPY2  LDA  (IND),Y     ;ROM
0396: 91 CE      93              STA  (IND),Y     ;RAM
0398: C8         94              INY
0399: D0 F9      95              BNE  ROMCOPY2
039B: E6 CF      96              INC  IND+1
039D: D0 F5      97              BNE  ROMCOPY2    ;$FFFF+1
039F: 60         98              RTS
```

```
                    1               ORG  $300
                    2   *
                    3   * LC.MOVER
                    4   * ========
                    5   *
                    6   * Diese Routine movt einen
                    7   * beliebigen Bereich Main
                    8   * $0400-$BFFF in die LC
                    9   * $C000-$FFFF, wobei immer nur
                   10   * ganze Pages = 256 Bytes über-
                   11   * tragen werden.
                   12   *
                   13   * High-Bytes von Main-Beginn,
                   14   * Main-Ende und LC-Beginn
                   15   * sowie Richtung poken und dann
                   16   * mit CALL 768 starten.
                   17   *
                   18   * Anmerkung 1: MEND = Main-Ende
                   19   * versteht sich inklusive, z.B.
                   20   * MBEG: 04, MEND: 04 überträgt
                   21   * $0400-$04FF.
                   22   *
                   23   * Anmerkung 2: Bank 1 der LC
                   24   * ist der Benutzerfreundlichkeit
                   25   * halber als virtueller Bereich
                   26   * $C000-CFFF definiert.
                   27   *
0300: 4C 0A 03     28           JMP  CHECK
                   29   *
                   30   * Main-Bereich  = $0400-$BF00
                   31   *
0303: 20           32   MBEG     HEX  20          ;04-BF
0304: 40           33   MEND     HEX  40          ;04-BF
                   34   *
                   35   * Language Card = $C000-FFFF
                   36   *
                   37   * $C000-CFFF: virtuell    Bank 1
                   38   *             $D000-DFFF  C08B
                   39   *
                   40   * $D000-FFFF: physisch    Bank 2
                   41   *             $D000-FFFF  C083
                   42   *
0305: D0           43   LBEG     HEX  D0          ;C0-FF
                   44   *
                   45   * Richtungs-Flag:
                   46   *
                   47   * 0 = Main nach LC
                   48   * 1 = LC nach Main
                   49   *
0306: 00           50   FLAG     HEX  00          ;00-01
                   51   *
                   52   MAIN     EQU  $FC         ;-$FD
                   53   LC       EQU  $FE         ;-$FF
```

```
                        54   PRERR    EQU  $FF2D       ;„ERR"
                        55   *
0307: 4C 2D FF          56   ERROR    JMP  PRERR
                        57   *
                        58   * Parameter okay?
                        59   *
030A: AD 06 03          60   CHECK    LDA  FLAG
030D: C9 02             61            CMP  #2
030F: B0 F6             62            BCS  ERROR
                        63   *
0311: AD 03 03          64            LDA  MBEG
0314: C9 04             65            CMP  #$04       ;<$0400
0316: 90 EF             66            BCC  ERROR
0318: C9 C0             67            CMP  #$C0       ;>=$C000
031A: B0 EB             68            BCS  ERROR
                        69   *
031C: AD 04 03          70            LDA  MEND
031F: C9 04             71            CMP  #$04       ;<$0400
0321: 90 E4             72            BCC  ERROR
0323: C9 C0             73            CMP  #$C0       ;>=$C000
0325: B0 E0             74            BCS  ERROR
                        75   *
0327: AD 05 03          76            LDA  LBEG
032A: C9 C0             77            CMP  #$C0       ;<$C000
032C: 90 D9             78            BCC  ERROR
                        79   *
                        80   * LBEG + (MEND-MBEG) < = $FFFF?
                        81   *
032E: 38                82            SEC
032F: AD 04 03          83            LDA  MEND
0332: ED 03 03          84            SBC  MBEG
0335: 90 D0             85            BCC  ERROR      ;Underflow
                        86   *
0337: 18                87            CLC
0338: 6D 05 03          88            ADC  LBEG
033B: B0 CA             89            BCS  ERROR      ;Overflow
                        90   *
                        91   * Eigentliche Move-Routine
                        92   *
033D: A0 00             93   INIT     LDY  #0
033F: 84 FC             94            STY  MAIN
0341: 84 FE             95            STY  LC
0343: AD 03 03          96            LDA  MBEG
0346: 85 FD             97            STA  MAIN+1
                        98   *
                        99   * Nur Bank 2 ?
                        100  *
0348: AD 83 C0          101  BANK2?   LDA  $C083      ;RDBK2
034B: AD 83 C0          102           LDA  $C083      ;WRBK2
034E: A2 02             103           LDX  #2         ;Bk2-Flag
0350: AD 05 03          104           LDA  LBEG
0353: C9 D0             105           CMP  #$D0
0355: B0 0F             106           BCS  RICHTUNG   ;>=$D000
```

```
                    107  *
                    108  * Nein,. erst Bank 1
                    109  *
0357: AD 8B C0      110          LDA  $C08B       ;RDBK1
035A: AD 8B C0      111          LDA  $C08B       ;WRBK1
035D: A2 01         112          LDX  #1          ;Bk2-Flag
035F: AD 05 03      113          LDA  LBEG
0362: 09 10         114          ORA  #%00010000  ;CX>DX
0364: 85 FF         115          STA  LC+1
                    116  *
0366: 85 FF         117  RICHTUNG STA  LC+1
0368: AD 06 03      118          LDA  FLAG
036B: D0 0E         119          BNE  LCTOMAIN
                    120  *
036D: B1 FC         121  MAINTOLC LDA  (MAIN),Y   ;MAIN>LC
036F: 91 FE         122          STA  (LC),Y
0371: C8            123          INY
0372: D0 F9         124          BNE  MAINTOLC
0374: 20 8E 03      125          JSR  BANKER1
0377: B0 F4         126          BCS  MAINTOLC    ;weiter
0379: 90 0C         127          BCC  RDROM
                    128  *
037B: B1 FE         129  LCTOMAIN LDA  (LC),Y     ;LC>MAIN
037D: 91 FC         130          STA  (MAIN),Y
037F: C8            131          INY
0380: D0 F9         132          BNE  LCTOMAIN
0382: 20 8E 03      133          JSR  BANKER1
0385: B0 F4         134          BCS  LCTOMAIN    ;weiter
                    135  *
0387: AD 81 C0      136  RDROM    LDA  $C081       ;RDROM
038A: AD 81 C0      137          LDA  $C081       ;WRBK2
038D: 60            138          RTS              ;Exit
                    139  *
038E: E6 FF         140  BANKER1  INC  LC+1
0390: E0 02         141          CPX  #2
0392: F0 12         142          BEQ  BANKER2
0394: A5 FF         143          LDA  LC+1
0396: C9 E0         144          CMP  #$E0        ;$E000
0398: D0 0C         145          BNE  BANKER2
039A: A9 D0         146          LDA  #$D0        ;$D000
039C: 85 FF         147          STA  LC+1
039E: AD 83 C0      148          LDA  $C083       ;RDBK2
03A1: AD 83 C0      149          LDA  $C083       ;WRBK2
03A4: A2 02         150          LDX  #2          ;Bk2-Flag
03A6: E6 FD         151  BANKER2  INC  MAIN+1
                    152  *
                    153  * MEND > = MAIN+1  - BCS ?
                    154  *
03A8: AD 04 03      155          LDA  MEND
03AB: C5 FD         156          CMP  MAIN+1
03AD: 60            157          RTS
```

```
10  PRINT  CHR$ (4)„BLOAD LC.MOVER"
15  TEXT : HOME
20  POKE 771,4: REM MAIN-BEGINN $0400
25  POKE 772,7: REM MAIN-ENDE $07FF
30  POKE 774,0: REM MAIN TO LC
35 L = 192: REM $C000
40  FOR X = 0 TO 15
45  VTAB 10: HTAB 10: CALL  - 868
50  PRINT „BILDSCHIRM-NR. ";X
55  POKE 773,L + X * 4: REM LC-BEGINN
60  CALL 768: REM MOVE
65  NEXT X
70  POKE 774,1: REM LC TO MAIN
75  FOR X = 0 TO 15
80  POKE 773,L + X * 4: REM LC-BEGINN
85  CALL 768: REM MOVE
90  NEXT X
```



Appel IIe mit 128 K

3.2.5. COPYROM

```
100  HOME : PRINT „COPYROM $F800-$FFFF in Language Card $F800-$FFFF"
110  PRINT : PRINT „Nur für Apple IIe mit DOS 3.3"
120  POKE 49159,0: CALL 53112: POKE 49158,0: PR# 0: IN# 0: CALL 1002
```

```
                        1           ORG  $0300
                        2   *
                        3   * BRUTAL.CLEAR
                        4   * ==========
                        5   *
                        6   * Löscht $0100-$BFFF rückwärts
                        7   * und zerstört dabei DOS!
                        8   * Für diejenigen, die ihre
                        9   * Programme schützen wollen.
                       10   *
0300: A2 00            11   MOVER1   LDX  #0
0302: BD 11 03         12   MOVER2   LDA  START,X
0305: C9 EA            13            CMP  #$EA        ;MARKER
0307: F0 05            14            BEQ  MOVER3
0309: 95 00            15            STA  0,X         ;ZERO
030B: E8               16            INX
030C: D0 F4            17            BNE  MOVER2
030E: 4C 00 00         18   MOVER3   JMP  0
                       19   *
                       20   * Beginnt nach Move ab $0000
                       21   *
0311: A2 00            22   START    LDX  #0          ;$0000
0313: A0 18            23            LDY  #MARKER-COPYR-1
                       24   *
                       25   * Lösch-Loop $BFFF-$0100
                       26   *
0315: 88               27   ST03     DEY
0316: 10 02            28            BPL  ST02
0318: A0 18            29            LDY  #MARKER-COPYR-1
031A: B9 1D 00         30   ST02     LDA  $001D,Y     ;COPYR
                       31   *
                       32   * ST01 beginnt ab $000C = 12
                       33   *
031D: 8D FF BF         34   ST01     STA  $BFFF       ;DUMMY
0320: C6 0D            35            DEC  13          ;ST01+1
0322: D0 F1            36            BNE  ST03
0324: C6 0E            37            DEC  14          ;ST01+2
0326: E4 0E            38            CPX  14          ;ST01+2
0328: D0 EB            39            BNE  ST03
                       40   *
                       41   * Endlos-Schleife, bis Reset
                       42   * gedrückt wird.
                       43   *
032A: A9 00            44   ENDLOS   LDA  #0
032C: F0 FC            45            BEQ  ENDLOS
                       46   *
                       47   * Der gesamte Speicher inkl.
                       48   * Bildschirm erhält dieses Muster
                       49   *
032E: 02 09 14         50   COPYR    INV  „BITTE ORIGINAL BENUTZEN! "
0331: 14 05 20 0F 12 09 07 09
0339: 0E 01 0C 20 02 05 0E 15
0341: 14 1A 05 0E 21 20
0347: EA               51   MARKER   HEX  EA
```

```
                        1               ORG  $220
                        2    *
                        3    * TEST.CLEAR
                        4    * ==========
                        5    *
                        6    * Löscht $0300-$03CF, $0800-$95FF
                        7    * und Language Card $D000-$FFFF
                        8    * der „unteren" 64K sowie
                        9    * die gesamten „oberen" 64K
                       10    * der Apple IIe 64K-Karte
                       11    *
                       12    * Nachher Ctrl-Reset tippen,
                       13    * um Softswitches hardwaremäßig
                       14    * zu normalisieren.
                       15    *
                       16    * DOS 3.3 bleibt unversehrt.
                       17    * Nicht für ProDOS gedacht,
                       18    * da ProDOS in LC liegt!
                       19    *
                       20    * U.Stiehl/1984
                       21    *
                       22    IND       EQU  $CE
                       23    HOME      EQU  $FC58
                       24    PRINT     EQU  $FDED
                       25    DOSCOLD   EQU  $03D3
                       26    *
                       27    * 80-Zeichenkarte abstellen
                       28    *
0220: A9 95            29               LDA  #$95        ;CTRL-U
0222: 20 ED FD         30               JSR  PRINT       ;PR#0
                       31    *
0225: A9 00            32               LDA  #$00        ;CLEAR
0227: 85 CE            33               STA  IND         ;LOWBYT
                       34    *
0229: 8C 00 C0         35               STY  $C000       ;80.OFF
022C: 8C 02 C0         36               STY  $C002       ;MAINRD
022F: 8C 04 C0         37               STY  $C004       ;MAINWR
0232: 8C 08 C0         38               STY  $C008       ;MAINZP
                       39    *
                       40    * $0300-03CF Main 64K
                       41    *
0235: A8               42               TAY
0236: 99 00 03         43    L1         STA  $300,Y
0239: C8               44               INY
023A: C0 D0            45               CPY  #$D0
023C: D0 F8            46               BNE  L1
                       47    *
                       48    * $0800-9600 Main 64K
                       49    *
023E: A0 08            50               LDY  #$08
0240: 84 CF            51               STY  IND+1
0242: A8               52               TAY
0243: 91 CE            53    L2         STA  (IND),Y
```

```
0245: C8          54            INY
0246: D0 FB       55            BNE  L2
0248: E6 CF       56            INC  IND+1
024A: A6 CF       57            LDX  IND+1
024C: E0 96       58            CPX  #$96
024E: D0 F3       59            BNE  L2
                  60   *
                  61   * $D000-FFFF Main Bank 2
                  62   *
0250: AC 83 C0    63            LDY  $C083
0253: AC 83 C0    64            LDY  $C083      ;MAINBK2
0256: A0 D0       65            LDY  #$D0
0258: 84 CF       66            STY  IND+1
025A: A8          67            TAY
025B: 91 CE       68   L3       STA  (IND),Y
025D: C8          69            INY
025E: D0 FB       70            BNE  L3
0260: E6 CF       71            INC  IND+1
0262: D0 F7       72            BNE  L3
                  73   *
                  74   * $D000-DFFF Main Bank 1
                  75   *
0264: AC 8B C0    76            LDY  $C08B
0267: AC 8B C0    77            LDY  $C08B      ;MAINBK1
026A: A0 D0       78            LDY  #$D0
026C: 84 CF       79            STY  IND+1
026E: A8          80            TAY
026F: 91 CE       81   L4       STA  (IND),Y
0271: C8          82            INY
0272: D0 FB       83            BNE  L4
0274: E6 CF       84            INC  IND+1
0276: A6 CF       85            LDX  IND+1
0278: E0 E0       86            CPX  #$E0
027A: D0 F3       87            BNE  L4
                  88   *
                  89   * $0200-BFFF Auxiliary 64K
                  90   *
027C: A0 02       91            LDY  #$02
027E: 84 CF       92            STY  IND+1
0280: 8C 05 C0    93            STY  $C005      ;AUXWR
0283: A8          94            TAY
0284: 91 CE       95   L5       STA  (IND),Y
0286: C8          96            INY
0287: D0 FB       97            BNE  L5
0289: E6 CF       98            INC  IND+1
028B: A6 CF       99            LDX  IND+1
028D: E0 C0      100            CPX  #$C0
028F: D0 F3      101            BNE  L5
0291: 8C 04 C0   102            STY  $C004      ;MAINWR
                 103   *
                 104   * $0000-00FF Auxiliary 64K
                 105   *
0294: 8C 09 C0   106            STY  $C009      ;AUXZP
0297: A8         107            TAY
```

```
0298: 99 00 00  108  L6       STA: $0000,Y
029B: C8        109           INY
029C: D0 FA     110           BNE  L6
                111  *
                112  * $0100-01FF Auxiliary 64K
                113  *
029E: 99 00 01  114  L7       STA  $0100,Y
02A1: C8        115           INY
02A2: D0 FA     116           BNE  L7
                117  *
                118  * $D000-FFFF Auxiliary Bank 2
                119  *
02A4: AC 83 C0  120           LDY  $C083
02A7: AC 83 C0  121           LDY  $C083       ;AUXBK2
02AA: 8D B4 02  122           STA  L8+1
02AD: A0 D0     123           LDY  #$D0
02AF: 8C B5 02  124           STY  L8+2
02B2: A8        125           TAY
02B3: 99 00 D0  126  L8       STA  $D000,Y
02B6: C8        127           INY
02B7: D0 FA     128           BNE  L8
02B9: EE B5 02  129           INC  L8+2
02BC: D0 F5     130           BNE  L8
                131  *
                132  * $D000-DFFF Auxiliary Bank 1
                133  *
02BE: AC 8B C0  134           LDY  $C08B
02C1: AC 8B C0  135           LDY  $C08B       ;AUXBK1
02C4: 8D CE 02  136           STA  L9+1
02C7: A0 D0     137           LDY  #$D0
02C9: 8C CF 02  138           STY  L9+2
02CC: A8        139           TAY
02CD: 99 00 D0  140  L9       STA  $D000,Y
02D0: C8        141           INY
02D1: D0 FA     142           BNE  L9
02D3: EE CF 02  143           INC  L9+2
02D6: AE CF 02  144           LDX  L9+2
02D9: E0 E0     145           CPX  #$E0
02DB: D0 F0     146           BNE  L9
02DD: 8C 08 C0  147           STY  $C008       ;MAINZP
02E0: AC 81 C0  148           LDY  $C081
02E3: AC 81 C0  149           LDY  $C081       ;ROMBK2
                150  *
                151  * Print „CLR"
                152  *
02E6: 20 58 FC  153  CLR      JSR  HOME
02E9: A9 C3     154           LDA  #„C"
02EB: 20 ED FD  155           JSR  PRINT
02EE: A9 CC     156           LDA  #„L"
02F0: 20 ED FD  157           JSR  PRINT
02F3: A9 D2     158           LDA  #„R"
02F5: 20 ED FD  159           JSR  PRINT
02F8: 4C D3 03  160           JMP  DOSCOLD
```

```
                    1               ORG  $9400      ;37888
                    2    *
                    3    * AUXMOVER
                    4    * ========
                    5    *
                    6    * für Apple IIe 64K-Karte
                    7    * (c) U. Stiehl Sept.1983
                    8    *
                    9    * Diese Routine verschiebt einen
                   10    * beliebigen Bereich der unteren
                   11    * $0200-$BFFF in einen beliebigen
                   12    * Bereich der oberen $0000-$FFFF.
                   13    * Auxmover muß sich „unten"
                   14    * im Bereich $0200-$BFFF befinden
                   15    *
                   16    * „Unten" = Motherboard 64K
                   17    * „Oben"  = Auxiliary 64K
                   18    *
                   19    * Virtuell    Physisch
                   20    * ---------   ----------------
                   21    * 0000-01FF   0000-01FF AUXZP
                   22    * 0200-BFFF   0200-BFFF AUX47.5
                   23    * C000-CFFF   D000-DFFF AUXBK1
                   24    * D000-FFFF   D000-DFFF AUXBK2
                   25    * ---------   ----------------
                   26    *
9400: 4C 0A 94     27               JMP  SAVER     ;ORIGIN
                   28    *
                   29    *======== Parameter poken ========
                   30    *
                   31    * FLAG 0 = MAIN => AUX
                   32    * FLAG 1 = AUX  => MAIN
                   33    *
9403: 00           34    FLAG     HEX  00          ;0+3
                   35    *
                   36    * Auxiliary Memory Begin Low/High
                   37    *
9404: 00           38    AUXBL    HEX  00          ;0+4
9405: 00           39    AUXBH    HEX  00          ;0+5
                   40    *
                   41    * Main Memory Begin Low/High
                   42    *
9406: 00           43    MAINBL   HEX  00          ;0+6
9407: 20           44    MAINBH   HEX  20          ;0+7
                   45    *
                   46    * Main Memory End Low/High
                   47    *
9408: FF           48    MAINEL   HEX  FF          ;0+8
9409: 3F           49    MAINEH   HEX  3F          ;0+9
                   50    *
                   51    *
                   52    MB       EQU  $CE         ;MAINB
                   53    AB       EQU  $FE         ;AUXB
```

```
                        54   STACK    EQU  $100
                        55   *
                        56   *═══════ Parameter prüfen ═══════
                        57   *
                        58   * Register und Zero-Page retten
                        59   *
940A: 8D B8 95          60   SAVER    STA  REGISTER
940D: 8E B9 95          61            STX  REGISTER+1
9410: 8C BA 95          62            STY  REGISTER+2
9413: A5 CE             63            LDA  MB
9415: 8D B4 95          64            STA  ZERO
9418: A5 CF             65            LDA  MB+1
941A: 8D B5 95          66            STA  ZERO+1
941D: A5 FE             67            LDA  AB
941F: 8D B6 95          68            STA  ZERO+2
9422: A5 FF             69            LDA  AB+1
9424: 8D B7 95          70            STA  ZERO+3
9427: 08                71            PHP                ;STATUS
9428: D8                72            CLD
                        73   *
                        74   * 80-Zeichenkarte an ?
                        75   *
9429: AD 18 C0          76            LDA  $C018         ;ON?
942C: 30 40             77            BMI  ERROR         ;YES!
                        78   *
                        79   * Flag <= 1 ?
                        80   *
942E: AD 03 94          81            LDA  FLAG
9431: C9 02             82            CMP  #$02
9433: B0 39             83            BCS  ERROR
                        84   *
                        85   * 0200 >= Main <= BFFF ?
                        86   *
9435: AD 07 94          87            LDA  MAINBH
9438: C9 02             88            CMP  #$02          ;<02
943A: 90 32             89            BCC  ERROR
943C: C9 C0             90            CMP  #$C0          ;>=C0
943E: B0 2E             91            BCS  ERROR
                        92   *
9440: AD 09 94          93            LDA  MAINEH
9443: C9 02             94            CMP  #$02          ;<02
9445: 90 27             95            BCC  ERROR
9447: C9 C0             96            CMP  #$C0          ;>=C0
9449: B0 23             97            BCS  ERROR
                        98   *
                        99   * AuxB + (MainE - MainB) < FFFF ?
                       100   *
944B: 38               101            SEC
944C: AD 08 94         102            LDA  MAINEL
944F: ED 06 94         103            SBC  MAINBL
9452: 85 CE            104            STA  MB
9454: AD 09 94         105            LDA  MAINEH
9457: ED 07 94         106            SBC  MAINBH
945A: 85 CF            107            STA  MB+1
```

```
945C: 90 10      108           BCC  ERROR       ;UNDERFL
                 109  *
945E: 18         110           CLC
945F: AD 04 94   111           LDA  AUXBL
9462: 65 CE      112           ADC  MB
9464: AD 05 94   113           LDA  AUXBH
9467: 65 CF      114           ADC  MB+1
9469: B0 03      115           BCS  ERROR       ;OVERFL
946B: 4C 76 94   116           JMP  START
946E: A9 87      117  ERROR    LDA  #$87        ;BELL
9470: 20 ED FD   118           JSR  $FDED       ;PRINT
9473: 4C 8A 95   119           JMP  RESTORER
                 120  *
                 121  *═══════ Start ═══════════════════
                 122  *
9476: A2 0A      123  START    LDX  #10
                 124  *
                 125  * Leseroutine für Aux. $0200-BFFF
                 126  * in Bereich außerhalb $0200-BFFF
                 127  * legen wegen Softswitches!
                 128  *
9478: BD A9 95   129  STACK1   LDA  STACK2,X
947B: 9D 00 01   130           STA  STACK,X     ;$0100
947E: CA         131           DEX
947F: 10 F7      132           BPL  STACK1
                 133  *
9481: AD 06 94   134           LDA  MAINBL
9484: 85 CE      135           STA  MB
9486: AD 07 94   136           LDA  MAINBH
9489: 85 CF      137           STA  MB+1
                 138  *
948B: AD 04 94   139           LDA  AUXBL
948E: 85 FE      140           STA  AB
9490: AD 05 94   141           LDA  AUXBH
9493: 85 FF      142           STA  AB+1
                 143  *
9495: A0 00      144           LDY  #0          ;Y=0
                 145  *
                 146  *═══════ Hauptschleife ═══════════
                 147  *
9497: A6 FF      148  LOOP     LDX  AB+1        ;PAGE
9499: CC 03 94   149           CPY  FLAG
949C: F0 03      150           BEQ  MAINAUX     ;0
949E: 4C 07 95   151           JMP  AUXMAIN     ;1
                 152  *
                 153  *═══════ Main nach Aux ═══════════
                 154  *
94A1: B1 CE      155  MAINAUX  LDA  (MB),Y      ;LOAD
                 156  *
94A3: E0 02      157           CPX  #$02        ;<$0200
94A5: 90 4C      158           BCC  M4
94A7: E0 C0      159           CPX  #$C0        ;<$C000
94A9: 90 3D      160           BCC  M3
94AB: E0 D0      161           CPX  #$D0        ;<$D000
```

```
94AD: 90 1A     162           BCC  M2
                163  *
                164  * D000-FFFF Bank 2
                165  *
94AF: 8E C2 94  166  M1       STX  BANK2WR+2
94B2: A6 FE     167           LDX  AB
94B4: 8E C1 94  168           STX  BANK2WR+1
94B7: AE 83 C0  169           LDX  $C083
94BA: AE 83 C0  170           LDX  $C083       ;RD/WR
94BD: 8E 09 C0  171           STX  $C009       ;AUXZP
94C0: 8D FF FF  172  BANK2WR  STA  $FFFF       ;DUMMY
94C3: 8E 08 C0  173           STX  $C008       ;MAINZP
94C6: 4C 5A 95  174           JMP  II0
                175  *
                176  * C000-CFFF => D000-DFFF Bank 1
                177  *
94C9: 8A        178  M2       TXA
94CA: 09 10     179           ORA  #%00010000 ;CX>DX
94CC: 8D E1 94  180           STA  BANK1WR+2
94CF: A6 FE     181           LDX  AB
94D1: 8E E0 94  182           STX  BANK1WR+1
94D4: B1 CE     183           LDA  (MB),Y
94D6: AE 8B C0  184           LDX  $C08B
94D9: AE 8B C0  185           LDX  $C08B       ;RD/WR
94DC: 8E 09 C0  186           STX  $C009       ;AUXZP
94DF: 8D FF FF  187  BANK1WR  STA  $FFFF       ;DUMMY
94E2: 8E 08 C0  188           STX  $C008       ;MAINZP
94E5: 4C 5A 95  189           JMP  II0
                190  *
                191  * 0200-BFFF
                192  *
94E8: 8E 05 C0  193  M3       STX  $C005       ;AUXWR
94EB: 91 FE     194           STA  (AB),Y
94ED: 8E 04 C0  195           STX  $C004       ;MAINWR
94F0: 4C 5A 95  196           JMP  II0
                197  *
                198  * 0000-01FF
                199  *
94F3: 8E 00 95  200  M4       STX  AUXZPWR+2
94F6: A6 FE     201           LDX  AB
94F8: 8E FF 94  202           STX  AUXZPWR+1
94FB: 8E 09 C0  203           STX  $C009       ;AUXZP
94FE: 8D FF FF  204  AUXZPWR  STA  $FFFF       ;DUMMY
9501: 8E 08 C0  205           STX  $C008       ;MAINZP
9504: 4C 5A 95  206           JMP  II0
                207  *
                208  *═══════ Aux nach Main ═══════
                209  *
9507: E0 02     210  AUXMAIN  CPX  #$02        ;<$0200
9509: 90 3C     211           BCC  A4
950B: E0 C0     212           CPX  #$C0        ;<$C000
950D: 90 35     213           BCC  A3
950F: E0 D0     214           CPX  #$D0        ;<$D000
9511: 90 17     215           BCC  A2
```

```
                        216  *
                        217  * D000-FFFF Bank 2
                        218  *
9513: 8E 23 95          219  A1        STX  BANK2RD+2
9516: A6 FE             220            LDX  AB
9518: 8E 22 95          221            STX  BANK2RD+1
951B: AE 80 C0          222            LDX  $C080       ;RDBK2
951E: 8E 09 C0          223            STX  $C009       ;AUXZP
9521: AD FF FF          224  BANK2RD  LDA  $FFFF       ;DUMMY
9524: 8E 08 C0          225            STX  $C008       ;MAINZP
9527: 4C 58 95          226            JMP  II
                        227  *
                        228  * C000-CFFF => D000-DFFF Bank 1
                        229  *
952A: 8A                230  A2        TXA
952B: 09 10             231            ORA  #%00010000  ;CX>DX
952D: 8D 3D 95          232            STA  BANK1RD+2
9530: A6 FE             233            LDX  AB
9532: 8E 3C 95          234            STX  BANK1RD+1
9535: AE 88 C0          235            LDX  $C088       ;RDBK1
9538: 8E 09 C0          236            STX  $C009       ;AUXZP
953B: AD FF FF          237  BANK1RD  LDA  $FFFF       ;DUMMY
953E: 8E 08 C0          238            STX  $C008       ;MAINZP
9541: 4C 58 95          239            JMP  II
                        240  *
                        241  * 0200-BFFF
                        242  *
9544: 4C 00 01          243  A3        JMP  STACK
                        244  *
                        245  * 0000-01FF
                        246  *
9547: 8E 54 95          247  A4        STX  AUXZPRD+2
954A: A6 FE             248            LDX  AB
954C: 8E 53 95          249            STX  AUXZPRD+1
954F: 8E 09 C0          250            STX  $C009       ;AUXZP
9552: AD FF FF          251  AUXZPRD  LDA  $FFFF       ;DUMMY
9555: 8E 08 C0          252            STX  $C008       ;MAINZP
                        253  *
                        254  * ═══ Nächstes Byte ═══
                        255  *
9558: 91 CE             256  II        STA  (MB),Y      ;STORE
                        257  *
955A: E6 FE             258  II0       INC  AB
955C: D0 02             259            BNE  II1
955E: E6 FF             260            INC  AB+1
9560: E6 CE             261  II1       INC  MB
9562: D0 02             262            BNE  II2
9564: E6 CF             263            INC  MB+1
9566: A5 CF             264  II2       LDA  MB+1
9568: CD 09 94          265            CMP  MAINEH
956B: 90 0B             266            BCC  II4
956D: D0 0C             267            BNE  NORMAL
956F: A5 CE             268  II3       LDA  MB
9571: CD 08 94          269            CMP  MAINEL
```

```
9574: F0 02     270           BEQ  II4
9576: B0 03     271           BCS  NORMAL
9578: 4C 97 94  272  II4      JMP  LOOP
                273  *
                274  *═══════ Normale Switches ═══════
                275  *
957B: 8E 02 C0  276  NORMAL   STX  $C002      ;MAINRD
957E: 8E 04 C0  277           STX  $C004      ;MAINWR
9581: 8E 08 C0  278           STX  $C008      ;MAINZP
9584: AE 81 C0  279           LDX  $C081      ;WRBK2
9587: AE 81 C0  280           LDX  $C081      ;RDROM
                281  *
958A: AD B4 95  282  RESTORER LDA  ZER0
958D: 85 CE     283           STA  MB
958F: AD B5 95  284           LDA  ZER0+1
9592: 85 CF     285           STA  MB+1
9594: AD B6 95  286           LDA  ZER0+2
9597: 85 FE     287           STA  AB
9599: AD B7 95  288           LDA  ZER0+3
959C: 85 FF     289           STA  AB+1
                290  *
959E: AD B8 95  291           LDA  REGISTER
95A1: AE B9 95  292           LDX  REGISTER+1
95A4: AC BA 95  293           LDY  REGISTER+2
95A7: 28        294           PLP
95A8: 60        295           RTS             ;EXIT
                296  *
                297  * Aux 0200-BFFF von außerhalb
                298  * 0200-BFFF lesen !!!
                299  *
95A9: 8E 03 C0  300  STACK2   STX  $C003      ;AUXRD
95AC: B1 FE     301           LDA  (AB),Y
95AE: 8E 02 C0  302           STX  $C002      ;MAINRD
95B1: 4C 58 95  303           JMP  II
                304  *
                305  * Zero-Page retten
                306  *
95B4: 00 00 00  307  ZER0     HEX  00000000   ;MB/AB
95B7: 00
                308  *
                309  * Register A-X-Y retten
                310  *
95B8: 00 00 00  311  REGISTER HEX  000000     ;A-X-Y

--End assembly-

443 bytes

Errors: 0
```

```
100  PRINT : PRINT  CHR$ (4)„BLOAD AUXMOVER"
110  CLEAR :0 = 37888: HIMEM: 0: REM $9400
120  POKE 0 + 3,0: REM Auf 64K-Karte schreiben
130  POKE 0 + 6,0: POKE 0 + 7,32: REM $2000 MAIN-BEGIN
140  POKE 0 + 8,255: POKE 0 + 9,63: REM $3FFF MAIN-END
150  POKE 0 + 4,0: POKE 0 + 5,0: REM $0000
160  FOR X = 0 TO 7
170  HGR : HCOLOR= 7:A = X + 5:A = A * A: HPLOT A,0 TO A,159
180  POKE 0 + 5,X * 32: REM *HIGHBYTE
190  CALL 0: NEXT X
200  REM *** Hier mit RUN neustarten ***
210  CLEAR :0 = 37888: HGR
220  POKE 0 + 3,1: REM Von 64K-Karte lesen
230  FOR Y = 1 TO 3
240  POKE 0 + 4,0: POKE 0 + 5,0: REM $0000
250  FOR X = 0 TO 7
260  POKE 0 + 5,X * 32: REM *HIGHBYTE
270  CALL 0: NEXT X,Y
280  GET X$: TEXT
```

4. Applesoft-ROM

Man beachte, daß sich die Darstellungen in diesem Kapitel auf den Apple II Plus und Apple IIe beziehen, denn beim neuen Apple IIc ist der Interpreter umgeschrieben worden, so daß sicherlich wieder geraume Zeit ins Land gehen wird, bis im einzelnen überprüft worden ist, ob und inwieweit noch Kompatibilität zu den vorangehenden Modellen vorliegt. Nachfolgend werden die wichtigsten Abweichungen aufgrund eigener Untersuchungen genannt, da die Apple IIc-Handbücher noch nicht vorliegen:

– Alle Kassettenbefehle sind verschwunden: SHLOAD, RECALL, STORE, LOAD und SAVE. Die entsprechenden Sprungadressen sind auf den Ampersand-Vektor umgelenkt worden ($03F5 - 1), so daß man sich diese Befehle offenbar bei Bedarf selbst schreiben soll.

– Dagegen wurde die PARSE-Routine, die u.a. die Befehlswörter PRINT, HOME usw. interpretiert, dahingehend geändert, daß nunmehr Befehlswörter auch in Kleinbuchstaben print, home usw. eingegeben werden können.

– Einige bekannte Bugs, z.B. bei Tab, wurden beseitigt, andere bekannte Bugs, z.B. bei POP, jedoch bewußt stehengelassen. Dies gehört zur Marketing-Strategie der Firma Apple: „Feature = bug as described by the marketing department" (siehe altes „Apple II Reference Manual", S. 180). Daß echte Profis wie z.B. Glen Bredon den Applesoft-Interpreter detailliert seziert haben, interessiert die heutigen Pepsi-Cola-Manager von Firma Apple nicht mehr.

– Der Bereich $DB3A-$F1D4, d.h. von STROUT bis zum Ende der gesamten Fließkommaroutinen, wurde nicht geändert (von einer unbedeutenden Stelle betreffend SCRN abgesehen).

– Die Grafik-Routinen ab $F1D5 wurden wegen Double-Graphics stark verändert.

4.1. Applesoft und Assembler

Manche Leser werden sich fragen, warum in einem Assemblerbuch über Applesoft gesprochen wird. Dies tangiert die Grundfrage, wann in welcher Sprache programmiert werden sollte. Bezüglich Applesoft und Assembler gilt:

a) Sogenannte „Wegwerfprogramme", also kurze Programme, die nur wenige Male, u.U. sogar nur ein einziges Mal benutzt werden, sollte man in Applesoft schreiben, da es hier auf Geschwindigkeit und Benutzerkomfort nicht ankommt. Die Entwicklung eines Assemblerprogramms kann nämlich u.U. 1000mal solange dauern wie die Entwicklung des entsprechenden Applesoftprogramms. „Typische" Assemblerprogramme, z.B. einfache Move-Routinen, sind natürlich oft sogar noch schneller geschrieben als entsprechende Applesoftprogramme. Aber wie lange würde die *reine* Assemblerprogrammierung für ein in wenigen Sekunden erstelltes Applesoftminiprogramm in der Art

```
FOR X = 1 TO 1000: PRINT INT (EXP (LOG (INT (SQR (X * X))))): NEXT
```

dauern, wenn man *nicht* auf ROM-Routinen zurückgreifen würde? Zweifellos mehr als 1000mal so lange, denn die Entwicklung eines professionellen Fließkommapakets ist eine „Lebensaufgabe". Ohne Mathematikstudium geht hier gar nichts!

b) Programme, bei denen die vorgegebenen Applesoftbefehle voll ausreichen, z.B. viele Mathematikprogramme, und bei denen Geschwindigkeit und Kompaktheit des Programmcodes eine untergeordnete Rolle spielen, können in Applesoft geschrieben und dann ggf. zur Geschwindigkeitssteigerung compiliert werden.

c) Wenn geeignete und/oder extrem mächtige Applesoftbefehle fehlen – z.B. ein leistungsfähiger INPUT-Befehl, der jeden erdenklichen Redigierkomfort mit Insert, Delete, Restore usw. zuläßt, oder ein PRINT.USING-Befehl zur Zahlenformatierung –, so muß man diese durch ein Assemblerprogramm simulieren. Diesem Zweck dient ein Großteil der Utilities in diesem Buch. Diese Befehle können dann in ein compiliertes oder nicht-compiliertes Applesoftprogramm eingebunden werden.

d) Wenn optimale Geschwindigkeit und Kompaktheit im Vordergrund stehen und Applesoft hier keine leistungsfähigen Befehle anbietet und ferner nach einer Compilierung zu umfangreich werden würde, muß man das gesamte

Programm, insbesondere wenn es sehr häufig benutzt wird, in Assembler schreiben.

Exkurs: Applesoft-Interpreter versus Applesoft-Compiler

Der Interpreter im ROM $D000-$F7FF ist ein lupenreines Maschinenprogramm, das grob gesagt aus zwei Teilen besteht, einem Befehlsinterpreter und einer Sammlung von Assembler-Routinen für die einzelnen Befehle. Ein Applesoft-Statement ist kein Maschinenprogramm, sondern eine Folge von Befehlstoken und Parametern, die einer Deutung = Interpretierung bedürfen.

Fall 1: 10 HOME

Wenn wir im Direktmodus HOME Return eingeben, wird die Zeichenfolge „HOME" dechiffriert und als HOME-Befehl erkannt, der dann zum Aufruf der bekannten $FC58-Routine führt. In einer Programmzeile steht statt der Zeichenfolge „HOME" das 1-Byte-Token $97, doch eine Dechiffrierung ist auch hier erforderlich. Würde ein reines Assemblerprogramm mit JSR $FC58 die HOME-Routine direkt aufrufen, dann würde auf die Dechiffrierung keine unnötige Zeit verschwendet. Allerdings wäre die Zeitersparnis minimal, da die Dechiffrierung von $97 hier erheblich schneller ist als die Ausführung des HOME-Befehls selbst.

Fall 2: 10 GOSUB 1000

Hier geschieht dreierlei: Erstens muß wiederum das Token $B0 für GOSUB entziffert werden; dies dauert nicht besonders lange. Zweitens muß der Zahlenstring „1000" in eine Hexzahl umgewandelt werden; dies dauert schon erheblich länger. Drittens muß die Zeile 1000 im Programmspeicher gesucht werden; hier hängt die Suchdauer von der Größe des Applesoftprogramms ab: Je weiter Zeile 1000 vom Programmanfang TXTTAB entfernt ist, desto länger dauert die Suche. Bei einem Assemblerprogramm wäre man mit JSR $HHLL (= Speicherstelle der GOSUB-1000-Routine) „blitzartig" an der gewünschten Stelle.

Fall 3: 10 PRINT A (X)

Neben der Dechiffrierung des Tokens $BA für PRINT muß erstens die Variable X im Speicherbereich der einfachen Variablen sowie zweitens die Variable A (X) im Speicherbereich der dimensionierten Variablen gesucht werden. Der

Wert der letzteren Variablen wird dann in den Fließkommaakkumulator FAC zwecks Konvertierung und Ausgabe als „normale" Dezimalzahl über PRNTFAC übertragen. Das Suchen der einfachen und dimensionierten Variablen hängt von der Größe der entsprechenden Variablenspeicherbereiche ab. Eine Variable, die am Anfang des Variablenspeichers steht, wird schneller gefunden als eine Variable, die weiter hinten oder gar ganz am Ende steht. Ein reines Assemblerprogramm findet seine Variablen zwar nicht „blitzartig", doch kann man die Suchverfahren dem jeweiligen Anwendungszweck des Assemblerprogramms anpasssen und vielfach mit absoluten Adressen arbeiten, während der Applesoft-Interpreter immer nach Schema F vorgeht.

Ein Compiler ist eine Assembler-Utility, die einen Applesoft-Source-Code in einen Applesoft-Objekt-Code umwandelt, der zwar ein reines, jedoch mechanisch und deshalb mehr oder weniger umständlich erstelltes Assemblerprogramm darstellt. Gerade diese „mechanische Assemblierung" bewirkt eine verhältnismäßig starke Aufblähung des Objekt-Codes, die naturgemäß mit Geschwindigkeitseinbußen einhergeht. Als Grundregeln gelten für alle Applesoft-Compiler:

a) Je größer das Applesoftprogramm und je größer die Anzahl der Variablen, desto relativ langsamer wird das Applesoftprogramm und desto relativ schneller wird die compilierte Version, weil der Compiler alle GOTOs und GOSUBs in absolute Adresssen umwandelt und (Zahlen-)Variablen aufgrund absoluter Adressen (bzw. bei Arrays aufgrund von errechneten Pointern) „sofort" findet.

b) Als Preis dafür wird der compilierte Objekt-Code mindestens doppelt so groß wie der nicht-compilierte Source-Code.

Um die Fähigkeiten einzelner Applesoft-Compiler genauer zu untersuchen, muß man gekünstelte Testprogramme schreiben, die Teilleistungen der Compiler isoliert herausarbeiten. Nachfolgend werden anhand von insgesamt 8 Tests die bekannten vier Applesoft-Compiler TASC, EXPEDITER, SPEEDSTAR und HAYDEN verglichen. (Bei HAYDEN wurde die verbesserte Version HAYDEN COMPILER + genommen. Die verbesserte EXPEDITER-Version namens EINSTEIN COMPILER lag mir nicht vor.)

```
10  REM !INTEGERX
15  HOME : PRINT „BILDSCHIRMTEST J/N ";: GET X$
20  IF X$ = „N" THEN  HOME : END
25  IF X$ <  > „J" THEN 15
30  VTAB 10: HTAB 1: PRINT „STOPPUHR !"
35  PRINT : PRINT „W = WEITER ";: GET X$
40  HOME
45  FOR X = 1 TO 1000: PRINT X„ ";: NEXT
50  HOME
55  FOR X = 1 TO 1000: PRINT „AAAAA ";: NEXT
60  FOR X = 1 TO 5: PRINT  CHR$ (7);: NEXT
65  PRINT „W = WEITER ";: GET X$: GOTO 15
```

```
10  REM <S>60
15  REM !INTEGERX
20  HOME : PRINT „DRUCKERTEST J/N ";: GET X$
25  IF X$ = „N" THEN  HOME : END
30  IF X$ <  > „J" THEN 20
35  PRINT : PRINT : PRINT „DRUCKER EINSCHALTEN !"
40  VTAB 10: HTAB 1: PRINT „STOPPUHR !"
45  PRINT : PRINT „W = WEITER ";: GET X$
50  IF X$ <  > „W" THEN 20
55  HOME :X$ = „XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    XXXXXXXXXXXXXXXXXXXXXXXXXXX"
60  PRINT : PRINT  CHR$ (4)„PR#1"
65  PRINT
70  FOR X = 1 TO 50
75  PRINT X$
80  NEXT X
85  PRINT  CHR$ (12): PRINT  CHR$ (4)„PR#0"
90  FOR X = 1 TO 5: PRINT  CHR$ (7);: NEXT
95  PRINT „W = WEITER ";: GET X$: GOTO 20
```

```
10  REM !INTEGERX
14  HOME : PRINT „SPEICHERTEST": PRINT : PRINT „(DISKETTE IN
    LAUFWERK 1)"
18  PRINT : PRINT „STARTEN J/N ";: GET X$: IF X$ = „N" THEN
    HOME : END
22  IF X$ <  > „J" THEN 14
26  VTAB 10: HTAB 1: PRINT „STOPPUHR BEREITLEGEN !": PRINT :
    PRINT „W = WEITER ";: GET X$
30  HOME : PRINT  CHR$ (7): PRINT „SPEICHERN..."
34  PRINT : PRINT  CHR$ (4)„OPEN XXX,D1": PRINT  CHR$ (4)
    „WRITE XXX"
38  FOR X = 1 TO 500
42  PRINT „AAAAAAAAAA": PRINT X
46  NEXT X
50  PRINT  CHR$ (4)„CLOSE"
54  HOME : PRINT  CHR$ (7): PRINT „EINLESEN..."
58  PRINT : PRINT  CHR$ (4)„OPEN XXX": PRINT  CHR$ (4)„READ XXX"
62  FOR X = 1 TO 500
```

```
66  INPUT X$: INPUT Y
70  NEXT X
74  PRINT  CHR$ (4)„CLOSE"
78  PRINT  CHR$ (4)„DELETE XXX"
82  FOR X = 1 TO 5: PRINT  CHR$ (7): NEXT
86  GOTO 14
```

```
10  REM !INTEGERX
15  HOME : PRINT „RECHENTEST J/N ";: GET X$
20  IF X$ = „N" THEN  HOME : END
25  IF X$ < > „J" THEN 15
30  VTAB 10: HTAB 1: PRINT „STOPPUHR !"
35  PRINT : PRINT „W = WEITER ";: GET X$
40  HOME
45  FOR X = 1 TO 1000: PRINT X„ ";
50 A = X:A =  RND (1) + A + 1000
55 B =  SQR (A):B = A ↑ 2:B =  INT (A):B =  ABS (A):B =  LOG (A)
60 B = A * A * A:B = A / A / A:B = A + A + A:B = A - A - A
65  NEXT X
70  FOR X = 1 TO 5: PRINT  CHR$ (7);: NEXT
75  PRINT
80  PRINT „W = WEITER ";: GET X$: GOTO 15
```

```
10  REM <S>30
14  REM !INTEGER*
18  DIM S$(100)
22  HOME : PRINT „STRINGTEST J/N ";: GET X$
26  IF X$ = „N" THEN  HOME : END
30  IF X$ < > „J" THEN 22
34  VTAB 10: HTAB 1: PRINT „STOPPUHR !"
38  PRINT : PRINT „W = WEITER ";: GET X$
42  HOME
46  FOR X = 1 TO 100:S$(X) = „"
50  FOR Y = 1 TO 30
54 A =  INT (26 *  RND (Y)) + 65
58 S$(X) = S$(X) +  CHR$ (A)
62  NEXT Y
66  PRINT X„-"S$(X)
70  NEXT X
74  FOR Y = 1 TO 10: HOME : PRINT Y
78  FOR X = 1 TO 100
82 S$(X) =  LEFT$ (S$(X),20) + „-" +  MID$ (S$(Y),10,1) + „-" +
    RIGHT$ (S$(1),1)
86  PRINT X„-"S$(X)
90  NEXT X: NEXT Y
94  FOR X = 1 TO 5: PRINT  CHR$ (7);: NEXT
98  PRINT „W = WEITER ";: GET X$: GOTO 22
```

```
10  REM !INTEGER*
15  DIM A%(5000)
20  HOME : PRINT „ZUWEISUNGSTEST J/N ";: GET X$
25  IF X$ = „N" THEN  HOME : END
30  IF X$ < > „J" THEN 20
35  VTAB 10: HTAB 1: PRINT „STOPPUHR !"
40  PRINT : PRINT „W = WEITER ";: GET X$
45  HOME : PRINT „ZUWEISUNG..."
50  FOR X = 1 TO 5000:A%(X) = 1: NEXT X
55  HOME : PRINT „VERGLEICHEN...":A%(2500) = 0
60  FOR Y = 1 TO 10
65  FOR X = 1 TO 5000: IF A%(X) = 0 THEN  PRINT 0: GOTO 75
70 C% = A%(X):A%(X) = A%(5000):A%(5000) = C%
75  NEXT X
80  NEXT Y
85  FOR X = 1 TO 5: PRINT  CHR$ (7);: NEXT
90  PRINT „W = WEITER ";: GET X$: GOTO 20
```

```
11  HOME : PRINT „SPRUNGTEST"
12  PRINT : PRINT : PRINT „STARTEN J/N ";: GET X$: IF X$ = „N"
    THEN HOME : END
13  IF X$ < > „J" THEN 11
14  HTAB 1: VTAB 10: PRINT „STOPPUHR BEREITLEGEN !": PRINT :
    PRINT „W = WEITER ";: GET X$
15  HOME
16  FOR X = 1 TO 5: PRINT X
17  FOR Y = 1 TO 1000: GOSUB 400
18  NEXT Y
19  NEXT X
30  FOR X = 1 TO 5: PRINT  CHR$ (7);: NEXT : PRINT : PRINT
    „W = WEITER ";: GET X$: GOTO 11
40  REM TEST-GOSUB-GOTO
50  GOSUB 60
60  GOTO 70
70  RETURN
100  REM FÜLLPROGRAMMZEILEN
101  HOME :X$ = „TESTZEILE":Y = 10: FOR X = 1 TO Y: PRINT X$:
     NEXT X
102  HOME :X$ = „TESTZEILE":Y = 10: FOR X = 1 TO Y: PRINT X$:
     NEXT X
103  HOME :X$ = „TESTZEILE":Y = 10: FOR X = 1 TO Y: PRINT X$:
     NEXT X
104  REM *** Beim Originaltest enthalten die Zeile 104-250
     jeweils dasselbe wie die vorangehenden 3 Zeilen 100-103 ***
290  REM TEST-GOSUB-GOTO
300  GOSUB 50
400  GOTO 500
500  RETURN
```

```
10  REM <S>10
15  REM !INTEGER*
20  HOME : PRINT „GRAFIKTEST J/N ";: GET X$
25  IF X$ = „N" THEN  HOME : END
30  IF X$ <  > „J" THEN 20
35  VTAB 10: HTAB 1: PRINT „STOPPUHR !"
40  PRINT : PRINT „W = WEITER ";: GET X$
45  HGR2
50  FOR X = 1 TO 10
55  FOR Y = 0 TO 7: HCOLOR= Y
60  FOR Z = 10 TO 110
65  HPLOT 80,Z TO 180,Z
70  NEXT Z
75  NEXT Y
80  NEXT X
85  FOR X = 1 TO 5: PRINT  CHR$ (7);: NEXT
90  TEXT
95  PRINT „W = WEITER ";: GET X$: GOTO 20
```

1. Bildschirmtest

Ausgabe von Zahlen und Strings über den Bildschirm: Die Geschwindigkeitssteigerung ist hier bei allen Compilern praktisch gleich null.

2. Druckertest

Ausgabe von Strings über den Drucker: Auch hier ist die Geschwindigkeitssteigerung praktisch gleich null.

3. Speichertest

Speichern/Einlesen von Zahlen und Strings auf/von Diskette: Compiler verbessern nicht das DOS – so enthält z.B. der TASC keinen einzigen Sprung in eine DOS-Routine –, so daß nur bei langen Programmen allein aufgrund der verbesserten Variablensuche eine Geschwindigkeitssteigerung zu verbuchen wäre.

4. Rechentest

Grundrechenarten und höhere Mathematik: Auch hier ist interessanterweise bei kurzen Programmen die Geschwindigkeitssteigerung praktisch gleich null, da alle Compiler die vorliegenden Interpreter-Mathematikroutinen benutzen. Hingegen ist bei Compilern, die über ein Integer-Paket verfügen (TASC, HAYDEN), die Ausführung von Integerarithemetik mindestens so schnell wie Integer-Basic selbst.

5. Stringtest

Stringverkettungen usw.: Die Geschwindigkeitssteigerung ist nennenswert, doch ist das Problem der Garbage Collection bei keinem Compiler optimal gelöst. Man beachte, daß der EXPEDITER als einziger Compiler mit festen Stringlängen (ähnlich wie Integer-Basic) arbeitet. Dadurch entfällt zwar die Garbage Collection, doch setzt dies zugleich eine detailliert geplante Festlegung der einzelnen Stringlängen voraus, denn bei einer generellen Länge von 255 Zeichen würde der String-Pool bereits allein durch DIM A$ (100) überlaufen.

6. Zuweisungstest

Wertzuweisungen in der Art X = Y: A (X) = 100 usw. bei größeren Variablenspeichern: Hier zeigt sich die eigentliche Stärke der Compiler, die bis zu 12mal schnellere Objekt-Codes erzeugen.

7. Sprungtest

GOTOs und GOSUBs bei größeren Programmen: Hier kann je nach Programmlänge mit einer mehr als 30fachen Geschwindigkeitssteigerung gerechnet werden.

8. Grafiktest

Hires-Plotten: Die Geschwindigkeitssteigerung ist hier praktisch gleich null. Die besseren Testergebnisse rühren von den von Compilern grundsätzlich schneller ausgeführen FOR-NEXT-Schleifen her. Im übrigen verwenden die Compiler die normalen Interpreter-Grafikroutinen.

Test-Nr.	Applesoft	Tasc	Expediter	Speedstar	Heyden
Bildschirm	19,5	16,5	17,2	17,5	16,7
Drucken	57	57	57	57	57
Speichern	60,5	56,5	59	62	55
Rechnen	170	144	143,5	146	143
Strings	137	75	91,6	88	70
Zuweisung	1343	108	197	535	190
Springen	145	4,5	4,5	-	4,5
Grafik	144	77	85	88	87

Anmerkungen

1. Bei großen Programmen ist mit einer ca. 2fachen (SPEEDSTAR), ca. 3-4fachen (EXPEDITER) und ca. 3,5-4fachen (HAYDEN, TASC) Steigerung der Verarbeitungsgeschwindigkeit zu rechnen.

2. Große Applesoftprogramme sind nach der Compilierung beim TASC ca. 1,9-2 mal, beim EXPEDITER ca. 2,2-2,3 mal, beim HAYDEN ca. 2,2-2,5 mal und beim SPEEDSTAR ca. 2,5 mal so lang. Sehr große Programme können nur

mit dem TASC compiliert werden. Wie oben ersichtlich, steigt der SPEEDSTAR bereits bei mittelgroßen Programmen aus.

3. Der Compilierungsvorgang selbst ist beim TASC am langsamsten, beim HAYDEN am schnellsten.

4. Meiner Meinung nach ist der TASC der beste Compiler, zumal er völlig DOS-neutral ist, sich also *nach* der Compilierung mit jedem DOS (z.B. DiversiDOS) verträgt. Zur Compilierung selbst muß allerdings das normale DOS 3.3 verwendet werden. (Das MMS-LC-DOS des HAYDEN läuft nicht auf dem Apple IIe.)

4.1.1. Methoden der Parameterübergabe

4.1.1.1. PEEK, POKE, CALL, & und USR

Für den Applesoft-Programmierer gibt es fünf Befehle, die eine Verbindung zu Assemblerprogrammen herstellen:

a) P = PEEK (S)

PEEK weist einer Variablen P den Inhalt (im Bereich 0-255) einer Speicherstelle S (im Bereich 0-65535) zu. Der PEEK-Befehl steht im Applesoft-ROM bei $E764:

```
LDA   LINNUM      ;LINNUM $0050-$0051 retten
PHA
LDA   LINNUM+1
PHA
JSR   GETADR      ;S in Hexadresse in LINNUM umwandeln
LDY   #0
LDA   (LINNUM),Y  ;Wert der Stelle S laden
TAY               ;und in Y-Register retten
PLA               ;LINNUM wieder herstellen
STA   LINNUM+1
PLA
STA   LINNUM
JMP   SNGFLT      ;Y-Register in FAC umwandeln (0-255)
```

b) POKE S, P

POKE weist der Speicherstelle S (0-65535) den Wert P (0-255) zu. Der POKE-Befehl steht im Applesoft-ROM bei $E77B:

```
JSR   GETNUM      ;S in LINNUM, P in X-Register umwandeln
TXA
LDY   #0
STA   (LINNUM),Y  ;P in S speichern
RTS
```

c) CALL S

CALL ruft eine Assemblerroutine bei Speicherstelle S auf, die nach RTS wieder zum Applesoftprogramm zurückführt. Der CALL-Befehl steht im Applesoft-ROM bei $F1D5:

```
JSR   FRMNUM  ;S in FAC umwandeln
JSR   GETADR  ;FAC in Hexadresse in LINNUM umwandeln
JMP   (LINNUM) ;indirekt nach S springen
```

d) &

Der &-Befehl oder Ampersand (Und-Zeichen) bewirkt einen Sprung zur Speicherstelle $03F5 (= Ampersand-Vektor), die ihrerseits einen Sprung zu einer eigenen Assemblerroutine enthalten kann, die mit RTS abschließen muß, z.B.:

```
03F5: 4C 00 03 JMP $0300
```

e) A = USR (B)

Der USR-Befehl wertet den Ausdruck B aus, überträgt dessen Wert in FAC, bereitet die Wertzuweisung zu A vor und springt dann zur Speicherstelle $000A, die ihrerseits einen Sprung zu einer eigenen Assemblerroutine enthalten kann, die mit RTS abschließen muß, z.B.:

```
000A: 4C 00 03: JMP $0300
```

Die Assemblerroutine übernimmt den Wert B, verarbeitet ihn und überträgt vor dem RTS bei Bedarf einen neuen Wert B in FAC, von wo er durch den Applesoft-Interpreter der Variablen A zugewiesen wird.

Der weiter unten besprochene Textpointer TXTPTR befindet sich nach CALL S, & und A = USR (B) immer an der Speicherstelle, die dem Befehl folgt, wobei bei & zusätzlich der Akkumulator das Byte dieser Speicherstelle enthält. Betrachten wir zu diesem Zweck folgende Beispiele:

```
10 CALL S: PRINT
15 CALL S, V1, V2: PRINT
20 &: PRINT
25 & V1, V2: PRINT
30 A = USR (B): PRINT
35 A = USR (B), V1, V2: PRINT
36 A = USR (B) V1, V2: PRINT
```

In Zeile 10 zeigt der Textpointer nach CALL S auf den Doppelpunkt. In der normalerweise illegalen Zeile 15 zeigt der Textpointer nach CALL S auf das Komma, doch enthält das A-Register nicht den Kommawert.
In Zeile 20 zeigt der Textpointer nach & auf den Doppelpunkt und A enthält $3A (ASCII-Code für Doppelpunkt; Bit 7 off). In der normalerweise illegalen Zeile 25 zeigt der Textpointer nach & auf V1 und das A-Register enthält den entsprechenden ASCII-Code von „V".
In Zeile 30 zeigt der Textpointer nach A = USR (B) auf den Doppelpunkt. In der normalerweise illegalen Zeile 35 zeigt der Textpointer nach A = USR (B) auf das Komma. Ferner zeigt in der ebenfalls illegalen Zeile 36 der Textpointer nachher auf V1. Das A-Register enthält in keinem der Fälle den Speicherwert.

Normalerweise gestattet der CALL-Befehl außer S keine zusätzlichen Parameter. Steht jedoch nach dem CALL S ein Komma, dann wird es zunächst ignoriert. Erst nach Rückkehr vom CALL entsteht eine Fehlermeldung. Ein Assemblerprogramm kann den Textpointer jedoch nach und nach vorrücken, um die Variablen V1, V2 usw. auszuwerten und um schließlich den Textpointer auf den Doppelpunkt zu setzen, damit Applesoft „nichts merkt". Auf diese Weise ist ein CALL mit Parametern möglich. Zwischen CALL S und dem ersten Parameter muß jedoch ein Komma als Delimiter stehen, denn z.B. CALL SV1 würde falsch interpretiert und CALL 768V1 würde sofort zur Fehlermeldung führen.

Normalerweise gestattet der &-Befehl keine zusätzlichen Parameter, doch kann wie bei CALL das gleiche Verfahren angewandt werden. Ein Komma als Delimiter ist jedoch entbehrlich, da Applesoft nach & keine Parameter im Sinne von S erwartet.

Gleiches gilt für den USR-Befehl. Hier kann aus Gründen der besseren Lesbarkeit ein Komma als Delimiter gesetzt werden. Erforderlich ist es jedoch nicht.

Der Ampersand ist die wichtigste Befehlserweiterungsmöglichkeit für nicht-compilierte Applesoftprogramme. Der CALL + Parameter hat gegenüber dem Ampersand den Vorteil, daß beliebig viele CALLs zu beliebig vielen Adressen möglich sind. Ampersand springt immer nur an eine einzige Stelle ($03F5), so daß einer Sammlung von Ampersand-Utilities ein Befehlsinterpreter vorangestellt werden muß, z.B. in der speicherökonomischen Form von Token: & COS..., & PRINT..., & CLEAR...usw. PEEK, POKE sowie USR und CALL ohne Parameter sind auch bei compilierten Applesoftprogrammen möglich, nicht hingegen Ampersand. USR hat wie & den Nachteil, daß es immer nur an eine einzige Stelle ($000A) springt. Ferner können mit USR nur Fließkommazahlen übergeben werden.

4.1.1.2. Wie funktioniert der Interpreter?

Wenn man die obigen Verfahren der Parameterübergabe praktizieren will, muß man mit der internen Funktionsweise des Interpreters sowie der Speicherorganisation vertraut sein. Zu diesem Zweck gebe man das nachfolgende Applesoftprogramm exakt in der vorliegenden Form ein. Es erzeugt dann von sich selbst einen Programm- und Variablenspeicher-Dump, den man eingehend studieren sollte.

```
$0800 00 24 08 64 00 89 3A 97 3A BA 22 41 50 50 4C 45   $ d  : ::"APPLE
$0810 53 4F 46 54 2E 53 50 45 49 43 48 45 52 3A 22 3A  SOFT.SPEICHER:":
$0820 BA 3A BA 00 45 08 6E 00 4B D0 33 32 37 36 32 3A  ::: E n KP32762:
$0830 4B 25 D0 33 32 37 36 32 3A 4B 24 D0 22 53 54 52  K%P32762:K$P"STR
$0840 49 4E 47 22 00 5B 08 78 00 86 41 28 35 29 2C 42  ING" [ x  A(5),B
$0850 25 28 35 29 2C 43 24 28 35 29 00 66 08 82 00 81  %(5),C$(5) f
$0860 58 D0 30 C1 35 00 73 08 8C 00 41 28 58 29 D0 4B  XP0A5 s   A(X)PK
$0870 C8 58 00 81 08 96 00 42 25 28 58 29 D0 4B C8 58  HX      B%(X)PKHX
$0880 00 92 08 AC 07 43 24 28 58 29 DC E4 28 4B C8 58        C$(X)Pd(KHX
$0890 29 00 98 08 AA 00 62 00 EE 08 B4 00 BA 22 5A 45  )    *   n 4 :"ZE
$08A0 52 4F 2D 50 4F 49 4E 54 45 52 3A 22 3A 3A 53 24  RO-POINTER:"::S$
$08B0 D0 22 4E 36 37 20 4E 36 38 20 4E 36 39 20 4E 36  P"N67 N68 N69 N6
$08C0 41 20 4E 36 42 20 4E 36 43 20 4E 36 44 20 4E 36  A N68 N6C N6D N6
$08D0 45 20 4E 36 46 20 4E 37 30 20 4E 37 33 20 4E 37  E N6F N70 N73 N7
$08E0 34 22 3A B0 32 32 30 3A 8C C9 31 34 34 00 28 09  4":0220: I144 (
$08F0 BE 00 BA 3A BA 3A BA 22 50 52 4F 47 52 41 4D 4D  > :::::"PROGRAMM
$0900 20 2B 20 50 4F 49 4E 54 45 52 3A 22 3A 53 24 D0   + POINTER:":S$P
$0910 22 4E 30 38 30 30 2E 30 41 31 37 22 3A B0 32 32  "N0800.0A17":022
$0920 30 3A 8C C9 31 34 34 00 57 09 C8 00 BA 3A BA 3A  0: I144 W H :::
$0930 BA 22 53 54 52 49 4E 47 53 3A 22 3A 53 24 D0 22  :"STRINGS:":S$P"
$0940 4E 39 35 37 38 2E 39 35 46 46 22 3A B0 32 32 30  N9578.95FF":0220
$0950 3A 8C C9 31 34 34 00 5D 09 D2 00 80 C0 9E 09 DC  : I144 ] R      ?
$0960 00 53 24 D0 53 24 C8 22 20 4E 44 37 44 32 47 22   S$PS$H" ND7D2G"
$0970 3A 81 58 D0 31 C1 E3 28 53 24 29 3A B9 35 31 31  : XP1Ac(S$):9511
$0980 C8 58 2C E6 28 EA 28 53 24 2C 58 2C 31 29 29 CB  HX,f(j(S$,X,1))H
$0990 31 32 38 3A 82 3A B9 37 32 2C 30 3A B1 00 00 00  128: :972,0:1
$09A0 4B 00 8F 7F F4 00 00 CB 80 7F FA 00 00 00 4B 80  K K% K$
$09B0 06 3D 08 00 00 5E 00 85 10 00 00 00 53 80 11 8A  X S$
$09C0 95 00 00 41 00 25 00 01 00 06 8F 7F F4 00 00 8F  A(X)
$09D0 7F F6 00 00 8F 7F F8 00 00 8F 7F FA 00 00 8F 7F
$09E0 FC 00 00 8F 7F FE 00 00 C2 80 13 00 01 00 06 7F  B%(X)
$09F0 FA 7F FB 7F FC 7F FD 7F FE 7F FF 43 80 19 00 01  C$(X)
$0A00 00 06 05 FB 95 05 F6 95 05 F1 95 05 EC 95 05 E7
$0A10 95 05 E2 95 00 00 00 00 00 00 00 00 00 00 00 00
$9580 00 00 00 00 00 00 00 00 00 47 4E 39 35 37 38 2E                GN9578.
$9590 39 35 46 46 20 4E 44 37 44 32 47 4E 30 38 30 30  95FF ND7D2GN0800
$95A0 2E 30 41 31 37 20 4E 44 37 44 32 47 4E 36 37 20  .0A17 ND7D2GN67
$95B0 4E 36 38 20 4E 36 39 20 4E 36 41 20 4E 36 42 20  N68 N69 N6A N6B
$95C0 4E 36 43 20 4E 36 44 20 4E 36 45 20 4E 36 46 20  N6C N6D N6E N6F
$95D0 4E 37 30 20 4E 37 33 20 4E 37 34 20 4E 44 37 44  N70 N73 N74 ND7D
$95E0 32 47 33 32 37 36 37 33 32 37 36 36 33 32 37 36  2G3276732766 3276
$95F0 35 33 32 37 36 34 33 32 37 36 33 33 32 37 36 32  5327643276332762
```

```
100  TEXT : HOME : PRINT „APPLESOFT.SPEICHER:": PRINT : PRINT
110 K = 32762:K% = 32762:K$ = „STRING"
120  DIM A(5),B%(5),C$(5)
130  FOR X = 0 TO 5
140 A(X) = K + X
150 B%(X) = K + X
160 C$(X) =  STR$ (K + X)
170  NEXT
180  PRINT „ZERO-POINTER:"::S$ = „N67 N68 N69 N6A N6B N6C N6D N6E N6F
    N70 N73 N74": GOSUB 220: CALL  - 144
190  PRINT : PRINT : PRINT „PROGRAMM + POINTER:":S$ = „N0800.0A17":
    GOSUB 220: CALL  - 144
200  PRINT : PRINT : PRINT „STRINGS:":S$ = „N9578.95FF": GOSUB 220:
    CALL  - 144
210  END
220 S$ = S$ + „ ND7D2G": FOR X = 1 TO  LEN (S$): POKE 511 + X, ASC (
    MID$ (S$,X,1)) + 128: NEXT : POKE 72,0: RETURN
```

Beginn und Ende der einzelnen Programm- und Variablenspeicher lassen sich den folgenden Zero-Page-Pointern entnehmen (die konkreten Werte beziehen sich auf unser Beispielprogramm):

$0067-$0068:	01 08	= $0801 =	Anfang des Programms (TXTTAB) (kann auch größer als $0801 sein)
$0069-$006A:	A0 09	= $09A0 =	Anfang der einfachen Variablen (VARTAB, LOMEM) (zugleich meist Ende des Programms PRGEND)
$006B-$006C:	C3 09	= $09C3 =	Anfang der dimensionierten Variablen (ARYPNT) (zugleich Ende der einfachen Variablen)
$006D-$006E:	14 0A	= $0A14 =	Absolute Untergrenze des String-Pools (STREND) (Ende der dimensionierten Variablen)
$006F-$0070:	AC 95	= $95AC =	Momentane Untergrenze des String-Pools (FRETOP)
$0073-$0074:	00 96	= $9600 =	Absolute Obergrenze des String-Pools + 1 (HIMEM, MEMSIZ)
$00AF-$00B0:	A0 09	= $09A0 =	Ende des Programms (PRGEND)

Ein Applesoft-Programm beginnt normalerweise bei $0801 (TXTTAB), genauer bei $0800, wobei $0800: 00. Es kann jedoch auch z.B. bei $4001 bzw. $4000 beginnen, wenn vor dem LOAD des Programms mit

```
CALL -151
0067: 01 40
4000: 00
Ctrl-C
```

der TXTTAB-Pointer auf $4001 sowie $4000 auf 0 gesetzt werden.

Die einzelnen Zeilen des Applesoft-Programms haben folgenden allgemeinen Aufbau:

1. Link (Verkettungsadresse): Pointer zur nächsten Programmzeile. Zeigt als absolute Adresse LL HH auf die Speicherstelle, wo sich das Link für die übernächste Programmzeile befindet.
2. Zeilennummer als Hexzahl (ohne Vorzeichen) in der Form LL HH

3. Eigentliche Zeile
4. EOL End of Line Delimiter $00

Vor der allerersten Zeile des Programms steht $00 als EOL. Nach der allerletzten Zeile des Programms folgen auf das eigentliche EOL dieser letzten Zeile zwei weitere $00 $00, quasi als Null-Link oder Null-Pointer.

Innerhalb der eigentlichen Zeile werden die Befehlswörter PRINT, INPUT usw. als 1-Byte-Token mit Bit 7 on, d.h. ab $80, abgelegt. Beispiele:

$8C = CALL
$AF = &
$D5 = USR

Zu den Befehlswörtern gehören auch +, -, *, /, <, >, = usw., nicht jedoch Komma ($2C), Klammern ($28-$29), Anführungszeichen ($22), Doppelpunkt ($3A), Dezimalpunkt ($2E) usw. Letztere werden mit Bit 7 off als normale ASCII-Zeichen im Speicher abgelegt. Das gleiche gilt für Zahlenkonstanten (X = 12345), Stringkonstanten (X$ = „ABCDE“) und Variablennamen, die in voller Länge (z.B. VVVVV = 12345) abgelegt werden. Fazit: Befehlstoken mit Bit 7 on, alles übrige mit Bit 7 off.

Im Anschluß an den Programmspeicher – nach den drei letzten $00 $00 $00 – folgt der Variablenspeicher, sofern letzterer nicht mit z.B. LOMEM: 16384 höhergesetzt wird. Der Variablenspeicher läßt sich in vier Bereiche unterteilen:

1. Einfache Zahlenvariablen – z.B. X, X% – mit deren Werten sowie einfache Stringvariablen – z.B. X$ – mit deren Pointern.
2. Dimensionierte Zahlenvariablen – z.B. X (100), X% (100) – mit deren Werten sowie dimensionierte Stringvariablen – z.B. X$ (100) – mit deren Pointern.
3. Momentan freier unterer Teil des String-Pools
4. Momentan belegter oberer Teil des String-Pools.

Bei dem Dump Seite 131 sind zur besseren Übersicht im Programmteil die Link-Bytes sowie im Variablenteil die Variablennamen unterstrichen. Die meisten Programme in diesem Hauptkapitel enthalten weitergehende Informationen zum Programmaufbau (bes. FNDLIN) und Variablenaufbau (bes. PTRGET und GETARYPT).

Wie funktioniert nunmehr der Interpreter? Vereinfacht gesagt, geschieht folgendes: Nach RUN wird der sogenannte Textpointer TXTPTR auf $0801 oder allgemein auf TXTTAB gesetzt. Dort registiert er das Link und die Zeilennummer, die z.B. bei einer Fehlermeldung mit ausgegeben wird (ERROR IN LINE XYZ). Dann setzt er den TXTPTR auf den Beginn der eigentlichen Zeile, die aus mehreren Statements – getrennt durch Doppelpunkt – bestehen kann. Ein Statement ist entweder eine Wertzuweisung (Variablenname = Definition), die nicht mit einem Token (LET) beginnen muß, oder ein sonstiger Befehl, der stets durch ein Token eingeleitet wird. Nehmen wir den denkbar einfachsten Fall von zwei Statements als nackte Token ohne Parameter:

```
100 TEXT :      HOME
    89   3A     97   00
```

Beim eigentlichen Zeilenbeginn steht das Token für TEXT = $89. Der Interpreter sucht aus seiner internen Befehlstabelle, die nach aufsteigenden Token-Werten sortiert ist, die Maschinenroutine für TEXT heraus, erhöht den TXTPTR um 1 zum Doppelpunkt und arbeitet daraufhin die Routine ab. Dann wird der Doppelpunkt = $3A übersprungen und das Token für HOME = $97 ermittelt, befehlsmäßig abgearbeitet und festgestellt, daß danach $00 als EOL folgt, womit diese Zeile erledigt ist.

Wir merken uns:

a) Der TXTPTR ist die Zero-Page-Speicherstelle $00B8-$00B9, die den momentanen Zeiger auf die Programmtextstelle in der Adreßform LL HH enthält. Bei dem obigen Beispiel wurde der TXTPTR sozusagen stetig erhöht. Bei GOSUB, GOTO wird er sprunghaft auf die neue Zeile eingestellt. Bei INPUT wird er vorübergehend auf $01FF (= 1 Byte vor $0200 = Eingabepuffer) gesetzt, wobei der alte Programmstellen-TXTPTR zwischengespeichert wird. Desgleichen wird der TXTPTR beim READ von DATA-Statements vorübergehend geändert, usw.

b) Bei CALL S, A = USR (B) und & zeigt der TXTPTR, wenn das eigene Assemblerprogramm die Kontrolle übernimmt, immer auf die Stelle nach dem Statement, also bei CALL auf die Stelle nach S, bei USR auf die Stelle nach dem Klammerausdruck und bei & auf die Stelle unmittelbar nach &, wobei nur im letzeren Fall zugleich der Akkumulator den Wert dieser Stelle enthält.

c) Mit den Interpreter-Routinen DATA (JSR $D995) bzw. REM (JSR $D9DC), deren Anwendung in unseren Programmen gezeigt wird, kann man aus der Assemblerroutine heraus den TXTPTR auf den nächsten Doppelpunkt bzw. auf das nächste EOL vorrücken, so daß nach CALL, USR und & auch beliebige illegale Parameter-Listen stehen können.

4.1.2.3. BLOAD, LAMPOKE, DATAPOKE und EINZELPOKE

Aus einem Applesoft-Programm heraus kann ein Assemblerprogramm auf folgende Weise in den Speicher gelangen:

a) BLOAD

Der BLOAD oder BRUN ist zumindest bei längeren Maschinenprogrammen die einzig sinnvolle Methode. PRINT CHR$ (4) „BLOAD PROGRAMM" beansprucht nur wenige Bytes in einer Applesoft-Zeile.

b) LAMPOKE

Dieses von einem Programmierer namens S.H.Lam entwickelte Verfahren pokt eine Monitor-Eingabezeile in den Eingabepuffer und ruft dann entsprechende Monitor-Routinen zur Übernahme des Programms in den Speicher auf. Mit NEWSTT ($D7D2) gelangt man zum Applesoftprogramm zurück, wobei zu beachten ist, daß die LAM-Routine nicht aus einem GOSUB heraus aufgerufen werden sollte.
Das LAMPOKE-Verfahren ist langsamer als die DATAPOKE-Methode. Das LAM-Verfahren empfiehlt sich jedoch dann, wenn die Monitor-Befehle L, M usw. als solche aus einem Applesoftprogramm heraus benutzt werden sollen. Man beachte, daß Monitor-Routinen zero-page-mäßig teils mit DOS-Routinen kollidieren.

c) DATAPOKE

Dieses Verfahren ist gelegentlich ganz nützlich, wenn in kürzere Applesoftprogramme Maschinenroutinen eingebunden werden sollen. Bei größeren Applesoftprogrammen nehmen jedoch DATA-Statements zuviel Platz vom eigentlichen Programmspeicher weg.

d) EINZELPOKE

Diese Methode ist nur zu empfehlen, wenn ganz kurze Minimaschinenprogramme gepokt werden sollen.

4.1.2. Interne Applesoft-Routinen

Die internen Interpreter-Routinen sind nicht so ergiebig wie die F8-Routinen. Dies gilt zumal dann, wenn Applesoft-Routinen aus einem reinen Assemblerprogramm heraus aufgerufen werden sollen.

4.1.2.1. Ausgewählte Zero-Page-Adressen

$0000-$0002:	Sprung zu WARMSTART ($E003, $D43C) (wird vom Interpreter selbst nicht benutzt!)
$0003-$0005:	Sprung zu STROUT ($DB3A) (wird vom Interpreter selbst nicht benutzt!) Damit sind $0000-$0005 normalerweise frei für eigene Pointer.
$000A-$000C:	Sprung zur USR-Adresse
$0050-$0051:	LINNUM LL HH (wird für Zeilennummer bei FNDLIN u.a. benutzt)
$0062-$0066:	RESULT (Resultat letzter Multiplikation oder Division)
$007B-$007C:	DATLIN LL HH (Momentane DATA-Zeile)
$007D-$007E:	DATPTR LL HH (Momentane DATA-Speicherstelle)
$0081-$0082:	VARNAM (Letztbenutzter Variablenname)
$0083-$0084:	VARPNT LL HH(Pointer zum Wert der letztbenutzten Variablen)
$009B-$009C:	LOWTR LL HH (wird bei GETARYPT usw. benutzt)
$009D-$00A2:	FAC (6-Byte-Hauptfließkomma-Akkumulator)
$009D:	Exponent von FAC
$009E-$00A1:	Mantisse von FAC
$00A2:	Vorzeichen von FAC
$00A5-$00AA:	ARG (6-Byte-Zweitfließkomma-Akkumulator)
$00A5:	Exponent von ARG
$00A6-$00A9:	Mantisse von ARG
$00AA:	Vorzeichen von ARG
$00AD-$00AE:	STRNG2 HH LL (wird bei MULT16 usw. benutzt)
$00B1-$00C8:	CHRGET (Zu CHRGET, CHRGOT und TXTPTR s. Programm TXTPTR)

$00B7-$00C8: CHRGOT
$00B8-$00B9: TXTPTR
$00C9-$00CD: RANDOM (Zufallszahl; $00CD:FF vor erster RND-Zahl!)

Zu den allgemeinen Programmzeigern siehe Kap. 4.1.1.2.

$0100-$0110: FBUFFR (FAC-String-Puffer für PRNTFAC am unteren Stack)
$0200-$02FF: Tastatureingabepuffer (nur 239 Zeichen zulässig!)
$03F5-$03F7: Sprung zur eigenen Ampersand-Routine

4.1.2.2. Ausgewählte Interpreter-Routinen

Für die Prozedur des Aufrufs der Routinen siehe den Hinweis in Kap. 2.1.2.

4.1.2.2.1. Initialisierungsroutinen

KALTSTART (JMP $F128 oder JMP $E000): Installiert WARMSTART- und STROUT-JMPs bei $0000-$0005, kopiert CHRGET-Kopie $F10B-$F122 nach $00B1 und inialisiert Stackpointer. (Bei der Ermittlung von HIMEM wird STA $C000 durchgeführt, wodurch beim Apple IIe dieser Softswitch geschaltet wird!). Danach Sprung zu WARMSTART.

WARMSTART (JMP $D43C oder $E003 oder $0000): Soft-Entry für Applesoft.

SCRTCH (JMP $D64B): Entspricht dem NEW-Befehl. Allgemeine Pointer und Stackpointer werden initialisiert und Programm „gelöscht", d.h. genauer gesagt nur die Stellen $0800-$0802! Kann aus einem Assemblerprogramm mit CLC JSR $D64B angesprungen werden (z.B. beim TASC-Compiler der Fall).

CLEARC (JSR $D66C): Entspricht dem CLEAR-Befehl. Setzt Variablenpointer und Stackpointer zurück.

RUN (JMP $D566): Entspricht dem RUN-Befehl. Reset-Vektor kann auf diese Adresse gesetzt werden.

STKINI (JSR $D683): Stack-Initialisierung. Wird von CLEAR, RUN usw. als *Subroutine* aufgerufen. Sie werden sich fragen, wie dies möglich ist. Dies rührt

daher, daß *zuerst* der Stackpointer auf $F8 hochgesetzt und *dann* die momentane Rücksprungadresse wieder auf den Stack geschoben wird.

4.1.2.2.2. Zahlenausdruck bei TXTPTR auswerten

NEWSTT (JMP $D7D2): Nächsten Befehl ausführen. Wird von uns bei der LAMPOKE-Methode angewandt. TXTPTR muß bereits auf „:" oder EOL zeigen.

DATA (JSR $D995): TXTPTR auf nächsten „:" oder EOL vorrücken, d.h. zum nächsten Statement.

REM (JSR $D9DC): TXTPTR auf nächstes EOL vorrücken (und dabei „:", falls vorhanden, ignorieren), d.h. zur nächsten Zeile.

CHRGET (JSR $00B1): Character Get erhöht TXTPTR um 1 und lädt dann A mit Zeichen der TXTPTR-Stelle. Leertasten ($20) werden übersprungen, d.h. bewirken erneutes CHRGET.

CHRGOT (JSR $00B7): Character Got lädt (erneut) Zeichen bei TXTPTR-Stelle, ohne zuvor TXTPTR um 1 zu erhöhen. Bei vielen der nachfolgenden Routinen, z.B. LINGET, FIN usw., ist es erforderlich, daß vor dem Einsprung in die Routine das A-Register das Zeichen der TXTPTR-Stelle enthält, womit zugleich die Status-Flags entsprechend gesetzt werden. Unbekannte, d.h. nachfolgend nicht dokumentierte Routinen starte man im Zweifelsfall mit JSR CHRGOT JSR ROUTINE.

SYNCHR ($DEC0): Prüft, ob Zeichen bei TXTPTR-Stelle mit A identisch ist. TXTPTR muß hier wie auch bei den nachfolgenden Routinen bereits auf Stelle zeigen!

vorher: LDA Zeichen (z.B. Anführungszeichen, Komma, Klammer usw.)
JSR SYNCHR

nachher: Wenn nicht identisch, dann Programmabbruch über Syntax-Error. Wenn identisch, dann fällt die SYNCHR-Routine in CHRGET, d.h. nach erfolgreichem SYNCHR enthält A jetzt das *nächste* Zeichen.

CHKCOM (JSR $DEBE): Prüft, ob TXTPTR-Stelle Komma ($2C) enthält. Gegenüber SYNCHR spart man sich vorheriges LDA #$2C.

FRMEVL (JSR $DD7B): Wertet einen beliebigen Zahlen- oder Stringausdruck bei TXTPTR-Stelle aus, z.B. A * B als Zahlenausdruck oder STR$ (A) als Stringausdruck. Der Ausdruck kann auch eine einzelne Variable oder eine Konstante sein oder Konstanten enthalten, z.B. A * 20. FRMEVL führt vorher automatisch JSR CHRGOT durch. Vorher daher JSR CHRGOT nicht erforderlich. Danach ist TXTPTR auf erstes Zeichen nach Ausdruck gerichtet (meistens Doppelpunkt oder EOL).

FRMNUM (JSR $DD67): Ruft FRMEVL auf und prüft dann, ob ein Zahlenausdruck vorlag. Wenn ja, nachher Wert des Zahlenausdrucks in FAC und TXTPTR auf erstes Zeichen nach Ausdruck gerichtet. Wenn nein, Type-Mismatch-Error.

Angenommen, wir wollten per Ampersand 3 Fließkommazahlen übernehmen:

```
& A1,    A2,    A3      (Variablen)
& 7 * X, 7 * 7 , X * X  (Ausdrücke)

JSR    FRMNUM
danach FAC von A1 retten
JSR    CHKCOM
JSR    FRMNUM
danach FAC von A2 retten
JSR    CHKCOM
JSR    FRMNUM
danach FAC von A3 retten
und jetzt Ampersandroutine ausführen
RTS
```

FIN ($EC4A): Zahlenkonstante (keine Variable!), z.B. 12345 (mit Bit 7 off), bei TXTPTR-Stelle auswerten und in FAC übertragen. Der Endmarker muß „:" oder EOL oder ein sonstiges Nichtzahlzeichen sein. TXTPTR-Stelle muß unbedingt ein Zahlzeichen sein, sonst passiert „Wundersames", da FIN Low-Level-Routine ist. Vergleiche hierzu FIN-Programm in Kap. 4.2.11.

```
vorher:   JSR    CHRGOT   ;A muß erstes Zahlzeichen enthalten!
```

LINGET ($DA0C): Zahlenkonstante (keine Variable!) im Bereich 0-63999 bei TXTPTR-Stelle auswerten, in 4stellige Hexzahl ohne Vorzeichen umwandeln

und nach LINNUM ($0050-$0051) übertragen. Ebenfalls Low-Level-Routine wie FIN.

```
vorher:   JSR    CHRGOT   ;A muß erstes Zahlzeichen enthalten!
```

Für die Konvertierung einer Zahl im Bereich 0-65535 verwende man statt LINGET die Folge:

```
JSR   FRMNUM
JSR   GETADR   (siehe weiter unten)
```

PARCHK (JSR $DEB2): Wertet einen eingeklammerten (Zahlen)ausdruck aus, der in FAC übertragen wird. TXTPTR muß auf geöffnete Klammer zeigen. Ansonsten wie FRMEVL, d.h. TXTPTR zeigt nachher auf erstes Zeichen nach geschlossener Klammer. Vorher JSR CHRGOT nicht erforderlich.

COMBYTE (JSR $E74C): Erst TXTPTR-Stelle auf Komma überprüfen, dann TXTPTR erhöhen und Zahl(enausdruck) auswerten und Ergebnis als Hexzahl in X-Register übertragen. Der Zahlenausdruck muß im Bereich 0-255 liegen, also 2stellige Hexzahl ohne Vorzeichen. COMBYTE endet mit CHRGOT. Vorher JSR CHRGOT nicht erforderlich.

GETBYT (JSR $E6F8): Ähnlich wie COMBYTE, d.h. Zahl(enausdruck) bei TXTPTR-Stelle in X-Register auswerten (0-255). GETBYT beginnt mit FRMNUM, benutzt dann CONINT und endet mit CHRGOT. Vorher JSR CHRGOT nicht erforderlich. Vgl. CONINT.

GETNUM (JSR $E746): Wertet erste(n) Zahl(enausdruck) bei TXTPTR-Stelle als 4stellige Hexzahl ohne Vorzeichen (0-65535) in LINNUM ($0050-$0051) aus, prüft ob Komma folgt, und wertet dann zweite(n) Zahl(enausdruck) nach Komma als 2stellige Hexzahl ohne Vorzeichen (0-255) in X-Register aus. Vorher JSR CHRGOT nicht erforderlich. Zur Erläuterung: GETNUM besteht aus

```
JSR   FRMNUM   Zahlenausdruck in FAC auswerten
JSR   GETADR   FAC in 4stellige Hexzahl in LINNUM umwandeln
JSR   CHKCOM   Kommaprüfung
JMP   GETBYT   Zahlenausdruck in FAC auswerten und dann in X übertra-
               gen
```

4.1.2.2.3. Zahlenkonvertierung (FAC-, Hex- und Dezimalzahlen)

LINPRT ($ED24): 4stellige Hexzahl $HHLL dezimal (0-65535) ausgeben.

vorher: LDX LL
LDA HH

PRNTFAC (JSR $ED2E): FAC dezimal ausgeben. Fließkommazahl vorher in FAC ($009D-$00A2) übertragen. Benutzt FOUT und STROUT.

FOUT (JSR $ED34): Verwandelt FAC in Zahlstring ab $0100-$0110 (FBUFFR) mit Bit 7 off und $00 als Endmarker. Fließkommazahl vorher in FAC übertragen.

nachher: A enthält LL von $0100, d.h. $00
Y enthält HH von $0100, d.h. $10

STROUT ($DB3A): String Output über COUT. String muß ASCII-Zeichen mit Bit 7 off enthalten und entweder mit $00 oder $22 („) enden.

vorher: LDA LL der Stringadresse
LDY HH der Stringadresse

GETADR (JSR $E752): FAC im Bereich 0-65535 in 4stellige Hexzahl ohne Vorzeichen in LINNUM ($0050-$0051) umwandeln. Benutzt QINT. Zahl muß im Bereich 0-65535 liegen!

GIVAYF ($E2F2): 4stellige Hexzahl mit Vorzeichen (0-32767 oder -1 bis -32768) in FAC umwandeln.

vorher: LDY LL
LDA HH

Man beachte, daß es keine Routine gibt, mit der man eine mehr als 4stellige Hexzahl (größer als $FFFF) in FAC umwandeln kann und umgekehrt. Ferner wird eine 4stellige Hexzahl von GIVAYF als „signed hex number" aufgefaßt. Will man eine 4stellige Hexzahl *ohne* Vorzeichen in FAC umwandeln, muß man zu der im Bereich $8000-$FFFF liegenden Hexzahl *nach* GIVAYF 65536 hinzuaddieren. Siehe hierzu unser GIVAYF-Programm.

SNGFLT ($E301): Verwandelt Y ohne Vorzeichen (0-255) in FAC. (SNGFLT setzt A auf 0 und fällt dann in GIVAYF).

vorher: LDY LL Hexzahl ohne Vorzeichen

FLOAT ($EB93): Verwandelt A mit Vorzeichen (0-127 oder -1 bis -128) in FAC.

vorher: LDA LL Hexzahl mit Vorzeichen

CONINT (JSR $E6FB): Verwandelt FAC im Bereich 0-255 (ohne Vorzeichen) in X. FAC muß im angegebenen Bereich liegen. Routine endet mit JMP CHRGOT. Vgl. GETBYT.

nachher: X-Register enthält FAC-Wert als Hexzahl ohne Vorzeichen

AYINT (JSR $E10C): Verwandelt FAC in Integer-FAC mit Vorzeichen. Ähnlich wie Applesoft-INT, doch Low-Level-Routine, die QINT benutzt und voraussetzt, daß FAC < = 2 ↑ 15, d.h. +32767 bis -32768, sonst erfolgt Illegal Quantity Error!

QINT (JSR $EBF2): Verwandelt FAC in Integer-FAC mit Vorzeichen. Setzt voraus, daß FAC < = 2 ↑ 23, d.h. größtmögliche Zahl ist ca. 8388608.

INT (JSR $EC23): Verwandelt FAC in Integer-FAC mit Vorzeichen. Gegenüber AYINT und QINT normale Applesoft-INT-Funktion. Abrundung auf Integer-FAC setzt voraus, daß FAC < = 2 ↑ 31, andernfalls bleibt Exponentialform erhalten!

FAC-Integerzahlen sind nicht mit Integer-Basic-Zahlen zu verwechseln. Eine FAC-Integerzahl ist lediglich eine Zahl ohne Nachkommastellen, die größer als +/- 32767 sein kann.

4.1.2.2.4. Fließkommamathematik-Routinen

Hinweis: Alle nachfolgenden Routinen können Sie ohne Schwierigkeiten anwenden, auch wenn Sie die keine Ahnung haben, wie die Fließkommazahlen von Interpreter intern dargestellt und manipuliert werden.

ROUND (JSR $EB72): Interne Rundungsroutine für FAC, die letztes Bit von

FAC als Binärzahl rundet. Wird z.B. intern aufgerufen, bevor FAC ($009D-$00A2) nach ARG ($00A5-$00AA) oder FAC in den Speicher als gepackte 5-Byte-Zahl übertragen wird. ROUND muß ggf. aufgerufen werden, wenn man eigenen FAC-Übertragungsroutinen schreibt.

Für die nachfolgenden Routinen gilt:

FAC: 6-Byte-Fließkommazahl in $009D-$00A2 (ungepackte Form)
ARG: 6-Byte-Fließkommazahl in $00A5-$00AA (ungepackte Form)
MEM: 5-Byte-Fließkommazahl in Memory (gepackte Form)

MEM ist also der Wert einer Applesoft-Fließkommavariablen.

Viele Routinen müssen durch LDA $009D (= FAC-Exponent) eingeleitet werden, damit die Status-Flags entsprechend gesetzt werden.

MOVMF ($EB2B): FAC nach MEM übertragen (FAC → MEM).

```
vorher:   LDX   LL Adresse von MEM
          LDY   HH Adresse von MEM
          JSR   MOVMF
nachher:  Status-Flags gemäß LDA $009D
```

MOVFM ($EAF9): MEM nach FAC übertragen (MEM → FAC).

```
vorher:   LDA   LL Adresse von MEM
          LDY   HH Adresse von MEM
          JSR   MOVFM
nachher:  Status-Flags gemäß LDA $009D
```

CONUPK ($E9E3): MEM nach ARG moven (MEM → ARG).

```
vorher:   LDA   LL Adresse von MEM
          LDY   HH Adresse von MEM
          JSR   CONUPK
nachher:  Status-Flags gemäß LDA $009D
```

MOVAF (JSR $EB63): FAC nach ARG moven (FAC → ARG). Ruft ROUND vor dem Move auf. Nachher FAC-Wert weiterhin in FAC-Register und Status-Flags gemäß LDA $009D gesetzt.

MOVFA (JSR $B53): ARG nach FAC moven (ARG → FAC). Nachher ARG-Wert weiterhin in ARG-Register und Status-Flags gemäß LDA $009D gesetzt.

Hinweis: Alle nachfolgenden Routinen müssen mit LDA $009D eingeleitet werden, sofern nicht eine der obigen Standard-Move-Routinen benutzt wird. Prozedur im einzelnen:

- Erste Zahl nach ARG moven
  (Summand, Minuend, Multiplikator, Dividend, Basis)
- Zweite Zahl nach FAC moven
  (Summand, Subtrahend, Multiplikand, Divisor, Exponent)
- LDA $009D (FAC-Exponent)
- JSR ROUTINE
- Nachher Ergebnis (Summe, Differenz, Produkt, Quotient, Potenzwert) in FAC

FADD ($E7BE): Addition als CONUPK + FADDT kombiniert, d.h. MEM in ARG übertragen, dann FAC = ARG + FAC als Addition ausführen. Nachher Summe in FAC.

vorher: LDA LL von MEM-Adresse
LDY HH von MEM-Adresse

FADDT ($E7C1): Addition von FAC und ARG, d.h. FAC = ARG + FAC.

vorher: MEM-Summand1 nach ARG
MEM-Summand2 nach FAC
LDA $009D
JSR FADDT
nachher: Summe in FAC

FSUB ($E7A7): Subtraktion als CONUPK + FSUBT kombiniert, d.h. MEM in ARG übertragen, dann FAC = ARG - FAC als Subtraktion ausführen. Nachher Differenz in FAC.

vorher: LDA LL von MEM-Adresse
LDY HH von MEM-Adresse

FSUBT ($E7AA): Subtraktion von ARG minus FAC, d.h. FAC = ARG - FAC.

vorher: MEM-Minuend in ARG
MEM-Subtrahend in FAC
LDA $009D
JSR FSUBT
nachher: Differenz in FAC

FMULT ($E97F): Multiplikation als CONUPK + FMULTT kombiniert, d.h. MEM in ARG übertragen, dann FAC = FAC * ARG als Multiplikation ausführen. Nachher Produkt in FAC. Prozedur wie bei FADD und FSUB.

FMULTT ($E982): Multiplikation von ARG * FAC, d.h. FAC = ARG * FAC, d.h. Produkt = Multiplikator * Multiplikand. Prozedur wie bei FADDT und FSUBT.

FDIV ($EA66): Division als CONUPK + FDIVT kombiniert, d.h. MEM in ARG übertragen, dann FAC = ARG / FAC als Division ausführen. Nachher Quotient in FAC. Prozedur wie bei FADD und FSUB.

FDIVT ($EA69): Division von ARG durch FAC, d.h. FAC = ARG / FAC, d.h. Quotient = Dividend / Divisor. Prozedur wie bei FADDT und FSUBT.

FPWRT ($EE97): Potenzieren von ARG (= Basis, Grundzahl) mit FAC (Exponent, Hochzahl), d.h. FAC = ARG ↑ FAC.

vorher: Basis nach ARG
Exponent nach FAC
LDA $009D
nachher: Potenzwert in FAC

SGN ($EB90): FAC-Signum-Funktion (Vorzeichenermittlung). Nachher FAC = Signum. LDA $009D vorher nicht erforderlich.

nachher: FAC = 1 wenn FAC vorher > 0 war
FAC = 0 wenn FAC vorher = 0 war
FAC = -1 wenn FAC vorher < 0 war

SIGN ($EB82): FAC-Signum-Funktion (Vorzeichenermittlung). Nachher A-Register = Signum (als Hexzahl mit Vorzeichen). LDA $009D vorher nicht erforderlich.

nachher:	A = $01	wenn FAC vorher > 0 war
	A = $00	wenn FAC vorher = 0 war
	A = $FF	wenn FAC vorher < 0 war ($FF = -1)

FCOMP ($EBB2): Vergleich von MEM mit FAC. Nachher Signum als Hexzahl mit Vorzeichen in A-Register.

vorher:	LDA LL	von MEM-Adresse
	LDY HH	von MEM-Adresse
	JSR FCOMP	
nachher:	A = $01	wenn MEM < FAC
	A = $00	wenn MEM = FAC
	A = $FF	wenn MEM > FAC ($FF = -1)

Hinweis: Alle nachfolgenden Routinen NEGOP bis ATN brauchen nicht mit LDA $009D eingeleitet zu werden.

NEGOP (JSR $EED0): Vorzeichen-Umkehrung von FAC, d.h. +FAC = -FAC oder -FAC = +FAC.

ABS (JSR $EBAF): Vorzeichen-Entfernung von FAC, d.h. Absolutwert: FAC = FAC ohne (Negativ)vorzeichen.

RND (JSR $EFAE): Random- oder Zufallszahl erzeugen. Nachher FAC = neue Zufallszahl.

SQR ($EE8D): Square root oder Quadratwurzel aus FAC ziehen: FAC = SQR (FAC). SQR wird berechnet als FAC ↑ 0.5.

LOG ($E941): Natürlichen Logarithmus von FAC ermitteln: FAC = LOGNAT von FAC (e = 2,7182... = Basis).

EXP ($EF09): „Natürliche Potenz": FAC = e ↑ FAC.

Hinweis: Bei den nachfolgenden Winkelfunktionen gilt das Winkelmaß rad = ca. 57 Grad.

SIN ($EFF1): Sinus: FAC = SIN (FAC)

COS ($EFEA): Kosinus: FAC = COS (FAC). Wird berechnet als SIN (FAC + π /Z).

TAN ($F03A): Tangens: FAC = TAN (FAC)

ATN ($F09E): Arkustangens: FAC = ATN (FAC)

MULT16 ($E2B8): Hexadezimale 16-Bit-Multiplikation. Achtung: Dies ist keine Fließkomma-Operation! Produkt darf $FFFF nicht überschreiben!

vorher:	Multiplikand	LL in $0065
	Multiplikand	HH in $0064
	Multiplikator	LL in $00AD
	Multiplikator	HH in $00AE
	JSR MULT16	
nachher:	Produkt LL in X-Register	
	Produkt HH in A-Register	

4.1.2.2.5. Suchroutinen

FNDLIN ($D61A): Diese Routine sucht Speicherstelle des Links einer beliebigen Applesoft-Programmzeile, die zuvor als Hexzahl ohne Vorzeichen in LINNUM gepokt werden muß.

vorher:	LL der Zeilennummer in LINNUM ($0050)
	HH der Zeilennummer in LINNUM+1 ($0051)
	ISR FNDLIN
nachher:	Falls gefunden, Carry-Flag = 1 (BCS) und Speicheradresse von Link in LOWTR LL HH (= $009B-$009C)
	Falls nicht gefunden, Carry-Flag = 0 (BCC) und Speicheradresse von Programmende in LOWTR

PTRGET (JSR $DFE3): Diese Routine liest Variablenname von TXTPTR-Stelle und sucht dann Speicheradresse des *Wertes* der Variablen. Zuvor kein JSR CHRGOT erforderlich, doch muß TXTPTR auf ersten Buchstaben des Variablennamens zeigen.

nachher:	LL der Speicheradresse in A-Register
	HH der Speicheradresse in Y-Register

Falls Variable *nicht* gefunden wurde, wird sie angelegt und auf Null initialisiert, bei Arrays mit DIM (10).

GETARYPT (JSR $F7D9): Diese Routine liest Variablenname eines Arrays von TXTPTR-Stelle und sucht dann die Speicheradresse des Array-Kopfs. Falls Array nicht existiert, erfolgt Fehlermeldung, d.h. Array wird *nicht* angelegt. Zuvor kein JSR CHRGOT erforderlich.

nachher: LL der Speicheradresse in LOWTR ($009B)
HH der Speicheradresse in LOWTR+1 ($009C)

```
10  REM --- LAMPOKE.DEMO -
14  REM ***     DEZHEX     ***
18 A$ = „02F0:A9 4C 8D F5 3 A9 0 8D F6 03 A9 3 8D F7 3 60"
22 A$ = A$ + „ ND7D2G": FOR X = 1 TO  LEN (A$): POKE 511 + X,
    ASC (MID$ (A$,X,1)) + 128: NEXT : POKE 72,0: CALL  - 144
26  CALL 752: REM AMPERSAND
30 A$ = „300:C9 44 F0 22 C9 48 D0 51 A0 FF C8 20 B1 0 F0 2 9 80
    99 0 2 D0 F3 A0 0 20 A7 FF A6 3E A5 3F 20 24 ED 4C 59 3"
34 A$ = A$ + „ ND7D2G": FOR X = 1 TO  LEN (A$): POKE 511 + X,
    ASC ( MID$ (A$,X,1)) + 128: NEXT : POKE 72,0: CALL  - 144
38 A$ = „326:20 B1 0 F0 2E 20 67 DD 20 52 E7 A6 50 86 FE A5 51
    85 FF 20 41 F9 A9 A0 20 ED FD A6 FE A5 FF 20 24 ED A9 A0 20
    ED FD A4 FE A5 FF 10 6 20 F2 E2 20 2E ED A9 8D 20 ED FD
    4C 95 D9"
42 A$ = A$ + „ ND7D2G": FOR X = 1 TO  LEN (A$): POKE 511 + X,
    ASC ( MID$ (A$,X,1)) + 128: NEXT : POKE 72,0: CALL  - 144
46  FOR X =  - 255 TO  + 255 STEP 5: & DX: NEXT : REM DEMO
50  PRINT : PRINT
54  & D72: REM P-REG.
58  & D511: REM PUFFER-1
62  & D - 144: REM ZMODE+GETNUM
66  & HD7D2: REM NEW STATEMENT
70  PRINT : PRINT : PRINT „DISASSEMBLIERT:": PRINT
74 A$ = „2F0LLL ND7D2G": FOR X = 1 TO  LEN (A$): POKE 511 + X,
    ASC ( MID$ (A$,X,1)) + 128: NEXT : POKE 72,0: CALL  - 144
78  PRINT : PRINT : PRINT „ALS HEXDUMP:": PRINT
82 A$ = „2F0.360 ND7D2G": FOR X = 1 TO  LEN (A$): POKE 511 + X,
    ASC ( MID$ (A$,X,1)) + 128: NEXT : POKE 72,0: CALL  - 144
```

```
100  REM --- DATAPOKE.DEMO -
110  REM ***   IIE.64K?    ***
120  DATA 174,129,192,174,129,192,173,179,251,201
130  DATA 6,208,79,174,20,192,48,74,174,22
140  DATA 192,48,69,169,255,141,251,4,142,0
150  DATA 192,142,12,192,142,14,192,32,137,254
160  DATA 32,147,254,32,47,251,32,88,252,32
170  DATA 234,3,162,5,142,0,4,141,9,192
180  DATA 165,0,162,1,134,0,162,0,166,0
190  DATA 224,1,208,5,162,160,142,0,4,133
200  DATA 0,141,8,192,173,0,4,201,5,240
210  DATA 1,96,76,221,251,0
220  RESTORE
230  FOR X = 768 TO 863: READ Y: POKE X,Y: NEXT
240  REM 64K-KARTE VORHANDEN?
250  CALL 768: IF  PEEK (1024) = 5 THEN  PRINT „64K-KARTE FEHLT!":
     END

260  PRINT „64K-KARTE VORHANDEN"
```

```
100  REM ---EINZELPOKE-DEMO-
110  REM *** MOVE = $FE2C ***
120  POKE 768,216: POKE 769,160: POKE 770,0: POKE 771,76:
    POKE 772,44: POKE 773,254: REM  CLD LDY #$00 JMP FE2C
130 SB = 8192: REM SOURCE-BEGINN:$2000
140 SE = 16383: REM  SOURCE-ENDE:$3FFF
150 ZB = 16384: REM  ZIELADRESSE-BEGINN:$4000
160  POKE 60,SB -  INT (SB / 256) * 256: POKE 61, INT (SB / 256):
    REM A1L-A1H=$3C-$3D
170  POKE 62,SE -  INT (SE / 256) * 256: POKE 63, INT (SE / 256):
    REM A2L-A2H=$3E-$3F
180  POKE 66,ZB -  INT (ZB / 256) * 256: POKE 67, INT (ZB / 256):
     REM A4L-A4H=$42-$43
190  CALL 768: REM   MOVE GRAFIK-SEITE 1 NACH GRAFIK-SEITE 2
```

```
                        1               ORG  $2F0
                        2    *
                        3    * DEZHEX
                        4    * ======
                        5    *
                        6    * &H + Hexzahl wird Dezzahl
                        7    *
                        8    * z.B. &HFFFF
                        9    *
                       10    * &D + Dezzahl wird Hexzahl
                       11    *
                       12    * z.B. &D65535
                       13    *
                       14    * Ampersand-Befehl sowohl im
                       15    * Direkt- als auch im RUN-Modus
                       16    * möglich.
                       17    *
                       18    A2L       EQU  $3E
                       19    A2H       EQU  $3F
                       20    LINNUM    EQU  $50          ;-$51
                       21    CHRGET    EQU  $B1
                       22    HEXL      EQU  $FE
                       23    HEXH      EQU  $FF
                       24    PUFFER    EQU  $0200        ;-$02FF
                       25    FRMNUM    EQU  $DD67        ;nach &
                       26    GETADR    EQU  $E752        ;FAC>LINNUM
                       27    LINPRT    EQU  $ED24        ;X=L,A=H
                       28    PRNTFAC   EQU  $ED2E
                       29    GIVAYF    EQU  $E2F2        ;Y=L,A=H
                       30    PRNTAX    EQU  $F941        ;X=L,A=H
                       31    COUT      EQU  $FDED
                       32    GETNUM    EQU  $FFA7        ;Monitor
                       33    DATA      EQU  $D995
                       34    *
                       35    * Ampersand-Vektor setzen
                       36    *
02F0: A9 4C            37    AMPER     LDA  #$4C         ;JMP
02F2: 8D F5 03         38              STA  $3F5
02F5: A9 00            39              LDA  #<START
02F7: 8D F6 03         40              STA  $3F6
02FA: A9 03            41              LDA  #>START
02FC: 8D F7 03         42              STA  $3F7
02FF: 60               43              RTS
                       44    *
                       45    * &D Dezzahl oder &H Hexzahl?
                       46    *
0300: C9 44            47    START     CMP  #'D'
0302: F0 22            48              BEQ  DEZHEX1
0304: C9 48            49              CMP  #'H'
0306: D0 51            50              BNE  EXIT1
                       51    *
                       52    * HEX in DEZ
                       53    * ----------
```

```
                        54   *
                        55   * Hexzahl über Chrget vom
                        56   * Puffer ab $0201 holen und
                        57   * mit Bit 7 on ab $0200 wieder
                        58   * im Puffer ablegen (bis zum
                        59   * Endmarker $00).
                        60   *
0308: A0 FF             61   HEXDEZ1  LDY  #$FF        ;Wrap
030A: C8                62   HEXDEZ2  INY
030B: 20 B1 00          63            JSR  CHRGET
030E: F0 02             64            BEQ  HEXDEZ3
0310: 09 80             65            ORA  #$80
0312: 99 00 02          66   HEXDEZ3  STA  PUFFER,Y
0315: D0 F3             67            BNE  HEXDEZ2
                        68   *
                        69   * Hex-String durch Monitor-
                        70   * GETNUM-Routine in A2L-A2H
                        71   * ablegen.
                        72   *
0317: A0 00             73            LDY  #0
0319: 20 A7 FF          74            JSR  GETNUM
                        75   *
                        76   * Hexzahl X=LL A=HH als Dezzahl
                        77   * ohne Vorzeichen ausgeben
                        78   * (0-65535)
                        79   *
031C: A6 3E             80            LDX  A2L          ;LL
031E: A5 3F             81            LDA  A2H          ;HH
0320: 20 24 ED          82   HEXDEZ4  JSR  LINPRT
0323: 4C 59 03          83            JMP  EXIT1
                        84   *
                        85   * DEZ in HEX
                        86   * ----------
                        87   *
0326: 20 B1 00          88   DEZHEX1  JSR  CHRGET      ;nach D
0329: F0 2E             89            BEQ  EXIT1
                        90   *
                        91   * Dezzahl erst in FAC und dann
                        92   * FAC in X-LL A-HH umwandeln
                        93   * und zunächst als Hex anzeigen
                        94   *
032B: 20 67 DD          95            JSR  FRMNUM       ;in FAC
032E: 20 52 E7          96            JSR  GETADR       ;FAC>LIN
0331: A6 50             97            LDX  LINNUM
0333: 86 FE             98            STX  HEXL
0335: A5 51             99            LDA  LINNUM+1
0337: 85 FF             100           STA  HEXH
0339: 20 41 F9          101           JSR  PRNTAX       ;A+X
                        102  *
                        103  * Nun als Dezzahl ohne Vorzeichen
                        104  * anzeigen
                        105  *
033C: A9 A0             106           LDA  #$A0
033E: 20 ED FD          107           JSR  COUT
```

```
0341: A6 FE      108           LDX  HEXL
0343: A5 FF      109           LDA  HEXH
0345: 20 24 ED   110           JSR  LINPRT
                 111  *
                 112  * Falls Hexzahl im Bereich
                 113  * $8000-$FFFF ist, ferner
                 114  * als Dezzahl mit Vorzeichen
                 115  * anzeigen.
                 116  *
0348: A9 A0      117           LDA  #$A0
034A: 20 ED FD   118           JSR  COUT
034D: A4 FE      119           LDY  HEXL        ;Y=LL
034F: A5 FF      120           LDA  HEXH        ;A=HH
0351: 10 06      121           BPL  EXIT1
0353: 20 F2 E2   122           JSR  GIVAYF      ;float
0356: 20 2E ED   123           JSR  PRNTFAC
                 124  *
0359: A9 8D      125  EXIT1    LDA  #$8D
035B: 20 ED FD   126           JSR  COUT
                 127  *
                 128  * DATA springt zum nächsten
                 129  * Statement eines Applesoft-
                 130  * Programms, also zum Doppel-
                 131  * punkt oder zum Zeilenende.
                 132  *
035E: 4C 95 D9   133           JMP  DATA
```

4.2.3. INVERTER

```
                   1           ORG  $0300
                   2  *
                   3  * INVERTER
                   4  * ========
                   5  *
                   6  * Invertiert HGR1 (oder HGR2)
                   7  *
                   8  IND       EQU  $FE         ;-$FF
                   9  *
0300: A0 00        10  INVERT    LDY  #0
0302: 84 FE        11           STY  IND
0304: A9 20        12           LDA  #>$2000     ;S2:$4000
0306: 85 FF        13           STA  IND+1
0308: A2 40        14           LDX  #>$4000     ;S2:$6000
030A: B1 FE        15  INVERT1  LDA  (IND),Y
030C: 49 FF        16           EOR  #%11111111
030E: 91 FE        17           STA  (IND),Y
0310: C8           18           INY
0311: D0 F7        19           BNE  INVERT1
0313: E6 FF        20           INC  IND+1
0315: E4 FF        21           CPX  IND+1
0317: D0 F1        22           BNE  INVERT1
0319: 60           23           RTS
```

```
                        1               ORG  $300
                        2   *
                        3   * IIE.64K?
                        4   * ========
                        5   *
                        6   * Dieser Test prüft, ob 64K-
                        7   * Karte existiert. Während des
                        8   * Tests wird auf 40 Z/Z um-
                        9   * geschaltet. Nur für IIe gedacht!
                       10   * Test funktioniert nur, wenn
                       11   * sich Testprogramm in dem
                       12   * unteren $0200-$BFFF-Bereich
                       13   * befindet!
                       14   *
                       15   * Language Card abschalten
                       16   *
0300: AE 81 C0         17   LCOFF    LDX  $C081      ;WRBK2
0303: AE 81 C0         18            LDX  $C081      ;RDROM
                       19   *
                       20   * Apple IIe?
                       21   *
0306: AD B3 FB         22   IIE?     LDA  $FBB3
0309: C9 06            23            CMP  #$06
030B: D0 4F            24            BNE  ERROR1     ;NO IIE!
                       25   *
                       26   * WRAUX oder AUXZP?
                       27   *
030D: AE 14 C0         28   AUX?     LDX  $C014
0310: 30 4A            29            BMI  ERROR1     ;WRAUX!
0312: AE 16 C0         30            LDX  $C016
0315: 30 45            31            BMI  ERROR1     ;AUXZP!
                       32   *
                       33   * 80-Zeichenkarte abschalten
                       34   *
0317: A9 FF            35   OFF80    LDA  #$FF
0319: 8D FB 04         36            STA  $04FB      ;MODE
031C: 8E 00 C0         37            STX  $C000      ;80OFF
031F: 8E 0C C0         38            STX  $C00C      ;40COL
0322: 8E 0E C0         39            STX  $C00E      ;ALTOFF
0325: 20 89 FE         40            JSR  $FE89      ;SETKBD
0328: 20 93 FE         41            JSR  $FE93      ;SETVID
032B: 20 2F FB         42            JSR  $FB2F      ;TEXT
032E: 20 58 FC         43            JSR  $FC58      ;HOME
                       44   *
                       45   * Für DOS 3.3   JSR $03EA
                       46   * Für ProDOS    JSR $9A17
                       47   *
0331: 20 EA 03         48            JSR  $03EA      ;CONNECT
                       49   *
                       50   * 64K-Karte?
                       51   *
                       52   * Wenn Peek (1024) = 5, nein!
                       53   *
```

```
0334: A2 05     54   CARD64?  LDX  #$05       ;'E'
0336: 8E 00 04  55            STX  $0400      ;1024
                56   *
                57   * Existenz der 64K-Karte wird
                58   * anhand der Speicherstelle
                59   * $0000:01 der Karte überprüft
                60   *
0339: 8D 09 C0  61            STA  $C009      ;AUXZP
033C: A5 00     62            LDA  $00        ;SAVE
                63   *
                64   * 1 = 1 ?
                65   *
033E: A2 01     66            LDX  #$01
0340: 86 00     67            STX  $00
0342: A2 00     68            LDX  #$00
0344: A6 00     69            LDX  $00
0346: E0 01     70            CPX  #$01
0348: D0 05     71            BNE  CARDN0
034A: A2 A0     72            LDX  #$A0       ;' '
034C: 8E 00 04  73            STX  $0400      ;1024
034F: 85 00     74   CARDN0   STA  $00        ;RESTORE
0351: 8D 08 C0  75            STA  $C008      ;MAINZP
0354: AD 00 04  76            LDA  $0400
0357: C9 05     77            CMP  #$05
0359: F0 01     78            BEQ  ERROR1
035B: 60        79            RTS
035C: 4C DD FB  80   ERROR1   JMP  $FBDD      ;Bell

--End assembly-

95 bytes

Errors: 0
```

```
                        1               ORG  $300
                        2    *
                        3    * TXTPTR
                        4    * ======
                        5    *
                        6    * Dieses Demo zeigt den
                        7    * momentanen TXTPTR sowie
                        8    * den Inhalt der Register
                        9    * nach &, CALL und USR
                       10    *
                       11    TXTPTR    EQU  $00B8       ;-$00B9
                       12    COUT      EQU  $FDED
                       13    PRNTAX    EQU  $F941
                       14    REGDSP    EQU  $FAD7       ;AXYPS
                       15    IOSAVE    EQU  $FF4A
                       16    DATA      EQU  $D995
                       17    *
                       18    ***************
                       19    *
                       20    * JMP INIT wird modifiziert
                       21    *
0300: 4C 03 03         22    CALL768   JMP  INIT
                       23    *
                       24    * &-Vektor setzen
                       25    *
0303: A9 4C            26    INIT      LDA  #$4C        ;JMP
0305: 8D F5 03         27              STA  $03F5
0308: A9 29            28              LDA  #<DISPLAY
030A: 8D F6 03         29              STA  $03F6
030D: A9 03            30              LDA  #>DISPLAY
030F: 8D F7 03         31              STA  $03F7
                       32    *
                       33    * USR-Vektor setzen
                       34    *
0312: A9 4C            35              LDA  #$4C        ;JMP
0314: 85 0A            36              STA  $000A
0316: A9 29            37              LDA  #<DISPLAY
0318: 85 0B            38              STA  $000B
031A: A9 03            39              LDA  #>DISPLAY
031C: 85 0C            40              STA  $000C
                       41    *
                       42    * CALL 768 initialisieren
                       43    *
031E: A9 29            44              LDA  #<DISPLAY
0320: 8D 01 03         45              STA  CALL768+1
0323: A9 03            46              LDA  #>DISPLAY
0325: 8D 02 03         47              STA  CALL768+2
0328: 60               48              RTS
                       49    *
0329: 20 2F 03         50    DISPLAY   JSR  DISPLAY1
032C: 20 95 D9         51              JSR  DATA
                       52    *
032F: 20 4A FF         53    DISPLAY1  JSR  IOSAVE
```

```
0332: 20 D7 FA  54            JSR  REGDSP
0335: A2 00     55            LDX  #0
0337: BD 49 03  56   TXTPTR1  LDA  TXTPTR3,X
033A: F0 06     57            BEQ  TXTPTR2
033C: 20 ED FD  58            JSR  COUT
033F: E8        59            INX
0340: D0 F5     60            BNE  TXTPTR1
0342: A5 B9     61   TXTPTR2  LDA  TXTPTR+1
0344: A6 B8     62            LDX  TXTPTR
0346: 4C 41 F9  63            JMP  PRNTAX
                64   *
0349: 8D        65   TXTPTR3  HEX  8D
034A: A0 D4 D8  66            ASC   5 TXTPTR: $"
034D: D4 D0 D4 D2 BA A0 A4
0354: 00        67            HEX  00
                68   *
                69   **************
                70   *
                71   * CHRGET-Routine
                72   * ==============
                73   *
                74            ORG  $00B1
                75   *
                76   * TXTPTR um eins erhöhen
                77   *
00B1: E6 B8     78   CHRGET   INC  CHRGOT+1
00B3: D0 02     79            BNE  CHRGOT
00B5: E6 B9     80            INC  CHRGOT+2
                81   *
                82   * Adresse nach CHRGOT wird durch
                83   * CHRGET ständig modifiziert
                84   *
                85   * Im RUN-Modus zeigt TXTPTR
                86   * auf die jeweilige Programm-
                87   * stelle, im Direkt-Modus auf
                88   * den Tastaturpuffer ab $0200
                89   *
                90   * Im RUN-Modus wird der TXTPTR
                91   * bei Input sowie bei READ von
                92   * DATA-Statements vorübergehend
                93   * auf den Puffer bzw. auf die
                94   * jeweilige Zeile der DATA-
                95   * Statements abgeändert.
                96   *
00B7: AD 01 08  97   CHRGOT   LDA  $0801      ;TXTPTR
                98   *
                99   * Dez 0-9 = Hex $30-$39; ':' = $3A
               100   * Wenn A > = ':', dann Carry-Flag
               101   * gesetzt, mithin keine Ziffer.
               102   * Ferner wenn A = ':', dann
               103   * Zero-Flag gesetzt.
               104   *
00BA: C9 3A    105            CMP  #':'      ;nach 9
00BC: B0 0A    106            BCS  RETURN    ;>=':'
               107   *
```

```
                    108  * Wenn Leertaste, dann nächstes
                    109  * Zeichen holen.
                    110  *
00BE: C9 20         111          CMP  #$20       ;Space
00C0: F0 EF         112          BEQ  CHRGET     ;ignore
                    113  *
                    114  * Nehmen wir an, A sei $30,
                    115  * dann gilt für C (Carry) und A
                    116  *
                    117  * $30 - $30 = A = $00 ; C = set
                    118  * $00 - $D0 = A = $30 ; C = clear
                    119  *
                    120  * Die 2. Subtraktion führt also
                    121  * stets zu einem Überlauf, be-
                    122  * wirkt jedoch erstens, daß
                    123  * der Ausgangs-Akkumulator-Wert
                    124  * wieder im Akku ist und daß
                    125  * zweitens C = 0 = clear ist,
                    126  * wenn A = Ziffer $30-$39.
                    127  *
                    128  * Ferner ist Zeroflag = 1 = set,
                    129  * wenn A <> $00 = End of line.
                    130  *
00C2: 38            131          SEC             ;1.Subt.
00C3: E9 30         132          SBC  #$30       ;'0'
00C5: 38            133          SEC             ;2.Subt.
00C6: E9 D0         134          SBC  #$D0
                    135  *
                    136  * Zusammenfassend gilt
                    137  * nach CHRGET oder CHRGOT:
                    138  *
                    139  * A = TXTPTR-Stelle
                    140  *     oder ':' bzw. $00
                    141  *
                    142  * Z-Flag = 1, wenn A = ':' oder EOL
                    143  * C-Flag = 0, wenn A = Ziffer
                    144  *
00C8: 60            145  RETURN   RTS

--End assembly-

109 bytes

Errors: 0
```

```
10  PRINT  CHR$ (4)„BRUN TXTPTR"
20  CALL 768: PRINT
30  & : PRINT
40 X =  USR (X): PRINT
```

```
                    1               ORG  $300
                    2    *
                    3    * CHRGET.TRACER
                    4    * ==============
                    5    *
                    6    CHRGET     EQU  $00B1
                    7    CHRGOT     EQU  $00B7
                    8    TXTPTR     EQU  $00B8      -$00B9
                    9    RESET      EQU  $03F2
                    10   MONITOR    EQU  $FF59
                    11   *
                    12   * CHRGET umbiegen auf TRACER
                    13   *
0300: A9 4C         14              LDA  #$4C       ;JMP
0302: 85 B1         15              STA  CHRGET
0304: A9 1C         16              LDA  #<TRACER1
0306: 85 B2         17              STA  CHRGET+1
0308: A9 03         18              LDA  #>TRACER1
030A: 85 B3         19              STA  CHRGET+2
                    20   *
                    21   * Reset-Vektor sicherheits-
                    22   * halber auf Monitor $FF59 setzen
                    23   *
030C: A9 59         24              LDA  #<MONITOR
030E: 8D F2 03      25              STA  RESET
0311: A9 FF         26              LDA  #>MONITOR
0313: 8D F3 03      27              STA  RESET+1
0316: 49 A5         28              EOR  #$A5
0318: 8D F4 03      29              STA  RESET+2
031B: 60            30              RTS
                    31   *
                    32   * Register retten
                    33   *
031C: 8D 88 03      34   TRACER1    STA  ASAVE
031F: 8E 89 03      35              STX  XSAVE
0322: 8C 8A 03      36              STY  YSAVE
                    37   *
                    38   * Die letzten 6 Txtptr nach
                    39   * links schieben und den neuen
                    40   * 7. Txtptr hinten anhängen.
                    41   *
0325: A0 00         42              LDY  #0
0327: A2 01         43              LDX  #1
0329: BD 92 03      44   TRACER2    LDA  PTRH,X
032C: 99 92 03      45              STA  PTRH,Y
032F: BD 8B 03      46              LDA  PTRL,X
0332: 99 8B 03      47              STA  PTRL,Y
0335: E8            48              INX
0336: C8            49              INY
0337: C0 06         50              CPY  #6
0339: D0 EE         51              BNE  TRACER2
033B: A5 B8         52              LDA  TXTPTR
033D: 99 8B 03      53              STA  PTRL,Y
```

```
0340: A5 B9     54           LDA  TXTPTR+1
0342: 99 92 03  55           STA  PTRH,Y
                56   *
                57   * Die letzten 7 Txtptr in die
                58   * 1. Bildschirmzeile poken
                59   *
0345: A0 00     60           LDY  #0          ;CH
0347: A2 06     61           LDX  #6
0349: BD 92 03  62   TRACER3 LDA  PTRH,X
034C: 20 70 03  63           JSR  HEXOUT
034F: BD 8B 03  64           LDA  PTRL,X
0352: 20 70 03  65           JSR  HEXOUT
0355: A9 A0     66           LDA  #$A0        ;Space
0357: 99 00 04  67           STA  $0400,Y
035A: C8        68           INY
035B: CA        69           DEX
035C: 10 EB     70           BPL  TRACER3
                71   *
                72   * Register herstellen, Txtptr
                73   * erhöhen und Exit nach Chrget
                74   *
035E: AD 88 03  75           LDA  ASAVE
0361: AE 89 03  76           LDX  XSAVE
0364: AC 8A 03  77           LDY  YSAVE
0367: E6 B8     78           INC  TXTPTR
0369: D0 02     79           BNE  TRACER4
036B: E6 B9     80           INC  TXTPTR+1
036D: 4C B7 00. 81   TRACER4 JMP  CHRGOT      ;Exit
                82   *
                83   * A als Hexzahl anzeigen
                84   *
0370: 48        85   HEXOUT  PHA              ;linkes
0371: 4A        86           LSR
0372: 4A        87           LSR
0373: 4A        88           LSR
0374: 4A        89           LSR
0375: 20 7B 03  90           JSR  HEXOUT3
0378: 68        91           PLA              ;rechtes
                92   *
0379: 29 0F     93   HEXOUT2 AND  #%00001111 ;$0F
037B: 09 B0     94   HEXOUT3 ORA  #"0"
037D: C9 BA     95           CMP  #":"        ;nach 9
037F: 90 02     96           BCC  HEXOUT4
0381: 69 06     97           ADC  #$06
0383: 99 00 04  98   HEXOUT4 STA  $0400,Y     ;V1,H1
0386: C8        99           INY              ;Htab+1
0387: 60        100          RTS
                101  *
0388: 00        102  ASAVE   HEX  00
0389: 00        103  XSAVE   HEX  00
038A: 00        104  YSAVE   HEX  00
                105  *
                106  PTRL     DS   7          ;0-6
                107  PTRH     DS   7          ;0-6
```

```
1               ORG  $300
2     *
3     * FNDLIN
4     * ======
5     *
6     * & + Zeilennummer, z.B. &10,
7     *
8     * zeigt Link-Speicherstelle
9     * der Applesoft-Zeile an. Dies
10    * ist die hexadezimale Adresse
11    * der Speicherstelle des Beginns
12    * der nächsten Applesoft-Zeile.
13    * Vor dem Link steht stets Eol
14    * = $00 als Zeilen-Endmarker.
15    * Dies gilt auch für den
16    * Programm-Beginn bei $0800.
17    * Am Programm-Ende finden sich
18    * hingegen 3 Eol-Zeichen. Danach
19    * folgen die Variablen.
20    *
21    LINNUM   EQU  $50        ;-$51
22    LOWTR    EQU  $9B        ;-$9C
23    *
24    FRMEVL   EQU  $DD7B      ;>FAC
25    GETADR   EQU  $E752      ;>LINNUM
26    FNDLIN   EQU  $D61A      ;>LOWTR
27    PRNTAX   EQU  $F941      ;A,X
28    HEXOUT   EQU  $FDDA      ;A>HEX
29    PRINT    EQU  $FDED      ;A>ASCII
30    *
31    *
32    * Applesoft-Beispiel:
33    * -------------------
34    *
35    * 10 HOME
36    * 20 PRINT
37    * 30 X% = 258
38    *
39    * Nach RUN sieht Applesoft-Programm
40    * speicherintern so aus:
41    *
42    * 0800: 00
43    *
44    *       1.Byte des Programms 00
45    * &10
46    * 0801: 07  08  0A  00  97   00
47    *
48    *       $0807   $000A   Home Eol
49    *               (10)
50    *       Link    Nr.d.Z. Token
51    * &20
52    * 0807: 0D  08  14  00  BA  00
53    *
```

```
                        *       $080D   $0014   ?   Eol
                        *               (20)
                        *
                        * &30
                        * 080D: 18  08  1E  00
                        *
                        *       $0818   $001E
                        *               (30)
                        *
                        *       58  25  D0  32  35  38  00
                        *
                        *       X   %   =   2   5   8
                        *
                        * &40
                        * 0818: 00  00  00
                        *
                        *       Programmende.
                        *
                        * 081B: D8  80  01  02
                        *
                        *       X%  --  $0102
                        *       V1  V2  (258; HH-LL!)
                        *
                        * Das „%" ist in $D8 „versteckt".
                        * Der Variablenname nimmt stets
                        * 2 Bytes ein. Fehlt wie hier der
                        * zweite Buchstabe des Namens,
                        * dann wird die entsprechende
                        * Stelle mit $80 aufgefüllt.
                        *
                        * Ampersand-Vektor setzen
                        *
0300: A9 4C             AMPER   LDA  #$4C       ;JMP
0302: 8D F5 03                  STA  $3F5       ;Vektor
0305: A9 10                     LDA  #<START
0307: 8D F6 03                  STA  $3F6
030A: A9 03                     LDA  #>START
030C: 8D F7 03                  STA  $3F7
030F: 60                        RTS
                        *
                        * 1. Mit FRMEVL Zeilennummer
                        *    nach & auswerten
                        * 2. Dann mit GETADR in Hex
                        *    umwandeln in LINNUM
                        * 3. Dann mit FNDLIN Zeile,
                        *    die in LINNUM steht, suchen
                        *
                        * a) Falls gefunden, Carry
                        *    setzen und Link-Byte-
                        *    Speicherstelle in
                        *    LOWTR ablegen.
                        *
                        * b) Falls nicht gefunden,
                        *    Carry zurücksetzen und
```

```
                    108  *    Speicherstelle vom
                    109  *    Programmende anzeigen.
                    110  *
                    111  *
0310: 20 7B DD      112  START    JSR  FRMEVL      ;Zahl>FAC
0313: 20 52 E7      113           JSR  GETADR      ;FAC>LINNUM
0316: 20 1A D6      114           JSR  FNDLIN      ;>LOWTR
0319: 08            115           PHP              ;Carry
031A: 68            116           PLA              ;retten
031B: 8D 4E 03      117           STA  STATUS
                    118  *
                    119  * X-Reg mit Lowbyte
                    120  * A-Reg mit Highbyte
                    121  * laden und Adresse als
                    122  * Hex-Doppelbyte ausgeben.
                    123  *
031E: A6 9B         124  GEFUNDEN LDX  LOWTR
0320: A5 9C         125           LDA  LOWTR+1
0322: 20 41 F9      126  HEXWERT  JSR  PRNTAX
0325: A9 BA         127           LDA  #„:"
0327: 20 ED FD      128           JSR  PRINT
032A: AD 4E 03      129           LDA  STATUS
032D: 48            130           PHA              ;Carry
032E: 28            131           PLP              ;holen
032F: B0 05         132           BCS  HEXDUMP
0331: A9 BF         133           LDA  #„?"
0333: 4C ED FD      134           JMP  PRINT
                    135  *
                    136  * Zeile in Hex-Form anzeigen
                    137  *
0336: A0 00         138  HEXDUMP  LDY  #0
0338: B1 9B         139  HEXDUMP1 LDA  (LOWTR),Y
033A: D0 04         140           BNE  HEXDUMP2
                    141  *
                    142  * HEX $00 bei den ersten 4
                    143  * Bytes ignorieren
                    144  *
033C: C0 04         145           CPY  #4          ;0.-3.Hex
033E: B0 0B         146           BCS  EXIT
0340: 20 DA FD      147  HEXDUMP2 JSR  HEXOUT
0343: A9 A0         148           LDA  #$A0
0345: 20 ED FD      149           JSR  PRINT
0348: C8            150           INY
0349: D0 ED         151           BNE  HEXDUMP1
034B: 4C DA FD      152  EXIT     JMP  HEXOUT
                    153  *
034E: 00            154  STATUS   HEX  00


--End assembly-

79 bytes

Errors: 0
```

```
                        1               ORG  $0300
                        2    *
                        3    * PTRGET
                        4    * ======
                        5    *
                        6    * PTRGET sucht Speicherstelle
                        7    * des Wertes der Variablen nach
                        8    *
                        9    * z.B. & X
                        10   *       & X%
                        11   *       & X$
                        12   *       & X  (100)
                        13   *       & X% (100)
                        14   *       & X$ (100)
                        15   *
                        16   * und legt Speicheradresse in
                        17   *
                        18   * A = Low Byte
                        19   * Y = High Byte ab
                        20   *
                        21   * Warnung: Variable muß bereits
                        22   * angelegt sein. Wenn nicht,
                        23   * wird sie nämlich angelegt,
                        24   * und zwar bei DIM-Variablen
                        25   * mit DIM Variable (10).
                        26   *
                        27   PTRGET   EQU  $DFE3
                        28   VARNAM   EQU  $81         ;-$82
                        29   *
                        30   * DATA = Txtptr bis zum nächsten
                        31   * Statement-„:" oder Eol-„00"
                        32   * vorrücken.
                        33   *
                        34   DATA     EQU  $D995
                        35   *
                        36   PRNTAX   EQU  $F941       ;A-X
                        37   HEXOUT   EQU  $FDDA       ;A
                        38   PRINT    EQU  $FDED       ;A
                        39   *
                        40   STRPTRL  EQU  $FA
                        41   STRPTRH  EQU  $FB
                        42   LOWADR   EQU  $FC
                        43   HIGHADR  EQU  $FD
                        44   STRFLAG  EQU  $FF
                        45   LENGTH   EQU  $FE
                        46   *-------------------------------
                        47   *
0300: A9 4C             48   AMPER    LDA  #$4C
0302: 8D F5 03          49            STA  $3F5
0305: A9 10             50            LDA  #<POINTER
0307: 8D F6 03          51            STA  $3F6
030A: A9 03             52            LDA  #>POINTER
030C: 8D F7 03          53            STA  $3F7
```

```
030F: 60          54            RTS
                  55   *
                  56   * Pointer als $HHLL anzeigen
                  57   *
0310: 20 E3 DF    58   POINTER  JSR  PTRGET
0313: 85 FC       59            STA  LOWADR
0315: 84 FD       60            STY  HIGHADR
0317: A9 A4       61            LDA  #„$"
0319: 20 ED FD    62            JSR  PRINT
031C: A5 FD       63            LDA  HIGHADR
031E: A6 FC       64            LDX  LOWADR
0320: 20 41 F9    65            JSR  PRNTAX
                  66   *
                  67   * Die Bytes 0-6 der Variablen
                  68   * als Hexzahlen anzeigen.
                  69   *
0323: A9 BA       70            LDA  #„:"
0325: 20 ED FD    71            JSR  PRINT
                  72   *
                  73   * Nach JSR PTRGET finden sich
                  74   * die ersten zwei Zeichen des
                  75   * Variablennamens in VARNAM
                  76   *
0328: A0 00       77            LDY  #0          ;Y=0
032A: 84 FF       78            STY  STRFLAG
                  79   *
032C: A2 02       80            LDX  #2          ;X=2
032E: 24 81       81            BIT  VARNAM
0330: 10 02       82            BPL  NONINT
0332: 30 0B       83            BMI  LAENGE
0334: E8          84   NONINT   INX              ;X=3
0335: 24 82       85            BIT  VARNAM+1
0337: 10 04       86            BPL  REAL
0339: E6 FF       87            INC  STRFLAG
033B: D0 02       88            BNE  LAENGE
033D: E8          89   REAL     INX
033E: E8          90            INX              ;X=5
033F: 86 FE       91   LAENGE   STX  LENGTH
                  92   *
                  93   * Wert in Hex anzeigen,
                  94   * bei String nur Pointer
                  95   *
0341: B1 FC       96   WERT     LDA  (LOWADR),Y
0343: 20 DA FD    97            JSR  HEXOUT
0346: A9 A0       98            LDA  #$A0        ;Space
0348: 20 ED FD    99            JSR  PRINT
034B: C8         100            INY
034C: C4 FE      101            CPY  LENGTH
034E: D0 F1      102            BNE  WERT
                 103   *
                 104   * String in Hex anzeigen
                 105   *
0350: A5 FF      106            LDA  STRFLAG
0352: F0 23      107            BEQ  EXIT        ;No Str!
```

```
0354: A0 00      108           LDY  #0
0356: B1 FC      109           LDA  (LOWADR),Y
0358: 85 FE      110           STA  LENGTH
035A: F0 1B      111           BEQ  EXIT        ;Nullstr!
035C: C8         112           INY
035D: B1 FC      113           LDA  (LOWADR),Y
035F: 85 FA      114           STA  STRPTRL
0361: C8         115           INY
0362: B1 FC      116           LDA  (LOWADR),Y
0364: 85 FB      117           STA  STRPTRH
0366: A0 00      118           LDY  #0
0368: A9 A0      119 STRPRINT  LDA  #$A0
036A: 20 ED FD   120           JSR  PRINT
036D: B1 FA      121           LDA  (STRPTRL),Y
036F: 20 DA FD   122           JSR  HEXOUT
0372: C8         123           INY
0373: C4 FE      124           CPY  LENGTH
0375: D0 F1      125           BNE  STRPRINT
0377: A9 8D      126 EXIT      LDA  #$8D        ;Return
0379: 20 ED FD   127           JSR  PRINT
037C: 4C 95 D9   128           JMP  DATA
                 129 *
                 130 *-------------------------------
                 131 *
                 132 * Applesoft-Variablen-Aufbau
                 133 * ==========================
                 134 *
                 135 * Beachte: Einfache Variablen
                 136 * umfassen stets 7 Bytes,
                 137 * numeriert von Byte 00-06.
                 138 * Eindimensionale Variablen
                 139 * haben stets einen 7 Bytes
                 140 * langen Kopf von Byte 00-06.
                 141 * Danach folgen die eigentlichen
                 142 * Variablen von 0 - N.
                 143 *
                 144 * Das zweite Zeichen eines
                 145 * Variablennamens kann fehlen
                 146 * und ist dann $00 (Bit 7 off)
                 147 * oder $80 (Bit 7 on). Das
                 148 * dritte und jedes weitere
                 149 * Zeichen des Variablennamens
                 150 * wird speicherintern ignoriert.
                 151 *
                 152 * Bei Strings werden nur die
                 153 * Pointer und das Länge-Byte
                 154 * abgespeichert, während die
                 155 * eigentlichen Strings ab HIMEM
                 156 * abwärts mit Bit 7 off abgelegt
                 157 * werden z.B. String „ABCDEF":
                 158 *
                 159 * $95FA: 41 42 43 44 45 46
                 160 * $95FA: A  B  C  D  E  F
                 161 *
```

```
162  * Bei dimensionierten Variablen
163  * ist der Offset LL-HH der
164  * Abstand zwischen dem momentanen
165  * DIM-Variablenkopf und dem
166  * nächsten DIM-Variablenkopf.
167  *
168  * PTRGET zeigt bei DIM-Variablen
169  * niemals zum Kopf, sondern stets
170  * zum Wert N selbst.
171  *
172  * (Mehrdimensionale Variablen
173  * und DEF-Variablen sind nach-
174  * folgend ignoriert.)
175  *
176  * Einfache Variablen
177  * ==================
178  *
179  * a) Real (bis +/- 1 * 10 ↑ 38)
180  * --------
181  *
182  * 00: 1. Zeichen Name Bit 7 off
183  * 01: 2. Zeichen Name Bit 7 off
184  * 02: 1. Zahl-Byte: Exponent
185  * 03: 2. Zahl-Byte: Mantisse HH
186  * 04: 3. Zahl-Byte: Mantisse MM
187  * 05: 4. Zahl-Byte: Mantisse MM
188  * 06: 5. Zahl-Byte: Mantisse LL
189  *
190  * b) Integer (bis +/- 32767)
191  * ----------
192  *
193  * 00: 1. Zeichen Name Bit 7 on
194  * 01: 2. Zeichen Name Bit 7 on
195  * 02: HH Hex-Zahl (HH<$80 = pos.)
196  * 03: LL Hex-Zahl
197  * 04: entfällt: 00
198  * 05: entfällt: 00
199  * 06: entfällt: 00
200  *
201  * c) String (bis 255 Zeichen)
202  * ----------
203  *
204  * 00: 1. Zeichen Name Bit 7 off
205  * 01: 2. Zeichen Name Bit 7 on
206  * 02: Länge-Byte (00-FF)
207  * 03: Pointer LL Hex
208  * 04: Pointer HH Hex
209  * 05: entfällt: 00
210  * 06: entfällt: 00
```

```
211  * Eindimensionale Variablen
212  * ==========================
213  *
214  * a) Real
215  * -------
216  *
217  * 7-Byte-Kopf + je 5 Bytes Werte
218  *
219  * 00: 1. Zeichen Name Bit 7 off
220  * 01: 2. Zeichen Name Bit 7 off
221  * 02: Offset-LL zur nächsten DIM
222  * 03: Offset-HH zur nächsten DIM
223  * 04: Dimensionsanzahl: 01 (eine)
224  * 05: Elementeanzahl HH
225  * 06: Elementeanzahl LL
226  *
227  * 07-0B: 1.-5. Zahl-Byte N=00
228  * 0C-10: 1.-5. Zahl-Byte N=01 usw.
229  *
230  * b) Integer
231  * ----------
232  *
233  * 7-Byte-Kopf + je 2 Bytes Werte
234  *
235  * 00: 1. Zeichen Name Bit 7 on
236  * 01: 2. Zeichen Name Bit 7 on
237  * 02: Offset-LL zur nächsten DIM
238  * 03: Offset-HH zur nächsten DIM
239  * 04: Dimensionsanzahl: 01 (eine)
240  * 05: Elementeanzahl HH
241  * 06: Elementeanzahl LL
242  *
243  * 07-08: HH LL Hex-Zahl N=00
244  * 09-0A: HH LL Hex-Zahl N=01 usw.
245  *
246  * c) String
247  * ---------
248  *
249  * 7-Byte-Kopf + je 3 Bytes für
250  * Länge und Pointer
251  *
252  * 00: 1. Zeichen Name Bit 7 off
253  * 01: 2. Zeichen Name Bit 7 on
254  * 02: Offset-LL zur nächsten DIM
255  * 03: Offset-HH zur nächsten DIM
256  * 04: Dimensionsanzahl: 01 (eine)
257  * 05: Elementeanzahl HH
258  * 06: Elementeanzahl LL
259  *
260  * 07-09: Länge + LL-HH-Ptr. N=00
261  * 0A-0C: Länge + LL-HH-Ptr. N=01
262  * usw.
```

```
100  PRINT  CHR$ (4)„BRUN PTRGET"
110 X = 111111:X% = 32767:X$ = „XXXXX"
120  DIM X(100)
130  DIM X%(100)
140  DIM X$(100)
150  FOR X = 0 TO 100
160 X(X) = X
170 X%(X) = X
180 X$(X) =  STR$ (X(X))
190  NEXT
200  PRINT „EINFACHE VARIABLEN": PRINT
210  PRINT „X  ";: & X: PRINT „X% ";: & X%: PRINT „X$ ";: & X$
220  PRINT : PRINT „X  (100)-ARRAY": PRINT
230  FOR X = 0 TO 100: PRINT „X  („;X;") „;: " X(X): NEXT
240  PRINT : PRINT „X" (100)-ARRAY„: PRINT
250  FOR X = 0 TO 100: PRINT "X% („;X;") „;: & X%(X): NEXT
260  PRINT : PRINT "X$ (100)-ARRAY„: PRINT
270  FOR X = 0 TO 100: PRINT "X$ („;X;") ";: & X$(X): NEXT
```

```
PTRGET-Beispiele:

X  $09BA:87 4A 00 00 00
X% $09C1:7F FF
X$ $09C8:05 40 08  58 58 58 58 58

X  (100)-ARRAY

X  (0) $09D4:00 00 00 00 00
X  (1) $09D9:81 00 00 00 00
X  (2) $09DE:82 00 00 00 00
X  (3) $09E3:82 40 00 00 00

X% (100)-ARRAY

X% (0) $0BD4:00 00
X% (1) $0BD6:00 01
X% (2) $0BD8:00 02
X% (3) $0BDA:00 03

X$ (100)-ARRAY

X$ (0) $0CA5:01 FF 95  30
X$ (1) $0CA8:01 FE 95  31
X$ (2) $0CAB:01 FD 95  32
X$ (3) $0CAE:01 FC 95  33
```

```
                        1               ORG  $02F0
                        2    *
                        3    * GETARYPT
                        4    * ========
                        5    *
                        6    * GETARYPT sucht Anfangsadresse
                        7    * des Array-Kopfes, dessen Name
                        8    * bei TXTPTR steht, und legt
                        9    * Adresse in LOWTR und LOWTR+1 ab.
                       10    * & ARRAYNAME ergibt Anzeige von
                       11    * A$HHLL, L$HHLL, E$HHLL
                       12    * falls Flag < > 0 ist.
                       13    *
                       14    * Im Gegensatz zu PTRGET erfolgt
                       15    * bei GETARYPT Fehlermeldung,
                       16    * falls Array nicht existiert,
                       17    * d.h. es wird nicht ein Array
                       18    * mit DIM NAME (10) automatisch
                       19    * angelegt.
                       20    *
                       21    TXTPTR   EQU  $B8         ;-$B9
                       22    LOWTR    EQU  $9B         ;-$9C
                       23    DATA     EQU  $D995       ;nach „:"
                       24    GETARYPT EQU  $F7D9
                       25    PRNTAX   EQU  $F941       ;A=H,X=L
                       26    COUT     EQU  $FDED
                       27    *
02F0: A9 4C            28    AMPER    LDA  #$4C
02F2: 8D F5 03         29             STA  $3F5
02F5: A9 07            30             LDA  #<ARRAY1
02F7: 8D F6 03         31             STA  $3F6
02FA: A9 03            32             LDA  #>ARRAY1
02FC: 8D F7 03         33             STA  $3F7
02FF: 60               34             RTS
                       35    *
                       36    * Die hier abgelegten Werte
                       37    * können benutzt werden, um
                       38    * den Array mit
                       39    * BSAVE NAME, A ANFADR, L LAENGE
                       40    * zu speichern sowie mit
                       41    * BLOAD NAME, A ANFADR
                       42    * später zu laden.
                       43    * Beachte, daß nur Integer- und
                       44    * Real-Arrays, NICHT String-
                       45    * Arrays gespeichert werden
                       46    * können.
                       47    *
0300: 00 00            48    ANFADR   HEX  0000        ;LLHH
0302: 00 00            49    ENDADR   HEX  0000        ;LLHH
0304: 00 00            50    LAENGE   HEX  0000        ;LLHH
                       51    *
0306: 01               52    FLAG     HEX  01          ;0=nein
                       53    *
```

```
0307: 20 D9 F7  54   ARRAY1   JSR  GETARYPT
                55   *
                56   * Nach Getarypt zeigt Lowtr auf
                57   * Byte 0 des Array-Kopfes,
                58   * der bei eindimensionalen
                59   * Arrays, z.B. DIM A (100),
                60   * aus 7 Bytes 0-6 besteht:
                61   *
                62   * Byte 0-1: 1. und 2. Buchstabe
                63   *           des Variable-Namens
                64   * Byte 2-3: LL-HH-Offset zur
                65   *           Anfangsadresse des
                66   *           nächsten Arrays
                67   * Byte 4:   01 für 1. Dimension
                68   * Byte 5-6: HH-LL-Anzahl der
                69   *           Elemente der 1.
                70   *           Dimension.
                71   *
                72   * Lowtr = Anfangsadresse
                73   *
030A: A5 9B     74            LDA  LOWTR
030C: 8D 00 03  75            STA  ANFADR
030F: A5 9C     76            LDA  LOWTR+1
0311: 8D 01 03  77            STA  ANFADR+1
                78   *
                79   * Offset = Länge
                80   *
0314: A0 02     81            LDY  #2          ;Byte 2
0316: B1 9B     82            LDA  (LOWTR),Y
0318: 8D 04 03  83            STA  LAENGE
031B: C8        84            INY              ;Byte 3
031C: B1 9B     85            LDA  (LOWTR),Y
031E: 8D 05 03  86            STA  LAENGE+1
                87   *
                88   * Endadresse = Lowtr + Offset - 1
                89   *
0321: 18        90            CLC
0322: A5 9B     91            LDA  LOWTR
0324: A0 02     92            LDY  #2          ;Byte 2
0326: 71 9B     93            ADC  (LOWTR),Y
0328: 8D 02 03  94            STA  ENDADR
032B: A5 9C     95            LDA  LOWTR+1
032D: C8        96            INY              ;Byte 3
032E: 71 9B     97            ADC  (LOWTR),Y
0330: 8D 03 03  98            STA  ENDADR+1
                99   *
0333: 38        100           SEC
0334: AD 02 03  101           LDA  ENDADR
0337: E9 01     102           SBC  #1
0339: 8D 02 03  103           STA  ENDADR
033C: AD 03 03  104           LDA  ENDADR+1
033F: E9 00     105           SBC  #0
0341: 8D 03 03  106           STA  ENDADR+1
```

```
                    107  * Falls Flag = 0, dann
                    108  * Anzeige unterdrücken.
                    109  *
0344: AD 06 03      110           LDA  FLAG
0347: F0 2F         111           BEQ  ARRAY2
                    112  *
0349: A9 C1         113           LDA  #„A"        ;Anf.
034B: 20 7B 03      114           JSR  DOLLAR
034E: AE 00 03      115           LDX  ANFADR
0351: AD 01 03      116           LDA  ANFADR+1
0354: 20 84 03      117           JSR  KOMMA
                    118  *
0357: A9 CC         119           LDA  #„L"        ;Län.
0359: 20 7B 03      120           JSR  DOLLAR
035C: AE 04 03      121           LDX  LAENGE
035F: AD 05 03      122           LDA  LAENGE+1
0362: 20 84 03      123           JSR  KOMMA
                    124  *
0365: A9 C5         125           LDA  #„E"        ;End.
0367: 20 7B 03      126           JSR  DOLLAR
036A: AE 02 03      127           LDX  ENDADR
036D: AD 03 03      128           LDA  ENDADR+1
0370: 20 41 F9      129           JSR  PRNTAX
                    130  *
0373: A9 8D         131           LDA  #$8D        ;Return
0375: 20 ED FD      132           JSR  COUT
                    133  *
                    134  * Txtptr auf „:" oder Eol
                    135  *
0378: 4C 95 D9      136  ARRAY2   JMP  DATA
                    137  *
                    138  * A, L oder E + „$" anzeigen
                    139  *
037B: 20 ED FD      140  DOLLAR   JSR  COUT
037E: A9 A4         141           LDA  #„$"
0380: 20 ED FD      142           JSR  COUT
0383: 60            143           RTS
                    144  *
                    145  * Adresse + „, " anzeigen
                    146  *
0384: 20 41 F9      147  KOMMA    JSR  PRNTAX
0387: A9 AC         148           LDA  #„,"
0389: 20 ED FD      149           JSR  COUT
038C: A9 A0         150           LDA  #$A0
038E: 20 ED FD      151           JSR  COUT
0391: 60            152           RTS

--End assembly-

162 bytes

Errors: 0
```

```
10  PRINT  CHR$ (4)„BRUN GETARYPT"
20  DIM A%(11)
30  FOR X = 0 TO 11:A%(X) = 32767: NEXT
40  DIM B%(11)
50  FOR X = 0 TO 11:B%(X) = 257: NEXT
60  & A%: & B%

NACH ZEILE 60 WÜRDE HIER ANGEZEIGT:
A$0879, L$001F, E$0897
A$0898, L$001F, E$08B6
```

```
                    1               ORG  $0300
                    2   *
                    3   * FAC
                    4   * ===
                    5   *
                    6   * Floating Point Accumulator
                    7   *
                    8   * Nach & + Zahl, z.B. &100
                    9   * FAC wie folgt darstellen:
                   10   * binär, hexadezimal und dezimal
                   11   * als gepackt und ungepackt
                   12   *
0300: A9 4C        13   AMPER    LDA  #$4C
0302: 8D F5 03     14           STA  $3F5
0305: A9 24        15           LDA  #<BINAER1
0307: 8D F6 03     16           STA  $3F6
030A: A9 03        17           LDA  #>BINAER1
030C: 8D F7 03     18           STA  $3F7
030F: 60           19           RTS
                   20   *
                   21   FAC       EQU  $009D       ;-$00A2
                   22   HEXOUT    EQU  $FDDA
                   23   PRINT     EQU  $FDED
                   24   FRMNUM    EQU  $DD67       ;>FAC
                   25   PRTFAC    EQU  $ED2E
                   26   MOVMF     EQU  $EB2B       ;FAC>X,Y
                   27   MOVFM     EQU  $EAF9       ;A,Y>FAC
                   28   *
0310: 00 00 00     29   FACSAVE   HEX  000000000000 ;6 Bytes
0313: 00 00 00
0316: 00 00 00     30   MEMSAVE   HEX  0000000000 ;5 Bytes
0319: 00 00
031B: B1 B1 B1     31   BINZAHL   ASC  „11111111"
031E: B1 B1 B1 B1 B1
0323: 00           32   XSAVE     HEX  00
                   33   *
                   34   * Zahl oder Zahlformel
                   35   * nach & auswerten in FAC
                   36   *
0324: 20 67 DD     37   BINAER1   JSR  FRMNUM
                   38   *
                   39   * Ungepacktes FAC retten
                   40   *
0327: A2 05        41           LDX  #5
0329: B5 9D        42   FACSAVER  LDA  FAC,X
032B: 9D 10 03     43           STA  FACSAVE,X
032E: CA           44           DEX
032F: 10 F8        45           BPL  FACSAVER
                   46   *
                   47   * Gepacktes FAC retten
                   48   *
0331: A2 16        49           LDX  #<MEMSAVE
0333: A0 03        50           LDY  #>MEMSAVE
0335: 20 2B EB     51           JSR  MOVMF       ;FAC>MEM
                   52   *
```

```
                        53   * Dezimal anzeigen
                        54   *
0338: 20 2E ED          55            JSR  PRTFAC
033B: A9 BA             56            LDA  #„:"
033D: 20 ED FD          57            JSR  PRINT
0340: A9 8D             58            LDA  #$8D
0342: 20 ED FD          59            JSR  PRINT
                        60   *
                        61   * Ungepacktes FAC anzeigen
                        62   *
0345: A2 00             63            LDX  #0
0347: BD 10 03          64   BINAER2  LDA  FACSAVE,X
034A: 20 80 03          65            JSR  BINPRINT
034D: BD 10 03          66            LDA  FACSAVE,X
0350: 20 DA FD          67            JSR  HEXOUT
0353: A9 A0             68            LDA  #$A0
0355: 20 ED FD          69            JSR  PRINT
0358: E8                70            INX
0359: E0 06             71            CPX  #6
035B: D0 EA             72            BNE  BINAER2
                        73   *
035D: A9 8D             74            LDA  #$8D
035F: 20 ED FD          75            JSR  PRINT
                        76   *
                        77   * Gepacktes FAC anzeigen
                        78   *
0362: A2 00             79            LDX  #0
0364: BD 16 03          80   BINAER3  LDA  MEMSAVE,X
0367: 20 80 03          81            JSR  BINPRINT
036A: BD 16 03          82            LDA  MEMSAVE,X
036D: 20 DA FD          83            JSR  HEXOUT
0370: A9 A0             84            LDA  #$A0
0372: 20 ED FD          85            JSR  PRINT
0375: E8                86            INX
0376: E0 05             87            CPX  #5
0378: D0 EA             88            BNE  BINAER3
                        89   *
037A: A9 8D             90            LDA  #$8D
037C: 20 ED FD          91            JSR  PRINT
037F: 60                92            RTS
                        93   *
                        94   * Binär anzeigen:
                        95   * Entry mit Accumulator = Hexzahl
                        96   *
0380: A8                97   BINPRINT TAY
0381: 8E 23 03          98            STX  XSAVE
0384: A2 07             99            LDX  #7
0386: 98                100  LOOP     TYA
0387: 2A                101           ROL
0388: 90 07             102           BCC  ZERO
038A: A9 B1             103  ONE      LDA  #„1"
038C: 9D 1B 03          104           STA  BINZAHL,X
038F: D0 05             105           BNE  DECREASE
0391: A9 B0             106  ZERO     LDA  #$B0
0393: 9D 1B 03          107           STA  BINZAHL,X
```

```
0396: 98          108  DECREASE TYA
0397: 2A          109           ROL
0398: A8          110           TAY
0399: CA          111           DEX
039A: 10 EA       112           BPL  LOOP
039C: A2 07       113           LDX  #7
039E: BD 1B 03    114  STRPRT   LDA  BINZAHL,X
03A1: 20 ED FD    115           JSR  PRINT
03A4: CA          116           DEX
03A5: 10 F7       117           BPL  STRPRT
03A7: A9 A0       118           LDA  #$A0
03A9: 20 ED FD    119           JSR  PRINT
03AC: AE 23 03    120           LDX  XSAVE
03AF: 60          121           RTS
```

```
10  PRINT  CHR$ (4)„PR#3"
20  PRINT  CHR$ (4)„BRUN FAC"
30  FOR X =  - 32 TO  + 32
40 F = X: & F
50  NEXT X
60  FOR X = 0 TO 32
70 F = (2 ↑ X): & F - 1: & F: & F + 1
80  PRINT
90  NEXT X
```

FAC-Beispiele:

```
0:
00000000 00 00000000 00 00000000 00 00000000 00 00000000 00 00000000 00
00000000 00 00000000 00 00000000 00 00000000 00 00000000 00

1:
10000001 81 10000000 80 00000000 00 00000000 00 00000000 00 00000000 00
10000001 81 00000000 00 00000000 00 00000000 00 00000000 00
-1:
10000001 81 10000000 80 00000000 00 00000000 00 00000000 00 10000000 80
10000001 81 10000000 80 00000000 00 00000000 00 00000000 00

3:
10000010 82 11000000 C0 00000000 00 00000000 00 00000000 00 00000000 00
10000010 82 01000000 40 00000000 00 00000000 00 00000000 00
-3:
10000010 82 11000000 C0 00000000 00 00000000 00 00000000 00 11000000 C0
10000010 82 11000000 C0 00000000 00 00000000 00 00000000 00

7:
10000011 83 11100000 E0 00000000 00 00000000 00 00000000 00 00000000 00
10000011 83 01100000 60 00000000 00 00000000 00 00000000 00
-7:
10000011 83 11100000 E0 00000000 00 00000000 00 00000000 00 11100000 E0
10000011 83 11100000 E0 00000000 00 00000000 00 00000000 00

15:
10000100 84 11110000 F0 00000000 00 00000000 00 00000000 00 00000000 00
10000100 84 01110000 70 00000000 00 00000000 00 00000000 00
-15:
10000100 84 11110000 F0 00000000 00 00000000 00 00000000 00 11110000 F0
10000100 84 11110000 F0 00000000 00 00000000 00 00000000 00
```

```
                        1               ORG  $0300
                        2    *
                        3    * FIN
                        4    * ===
                        5    *
                        6    * Mit der Applesoft-Routine
                        7    * Fin läßt sich aus einem
                        8    * Assemblerprogramm heraus
                        9    * ein Zahlstring in eine
                        10   * Fließkommazahl umwandeln,
                        11   * die dann ihrerseits ggf.
                        12   * in eine Hex-Zahl umgewandelt
                        13   * werden könnte.
                        14   *
                        15   USR       EQU  $000A      ;-$000C
                        16   FAC       EQU  $009D      ;-$00A2
                        17   CHRGOT    EQU  $00B7
                        18   TXTPTR    EQU  $00B8      ;-$00B9
                        19   FIN       EQU  $EC4A
0300: 4C 0E 03          20             JMP  START
                        21   *
                        22   * 10stelliger Zahlstring Bit 7 off
                        23   * mit 00 als Endmarker
                        24   *
0303: 31 32 33          25   ZAHLSTR   ASC  '1234567890'
0306: 34 35 36 37 38 39 30
030D: 00                26   ENDMARK   HEX  00
                        27   *
                        28   * USR-Vektor setzen
                        29   *
030E: A9 4C             30   START     LDA  #$4C
0310: 85 0A             31             STA  USR
0312: A9 1B             32             LDA  #<START1
0314: 85 0B             33             STA  USR+1
0316: A9 03             34             LDA  #>START1
0318: 85 0C             35             STA  USR+2
031A: 60                36             RTS
                        37   *
                        38   * Text-Pointer retten
                        39   *
031B: A5 B8             40   START1    LDA  TXTPTR
031D: 8D 48 03          41             STA  TXTSAVL
0320: A5 B9             42             LDA  TXTPTR+1
0322: 8D 49 03          43             STA  TXTSAVH
                        44   *
                        45   * Text-Pointer auf Zahlstring
                        46   * setzen und erstes Zahlzeichen
                        47   * in Accumulator laden (per
                        48   * Chrgot) und dann Fin ausführen
                        49   *
0325: A9 03             50             LDA  #<ZAHLSTR
0327: 85 B8             51             STA  TXTPTR
0329: A9 03             52             LDA  #>ZAHLSTR
032B: 85 B9             53             STA  TXTPTR+1
```

```
032D: 20 B7 00  54            JSR  CHRGOT
0330: 20 4A EC  55            JSR  FIN
                56  *
                57  * Floating-Point-Accumulator
                58  * retten
                59  *
0333: A2 06     60            LDX  #6
0335: B5 9D     61  LOOP      LDA  FAC,X       ;FAC
0337: 9D 4A 03  62            STA  FACSAVE,X
033A: CA        63            DEX
033B: 10 F8     64            BPL  LOOP
                65  *
                66  * Text-Pointer wiederherstellen
                67  *
033D: AD 48 03  68            LDA  TXTSAVL
0340: 85 B8     69            STA  TXTPTR
0342: AD 49 03  70            LDA  TXTSAVH
0345: 85 B9     71            STA  TXTPTR+1
0347: 60        72            RTS
0348: 00        73  TXTSAVL   HEX  00
0349: 00        74  TXTSAVH   HEX  00
034A: 00 00 00  75  FACSAVE   HEX  000000000000
034D: 00 00 00

--End assembly-

80 bytes

Errors: 0
```

```
10  PRINT  CHR$ (4)„BRUN FIN"
20  INPUT X$: IF  VAL (X$) = 0 AND X$ <  > „0" THEN 20
30  IF  LEN (X$) > 10 THEN 20
40  FOR X = 1 TO  LEN (X$): POKE 770 - 1 + X, ASC ( MID$ (X$,X,1)):
    NEXT : POKE 770 - 1 + X,0
50 Y = 0: PRINT Y:Y =  USR (Y): PRINT Y
60  GOTO 20
```

```
                        1               ORG  $300
                        2    *
                        3    * GIVAYF
                        4    * ======
                        5    *
                        6    * Umwandlung einer 16-Bit-Hexzahl
                        7    * in FAC-Dezzahl mit/ohne Vor-
                        8    * zeichen.
                        9    *
                        10   FADD     EQU  $E7BE      ;A=L,Y=H
                        11   GIVAYF   EQU  $E2F2      ;A=H,Y=L
                        12   PRNTFAC  EQU  $ED2E      ;FAC-OUT
                        13   PRNTAX   EQU  $F941      ;AX-OUT
                        14   PRINT    EQU  $FDED      ;A-OUT
                        15   *
0300: 60                16   HEXL     HEX  60         ;Lowbyte
0301: 00                17   HEXH     HEX  00         ;Highbyte
                        18   *
                        19   * Hex als echte Hexzahl anzeigen
                        20   * $0000-$FFFF
                        21   *
0302: A9 A4             22            LDA  #„$"
0304: 20 ED FD          23            JSR  PRINT
0307: AD 01 03          24            LDA  HEXH
030A: AE 00 03          25            LDX  HEXL
030D: 20 41 F9          26            JSR  PRNTAX     ;$AAXX
                        27   *
                        28   * Hex als Dezzahl mit Vorzeichen
                        29   * anzeigen:  -32768 bis +32767
                        30   *
                        31   * (Beachte: $8000 = -32768 ist
                        32   * nicht als Integerzahl möglich)
                        33   *
0310: A9 A0             34            LDA  #$A0
0312: 20 ED FD          35            JSR  PRINT
0315: AC 00 03          36            LDY  HEXL       ;Y=LL
0318: AD 01 03          37            LDA  HEXH       ;A=HH
031B: 20 F2 E2          38            JSR  GIVAYF     ;YY-AA
                        39   *
                        40   * Hier RTS, wenn Dezzahl mit
                        41   * Vorzeichen in FAC bleiben soll
                        42   *
031E: 20 2E ED          43            JSR  PRNTFAC
                        44   *
                        45   * Hex als Dezzahl ohne Vorzeichen
                        46   * anzeigen:  0 bis 65535
                        47   *
0321: A9 A0             48            LDA  #$A0
0323: 20 ED FD          49            JSR  PRINT
0326: AC 00 03          50            LDY  HEXL       ;Y=LL
0329: AD 01 03          51            LDA  HEXH       ;A=HH
032C: 20 F2 E2          52            JSR  GIVAYF
                        53   *
```

```
                    54   * Nach GIVAYF ist zunächst
                    55   * Integerzahl in FAC. Wenn FAC
                    56   * negativ, dann 65536 hinzu-
                    57   * addieren.
                    58   *
032F: AD 01 03      59            LDA  HEXH
0332: 10 07         60            BPL  OKAY        ;positiv
                    61   *
0334: A9 3E         62            LDA  #<Z65536    ;A=PTR-L
0336: A0 03         63            LDY  #>Z65536    ;Y=PTR-H
0338: 20 BE E7      64            JSR  FADD        ;FAC+MEM
                    65   *
                    66   * Hier RTS, wenn Dezzahl ohne
                    67   * Vorzeichen in FAC bleiben soll
                    68   *
033B: 4C 2E ED      69   OKAY     JMP  PRNTFAC
                    70   *
033E: 91 00 00      71   Z65536   HEX  9100000000
0341: 00 00

--End assembly-

67 bytes

Errors: 0
```

GIVAYF-Beispiele:

$0000 0 0	$FF00 -256 65280	$8000 -32768 32768
$0001 1 1	$FF01 -255 65281	$8001 -32767 32769
$0002 2 2	$FF02 -254 65282	$8002 -32766 32770
$0003 3 3	$FF03 -253 65283	$8003 -32765 32771
$0004 4 4	$FF04 -252 65284	$8004 -32764 32772
$0005 5 5	$FF05 -251 65285	$8005 -32763 32773
$0006 6 6	$FF06 -250 65286	$8006 -32762 32774
$0007 7 7	$FF07 -249 65287	$8007 -32761 32775
$0008 8 8	$FF08 -248 65288	$8008 -32760 32776
$0009 9 9	$FF09 -247 65289	$8009 -32759 32777

```
                    1    * MATHE
                    2    * =====
                    3    *
                    4    * Neu zusammengestellt Juni 1984
                    5    *
                    6            ORG  $300
0300: 4C 12 03      7            JMP  DEMO
                    8    *
                    9    * FAC und ARG
                    10   * -----------
                    11   *
                    12   * FAC =Floating Point Accumulator
                    13   * ARG =Argument
                    14   *
                    15   FAC      EQU  $9D        ;$9D-$A2
                    16   ARG      EQU  $A5        ;$A5-$AA
                    17   *
                    18   * Mathe-Routinen: FAC und ARG
                    19   * --------------------------
                    20   *
                    21   * FAC ist 6-Byte nicht-gepackte N
                    22   * ARG ist 6-Byte nicht-gepackte N
                    23   *
                    24   *  Prozedur:
                    25   *  LDA $9D      ;$9D=FAC-Exponent
                    26   *  JSR ROUTINE
                    27   *  Ergebnis in FAC
                    28   *
                    29   * FADDT
                    30   ADD      EQU  $E7C1      ;ARG+FAC
                    31   * FSUBT
                    32   SUBTRACT EQU  $E7AA      ;ARG-FAC
                    33   * FMULTT
                    34   MULTIPLY EQU  $E982      ;ARG*FAC
                    35   * FDIVT
                    36   DIVIDE   EQU  $EA69      ;ARG/FAC
                    37   * FPWRT
                    38   EXPONENT EQU  $EE97      ;ARG↑FAC
                    39   *
                    40   * Mathe-Routinen: FAC allein
                    41   * --------------------------
                    42   *
                    43   * Zufallszahl erzeugen
                    44   *
                    45   RND      EQU  $EFAE      ;FAC
                    46   *
                    47   * Sinus usw. in rad = ca. 57 Grad
                    48   *
                    49   * SIN
                    50   SINE     EQU  $EFF1      ;FAC
                    51   * COS
                    52   COSINE   EQU  $EFEA      ;FAC
                    53   * TAN
```

```
54  TANGENT  EQU  $F03A      ;FAC
55  * ATN
56  ARCTAN   EQU  $F09E      ;FAC
57  *
58  * SQR
59  SQROOT   EQU  $EE8D      ;FAC
60  *
61  * Negation: FAC=-FAC
62  *
63  * NEGOP
64  NEGATION EQU  $EED0      ;FAC
65  *
66  * ABS
67  ABSOLUTE EQU  $EBAF      ;FAC
68  *
69  * Normale Integer-Funktion
70  * für FAC < 2 ↑ 31
71  *
72  * INT
73  INTEGER  EQU  $EC23      ;FAC
74  *
75  * QUICK Integer-Funktion
76  * für FAC < 2 ↑ 23 = 8388608
77  *
78  QINT     EQU  $EBF2      ;FAC
79  *
80  * AYINT Integer-Funktion
81  * für FAC < 2 ↑ 15 = 32768
82  *
83  AYINT    EQU  $E10C      ;FAC
84  *
85  * LOGNAT = Natürlicher ln von FAC
86  *
87  * LOG
88  LOGNAT   EQU  $E941      ;FAC
89  *
90  * EXPNAT = 2.718289 ↑ FAC = e ↑ FAC
91  * EXP
92  EXPNAT   EQU  $EF09      ;FAC
93  *
94  * SGN-Funktion (Vorzeichen) läßt
95  *  1 in FAC, wenn FAC > 0 ist
96  *  0 in FAC, wenn FAC = 0 ist
97  * -1 in FAC, wenn FAC < 0 ist
98  *
99  SGN      EQU  $EB90      ;FAC
100 *
101 * SIGN-Funktion (Vorzeichen) läßt
102 *  01 in A, wenn FAC > 0 ist
103 *  00 in A, wenn FAC = 0 ist
104 *  FF in A, wenn FAC < 0 ist
105 *
106 SIGN     EQU  $EB82      ;FAC
107 *
```

```
108  * Mathe-Routinen: FAC und MEM
109  * --------------------------
110  *
111  * FAC ist 6-Byte nicht-gepackte N
112  * MEM ist 5 Byte gepackte N
113  *  im Speicher bei Stelle
114  *  A = Lowbyte; Y = Highbyte
115  *
116  *  Prozedur:
117  *  LDA Lowbyte MEM-Adresse
118  *  LDY Highbyte MEM-Adresse
119  *  JSR ROUTINE
120  *  Ergebnis in FAC
121  *
122  * FADD
123  ADDMEM   EQU  $E7BE      ;MEM+FAC
124  * FSUB
125  SUBMEM   EQU  $E7A7      ;MEM-FAC
126  * FMULT
127  MULMEM   EQU  $E97F      ;MEM*FAC
128  * FDIV
129  DIVMEM   EQU  $EA66      ;MEM/FAC
130  *
131  * Vergleiche MEM mit FAC
132  * Ergebnis bleibt in A-Register
133  * Wenn MEM < FAC dann A = 01
134  * Wenn MEM = FAC dann A = 00
135  * Wenn MEN > FAC dann A = FF
136  *
137  * FCOMP
138  COMPARE  EQU  $EBB2      ;MEM,FAC
139  *
140  * Umwandlung von FAC in String
141  * ---------------------------
142  *
143  * FOUT verwandelt FAC in String
144  * das in FBUFFER = Stringpuffer
145  * mit Bit 7 off und Endmarker 00
146  * abgelegt wird. FBUFFER = Stack
147  * Danach ist A = Lowbyte FBUFFER
148  * und Y = Highbyte FBUFFER und
149  * Zahl kann mit JSR STROUT aus-
150  * gedruckt werden.
151  *
152  * Beispiel:
153  *
154  * FAC   : 819E0652131E
155  * $0100: 312E323334353637383900
156  *        1 . 2 3 4 5 6 7 8 9
157  *
158  FOUT     EQU  $ED34
159  FBUFFER  EQU  $0100      ;STACK
160  *
```

```
                    161 * STROUT: LDA Lowbyte FBUFFER
                    162 *          LDY Highbyte FBUFFER
                    163 *          JSR STROUT
                    164 *          (sofern FOUT nicht vorausging)
                    165 STROUT   EQU  $DB3A
                    166 *
                    167 * PRNTFAC = entspricht
                    168 *           JSR FOUT
                    169 *           JSR STROUT
                    170 *
                    171 PRNTFAC  EQU  $ED2E
                    172 *
                    173 * Move und Pack-/Entpack-Routinen
                    174 * ------------------------------
                    175 *
                    176 *  LDA Lowbyte  von MEM
                    177 *  LDY Highbyte von MEM
                    178 *  JSR MEMFAC oder MEMARG
                    179 *
                    180 * MOVFM
                    181 MEMFAC   EQU  $EAF9       ;MEM>FAC
                    182 * CONUPK
                    183 MEMARG   EQU  $E9E3       ;MEM>ARG
                    184 *
                    185 *  LDX Lowbyte  von MEM
                    186 *  LDY Highbyte von MEM
                    187 *  JSR FACMEM
                    188 *
                    189 * MOVMF
                    190 FACMEM   EQU  $EB2B       ;FAC>MEM
                    191 *
                    192 *  JSR FACARG oder JSR ARGFAC
                    193 *
                    194 * MOVAF
                    195 FACARG   EQU  $EB63       ;FAC>ARG
                    196 * MOVFA
                    197 ARGFAC   EQU  $EB53       ;ARG>FAC
                    198 *
                    199 * Verschiedenes
                    200 * -------------
                    201 *
                    202 ROUND    EQU  $EB72       ;LAST BIT
                    203 DOSWARM  EQU  $3D0
                    204 *
                    205 *--------DEMO--------------------
                    206 *
                    207 * FOR X = 1 TO 1000:
                    208 * PRINT X * X: NEXT
                    209 *
0303: 81 00 00      210 EINS     HEX  8100000000 ;5 Bytes
0306: 00 00
0308: 8A 7A 00      211 TAUSEND  HEX  8A7A000000 ;5 Bytes
030B: 00 00
030D: 00 00 00      212 VARX     HEX  0000000000 ;5 Bytes
0310: 00 00
```

```
                    213  *
0312: A2 04         214  DEMO      LDX  #4
0314: A9 00         215            LDA  #0
0316: 9D 0D 03      216  CLEAR     STA  VARX,X
0319: CA            217            DEX
031A: 10 FA         218            BPL  CLEAR
                    219  *
                    220  * Move X in FAC
                    221  *
031C: A9 0D         222  XTOFAC    LDA  #<VARX
031E: A0 03         223            LDY  #>VARX
0320: 20 F9 EA      224            JSR  MEMFAC
                    225  *
                    226  * X = X + 1
                    227  *
0323: A9 03         228            LDA  #<EINS
0325: A0 03         229            LDY  #>EINS
0327: 20 BE E7      230            JSR  ADDMEM
                    231  *
                    232  * Move FAC in X
                    233  *
032A: A2 0D         234  FACTOX    LDX  #<VARX
032C: A0 03         235            LDY  #>VARX
032E: 20 2B EB      236            JSR  FACMEM
                    237  *
                    238  * Wenn 1000 > = X dann weiter
                    239  *
0331: A9 08         240            LDA  #<TAUSEND
0333: A0 03         241            LDY  #>TAUSEND
0335: 20 B2 EB      242            JSR  COMPARE
0338: C9 01         243            CMP  #$01        ;MEM>=FAC
033A: D0 03         244            BNE  XTOARG
033C: 4C D0 03      245            JMP  DOSWARM     ;EXIT
                    246  *
                    247  * Move X in ARG
                    248  *
033F: A9 0D         249  XTOARG    LDA  #<VARX
0341: A0 03         250            LDY  #>VARX
0343: 20 E3 E9      251            JSR  MEMARG
                    252  *
                    253  * Multipliziere X mit X
                    254  *
0346: 20 82 E9      255            JSR  MULTIPLY
                    256  *
                    257  * Zahl anzeigen
                    258  *
0349: 20 2E ED      259  DISPLAY   JSR  PRNTFAC
034C: A9 8D         260            LDA  #$8D
034E: 20 ED FD      261            JSR  $FDED       ;RETURN
0351: 4C 1C 03      262            JMP  XTOFAC
                    263  *
```

```
                        1               ORG  $240
                        2    *
                        3    * PRIMZAHLEN
                        4    * ==========
                        5    *
                        6    * Berechnet Primzahlen 2 - 37369
                        7    * in knapp 2.3 Sekunden!
                        8    *
                        9    * Verbesserte Version 1.7.84
                       10    * von U.Stiehl
                       11    *
0240: 4C 49 02         12    START1   JMP  START2
                       13    *
                       14    * FLAG: 0=mit Anzeige, 1=ohne
                       15    *
0243: 01               16    FLAG     HEX  01
                       17    *
                       18    * Anzahl der Zahlen pro Zeile
                       19    * für den Ausdruck
                       20    *
0244: 06               21    ANZAHL   HEX  06
                       22    *
                       23    * Gesamtzahl aller berechneten
                       24    * Primzahlen (3958 Primzahlen)
                       25    *
0245: 00 00            26    GESAMT   HEX  0000        ;LL-HH
                       27    *
                       28    COUNT    EQU  $F9         ;$F9
                       29    *
                       30    IND      EQU  $FA         ;$FA-FB
                       31    IND1     EQU  $FC         ;$FC-FD
                       32    OFFSET   EQU  $FE         ;$FE-FF
                       33    *
                       34    RDKEY    EQU  $FD0C
                       35    PRINT    EQU  $FDED
                       36    HOME     EQU  $FC58
                       37    LINPRT   EQU  $ED24
                       38    DOSCOLD  EQU  $03D3
                       39    BELL1A   EQU  $FBE4       ;FBDD+7
                       40    LOMEM    EQU  $0800       ;2048
                       41    HIMEM    EQU  $9A00       ;39424
0247: 00 00            42    SQRHIMEM HEX  0000        ;LL-HH
                       43    *
                       44    * Der Speicher LOMEM bis HIMEM
                       45    * enthält später die Primzahlen
                       46    * (LOMEM-2048) bis (HIMEM-2048)
                       47    * als Flags: 0 Prim, 1 keine Prim
                       48    *
                       49    * 2048 - 2048 = 0 = entfällt
                       50    * 2049 - 2048 = 1 = entfällt
                       51    * 2050 - 2048 = 2 = 0 Prim
                       52    * 2051 - 2048 = 3 = 0 Prim
                       53    * 2052 - 2048 = 4 = 1 keine Prim
```

```
                        55   * usw.
                        56   * SQRHIMEM ist die Quadratwurzel
                        57   * aus (HIMEM - LOMEM), d.h.
                        58   * die Obergrenze für das Streichen
                        59   * nach der Eratosthenes-Methode
                        60   *
                        61   *        Menü
                        62   *
0249: 20 58 FC          63   START2   JSR  HOME
024C: A2 00             64            LDX  #$00
024E: BD 70 02          65   TEXT     LDA  STRING,X
0251: F0 06             66            BEQ  TASTE
0253: 20 ED FD          67            JSR  PRINT
0256: E8                68            INX
0257: D0 F5             69            BNE  TEXT
0259: 20 0C FD          70   TASTE    JSR  RDKEY
025C: C9 B1             71            CMP  #„1"
025E: F0 07             72            BEQ  START3
0260: C9 B0             73            CMP  #„0"
0262: D0 F5             74            BNE  TASTE
0264: 4C D3 03          75   ENDE     JMP  DOSCOLD
                        76   *
0267: 20 D6 02          77   START3   JSR  LOESCH0
026A: 20 AB 02          78            JSR  CONVERT
026D: 4C FB 02          79            JMP  PRIME1
                        80   *
                        81   *        Menü-String
                        82   *
0270: 8D 8D 84          83   STRING   HEX  8D8D84
0273: CD C1 D8          84            ASC  „MAXFILES1"
027C: 8D                85            HEX  8D
027D: D5 AE D3          86            ASC  „U.STIEHL"
0285: 8D 8D             87            HEX  8D8D
0287: D0 D2 C9          88            ASC  „PRIMZAHLEN"
0291: 8D 8D             89            HEX  8D8D
0293: B1 A0 D3          90            ASC  „1 START"
029A: 8D 8D             91            HEX  8D8D
029C: B0 A0 C5          92            ASC  „0 ENDE"
02A2: 8D 8D 00          93            HEX  8D8D00
                        94   *
                        95   * Applesoft-Routinen
                        96   *
                        97   LINNUM   EQU  $0050      ;-$0051
                        98   TXTPTR   EQU  $00B8      ;-$00B9
                        99   FAC      EQU  $009D      ;-$00A2
                       100   CHRGOT   EQU  $00B7
                       101   FIN      EQU  $EC4A      ;Zahl nach FAC
                       102   SQR      EQU  $EE8D      ;SQR(FAC) nach FAC
                       103   INTEGER  EQU  $EC23      ;INT(FAC) nach FAC
                       104   GETADR   EQU  $E752      ;FAC nach LINNUM
02A5: 33 37 33         105   FINZAHL  ASC  '37369'
02A8: 36 39
02AA: 00               106            HEX  00
                       107   *
```

```
                    109  * SQRHIMEM = INT (SQR (37369))
                    110  *
02AB: A9 A5         111  CONVERT  LDA  #<FINZAHL
02AD: 85 B8         112           STA  TXTPTR
02AF: A9 02         113           LDA  #>FINZAHL
02B1: 85 B9         114           STA  TXTPTR+1
02B3: 20 B7 00      115           JSR  CHRGOT
02B6: 20 4A EC      116           JSR  FIN
02B9: A5 9D         117           LDA  FAC
02BB: 20 8D EE      118           JSR  SQR
02BE: A5 9D         119           LDA  FAC
02C0: 20 23 EC      120           JSR  INTEGER
02C3: 20 52 E7      121           JSR  GETADR
                    122  *
02C6: 18            123           CLC
02C7: A5 50         124           LDA  LINNUM
02C9: 69 00         125           ADC  #<LOMEM
02CB: 8D 47 02      126           STA  SQRHIMEM
02CE: A5 51         127           LDA  LINNUM+1
02D0: 69 08         128           ADC  #>LOMEM
02D2: 8D 48 02      129           STA  SQRHIMEM+1
02D5: 60            130           RTS
                    131  *
                    132  * Zahl 2 als Primzahl markieren
                    133  * und alle Vielfachen von 2 auf
                    134  * 1=NONPRIM sowie alle Nicht-
                    135  * vielfachen von 2 auf 0=PRIM?
                    136  * setzen: $0800:0 0 1 0 1 0 1 0
                    137  *                 0 1 2 3 4 5 6 7
                    138  *                       ?   ?   ?
                    139  * D.h. es wird von der Prämisse
                    140  * ausgegangen, daß alle Nicht-
                    141  * vielfachen von 2 Primzahlen
                    142  * sein könnten.
                    143  *
02D6: A9 00         144  LOESCH0  LDA  #<LOMEM
02D8: 85 FA         145           STA  IND
02DA: A9 08         146           LDA  #>LOMEM
02DC: 85 FB         147           STA  IND+1
02DE: A2 9A         148           LDX  #>HIMEM
02E0: A0 02         149           LDY  #$02          ;$800+2
02E2: A9 00         150           LDA  #0            ;$802:0
02E4: 91 FA         151           STA  (IND),Y
02E6: F0 04         152           BEQ  LOESCH2
02E8: A9 01         153  LOESCH1  LDA  #1           ;NOPRIM!
02EA: 91 FA         154           STA  (IND),Y
02EC: C8            155  LOESCH2  INY
02ED: A9 00         156           LDA  #0            ;PRIM?
02EF: 91 FA         157           STA  (IND),Y
02F1: C8            158           INY                ;usw.
02F2: D0 F4         159           BNE  LOESCH1
02F4: E6 FB         160           INC  IND+1
02F6: E4 FB         161           CPX  IND+1
```

```
02F8: D0 EE      163           BNE  LOESCH1
02FA: 60         164           RTS
                 165  *
                 166  *        Primzahlen suchen
                 167  *
                 168  *     Primzahlflags
                 169  * 0 = Primzahl
                 170  * 1 = keine Primzahl
                 171  *
02FB: A9 03      172  PRIME1   LDA  #<LOMEM+3  ;$0803
02FD: 85 FA      173           STA  IND
02FF: A9 08      174           LDA  #>LOMEM
0301: 85 FB      175           STA  IND+1
0303: A0 00      176           LDY  #0
0305: AE 48 02   177           LDX  SQRHIMEM+1
0308: E8         178           INX             ;+1 Page
                 179  *
                 180  *        Loop
                 181  *
0309: B1 FA      182  PRIME2   LDA  (IND),Y
030B: F0 0C      183           BEQ  PRIME4
                 184  *
030D: E6 FA      185  PRIME3   INC  IND        ;NO-PRIM
030F: D0 F8      186           BNE  PRIME2
0311: E6 FB      187           INC  IND+1
0313: E4 FB      188           CPX  IND+1
0315: D0 F2      189           BNE  PRIME2
0317: F0 28      190           BEQ  ZEIGE
                 191  *
0319: 38         192  PRIME4   SEC             ;PRIM
031A: A5 FA      193           LDA  IND
031C: 85 FC      194           STA  IND1
031E: E9 00      195           SBC  #<LOMEM
0320: 85 FE      196           STA  OFFSET
0322: A5 FB      197           LDA  IND+1
0324: 85 FD      198           STA  IND1+1
0326: E9 08      199           SBC  #>LOMEM
0328: 85 FF      200           STA  OFFSET+1
                 201  *
                 202  *        Vielfache von Prim
                 203  *
032A: 18         204  PRIME5   CLC
032B: A5 FE      205  PRIME6   LDA  OFFSET
032D: 65 FC      206           ADC  IND1
032F: 85 FC      207           STA  IND1
0331: A5 FF      208           LDA  OFFSET+1
0333: 65 FD      209           ADC  IND1+1
                 210  *
                 211  *        Größer als Himem?
                 212  *
0335: C9 9A      213           CMP  #>HIMEM
0337: B0 D4      214           BCS  PRIME3
0339: 85 FD      215           STA  IND1+1     ;CLC!
```

```
                        217  *
033B: A9 01             218  PRIME7   LDA  #1          ;NO-PRIM
033D: 91 FC             219           STA  (IND1),Y
033F: 90 EA             220           BCC  PRIME6
                        221  *
                        222  *        Zeige Primzahlen
                        223  *
0341: A0 02             224  ZEIGE    LDY  #$02        ;Kurzpiep
0343: 20 E4 FB          225           JSR  BELL1A
0346: A9 D0             226           LDA  #„P"
0348: 20 ED FD          227           JSR  PRINT
034B: AD 43 02          228           LDA  FLAG
034E: F0 03             229           BEQ  ZEIGE0
0350: 4C 59 02          230           JMP  TASTE
0353: 20 58 FC          231  ZEIGE0   JSR  HOME
                        232  *
0356: A9 02             233           LDA  #2          ;ab 2
0358: 85 FC             234           STA  IND1
035A: A9 00             235           LDA  #0
035C: 85 FD             236           STA  IND1+1
                        237  *
035E: 8D 45 02          238           STA  GESAMT      ;0
0361: 8D 46 02          239           STA  GESAMT+1
                        240  *
0364: 85 F9             241           STA  COUNT       ;0
                        242  *
0366: A9 02             243           LDA  #<LOMEM+2   ;$0802
0368: 85 FA             244           STA  IND
036A: A9 08             245           LDA  #>LOMEM
036C: 85 FB             246           STA  IND+1
                        247  *
                        248  *        Zeige-Loop
                        249  *
036E: A0 00             250  ZEIGE1   LDY  #0
0370: B1 FA             251           LDA  (IND),Y
0372: D0 26             252           BNE  WEITER1
0374: A6 FC             253  ZEIGE2   LDX  IND1        ;X=LOW
0376: A5 FD             254           LDA  IND1+1      ;A=HIGH
0378: 20 24 ED          255           JSR  LINPRT
037B: A9 A0             256           LDA  #$A0        ;Space
037D: 20 ED FD          257           JSR  PRINT
                        258  *
0380: EE 45 02          259           INC  GESAMT
0383: D0 03             260           BNE  ZEIGE3
0385: EE 46 02          261           INC  GESAMT+1
                        262  *
                        263  *        Anzahl Zahlen/Zeile
                        264  *
0388: E6 F9             265  ZEIGE3   INC  COUNT
038A: A5 F9             266           LDA  COUNT
038C: CD 44 02          267           CMP  ANZAHL
038F: 90 09             268           BCC  WEITER1
0391: A9 8D             269  ZEIGE4   LDA  #$8D        ;Return
```

```
0393: 20 ED FD  271           JSR  PRINT
0396: A9 00     272           LDA  #$00
0398: 85 F9     273           STA  COUNT
                274  *
                275  *        Ende ?
                276  *
039A: E6 FC     277  WEITER1  INC  IND1
039C: D0 02     278           BNE  WEITER2
039E: E6 FD     279           INC  IND1+1
03A0: E6 FA     280  WEITER2  INC  IND
03A2: D0 CA     281           BNE  ZEIGE1
03A4: E6 FB     282           INC  IND+1
03A6: A5 FB     283           LDA  IND+1
03A8: C9 9A     284           CMP  #>HIMEM
03AA: 90 C2     285           BCC  ZEIGE1
03AC: A9 8D     286           LDA  #$8D       ;Return
03AE: 20 ED FD  287           JSR  PRINT
03B1: AE 45 02  288           LDX  GESAMT
03B4: AD 46 02  289           LDA  GESAMT+1
03B7: 20 24 ED  290           JSR  LINPRT
03BA: 4C 64 02  291           JMP  ENDE
                292  *
                293  * Man beachte, daß die zeit-
                294  * kritische Routine ab LOESCH1
                295  * sowie die noch kritischere
                296  * Routine ab PRIME2 jeweils
                297  * innerhalb einer Page liegen
                298  * (LOESCH1 in Page $200-$2FF
                299  * und PRIME2 in Page $300-3FF),
                300  * wodurch eine Vielzahl von
                301  * Prozessor-Takten eingespart
                302  * wird.
                303  * Da die Routine ab PRIME2
                304  * nur Zero-Page-Adressierung
                305  * benutzt, würde die Verlegung
                306  * dieser Routine in die Zero-
                307  * Page keine Geschwindigkeits-
                308  * verbesserung bringen.

--End assembly-

381 bytes

Errors: 0
```

```
                    1               ORG  38000       ;$9470
                    2    *
                    3    ****************
                    4    * PRINT USING  *
                    5    ****************
                    6    *
                    7    */BY U.STIEHL, JULY 1,1983
                    8    *
                    9    * PURPOSE: ROUND NUMBER N
                   10    * TO 1 - 3 DECIMAL PLACES AND
                   11    * PRINT N AS FORMATTED NUMBER
                   12    *
                   13    * INVOKE PRINT USING FROM
                   14    * (COMPILED) BASIC BY USR(N)
                   15    *
                   16    * SAMPLE APPLESOFT PROGRAM
                   17    *
                   18    * 10 HIMEM: 38000
                   19    * 20 PRINT CHR$(4) „BLOAD PRINT
                   20    *    USING"
                   21    * 30 POKE 10,76 : POKE 11,112
                   22    *    POKE 12,148:REM USR-JMP
                   23    * 40 INPUT N
                   24    * 45 REM LINE 50 DISPLAYS N
                   25    * 50 N = USR (N)
                   26    * 60 GOTO 40
                   27    *
                   28    * THIS WORKS WITH TASC AND
                   29    * OTHER APPLESOFT COMPILERS
                   30    *
                   31    FAC       EQU  $9D          ;9D-A2
                   32    SIGN      EQU  $A2
                   33    STRBUFF   EQU  $100
                   34    MULT10    EQU  $EA39
                   35    ROUND     EQU  $EB72
                   36    ABSOLUTE  EQU  $EBAF
                   37    ADDHALF   EQU  $E7A0
                   38    INTEGER   EQU  $EC23
                   39    NEGATIVE  EQU  $EED0
                   40    NTOSTR    EQU  $ED34
                   41    PRINT     EQU  $FDED
                   42    *
                   43    * 0 = ORIGIN = 38000 = $9470
                   44    *
9470: 4C 91 94     45              JMP  START       ;0
                   46    *
                   47    ************************
                   48    * MODIFICATION BY POKING *
                   49    ************************
                   50    *
                   51    * POKE 0+3,0 IF NO LINEFEED
                   52    *
9473: 8D           53    RETURN    HEX  8D          ;RETURN
```

```
                    54   *
                    55   * POKE 0+4,174 FOR LEADERS .....
                    56   *
9474: A0            57   FILLER   HEX  A0          ;SPACE
                    58   *
                    59   * POKE 0+5,LENGTH DEFINED
                    60   *
9475: 0B            61   LENDEF   HEX  0B          ;11
                    62   *
                    63   * LENGTH MUST BE 11 THROUGH 15
                    64   *
                    65   *
                    66   * LENGTH = 11 IS RECOMMENDED
                    67   * FOR NON EXPONENTIAL NUMBERS
                    68   *
                    69   * LENGTH = 15 IS RECOMMENDED,
                    70   * IF EXPONENTIAL NUMBERS ARE
                    71   * EXPECTED
                    72   *
                    73   * IF N IS TOO BIG OR TOO SMALL
                    74   * FOR PROPER ROUNDING, THEN
                    75   *
                    76   * 1) FILLER IS DISPLAYED,
                    77   *    IF LENGTH < 15
                    78   *
                    79   * 2) N IS DISPLAYED UNMODIFIED,
                    80   *    IF LENGTH = 15
                    81   *
                    82   * POKE 0+6,DECIMAL PLACES
                    83   *
9476: 02            84   LENDEC   HEX  02          ;2
                    85   *
                    86   * DECIMAL PLACES MUST BE 1 - 3
                    87   *
                    88   * „0.0" OR „0.00" OR „0.000"
                    89   *
                    90   ********************************
                    91   *
9477: 00            92   LENA     HEX  00          ;-
9478: 00            93   LENB     HEX  00          ;0
9479: 00            94   LENC     HEX  00          ;.
947A: 00            95   LENACT   HEX  00          ;ACTUAL
                    96   *
                    97   * ..........-0.00
                    98   * 0123456789ABCDEF
                    99   *
                   100   *
                   101   * PATTERN LENGTH = 15 BYTES
                   102   *
947B: AE AE AE      103   PATTERN  ASC  „..........."
947E: AE AE AE AE AE AE AE AE
9486: B0 AE B0      104            ASC  „0.00"
9489: B0
```

```
                        105  *
                        106  * SAVE AREA FOR NUMBER N
                        107  *
948A: 00 00 00          108  FACSAVE  HEX  000000000000
948D: 00 00 00
9490: 00                109  SIGNSAVE HEX  00
                        110  *
                        111  ******************
                        112  * PROGRAM START *
                        113  ******************
                        114  *
                        115  * SAVE FLOATING POINT ACCUMULATOR
                        116  *
9491: A0 00             117  START    LDY  #0
9493: B9 9D 00          118  FACSAVER LDA  FAC,Y
9496: 99 8A 94          119           STA  FACSAVE,Y
9499: C8                120           INY
949A: C0 07             121           CPY  #7
949C: D0 F5             122           BNE  FACSAVER
                        123  *
                        124  * CHECK PARAMETERS
                        125  *
949E: AD 75 94          126           LDA  LENDEF      ;11-15
94A1: C9 0B             127           CMP  #11
94A3: 90 04             128           BCC  PARAM
94A5: C9 10             129           CMP  #16
94A7: 90 06             130           BCC  PARAM1
94A9: A9 87             131  PARAM    LDA  #$87        ;ERROR
94AB: 20 ED FD          132           JSR  PRINT       ;BELL
94AE: 60                133           RTS
94AF: AD 76 94          134  PARAM1   LDA  LENDEC      ;1-3
94B2: C9 01             135           CMP  #1
94B4: 90 F3             136           BCC  PARAM
94B6: C9 04             137           CMP  #4
94B8: 90 03             138           BCC  ADJUST
94BA: 4C A9 94          139           JMP  PARAM
                        140  *
                        141  * ADJUST TO 1 - 3 DECIMAL PLACES
                        142  *
94BD: A9 0A             143  ADJUST   LDA  #$0A        ;2 PLACES
94BF: 8D 77 94          144           STA  LENA
94C2: A9 0B             145           LDA  #$0B
94C4: 8D 78 94          146           STA  LENB
94C7: A9 0C             147           LDA  #$0C
94C9: 8D 79 94          148           STA  LENC
94CC: AD 76 94          149           LDA  LENDEC
94CF: C9 02             150           CMP  #2
94D1: F0 19             151           BEQ  FILLOUT
94D3: C9 01             152           CMP  #1
94D5: F0 0C             153           BEQ  ADJUST2
94D7: CE 77 94          154  ADJUST1  DEC  LENA        ;3 PLACE
94DA: CE 78 94          155           DEC  LENB
94DD: CE 79 94          156           DEC  LENC
94E0: 4C EC 94          157           JMP  FILLOUT
94E3: EE 77 94          158  ADJUST2  INC  LENA        ;1 PLACE
94E6: EE 78 94          159           INC  LENB
```

```
94E9: EE 79 94  160           INC  LENC
                161  *
                162  * PREPARE EDIT PATTERN
                163  *
94EC: AD 74 94  164  FILLOUT  LDA  FILLER
94EF: A2 0E     165           LDX  #14
94F1: 9D 7B 94  166  FILLOUT1 STA  PATTERN,X
94F4: CA        167           DEX
94F5: 10 FA     168           BPL  FILLOUT1
94F7: A9 B0     169           LDA  #"0"
94F9: AE 78 94  170           LDX  LENB
94FC: 9D 7B 94  171           STA  PATTERN,X
94FF: E8        172           INX
9500: A9 AE     173           LDA  #"."
9502: 9D 7B 94  174           STA  PATTERN,X
9505: A9 B0     175  FILLOUT2 LDA  #"0"
9507: E8        176           INX
9508: E0 0F     177           CPX  #15
950A: F0 06     178           BEQ  ROUND1
950C: 9D 7B 94  179           STA  PATTERN,X
950F: 4C 05 95  180           JMP  FILLOUT2
                181  *
                182  * MULTIPLY N BY 10 OR 100 OR 1000
                183  *
9512: A5 9D     184  ROUND1   LDA  FAC
9514: 20 39 EA  185           JSR  MULT10      ;N=N*10
9517: AD 76 94  186           LDA  LENDEC
951A: C9 01     187           CMP  #1
951C: F0 11     188           BEQ  ROUND2
951E: A5 9D     189           LDA  FAC
9520: 20 39 EA  190           JSR  MULT10      ;N=N*10
9523: AD 76 94  191           LDA  LENDEC
9526: C9 02     192           CMP  #2
9528: F0 05     193           BEQ  ROUND2
952A: A5 9D     194           LDA  FAC
952C: 20 39 EA  195           JSR  MULT10      ;N=N*10
                196  *
                197  * ROUND N BY ADDITION OF 0.50
                198  * AND AND BY TRUNCATION=INTEGER
                199  *
                200  * VERY SMALL NUMBERS SUCH AS
                201  * -1.2E-20 MAY EVALUATE TO
                202  * 0.00 BY TRUNCATION
                203  *
952F: 20 72 EB  204  ROUND2   JSR  ROUND       ;ROUND N
9532: A5 A2     205           LDA  SIGN
9534: 8D 90 94  206           STA  SIGNSAVE    ;SAVE +-
9537: 20 AF EB  207           JSR  ABSOLUTE    ;N=/N/
953A: 20 A0 E7  208           JSR  ADDHALF     ;N=N+0.5
953D: 20 23 EC  209           JSR  INTEGER     ;TRUNC.
9540: AD 90 94  210           LDA  SIGNSAVE    ;LOAD +-
9543: 10 03     211           BPL  TRANSFER
9545: 20 D0 EE  212           JSR  NEGATIVE
```

```
                        213  *
                        214  * TRANSFER N AS STRING TO STRBUFF
                        215  * AND CALCULATE STRING LENGTH
                        216  *
9548: 20 34 ED          217  TRANSFER JSR  NTOSTR      ;N->STR
954B: A0 00             218           LDY  #0
954D: B9 00 01          219  STRLEN   LDA  STRBUFF,Y
9550: F0 0D             220           BEQ  STRLEN1
9552: 09 80             221           ORA  #$80
9554: 99 00 01          222           STA  STRBUFF,Y
9557: C9 C5             223           CMP  #„E"         ;EXPON.
9559: F0 49             224           BEQ  TOOBIG
955B: C8                225           INY
955C: 4C 4D 95          226           JMP  STRLEN
955F: 8C 7A 94          227  STRLEN1  STY  LENACT
9562: CC 75 94          228           CPY  LENDEF
9565: F0 02             229           BEQ  MOVEOKAY
9567: B0 3B             230           BCS  TOOBIG
                        231  *
                        232  * MOVE STRBUFF TO EDIT PATTERN
                        233  *
9569: A2 0F             234  MOVEOKAY LDX  #$0F
956B: AC 7A 94          235           LDY  LENACT
956E: 88                236  MOVE     DEY
956F: 30 1E             237           BMI  DISPLAY
9571: CA                238  MOVE1    DEX
9572: EC 79 94          239           CPX  LENC         ;„."
9575: F0 FA             240           BEQ  MOVE1        ;SKIP
9577: B9 00 01          241           LDA  STRBUFF,Y
957A: C9 AD             242           CMP  #„-"
957C: F0 06             243           BEQ  MOVE3
957E: 9D 7B 94          244  MOVE2    STA  PATTERN,X
9581: 4C 6E 95          245           JMP  MOVE
9584: EC 78 94          246  MOVE3    CPX  LENB
9587: 90 F5             247           BCC  MOVE2        ;OKAY
                        248  *
                        249  * SPECIAL CASES OF NEGATIVE N
                        250  * -0.01  > =  N  < = -0.09
                        251  * -0.10  > =  N  < = -0.99
                        252  *
9589: AE 77 94          253           LDX  LENA
958C: 9D 7B 94          254           STA  PATTERN,X
                        255  *
                        256  * DISPLAY FORMATTED NUMBER
                        257  *
958F: A9 0F             258  DISPLAY  LDA  #15
9591: 38                259           SEC
9592: ED 75 94          260           SBC  LENDEF
9595: A8                261           TAY
9596: B9 7B 94          262  DISPLAY1 LDA  PATTERN,Y
9599: 20 ED FD          263           JSR  PRINT
959C: C8                264           INY
959D: C0 0F             265           CPY  #15
```

```
959F: D0 F5     266           BNE  DISPLAY1
95A1: 4C E5 95  267           JMP  END
                268  *
                269  * N TOO BIG: EXPONENTIATION OR
                270  * ACTUAL LENGTH > DEFINED LENGTH
                271  *
95A4: AC 75 94  272  TOOBIG   LDY  LENDEF
95A7: C0 0F     273           CPY  #15         ;MAXIMUM
95A9: F0 0C     274           BEQ  EXP0
                275  *
                276  * DISPLAY SPACES OR LEADERS
                277  *
95AB: AD 74 94  278  TOOBIG1  LDA  FILLER
95AE: 20 ED FD  279           JSR  PRINT
95B1: 88        280           DEY
95B2: D0 F7     281           BNE  TOOBIG1
95B4: 4C E5 95  282           JMP  END
                283  *
                284  * DISPLAY IN SCIENTIFIC NOTATION
                285  *
95B7: 20 EF 95  286  EXP0     JSR  END1
95BA: 20 34 ED  287           JSR  NTOSTR
95BD: A2 00     288           LDX  #0
95BF: BD 00 01  289  EXP01    LDA  STRBUFF,X
95C2: F0 03     290           BEQ  EXP02
95C4: E8        291           INX
95C5: D0 F8     292           BNE  EXP01
95C7: E0 0F     293  EXP02    CPX  #15
95C9: F0 0A     294           BEQ  EXP03
95CB: AD 74 94  295           LDA  FILLER
95CE: 20 ED FD  296           JSR  PRINT
95D1: E8        297           INX
95D2: 4C C7 95  298           JMP  EXP02
95D5: A2 FF     299  EXP03    LDX  #$FF
95D7: E8        300  EXP04    INX
95D8: BD 00 01  301           LDA  STRBUFF,X
95DB: F0 08     302           BEQ  END
95DD: 09 80     303           ORA  #$80
95DF: 20 ED FD  304           JSR  PRINT
95E2: 4C D7 95  305           JMP  EXP04
                306  *
                307  * ENDMARK RETURN YES/NO ?
                308  *
95E5: AD 73 94  309  END      LDA  RETURN
95E8: C9 8D     310           CMP  #$8D
95EA: D0 03     311           BNE  END1
95EC: 20 ED FD  312           JSR  PRINT
                313  *
                314  * RESTORE FLOATING POINT ACCU
                315  *
95EF: A0 00     316  END1     LDY  #0
95F1: B9 8A 94  317  END2     LDA  FACSAVE,Y
95F4: 99 9D 00  318           STA  FAC,Y
95F7: C8        319           INY
95F8: C0 07     320           CPY  #7
95FA: D0 F5     321           BNE  END2
95FC: 60        322           RTS
```

```
100  REM *** PRINT USING DEMO ***
110  TEXT : HOME : HIMEM: 38000
120  PRINT  CHR$ (4)„BLOAD PRINT.USING"
130  POKE 10,76: POKE 11,112:: POKE 12,148: REM USR-JMP
140  REM * GENERATE TEST NUMBERS *
150  DIM A(100)
160 Z =  - 1
170 B = 9.87654321:B = B * B
180  FOR Y =  - 50 TO 50 STEP 2
190 Z = Z + 1
200 A(Z) = B * Y * Y * Y
210  NEXT Y
220  REM * DISPLAY NUMBERS *
230 O = 38000: REM ORIGIN
240  POKE O + 3,0: REM  NO LINEFEED
250  POKE O + 4,174: REM LEADERS
260  PRINT : PRINT „1 DECIMAL PLACE"
270  POKE O + 6,1: GOSUB 350
280  PRINT : PRINT „2 DECIMAL PLACES"
290  POKE O + 6,2: GOSUB 350
300  PRINT : PRINT „3 DECIMAL PLACES"
310  POKE O + 6,3: GOSUB 350
320  END
330  REM *** LENGTH 11 ***
340  REM LENGTH < 15 EXCLUDES EXPONENTIATION
350  POKE O + 5,11: GOSUB 420
360  PRINT
370  REM *** LENGTH 15 ***
380  REM LENGTH = 15 INCLUDES EXPONENTIATION
390  POKE O + 5,15: GOSUB 420
400  PRINT
410  RETURN
420  FOR Z = O TO 50
430 N = A(Z)
440  REM * DISPLAY FORMATTED N *
450  PRINT „VALUE";
460 N =  USR (N)
470  REM * DISPLAY UNFORMATTED N *
480  HTAB 25: PRINT N
490  NEXT Z
500  RETURN
```

PRINT.USING-Beispiele:

```
VALUE....-4334809.29  -4334809.29
VALUE....-2219422.36  -2219422.36
VALUE.....-936318.81  -936318.807
VALUE.....-277427.79  -277427.795
VALUE......-34678.47  -34678.4743
VALUE..........0.00  0
VALUE.......34678.47  34678.4743
VALUE......277427.79  277427.795
VALUE......936318.81  936318.807
VALUE.....2219422.36  2219422.36
VALUE.....4334809.29  4334809.29
```

5. Text- und Grafikspeicher

Über den Text- und Grafikspeicher wurde bereits in Kap. 2.1.1 und Kap. 3.1.5 geschrieben, so daß wir uns hier auf die Darstellung der Softswitches beschränken können, die überwiegend nur für den Apple IIe gelten.

LDA	$C050	Grafik-Modus
LDA	$C051	Text-Modus
LDA	$C052	„Ganzer" Bildschirm (Entweder nur Text oder nur Grafik)
LDA	$C053	„Geteilter" Bildschirm (Text *und* Grafik)
LDA	$C054	Seite 1: $0400-$07FF Main. oder $2000-$3FFF Main. (40 + 80 Z/Z)
LDA	$C055	Seite 2: $0800-$0BFF oder $4000-$5FFF, nur wenn 40 Z/Z Seite 2: $0400-$07FF Aux. oder $2000-$3FFF Aux., nur wenn 80 Z/Z (nur IIe)
LDA	$C056	Lores-Modus (bedeutet auch 40 Z/Z-Text)
LDA	$C057	Hires-Modus
LDA	$C01A	BPL Grafik-Modus, BMI Text-Modus (nur IIe)
LDA	$C01B	BPL Ganzer Bildschirm, BMI Geteilter Bildschirm (nur IIe)
LDA	$C01C	BPL Seite 1 aktiv, BMI Seite 2 aktiv
LDA	$C01D	BPL Lores-Modus aktiv, BMI Hires-Modus aktiv
STA	$C000	40 Z/Z Nur-Main-Store-Modus (nur IIe)
STA	$C001	80 Z/Z Auxiliary-Store-Modus: Main *und* Aux. (nur IIe)
STA	$C00C	40 Z/Z Darstellung
STA	$C00D	80 Z/Z Darstellung
LDA	$C05E	80 Z/Z Annunciator Auxiliary-Store-Modus: Main *und* Aux. (nur IIe)
LDA	$C05F	80 Z/Z Annunciator Nur-Main-Store-Modus

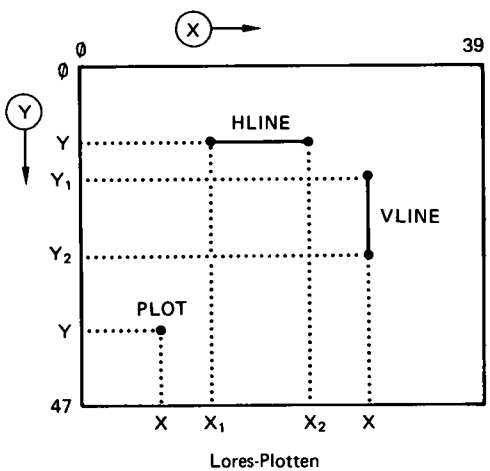
```
LDA   $C018    BPL Nur-Main-Store-Modus aktiv, BMI Auxiliary-Store-
               Modus aktiv
LDA   $C01F    BPL 40 Z/Z Darstellung aktiv, BMI 80 Z/Z Darstellung
               aktiv

STA   $C00E    Erstzeichensatz (mit FLASH)
STA   $C00F    Zweitzeichensatz (ohne FLASH)
LDA   $C01E    BPL Erstzeichensatz, BMI Zweitzeichensatz
```

Um 80-Zeichen-Textdarstellung sowie doppelte Lores- und Hires-Grafik zu ermöglichen, muß ein ganzes Sortiment von Softswitches betätigt werden, die man am besten den nachfolgenden Programmen entnehmen möge.

Für den Fall, daß der Bildschirm, insbesondere bei Hires-Grafik, nicht flackerfrei sein sollte, gibt es noch folgende Softswitch-Routine (Vertical Blanking), die allerdings im ROM (aus Platzmangel!) selbst nicht benutzt wird.

```
BLANK1 LDA   $C019
       BPL   BLANK1
BLANK2 LDA   $C019
       BMI   BLANK2
(Jetzt Screen-Store ausführen)
```



Lores-Plotten

Lores-Plotten (siehe hierzu Seite 43–44 und Seite 221–226).

```
                        1               ORG  $0300
                        2    *
                        3    * SCREENDUMP
                        4    * ==========
                        5    *
                        6    * Für 40-Z/Z-Bildschirm
                        7    *
                        8    IND        EQU  $CE
                        9    TOP        EQU  $22
                        10   CH         EQU  $24
                        11   CV         EQU  $25
                        12   VTAB       EQU  $FC22
                        13   PRINT      EQU  $FDED
                        14   INPORT     EQU  $FE8B
                        15   OUTPORT    EQU  $FE95
                        16   CONNECT    EQU  $3EA
                        17   *
0300: 4C 04 03          18              JMP  START
                        19   *
                        20   * Hier Slot-Nummer poken 1-7
                        21   *
0303: 01                22   SLOT       HEX  01
                        23   *
                        24   * Bildschirmfenster auf
                        25   * Zeile 24 begrenzen, damit
                        26   * die oberen 23 Zeilen nicht
                        27   * verrutschen. Ferner „P"
                        28   * anzeigen als Warnung, daß
                        29   * (P)rinter eingeschaltet
                        30   * sein muß.
                        31   *
0304: A5 22             32   START      LDA  TOP
0306: 8D 97 03          33              STA  TOPSAVE
0309: A9 00             34              LDA  #0
030B: 85 24             35              STA  CH
030D: A9 17             36              LDA  #23
030F: 85 25             37              STA  CV
0311: 20 22 FC          38              JSR  VTAB
0314: A9 D0             39              LDA  #„P"
0316: 20 ED FD          40              JSR  PRINT
0319: A9 17             41              LDA  #23
031B: 85 22             42              STA  TOP
                        43   *
                        44   * PR#0 IN#0 ausführen und
                        45   * dann mit PR#S Drucker ein-
                        46   * schalten.
                        47   *
031D: A9 00             48              LDA  #0
031F: 20 8B FE          49              JSR  INPORT
0322: AD 03 03          50              LDA  SLOT
0325: 20 95 FE          51              JSR  OUTPORT
                        52   *
                        53   * Vorab ein Return senden,
                        54   * um Druckerpuffer zu leeren.
                        55   *
```

```
0328: A9 8D      56             LDA  #$8D
032A: 20 ED FD   57             JSR  PRINT
                 58   *
032D: A9 00      59             LDA  #0
032F: 8D 98 03   60             STA  ZEILE
0332: 20 7F 03   61   LOOP1     JSR  BASCALC
0335: A0 00      62             LDY  #0
0337: B1 CE      63   LOOP2     LDA  (IND),Y
0339: C9 A0      64             CMP  #$A0        ;Space
033B: D0 07      65             BNE  LOOP3
033D: C8         66             INY
033E: C0 28      67             CPY  #40
0340: D0 F5      68             BNE  LOOP2
0342: F0 1E      69             BEQ  LOOP6
0344: A0 00      70   LOOP3     LDY  #0
0346: B1 CE      71   LOOP4     LDA  (IND),Y
0348: 09 80      72             ORA  #$80
                 73   *
                 74   * Zeilen, die nur Leertasten
                 75   * enthalten, werden nicht
                 76   * ausgedruckt!
                 77   *
034A: C9 A0      78             CMP  #$A0
034C: B0 07      79             BCS  LOOP5
034E: C0 00      80             CPY  #0
0350: F0 10      81             BEQ  LOOP6
0352: 18         82             CLC
0353: 69 40      83             ADC  #$40
0355: 20 ED FD   84   LOOP5     JSR  PRINT
0358: C8         85             INY
0359: C0 28      86             CPY  #40
035B: D0 E9      87             BNE  LOOP4
035D: A9 8D      88             LDA  #$8D
035F: 20 ED FD   89             JSR  PRINT
                 90   *
                 91   * Es werden die Zeilen 0-16,
                 92   * d.h. insgesamt 17 Zeilen,
                 93   * ausgedruckt. Wert ggf.
                 94   * erhöhen auf bis maximal #23
                 95   *
0362: EE 98 03   96   LOOP6     INC  ZEILE
0365: AD 98 03   97             LDA  ZEILE
0368: C9 11      98             CMP  #17         ;0-16
036A: D0 C6      99             BNE  LOOP1
036C: A9 8D     100             LDA  #$8D
036E: 20 ED FD  101             JSR  PRINT
0371: AD 97 03  102             LDA  TOPSAVE
0374: 85 22     103             STA  TOP
0376: A9 00     104             LDA  #0
0378: 20 95 FE  105             JSR  OUTPORT
037B: 20 EA 03  106             JSR  CONNECT
037E: 60        107             RTS
                108   *
                109   * Dem Monitor entnommen
                110   *
```

```
037F: 48          111  BASCALC  PHA
0380: 4A          112           LSR
0381: 29 03       113           AND  #$03
0383: 09 04       114           ORA  #$04
0385: 85 CF       115           STA  IND+1
0387: 68          116           PLA
0388: 29 18       117           AND  #$18
038A: 90 02       118           BCC  BASCALC1
038C: 69 7F       119           ADC  #$7F
038E: 85 CE       120  BASCALC1 STA  IND
0390: 0A          121           ASL
0391: 0A          122           ASL
0392: 05 CE       123           ORA  IND
0394: 85 CE       124           STA  IND
0396: 60          125           RTS
                  126  *
0397: 00          127  TOPSAVE  HEX  00
0398: 00          128  ZEILE    HEX  00
```

5.1.2. RDKEY

```
                  1             ORG  768
                  2    *
                  3    * RDKEY
                  4    * ======
                  5    *
                  6    * Für Menüauswahl in Basic-
                  7    * programm: ON N GOTO ...
                  8    *
                  9    RDKEY     EQU  $FD0C
                  10   *
0300: 20 0C FD    11   GET1      JSR  RDKEY
0303: A2 01       12            LDX  #1
0305: DD 19 03    13   GET2      CMP  AUSWAHL1-1,X
0308: F0 0C       14            BEQ  GET3
030A: DD 23 03    15            CMP  AUSWAHL2-1,X
030D: F0 07       16            BEQ  GET3
030F: E8          17            INX
0310: E4 0A       18            CPX  AUSWAHL2-AUSWAHL1
0312: D0 F1       19            BNE  GET2
0314: F0 EA       20            BEQ  GET1
0316: 8E FF 02    21   GET3      STX  767        ;N
0319: 60          22            RTS
031A: C1 C2 C3    23   AUSWAHL1 ASC  „ABCDEFGHI"
031D: C4 C5 C6 C7 C8 C9
0323: 00          24            HEX  00
0324: E1 E2 E3    25   AUSWAHL2 ASC  „abcdefghi"
0327: E4 E5 E6 E7 E8 E9
032D: 00          26            HEX  00
```

```
                        1            ORG  37666
                        2    *-------------------------------
                        3    *      BILDSCHIRMMASKE
                        4    * für 40 Z/Z Apple II Plus
                        5    * mit Kleinschreibumrüstsatz
                        6    *     (C) U.Stiehl 1984
                        7    *-------------------------------
9322: 4C 64 93          8            JMP  START       ;37666
                        9    *--------POKE-PARAMETER---------
9325: 00                10   ZAHLFL   DFB  $00         ;0-1
9326: 00                11   CLRFL    DFB  $00         ;0-1
9327: 01                12   VTAB     DFB  $01         ;1-24
9328: 01                13   HTAB     DFB  $01         ;1-38
9329: 27                14   LAENGE   DFB  $27         ;1-39
                        15   *--------DUMMY-ZEILE----------
932A: 99                16   POKER    DFB  $99         ;STA,Y
                        17   *--------$0879---------------
932B: 79                18   STRADRL  DFB  $79         ;LOWBYT
932C: 08                19   STRADRH  DFB  $08         ;HIGHBY
932D: 60                20            DFB  $60         ;RTS
                        21   *--------PEEK-PARAMETER-------
932E: 00                22   STRLEN   DFB  $00         ;LAENGE
                        23   *--------40 Z/Z--------------
932F: 00 04             24   ZEILEN   DA   $0400       ;1.LINE
9331: 80 04             25            DA   $0480       ;2.LINE
9333: 00 05             26            DA   $0500       ;USW.
9335: 80 05             27            DA   $0580
9337: 00 06             28            DA   $0600
9339: 80 06             29            DA   $0680
933B: 00 07             30            DA   $0700
933D: 80 07             31            DA   $0780
933F: 28 04             32            DA   $0428
9341: A8 04             33            DA   $04A8
9343: 28 05             34            DA   $0528
9345: A8 05             35            DA   $05A8
9347: 28 06             36            DA   $0628
9349: A8 06             37            DA   $06A8
934B: 28 07             38            DA   $0728
934D: A8 07             39            DA   $07A8
934F: 50 04             40            DA   $0450
9351: D0 04             41            DA   $04D0
9353: 50 05             42            DA   $0550
9355: D0 05             43            DA   $05D0
9357: 50 06             44            DA   $0650
9359: D0 06             45            DA   $06D0
935B: 50 07             46            DA   $0750
935D: D0 07             47            DA   $07D0       ;24.
                        48   *--------ALLGEMEINE LABELS------
                        49   PUFFER   EQU  $280        ;MITTE
                        50   KEYIN    EQU  $C000
                        51   KEYCLR   EQU  $C010
935F: 00                52   TASTE    DFB  $00
9360: 00                53   HT       DFB  $00
```

```
9361: 00          54   VT       DFB  $00
9362: 00          55   LENGTH   DFB  $00
9363: 00          56   INSFLG   DFB  $00
                  57   *--------START-----------------
9364: AD 28 93    58   START    LDA  HTAB
9367: 8D 60 93    59            STA  HT
936A: AD 27 93    60            LDA  VTAB
936D: 8D 61 93    61            STA  VT
9370: AD 29 93    62            LDA  LAENGE
9373: 8D 62 93    63            STA  LENGTH
9376: CE 60 93    64            DEC  HT          ;HTAB-1
9379: CE 61 93    65            DEC  VT          ;VTAB-1
937C: CE 62 93    66            DEC  LENGTH      ;LEN-1
937F: 4C B5 95    67            JMP  CHECK
                  68   *--------SCREENADRESSE-----------
9382: AD 61 93    69   SUCH     LDA  VT
9385: 0A          70            ASL              ;2XVT
9386: AA          71            TAX
9387: A0 01       72            LDY  #1          ;ADR+1
9389: BD 2F 93    73            LDA  ZEILEN,X
938C: 18          74            CLC
938D: 6D 60 93    75            ADC  HT
9390: 99 A4 93    76            STA  STORE,Y     ;LOW
9393: 99 A8 93    77            STA  LOAD,Y      ;BYTE
9396: C8          78            INY              ;ADR+2
9397: E8          79            INX
9398: BD 2F 93    80            LDA  ZEILEN,X
939B: 99 A4 93    81            STA  STORE,Y     ;HIGH
939E: 99 A8 93    82            STA  LOAD,Y      ;BYTE
93A1: 4C AC 93    83            JMP  BEGINN
                  84   *--------STORE/LOAD-------------
93A4: 9D FF FF    85   STORE    STA  $FFFF,X     ;FIKTIV
93A7: 60          86            RTS
93A8: BD FF FF    87   LOAD     LDA  $FFFF,X     ;FIKTIV
93AB: 60          88            RTS
                  89   *--------ENDMARKER < ------------
93AC: A9 BC       90   BEGINN   LDA  #$BC        ;„<"
93AE: AE 62 93    91            LDX  LENGTH
93B1: E8          92            INX
93B2: 20 A4 93    93            JSR  STORE
93B5: AD 26 93    94            LDA  CLRFL
93B8: C9 01       95            CMP  #1
93BA: F0 0C       96            BEQ  NONCLR
93BC: A9 A0       97            LDA  #$A0        ;SPACE
93BE: CA          98            DEX
93BF: 20 A4 93    99   FELD     JSR  STORE
93C2: CA          100           DEX
93C3: 10 FA       101           BPL  FELD
93C5: 4C D4 93    102           JMP  BEG1
93C8: AE 62 93    103  NONCLR   LDX  LENGTH
93CB: 20 A8 93    104  NONCL1   JSR  LOAD
93CE: 9D 80 02    105           STA  PUFFER,X
93D1: CA          106           DEX
93D2: 10 F7       107           BPL  NONCL1
```

```
93D4: A2 00      108 BEG1     LDX  #0
                 109 *--------REFRESH----------------
93D6: 8A         110 FRISCH   TXA
93D7: A8         111          TAY              ;XSAVE
93D8: AE 62 93   112          LDX  LENGTH
93DB: 20 A8 93   113 FRESH    JSR  LOAD
93DE: C9 A0      114          CMP  #$A0        ;NORM.
93E0: B0 1A      115          BCS  NEXT1
93E2: C9 80      116          CMP  #$80        ;80-9F
93E4: 90 06      117          BCC  INVKL
93E6: 18         118 INVGR    CLC
93E7: 69 60      119          ADC  #$60        ;INVGR
93E9: 4C F9 93   120          JMP  NEXT
93EC: C9 20      121 INVKL    CMP  #$20        ;0-1F
93EE: B0 06      122          BCS  INVPIE
93F0: 18         123          CLC
93F1: 69 C0      124          ADC  #$C0        ;INVPIE
93F3: 4C F9 93   125          JMP  NEXT
93F6: 18         126 INVPIE   CLC
93F7: 69 80      127          ADC  #$80        ;20-3F
93F9: 20 A4 93   128 NEXT     JSR  STORE
93FC: CA         129 NEXT1    DEX
93FD: 10 DC      130          BPL  FRESH
                 131 *--------CURSOR---------------
93FF: 98         132          TYA
9400: AA         133          TAX
9401: 20 A8 93   134          JSR  LOAD
9404: C9 E0      135          CMP  #$E0        ;KLEIN
9406: B0 05      136          BCS  KLEIN
9408: 29 3F      137          AND  #$3F
940A: 4C 10 94   138          JMP  CURSOR
940D: 38         139 KLEIN    SEC
940E: E9 60      140          SBC  #$60        ;E0-FF
9410: 20 A4 93   141 CURSOR   JSR  STORE
9413: A0 00      142          LDY  #0
                 143 *--------EINGABE--------------
9415: AD 00 C0   144 KEY      LDA  KEYIN
9418: 8D 5F 93   145          STA  TASTE
941B: 10 F8      146          BPL  KEY         ;<128
941D: 8C 10 C0   147          STY  KEYCLR
9420: AD 5F 93   148          LDA  TASTE
9423: C9 92      149          CMP  #$92        ;CRTL-R
9425: F0 2F      150          BEQ  RESTORE
9427: C9 82      151          CMP  #$82        ;CRTL-B
9429: F0 2E      152          BEQ  ANFANG
942B: C9 85      153          CMP  #$85        ;CTRL-E
942D: F0 2D      154          BEQ  ENDE
942F: C9 84      155          CMP  #$84        ;CRTL-D
9431: F0 2C      156          BEQ  DELETE
9433: C9 89      157          CMP  #$89        ;CRTL-I
9435: F0 2B      158          BEQ  INSERT
9437: C9 95      159          CMP  #$95        ;CUR=>
9439: F0 2A      160          BEQ  RECHTS
943B: C9 88      161          CMP  #$88        ;<=CUR
```

```
943D: F0 29      162            BEQ  LINKS
943F: C9 8D      163            CMP  #$8D        ;RTN
9441: F0 28      164            BEQ  RETURN
9443: C9 BA      165            CMP  #$BA        ;„:"
9445: F0 CE      166            BEQ  KEY
9447: C9 AC      167            CMP  #$AC        ;„,"
9449: F0 CA      168            BEQ  KEY
944B: C9 A2      169            CMP  #$A2        ;„
944D: F0 C6      170            BEQ  KEY
944F: C9 A0      171            CMP  #$A0        ;<SPACE
9451: 90 C2      172            BCC  KEY
9453: 4C 6E 94   173            JMP  ZABU
                 174  *--------REDIGEREN----------
9456: 4C 9B 94   175  RESTORE    JMP  RES
9459: 4C B4 94   176  ANFANG     JMP  ANF
945C: 4C B9 94   177  ENDE       JMP  END
945F: 4C 0D 95   178  DELETE     JMP  DEL
9462: 4C 41 95   179  INSERT     JMP  INS
9465: 4C 49 95   180  RECHTS     JMP  RE
9468: 4C 5A 95   181  LINKS      JMP  LI
946B: 4C 66 95   182  RETURN     JMP  RTN
                 183  *--------ZAHL/STRING----------
946E: AC 25 93   184  ZABU       LDY  ZAHLFL
9471: F0 25      185             BEQ  STRING
9473: AD 5F 93   186  ZAHL       LDA  TASTE
9476: C9 BA      187             CMP  #$BA        ;NACH.9
9478: B0 0B      188             BCS  ZAHL1
947A: C9 AE      189             CMP  #$AE        ;VOR„."
947C: 90 07      190             BCC  ZAHL1
947E: C9 AF      191             CMP  #$AF        ;„/"
9480: F0 03      192             BEQ  ZAHL1
9482: 4C BF 94   193             JMP  NORMAL
9485: C9 A0      194  ZAHL1      CMP  #$A0        ;SPACE
9487: F0 0F      195             BEQ  STRING
9489: C9 C5      196             CMP  #$C5        ;„E"
948B: F0 0B      197             BEQ  STRING
948D: C9 AB      198             CMP  #$AB        ;„+"
948F: F0 07      199             BEQ  STRING
9491: C9 AD      200             CMP  #$AD        ;„-"
9493: F0 03      201             BEQ  STRING
9495: 4C 15 94   202             JMP  KEY
9498: 4C BF 94   203  STRING     JMP  NORMAL
                 204  *--------RESTORE-----------------
949B: AD 26 93   205  RES        LDA  CLRFL
949E: C9 01      206             CMP  #1
94A0: F0 03      207             BEQ  RES1
94A2: 4C D6 93   208             JMP  FRISCH
94A5: AE 62 93   209  RES1       LDX  LENGTH
94A8: BD 80 02   210  RES2       LDA  PUFFER,X
94AB: 20 A4 93   211             JSR  STORE
94AE: CA         212             DEX
94AF: 10 F7      213             BPL  RES2
94B1: 4C D4 93   214             JMP  BEG1
```

```
                        215  *--------ZEILENANFANG----------
94B4: A2 00             216  ANF       LDX  #0
94B6: 4C D6 93          217            JMP  FRISCH
                        218  *--------ZEILENENDE------------
94B9: AE 62 93          219  END       LDX  LENGTH
94BC: 4C D6 93          220            JMP  FRISCH
                        221  *--------INSERT----------------
94BF: AC 63 93          222  NORMAL    LDY  INSFLG
94C2: F0 37             223            BEQ  NORM1
94C4: 8E 63 93          224            STX  INSFLG
94C7: EC 62 93          225            CPX  LENGTH
94CA: F0 24             226            BEQ  INSER1
94CC: AE 62 93          227            LDX  LENGTH
94CF: 20 A8 93          228            JSR  LOAD
94D2: C9 A0             229            CMP  #$A0        ;SPACE
94D4: D0 1A             230            BNE  INSER1
94D6: CA                231            DEX
94D7: 20 A8 93          232  INSER     JSR  LOAD
94DA: E8                233            INX
94DB: 20 A4 93          234            JSR  STORE
94DE: CA                235            DEX
94DF: CA                236            DEX
94E0: EC 63 93          237            CPX  INSFLG
94E3: 10 F2             238            BPL  INSER
94E5: AE 63 93          239            LDX  INSFLG
94E8: A9 01             240            LDA  #1
94EA: 8D 63 93          241            STA  INSFLG
94ED: 4C FB 94          242            JMP  NORM1
94F0: AE 63 93          243  INSER1    LDX  INSFLG
94F3: A9 00             244            LDA  #0
94F5: 8D 63 93          245            STA  INSFLG
94F8: 4C D6 93          246            JMP  FRISCH
                        247  *--------NORMALE EINGABE--------
94FB: AD 5F 93          248  NORM1     LDA  TASTE
94FE: 20 A4 93          249            JSR  STORE
9501: E8                250            INX
9502: EC 62 93          251            CPX  LENGTH
9505: 90 03             252            BCC  NORM2
9507: F0 01             253            BEQ  NORM2
9509: CA                254            DEX
950A: 4C D6 93          255  NORM2     JMP  FRISCH
                        256  *--------DELETE----------------
950D: EC 62 93          257  DEL       CPX  LENGTH
9510: D0 08             258            BNE  DEL1
9512: A9 A0             259            LDA  #$A0        ;SPACE
9514: 20 A4 93          260            JSR  STORE
9517: 4C D6 93          261            JMP  FRISCH
951A: 8E 63 93          262  DEL1      STX  INSFLG
951D: E8                263            INX
951E: 20 A8 93          264  DELE      JSR  LOAD
9521: CA                265            DEX
9522: 20 A4 93          266            JSR  STORE
9525: E8                267            INX
9526: E8                268            INX
9527: EC 62 93          269            CPX  LENGTH
```

```
952A: 90 F2     270           BCC  DELE
952C: F0 F0     271           BEQ  DELE
952E: A9 A0     272           LDA  #$A0        ;SPACE
9530: AE 62 93  273           LDX  LENGTH
9533: 20 A4 93  274           JSR  STORE
9536: AE 63 93  275           LDX  INSFLG
9539: A9 00     276           LDA  #0
953B: 8D 63 93  277           STA  INSFLG
953E: 4C D6 93  278           JMP  FRISCH
                279 *---------INSERT-FLAG-------------
9541: A0 01     280 INS       LDY  #1
9543: 8D 63 93  281           STA  INSFLG
9546: 4C D6 93  282           JMP  FRISCH
                283 *---------RECHTS-PFEIL------------
9549: A0 00     284 RE        LDY  #0
954B: 8C 63 93  285           STY  INSFLG
954E: E8        286           INX
954F: EC 62 93  287           CPX  LENGTH
9552: 90 03     288           BCC  RE2
9554: F0 01     289           BEQ  RE2
9556: CA        290           DEX
9557: 4C D6 93  291 RE2       JMP  FRISCH
                292 *---------LINKS-PFEIL-------------
955A: A0 00     293 LI        LDY  #0
955C: 8C 63 93  294           STY  INSFLG
955F: CA        295           DEX
9560: 10 01     296           BPL  LI1
9562: E8        297           INX
9563: 4C D6 93  298 LI1       JMP  FRISCH
                299 *---------RETURN-----------------
9566: 20 A8 93  300 RTN       JSR  LOAD
9569: C9 A0     301           CMP  #$A0
956B: B0 17     302           BCS  RTN3
956D: C9 80     303           CMP  #$80
956F: 90 06     304           BCC  RTN1
9571: 18        305 RTN0      CLC
9572: 69 60     306           ADC  #$60
9574: 4C 84 95  307           JMP  RTN3
9577: C9 20     308 RTN1      CMP  #$20
9579: B0 06     309           BCS  RTN2
957B: 18        310           CLC
957C: 69 C0     311           ADC  #$C0
957E: 4C 84 95  312           JMP  RTN3
9581: 18        313 RTN2      CLC
9582: 69 80     314           ADC  #$80
9584: 20 A4 93  315 RTN3      JSR  STORE
                316 *---------NULLSTRING-------------
9587: AE 62 93  317           LDX  LENGTH
958A: 20 A8 93  318 NULLSTR   JSR  LOAD
958D: C9 A0     319           CMP  #$A0        ;SPACE
958F: D0 11     320           BNE  OKAY
9591: CA        321           DEX
9592: 10 F6     322           BPL  NULLSTR
9594: A0 01     323           LDY  #1
```

```
9596: A9 20     324          LDA  #$20        ;SPACE
9598: 20 2A 93  325          JSR  POKER       ;STRIPP
959B: 88        326          DEY
959C: A9 01     327          LDA  #1          ;LAENGE
959E: 8D 2E 93  328          STA  STRLEN
95A1: 60        329          RTS
                330  *--------SPEICHERN----------------
95A2: 8A        331  OKAY    TXA
95A3: A8        332          TAY
95A4: C8        333          INY              ;LAENGE
95A5: 8C 2E 93  334          STY  STRLEN      ;DANACH
95A8: 20 A8 93  335  SAVE    JSR  LOAD
95AB: 29 7F     336          AND  #$7F        ;STRIPP
95AD: 20 2A 93  337          JSR  POKER
95B0: 88        338          DEY
95B1: CA        339          DEX
95B2: 10 F4     340          BPL  SAVE
95B4: 60        341          RTS
                342  *--------RANGE-CHECK-------------
95B5: AD 60 93  343  CHECK   LDA  HT          ;HT<0
95B8: 30 34     344          BMI  ERROR
95BA: AD 61 93  345          LDA  VT          ;VT<0
95BD: 30 2F     346          BMI  ERROR
95BF: AD 62 93  347          LDA  LENGTH      ;LEN<0
95C2: 30 2A     348          BMI  ERROR
95C4: AD 60 93  349          LDA  HT          ;HT>=38
95C7: C9 26     350          CMP  #38
95C9: B0 23     351          BCS  ERROR
95CB: AD 61 93  352          LDA  VT          ;VT>=24
95CE: C9 18     353          CMP  #24
95D0: B0 1C     354          BCS  ERROR
95D2: 18        355          CLC
95D3: AD 62 93  356          LDA  LENGTH
95D6: 6D 60 93  357          ADC  HT
95D9: C9 27     358          CMP  #39
95DB: B0 11     359          BCS  ERROR
95DD: AD 26 93  360          LDA  CLRFL
95E0: C9 02     361          CMP  #2
95E2: B0 0A     362          BCS  ERROR
95E4: AD 25 93  363          LDA  ZAHLFL
95E7: C9 02     364          CMP  #2
95E9: B0 03     365          BCS  ERROR
95EB: 4C 82 93  366          JMP  SUCH        ;OKAY!
95EE: A9 45     367  ERROR   LDA  #$45        ;FLASH-E
95F0: 8D 00 04  368          STA  $0400
95F3: 60        369          RTS

--End assembly-

722 bytes

Errors: 0
```

```
100  PRINT : PRINT  CHR$ (4)„BLOAD BILDSCHIRMMASKE,A37666": HIMEM:
    37666
110  PRINT „COPYRIGHT 84 ULRICH STIEHL * HEIDELBERG"
120  GOTO 160
130 X$ = „":X$ =  LEFT$ („12345678901234567890123456789012345678901234567890",
    PEEK (37678)): RETURN
140  REM *** DIE PROGRAMM-ZEILEN 100-130 DUERFEN NICHT GEAENDERT
    WERDEN !!!!!
150  REM ** PARAMETER INPUT-CALL,ZAHLFLAG,CLEARFLAG,VTAB,HTAB,LENGTH
160 IN = 37666:ZF = 37669:CF = 37670:VT = 37671:HT = 37672:LE = 37673
170  ONERR  GOTO 180
180  HOME : HTAB 15: INVERSE : PRINT „MASKE-DEMO"
190  REM **** 1.STRING
200  VTAB 3: HTAB 1: PRINT „20-STRING"
210  POKE ZF,0: POKE CF,0: POKE VT,3: POKE HT,11: POKE LE,20: CALL IN:
    GOSUB 130:S1$ = X$
220  REM **** 2.STRING
230  VTAB 6: HTAB 1: PRINT „39-STRING"
240  POKE VT,7: POKE HT,1: POKE LE,39: CALL IN: GOSUB 130:S2$ = X$
250  REM **** 3.STRING=ZAHL
260  VTAB 10: HTAB 10: PRINT „15-ZAHL"
270  POKE ZF,1: POKE VT,10: POKE HT,18: POKE LE,15: CALL IN: GOSUB
    130:Z1 =  VAL (X$)
280  REM  *** 4.STRING MIT ALTEM STRING ANGEZEIGT
290  NORMAL : VTAB 13: HTAB 1: PRINT „......ULRICH STIEHL WIEDER
    DA......": INVERSE
300  POKE ZF,0: POKE CF,1: POKE VT,13: POKE HT,1: POKE LE,35: CALL IN:
    GOSUB 130:S4$ = X$
310  REM *** WORT/WORT/ZAHL IN   EINER ZEILE MIT ALTER ZAHL  ANGEZEIGT
320  VTAB 16: HTAB 1: PRINT „VORNAME        ";: NORMAL : PRINT „ ";:
    INVERSE : PRINT „ZUNAME          ";: NORMAL : PRINT „ ";: INVERSE :
    PRINT „JAHR";
330  POKE CF,0: POKE VT,17: POKE HT,1: POKE LE,15: CALL IN: GOSUB
    130:S$(1) = X$
340  POKE HT,17: CALL IN: GOSUB 130:S$(2) = X$
350  NORMAL : VTAB 17: HTAB 33: PRINT „1982";: POKE HT,33: POKE LE,4:
    POKE ZF,1: POKE CF,1: CALL IN: GOSUB 130:Z2 =  VAL (X$)
360  REM **** OUTPUT
370  VTAB 4: HTAB 11: PRINT S1$
380  VTAB 8: HTAB 1: PRINT S2$
390  VTAB 11: HTAB 18: PRINT Z1
400  VTAB 14: HTAB 1: PRINT S4$
410  VTAB 18: HTAB 1: PRINT S$(1);: HTAB 17: PRINT S$(2);: HTAB 33:
    PRINT Z2;
420  REM **** ENDE
430  VTAB 20: HTAB 13: PRINT „ERNEUT J/N ";
440  GET G$: IF G$ = „J" THEN 180
450  IF G$ <  > „N" THEN 440
460  END
```

```
                        1               ORG  $4000
                        2   *-------------------------------
                        3   *
                        4   * PRINT80  Apple IIe 80 Z/Z-
                        5   * ======== Screen-Print-Routine
                        6   *
                        7   * Fast doppelt so schnell wie
                        8   * wie Firmware-Routine in $C300
                        9   *
                       10   * U.Stiehl/1984
                       11   *-------------------------------
                       12   WLFT      EQU  $20
                       13   WWDTH     EQU  $21
                       14   WTOP      EQU  $22
                       15   WBTM      EQU  $23
                       16   CH        EQU  $24
                       17   CV        EQU  $25
                       18   BASL      EQU  $28
                       19   BASH      EQU  $29
                       20   BAS2L     EQU  $2A
                       21   BAS2H     EQU  $2B
                       22   YSAVE     EQU  $FF
                       23   PAGE1     EQU  $C054       ;MAIN
                       24   PAGE2     EQU  $C055       ;AUX
                       25   CONNECT   EQU  $3EA
                       26   *--------IN/OUT-HOOK----------------
4000: 4C 0A 40         27   HOOKER    JMP  HOOK
4003: 4C 2E 40         28   UNHOOKER  JMP  UNHOOK
4006: 00               29   SAVE36    HEX  00
4007: 00               30   SAVE37    HEX  00
4008: 00               31   AREG      HEX  00
4009: 00               32   YREG      HEX  00
                       33   *--------HOOK------------------
400A: A5 36            34   HOOK      LDA  $36
400C: 8D 06 40         35             STA  SAVE36
400F: A5 37            36             LDA  $37
4011: 8D 07 40         37             STA  SAVE37
4014: A9 46            38             LDA  #<PRINT80
4016: 85 36            39             STA  $36
4018: A9 40            40             LDA  #>PRINT80
401A: 85 37            41             STA  $37
                       42   *--------HOME------------------
401C: 8D 08 40         43   HOME      STA  AREG
401F: 8C 09 40         44             STY  YREG
4022: A9 00            45             LDA  #0
4024: 85 24            46             STA  CH
4026: 85 25            47             STA  CV
4028: 20 F0 40         48             JSR  BASCALC
402B: 4C 66 40         49             JMP  RESTORE
                       50   *--------UNHOOK----------------
402E: AD 06 40         51   UNHOOK    LDA  SAVE36
4031: 85 36            52             STA  $36
4033: AD 07 40         53             LDA  SAVE37
```

```
4036: 85 37      54            STA  $37
4038: A4 24      55            LDY  CH
403A: 8C 7B 05   56            STY  $57B        ;OUTCH
403D: 8C 7B 04   57            STY  $47B        ;OLDCH
4040: A4 25      58            LDY  CV
4042: 8C FB 05   59            STY  $5FB        ;OUTCV
4045: 60         60            RTS
                 61    *--------PRINT--------------------
4046: 8D 08 40   62    PRINT80  STA  AREG
4049: 8C 09 40   63            STY  YREG
404C: C9 8D      64            CMP  #$8D
404E: F0 20      65            BEQ  RETURN
                 66    *--------STORE A IN PAGE 1/2------
4050: A5 24      67    STA80    LDA  CH
4052: 4A         68            LSR
4053: 90 05      69            BCC  STAPG2
4055: 8C 54 C0   70    STAPG1   STY  PAGE1
4058: B0 03      71            BCS  STA80A
405A: 8C 55 C0   72    STAPG2   STY  PAGE2
405D: A8         73    STA80A   TAY
405E: AD 08 40   74            LDA  AREG
4061: 91 28      75            STA  (BASL),Y
                 76    *--------ADVANCE CURSOR-----------
4063: 20 7B 40   77            JSR  NEXTCOL
                 78    *--------RESTORE REGISTERS--------
4066: AD 08 40   79    RESTORE  LDA  AREG
4069: 8C 54 C0   80            STY  PAGE1
406C: AC 09 40   81            LDY  YREG
406F: 60         82            RTS
                 83    *--------CARRIAGE RETURN----------
4070: A4 24      84    RETURN   LDY  CH
4072: 20 D3 40   85            JSR  CLEOL1
4075: 20 84 40   86            JSR  NEXTLINE
4078: 4C 66 40   87            JMP  RESTORE
                 88    *--------ADVANCE 1 COLUMN---------
407B: E6 24      89    NEXTCOL  INC  CH
407D: A5 24      90             LDA  CH
407F: C5 21      91             CMP  WWDTH
4081: B0 01      92             BCS  NEXTLINE
4083: 60         93             RTS
                 94    *--------ADVANCE 1 LINE----------
4084: A9 00      95    NEXTLINE LDA  #$00
4086: 85 24      96             STA  CH
4088: E6 25      97             INC  CV
408A: A5 25      98             LDA  CV
408C: C5 23      99             CMP  WBTM
408E: 90 60     100             BCC  BASCALC
4090: C6 25     101             DEC  CV
                102    *--------SCROLL UP---------------
4092: A5 22     103             LDA  WTOP
4094: 48        104             PHA
4095: 20 F0 40  105             JSR  BASCALC
4098: A5 28     106    SCRL1    LDA  BASL
409A: 85 2A     107             STA  BAS2L
```

```
409C: A5 29      108          LDA  BASH
409E: 85 2B      109          STA  BAS2H
40A0: 68         110          PLA
40A1: 69 01      111          ADC  #$01
40A3: C5 23      112          CMP  WBTM
40A5: B0 24      113          BCS  SCRL3
40A7: 48         114          PHA
40A8: 20 F0 40   115          JSR  BASCALC
                 116 *--------COPY LINE+1 TO LINE-----
40AB: A5 21      117 SCRL2    LDA  WWDTH
40AD: 4A         118          LSR
40AE: A8         119          TAY
40AF: 88         120          DEY
40B0: 84 FF      121          STY  YSAVE
40B2: 8C 54 C0   122          STY  PAGE1
40B5: B1 28      123 SCRL2A   LDA  (BASL),Y   ;LINE+1
40B7: 91 2A      124          STA  (BAS2L),Y  ;LINE
40B9: 88         125          DEY
40BA: 10 F9      126          BPL  SCRL2A
40BC: A4 FF      127          LDY  YSAVE
40BE: 8C 55 C0   128          STY  PAGE2
40C1: B1 28      129 SCRL2B   LDA  (BASL),Y   ;LINE+1
40C3: 91 2A      130          STA  (BAS2L),Y  ;LINE
40C5: 88         131          DEY
40C6: 10 F9      132          BPL  SCRL2B
40C8: 4C 98 40   133          JMP  SCRL1
                 134 *--------CLEAR BOTTOM LINE-------
40CB: A0 00      135 SCRL3    LDY  #$00
40CD: 20 D3 40   136          JSR  CLEOL1
40D0: 4C EE 40   137          JMP  VTAB1
                 138 *--------CLEAR END OF LINE-------
40D3: 84 FF      139 CLEOL1   STY  YSAVE
40D5: 98         140          TYA
40D6: 4A         141          LSR
40D7: 90 05      142          BCC  CLEOLPG2
40D9: 8C 54 C0   143 CLEOLPG1 STY  PAGE1
40DC: B0 03      144          BCS  CLEOL2
40DE: 8C 55 C0   145 CLEOLPG2 STY  PAGE2
40E1: A8         146 CLEOL2   TAY
40E2: A9 A0      147          LDA  #$A0       ;SPACE
40E4: 91 28      148          STA  (BASL),Y
40E6: A4 FF      149          LDY  YSAVE
40E8: C8         150          INY
40E9: C4 21      151          CPY  WWDTH
40EB: 90 E6      152          BCC  CLEOL1
40ED: 60         153          RTS
                 154 *--------VTAB--------------------
40EE: A5 25      155 VTAB1    LDA  CV
                 156 *--------BASCALC------------------
40F0: A8         157 BASCALC  TAY
40F1: B9 17 41   158          LDA  BASHADR,Y
40F4: 85 29      159          STA  BASH
40F6: 18         160          CLC
40F7: B9 FF 40   161          LDA  BASLADR,Y
```

```
40FA: 65 20       162             ADC  WLFT
40FC: 85 28       163             STA  BASL
40FE: 60          164             RTS
                  165  *---------SCREEN BASE LINES-------
40FF: 00 80 00    166  BASLADR    HEX  0080008000800080
4102: 80 00 80 00 80
4107: 28 A8 28    167             HEX  28A828A828A828A8
410A: A8 28 A8 28 A8
410F: 50 D0 50    168             HEX  50D050D050D050D0
4112: D0 50 D0 50 D0
4117: 04 04 05    169  BASHADR    HEX  0404050506060707
411A: 05 06 06 07 07
411F: 04 04 05    170             HEX  0404050506060707
4122: 05 06 06 07 07
4127: 04 04 05    171             HEX  0404050506060707
412A: 05 06 06 07 07

--End assembly-

303 bytes

Errors: 0
```

```
100  PRINT  CHR$ (4)"BLOAD PRINT80": PRINT  CHR$ (4)"PR#3": PRINT
110  CALL 16384
120  FOR X = 1 TO 23: PRINT
     "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
     AAAAAAAAAAAAAAAAAAA": NEXT
130  FOR X = 1 TO 10: LIST : NEXT
140  CALL 16387
```

```
                        1               ORG   $6000
                        3   *
                        4   * DOUBLEPLOT
                        5   * ==========
                        6   *
                        7   * 192 * 560 Bit-Punkte HGR-Grafik
                        8   * für Apple IIe mit 64K-Karte
                        9   * von U.Stiehl/Juni 1984
                       10   *
                       11   IND1      EQU   $CE
                       12   IND2      EQU   $50
                       13   GETNUM    EQU   $E746
                       14   *
6000: 4C 0C 60         15   H6000     JMP   HOOK         ;24576
6003: 4C 2E 60         16   H6003     JMP   UNHOOK       ;24579
6006: 4C 3E 60         17   H6006     JMP   CLEAR        ;24582
                       18   *
6009: 00               19   YACHSE    HEX   00
600A: 00               20   XACHSEL   HEX   00
600B: 00               21   XACHSEH   HEX   00
                       22   *
600C: A9 4C            23   HOOK      LDA   #$4C
600E: 8D F5 03         24             STA   $3F5
6011: A9 62            25             LDA   #<AMPER
6013: 8D F6 03         26             STA   $3F6
6016: A9 60            27             LDA   #>AMPER
6018: 8D F7 03         28             STA   $3F7
                       29   *
601B: AD 52 C0         30             LDA   $C052        ;NO-MIX
601E: AD 57 C0         31             LDA   $C057        ;HI-RES
6021: AD 50 C0         32             LDA   $C050        ;GRAPHIC
6024: 8D 01 C0         33             STA   $C001        ;80STORE
6027: 8D 0D C0         34             STA   $C00D        ;80COL
602A: AD 5E C0         35             LDA   $C05E        ;AN3
602D: 60               36             RTS
                       37   *
602E: AD 56 C0         38   UNHOOK    LDA   $C056        ;LO-RES
6031: AD 51 C0         39             LDA   $C051        ;TEXT
6034: AD 54 C0         40             LDA   $C054        ;PAGE1
6037: 8D 00 C0         41             STA   $C000        ;80OFF
603A: 8D 0C C0         42             STA   $C00C        ;40COL
603D: 60               43             RTS
                       44   *
                       45   * CLEAR $2000-$3FFF: entspricht
                       46   * dem HGR-Befehl in Applesoft
                       47   *
603E: A9 00            48   CLEAR     LDA   #$00
6040: 85 CE            49             STA   IND1
6042: A9 20            50             LDA   #$20
6044: 85 CF            51             STA   IND1+1
6046: A0 00            52             LDY   #$00
6048: AD 55 C0         53   CLEAR1    LDA   $C055
604B: A9 00            54             LDA   #0
```

```
604D: 91 CE     55            STA  (IND1),Y
604F: AD 54 C0  56            LDA  $C054
6052: A9 00     57            LDA  #0
6054: 91 CE     58            STA  (IND1),Y
6056: C8        59            INY
6057: D0 EF     60            BNE  CLEAR1
6059: E6 CF     61            INC  IND1+1
605B: A5 CF     62            LDA  IND1+1
605D: C9 40     63            CMP  #$40
605F: 90 E7     64            BCC  CLEAR1
6061: 60        65            RTS
                66  *
                67  * GETNUM: 16-BIT IN $50-$51
                68  *          8-BIT IN X-REG
                69  *
                70  * & X-ACHSE, Y-ACHSE
                71  *
6062: 20 46 E7  72  AMPER     JSR  GETNUM
6065: 8E 09 60  73            STX  YACHSE
6068: A5 50     74            LDA  IND2
606A: 8D 0A 60  75            STA  XACHSEL
606D: A5 51     76            LDA  IND2+1
606F: 8D 0B 60  77            STA  XACHSEH
                78  *
6072: AC 09 60  79            LDY  YACHSE
6075: B9 AE 60  80            LDA  HGRBASL,Y
6078: 85 CE     81            STA  IND1
607A: B9 6E 61  82            LDA  HGRBASH,Y
607D: 85 CF     83            STA  IND1+1
                84  *
607F: AD 0A 60  85            LDA  XACHSEL
6082: 8D 40 62  86            STA  DIVDENDL
6085: AD 0B 60  87            LDA  XACHSEH
6088: 8D 41 62  88            STA  DIVDENDH
608B: 20 3D 62  89            JSR  DIV0
                90  *
608E: AE 46 62  91            LDX  DIVRESTL
6091: BD 2E 62  92            LDA  BITTABLE,X
6094: 8D 3C 62  93            STA  BITBYTE
                94  *
6097: AD 44 62  95            LDA  DIVQUOTL
609A: 4A        96            LSR
609B: 90 05     97            BCC  STAPG2
609D: AC 54 C0  98  STAPG1    LDY  $C054      ;PAGE1
60A0: B0 03     99            BCS  PAGER
60A2: AC 55 C0  100 STAPG2    LDY  $C055      ;PAGE2
                101 *
60A5: A8        102 PAGER     TAY
60A6: B1 CE     103           LDA  (IND1),Y
60A8: 0D 3C 62  104           ORA  BITBYTE
60AB: 91 CE     105           STA  (IND1),Y
60AD: 60        106           RTS
                107 *
```

```
                        108  * 192 ZEILEN 0-191: Y-ACHSE
                        109  * ------------------------
                        110  *
60AE: 00 00 00          111  HGRBASL  HEX  0000000000000000
60B6: 80 80 80          112           HEX  8080808080808080
60BE: 00 00 00          113           HEX  0000000000000000
60C6: 80 80 80          114           HEX  8080808080808080
60CE: 00 00 00          115           HEX  0000000000000000
60D6: 80 80 80          116           HEX  8080808080808080
60DE: 00 00 00          117           HEX  0000000000000000
60E6: 80 80 80          118           HEX  8080808080808080
                        119  *
60EE: 28 28 28          120           HEX  2828282828282828
60F6: A8 A8 A8          121           HEX  A8A8A8A8A8A8A8A8
60FE: 28 28 28          122           HEX  2828282828282828
6106: A8 A8 A8          123           HEX  A8A8A8A8A8A8A8A8
610E: 28 28 28          124           HEX  2828282828282828
6116: A8 A8 A8          125           HEX  A8A8A8A8A8A8A8A8
611E: 28 28 28          126           HEX  2828282828282828
6126: A8 A8 A8          127           HEX  A8A8A8A8A8A8A8A8
                        128  *
612E: 50 50 50          129           HEX  5050505050505050
6136: D0 D0 D0          130           HEX  D0D0D0D0D0D0D0D0
613E: 50 50 50          131           HEX  5050505050505050
6146: D0 D0 D0          132           HEX  D0D0D0D0D0D0D0D0
614E: 50 50 50          133           HEX  5050505050505050
6156: D0 D0 D0          134           HEX  D0D0D0D0D0D0D0D0
615E: 50 50 50          135           HEX  5050505050505050
6166: D0 D0 D0          136           HEX  D0D0D0D0D0D0D0D0
                        137  *
616E: 20 24 28          138  HGRBASH  HEX  2024282C3034383C
6176: 20 24 28          139           HEX  2024282C3034383C
617E: 21 25 29          140           HEX  2125292D3135393D
6186: 21 25 29          141           HEX  2125292D3135393D
618E: 22 26 2A          142           HEX  22262A2E32363A3E
6196: 22 26 2A          143           HEX  22262A2E32363A3E
619E: 23 27 2B          144           HEX  23272B2F33373B3F
61A6: 23 27 2B          145           HEX  23272B2F33373B3F
                        146  *
61AE: 20 24 28          147           HEX  2024282C3034383C
61B6: 20 24 28          148           HEX  2024282C3034383C
61BE: 21 25 29          149           HEX  2125292D3135393D
61C6: 21 25 29          150           HEX  2125292D3135393D
61CE: 22 26 2A          151           HEX  22262A2E32363A3E
61D6: 22 26 2A          152           HEX  22262A2E32363A3E
61DE: 23 27 2B          153           HEX  23272B2F33373B3F
61E6: 23 27 2B          154           HEX  23272B2F33373B3F
                        155  *
61EE: 20 24 28          156           HEX  2024282C3034383C
61F6: 20 24 28          157           HEX  2024282C3034383C
61FE: 21 25 29          158           HEX  2125292D3135393D
6206: 21 25 29          159           HEX  2125292D3135393D
620E: 22 26 2A          160           HEX  22262A2E32363A3E
6216: 22 26 2A          161           HEX  22262A2E32363A3E
```

```
621E: 23 27 2B  162          HEX  23272B2F33373B3F
6226: 23 27 2B  163          HEX  23272B2F33373B3F
                164  *
                165  * Bit-Tabelle
                166  * ------------
                167  *
622E: 01        168  BITTABLE DFB  %00000001  ;0
622F: 02        169          DFB  %00000010  ;1
6230: 04        170          DFB  %00000100  ;2
6231: 08        171          DFB  %00001000  ;3
6232: 10        172          DFB  %00010000  ;4
6233: 20        173          DFB  %00100000  ;5
6234: 40        174          DFB  %01000000  ;6
                175  *
6235: 01        176          DFB  %00000001  ;0
6236: 02        177          DFB  %00000010  ;1
6237: 04        178          DFB  %00000100  ;2
6238: 08        179          DFB  %00001000  ;3
6239: 10        180          DFB  %00010000  ;4
623A: 20        181          DFB  %00100000  ;5
623B: 40        182          DFB  %01000000  ;6
                183  *
623C: 00        184  BITBYTE  HEX  00
                185  *
                186  *
                187  * Division 16 Bit : 16 Bit
                188  * ------------------------
                189  *
623D: 4C 4B 62  190  DIV0     JMP  DIV1
                191  *
                192  * DIVIDEND + DIVISOR
                193  *
6240: 00        194  DIVDENDL HEX  00
6241: 00        195  DIVDENDH HEX  00
6242: 07        196  DIVISORL HEX  07
6243: 00        197  DIVISORH HEX  00
                198  *
                199  * QUOTIENT + DIVISIONSREST
                200  *
6244: 00        201  DIVQUOTL HEX  00
6245: 00        202  DIVQUOTH HEX  00
6246: 00        203  DIVRESTL HEX  00
6247: 00        204  DIVRESTH HEX  00
                205  *
6248: 00        206  AREG     HEX  00
6249: 00        207  XREG     HEX  00
624A: 00        208  YREG     HEX  00
                209  *
624B: 8D 48 62  210  DIV1     STA  AREG
624E: 8E 49 62  211          STX  XREG
6251: 8C 4A 62  212          STY  YREG
                213  *
```

```
6254: A9 00     214           LDA  #0
6256: 8D 46 62  215           STA  DIVRESTL
6259: 8D 47 62  216           STA  DIVRESTH
                217  *
625C: AD 42 62  218           LDA  DIVISORL
625F: D0 0D     219           BNE  DIV2
6261: AD 43 62  220           LDA  DIVISORH
6264: D0 08     221           BNE  DIV2
                222  *
                223  * DIVISION DURCH NULL VERBOTEN
                224  *
6266: 8D 44 62  225           STA  DIVQUOTL
6269: 8D 45 62  226           STA  DIVQUOTH
626C: F0 36     227           BEQ  DIV5
                228  *
626E: AD 40 62  229  DIV2      LDA  DIVDENDL
6271: 8D 44 62  230           STA  DIVQUOTL
6274: AD 41 62  231           LDA  DIVDENDH
6277: 8D 45 62  232           STA  DIVQUOTH
                233  *
627A: A0 10     234           LDY  #$10       ;16BITS
627C: 0E 44 62  235  DIV3      ASL  DIVQUOTL
627F: 2E 45 62  236           ROL  DIVQUOTH
6282: 2E 46 62  237           ROL  DIVRESTL
6285: 2E 47 62  238           ROL  DIVRESTH
6288: 38        239           SEC
6289: AD 46 62  240           LDA  DIVRESTL
628C: ED 42 62  241           SBC  DIVISORL
628F: AA        242           TAX
6290: AD 47 62  243           LDA  DIVRESTH
6293: ED 43 62  244           SBC  DIVISORH
6296: 90 09     245           BCC  DIV4
6298: 8E 46 62  246           STX  DIVRESTL
629B: 8D 47 62  247           STA  DIVRESTH
629E: EE 44 62  248           INC  DIVQUOTL
62A1: 88        249  DIV4     DEY
62A2: D0 D8     250           BNE  DIV3
                251  *
62A4: AD 48 62  252  DIV5      LDA  AREG
62A7: AE 49 62  253           LDX  XREG
62AA: AC 4A 62  254           LDY  YREG
62AD: 60        255           RTS
```

```
1  PRINT  CHR$ (21): PRINT  CHR$ (4)„BLOAD DOUBLEPLOT"
2  CALL 24576: CALL 24582: REM HOOK+CLEAR
3  XE = 559:YE = 191:A = XE / 2:B = YE / 2:R = 95:F1 = 45:
   F2 = 45: REM 2 PARABELN
4  FOR X = A - R TO A + R: IF X < 0 OR X > XE GOTO 9
5 H = R * R - (X - A) * (X - A): IF H <  = 0 GOTO 9
6 Y1 = B +  SQR (H) - F1:Y2 = B -  SQR (H) + F1
7  IF Y1 > 0 AND Y1 <  = YE THEN  & X,Y1
8  IF Y2 > 0 AND Y2 <  = YE THEN  & X,Y2
9  NEXT X: GET X$: CALL 24579: REM UNHOOK
```

```
1               ORG  $300
2     *
3     * LORES.NORMAL
4     * ============
5     *
6     *-----------------------------
7     * Waagrechte X-Achse
8     * 0 (links) bis 39 (rechts)
9     *
10    * Senkrechte Y-Achse:
11    * 0 (oben) bis 39 (unten)   MIX
12    * 0 (oben) bis 47 (unten)  FULL
13    *-----------------------------
14    * H2=X2; V2=Y2 Linien-Endpunkte
15    *
16    H2        EQU  $2C         ;X2
17    V2        EQU  $2D         ;Y2
18    *-----------------------------
19    * Mixed-Lores- und Text-Modus
20    *
21    SETGR     EQU  $FB40
22    *
23    SETTXT    EQU  $FB39
24    *-----------------------------
25    * SETCOL: Farbe festlegen
26    * Color-Nr. (0-15) in A-Register
27    *
28    SETCOL    EQU  $F864
29    *
30    * SCRN: Farbe eines Punktes ermitteln
31    * X-Abschnitt in Y-Register
32    * Y-Abschnitt in A-Register
33    * Danach Farbe in A-Register
34    *
35    SCRN      EQU  $F817
36    *-----------------------------
37    * CLRSCR: Zeilen 0-39 löschen
38    * CLRTOP: Zeilen 0-47 löschen
39    *
40    CLRSCR    EQU  $F832       ;0-39
41    CLRTOP    EQU  $F836       ;0-47
42    *-----------------------------
43    * PLOT: Punkt plotten
44    * X-Abschnitt in Y-Register
45    * Y-Abschnitt in A-Register
46    *
47    PLOT      EQU  $F800
48    *-----------------------------
49    * HLINE: Horizontale Linie plotten
50    * X1-links    in Y-Register
51    * X2-rechts   in H2
52    * Y-Abschnitt in A-Register
53    *
```

```
                        54  HLINE    EQU  $F819
                        55  *-------------------------------
                        56  * VLINE: Vertikale Linie plotten
                        57  * X-Abschnitt in Y-Register
                        58  * Y1-oben     in A-Register
                        59  * Y2-unten    in V2
                        60  *
                        61  VLINE    EQU  $F828
                        62  *
                        63  *-------------------------------
                        64  *
                        65  * Demo
                        66  * ====
                        67  *
0300: 20 40 FB          68          JSR  SETGR
0303: A9 0F             69          LDA  #15
0305: 20 64 F8          70          JSR  SETCOL
                        71  *
                        72  * Punkt in der Mitte der
                        73  * untersten Zeile plotten
                        74  *
0308: A0 14             75          LDY  #20        ;X
030A: A9 27             76          LDA  #39        ;Y
030C: 20 00 F8          77          JSR  PLOT
                        78  *
                        79  * Waagrechte Linie am oberen
                        80  * Lores-Bildschirmrand
                        81  *
030F: A0 00             82          LDY  #0         ;X1
0311: A9 27             83          LDA  #39        ;X2
0313: 85 2C             84          STA  H2
0315: A9 00             85          LDA  #0         ;Y
0317: 20 19 F8          86          JSR  HLINE
                        87
                        88  * Senkrechte Linie am rechten
                        89  * Lores-Bildschirmrand
                        90  *
031A: A0 27             91          LDY  #39        ;X
031C: A9 27             92          LDA  #39        ;Y2
031E: 85 2D             93          STA  V2
0320: A9 00             94          LDA  #0         ;Y1
0322: 20 28 F8          95          JSR  VLINE
                        96  *
                        97  * Senkrechte Linie am linken
                        98  * Lores-Bildschirmrand
                        99  *
0325: A0 00            100          LDY  #0         ;X
0327: A9 27            101          LDA  #39        ;Y2
0329: 85 2D            102          STA  V2
032B: A9 00            103          LDA  #0         ;Y1
032D: 20 28 F8         104          JSR  VLINE
0330: 60               105          RTS
```

```
1             ORG  $300
2     *
3     * DOUBLE.LORES
4     * ============
5     * von U.Stiehl 1984
6     *---------------------------
7     * Waagrechte X-Achse
8     * 0 (links) bis 39 (rechts)
9     *
10    * Senkrechte Y-Achse:
11    * 0 (oben) bis 39 (unten)   MIX
12    * 0 (oben) bis 47 (unten)  FULL
13    *---------------------------
14    * H2=X2; V2=Y2 Linien-Endpunkte
15    *
16    H2        EQU  $2C        ;X2
17    V2        EQU  $2D        ;Y2
18    *---------------------------
19    * Mixed-Lores- und Text-Modus
20    *
21    SETGR     EQU  $FB40      ;GR-mixed
22    *
23    INIT      EQU  $FB39      ;Textmodus
24    *
25    PAGE1     EQU  $C054
26    PAGE2     EQU  $C055
27    *---------------------------
28    * SETCOL: Farbe festlegen
29    * Color-Nr. (0-15) in A-Register
30    *
31    SETCOL    EQU  $F864
32    *
33    * SCRN: Farbe eines Punktes ermitteln
34    * X-Abschnitt in Y-Register
35    * Y-Abschnitt in A-Register
36    * Danach Farbe in A-Register
37    *
38    SCRN      EQU  $F817
39    *---------------------------
40    * CLRSCR: Zeilen 0-47 löschen
41    * CLRTOP: Zeilen 0-39 löschen
42    *
43    CLRSCR    EQU  $F832      ;0-47
44    CLRTOP    EQU  $F836      ;0-39
45    *---------------------------
46    * PLOT: Punkt plotten
47    * X-Abschnitt in Y-Register
48    * Y-Abschnitt in A-Register
49    *
50    PLOT      EQU  $F800
51    *---------------------------
52    * HLINE: Horizontale Linie plotten
53    * X1-links    in Y-Register
```

```
                        54   * X2-rechts   in H2
                        55   * Y-Abschnitt in A-Register
                        56   *
                        57   HLINE    EQU  $F819
                        58   *-----------------------------
                        59   * VLINE: Vertikale Linie plotten
                        60   * X-Abschnitt in Y-Register
                        61   * Y1-oben     in A-Register
                        62   * Y2-unten    in V2
                        63   *
                        64   VLINE    EQU  $F828
                        65   *
                        66   *-----------------------------
                        67   *
                        68   * Demo
                        69   * ====
                        70   *
0300: AD 50 C0          71            LDA  $C050      ;GRAFIK
0303: AD 56 C0          72            LDA  $C056      ;LORES
0306: AD 52 C0          73            LDA  $C052      ;NOMIX
0309: 8D 01 C0          74            STA  $C001      ;80STORE
030C: 8D 0D C0          75            STA  $C00D      ;80COL
030F: AD 5E C0          76            LDA  $C05E      ;80AN3
                        77   *
0312: AD 54 C0          78            LDA  PAGE1
0315: 20 32 F8          79            JSR  CLRSCR
0318: AD 55 C0          80            LDA  PAGE2
031B: 20 32 F8          81            JSR  CLRSCR
                        82   *
031E: A9 0A             83            LDA  #10
0320: 20 64 F8          84            JSR  SETCOL
                        85   *
                        86   * Punkt in der Mitte der
                        87   * untersten Zeile plotten
                        88   *
0323: AD 55 C0          89            LDA  PAGE2
0326: A0 13             90            LDY  #19        ;X
0328: A9 2F             91            LDA  #47        ;Y
032A: 20 00 F8          92            JSR  PLOT
                        93   *
032D: AD 54 C0          94            LDA  PAGE1
0330: A0 14             95            LDY  #20        ;X
0332: A9 2F             96            LDA  #47        ;Y
0334: 20 00 F8          97            JSR  PLOT
                        98   *
                        99   * Waagrechte Linie am oberen
                        100  * Lores-Bildschirmrand
                        101  *
0337: AD 55 C0          102           LDA  PAGE2
033A: A0 00             103           LDY  #0         ;X1
033C: A9 27             104           LDA  #39        ;X2
033E: 85 2C             105           STA  H2
0340: A9 00             106           LDA  #0         ;Y
```

```
0342: 20 19 F8  107          JSR  HLINE
                108  *
0345: AD 54 C0  109          LDA  PAGE1
0348: A0 00     110          LDY  #0          ;X1
034A: A9 27     111          LDA  #39         ;X2
034C: 85 2C     112          STA  H2
034E: A9 01     113          LDA  #1          ;Y
0350: 20 19 F8  114          JSR  HLINE
                115
                116  * Senkrechte Linie am rechten
                117  * Lores-Bildschirmrand
                118  *
0353: AD 55 C0  119          LDA  PAGE2
0356: A0 26     120          LDY  #38         ;X
0358: A9 2F     121          LDA  #47         ;Y2
035A: 85 2D     122          STA  V2
035C: A9 02     123          LDA  #2          ;Y1
035E: 20 28 F8  124          JSR  VLINE
                125  *
0361: AD 54 C0  126          LDA  PAGE1
0364: A0 27     127          LDY  #39         ;X
0366: A9 2F     128          LDA  #47         ;Y2
0368: 85 2D     129          STA  V2
036A: A9 02     130          LDA  #2          ;Y1
036C: 20 28 F8  131          JSR  VLINE
                132  *
                133  * Senkrechte Linie am linken
                134  * Lores-Bildschirmrand
                135  *
036F: AD 55 C0  136          LDA  PAGE2
0372: A0 00     137          LDY  #0          ;X
0374: A9 2F     138          LDA  #47         ;Y2
0376: 85 2D     139          STA  V2
0378: A9 02     140          LDA  #2          ;Y1
037A: 20 28 F8  141          JSR  VLINE
                142  *
037D: AD 54 C0  143          LDA  PAGE1
0380: A0 01     144          LDY  #1          ;X
0382: A9 2F     145          LDA  #47         ;Y2
0384: 85 2D     146          STA  V2
0386: A9 02     147          LDA  #2          ;Y1
0388: 20 28 F8  148          JSR  VLINE
                149  *
038B: 2C 10 C0  150  KEY1    BIT  $C010       ;STROBE
038E: AD 00 C0  151  KEY2    LDA  $C000       ;KEYBD
0391: 10 FB     152          BPL  KEY2
0393: 2C 10 C0  153          BIT  $C010       ;STROBE
0396: AD 54 C0  154          LDA  PAGE1
0399: 8D 00 C0  155          STA  $C000       ;40STORE
039C: 8D 0C C0  156          STA  $C00C       ;40COL
039F: AD 5F C0  157          LDA  $C05F       ;40AN3
03A2: 4C 39 FB  158          JMP  INIT
```

```
10.  PRINT „DOUBLE.LORES BY U.STIEHL 84"
13 P =  PEEK (49232): REM GR
16  POKE 49153,0: REM 80STORE
19  POKE 49165,0: REM 80COL
22 P =  PEEK (49234): REM NOMIX
25 P =  PEEK (49246): REM AN3
28 P =  PEEK (49237): CALL 63538: REM CLRP2
31 P =  PEEK (49236): CALL 63538: REM CLRP1
34 P =  PEEK (49152): ON P < 128 GOTO 34:P =  PEEK (49168):
   REM  KEY
37  FOR C = 15 TO 0 STEP  - 1: COLOR= C
40  FOR X = 0 TO 36 STEP 4
43 P =  PEEK (49237): REM PAGE2
46  VLIN 0,47 AT X
49 P =  PEEK (49236): REM PAGE1
52  VLIN 0,47 AT X + 1
55  NEXT X,C
58  COLOR= 10
61 X = 0:Y = 0
64 P =  PEEK (49237): REM PAGE2
67  PLOT X,Y
70 Y = Y + 1
73 P =  PEEK (49236): REM PAGE1
76  PLOT X,Y
79 X = X + 1:Y = Y + 1
82  IF Y < 48 GOTO 64
85 P =  PEEK (49152): ON P < 128 GOTO 85:P =  PEEK (49168):
   REM KEY
88  POKE 49152,0: REM 80OFF
91  POKE 49164,0: REM 40COL
94 P =  PEEK (49236): REM PAGE1
97  TEXT
```

Register der ROM-Adressen

Apple DOS 3.3 — Tips und Tricks

von U.Stiehl
1984, 216 S., mit zahlreichen, ausführlich kommentierten Programmlistings, kart., DM 28,—
ISBN 3-7785-1010-X

Dies ist die erste deutschsprachige Darstellung des Diskettenbetriebssystems DOS 3.3 für den Apple II/II Plus/IIe, die sich sowohl an Applesoft - als auch an Assembler-Programmierer wendet. Sinngemäß ist das Buch zweigeteilt:

Der erste Teil behandelt ausführlich die dem Applesoft-Programmierer zur Verfügung stehenden DOS-Befehle, wobei die Textfiles wegen ihrer großen Bedeutung und der vergleichsweise komplizierten Handhabung besonders eingehend dargestellt werden. Viele Textfile-Tricks aus der langjährigen Programmiererfahrung des Autors werden hier zum erstenmal geschildert.

Aber auch im zweiten Teil findet der reine Applesoft-Programmierer insbesondere in dem Kapitel „Vermischte Tips, Tricks und Patches" zahlreiche Anregungen. Im übrigen ist der zweite Teil für Assembler-Programmierer gedacht. Neben einer detaillierten Beschreibung der DOS-Interna enthält dieser Teil elf vollständige RWTS-Anwendungsprogramme — z.B. CPM-Refiner, DOS-lose Datendisk, TSL-Maker, File-Reader, Pseudo-Disk-Driver und Fastbrun-Routine —, die Techniken enthüllen, die bislang noch niemals publiziert worden sind. So ist z.B. die Fastbrun-Routine 15mal schneller als das normale BRUN. Mit dem File-Reader lassen sich Textfiles beliebiger Größe mit ca. 6000 Zeichen/Sekunde einlesen. Der Pseudo-Disk-Driver ist der erste Apple IIe-64K-Karte-Disk-Emulator für DOS in der Language Card.

Dieses DOS-Buch ist deshalb der unentbehrlich Begleiter für jeden Apple-Programmierer.

DOS 3.3 für Applesoft- und Assembler-Programmierer

Begleitdiskette

Jetzt auch Begleitdiskette zu „Apple DOS 3.3" erhältlich, DM 28,—
ISBN 3-7785-1033-9

Apple ProDOS für Aufsteiger, Band 1

Mit ausführlichen Programmbeispielen

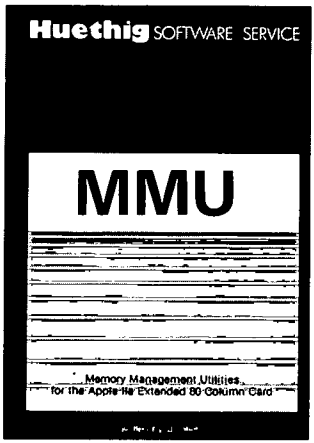
von U. Stiehl

1984, 208 S., kart., DM 28,–
ISBN 3-7785-1027-4

Begleitdiskette zu „Apple ProDOS für Aufsteiger", DM 28,–
ISBN 3-7785-1032-0

„Apple ProDOS für Aufsteiger, Band 2", ca. 180 S., kart., DM 28,–, erscheint im Oktober 1984.
ISBN 3-7785-1036-3
Begleitdiskette, DM 28,–, ISBN 3-7785-1040-1

ProDOS ist das neue Betriebssystem für den Apple II Plus/IIe. Applesoft-Programmierer, die unter DOS 3.3 gearbeitet haben, werden sich schnell an ProDOS gewöhnen, da ProDOS und DOS 3.3 in dieser Hinsicht weitgehend kompatibel sind. Dagegen müssen Assembler-Programmierer völlig umdenken. Deshalb liegt das Schwergewicht von „Apple ProDOS für Aufsteiger, Band 1" auf der Assemblerprogrammierung und der minutiösen Darstellung der ProDOS-internen Systemadressen, die jedoch auch für Applesoft-Programmierer von großer Bedeutung sind.
Zunächst wird zunächst ein allgemeiner Überblick über das neue „Professional Disk Operation System" gegeben. Im Anschluß daran folgt eine Gegenüberstellung der Geschwindigkeit des Diskettenzugriffs bei DOS und ProDOS. Dann wird die interne Speicherorganisation detailliert beschrieben (Boot-Vorgang, Zero-Page, ProDOS-Vektoren, Basic-System-Puffer, Basic-System-Global-Page, Basic-Command-Handler, I/O-Vektoren, ProDOS-Global-Page, Language-Card-Organisation, Interrupt, Disk-Driver, Reboot-Programm usw.). Ebenso ausführlich wird die externe Speicherorganisation geschildert (Spuren, Sektoren, Blocks, Directory-Struktur, Volume Bit Map, Dateistrukturen usw.). Schließlich wird das MLI (Machine Language Interface) mit zahlreichen praktischen Anwendungsbeispielen erläutert. Insgesamt enthält ProDOS-Buch ca. 70 Seiten mit eigens für dieses Werk entwickelten Programmen. Danach wird das Basic-System technisch für fortgeschrittene Applesoft-Programmierer kurz erläutert.
„ProDOS für Aufsteiger, Band 2" beschreibt ausführlich die Anwendung des ProDOS-BASIC-SYSTEM für Applesoft-Programmierer und enthält darüber hinaus zahlreiche größere Assembler-Anwenderprogramme, die aus Platzgründen in dem ersten Band nicht mehr untergebracht werden konnten.



MMU 2.0
Memory Managements Utilities

für die Apple IIe 64K-Karte

DOS 3.3 (und ProDOS)

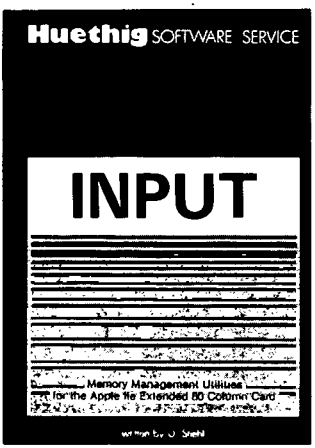
von U. Stiehl

1984, Diskette und Manual, DM 98,–
ISBN 3-7787-1023-1

Bei der Version 2.0 der MMU's sind die Utilities teilweise so umgeschrieben worden, daß sie sowohl unter DOS 3.3 als auch unter ProDOS lauffähig sind. Da dies nicht immer möglich war, sind zusätzlich zu den reinen DOS-Hilfsprogrammen, speziell den RAM-Disk-Drivern, einige reine ProDOS-Utilities aufgenommen worden. Insgesamt enthält die neue „MMU 2.0"-Diskette über 25 Programme, die neue Einsatzmöglichkeiten für die Extended 80 Column Card (erweiterte 80-Z-Karte = 64K-Karte für den Apple IIe) erschließen. Ein Teil der Programme laufen auch auf dem Apple II Plus, doch ist „MMU 2.0" primär für 64K-Karte-Besitzer gedacht.
Im einzelnen umfaßt „MMU 2.0"

- Drei RAM-Disk-Driver für DOS 3.3: „INIT62" benutzt nur die 64K-Karte als RAM-Disk, „INIT78" benutzt zusätzlich die Motherboard-LC als RAM-Karte und „DOSMOVER. INIT62" gilt für den Fall, daß sich das DOS selbst in der Motherboard-LC befindet.
- Eine sehr nützliche Pseudo-Coprocessor-Utility, die das Hin- und Herschalten zwischen zwei Programm-Modulen ermöglicht, von denen sich das eine Modul auf der 64K-Karte befindet.
- Zwei schnelle Kopierprogramme (für DOS 3.3 und ProDOS).
- Mehrere Move-Programme zum Verschieben von Daten auf die 64K-Karte sowie auf die Language Card und umgekehrt.
  Mehrere Hilfsprogramme zum Untersuchen und Löschen bestimmter Speicherbereiche der 64K-Karte und der LC, zur Ermittlung des Softswitch-Status usw.
- Zwei Simulator-Programme zum Simulieren von Apple II und Apple II Plus auf dem Apple IIe.

Schließlich enthält „MMU 2.0" auch verschiedene Move-Utilities für die RAM-Karten AP20 und AP17 der Firma IBS.



INPUT 2.0
Ein Bildschirm-Maskengenerator für DOS 3.3 und ProDOS

von U. Stiehl

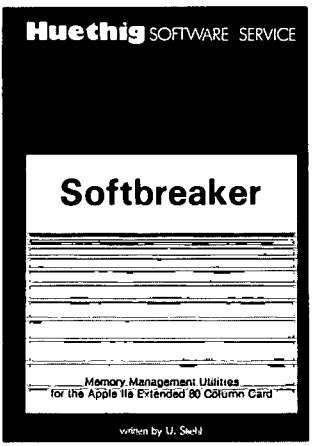
1984, Diskette und Manual, DM 98,–
ISBN 3-7785-1021-5

Der für den Apple II bestimmte Maskengenerator „Input 2.0" basiert auf den früheren Programmen „Input 1.0" und „Input 80 1.0" (von denen noch Restbestände lieferbar sind) und ist sowohl unter DOS 3.3 wie auch unter dem neuen ProDOS lauffähig. Der Maskengenerator setzt einen Apple II Plus mit Language Card oder einen Apple IIe voraus. Im 40 Z/Z-Modus funktioniert er auf beiden Gerätetypen, im 80 Z/Z-Modus dagegen nur auf dem Apple IIe mit 80-Zeichen-Karte. (Die alte Videx-Karte für den Apple II wird nicht unterstützt!)

„Input 2.0" liegt wahlweise in der Bank 1 oder Bank 2 der Language Card und wird durch einen kurzen Driver in den unteren 48K aufgerufen. „Input 2.0" läßt sich problemlos in nicht-compilierte und compilierte Applesoft- sowie in Assemblerprogramme einbinden. Die Übergabe der Feldinhalte an das Anwenderprogramm erfolgt durch ein einfaches Verfahren, das auch bei Compilern funktioniert.

Für jedes Feld der Bildschirmmaske lassen sich u. a. definieren: Feldlänge (bis zu 255 Zeichen) – Vtab – Htab – Datentyp (insgesamt 8 Typen) – Scrollflag (starre oder dynamische Maske) – Ctrlflag – Füllflag – Löschflag – Bildschirmflag (40- oder 80-Z-Darstellung). Innerhalb eines Eingabefeldes besteht jeder denkbare Redigierkomfort (Insert, Delete, Rubout, Restore usw.).

Bei der neuen Version des Maskengenerators können jetzt auch Ctrl-Zeichen beim Datentyp String eingegeben werden. Ferner sind – dies gilt nur für IIe – die Apfeltasten als schnelle Cursortasten definiert. Schließlich wurden Features implementiert, die den Einsatz von „Input 2.0" als zeilenorientiertes Textverarbeitungsprogramm ermöglichen. Die „Input 2.0"-Diskette enthält zahlreiche Demos zur Veranschaulichung der Anwendung.



Softbreaker 1.0

Eine softwaremäßige Interrupt-Utility für die Apple IIe 64K-Karte

von U. Stiehl

1984, Diskette und Manual, DM 48,–
ISBN 3-7785-1022-3

„Softbreaker" ist ein Assemblerprogramm, mit dessen Hilfe Programme, die sich von der 64K-Karte (= Extended 80 Column Card für den Apple IIe) starten lassen, unterbrochen, gespeichert, geladen und exakt an der Stelle der Unterbrechung fortgeführt werden können. Dadurch ist es auch möglich, Sicherungskopien von sogenannten kopiergeschützten Programmen herzustellen. Es sei hier ausdrücklich darauf hingewiesen, daß laut Urheberrechtsgesetz die Herstellung von Vervielfältigungsstücken nur von rechtmäßig erworbenen Werken und nur zum persönlichen Gebrauch zulässig ist.

„Softbreaker" läßt sich bei folgenden Programmtypen anwenden:

- bei grundsätzlich allen für den Apple II Plus konzipierten Programmen, die die 40-Zeichen-Darstellung und/oder die Grafikseite 1 (also nicht HGR2) benutzen, gleichviel ob sie durch Reset abgesichert sind oder nicht.
- bei grundsätzlich allen durch Reset abgesicherten Programmen, auch wenn sie die 80-Zeichen-Darstellung benutzen.

Softbreaker funktioniert selbstverständlich nicht bei Programmen, die die 64K-Karte des Apple IIe in irgendeiner Form benutzen, z. B. nicht bei Programmen, die die 64K-Karte als RAM-Disk oder als Zusatzspeicher verwenden.

Mit Softbreaker unterbrochene Programme werden komplett, d. h. die ganzen 64K einschließlich Language Card, in nur ca. 11 Sekunden auf einer formatierten Diskette gesichert, wobei die Rücksprungadresse und diverse Status-Werte automatisch mit abgespeichert werden, so daß keinerlei technischen oder Programmierkenntnisse erforderlich sind.

6502-Befehlstabelle

Function	Mnemon.	Accu.		Immedi.		Zero-P.		Zero,X		Zero,Y		Absol.		Abs,X		Abs,Y		(Ind,X)		(Ind),Y		Implied		Relative		Indirect		Status
		Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	N Z C I D V
Load	LDA			A9	2	A5	3	B5	4			AD	4	BD	4*	B9	4*	A1	6	B1	5*							N Z – – – –
	LDX			A2	2	A6	3			B6	4	AE	4			BE	4*											N Z – – – –
	LDY			A0	2	A4	3	B4	4			AC	4	BC	4*													N Z – – – –
Store	STA					85	3	95	4			8D	4	9D	5	99	5	81	6	91	6							– – – – – –
	STX					86	3			96	4	8E	4															– – – – – –
	STY					84	3	94	4			8C	4															– – – – – –
Transfer	TAX																					AA	2					N Z – – – –
	TXA																					8A	2					N Z – – – –
	TAY																					A8	2					N Z – – – –
	TYA																					98	2					N Z – – – –
Stack Ptr	TSX																					BA	2					N Z – – – –
	TXS																					9A	2					– – – – – –
Stack	PHA																					48	3					– – – – – –
	PLA																					68	4					N Z – – – –
Status R.	PHP																					08	3					– – – – – –
	PLP																					28	4					N Z C I D V
Flags	CLC																					18	2					– – 0 – – –
	SEC																					38	2					– – 1 – – –
	CLI																					58	2					– – – 0 – –
	SEI																					78	2					– – – 1 – –
	CLD																					D8	2					– – – – 0 –
	SED																					F8	2					– – – – 1 –
	CLV																					B8	2					– – – – – 0
Jump	JMP											4C	3													6C	5	– – – – – –
	JSR											20	6															– – – – – –
Return	RTS																					60	6					– – – – – –
	RTI																					40	6					N Z C I D V
Compare	CMP			C9	2	C5	3	D5	4			CD	4	DD	4*	D9	4*	C1	6	D1	5*							N Z C – – –
	CPX			E0	2	E4	3					EC	4															N Z C – – –
	CPY			C0	2	C4	3					CC	4															N Z C – – –
	BIT					24	3					2C	4															– Z – – – –
Branch	BMI																							30	2+			– – – – – –
N = 0	BPL																							10	2+			– – – – – –
	BEQ																							F0	2+			– – – – – –
Z = 0	BNE																							D0	2+			– – – – – –
	BCS																							B0	2+			– – – – – –
C = 0	BCC																							90	2+			– – – – – –
	BVS																							70	2+			– – – – – –
V = 0	BVC																							50	2+			– – – – – –
Increment	INC					E6	5	F6	6			EE	6	FE	7													N Z – – – –
	INX																					E8	2					N Z – – – –
	INY																					C8	2					N Z – – – –
Decrement	DEC					C6	5	D6	6			CE	6	DE	7													N Z – – – –
	DEX																					CA	2					N Z – – – –
	DEY																					88	2					N Z – – – –
Add/Subt.	ADC			69	2	65	3	75	4			6D	4	7D	4*	79	4*	61	6	71	5*							N Z C – – V
	SBC			E9	2	E5	3	F5	4			ED	4	FD	4*	F9	4*	E1	6	F1	5*							N Z C – – V
Boolean	AND			29	2	25	3	35	4			2D	4	3D	4*	39	4*	21	6	31	5*							N Z – – – –
	ORA			09	2	05	3	15	4			0D	4	1D	4*	19	4*	01	6	11	5*							N Z – – – –
	EOR			49	2	45	3	55	4			4D	4	5D	4*	59	4*	41	6	51	5*							N Z – – – –
Shift	ASL	0A	2			06	5	16	6			0E	6	1E	7													N Z C – – –
	LSR	4A	2			46	5	56	6			4E	6	5E	7													N Z C – – –
Rotate	ROL	2A	2			26	5	36	6			2E	6	3E	7													N Z C – – –
	ROR	6A	2			66	5	76	6			6E	6	7E	7													N Z C – – –
Miscell.	NOP																					EA	2					– – – – – –
	BRK																					00	7					– – – 1 – –
No. of Bytes		1		2		2		2		2		3		3		3		2		2		1		2		3		

* = 1 Takt mehr bei Seitenübergang;
\+ = 2 Takte ohne Verzweigung, 3 Takte bei Verzweigung, 4 Takte bei Verzweigung mit Seitenübergang.

„Apple Assembler" wendet sich an alle, die bereits Anfängerkenntnisse der 6502-Programmierung haben und nunmehr ein Nachschlagewerk für ihren Apple II Plus/IIe/IIc suchen, in dem alle wichtigen ROM-Routinen sowie eine Vielzahl sonstiger Hilfsprogramme in einer systematischen Form zusammengestellt werden. Insgesamt umfaßt dieses Buch über 40 Utilities, darunter mehrere völlig neuartige Programme wie Double-Lores, Double-Hires, Screen-Format u. a.
Der erste Teil enthält ein Repetitorium der wichtigsten Befehle, Adressierungsarten und sonstigen Besonderheiten des 6502.
Im zweiten Teil werden alle Adressen des Monitors zusammengestellt, die für Assemblerprogrammierer von Nutzen sein können. Darüber hinaus findet der Leser Unterroutinen für hexadezimale Addition/Substraktion/Multiplikation/Division, Binär-Hex-ASCII-Umwandlung usw.
Der dritte Teil befaßt sich mit der Speicherverwaltung der Language-Card und der IIe-64K-Karte und enthält Move-Programme zum Verschieben von Daten in die und aus der Language Card sowie der 64K-Karte.
Der vierte Teil ist dem Applesoft-ROM gewidmet und listet eine große Anzahl nützlicher Interpreter-Adressen. Bei den Utility-Programmen liegt das Schwergewicht auf Fließkommamathematik einschließlich Print Using.
Der letzte Teil behandelt den Text- und Grafikspeicher. Neben einem professionellen Maskengeneratorprogramm werden auch Routinen zur Double-Lores- und Double-Hires-Grafik vorgestellt.