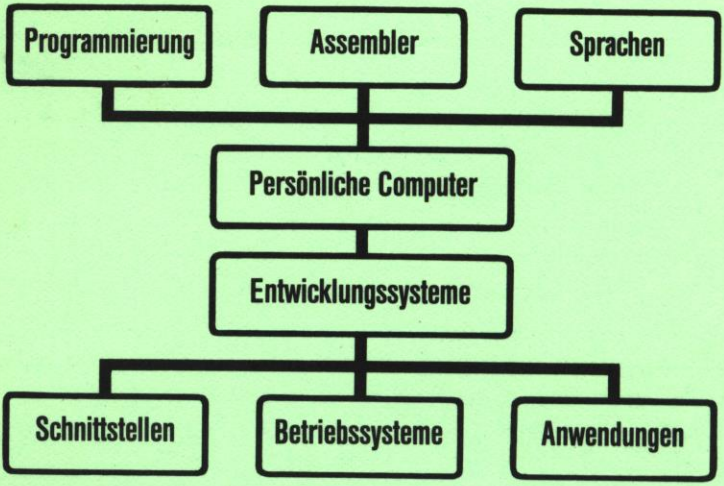
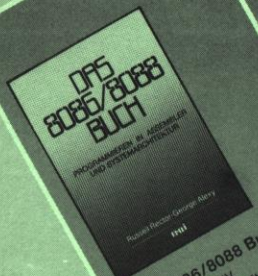
MICRO MAG

DM 9,50 Nr. 40 Dezember 1984



Inhaltsverzeichnis

AKTUELLE MIKRO-COMPUTER LITERATUR VON te-wi



Das 8086/8088 Buch
Rector/Alexy
560 Seiten, Softcover
ISBN 3-921803-11-X
DM 79,–

C 64/IEEE-488 Buch und Modul
ISBN 3-921803-29-2
DM 239,– (2. Q. 84)

M68000-Familie
Hilf/Nausch
Teil 1 – Grundlagen und Architektur
550 Seiten, Softcover
ISBN 3-921803-16-0, DM 79,–
Teil 2 – Anwendung und 68000-Bausteine (2. Q. 84)
350 Seiten, Softcover
ISBN 3-921803-30-6 DM 59,–

Die 8085/8086 Interfaces
Klaus-Dieter Thies
656 Seiten, DIN A 4,
Softcover
ISBN 3-921803-23-3
DM 89,–

UNIX Anwenderhandbuch
Thomas/Yates
630 Seiten, Softcover
Viele Abbildungen
ISBN 3-921803-17-9
DM 79,–

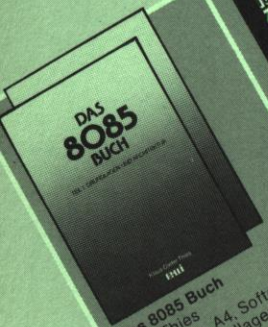
Weitere Bücher:

Einführung in die Mikrocomputer-Technik
DM 66,–

MC-Bücher:
Apple II Anwenderhandbuch DM 56,–
Apple II PASCAL DM 59,–
Apple Maschinensprache DM 49,–
CBM Computerhandbuch DM 59,–
C64 Computer Handbuch (2./3. Q. 84) DM 56,–
Mein ATARI Computer DM 59,–
IBM-PC Handbuch (2. Q. 84) DM 59,–
CP/M und WordStar DM 29,80
VisiCalc, mit Diskette DM 79,–

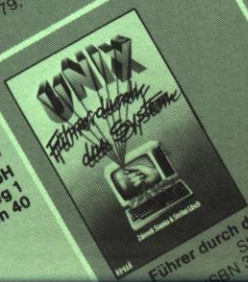
6502 Programmieren in Assembler
Lance A. Leventhal
704 Seiten, Softcover
Viele Funktionsdiagramme und Abbildungen
ISBN 3-921803-10-1
DM 59,–

Z8000 – Aufbau und Anwendungen
P. Stuhlmüller
464 Seiten, Hardcover
ISBN 3-912803-07-1
DM 79,–



Das 8085 Buch
K.-D. Thies
310 Seiten, A4, Softcover
Band 1, Grundlagen und Architektur
ISBN 3-921803-25-X, DM 49,–
Band 2, Software,
ISBN 3-921803-26-8,
DM 49,–

Die C-Sprache
Stanka/Lösch
ca. 250 Seiten, Softcover
ISBN 3-912803-28-4
DM 59,–



UNIX – Führer durch das System
Stanka/Lösch
ISBN 3-912803-27-6
ca. 250 Seiten,
Softcover
DM 59,–

te-wi
te-wi Verlag GmbH
Theo-Prosel-Weg 1
8000 München 40

Roland Löhr

Betrieb des 65802 und 65816

Die 8/16 Bit Prozessoren von GTE

Übersicht

In Heft 38 wiesen wir bereits auf die kommenden Prozessoren mit o.a. Bezeichnung hin, welche vom Western Design Center entwickelt wurden und von der Firma GTE Microcircuits hergestellt werden sollen (Niederlassung in München). Mit der Verfügbarkeit ist nach neuesten Mitteilungen (leider) erst zur Mitte 1985 zu rechnen. - Der Autor erhielt wenige Tage vor Redaktionsschluß eines der wenigen Muster des GS65SC802. Es handelt sich hier um einen Chip, der zur CPU 6502 pin-kompatibel ist (kleine Abweichungen noch in der Versorgungsspannung und in der Output Low Voltage gegenüber dem Datenblatt). Er konnte auf Anhieb auf dem AIM 65 des Autors in Betrieb genommen werden und ist seither in die Weiterentwicklung des hiesigen 65xxx-Assemblers einbezogen. Mit ihm wurden auch die nachfolgend protokollierten Programmierversuche ausgeführt, die interessante neue Möglichkeiten aufzeigen.

Es kommt dem Herausgeber darauf an, den interessierten Entwicklern und umrüstungsbereiten Betreibern eines 65xxx-Systems möglichst früh nützliche Hinweise zu geben, um Entscheidungshilfen zu geben und um die Programmierung vorzubereiten. Zur Programmierung ist insbesondere zu bemerken, daß sie für den Betreiber eines 6xxxx weiter in überschaubaren Kategorien erfolgt.

Zur Hardware

Das Registermodell der Prozessoren und das Pinout der beiden Typen ist der nebenstehenden Abbildung zu entnehmen. Während der Typ 65802 zur 6502 pinkompatibel ist, wird man bei bestehenden Systemen für den 65816 eine Neubeschaltung vornehmen müssen (ggfs. Adaptersockel) und dabei bedenken, daß der Datenbus dort gemultiplext ist. Während des Taktes Phi 1 sendet er dort auf 8 Bit Bankadressen aus. Phase 2 dient dem reinen Datenverkehr. Man wird die Banksignale also latchen müssen. Dann kann man 256 Bänke mit zusammen 16 MB adressieren, ggfs. auch mehr, wenn man die Signale VPA (Valid Program Address) und VDA (Valid Data Address) ausnutzt und den Programmbereich elektrisch vor dem/den Benutzer(n) schützt. Beim 65802 ist man ohne Tricks zunächst auf 64 KB begrenzt, hat aber gleichwohl den erheblich erweiterten Befehlssatz, die Verarbeitung in 8 oder 16 Bit und eine beschleunigte Taktrate, z.Zt. wohl 4 MHz. - Nach einer hier vorliegenden Fotokopie einer sog. Benchmark soll eine 65816 sich vorteilhaft mit einer 68000 CPU vergleichen, die beide unter 8 MHz laufen.

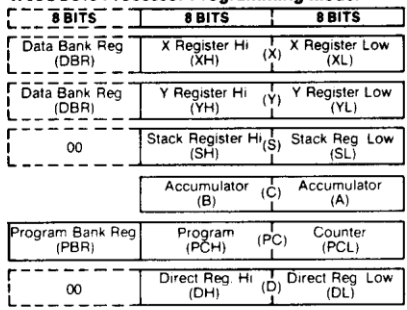
Die Prozessoren melden sich beim Power Up im sog. Emulation Mode, d.h. sie fahren den Befehlssatz der CPU 6502 in 8 Bit breiter Verarbeitung ab (E-Bit =1). Per Software wird in den 'native' Modus (natürlich) umgeschaltet (Emulation Bit E=0). Durch den Befehl XCE (Exchange Carry and Emulation Bit) wird das herbeigeführt. Man kommt also durch die Befehlsfolge CLC und XCE in den native Modus, den man mit SEC und XCE wieder verläßt.

Man beachte, daß jeder Modus einen eigenen Satz von Interrupt-/Exceptionvektoren hat, mit unterschiedlichen Adressen im High Memory. Wenn die vorhandenen F-ROMs noch nicht entsprechend eingerichtet sind, tut man gut daran, vor einer Programmerprobung, die mit dem BRK-Befehl endet, zunächst wieder auf den Modus E=1 (Emulation) umzuschalten, um keine Speicherinhalte zu verlieren.

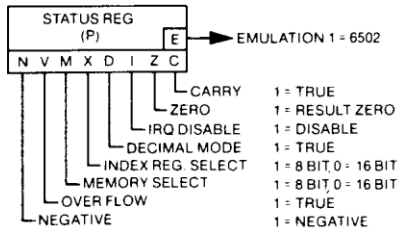
Der 8/16 Bit Modus

Das Statusregister der Prozessoren ist, wie nebenstehend abgebildet, verändert und erweitert worden. Das BRK-Bit fiel weg, dafür hat man im native Modus einen eigenen BRK Exception-Vektor. Das M-Bit steuert die Breite in der der Akku arbeitet (M=1 entspricht 8 Bit, M=0 ent-

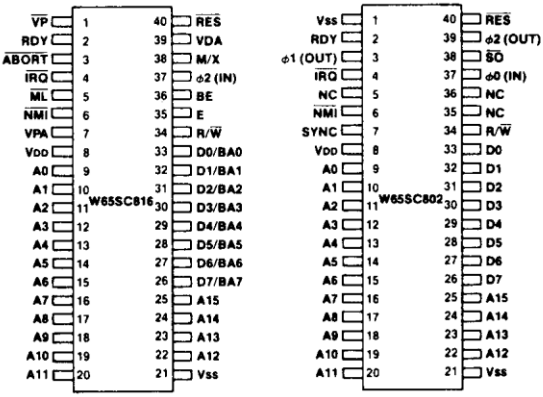
W65SC816 Processor Programming Model



Status Register Coding



Pin Configuration



spricht 16 Bit). Entsprechend bezieht sich das X-Bit auf die Breite der beiden Indexregister X und Y zugleich.

Beim Einschalten sind E=1, M=1 und X=1, wir arbeiten in normaler 8 Bit Breite. Die Betriebsweise der Register wird durch die Befehle SEP und REP gesteuert, die einen Direktoperanden mit entsprechendem Bitmuster erfordern (SEP = Set Status Bits an den mit '1' im Direktoperanden gekennzeichneten Bits, REP = Reset Status Bits dementsprechend. Hier löschen '1'-er Bits des Direktoperanden im Status).

Die Registersätze Akku und X/Y können also komplett oder auch gemischt in 8/16 Bits gefahren werden. Den Akku läßt man z.B. bei der Textverarbeitung oder beim Parsen in 8 Bit laufen, während eines der Register, vornehmlich wieder Y, über eine komplette Bank indizieren kann. Hier ist die leistungsfähige Adressierungsart (POINTER),Y im Spiel und die neue (POINTER,L),Y, wo der Pointer eine Basisadresse bildet und Y den 'Läufer'.

Der Befehl SEP #$30 setzt alle Register auf 16 Bit. Der Befehl REP #$30 holt die CPU voll auf 8 Bit zurück.

Neue Assembler-Direktiven

Die meisten Befehle werden durch den Modus im Status für M oder X in ihrer Notierung und hinsichtlich der Befehlslänge nicht beeinflußt. Hex A5 40 heißt für M=1: Hole 1 Byte aus Zelle 40. A5 40 heißt für M=0: Hole ein Wort aus den Zellen 40 (Low Byte) und 41 (High Byte). Ebenso bedeutet PHA (Code 48): Push Accu in Byte- oder Wortbreite, je nach Modus für den Akku.

Die Gruppe der Immediate-Befehle (Direktoperand) macht die Ausnahme. Für M=1 hat der Operand 1 Byte, für M=0 2 Byte. Im 16 Bit Modus heißt ein Befehl z.B. LDA #$4567, der vom Assembler generiert werden muß zu A9 67 45 (low Byte, High Byte). Wir werden das in den nachfolgenden Listings sehen.

Die betroffenen Immediate-Befehle sind für M=0 mit 3 Byte Gesamtlänge: LDA, LDX, LDY, ORA, AND, EOR, ADC, SBC, BIT, CMP, CPX und CPY.

Ein Assembler für 65802 und 65816 muß also wissen, wieviele Bytes zu generieren sind. Da eine Assembler-Syntax vom Western Design Center oder von GTE noch nicht erhältlich war, hat der Herausgeber folgende mnemonische Anweisungen geschaffen (für das Verständnis der folgenden Listings):

.A08	Akku-Betriebsweise in 8 Bit
.A16	Akku-Betriebsweise in 16 bit
.X08	Register X und Y werden in 8 Bit Breite betrieben
.X16	Dito in 16 Bit Breite
.R08	Alle Register werden in 8 Bit Breite betrieben
.R16	Alle Register werden in 16 Bit Breite betrieben.

Neue Adressierungsarten

Von den neuen Befehlen sind viele 'implied', haben also 1 Byte und brauchen keinen Operanden. Es sind solche mit Opcode xB. Sie dienen vor allem der Registerbewegung und werden weiter unten bei ihrem Gebrauch besprochen.

Die Erweiterungen und Abweichungen der neuen CPU's im Vergleich zur CMOS-CPU R65C02 liegen vor allem bei den Opcodes mit X3, x7 und xF. Mit solchen Opcodes haben wir folgende neue Adressierungen in der Gruppe 1 (arithmetische und logische Befehle, LDA und STA) in Assemblerschreibweise:

,S	z.B. 4,S, relativ zum Stackpointer, im Datenblatt (sr)
(r,S),Y	Indirekt, Pointer liegt r-relativ zum Stackpointer, Nachindizierung mit Y
(d,L)	Indirekt, Pointer 3 Byte (mit Bankadresse) in der Direct Page
(d,L),Y	Indirekt, die effektive Adresse bildet sich wie bei (d,L) mit Nachindizierung durch Y. Auch hier hat der Pointer 3 Byte, inkl. Bankadresse, der Befehl hat wie bei (d,L) jedoch nur 2 Byte
,L	Lang absolut. Die Bankadresse steht im 3. Byte des Befehles
,LX	Lang absolut wie ,L, jedoch Nach-Indizierung mit X

Die hexadezimale Matrix der Opcodes findet sich zur Kontrolle für die späteren Beispiele nebenstehend. Der hiesige Assembler für 65802/65816 ist noch nicht ganz fertig geworden, so daß für die nachstehenden Beispiele noch einige Kompromisse im Listbild in Kauf genommen werden

müssen. Wesentlich ist, daß Symbole und Labels schon mit ihrem Wert in 3 Byte verwaltet werden können und daß für fast alle Befehle schon die erforderlichen Hexbytes generiert werden können.

W65SC816 Microprocessor Op Code Matrix

MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	MSD
0	BRK s 2 8	ORA(d,x) 2 6	COP s 2*8	ORA sr 2*4	TSB d 2*5	ORA d 2 3	ASL d 2 5	ORA(dl) 2*6	PHP s 1 3	ORA imm 2 2	ASL acc 1 2	PHD s 1*4	TSB a 3*6	ORA a 3 4	ASL a 3 6	ORA al 4*5	0
1	BPL r 2 2	ORA(d),y 2 5	ORA (d) 2*5	ORA(sr),y 2*7	TRB d 2*5	ORA d,x 2 4	ASL d,x 2 6	ORA(dl),y 2*6	CLC imp 1 2	ORA a,y 3 4	INC acc 1•2	TCS imp 1*2	TRB a 3•6	ORA a,x 3 4	ASL a,x 3 7	ORA al,x 4*5	1
2	JSR a 3 6	AND(d,x) 2 6	JSL al 4*8	AND sr 2*4	BIT d 2 3	AND d 2 3	ROL d 2 5	AND(dl) 2*6	PLP s 1 4	AND imm 2 2	ROL acc 1 2	PLD s 1*5	BIT a 3 4	AND a 3 4	ROL a 3 6	AND al 4*5	2
3	BMI r 2 2	AND(d),y 2 5	AND(d) 2*5	AND(sr),y 2*7	BIT d,x 2*4	AND d,x 2 4	ROL d,x 2 6	AND(dl),y 2*6	SEC imp 1 2	AND a,y 3 4	DEC acc 1•2	TSC imp 1*2	BIT a,x 3*4	AND a,x 3 4	ROL a,x 3 7	AND al,x 4*5	3
4	RTI s 1 7	EOR(d,x) 2 6	WDM RESERVED	EOR sr 2*4	MVP xyc 3*7	EOR d 2 3	LSR d 2 5	EOR(dl) 2*6	PHA s 1 3	EOR imm 2 2	LSR acc 1 2	PHK s 1*3	JMP a 3 3	EOR a 3 4	LSR a 3 6	EOR al 4*5	4
5	BVC r 2 2	EOR(d),y 2 5	EOR (d) 2*5	EOR(sr),y 2*7	MVN xyc 3*7	EOR d,x 2 4	LSR d,x 2 6	EOR(dl),y 2*6	CLI imp 1 2	EOR a,y 3 4	PHY s 1•3	TCD imp 1*2	JMP al 3*4	EOR a,x 3 4	LSR a,x 3 7	EOR al,x 4*5	5
6	RTS s 1 6	ADC(d,x) 2 6	PER s 3*6	ADC sr 2*4	STZ d 2•3	ADC d 2 3	ROR d 2 5	ADC(dl) 2*6	PLA s 1 4	ADC imm 2 2	ROR acc 1 2	RTL s 1*6	JMP (a) 3 5	ADC a 3 4	ROR a 3 6	ADC al 4*5	6
7	BVS r 2 2	ADC(d),y 2 5	ADC(d) 2*5	ADC(sr),y 2*7	STZ d,x 2*4	ADC d,x 2 4	ROR d,x 2 6	ADC(dl),y 2*6	SEI imp 1 2	ADC a,y 3 4	PLY s 1*4	TDC imp 1*2	JMP(a,x) 3*6	ADC a,x 3 4	ROR a,x 3 7	ADC al,x 4*5	7
8	BRA r 2*2	STA(d,x) 2 6	BRL rl 3*3	STA sr 2*4	STY d 2 3	STA d 2 3	STX d 2 3	STA(dl) 2*6	DEY imp 1 2	BIT imm 2•2	TXA imm 1 2	PHB s 1*3	STY a 3 4	STA a 3 4	STX a 3 6	STA al 4*5	8
9	BCC r 2 2	STA(d),y 2 6	STA (d) 2*5	STA(sr),y 2*7	STY d,x 2 4	STA d,x 2 4	STX d,y 2 4	STA(dl),y 2*6	TYA imp 1 2	STA a,y 3 5	TXS imp 1 2	TXY imp 1*2	STZ a 3*4	STA a,x 3 5	STZ a,x 3*5	STA al,x 4*5	9
A	LDY imm 2 2	LDA(d,x) 2 6	LDX imm 2 2	LDA sr 2*4	LDY d 2 3	LDA d 2 3	LDX d 2 3	LDA(dl) 2*6	TAY imp 1 2	LDA imm 2 2	TAX imp 1 2	PLB s 1*4	LDY a 3 4	LDA a 3 4	LDX a 3 4	LDA al 4*5	A
B	BCS r 2 2	LDA(d),y 2 5	LDA(d) 2*5	LDA(sr),y 2*7	LDY d,x 2 4	LDA d,x 2 4	LDX d,y 2 4	LDA(dl)y 2*6	CLV imp 1 2	LDA a,y 3 4	TSX imp 1 2	TYX imp 1*2	LDY a,x 3 4	LDA a,x 3 4	LDX a,y 3 4	LDA al,x 4*5	B
C	CPY imm 2 2	CMP(d,x) 2 6	REP imm 2*3	CMP sr 2*4	CPY d 2 3	CMP d 2 3	DEC d 2 5	CMP(dl) 2*6	INY imp 1 2	CMP imm 2 2	DEX imm 1 2	WAI imp 1*3	CPY a 3 4	CMP a 3 4	DEC a 3 6	CMP al 4*5	C
D	BNE r 2 2	CMP(d),y 2 5	CMP (d) 2*5	CMP(sr),y 2*7	PEI s 2*6	CMP d,x 2 4	DEC d,x 2 6	CMP(dl),y 2*6	CLD imp 1 2	CMP a,y 3 4	PHX s 1•3	STP imp 1*3	JML (a) 3*6	CMP a,x 3 4	DEC a,x 3 7	CMP al,x 4*5	D
E	CPX imm 2 2	SBC(d,x) 2 6	SEP imm 2*3	SBC sr 2*4	LPX d 2 3	SBC d 2 3	INC d 2 5	SBC(dl) 2 6	INX imp 1 2	SBC imm 2 2	NOP imp 1 2	XBA imp 1*3	CPX a 3 4	SBC a 3 4	INC a 3 6	SBC al 4*5	E
F	BEQ r 2 2	SBC(d),y 2 5	SBC (d) 2*5	SBC(sr),y 2*7	PEA s 3*5	SBC d,x 2 4	INC d,x 2 6	SBC(dl),y 2*6	SED imp 1 2	SBC a,y 3 4	PLX s 1•4	XCE imp 1*2	JSR(a,x) 3*6	SBC a,x 3 4	INC a,x 3 7	SBC al,x 4*5	F

* New W65SC816 Op Codes
• W65SC02 Op Codes

Initialisierung und Rückkehr

Im Beispiel 1 wird die CPU nach dem kalten Reset in den vollen 'native' Modus mit 16 Bit Verarbeitung gerufen und am Ende von dort zurückgeholt. Es ist anmerkenswert, daß der Rechner sich verabschiedet, wenn man die Zeilen 4-6 in die Abfolge REP–CLC–XCE bringt. - Wir sehen, daß die Befehle LDA # und LDY # im 16 Bit Modus nun 3 Byte lang werden. Die Befehle STA und STY haben jedoch die gleiche Länge wie früher. Am Memory Dump ab Stelle 300 bemerken wir, daß letztere jetzt 2 Byte ablegen.

An der Monierung des BRK-Befehles durch den Assembler möge man sich nicht stören. Er ist noch nicht voll implementiert. Wenn E=1, dann ist er wie üblich 1 Byte lang, für E=0 aber hat er die Länge von 2 Byte. – Bei den späteren Listings werden wir die Initialisierung mit CLC–XCE–REP und den Abschluß mit SEP–SEC–XCE meist fortlassen, um die Listen nicht zu sehr auszudehnen.

Initialisieren des Stackpointers

Der Hardware-Stack kann überall in der Bank 0 liegen, und zwar nicht nur in der Speicherseite 1, wie beim 6502. Dementsprechend ist er ein in 16 Bit Breite voll dem Benutzer zugängliches Register. Das gibt größere Freiheiten in der Programmierung, auch im Zusammenhang mit den neuen stack-relativen Adressierungsarten. Insbesondere wird jetzt die Übergabe von Parametern

```
*******************************************************************************
BEISPIEL 1: INITIALISIERUNG UND EXIT
000000                  0001 NAME   =$34
000000                  0002        *=$4000
004000                  0003        .R16            ;ASSEMBLER-ANWEISUNG: ALLES 16
BIT
004000  18              0004        CLC
004001  FB              0005        XCE             ;SET E=0 WITH CARRY=0
004002  C2 30           0006        REP #$30        ;CPU SOLL 16 BIT FAHREN
004004  A9 67 45        0007        LDA #$4567      ;DIREKTOPERAND 16 BIT
004007  8D 00 03        0008        STA $300        ;ZUR KONTROLLE
00400A                  0009
00400A  A0 34 00        0010        LDY #NAME       ;AUCH HIER 16 BIT
00400D  8C 02 03        0011        STY $302        ;IN D3N ZELLEN 302 UND 303
004010  E2 30           0012        SEP #$30
004012                  0013 ;HOLE CPU AUF 8 BIT ZURUECK
004012  38              0014        SEC             ;VORBEREITUNG DES XCE
004013  FB              0015        XCE             ;CPU AUF 8 BIT
004014                  0016        .R08            ;ASSEMBLER-ANWEISUNG
004014  00 EA EA        0017        BRK             ;NOCH NICHT NEU IMPLEMENTIERT
**ERROR 07
004017                  0018        .END
*******************************************************************************
BEISPIEL 2: STACK INITIALISIEREN
000000                  0001        *=$4000
04000                  0002        .R16            ;ASSEMBLER-ANWEISUNG: ALLES 16 E
IT
004000  18              0003        CLC
004001  FB              0004        XCE             ;SET E=0 WITH CARRY=0
004002  C2 30           0005        REP #$30        ;CPU SOLL 16 BIT FAHREN
004004  A2 FF 2F        0006        LDX #$2FFF      ;STACK AB $3000 ABWAERTS
004007  9A              0007        TXS             ;UEBERTRAGUNG NACH SP
004008  20 00 50        0008        JSR $5000       ;LEGE RETURNADRESSE AUF DEN STA
C
00400B                  0009
00400B                  0010        *=$5000         ;HIER LIEGT DAS UNTERPROGRAMM
005000  E2 30           0011        SEP #$30
005002                  0012 ;HOLE CPU AUF 8 BIT ZURUECK
5002  38              0013        SEC             ;VORBEREITUNG DES XCE
005003  FB              0014        XCE             ;CPU AUF 8 BIT
005004                  0015        .R08            ;ASSEMBLER-ANWEISUNG
005004                  0016 ;HIER ERFOLGT ZWANGSABBRUCH
005004  00 FF 00        0017        BRK             ;NOCH NICHT NEU IMPLEMENTIERT
**ERROR 07
005007                  0018        .END
 ERRORS=0001

 AUSGABE SYMBOLTAFEL?
<M>=2FF0 00 DF 00 FF 00 FF 00 FF 66 66 66 66 66 66 0A 40
```

an Unterprogramme erleichtert und auch die Reservierung von Stackbereichen ('frames') für lokale Variablen einzelner Programmteile.

In Beispiel 2 und den nachfolgenden initialisieren wir den Stackpointer auf $2FFF (er zeigt auf die erste freie Stelle). Mit dem Unterprogrammaufruf nach hex 5000 und dem Zwangsabbruch erreichen wir, daß der Stack nach dem Programmlauf besichtigt werden kann. Wir finden dann in Adresse =FFE eine 0A und in 2FFF eine 40. Das ist die übliche Ablage einer Return-Adresse bei JSR, nur diesmal in einem beliebigen Stackbereich.

JSL – Jump Subroutine Long

Dieser neue Befehl gestattet es, ein Unterprogramm in einer Zielbank anzusprechen. Dabei wird zunächst die Adresse der Quellbank auf den Stack gebracht (in unserem Falle 00), danach das High Byte und dann das Low Bytes des Aufrufes, genauer: Adresse des auf JSL folgenden Befehles-1, wie üblich und erwartet. - Im Beispiel 3 ist der JSL-Befehl noch nicht voll im Assembler implementiert. Seine Zieladressen werden behelfsweise noch mit .WOR und .BYT definiert, wie auch bei einigen wenigen anderen Befehlen auch noch. JSL hat also 3 Adreßbytes. Nach dem Pro-

```
*******************************************************************************
      3: JUMP SUBROUTINE LONG

004008  22              0011        JSL $5000       ;OPCODE FUER JUMP SUBROUTINE LO
N
004009  00 50           0012        .WOR $5000
00400B  33              0013        .BYT $33        ;ERSATZ FUER ZIELBANK
00400C                  0014
00400C                  0015        *=$5000         ;HIER LIEGT DAS UNTERPROGRAMM
05000   E2 30           0016       SEP #$30
005002                  0017 ;HOLE CPU AUF 8 BIT ZURUECK
005002  38              0018        SEC             ;VORBEREITUNG DES XCE
005003  FB              0019        XCE             ;CPU AUF 8 BIT
005004                  0020        .ROB            ;ASSEMBLER-ANWEISUNG
005004                  0021 ;HIER ERFOLGT ZWANGSABBRUCH
005004  00 EA EA        0022        BRK             ;NOCH NICHT NEU IMPLEMENTIERT
**ERROR 07
005007                  0023        .END
```

grammlauf hat JSL seine Return-Adresse auf dem Stack abgelegt. Wir finden dort ab Zelle 2FFD die Hexbytes 0B 40 00, wie erwartet.

Zu JSL auf dem hier verwendeten Chip 65802 noch folgender Hinweis: Die CPU kann auf den Pins zwar keine Bankadressen aussenden, versteht gleichwohl die Sprache des 65816 und legt bei JSL ebenfalls eine Bankadresse ab, obwohl sie die Bank 00 gar nicht verlassen kann. Die CPU's sind also insoweit kompatibel. – JSL gehört zu den wenigen Befehlen, die die Bankadresse im höheren Teil des Programmzählers beeinflussen.

JML – Jump Long

Dieser neue Befehl erlaubt Sprünge in eine andere Programmbank. Auch bei ihm wird die Bankadresse im Programmzähler verändert, wie wir am Beispiel 4 zeigen wollen: Wir starten in der Programmbank 00 und simulieren ab Zeile 8 einen absoluten Sprung in Bank 33 (Zeile 10). Dort rufen wir ab Zeile 13 ein Unterprogramm in Bank 45, Adresse 5000 auf und erwarten, daß dabei auf dem Stack auch die Bankadresse 33 für die Rückkehr abgelegt wird, weil wir ja angeblich in dieser Simulation aus Bank 33 kommen. In der Tat finden wir nachher ab Adresse 2FFD die Bytes 03 45 33 für die Adresse-1 des Befehls, der auf JSL folgen würde.

```
*******************************************************************************
      4: JUMP LONG

004008  5C              0008        .BYT $5C        ;OPCODE FUER JML
004009  00 45           0009        .WOR $4500      ;ADRESSE INNERHALB BANK
00400B  33              0010        .BYT $33        ;ERSATZ FUER ZIELBANK
00400C                  0011
00400C                  0012        *=$4500         ;SIMULIERE IN BANK 33
004500  22              0013        JSL $5000       ;SUBROUTINE IN BANK 45
004501  00 50           0014        .WOR $5000
004503  45              0015        .BYT $45
004504                  0016
004504                  0017        *=$5000         ;HIER LIEGT DAS UNTERPROGRAMM
005000  E2 30           0018        SEP #$30
005002                  0019 ;HOLE CPU AUF 8 BIT ZURUECK
005002  38              0020        SEC             ;VORBEREITUNG DES XCE
005003  FB              0021        XCE             ;CPU AUF 8 BIT
005004                  0022        .ROB            ;ASSEMBLER-ANWEISUNG
005004                  0023 ;HIER ERFOLGT ZWANGSABBRUCH
005004  00 04 00        0024        BRK             ;NOCH NICHT NEU IMPLEMENTIERT
```

JML –Jump Long (Indirect)

Eine andere Form der Adressierung ist JML (Adresse). Der Adreßpointer für JML hat jetzt 3 Bytes, wobei das dritte die Nummer der Zielbank enthält, die bei dieser Adressierung in die Bankadresse des Programmzählers geladen wird. Das läßt sich wieder sehr schön beobachten, wobei zu erwähnen ist, daß dieser JML-Befehl selbst auch nur 3 Bytes insgesamt hat.

```
*******************************************************************************
      5: JUMP LONG INDIRECT

004008  DC            0008          JML $4500         ;INNERHALB DER QUELLBANK
004009  00 45         0009          .WOR PTRL         ;ADRESSE INNERHALB BANK
00400B                0010
00400B                0011          *=$4500
004500  00 50         0012 PTRL     .WOR $5000        ;INDIREKTE ADRESSE
004502  45            0013          .BYT $45          ;INDIREKTE ZIELBANK
004503                0014
004503                0015          *=$5000           ;HIER LIEGT DAS 1. ZIEL
005000  22            0016          JSL $5500         ;BANKADRESSE 45 AUF STACK
005001  00 55         0017          .WOR $5500
005003  66            0018          .BYT $66
005004                0019          *=$5500           ;HIER LIEGT DAS UNTERPROGRAMM
005500  E2 30         0020          SEP #$30
005502                0021 ;HOLE CPU AUF 8 BIT ZURUECK
005502  38            0022          SEC
05503  FB            0023          XCE               ;CPU AUF 8 BIT
005504                0024          .R08              ;ASSEMBLER-ANWEISUNG
005504                0025 ;HIER ERFOLGT ZWANGSABBRUCH
005504  00            0026          BRK               ;NOCH NICHT NEU IMPLEMENTIERT

*******************************************************************************
BEISPIEL 6: BRANCH LONG ALWAYS

004008  82            0008          BRL $5000
004009  F5 0F         0009          .WOR $0FF5        ;DIESES IST DER OFFSET
00400B                0010
00400B                0011          *=$5000           ;HIER LIEGT DAS ZIEL DES BRL
005000  E2 30         0012          SEP #$30
005002                0013 ;HOLE CPU AUF 8 BIT ZURUECK
005002  38            0014          SEC               ;VORBEREITUNG DES XCE
005003  FB            0015          XCE               ;CPU AUF 8 BIT
005004                0016          .R08              ;ASSEMBLER-ANWEISUNG
005004                0017 ;HIER ERFOLGT ZWANGSABBRUCH
005004  00            0018          BRK               ;NOCH NICHT NEU IMPLEMENTIERT
```

In Zeile 8 des Beispieles 5 wird mit JML ein bei hex 4500 in der Quellbank liegender Pointer indirekt angesprungen. Er zeigt auf die Adresse 45 5000 (45 ist die Bank). In Zeile 16 erfolgt ein Weitersprung mit JSL $665500, um die Bankadresse später auf dem Stack beobachten zu können. Wir finden dann die Bytes 03 50 45 auf dem Stack, wobei 45 für die Bank steht, aus der in Zeile 16 aufgerufen wurde.

Der Befehl JML (indirekt) ist also eine logische Erweiterung des bekannten JMP (indirekt) mit Opcode 6C, wie wir ihn beim 6502 kennen. Wir sind damit in der Lage, in jeder Programmbank Sprungleisten anzulegen, die Programmteile in anderen Banken anspringen. – Das Verhalten des 65802 ist auch damit kompatibel zum 65816, auch wenn ersterer nicht aus seiner Bank 0 herauskommt.

BRL – Branch Long, Always

Auch dieser Befehl ist noch nicht voll im Assembler implementiert. Er wird behelfsweise im Beispiel 6 erzeugt. Die Logik ist wie gewohnt: Die Adreßdistanz (der Offset) wird wie üblich ab Adresse des Folgebefehles berechnet. Wir machen eine Verzweigung (always) nach Adresse 5000, wo wir wieder wie bisher abbrechen. Diesem lauffähigen Beispiel ist zu entnehmen, daß das Low Byte des Offsets dem Opcode direkt folgt. Das dritte Byte des Befehls ist das High Byte des Offsets.

BRL ersetzt den JMP-Befehl, wenn positions-unabhängiger Code erzeugt werden soll.

PER – Push Effectice Program Counter Relative Indirect Address

Auch: Push Program Counter relative Data Word on Stack. Diese recht gefährlich klingenden Bezeichnungen des PER-Befehles sind zunächst einmal etwas verwirrend. Wir müssen die Codierungsphase gedanklich von der RUN-Time trennen. In der Codierung hängen wir an den Opcode einen

```
000000                  0000 ZULU   =$5000          ;ADRESSE EINES POINTERS
00000                   0001        *=$4000
004000                  0002 START
004000                  0003        .R16            ;ASSEMBLER-ANWEISUNG: ALLES 16
BIT
004000  18              0004        CLC
004001  FB              0005        XCE             ;SET E=0 WITH CARRY=0
004002  C2 30           0006        REP #$30        ;CPU SOLL 16 BIT FAHREN
004004  A2 FF 2F        0007        LDX #$2FFF      ;STACK AB $3000 ABWAERTS
04007   9A              0008        TXS             ;UEBERTRAGUNG NACH SP
********************************************************************************
BEISPIEL 8: DER PER-BEFEHL

00400B  62 F5 0F        0010        PER ZULU-*-3    ;LABEL - PC DES PER-BEFEHLES
00400B                  0011
00400B  E2 30           0012        SEP #$30
00400D                  0013 ;HOLE CPU AUF 8 BIT ZURUECK
00400D  38              0014        SEC             ;VORBEREITUNG DES XCE
00400E  FB              0015        XCE             ;CPU AUF 8 BIT
00400F                  0016        .R08            ;ASSEMBLER-ANWEISUNG
00400F  4C 82 E1        0017        JMP $E182       ;COMIN BEIM AIM
004012                  0018 ;HIER ERFOLGT ZWANGSABBRUCH
004012  00              0019        BRK             ;NOCH NICHT NEU IMPLEMENTIERT
```

Offset an (eine relative Adresse wie bei BRL), der nach vorne oder rückwärts zeigt, im allgemeinen auf ein Label, das einen Pointer enthält. Zur Laufzeit addiert PER diesen Offset zum augenblicklichen Stand des Programmzählers PC (der schon auf den Folgebefehl des PER zeigt). Die Summe bildet eine effektive Adresse, die in der Form von 2 Bytes auf den Stack gelegt wird, natürlich unter Anpassung des Stapelzeigers. Zur Runtime besteht also eine gewisse Verwandschaft zum Befehl BRL, Branch Long. Nur daß PER nicht den Programmzähler umlädt, sondern das entsprechende Rechenergebnis auf den Stack legt.

Die auf dem Stack abgelegte effektive Adresse kann einen Basis-Pointer für die stack-relativen Adressierungen bilden. Der Vorteil des PER liegt darin, daß er positions-unabhängigen Code ermöglicht, der sich erst zur Laufzeit in wirkliche Adressen umsetzt. – Beim Lauf des Beispieles 7 finden wir den Offset F50F (Low/High) im Code. Bei der Ausführung gelangt dann die effektive Adresse von ZULU=$5000 auf den Stack.

Wenn der Start des Programmes in 4002 wäre, und damit auch der PER-Befehl 2 Adressen weiter, so wäre der Offset F30F. Er würde zur Laufzeit wiederum 5000 für ZULU ergeben.

Den Befehl PER wird man auch für berechnete Sprünge auf Unterprogramme verwenden können. Hierfür steht allerdings bequemer die neue Adressierungsart JSR (Adresse,X) mit Opcode FC zur verfügung, die wie in BASIC ein ON GOSUB ermöglicht. Für den entsprechenden Befehl JMP (Adresse,X) haben wir in früheren Heften schon die Benutzung von Sprungverteilern beschrieben.

PEI – Push Effective Indirect Address

Oder: Push Direct Data Word on Stack. Auch hier ist die englische Bezeichnung nicht so ganz hilfreich. Wir sollten lieber sagen: Kopiere den Inhalt eines in der Direct Page liegenden Pointers auf den Stack. Also: Nicht eine Adresse (wie bei PEA), sondern den Inhalt (2 Bytes) unter Adresse. Oder man sagt: Dupliziere einen Pointer auf den Stack, wo er wie das Original (die Matrize) für die stack-relativen Adressierungsarten dienen kann = Parameterübergabe, Übergabe von Pointern an ein Unterprogramm.

Damit ist der Gebrauch von PEI nicht erschöpft. Denken wir an Unterprogramme des Rechnens. Ein Zwischenergebnis ist z.B. in der Direct Page in 8 Byte enthalten. Es soll einem Unterprogramm zur Verfügung gestellt werden, das z.B. multipliziert oder umformt. Man kann dann mit PEA den Pointer auf die Variable übergeben, so daß das Unterprogramm sich den Wert selbst verschafft. Man kann aber auch 4mal den Befehl PEI, jeweils um 2 Bytes weitergesetzt geben, um den Inhalt der Variablen auf den Stack zu bringen. Es hängt vom zu lösenden Problem ab, welchen Weg man gehen wird.

```
000000                  0000 ZULU   =$5000          ;ADRESSE EINES POINTERS
000000                  0001 POINTR =$33
000000                  0002        *=POINTR
000033  65 87           0003        .WOR $8765
000035                  0004
000035                  0005        *=$4000
04000                  0006 START
004000                  0007        .R16            ;ASSEMBLER-ANWEISUNG: ALLES 16
BIT
004000  18              0008        CLC
004001  FB              0009        XCE             ;SET E=0 WITH CARRY=0
004002  C2 30           0010        REP #$30        ;CPU SOLL 16 BIT FAHREN
004004  A2 FF 2F        0011        LDX #$2FFF      ;STACK AB $3000 ABWAERTS
004007  9A              0012        TXS             ;UEBERTRAGUNG NACH SP
*******************************************************************************
 BEISPIEL 9: PEA UND PEI
00400B  F4 00 50        0014        PEA ZULU        ;BRINGE ADRESSE DES SYMBOLS
00400B  D4 33           0015        PEI POINTR      ;BRINGE INHALT VON POINTR
00400D                  0016
00400D  E2 30           0017        SEP #$30
00400F                  0018 ;HOLE CPU AUF 8 BIT ZURUECK
00400F  38              0019        SEC             ;VORBEREITUNG DES XCE
004010  FB              0020        XCE             ;CPU AUF 8 BIT
004011                  0021        .ROB            ;ASSEMBLER-ANWEISUNG
004011  4C B2 E1        0022        JMP $E182       ;COMIN BEIM AIM
004014                  0023        .END
 ERRORS=0000

 AUSGABE SYMBOLTAFEL?
<M>=2FF0 99 99 99 99 99 99 99 99 99 99 99 99 65 87 00 50
```

Die Ansprache von Zeigern (Pointern) und Parametern erfolgt relativ zum Stapelzeiger, wie wir noch sehen werden. Beispiel 9 zeigt die Befehle PEA und PEI im Einsatz. Nachdem wir POINTR mit der Assembler-Anweisung .WOR $8765 vorgeladen haben, finden wir am Schluß folgendes auf dem Stack: 65 87 von PEI (Low/High) und 00 50 von PEA (Adresse).

Stack-Relative Adressierung, ,S

Nachdem wir mit PEA und PEI (ggfs. auch mit PHA, PHX, PHY),Adressen und andere Parameter auf den Stack gebracht haben, sollten wir sie dort auf eine zunächst einfache Weise ansprechen. Mit den Befehlen des 6502 war das immer sehr umständlich: Wir mußten uns mit TSX (Stackpointer nach X übertragen) zunächst den Inhalt des Stackregisters besorgen, um dann relativ zur Adresse hex 100 Parameter in der Adressierungsart vornehmlich Absolut,X eine Entnahme, Beschickung oder Verarbeitung vorzunehmen.

Bei den neuen Prozessoren nun kann der Stackpointer als ein weiteres Indexregister benutzt werden. Der Stack der 65802 und 65816 liegt immer in der Bank 0. Er kann daher aus allen Bänken heraus benutzt werden, um solche Bearbeitungen vorzunehmen. Unser Beispiel 10 bringt in Zeile 15 den definierten Direktoperanden 11 22 auf den Stack. Er wird in Zeile 19 angesprochen, und zwar um 1 relativ zum Stackpointer, weil dieser ja in der 65xxx-Familie immer auf den nächst freien Platz zeigt (im Unterschied zu Motorola, dort zeigt er auf den zuletzt beschickten Platz). Die Addition in Zeile 19 bedeutet in diesem einfachen Beispiel eine Verdoppelung des Akkus, weil er zwischen den Zeilen nicht verändert wurde. Wir finden nach dem STA-Befehl in den Zellen 300/301 den erwarteten Wert von 44 22 (Low/High Order).

Statt nach Adresse 300 hätte der STA-Befehl auch relativ zum Stackpointer gegeben werden können, um z.B. innerhalb eines Unterprogrammbereiches auf dem Stack ('frame') neu berechnete Werte lokaler Variablen dort abzulegen.

Wir versuchen das im Beispiel 11: In allen diesen Beispielen ist der Stackpointer willkürlich auf hex 2FFF initialisiert. Sofort oberhalb, ab 3000 haben wir zwei Operanden plaziert, hier mit der Anweisung .WOR des Assemblers. Sie hätten natürlich auch von Programm dorthin gelangen können. Die Operanden werden jetzt dezimal addiert. Das Ergebnis wird mit STA-Befehl in

```
*******************************************************************************
BEISPIEL 10: STACK-RELATIV
004008  A9 22 11       0014          LDA #$1122        ;DIREKTOPERAND
00400B  48              0015          PHA               ;PARAMETER AUF DEN STACK
00400C  18              0016          CLC               ;VORBEREITUNG ADDITION
0400D                  0017 ;DER SP ZEIGT AUF DIE FREIE STELLE,
00400D                  0018 ;DAHER OFFSET 1 GEGEN SP IM BEFEHL ADC
00400D  63 01           0019          ADC 1,S           ;STACK-RELATIV
00400F                  0020 ;ADC BEDEUTET HIER VERDOPPELUNG IM AKKU
00400F                  0021 ;WEIL ER NOCH NICHT VERAENDERT WURDE
00400F  8D 00 03        0022          STA $300          ;ABLAGE ZUR KONTROLLE
004012                  0023
004012  E2 30           0024          SEP #$30
004014                  0025 ;HOLE CPU AUF 8 BIT ZURUECK
004014  38              0026          SEC               ;VORBEREITUNG DES XCE
004015  FB              0027          XCE               ;CPU AUF 8 BIT
004016                  0028          .R08              ;ASSEMBLER-ANWEISUNG
004016  4C 82 E1        0029          JMP $E182         ;COMIN BEIM AIM
004019                  0030          .END
 ERRORS=0000

 AUSGABE SYMBOLTAFEL?
<M>=0300 44 22 00 FE FF F7 00 63 00 63 00 FE FF F7 00 63

*******************************************************************************
 BEISPIEL 11: ADDITION IM FRAME
004008  F8              0009          SED               ;DEZIMALER MODUS
004009  18              0010          CLC               ;VORBERREITUNG DES ADC
00400A  A3 01           0011          LDA 1,S           ;1. OPERAND STACK-RELATIV
00400C  63 03           0012          ADC 3,S           ;2. OPERAND, DITO
00400E  83 01           0013          STA 1,S           ;ABLAGE ERGEBNIS, DITO
004010  D8              0014          CLD               ;BINAERER MODUS ZURUECK
004011                  0015
004011  E2 30           0016          SEP #$30
004013                  0017 ;HOLE CPU AUF 8 BIT ZURUECK
004013  38              0018          SEC               ;VORBEREITUNG DES XCE
004014  FB              0019          XCE               ;CPU AUF 8 BIT
004015                  0020          .R08              ;ASSEMBLER-ANWEISUNG
004015  4C 82 E1        0021          JMP $E182         ;COMIN BEIM AIM
004018                  0022
004018                  0023          *=$3000           ;OBERHALB DES SP
003000  09 01           0024          .WOR $109         ;ABLAGE PER ASSEMBLER
003002  08 01           0025          .WOR $108         ;DITO, 2. OPERAND
003004                  0026          .END
 ERRORS=0000

 AUSGABE SYMBOLTAFEL?

<M>=3000 09 01 08 01 05 04 00 00 05 04 00 00 05 04 00 00
```

den Stackbereich nach 3000/3001 ('in den frame') zurückgeschrieben. An diesem Beispiel sehen wir, daß der dezimale Modus auch in 16 Bit Breite arbeitet. Bemerkenswert ist, daß er in entsprechender Form nicht auf den CPU's anderer Familien implementiert wurde und daß auch das verbreitete Microsoft BASIC für 65xx von ihm keinen Gebrauch macht.

Zeile 9 ruft den dezimalen Modus herein, Zeile 14 verläßt ihn. Das Ergebnis der Addition in den Zeilen 11 und 12 ist um 1 relativ zum Stackpointer 2FFF zu erwarten, nämlich in den Zellen 3000/3001. Tatsächlich finden wir dort später die Bytes 17 02 (Low/High) aus der Addition von 109+108=0217.

Anmerkung: Bei der Verarbeitung von Datenfeldern in den arithmetischen Operationen Addition, Subtraktion, Multiplikation und Division waren wir es nach den meisten veröffentlichten Vorschlägen gewohnt, die geringwertigste Ziffer 'ganz rechts' an die höchste Adresse im Datenfeld zu legen und uns dann mit DEX oder DEY zu niedrigeren Adressen durchzuarbeiten, um die Befehle CPX # und entsprechend für CPY zu umgehen, die bei aufsteigender Verarbeitung notwendig werden. In Schleifen können wir mit BMI und BEQ abbrechen, wenn wir mit DEX oder DEY

programmiert haben. - An den Beispielen sehen wir, daß die CPU's im 16 Bit Modus das High Byte immer an die höhere Adresse ablegen (bei Motorola ist auch das wieder umgekehrt, was kein Werturteil bedeutet, nur einen Hinweis). Das erfordert im Prinzip ein Denken 'von links nach rechts' (politisch heißt das vielversprechend: Die Wende) mit den Befehlen INX und INY für Datenfelder (doppelte Erhöhung der Indices wegen des 16 Bit Modus). Die Prüfung, ob ein Feld abgearbeitet ist, wird danach also mit CPX und CPY Direktwert erfolgen.. Diese Verarbeitung 'von links nach rechts' ist bei den arithmetischen Operationen im 16 Bit Modus allerdings nicht zwingend. Wir können weiterhin 'stur' wie bisher mit rückläufigen Adressen arbeiten, wenn wir das Datenformat insbesondere für die Ausgabe beachten. Wichtig wird dann wohl der SWA-Befehl sein, der in der abgedruckten Opcode-Matrix mit hex EB noch mit XBA bezeichnet wurde. Er vertauscht die oberen und unteren Hälften des gemeinsamen Akkus C, der aus den Hälften A und B besteht (daher XBA). – Über die Initialisierung der höheren jeweiligen Registerhälften informiere man sich im Datenblatt.

Beim AIM haben wir im Betriebssystem die Routine NUMA, die den Akku als 2 Hexbytes auf die Ausgabe schickt. WRAX tut entsprechendes zuerst für den Akku, dann für den Inhalt des Registers X. Unter Einsatz des Befehles SWA könnte man nun gleiches für die oberen und unteren Hälften des 16 Bit breiten Akkus C vornehmen.

Register-Transferbefehle

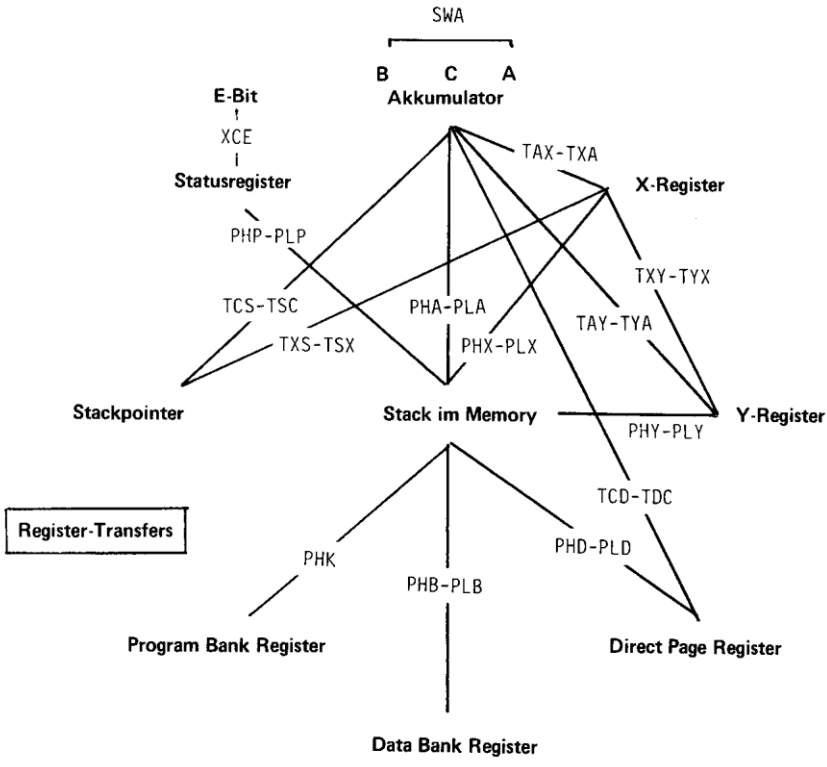
In den vorstehenden Beispielen wurde bereits der Befehl XCE zum Austausch des 'Emulation Mode' benutzt: E=1 = 6502-Modus, E=0 = native Modus. Er arbeitet über Austausch des Carry-Bit gegen das E-Bit in der Statuserweiterung.

Es sind daneben viele neue Befehle zum Registertransfer hinzugekommen, für die wir am besten die möglichen Pfade in der nebenstehenden Abbildung aufzeigen. Zum Merken der Befehle kann man folgende Hinweise geben: Ein 'PH' bedeutet 'Push on Stack', ein 'PL' ist 'Pull from Stack', also erstens Ablage auf dem Stack und zweitens Laden aus dem Stack. Ein 'T' bedeutet Transfer, Übertragung zwischen Registern. Ein 'X' steht für Exchange, Austausch. Bei 'T' und 'X' steht der erste Folgebuchstabe für das Quell- und der zweite für das Zielregister. Der in den Datenblättern gelegentlich noch XBA genannte Befehl vertauscht die obere (B) und untere Hälfte (A) im 16 Bit breiten Akkumulator C. Er wird künftig wohl SWA heißen (Swap).

Wenn wir die Grafik betrachten, so sehen wir erhebliche Unterschiede zum 6502 und auch große Unterschiede noch zur CMOS-CPU R65C02. Die Indexregister X und Y können direkt ge- und entstackt sowie untereinander übertragen werden, ohne daß der Akku als Durchgangsstation benötigt wird (PHX, PLX, PHY, PLY, TXY, TYX).

Der Inhalt aller Register kann auf dem im Memory der Bank 0 gelegenen Stack gebracht werden. Nur das Program Bank Register kann von dort nicht geladen werden, was ganz sicher einen Schutz vor der unbeabsichtigten Zerstörung der Programm-Umgebung bedeuten soll. Die Befehle sind 'implied', haben also nur 1 Byte als Opcode und dahinter keinen Operanden. Sie sind für den jeweiligen 8- oder 16-Bit Modus gleich.

Neu sind die Verbindungswege Akku zu Stackpointer TCS und TSC. Bisher konnte nur der Index X den Stackpointer laden oder beschicken (TSX, TXS) Auch neu sind die Übertragungsmöglichkeiten für das Datenbankregister mit PHB und PLB. Es ist in die Bildung der effektiven Adresse bei vielen datenverarbeitenden Befehlen einbezogen. Eine wichtige Rolle spielt das Direct Register für die Positionierung der 'Zero Page' irgendwo im RAM mit PHD und PLD, bzw. TCD und TDC in Verbindung mit dem Akku C. Dieses Register legt gewissermaßen ein Fenster von 256 Zellen in den Speicher, in dem im Context mit der verkürzten Adressierung und schnelleren Befehlsausführung jeweils gearbeitet werden kann.



Direct Page Adressierung beliebig eingerichtet

In der Programmierung des 6502 hat man bisher fast immer am Mangel freier Speicherstellen in der Zero Page gelitten, zuviele wurden vom System (Betriebssystem, Editor, Sprachen) verbraucht. Es ging dabei um die für die Zero Page zur Verfügung stehenden leistungsfähigen Adressierungsarten, die dem 6502 eine große Zahl von Hilfsregistern (Zeigern) verleihen. - Die hier besprochenen Prozessoren helfen dem Mangel gründlich ab. Das 'Direct Register' (wir sollten vielleicht wie bei Motorola besser sagen: Das Direct Page Register) kann auf jede auch krumme Adresse geladen oder gelegt werden. Man kann je Programm, Betriebssystem, Sprache, Task usw. eine oder mehrere voneinander getrennt geführte Direct Pages verwalten, wenn man das Direct Page Register beim Eintritt in ein Programmsegment umlädt. - Frühere Techniken arbeiteten oft mit einer sichernden Umlagerung der Zero Page-Inhalte.

Unser Beispiel 12 gibt eine Demo: Der Akku wird 'immediate' mit $3002 geladen. Über den Verbindungsweg TCD (Transfer Accu to Direct Page Register) wird dieses auf 3002 für die Basis der Direct Page initialisiert.

Am Ende des Programmes wird ein Pointer per .WOR $4008 in den Zellen ab 3002 abgelegt. Der Pointer weist also auf eine Speicherstelle, die der Zeile 9 entspricht. Wir dürfen also damit rechnen, daß der Akku in Zeile 12 indirekt per Pointer die Bytes A9 02 aus den Zellen 4008/4009 lädt. Er tut es tatsächlich! Später finden wir in der Ablage ab Adresse 300 die Bytes A9 und 02.

```
*******************************************************************************
BEISPIEL 12: BELIEBIGE DIRECT PAGE
004008  A9 02 30      0009         LDA #$3002       ;DIREKTOPERAND
00400B  5B            0010         TCD              ;INS DIRECT PAGE REGISTER
00400C                0011 ;HOLE JETZT INDIREKT PER POINTER
00400C  B2 00         0012         LDA (0)
00400E  8D 00 03      0013         STA $300         ;ABLAGE ZUR BEOBACHTUNG
004011                0014
004011  A9 00 00      0015         LDA #0           ;WIEDERHERSTELLUNG DER
004014  5B            0016         TCD              ;ALTEN ZERO PAGE
004015                0017
004015  E2 30         0018         SEP #$30
004017                0019 ;HOLE CPU AUF 8 BIT ZURUECK
004017  38            0020         SEC              ;VORBEREITUNG DES XCE
004018  FB            0021         XCE              ;CPU AUF 8 BIT
004019                0022         .R08             ;ASSEMBLER-ANWEISUNG
004019  4C 82 E1      0023         JMP $E182        ;COMIN BEIM AIM
401C                 0024
00401C                0025         *=$3002
003002  08 40         0026         .WOR $4008       ;ADRESSPOINTER FUER INDIREKT
03004                 0027         .END
*******************************************************************************
      13: BELIEBIGE DIRECT PAGE, INDIZIERT
004008  A9 02 30      0009         LDA #$3002       ;DIREKTOPERAND
00400B  5B            0010         TCD              ;INS DIRECT PAGE REGISTER
00400C                0011 ;HOLE JETZT INDIREKT PER POINTER
00400C  A0 FB 0F      0012         LDY #$FFB        ;DAMIT $4008+$FFB=$5003
00400F  B1 00         0013         LDA (0),Y        ;INDIREKT, INDIZIERT
004011  8D 00 03      0014         STA $300         ;ABLAGE ZUR BEOBACHTUNG
004014                0015
004014  A9 00 00      0016         LDA #0           ;WIEDERHERSTELLUNG DER
004017  5B            0017         TCD              ;ALTEN ZERO PAGE
004018                0018
004018  E2 30         0019         SEP #$30
00401A                0020 ;HOLE CPU AUF 8 BIT ZURUECK
00401A  38            0021         SEC              ;VORBEREITUNG DES XCE
00401B  FB            0022         XCE              ;CPU AUF 8 BIT
00401C                0023         .R08             ;ASSEMBLER-ANWEISUNG
00401C  4C 82 E1      0024         JMP $E182        ;COMIN BEIM AIM
00401F                0025
00401F                0026         *=$3002
003002  08 40         0027         .WOR $4008       ;ADRESSPOINTER FUER INDIREKT
3004                  0028        *=$5003          ;HIER PARAMETER ANLEGEN
005003  67 55         0029         .WOR $5567
005005                0030         .END
```

Dieses einfache Beispiel zeigt: Das Direct Page Register kann auf einen beliebigen Speicherbereich zeigen, der dann die Basis für die verkürzte und leistungsfähige Adressierung in der Direct Page ist. Wir erhalten damit ausreichend 'Hilfsregister' im Memory. - Auf den Einfluß des Datenbank-Registers bei der Bildung der Adressen kommen wir später im Zusammenhang zurück.

Der Läufer Y erfaßt eine ganze Bank

Nach dieser angenehmen Entdeckung bezüglich des Direct Page Registers wollen wir im Beispiel 13 das I-Tüpfelchen auf die indirekte Adressierung setzen. Diesmal indizieren wir mit dem Y-Register in 16 Bit Breite, mit Werten größer als dez. 255 also. Wir wollen per Pointer auf die effektive Adresse hex 5003/5004 aufsetzen, wo wir per .WOR willkürlich 55 67 abgelegt haben. Der Pointer liegt in Zelle 3002 und weist auf die Adresse 4008. Die Distanz von dort zu 5003 ist hex FFB. Mit diesem Wert laden wir Y 'immediate'.

Nach dem Programmlauf finden wir in der Ablage ab Adresse 300 tatsächlich 67 und 55, unseren bei 5000 versteckten Parameter. Damit ist gezeigt, daß Y bei der indirekten Adressierung beliebig über den Speicher laufen kann. Das Datenblatt hebt sogar hervor, daß der Datenspeicher 'contiguous' ansprechbar sei, d.h. 'zusammenhängend'. An den Grenzen der Datenbänke gibt es keinen Sprung oder ein 'Umwickeln' (wrap around). - Demgegenüber ist der Programmspeicher in Bänke segmentiert.

Eine Beobachtung sei zur Sicherheit mitgeteilt: Wenn man das Direct Page Register nicht wie in den Zeilen 16/17 auf Null zurücksetzt, dann wird man Überraschungen erleben, weil das Betriebsprogramm (hier des AIM), soweit es auf die Zero Page angewiesen ist, diese nun mit einem Versatz anspricht, der dem niederwertigen Teil im Direct Page Register entspricht. Typisch ist das beim Wiedereintritt in den Editor zu bemerken.

Weitere Registerbewegungen

Das vorstehende Diagramm zeigt viele schnell zu verstehende Bewegungsmöglichkeiten zwischen Registern und auch dem Stack. Man wird künftig die Rollen z.B. der Register X und Y gegeneinander vertauschen können, wenn es die gewollten Adressierungsarten erfordern sollten. Man stackt dann das eine Register, lädt es vom anderen her und lädt im Dreieckstausch das zuerst abgebende Register. - Sorgfältig müssen wir uns jetzt die Register für die Datenbank und für die Programmbank und auch ihre möglichen Bewegungen ansehen, weil sie an der Adressenbildung beteiligt sind.

Das Datenbankregister der CPU

Bei sehr vielen Adressierungen ist der augenblickliche Stand des Datenbankregisters bei der Bildung der effektiven Adresse im Spiel. Bei anderen wird immer nur die Datenbank Null angesprochen und schließlich gibt es Adressierungen, bei denen die anzusprechende Datenbank entweder als Byte in einem (verlängerten) Befehl enthalten ist oder aber im dritten Byte eines Pointers in der Direct Page.

Das nachfolgende Schema lehnt in seinen Bezeichnungen an diejenigen im Datenblatt von Western Design Center an. E, M und X stehen für die Status-Bits für Emulation, Akku- und Registerbetrieb in den Breiten. Bytes steht für Gesamt-Befehlslänge und Cycles für die Maschinenzyklen pro Befehl, vor dem Schrägstrich im 8 Bit-Modus, dahinter im 16 Bit Modus. +DR heißt, daß der Inhalt des Direct Page Registers zur im Befehl genannten Adresse hinzuaddiert wird. Das sind Befehle vom Typ d, d,x und d,y, die immer nur in Bank 0 ausgeführt werden. Das Datenbankregister ist

Adressierungsarten			**Bytes E=1/0**	**Cycles E/M/X=1/0**	**Adreßbreite 65816**
	Immediate		2/3	2/3	1/2 Byte
A	Akku		1	2	–
d	Direct (Zero Page)	Bank 0	2	3/4	+DR=16
d,x	Direct, Indexed with X	Bank 0	2	4/5	+DR=16
d,y	Direct, Indexed with Y	Bank 0	2	4/5	+DR=16
a	Absolute		3	4/5	+DB=24
a,x	Absolute, Indexed with X		3	4/5	+DB=24
a,y	Absolute, Indexed with Y		3	4/5	+DB=24
(d)	Direct Indirect	+DR *	2	5/6	+DB=24
(d,x)	Direct Indexed Indirect	+DR *	2	6/7	+DB=24
(d),y	Direct Indirect Indexed	+DR *	2	5/6	+DB=24
(a)	Absolute Indirect	Bank 0	3	6	+PB=24
(a,x)	Absolute Indexed Indirect	Bank 0	3	6	+PB=24
al	* Absolute Long		4	5/6	24
al,x	* Absolute Indexed Long		4	5	24
(dl)	* Direct Indirect Long	1)	2	6/7	24
(dl),y	* Direct Indirect Indexed Long	+DR 1)	2	6/7	24
r,s	* Stack Relative	(Bank 0)	2	4/5	16
(r,s),y	* Stack Relative Indirect Indexed	(Bank 0)	2	7/8	+DB=24

*) Nur GS65SCxxx 1) Pointer in 3 Bytes

```
*******************************************************************************
 BEISPIEL 14: PARAMETER AUS ANDERER BANK
004008  A2 00 00      0009          LDX #0
41400B  BF 02 30 41   0010          LDA $413002,LX
00400F  8D 00 03      0011          STA $300          ;ABLAGE ZUR BEOBACHTUNG
4012                  0012
004012  8B            0013          PHB               ;PUSH DATA BANK REGISTER
004013                0014
004013  E2 30         0015          SEP #$30
004015                0016 ;HOLE CPU AUF 8 BIT ZURUECK
004015  68            0017          PLA               ;HOLE DBR
004016  8D 02 03      0018          STA $302
004019  38            0019          SEC               ;VORBEREITUNG DES XCE
401A  FB              0020        XCE               ;CPU AUF 8 BIT
00401B                0021          .ROB              ;ASSEMBLER-ANWEISUNG
00401B  4C 82 E1      0022          JMP $E182         ;COMIN BEIM AIM
00401E                0023
00401E                0024          *=$3002
003002  DC FE         0025          .WOR $FEDC        ;EIN PARAMETER
003004                0026          .END
```

hier ausgeschaltet. +DB heißt, daß das jeweilige Datenbankregister in der Phase Phi 1 des Taktes als Bankadresse ausgesandt wird. +PB entsprechend für das Programmbankregister. Die Zahl 24 bedeutet, daß die Datenbank entweder iin den Bytes des Befehles oder im Pointer definiert ist. 16 heißt zwangsweise Bank 0.

Das Programmbank-Register

Das vorstehende Schema enthält bereits auch Hinweise für die Beeinflussung der Programmbank. Sie wird durch folgende Befehle und Ereignisse beeinflußt/umgeladen: Interrupt, RTI, RTL, JML, JSL und JMP absolut lang.

Weiter zur Datenbank

In Beispiel 14 machen wir eine Probe auf's Exempel. Wir wollen feststellen, ob das Holen eines Parameters aus der Zelle $413002 etwa das Datenbankregister verändern würde, weil aus der Bank 0 heraus in die Bank 41 gegriffen wird. - Das ist nicht der Fall. Nach dem Programmlauf finden wir den Parameter FEDE in den Zellen 300/301 und in 302 eine 00 für die Datenbank, dieweil wir in Zeile 15 schon auf den 8 Bit-Modus zurückgeschaltet haben. Wenn wir das nicht tun, findet man auf der hiesigen Maschine 65802 eine hex F1 in Zelle 303, die wohl einen imaginären Wert bildet.

Die vielen Beispiele in diesem Aufsatz, wie auch hier bei Nr. 14, mögen den Leser ermuntern, an Tage 'X' viele ähnliche Versuche mit einer neuen CPU anzustellen und zu protokollieren. Nur so kann man sich Gewißheiten und Erfahrungen verschaffen, für die die Datenblätter im allgemeinen nicht ausreichen.

Man wird nach diesen ersten Versuchen sagen dürfen, daß das Datenbankregister nicht umgeladen wird, wenn man datenverarbeitende Befehle gibt, die entweder die anzusprechende Datenbank im Befehl oder bei der langen indirekten Indizierung mit (dl),y im Pointer enthalten, wie es auch aus Beispiel 15 spricht, in dem wir einen Parameter in der simulierten Bank 77 indirekt über den Pointer in $3002 ansprechen. Das zur Kontrolle zurückgeholte Datenbankregister bleibt bei 00; wir finden in den Zellen 300-302 die Bytes A0 00 00, letzteres für das Datenbankregister gemäß Zeilen 13-18. Oder: Der Kontext (Umgebung) der Datenbank wird nur dann umgeschaltet, wenn man sie durch einen über den Stack übergebenen Parameter mit Befehl PLB lädt, auch dieses zum Schutze der Umgebung des Programmierers.

Wir reiten über den Stack

Anders wie beim Ritt über den Bodensee lassen sich hier im Vorwege schon Gewißheiten mitteilen, es kommt nicht zum Einbruch und zum Versinken. Einige Seiten früher haben wir eine Addition mit Parametern durchgeführt, die relativ zum Stackpointer lagen. Im Beispiel 16 wollen wir möglichst viele neue Techniken vereinigen. Ein Unterprogramm AUSGAB der Ausgabe wird mit JSL (Jump Subroutine Long) in einer imaginären Bank 21 aufgerufen (Zeile 10). Es wird später in Zeile 30 mit RTL verlassen (Return from Subroutine Long).

```
*******************************************************************************
 BEISPIEL 15: PARAMETER AUS ANDERER BANK, INDIREKT
004008  A0 00 00      0009         LDY #0
0400B  B7 20          0010        LDA ($20,L),Y    ;POINTER MIT BANKADRESSE
0400D  8D 00 03       0011        STA $300         ;ABLAGE ZUR BEOBACHTUNG
010                 0012
004010  8B            0013         PHB             ;PUSH DATA BANK REGISTER
04011                 0014
004011  E2 30         0015         SEP #$30
004013                0016 ;HOLE CPU AUF 8 BIT ZURUECK
004013  68            0017         PLA             ;HOLE DBR
004014  8D 02 03      0018         STA $302
004017  38            0019         SEC             ;VORBEREITUNG DES XCE
04018  FB             0020        XCE             ;CPU AUF 8 BIT
004019                0021         .R08            ;ASSEMBLER-ANWEISUNG
004019  4C 82 E1      0022         JMP $E182       ;COMIN BEIM AIM
00401C                0023
00401C                0024         *=$20
000020  08 40         0025         .WOR $4008
000022  77           0026         .BYT $77         ;BANKADRESSE IM POINTER
------

*******************************************************************************
BESPIEL 16: VERRSCHIEDENE BEFEHLE
000000                0001         *=$20           ;IN DER ZEROPAG
000020  00 51         0002 POINTR .WOR $5100       ;ABLAGE PER ASSEMBLER
000022                0003
000022                0004         *=$4000
004000                0005         .R08            ;ALLES IN 8 BIT ERZEUGEN
004000  18            0006         CLC
004001  FB            0007         XCE             ;SET E=0 WITH CARRY=0
004002                0008
004002  D4 20         0009         PEI POINTR      ;POINTER AUF STACK
004004  22            0010         JSL AUSGAB      ;JUMP SUBROUTINE LONG
004005  00 50         0011         .WOR $5000      ;ADRESSE IN DER BANK
004007  15            0012         .BYT 21         ;BANKADRESSE DEZIMAL
00400B                0013 ;PROGRAMMFORTSETZUNG NACH JSL UND SCHLUSS
00400B  68            0014         PLA             ;POINTER VOM STACK
004009  68            0015         PLA             ;RAUEMEN
00400A  38            0016         SEC             ;VORBEREITUNG DES XCE
00400B  FB            0017         XCE             ;CPU AUF 8 BIT
00400C                0018         .R08            ;ASSEMBLER-ANWEISUNG
00400C  4C 82 E1      0019         JMP $E182       ;COMIN BEIM AIM
00400F                0020
00400F                0021         *=$5000
005000  5A            0022 AUSGAB PHY              ;RETTE Y
005001  A0 00         0023         LDY #0          ;ZUR INDIZIERUNG
005003  B3 05         0024 LOOP   LDA (5,S),Y      ;PER POINTER
005005  F0 07         0025         BEQ *+9         ;ZEICHEN IST 0
005007  20 BC E9      0026         JSR OUTALL      ;AKTIVE AUSGABE
00500A  C8            0027         INY             ;NAECHSTES ZEICHEN
00500B  4C 03 50      0028         JMP LOOP
00500E  7A            0029         PLY             ;RESTORE Y
00500F  6B            0030         RTL             ;RUECKKEHR
005010                0031         *=$5100
005100  44 41         0032         .BYT 'DAS IST DER 65816',$0 ;BEGRENZER
005111  00            0033
005112                0034 OUTALL =$E9BC           ;BEI AIM 65
005112                0035         .END
 ERRORS=0000
```

Wenn das auszugebende Zeichen hex 00 ist (Begrenzer), dann soll die Ausgabe abgebrochen werden. In Zeile 9 machen wir zunächst eine Parameterübergabe auf den Stack. Ehe wir JSL geben, kopieren wir mit PEI den Inhalt eines Pointers POINTR auf den Stack. Er liegt dort später für das Unterprogramm etwas versetzt, aber das stört uns noch nicht. Dem Pointer werden später nämlich noch eine Returnadresse in 3 Bytes und ein PHY in 1 Byte nachgeworfen, so daß der Stackpointer (der auf die nächst freie Adresse zeigt) um 5 gegen die auf dem Stack befindliche Pointeradresse versetzt liegt. Das wird in der stack-relativen Adressierung in Zeile 24 berücksichtigt..

```
  65XXX-ASSEMBLER COPYRIGHT 1984 BY ROLAND LöHR
FROM= PASS1
 PASS2=004017
*******************************************************************************
 BEISPIEL 17: BLOCK MOVE
000000                  0001          *=$4000
004000                  0002          .R16              ;ASSEMBLER-ANWEISUNG: ALLES 16
BIT
004000  18              0003          CLC
004001  FB              0004          XCE               ;SET E=0 WITH CARRY=0
004002  C2 30           0005          REP #$30          ;CPU SOLL 16 BIT FAHREN
004004  A9 00 02        0006          LDA #$200         ;2 PAGES VERSCHIEBEN
04007   A2 00 20       0007          LDX #$2000        ;QUELLADRESSE
00400A  A0 FF 1F        0008          LDY #$1FFF        ;ZIELADRESSE 1 DAVOR
00400D  54              0009          MVN $5544         ;BLOCK MOVE, INKREMENTIERE ADRE
SSEN
00400E  55              0010          .BYT $55          ;ZIELBANK IM OPERANDEN
00400F  44              0011          .BYT $44          ;QUELLBANK
004010  E2 30           0012          SEP #$30
004012                  0013 ;HOLE CPU AUF 8 BIT ZURUECK
004012  38              0014          SEC               ;VORBEREITUNG DES XCE
004013  FB              0015          XCE               ;CPU AUF 8 BIT
004014                  0016          .R08              ;ASSEMBLER-ANWEISUNG
004014  4C 82 E1        0017          JMP $E182         ;COMIN BEIM AIM
004017                  0018
004017                  0019          .END
```

Das Unterprogramm AUSGAB muß mit der stack-relativen (stack-indizierten) Adressierung (5,S),Y über den übergebenen Pointer adressieren und bis zum Begrenzer 00 ausgeben. Zur Erschwernis für den Programmierer verlangen wir, daß das Indexregister Y durch das Unterprogramm nicht 'verbogen' werden darf, es muß also im alten Zustand zurückgegeben werden. Wir stacken es daher entweder vor JSL oder innerhalb des Unterprogrammes und holen es entsprechend zurück. Hier machen wir das im Unterprogramm AUSGAB.

Mit den neuen Möglichkeiten der 65xxx ist das leicht getan. Alle Register werden in 8 Bit gefahren. was eigentlich nur für den Akku notwendig wäre. Beispiel 16 zeigt das. Es wurde hier ausgetestet und liefert tatsächlich den Text 'DAS IST DER 65816' auf den Bildschirm.

Block Move

Mit Nr. 17 wollen wir die Reihe der Beispiele zunächst abschließen. Da gibt es noch die beiden Befehle MVN und MVP die wie folgt englisch beschrieben sind: Move Block from Source (X-addressed) to Destination (Y-addressed), Block Length defined by C. Zu deutsch: Blockverschiebung, wobei das X-Register auf die Quelle und das Y-Register auf das Ziel zeigen muß. Die Zahl der zu bewegenden Bytes soll im 16 Bit-Akku C geladen sein. Bei MVN werden die beiden Indexregister inkrementiert (automatisch hochgezählt), bei MVP dekrementiert. Beide Befahlsarten werden benötigt, wie die Skizze der möglichen Überlappungen von Quell- und Zielbereich zeigt.

Zum Aufbau der Befehle selbst ist zu sagen, daß sie 3 Byte Gesamtlänge haben. Auf den Opcode folgen die Parameter Zielbank und dann Quellbank. Das Datenbankregister wird dabei mit der Zielbank geladen.

Beispiel 17 führt wie erwartet aus, so daß es als Vorlage für Programmierungen dienen mag.

Zusammenfassung

Die Architektur und Leistungsfähigkeit der neuen Chips stellt in der 65xxx-Familie einen deutlichen Fortschritt dar. Nicht nur daß OXI-CMOS geringeren Energieverbrauch gegenüber NMOS-Prozessoren hat und damit den Bau von Platinen erleichtert, die in der Versorgungsspannung gepuffert sind. Die Datenblätter für den 65816 enthalten auch Angaben für Takte von 2, 4, 8 und sogar 10 MHz. Die beiden letzteren Geschwindigkeiten muß man dabei wohl als ein Ziel ansehen, das im Verlaufe erreicht werden soll. Aber auch mit 4 MHz schon sind deutliche Fortschritte in der Geschwindigkeit erreicht.

Abgesehen von Echtzeitanwendungen ist die Geschwindigkeit in den Augen des Autors gar nicht einmal ein so wichtiges Argument. Für den Systemprogrammierer, den diese Zeitschrift u.a. anspricht, ist die Frage der Anpassung an die neuen Möglichkeiten wesentlich. Hier darf man sagen, daß die CPU's 65802 und 65816 Möglichkeiten einräumen, die wir beim 6502 und seinen bisherigen Abkömmlingen oft schmerzlich vermißt haben, auch bei Systemen mit 6509 (im CBM 720) und auch noch bei 65C02. Es hat im Grunde genommen Freude gebracht, die wichtigsten Merkmale in diesem Artikel zu erkunden und mit Beispielen zu belegen. - Für vorhandene Systeme, wird allein wegen des Programmierkomforts im Verlaufe eine entsprechende Umrüstung abzuwägen sein.

Auf der ELECTRONICA gab es 'vorbeugend' abwertende Hinweise des Wettbewerbes, daß viele Zyklen ja länger als bei 6502 seien. Nicht erwähnt wird dabei, daß Befehle, die sich auf 16 Bit auswirken, natürlich mehr Zeit als solche für 8 Bit brauchen. Sie schaffen die doppelte Arbeit und sparen oft einen bei 6502 sonst notwendigen Folgebefehl mit seiner Zeit, der das zweite Byte bearbeitet. Oder man denke an die bei 6502 bisher mühsamen Parameterübergaben mit ihren langen Befehlsfolgen, die jetzt durch PEA und PEI abgelöst werden, wozu die stack-relativen Adressierungen kommen. Die Hinweise ließen sich für viele andere häufig gebrauchte Programmiertechniken mit positiver Erwähnung des Zeitgewinns fortsetzen.

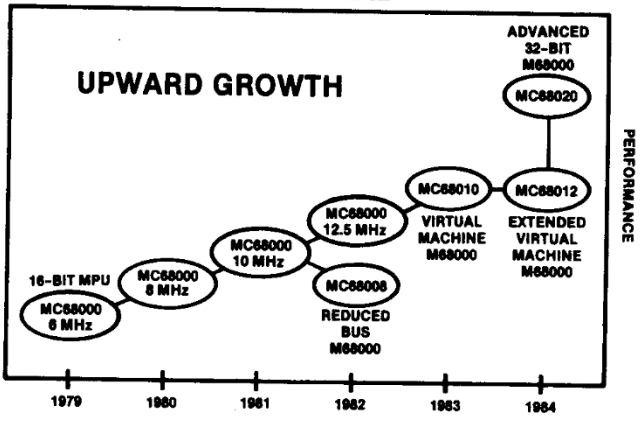
Beim 65816 ist darauf hinzuweisen, daß er einen Adreßraum von 16 Megybyte öffnet. Künftige Systeme werden von vielen Speicherbänken Gebrauch machen, in wenigen Jahren sind nicht nur bei 65xx Arbeitsspeicher von 1 MB ganz normal und auch erforderlich. Die 65816 wird mit ihren neuen Adressierungsmöglichkeiten dabei auch ein großes Video-RAM einfach ansprechen und verwalten können.

Roland Löhr

Der MC 68020

Unter dem Slogan "The 32-Bit Performance Standard" stellte Motorola in den vergangenen Wochen seinen neuesten Mikroprozessor in Tagesseminaren und auf der ELECTRONICA dem Fachpublikum vor. Nachdem der MC68000 mit dem 16 Bit breiten Datenbus und auch der MC68008 (Datenbus auf 8 Bit reduziert) jetzt zunehmende Verbreitung finden, sollen hier vor allem die neuen Merkmale des MC68020 erwähnt werden. Der Autor verweist in diesem Zusammenhang auf die Besprechung des 68000 in Heft 30 dieser Zeitschrift und geht davon aus, daß der interesssierte Leser entweder schon ein Datenblatt für einen der Prozessoren besitzt oder es sich beschaffen wird. Eine ausführliche Übersicht findet er in dem neuen Buch "MC68020 32-Bit Microprocessor User's Manual" (ISBN 0-13-541418-0, Motorola Document MC68020UM-ADI).

Kurz zur Übersicht: Die CPU-Familie hat inzwischen viele Mitglieder. Der Arbeitstakt der 68000 wurde inzwischen bis auf 12,5 MHz gesteigert, wobei der Typ mit 8 MHz derzeit wohl am häufigsten eingesetzt wird. Im 68010 und 68012 ist zusätzlich das Konzept der virtuellen Maschine verwirklicht. Das wird weiter unten dargestellt. Diese Zentraleinheiten haben durch gehend einen Datenbus mit 16 Pins und können 16 MB Speicher adressieren. Der 68008 ist eine Version, die den sparsameren Aufbau eines Computers zuläßt: Der Datenbus ist auf 8 Bit reduziert und die Adressierung auf 1 MB beschränkt (Gehäuse mit 48 Pins). Diese Typen haben einen gemeinsamen Befehlssatz mit sehr leistungsfähigen Adressierungsarten. In ihnen ist das Handshaking der angesprochenen Speicher und sonstigen Bausteine mit dem Signal DTACK verwirklicht, das einen asynchronen Betrieb mit verschieden schnellen Einheiten ermöglicht. Diese müssen also nicht 'synchron' innerhalb eines starren Maschinentaktes reagieren.



Virtuelle Maschinen

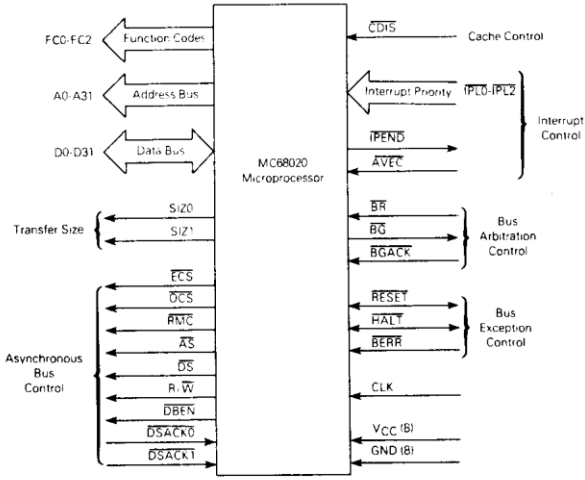
Ab Typ 68010 ist das Konzept der virtuellen Maschine verwirklicht. Das bedeutet: Die CPU kann den gesamten Adreßraum logisch richtig ansprechen und verwerten, obwohl nur ein kleinerer Teil des Speichers tatsächlich bestückt ist. Solche Ansprachen führen zu einem 'Bus Error' und damit zum Trap bzw. zur Exception Nr. 2 in Adresse 8. Man nennt einen solchen Fehler auch 'page fault'. Man kann nun per Software auf eine Fehlerbehandlungsroutine vektorieren und nicht vorhandene Daten vom Massenspeicher in den wirklich vorhandenen Speicher holen und bearbeiten. Mit dem Befehl RTE (Return from Exception) erfolgt dann die Programmfortsetzung im unterbrochenen Befehl. Diese Art Trap ist auch nützlich, wenn auf einem Entwicklungssystem für eine Maschine programmiert wird, die noch gar nicht existiert und die Ein- und Ausgabeeinheiten enthalten soll, die ebenfalls noch nicht auf dem Entwicklungssystem vorhanden sind. In diesem Falle kann man die E/A per Trap emulieren.

Zum Konzept der virtuellen Maschine gehört auch der wahlweise Anschluß einer Memory Management Unit (MMU oder PMMU68851), die logische Adressen in physisch vorhandene umsetzt und entsprechend verwaltet. Die CPU 68020 ist innerhalb der Familie die erste Ausführung mit 32 Bit Adreßbus (Adressierung von logisch 4 Gigabyte) und 32 Bit breitem Datenbus. Die für Mitte 1985 vorgesehene PMMU68851 verwaltet dementsprechend auch Adressen von 32 Bit.

Hardware und neue Signale am 68020

Der Prozessor ist in einem quadratischen Gehäuse mit der Kantenlänge von 3,4 cm untergebracht. Das Gehäuse wird Pin Grid Array genannt, es hat 114 Pins.

Die derzeitigen Bemusterungen haben eine Taktfrequenz von 12,5 MHz. Die reguläre Produktion des Typs mit 12,5 MHz soll im 2. Quartal 1985 beginnen. Der Typ mit 16,7 MHz soll im 2. Halbjahr 1985 produziert werden. Und Ende 1985 will man eine Taktfrequenz von 20 MHz erreichen. - Für die Version 12.5 MHz findet man in den Seminarunterlagen einen Preis von $487 für ein Sample, genannt wurden auch etwa DM 2000 pro Stück, wobei nicht ganz klar wurde, welche Version damit gemeint sei. - Immerhin betont Motorola, daß die Version mit 16,7 MHz eine Leistungssteigerung um den Faktor 2,5 bedeutet gegenüber einem 68000 mit 12,5 MHz. Daß man sich auch eine Überlegenheit gegenüber Intel IPAX286 mit 8 MHz und gegenüber NS32032 mit 10 MHz ausrechnet, gehört zum legalen Handwerkszeug. Auf die jedenfalls große Leistungsfähigkeit kommen wir noch zu sprechen.



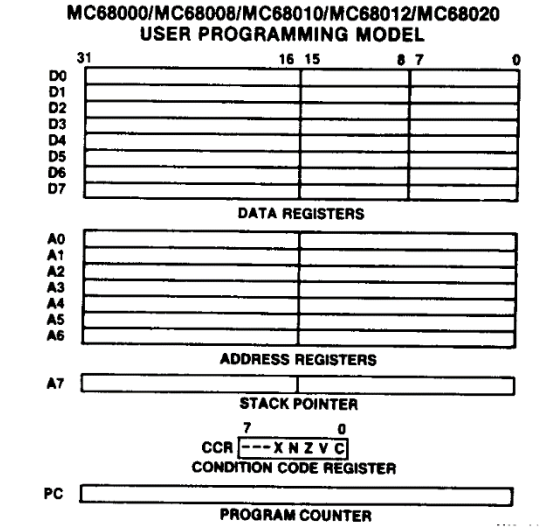
Functional Signal Groups

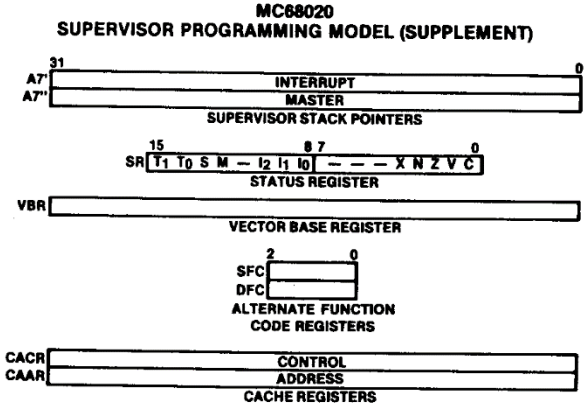
Die nebenstehende Abbildung beschreibt die Signale und Signalgruppen des 68020. Die verbreiterten Daten- und Adreßbusse wurden schon erwähnt. Der 68020 ist mit Ihnen sowohl intern (Register) als auch extern eine echte 32-Bit-Maschine, wobei das Prinzip beibehalten wurde, die Busse nicht zu multiplexen und Peripheriebausteine wie Speicherstellen ansprechen zu können, sie sind 'memory mapped'. Neu sind die beiden DSACKx-Signale, die das DTACK ablösen. Mit ihnen kann der Anwender von außen her dynamisch die Busbreite beinflussen (8, 16 oder 32 Bit). Demgegenüber melden die SIZx-Signale die Zahl der im Zyklus noch zu übertragenden Bytes nach außen. Die Signale ECS, OCS und RMC dienen der Bus-Steuerung und -Sicherung bei Read/modify/Write. Erwähnenswert sind ferner das Eingangssignal CDIS, das den Cache-Speicher (s.u.) abschaltet und das AVEC, das die CPU in einem Interrupt-Zyklus auffordert, einen Auto-Vektor zu berechnen (das Interrupt-System wurde ebenfalls in Heft 30 erklärt).

Der Cache-Speicher
(Sprich: Käsch). Die Analyse vieler Programme zeigt, daß zusammengehörige Befehlsfolgen z.B. in Schleifen immer wieder durchlaufen werden. Normalerweise ist das mit Buszyklen verbunden, in denen die gleichen Maschinenbefehle immer wieder in die CPU geholt werden. Der Cache-Speicher (cache bedeutet 'verborgen') auf der CPU speichert nun Befehlsfolgen bis 256 Byte zwischen. Wenn ein Befehl schon in der CPU ist, dann braucht er nicht vom Bus geholt zu werden. Die Busbenutzung von etwa 90% beim 68000 soll damit je nach Programm auf 50-60% reduziert werden. Natürlich hat das auch positiven Einfluß auf die Geschwindigkeit. Hinzu kommt eine dreistufige Befehls-Pipeline auf der CPU, mit der vom Bus oder vom Cache kommende Befehle aufbereitet werden. Mit dem Signal CDIS kann man z.B. in einem Interrupt den Cache 'einfrieren', wenn man dadurch zeitliche Vorteile erzielen kann und will. Befehle im Cache kann man ferner durch die Register CACR und CAAR (Control und Address Register, siehe Registermodell) per Software manipulieren. Mit dem CACR kann man den Cache ein- und ausschalten, ihn einfrieren, Einträge löschen oder den Cache ganz löschen, wenn ein Context geändert wird.

Das Registermodell

Die je sieben Adreß und Datenregister für den Benutzer und der Programmzähler (je 32 Bit Breit) sowie das Condition Code Register sind gleich geblieben. Für die Systemüberwachung wurden die dem Cache dienenden neuen Register bereits erwähnt. Neu ist die Aufteilung des Supervisor-Stacks in einen solchen für Interrupte und einen Master-Stack. Wird im Status das neue M-Bit gesetzt, so landen Interrupte auf dem Interrupt Stackpointer, während alle anderen Exceptions den Master Stack berühren, wo z.B. ein Task Control Block ungestört geführt wird. Im Status findet man ein neues Trace Control Bit. Es erlaubt den Trace abzuschalten, außer bei Branches, Jumps, JSR's und Return. Damit können fehlerfreie Unterprogramme z.B. im 'Schnellgang' durchfahren werden.





Das Vektor Basisregister (VBR) bildet einen aktualisierbaren Zeiger auf den Speicher. Beim 68000 liegen bekanntlich 256 Exception Vektoren im Adreßbereich von hex 0-3FF. Mit dem VBR kann man jetzt nach Bedarf auf alternierende Bereiche zeigen. Die Bedeutung der zusätzlichen Register DFC und SFC (Destination und Source Function Code) ist nicht so ganz klar. Sie geben die Signalkombination an den Pins FC0-FC3 wieder und haben eine Bedeutung bei der neuen Instruktion MOVES: Bringe oder hole einen Operanden aus dem oder in den Adreßraum, der durch DFC oder SFC beschrieben ist. Es handelt sich um einen privilegierten Befehl des Supervisors.

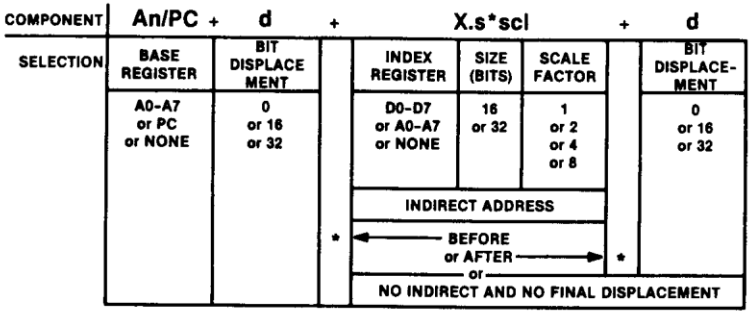
Neue Datentypen und Befehle
Außer Bits, Bytes, Words und Longwords gibt es jetzt neu die Bitfelder, für die ein Arsenal von Befehlen bereitgestellt wurde, den Datentyp BCD-Zahl mit den Befehlen PACK und UNPCK (Verdichtung eines ASCII-Zahlenstrings auf BCD-Darstellung und umgekehrt entsprechende Erweiterung, Codewandlung), sowie den Datentyp Quad Word (64 Bit). Dieser dient der Produktablage von z.B. 32x32 Bit Multiplikation und umgekehrt der Division. Bemerkenswert ist dabei, daß benutzte Register nicht beisammen liegen müssen.

Bitfelder dürfen eine Variable Länge von bis zu 32 bits haben und müssen nicht an Byte-Grenzen ausgerichtet sein. Die Befehle speziell für sie sind: komplementiere Feld, teste und lösche, extrahiere mit oder ohne Vorzeichen, finde erstes Bit mit '1' im Feld, füge in ein Feld ein, setze alle Bits, teste Bitfeld und setze entsprechend den Status. Bei der Adressierung ist die Ansprache eines Bitfeldes einfach: Basisadresse + Offset + Menge. Der Offset kann ein Direktwert von 0-31 sein oder er kann in einem Datenregister stehen und dann plus/minus 2 Gigabyte sein, den ganzen Adreßraum überspannend. Die Menge von 1-32 betrifft die Zahl der betroffenen Bits. - Diese Befehle und ihre Adressierungen erlauben das Führen großer Wahrheitstabellen oder die Bearbeitung komprimierten oder wie immer manipulierten Codes, der in Form von Records abgelegt sein mag. - Ein sog. Barrel Shifter (=Trommel) beschleunigt zudem die Verschiebebefehle, deren Dauer nicht mehr von der Zahl der zu verschiebenden Stellen abhängig ist.

Neue Befehle sind auch CHK2 prüfe den Operanden in einem Register zugleich gegen einen unteren und einen oberen Grenzwert. Bei 'out of bounds' wird eine Exception eingeleitet. CMP2 tut das Gleiche ohne Trap, setzt also nur den Status. Der Befehl Trap on Condition spricht für sich. Das Paar CALLM und RTM bedeutet Call Module bzw. Return davon. Der gegenwärtige zustand eines Modules wird gestackt. Ein Direktoperand spezifiziert die Zahl der zu übergebenden Bytes, die effektive Adresse zeigt auf einen Deskriptor für ein zu ladendes Modul. Diese Befehle vereinfachen die Schaffung von 'Frames' (Rahmen) auf dem Stack für lokale Variablen zu Unterprogrammen. Schließlich sind CAS und CAS2 zu nennen (Compare and Swap with operand), die der Systemprogrammierung zur Absicherung dienen. - Weitere Befehle, die der 'F-Line' (bisher nicht implementiert, mit hex F beginnend), dienen der Ansteuerung von Co-Prozessoren.

Neue Adressierungsarten
In den Seminarunterlagen werden 51 neue Adressierungsformen genannt. Es würde zu weit führen, sie hier alle vorzustellen. Adreßdistanzen sind nach 8 und 16 Bit jetzt auch in 32 Bit anzugeben, so daß der ganze logische Speicherraum überstrichen werden kann. Ausgebaut wurden die indirekten Adressierungsarten. Sie konnten auch bisher schon mit einem der Register A0-A7 und D0-D7 indiziert werden. Neu ist die Möglichkeit der Skalierung. Das Indexregister kann vor der Addition zu einer zwischenzeitlich gebildeten Adresse mit 0, 2, 4 oder 8

68020 Memory Indirect Addressing



<ea> = ([d, An, Xn.s*scl], d)

multipliziert werden, was z.B. die Bearbeitung von Datenfeldern erleichtert. Neu eingeführt wurde ferner die indirekte Adressierung über das Memory. Hier ist eine Nach-Indizierung möglich, ähnlich wie wir sie beim 6502 mit (Pointer),Y kennen: Ein Adreßregister nebst Displacement (oder Offset) zeigt auf ein Langwort im Speicher. Dieser enthält eine indirekte Basisadresse. Hierzu wird ein ggfs. skalierter Index (inhalt eines Registers) addiert und möglicherweise noch ein 'äußerer' Offset. Die summe ist dann die effektive Adresse.

Bei der memory-indirekten Adressierung ist aber auch eine Vor-Indizierung möglich: Adreßregister + Offset + ggfs. skalierter Index weisen auf ein Adreßlangwort im Speicher. Dieses bildet eine Basisadresse. Hierzu kann ggfs. ein äußerer Offset addiert werden, um die effektive Adresse zu ergeben. Es ist zu betonen, daß statt eines Adreßregisters auch der Programmzähler als Basisadresse herangezogen werden kann. Damit kann man Daten unabhängig von der Ladeadresse eines Modules bearbeiten. - Die erste Art der indirekten indizierung via Basisadresse im Memory wird man benutzen, um Datenfelder zu überstreichen. Die Vor-Indizierung benutzt man, wenn es im Memory ein Feld mit Pointern liegt. In der Summe bedeuten diese Adressierungsarten, daß die 68020 nun beliebig viele Hilfsregister im Speicher haben kann, so wie der 6502 es in der Zero Page erlaubt. Natürlich lassen sich diese neuen Möglichkeiten auf der 68000 durch Laden von Adreßworten aus dem Memory mit mehr Code nachbilden. - Erwähnenswert ist noch, daß Datenwörter (im Gegensatz zu Instruktionen) nicht an Wortgrenzen ausgerichtet sein müssen (bei 68000: Address Error).

Zusammenfassung

Die Erweiterungen der 68020 gegenüber den früheren Familienmitgliedern sind beachtlich, nicht nur wegen des schnelleren Taktes. Leistungssteigernde Merkmale sind durch den Cache-Speicher für Befehle bedingt, durch das Instruction-Pipelining über drei Stufen, durch mögliche Parallelverarbeitungen und durch den Barrel Shifter. Soweit zur Hardware. Leistungssteigernd wirken auch die Befehle zur Codewandlung PACK und UNPK auf/von BCD, auf 64 Bit wirkende Befehle bei Multiplikation und Division und die Adressierung und Bearbeitung von Bitfeldern. Hinzu kommen die Anschlußmöglichkeiten für Co-Prozessoren für Fließkomma-Arithmetik und Bedienung virtueller Speicheradressen (MMU) und weiterer Peripherie

Wenn auch der einzelne Anwender sicher nicht alle guten Merkmale benötigt, so darf sich Motorola derzeit wohl mit Recht einen Vorsprung gegenüber dem

Wettbewerb ausrechnen. Bemerkenswert ist immerhin auch der ungeteilte Adreßraum, der kontinuierliche Adressen ohne Banking enthält. Und die Busse sind nicht gemultiplext.

Das erste Seminar für 68000 fand hierzulande im Dezember 1979 in Unterföhring statt. Damals hatte es etwa 10 Teilnehmer. Heute kommen 60 und über 200 Teilnehmer zu den Übersichtsvorträgen von Motorola. Und man muß davon ausgehen, daß diese nicht Personal Computer entwerfen wollen, wie etwa LISA oder McIntosh, sondern daß es ihnen vor allem um schnelle Steuerungen geht. Und hier scheint die Prozessorfamilie, wie die Teilnehmerzahlen verraten, große Akzeptanz zu finden. Man muß sich jedoch vor Augen halten, daß es nach der ersten Ankündigung des 68000 Jahre dauerte, bis fertige Computer mit Betriebssystemen und Anwenderprogrammen auf den Markt kamen. Man geht wohl nicht Fehl in der Annahme, daß der 68020 (oder wie kalifornische Vortragende sich in ihrer Mundart ausdrücken: 'The sixtyeight-o-twinny') wieder einmal eine Herausforderung an die Phantasie der System-Designer und der Programmierer darstellt, die neue Horizonte fordert, um die gegebenen Möglichkeiten sinnvoll und in Hinsicht auf die 'Performance' günstig zu nutzen. Und das wird über die notwendigen Erfahrungs- und Erkenntnisstufen sicher auch einmal wieder drei Jahre dauern. Erwarten darf man aber etwa für diese Zeit, daß uns dann Computer mit einer enormen Leistungsfähigkeit und zu akzeptablen Preisen mit 68020 zur Verfügung stehen, von denen wir uns heute sicher noch kein zuverlässiges Bild in der Vorausschau machen können. - Ziemlich sicher bleibt wohl die Vorhersage, daß die Computer unser Arbeitsleben weiterhin verändern werden, auch mit Effekten für den Arbeitsmarkt. Große und preiswerte Computerspeicher und Massenspeicher (Laser-Disk) sind im Zusammenhang mit der Leistungsfähigkeit der CPU Voraussetzung nicht nur für Informationssysteme und assoziative Speicher, sondern auch für künstliche Intelligenz, amerikanisch AI und deutsch inzwischen KI. - Der Ritt über den Bodensee wird neu bedacht werden müssen.

Martin Albrecht, 4350 Recklinghausen

Druckermenue für 6502

Moderne Drucker, insbesondere Matrixdrucker, verfügen über eine Vielfalt von Einstellmöglichkeiten. Die wichtigsten Grundeinstellungen können meist über DIP-Schalter vorgenommen werden. Diese 'Mäuseklaviere' sind umständlich zu bedienen, und zudem ist das Gros der Funktionen nur programmgesteuert anwählbar. Viele ältere Systeme verfügen jedoch nicht über komfortable Software zur Nutzung aller Druckerfunktionen. Selbst einfache Voreinstellungen des Randes, der Schriftart, des Zeilenabstandes usw. sind kaum möglich.

Abhilfe schafft das vorliegende Druckermenue. Es ist praktisch eine Erweiterung der DIP-Schalter mit Mitteln der Software. Nach dem Programmstart befinden Sie sich in einem Menue. Mit den Tasten T (=Top), B (=Bottom), U (=Up) und D (=Down) bewegen Sie sich in einer Liste der Einstellungen. Alle Möglichkeiten werden im Klartext angezeigt. Durch Drücken von RETURN werden die Steuerzeichen der zuletzt angewählten Funktion zum Drucker übertragen. Einige Einstellungen, z.B. Randeinstellung, benötigen zusätzlich einen Parameter. Er wird nach der Auswahl der Funktion mit RETURN angefordert. Vorgesehen ist die Eingabe von Dezimalzahlen zwischen 0 und 255. Bei darüber liegenden Werten muß die Eingabe wiederholt werden. Drücken einer anderen Taste als 0 ... 9 beendet das Einlesen. Wenn Sie keine Zahl eingeben, wird der Parameter automatisch auf Null gesetzt. Durch die Taste Q (=Quit) wird das Menue wieder verlassen.

Die eigentlichen Menueeinträge sind in einer Tabelle ab Label TEXTS zusammengefaßt. Beim

Aufbau eigener Einträge ist die genaue Syntax zu beachten. Jede Meldung besteht aus zwei Teilen: einem Displaytext und einem Steuerteil. Der Eintrag beginnt mit dem Displaytext. Er wird durch ein Komma abgeschlossen. Ist die Eingabe eines Parameters erforderlich, so erfolgt die Begrenzung durch einen Punkt. Ein Beispiel:

```
.BYT 'Eliteschrift,',ESC,'M;'                ohne Parameter
.BYT 'linker Rand=.',ESC,'l.;'               mit Parameter
```

Der Punkt im Displayteil veranlaßt die Parametereingabe. Nach dem Komma oder Punkt folgen die Steuerzeichen. Dieser Teil wird durch ein Semikolon begrenzt. Hier tritt der Punkt an der Stelle wieder auf, wo der aktuelle Parameter wieder übergeben werden soll. Der Punkt dient nur als Platzhalter (siehe Beispiel 'linker Rand'). Der Displaytext und die Steuersequenz dürfen beliebig lang sein. Alle Zeichen bis auf '', '', '.' und ';' können benutzt werden.

Das Menue kann leicht für andere Drucker angepaßt werden. Die Anzahl und Reihenfolge der Menuepositionen hängt nur von Ihren Vorstellungen ab. Beim Aufbau eines Menues ist auf die Markierungen $FF und die Label TEXTS und TEXTE am Textbedinn und Textende zu achten.

```
------------------------------------------------------------
Druckervoreinstellung über ein Menü

0000          ; (c) Martin Albrecht, 11.11.1984
0000          ;in der vorliegenden Version für AIM-65 mit EPSON FX-80
0000          ;Anpassung an andere Rechner und Drucker leicht möglich
0000
------------------------------------------------------------
Konstanten

0000 CR               =$D             ;Carriage Return
0000 ESC              =$1B            ;Escape
0000
0000 KOMMA            =$2C            ;Begrenzer
0000 PUNKT            =$2E            ;Platzhalter
0000 SEMI             =$3B            ;Begrenzer
0000
------------------------------------------------------------
Programmvariablen

0000          ;Zeropage nur für ZEIGER erforderlich
0000 ZEIGER           =$F0            ;Zeiger in Menütext
0000 INWERT           =$F2            ;Zwischenspeicher für Eingabewert
0000
------------------------------------------------------------
Rechnerspezifische Unterprogramme

0000          ;je nach Rechner anpassen
0000 CENTRO           =$C000          ;Centronics Ausgabe
0000 CINIT            =$C04A          ;Centronics Schnittst. initial.
0000 CRLF             =$EA13          ;CR LF auf Display ausgeben
0000 INPUT            =$E93C          ;ein Zeichen von der Tastatur einlesen
0000 OUTPUT           =$E97A          ;ein Zeichen auf Display ausgeben
0000
0000
0000                  *=$3000
3000
------------------------------------------------------------
Start des Menüprogramms

3000 MENU   204AC0 JSR CINIT        ;Centronics initialisieren
3003 MENU1  A925   LDA #<TEXTS      ;Textbeginn in ZEIGER
3005        A231   LDX #>TEXTS
3007        85F0   STA ZEIGER
3009        86F1   STX ZEIGER+1
300B
------------------------------------------------------------
Hauptbefehlsschleife

300B MENU2  206730 JSR DISMES       ;erste Meldung ausgeben
300E        201430 JSR BEFEHL       ;Befehlsschleife
3011        4C0B30 JMP MENU2
3014
```

```
-------------------------------------------------------------
Kommandoeingabe und Ausführung

3014 BEFEHL 08      PHP             ;C retten, C=1 <=> Eingabe
3015 BEF1   203CE9  JSR INPUT       ;lies Tastatur
3018        20E130  JSR CAPCHR      ;Klein- zu Großbuchstaben
-------------------------------------------------------------
T-Top: an den Menüanfang

301B        C954    CMP #'T'        ;'T'-Top ?
301D        D006    BNE BOTTOM      ;nein
301F        28      PLP             ;Status vom Stack
3020        68      PLA             ;Returnadresse vom Stack
3021        68      PLA
3022        4C0330  JMP MENU1       ;auf Menüanfang
-------------------------------------------------------------
B-Bottom: an das Menüende

3025 BOTTOM C942    CMP #'B'        ;'B'-Bottom ?
3027        D011    BNE UP          ;nein
3029        28      PLP             ;Status vom Stack
302A        A904    LDA #<TEXTE     ;ZEIGER auf Textende
302C        A232    LDX #>TEXTE
302E        85F0    STA ZEIGER
3030        86F1    STX ZEIGER+1
3032        20B630  JSR SCANB       ;ZEIGER auf letzten Eintrag

3035        68      PLA             ;Returnadresse vom Stack
3036        68      PLA
3037        4C0B30  JMP MENU2       ;weiter mit MENU2
-------------------------------------------------------------
U-Up: im Menü nach oben

303A UP     C955    CMP #'U'        ;'U'-Up ?
303C        D004    BNE DOWN        ;nein
303E        28      PLP             ;Status vom Stack
303F        4CB630  JMP SCANB       ;eine Position zurück im Text
-------------------------------------------------------------
D-Down: im Menü nach unten

3042 DOWN   C944    CMP #'D'        ;'D'-Down ?
3044        D004    BNE RETURN      ;nein
3046        28      PLP             ;Status vom Stack
3047        4CA030  JMP SCANF       ;eine Position weiter im Text
-------------------------------------------------------------
RETURN: Menüeintrag ausführen, im Menü nach unten

304A RETURN C90D    CMP #CR         ;'CR'-Ausführen
304C        D011    BNE QUIT        ;nein
304E        28      PLP             ;Status, C=1 <=> Eingabe
304F        9003    BCC RET1        ;keine Eingabe
3051        20E830  JSR NUM         ;Eingabe holen
3054 RET1   20B330  JSR CENMES      ;Steuerzeichen an Centronics
3057        B1F0    LDA (ZEIGER),Y
3059        1003    BPL RET2        ;Negativ wenn Textende
305B        20B630  JSR SCANB       ;eine Position zurück im Text
305E RET2   60      RTS
-------------------------------------------------------------
Q-Quit: Programm verlassen

305F QUIT   C951    CMP #'Q'        ;'Q'-Quit ?
3061        D0B2    BNE BEF1        ;nein
3063        28      PLP             ;Status vom Stack
3064        68      PLA             ;Returnadresse vom Stack
3065        68      PLA
3066        60      RTS             ;Programm beenden
3067
-------------------------------------------------------------
Displaymeldung ausgeben

3067 DISMES 2013EA  JSR CRLF        ;CR, LF
306A DISM1  A000    LDY #0
306C        B1F0    LDA (ZEIGER),Y  ;Zeichen holen
306E        C92E    CMP #PUNKT      ;Punkt, dann Eingabe
3070        D002    BNE DISM2       ;teste auf Komma
3072        38      SEC             ;C=1 <=> Eingabe
3073        60      RTS
3074 DISM2  C92C    CMP #KOMMA      ;teste Komma
3076        D002    BNE DISM3       ;noch nicht, Zeichen ausgeben
```

```
3078          18      CLC               ;C=0 <=> keine Eingabe
3079          60      RTS
307A DISM3    207AE9  JSR OUTPUT        ;Zeichen ausgeben
307D          20CF30  JSR INCZEI        ;Textzeiger erhöhen
3080          4C6A30  JMP DISM1         ;Schleife
3083
-------------------------------------------------------------
Steuerzeichen an Centronics Schnittstelle

3083 CENMES 20CF30 JSR INCZEI       ;ZEIGER erhöhen
3086        A000   LDY #0
3088        B1F0   LDA (ZEIGER),Y   ;Zeichen holen
308A        C92E   CMP #PUNKT       ;Punkt ?
308C        D005   BNE CENM1        ;nein
308E        A5F2   LDA INWERT       ;ja, hole Eingabewert
3090        4C9A30 JMP CENM2        ;auf Centronics ausgeben
3093 CENM1  C93B   CMP #SEMI        ;Textende ?
3095        D003   BNE CENM2        ;nein, Zeichen ausgeben
3097        4CCF30 JMP INCZEI       ;ZEIGER erhöhen
309A CENM2  2000C0 JSR CENTRO       ;Zeichen auf Centronics ausgeben
309D        4C8330 JMP CENMES       ;Schleife
30A0
-------------------------------------------------------------
im Text bis hinter das nächste ";" lesen

30A0 SCANF  20CF30 JSR INCZEI       ;ZEIGER erhöhen
30A3        A000   LDY #0
30A5        B1F0   LDA (ZEIGER),Y   ;Zeichen lesen
30A7        C93B   CMP #SEMI        ;bis Semikolon
30A9        D0F5   BNE SCANF
30AB        20CF30 JSR INCZEI       ;ein Zeichen weiter
30AE        B1F0   LDA (ZEIGER),Y
30B0        1003   BPL SCANF1       ;negativ, wenn Textende

30B2        20B630 JSR SCANB
30B5 SCANF1 60     RTS
30B6
-------------------------------------------------------------
im Text einen Eintrag zurück

30B6 SCANB  A000   LDY #0
30B8        A200   LDX #0
30BA SCANB1 20D630 JSR DECZEI       ;Zeiger erniedrigen
30BD        B1F0   LDA (ZEIGER),Y   ;Zeichen holen
30BF        C9FF   CMP #$FF         ;Textbeginn, dann nicht weiter
30C1        F009   BEQ SCANB2
30C3        C93B   CMP #SEMI        ;Semikolon ?
30C5        D0F3   BNE SCANB1       ;bis Semikolon
30C7        E8     INX              ;X + 1
30C8        E002   CPX #2           ;zweites Semikolon
30CA        D0EE   BNE SCANB1       ;nein
30CC SCANB2 4CCF30 JMP INCZEI       ;ja, ein Zeichen weiter
30CF
-------------------------------------------------------------
ZEIGER + 1

30CF INCZEI E6F0   INC ZEIGER       ;ZEIGER + 1
30D1        D002   BNE INL
30D3        E6F1   INC ZEIGER+1
30D5 INL    60     RTS
30D6
-------------------------------------------------------------
ZEIGER - 1

30D6 DECZEI C6F0   DEC ZEIGER       ;ZEIGER - 1
30D8        A5F0   LDA ZEIGER
30DA        C9FF   CMP #$FF
30DC        D002   BNE DECZ1
30DE        C6F1   DEC ZEIGER+1
30E0 DECZ1  60     RTS
30E1
-------------------------------------------------------------
Kleinbuchstaben zu Großbuchstaben

30E1 CAPCHR C941   CMP #$41
30E3        3002   BMI CAP1
30E5        29DF   AND #$DF
30E7 CAP1   60     RTS
30E8
```

```
---------------------------------------------------------------
Taastatureingabe holen (Dezimal zwischen 0 und 255)

30E8          ;Abbruch, falls keine Zahl eingegeben wurde
30E8          ;Wiederholung, falls Eingabe > 255
30E8 NUM      A900    LDA #0
30EA          85F2    STA INWERT      ;Startwert = 0
30EC NUM1     203CE9  JSR INPUT       ;Zeichen einlesen
30EF          C930    CMP #$30        ;auf Zahl testen: >=$30, <$3A
30F1          9023    BCC NUM2        ;keine Zahl, Ende
30F3          C93A    CMP #$3A
30F5          B01F    BCS NUM2        ;keine Zahl, Ende
30F7          207AE9  JSR OUTPUT      ;Zahl ausgeben
30FA          290F    AND #$0F        ;Bits 0 bis 3 isolieren
30FC          48      PHA             ;und retten
30FD          A5F2    LDA INWERT
30FF          0A      ASL A           ;ACCU = INWERT * 4
3100          0A      ASL A
3101          B014    BCS NUMERP      ;>255
3103          65F2    ADC INWERT      ;ACCU = INWERT * 5
3105          B010    BCS NUMERP      ;>255
3107          0A      ASL A           ;ACCU = INWERT * 10
3108          B00D    BCS NUMERP      ;>255
310A          85F2    STA INWERT
310C          68      PLA
310D          65F2    ADC INWERT      ;eingelesene Zahl addieren
310F          B007    BCS NUMERR      ;>255
3111          85F2    STA INWERT      ;in INWERT
3113          4CEC30  JMP NUM1
3116 NUM2     60      RTS
3117
3117 NUMERP   68      PLA
3118 NUMERR   20A030  JSR SCANF       ;>255: Meldung wiederholen
311B          20B630  JSR SCANB
311E          206730  JSR DISMES
3121          4CE830  JMP NUM         ;Eingabe anfordern
3124
---------------------------------------------------------------
Tabelle der Displaytexte und Druckersteuerzeichen

3124          ;Aufbau der Einträge:
3124
3124          ;(Displaytext),(',',','.'),(Steuerzeichen),
3124          ;('.',''),(Steuerzeichen,''),(;)
3124
3124          ;Der Displaytext ist durch ein Komma begrenzt, wenn eine
3124          ;Eingabe erforderlich ist durch einen Punkt. Der Punkt muß
3124          ;dann auch bei den Druckersteuerzeichen stehen, wo der
3124          ;Parameter übergeben werden soll.
3124          ;Die Sequenz wird durch ein Semikolon begrenzt.
3124          ;Die gezeigten Einträge dienen nur als Beispiele.
3124
3124          FF      .BYT $FF        ;Textbeginn
3125
3125 TEXTS
3125          4658    .BYT 'FX-80 MENUE,;'  ;Startmeldung, keine Steuerzeichen
3132          454C    .BYT 'ELITESCHRIFT,',ESC,'M;'
313F          1B
3140          4D3B
3142          4E4F    .BYT 'NORMALSCHRIFT,',ESC,'P;'
3150          1B
3151          503B
3153          5052    .BYT 'PROPORTIONAL,',ESC,'p',1,';'
3160          1B
3161          70
3162          01
3163          3B
3164          5052    .BYT 'PROPORTIONAL AUS,',ESC,'p',0,';'
3175          1B
3176          70
3177          00
3178          3B
3179          4C49    .BYT 'LINKER RAND=.',ESC,'l.;'
3186          1B
3187          6C2E3B
318A          4645    .BYT 'FETTDRUCK,',ESC,'E;'
3194          1B
3195          453B
```

INHALTSVERZEICHNIS

Für die Jahrgänge 1982 - 1984 mit den Heften Nr. 23 - 40

Allgemeine Themen

6502 - 65xxx

6805/68705

6809

FORTH

Produktbesprechungen

MICRO MAG (vormals 65xx MICRO MAG) erscheint zweimonatlich, jeweils in der zweiten Hälfte der 'geraden' Monate.

Herausgeber und Bezug: Dipl.-Volkswirt Roland Löhr, Hansdorfer Str. 4, D-2070 Ahrensburg.

Bezugsbedingungen: Abonnement ab laufender Ausgabe für 6 Hefte DM 54,-- (Inlandsendpreis). Ausland/foreign via Surface mail DM 59,--. Abonnements laufen bis auf Wiederruf mit Kündigungsmöglichkeit bis zwei Wochen vor deren Ablauf.

Nachliefermöglichkeiten: Die früheren Hefte 7-13 können in Buchform bezogen werden. **Das Buch 7-13** kostet DM 42,--. Die Hefte ab Nr. 14 bis zur Gegenwart sind nachlieferbar, solange der Vorrat reicht, Stück DM 7,80, bei Abnahme von 10 Stück und mehr in einer Sendung DM 6,-- pro Stück. Hinzu kommen DM 2,50 je Sendung als Versandkosten. - Preisstand per 15.12.1984.

Private Besteller werden um Überweisung/Scheck zusammen mit der Bestellung gebeten. Besteller aus dem Ausland sollten per Postüberweisung/PTT bezahlen oder einen Scheck über DM herreichen. Konto: Roland Löhr, Nr. 654 70-202 Postgiroamt Hamburg.

Michael Beland, 2206 Sparrienshoop

Rekursionen

in FIG FORTH

Unter rekursiven Strukturen versteht man Funktionen oder Prozeduren, die sich selbst aufrufen können. Rekursive Lösungen sind bei bestimmten Problemen von einer Einfachheit, die mit iterativen Lösungen nicht zu erreichen wäre.

In FORTH sind rekursive Strukturen nicht möglich, denn wie man sich leicht überzeugen kann, kennt FORTH das Wort nicht, daß gerade compiliert wird. Dadurch können Worte, die fehlerhaft compiliert werden normalerweise dann auch nicht aufgerufen werden; ein durchaus sinnvolles Prinzip also. Die 'Tarnkappe' jedes Wortes ist das SMUDGE-Flag im Wort–Header, das während der Compilation gesetzt ist (=1) und erst am Wortende mit dem Wort SMUDGE umgeschaltet wird (=0). Um einen Aufruf aus dem Wort selbst zu ermöglichen, wäre folgende Lösung denkbar:

```
: name ... SMUDGE name SMUDGE ... ;
```

Ohne das zweite SMUDGE im Wort 'name' würde das SMUDGE-Flag erst am Wortende gesetzt werden. Bis dahin wäre das Wort unbekannt. Die gleiche Funktion wie die Sequenz SMUDGE name SMUDGE erfüllt das folgende Wort:

```
: RECUR LATEST PFA CFA , ;
```

Seine Funktion wollen wir am Beispiel der Fakultät ausprobieren. Sie ist wie folgt definiert:

FAK(X) = X*FAK(X-1), wobei FAK(0)=1

```
0 FAK . 1 OK
1 FAK . 1 OK
2 FAK . 2 OK
3 FAK . 6 OK
4 FAK . 24 OK
5 FAK . 120 OK
```

RECUR arbeitet also sehr zufriedenstellend und hat gegenüber der ersten Lösung den Vorteil, daß die Compiler-Sicherheit voll erhalten bleibt. RECUR (s.u.) dürfte übrigens in allen FIG-FORTH-Versionen arbeiten.

Doch bevor wir unser neues Werkzeug einsetzen, möchte ich zum Parameter- und Returnstack auf 65xx-Maschinen einige Anmerkungen machen. Der Parameterstack hat eine Kapazität von 65, der Returnstack eine solche von 128 Werten. Während die Stacktiefe für nicht rekursive Worte voll ausreichend ist, stößt man bei rekursiven Worten schnell an die Grenzen der Stacks, wie beim Beispiel 2 (FADEN).

Daher wurde das Modul REKURSION entwickelt. Es benötigt 470 Byte (hex 1D6) Speicher und stellt vier Befehle zur Verfügung:

:R	Definition eines rekursiven Wortes
;R	Ende einer solchen Definition
RECUR	rekursiver Aufruf, d.h. aus dem Wort selbst
RSTACK	Installation des Rekursionsstacks

Die von FORTH gewohnte Compiler-Sicherheit ist eingebaut, d.h. Kombinationen wie :R ... ; , : ... RECUR ... ;R usw. in falscher Paarung enden mit geordnetem Abbruch des Compilationsvorganges.

Der Returnstack

Der Returnstack wird nicht mehr zur Abspeicherung der Rückkehradressen rekursiver Worte benutzt. Dessen Funktion übernimmt der Rekursionsstack (s.u.). Gleichwohl muß man beachten,

daß z.B. Schleifenparameter auf dem Returnstack gespeichert werden. Ein wiederholter rekursiver Aufruf aus Schleifenstrukturen führt daher zum Überlauf des Returnstacks und ist sehr vorsichtig zu handhaben.

Der Parameterstack

Um einen für rekursive Strukturen ausreichend großen Parameterstack zur Verfügung zu haben, wird mit einem virtuellen Parameterstack gearbeitet. Er verhält sich für Benutzer wie der reale Parameterstack. Folgende Besonderheiten sichd jedoch zu beachten: Auf die letzten 22 Werte kann zugegriffen und pro rekursiven Aufruf dürfen max. 21 weitere Werte hinzugefügt werden. Der Stackoverflow beträgt maximal 10 Werte. Die Adressenzuordnung zu den im Programm verwendeten Konstanten ist im folgenden Schema zu sehen.

Der Rekursionsstack

Er liegt im freinen Speicher beginnend unterhalb FIRST und bis DP + D.DP*256 reichend. Dabei ist D.DP gleich der Anzahl der Speicherseiten (256 Byte), die oberhalb des Dictionaries DP als von Rekursionsstack unangetasteter Speicher zur Verfügung stehen. Wächst der Rekursionsstack über dieses Limit (nur das Hi-Byte wird zur Abfrage herangezogen), so wird die Fehlermeldung 'RE–CURSIVE WORD name ? STACK FULL' ausgegeben, der Rekursionsstack reininialisiert und das Programm beendet. Installiert wird der Rekursionsstack mit dem Wort RSTACK. Dies geschieht automatisch bei der Compilation rekursiver Worte.

Arbeitsweise beim Aufruf eines rekursiven Wortes

Die Rücksprungadresse wird auf den Rekursionsstack gepusht. Es wird geprüft, ob die Anzahl der Werte auf dem Parameterstack 2/3 seiner Kapazität überschritten hat. Ist das der Fall, werden die Werte des untersten Drittels des P.-Stacks auf den Rekursionstack geschaufelt. Auf diese kann nicht mehr mit Manipulationsbefehlen des Parameterstacks zugegriffen werden. Als Kennzeichnung werden zwei Null-Bytes auf den Rekursionsstack geschrieben.

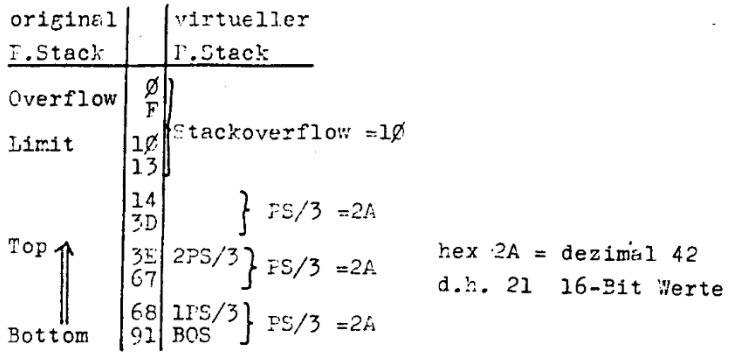
Arbeitsweise beim Rücksprung aus einem rekursiven Wort

Ist die vom Rekursionsstack geholte Rücksprungadresse gleich null, handelt es sich um die Kennzeichnung, daß PS/3 Werte vom Rekursionsstack auf den Parameterstack geschaufelt werden müssen. Der Darauf folgende Wert ist die Rücksprungadresse.

Anwendung: Labyrinth (FADEN)

Eine häufige Anwendung der Rekursion ist Backtracking, d.h. bildlich gesehen das Zurückgehen eines zuvor beschrittenen Weges bis zu einer Verzweigung, um von dort einen neuen Versuch zu starten. Um in unserem Bild zu bleiben, wollen wir am Beispiel 2 aus einem Labyrinth alle Wege nach außen finden. Das Labyrinth wird durch eine N*M Matrix dargestellt. Die Mauern symbolisiert der Buchstabe M, freien Weg ein Space. Von einem beliebig zu wählenden Startpunkt aus werden die angrenzenden Feldelemente begutachtet. Wird dabei ein freies Feld gefunden, so wird es mit einem Sternchen markiert. Der Vorgang wiederholt sich von dem gerade markierten Feldelement aus, bis ein Randfeld gefunden wird. Der Weg wird ausgegeben und das Backtracking beginnt bis zur letzten Verzweigung, von wo aus weiter gesucht wird. So werden alle möglichen Wege durchlaufen.

Anm.: d Hrsg. zur Hantierung: Die nachfolgenden Ausdrucke geben bereits wichtige Hinweise. BILD ist ein Befehlswort, das das Labyrinth zeigt. Es besteht aus 9 Zeilen zu je 17 Zeichen. Eine Zeile wird nach Zeilennummer und 'Z' und Return gefüllt, mit Spaces und 'Hindernissen'. Die Reihenfolge der Eingabe für die neun 'Z' ist beliebig. Man kontrolliert mit BILD. Nach fertiggestelltem Labyrinth gibt man einen Startpunkt, z.B. 7 8 FADEN und findet dann in aufeinanderfolgenden Bildern die möglichen Wege aus dem Labyrinth mit Sternchen gekennzeichnet. Das hier abgedruckte Labyrinth ist einfach, nicht alle Wege sind abgedruckt.



Belegung der Zero Page beim AIM 65. - AB/AC = RSP (Recursion Stack Pointer)

```
( REKURSION IN 6502 FIG - FORTH )

( MICHAEL BELAND     2 - DEZ - 1984   )
( WALDSTR. 41                         )
( 2206 SPARRIESHOOP                   )

HEX
AB CONSTANT RSP ( RECURSION STACK POINTER)
CONSTANT D.DP ( *100= FREIER SPEICHER HINTER DICTIONARY)
91 CONSTANT BOS ( BOTTOM OF STACK)
2A CONSTANT PS/3 ( DRITTEL DES PARAMETERSTACKS)
BOS PS/3 - 1+ CONSTANT 1PS/3
1PS/3 PS/3 - CONSTANT 2PS/3
100 PS/3 - CONSTANT -PS/3

: BOX ASSEMBLER ,X FE ;

: RSTACK ( --)    FIRST RSP ! ;

: RSOV ( --)     ( RETURNSTACK OVERFLOW HANDLER)
  RSTACK ." RECURSIVE WORD" 7 ERROR ;

CODE PUSHR ( BOX -> RS )
  ( RSP AUF NAECHSTE LEERE ZELLE)
  SEC, RSP LDA, 2 # SBC, RSP STA,
  CS NOT IF, RSP 1+ DEC, THEN,
  DP 1+ LDA, SEC, D.DP # ADC, RSP 1+ CMP, CS
   IF, ' RSOV CFA 0 100 M/
   LITERAL # LDA, W 1+ STA, LITERAL # LDA, W STA,
   W 1- JMP, ( RETURNSTACK OVERFLOW) THEN,
  BOX 1+ LDA, 1 # LDY, RSP )Y STA, ( BOX IN RS)
  BOX LDA, DEY, RSP )Y STA,
  RTS, END-CODE

CODE PULLR ( RS -> BOX )
  1 # LDY, RSP )Y LDA, BOX 1+ STA,
  DEY, RSP )Y LDA, BOX STA,
  RSP LDA, CLC, 2 # ADC, RSP STA,
  CS IF, RSP 1+ INC, THEN,
  RTS, END-CODE
```

```
: RECUR ( N -- N)
 ?COMP CSP § 2- § 53 =
   IF LATEST PFA CFA ,
   ELSE 14 ERROR THEN
  ; IMMEDIATE

: ;R ( -- N)
  ?EXEC !CSP 53
  RSTACK CURRENT § CONTEXT ! CREATE Ü ;CODE IMMEDIATE
  IP LDA, BOX STA, IP 1+ LDA, BOX 1+ STA, ' PUSHR JSR,
  CLC, W LDA, 2 # ADC, IP STA, TYA, W 1+ ADC, IP 1+ STA,
  2PS/3 # CPX, CS NOT ( X < 2/3 DER PS.-KAPAZITAET ?)
   IF, XSAVE STX, 1PS/3 # LDX, ( DATEN AUF REKURSIONSS.)
    BEGIN, INX, INX, ' PUSHR JSR,
    BOS 1+ LITERAL # CPX, 0= UNTIL,
   BOS 1- STY, BOS STY, ' PUSHR JSR, ( NULL-BYTES)
  1PS/3 # LDX, ( DATEN AUF PARAMETERSTACK VERSCHIEBEN)

  BEGIN, DEX, TOP LDA, PS/3 ,X STA, XSAVE CPX, 0= UNTIL,
   TXA, CLC, PS/3 # ADC, TAX, THEN,
  NEXT JMP, END-CODE

CODE ;SR ( --)
  ' PULLR JSR, BOX LDA, BOX 1+ ORA, 0=
   IF, XSAVE STX,
    BEGIN, TOP LDA, ( DATEN AUF PARAMETERS. VERSCHIEBEN)
    -PS/3 ,X STA, INX, BOS 1+ LITERAL # CPX, 0= UNTIL,
    BEGIN, ( DATEN VOM REKURSIONS- AUF PARAMETERSTACK)
     ' PULLR JSR, DEX, DEX, 1PS/3 # CPX, 0= UNTIL,
    XSAVE LDA, SEC, PS/3 # SBC, TAX,
   ' PULLR JSR, ( NEUE RUECKSPRUNGADRESSE)
   THEN,
  BOX LDA, IP STA, BOX 1+ LDA, IP 1+ STA,
  NEXT JMP, END-CODE

: ;R ( N --)
  53 = NOT IF 14 ERROR THEN ?CSP
  COMPILE ;SR SMUDGE ACOMPILEÜ A
  ; IMMEDIATE

DECIMAL
FINIS



( FAKULTAET              BEISPIEL  - 1 -  )
:R FAK ( N -- N)
   -DUP IF DUP 1- RECUR * ELSE 1 THEN ;R
FINIS


( BACKTRACKING & REKURSION                BEISPIEL  - 2 - )
DECIMAL
17 9 CONSTANT ZEILEN CONSTANT SPALTEN
0 VARIABLE LAB ZEILEN SPALTEN 1+ * ALLOT
: Z ( N --)    ( EINGABE EINER ZEILE)
  ZEILEN MIN 1 MAX 1- SPALTEN 1+ * LAB + SPALTEN EXPECT ;
```

```
:BILD ( --)    ( AUSGABE DES LABYRINTHS)
  ZEILEN 0 DO CR LAB I SPALTEN 1+ * + SPALTEN TYPE LOOP
  CR ." IRGENDEINE TASTE ?" HANG ; HEX
: ADRESSE ( ZEILE SPALTE -- ZEILE SPALTE ADRESSE)
  2DUP 1- SWAP 1- SPALTEN 1+ * + LAB + ;
: ?RANDFELD ( ZEILE SPALTE -- ZEILE SPALTE FLAG) 2DUP 2DUP
 1 = SWAP 1 = OR SWAP SPALTEN = OR SWAP ZEILEN = OR ;
:R FADEN ( ZEILE SPALTE --)    ( STARTFELD)
   ADRESSE C@ 20 = ( FELD FREI?)
    IF ADRESSE 2A SWAP C! ( FADEN AUSLEGEN) ?RANDFELD
     IF BILD ELSE 2DUP 1+ RECUR ( NACH RECHTS)
                  2DUP 1- RECUR ( NACH LINKS)
                  OVER 1+ OVER RECUR ( NACH UNTEN)
                  OVER 1- OVER RECUR ( NACH OBEN)
     THEN
    ADRESSE 20 SWAP C! ( FADEN EINHOLEN)
    THEN
   2DROP
   ;R DECIMAL

FINIS
```

```
ON BILD
MMMMMMMMMMMMMMMMM
M               M
   MMMMMMMMMMMMMMM
                MMMM
M MMMMM   MMMMMMMMM
M MMMMM    MMMMMMMM
M MMMMM MM MMMMMMM
M
MMMMMMMMMMMMMMMMM
IRGENDEINE TASTE ?0I
7 8 FADEN
MMMMMMMMMMMMMMMMM
M               M
   MMMMMMMMMMMMMMM
                MMMM
M MMMMM   MMMMMMMMM
M MMMMM    MMMMMMMM
M MMMMM*MM MMMMMMM
M       **********
MMMMMMMMMMMMMMMMMM
IRGENDEINE TASTE ?
MMMMMMMMMMMMMMMMM
M               M
   MMMMMMMMMMMMMMM
**              MMMM
M*MMMMM   MMMMMMMMM
M*MMMMM    MMMMMMMM
M*MMMMM*MM MMMMMMM
M*******
MMMMMMMMMMMMMMMMMM
IRGENDEINE TASTE ?
```

```
MMMMMMMMMMMMMMMMM
M               M
**MMMMMMMMMMMMMMM
  *               MMMM
M*MMMMM   MMMMMMMMM
M*MMMMM    MMMMMMMM
M*MMMMM*MM MMMMMMM
M*******
MMMMMMMMMMMMMMMMMM
IRGENDEINE TASTE ?
MMMMMMMMMMMMMMMMM
M               M
   MMMMMMMMMMMMMMM
********        MMMM
M MMMMM**MMMMMMMMM
M MMMMM** MMMMMMMM
M MMMMM*MM MMMMMMM
M
MMMMMMMMMMMMMMMMM
IRGENDEINE TASTE ?
MMMMMMMMMMMMMMMMM
M               M
   MMMMMMMMMMMMMMM
 *******        MMMM
M*MMMMM**MMMMMMMMM
M*MMMMM** MMMMMMMM
M*MMMMM*MM MMMMMMM
M****************
MMMMMMMMMMMMMMMMMM
IRGENDEINE TASTE ?
MMMMMMMMMMMMMMMMM
```

Wolfgang Radeloff, 2080 Pinneberg

Anpassung des PL/65

an das SPS des AIM 65

PL/65 ist die einzige Sprache des AIM 65, die vom neuen 'Software Preparation System' nicht unterstützt wird. Dies mag an der geringen Verbreitung liegen, aber auch an dem im ROM des Compilers notwendig werdenden Änderungen.

PL/65 läßt die Anwendung des erweiterten Befehlssatzes ohne Änderungen zu, weil in die Anweisungen der Sprache Formulierungen in einfachen Anführungsstrichen eingefügt werden dürfen, die unverändert als Strings an den Assembler weitergegeben werden. Das ist auf den Seiten 4-16 und 4-17 des Handbuches erklärt worden. Eine einzelne Anweisung dieser Art wird z.B. eingeschoben als LABEL: 'JMP (VEKTOR,X)'. Blöcke von Anweisungen, die als Assemblerbefehle ausgelegt werden sollen werden zwischen Sterne plaziert, wobei der erste Stern den Compiler ausschaltet, während ihn der zweite wieder hereinruft. Mit neuen Befehlen dafür ein Beispiel:

```
* UMSCHALTUNG AUF ASML.KODE
L2 PLA
LSR A
LDA #8
BCS L3
TRB CRA
RTS
L3 TSB CRA
RTS
* RUECKSCHALTEN PL/65
```

Wird das SPS ohne die Video-Karte des RM-65 Systems betrieben, so sind nur zwei Anpassungen notwendig:

Das Aufsetzen auf die erste Zeile des Texteditors wird mit 'TOPNO' vorgenommen. Hier ist im SPS die Adresse geändert: TOPNO=$F8CE.

Der Compilerlauf wird bei nicht interpretierbaren Fehlern durch einen Softwareinterrupt (BRK) abgebrochen. Der AIM Monitor disassembliert die folgende Anweisung nicht, wenn REGF=1 ist. PL/65 setzt REGF, bevor BRK ausgeführt wird. Im SPS wird dazu DISFLG=1 benötigt.

Der Betrieb der RM-65 CRTC-Karte macht eine weitere Änderung notwendig:

- Der von PL/65 ab Adresse 2A0 unterhaltene Buffer (81) Bytes kommt mit den Variablen des CRTC in Konflikt. Als neuer Bereich wird DBUFF des CRTC benutzt. Dieser Buffer wird für Insert und Delete des Video-Betriebssystems benutzt. Er wird mit Sicherheit nicht während des Compilerlaufes benutzt werden.

Zwei neue Funktionen des SPS erleichtern den Betrieb des PL/65-Compilers besonders: Memory als Ausgabekanal und das Umsetzen der Editorvariablen (RECOVER) auf einen neuen Text. Dazu sind die folgenden Anpassungen notwendig:

- Die Ausgabe von CR/LF am Ende einer jeden von PL/65 generierten Zeile wird auf CR allein eingeschränkt.
- Wird die Ausgabe in Pass 1 oder Pass 2 in das Memory gelenkt, so wird der Text mit CR/00 abgeschlossen. Das muß auch für den Abbruch mit BRK gelten.

Diese Änderungen sind auch für den Betrieb unter dem AIM-Monitor sinnvoll. - Unter SPS wird PL/65-Sourcetext jetzt so bearbeitet:

```
ROCKWELL SPS - PL/65 V2.0
IN=M      OUT=U
DEVICE NO.  =4
FROM=2000  TO=4000
PASS(1 or 2)? =2
```

Der generierte Assembler-Sourcecode wird im Bereich hex 2000 bis 4000 abgelegt. Zur Weiterverarbeitung wird mit der U-KEY-Funktion 7 der Editorbereich umgeschaltet:

```
<U>

FUNKTION NO. =7
EDITOR
1000  FROM=2000   2000  TO=4000
(1. Zeile des Assmbl.-Sourcetextes.)
```

Der Editor, der vorher im Bereich $1000-2000 den PL/65 Sourcetext verwaltete, ist umgeschaltet und verwaltet nun den generierten Assembler-Sourcetext. Ein Rückschalten ist auf die gleiche Weise möglich.

Da PL/65 und der SPS-Assembler jetzt in den gleichen Speicherbereichen arbeiten, muß eine Umschaltung der ROMs vorgenommen werden. Hierzu sei auf (4) verwiesen.

Der Verfasser meint, daß die Sprache PL/65 zu Unrecht bei den AIM 65 Betreibern wenig Beachtung gefunden hat. Der erzeugte Code ist übersichtlich und folgt den im Handbuch für jedes Statement dokumentierten Beispielen. Sie kommt den Bestrebungen, leicht zu überprüfende strukturierte Programme zu schreiben, entgegen.

Literaturhinweise:
(1) SPS für AIM 65. R. Löhr, MICRO MAG Nr. 39/1984
(2) Strukturiert und schnell: PL/65. Dipl.-Ing. H. W. Lange, 65xx MICRO MAG Nr. 24/1982
(3) Der AIM 65 PL/65-Compiler. Ing. O. Kreil, 65xx MICRO MAG 25/1982
(4) ROM-Umschaltung per Software für AIM 65. W. Jansen, MICRO MAG 36/1984
(5) Formatting AIM Assembler Listings Using PL/65. MICRO, the 6502 Journal 45/1982
(6) PIPeLINE, 4. Strukturiertes Programmieren mit PL/65. R.G. Briener, Der Elektroniker 12/80
(7) PL/65 User's Manual, Rockwell Document No. 29650 N65, May 1980.

```
PAGE 01
LINE  ADDR  OBJECT          SOURCE

0001  0000                      .OPT NOS,NOC
0002  0000                ;*****************************************
0003  0000                ;** ANPASSUNG PL65 AN DAS SPS SYSTEM     **
0004  0000                ;** ------------------------------      **
0005  0000                ;**                                     **
0006  0000                ;** 03.11.1984 WOLFGANG RADELOFF        **
0007  0000                ;** FILE: 'PLSPS'  ARCHIV: 34/84        **
0008  0000                ;**                                     **
0009  0000                ;*****************************************
0010  0000                VAR1   =$0366
0011  0000                BUFF5  =$0367          ;INS/DEL BUFFER CRTC
0012  0000                DISFLG =$A40F          ;=1 : KEIN DISASSEMBL.
0013  0000                TOPNO  =$FBCE          ;SPS
0014  0000                OUTALL =$E9BC
0015  0000                OUTFLG =$A413
0016  0000                TEMP   =$05F3          ;SPS DEVICE NR.
0017  0000                ;
0018  0000                ;1. AUFSETZEN AUF DIE ERSTE EDITOR ZEILE
0019  0000                ;
0020  0000                      *=$B022
0021  B022  20 CE FB           JSR TOPNO
0022  B025                ;
```

```
0023  B025                ;2. BUFFER AB $02A0 VERLEGEN
0024  B025                ;
0025  B025                      *=$B2ED
0026  B2ED  8D 66 03            STA VAR1
0027  B2F0                      *=$B6F1
0028  B6F1  AD 66 03            LDA VAR1
0029  B6F4                ;
0030  B6F4                      *=$B4F4
0031  B4F4  99 67 03            STA BUFF5,Y
0032  B4F7                      *=$B52B
0033  B52B  99 67 03            STA BUFF5,Y
0034  B52F                      *=$B54F
0035  B54F  99 67 03            STA BUFF5,Y
0036  B552                      *=$B571
0037  B571  99 67 03            STA BUFF5,Y
0038  B574                      *=$B590
0039  B590  BD 67 03            LDA BUFF5,X
0040  B593                      *=$B5A4
0041  B5A4  BD 67 03            LDA BUFF5,X
0042  B5A7                ;
0043  B5A7                ;3. ANPASSUNG AN DEN SPS TEXTEDITOR
0044  B5A7                ;3.1 CR STATT CR/LF AM ZEILENENDE.
0045  B5A7                ;
0046  B5A7                      *=$B934
0047  B934  4C BC E9            JMP OUTALL
0048  B937                ;
0049  B937                ;ABBRUCH UEBER BRK-VEKTOR: CR/00 AN EDITOR
0050  B937                ;
0051  B937                      *=$B83D
0052  B83D  20 CB CF            JSR PATCH1
0053  B840                ;
0054  B840                ;3.3 ENDE PASS1/2, 00 AN EDITOR
0055  B840                ;
0056  B840                      *=$B049
0057  B049  20 D1 CF            JSR PATCH2
0058  B04C                ;
0059  B04C                ;4. PL/65 MELDET SICH NUN MIT:
0060  B04C                ;
0061  B04C                      *=$CFB0
0062  CFB0  0D                  .BYT $0D,'ROCKWELL $PS - PL/65 V2.0',0
0063  CFCB                ;


PAGE 02
LINE  ADDR  OBJECT          SOURCE

0064  CFCB                ;5. IM ANSCHLUSS: DIE KORREKTUR-ROUTINEN:
0065  CFCB                ;
0066  CFCB  8E 0F A4      PATCH1 STX DISFLG      ;AIM MON: REGF =$A40F
0067  CFCE  20 E3 CF            JSR CR           ;SEND CR TO A.O.D.
0068  CFD1                ;
0069  CFD1  AD 13 A4      PATCH2 LDA OUTFLG      ;TEST, OB OUT=U & DEV=4
0070  CFD4  C9 55                CMP #'U'
0071  CFD6  D0 0B                BNE CR           ;NUR CR ALS ABSCHLUSS
0072  CFD8  AD F3 05             LDA TEMP
0073  CFDB  C9 04                CMP #4
0074  CFDD  D0 04                BNE CR
0075  CFDF  A9 00                LDA #0           ;EOT MARKIERUNG
0076  CFE1  F0 02                BEQ OUT
0077  CFE3  A9 0D         CR     LDA #$0D
0078  CFE5  4C BC E9      OUT    JMP OUTALL
0079  CFE8                ;
0080  CFE8                      .END PLSPS 34/84

 ERRORS= 0000
```

Peter Porbadnigk, 2070 Ahrensburg

AS - 64

Ein Assembler für den Commodore C–64

Der AS–64 ist ein von der Firma Roßmüller vertriebenes Software-Paket, bestehend aus einem Assembler für 6502/65C02, einem Editor, einem Monitor und einem Re-Assembler. Assembler, Editor und ein einfacher Monitor werden als Steckmodul für den Expansionsport des C-64 geliefert. Auf der beiliegenden Diskette findet man einen Spezialmonitor, den Reassembler sowie zwei Demofiles.

Der Editor ist ein Full Screen Editor, der ohne Zeilennummern arbeitet. Zeilen werden direkt auf dem Schirm editiert, eingefügt oder gelöscht. Innerhalb des Textes kann man sich auf einfache Weise durch Auf- oder Abwärtsscrollen oder bildschirmweises Blättern bewegen. Zur Bearbeitung des Textes stehen folgende Befehle zur Verfügung:

Insertline	Zeile einfügen
Deletline	Zeile löschen
Insix	6 Zeilen einfügen
Delleft	vom Cursor aus zum linken Rand löschen
Gerät	I/O Device festlegen
Save	Text saven
Load	Text laden
Backup	Saven eines Files 'Name' mit vorherigem Umbenennen des alten Files 'Name' in 'BU-Name'
Merge	Anhängen oder Einfügen von Text
Disk	Floppy Befehle senden
Status	Status der Floppy holen (Fehlermeldung)
Katalog	Auslisten des Directory
Hardstart	löscht gesamten Speicher
Assemblieren	Start des Assemblers
Finden	Suchen nach einer Zeichenkette
Edit	Austauschen einer Zeichenkette gegen eine andere
Killrest	Text vom Cursor bis Ende löschen
Colour	Bildschirmfarben ändern
Print	Text ab Cursor drucken
Repeat	Geschwindigkeit der Repeatfunktion ändern
Tabulator	Text im Editor formatieren (Herausheben der Labels)
Exit	Start des Maschinensprache-Monitors
Symbols	listet die Symboltabelle
Umwandlung	wandelt den Text in ASCII-Code
40Zeichen	formatiert den Text auf 40 Zeichen pro Zeile
Deffunk5	Es kann eine 14 Zeichen lange Tastenfolge programmiert werden, die dann mit der Taste F5 wieder aufgerufen bzw. auch ausgeführt werden kann, wenn Steuercodes eingegeben wurden
Deffunk6	Wie Deffunk5, nur für Taste F6
Zeilen	Markieren von Zeilen (Anfang–Ende)
Kopieren	Die markierten Zeilen werden hinter den Cursor kopiert, die alten Zeilen bleiben erhalten
Verschieben	Die markierten Zeilen werden an die Cursorposition verschoben
Killen	Die markierten Zeilen werden gelöscht

Hotcopy	Die markierten Zeilen werden ausgedruckt
Memo	Die markierten Zeilen werden abgespeichert.

Die vorgenannten Befehle werden durch die Funktionstasten oder durch andere Tasten mit vorgestellter CTRL-Taste aufgerufen. Sie ermöglichen dem Anwender ein relativ komfortables Editieren der Quelltexte. Diese Texte können bis zu 30 kByte lang sein bzw. in einem Stück im Editor bearbeitet werden.

Der AS–64 ist ein 2 Pass Assembler, der auch die Opcodes der 65C02 CPU bearbeiten kann. Der Quelltext kann aus dem Speicher oder von Floppy bzw. Cassette gelesen werden. Somit ist auch ein Assemblieren von längeren Texten mit Verkettungen möglich. Weiterhin kann man nur Pass 1 oder nur Pass 2 ausführen lassen. Die Assemblerliste kann entweder über die IEEE-Schnittstelle oder eine Centronix-Schnittstelle am Userport ausgegeben werden. Es wird eine Symboltafel erstellt, die ebenfalls ausgedruckt werden oder auch auf Floppy oder Cassette abgespeichert werden kann. Für die Steuerung des Assemblers stehen dem Anwender einige Pseudo-Opcodes zur Verfügung:

.BA LABEL	Startadresse
.MC LABEL	Startadresse für Assemblieren mit Offset
.LA LABEL LABEL	legt Start- und Endadresse der Symboltafel fest
.OS	Objektcode im RAM ablegen
.CE	Fehler beim Assemblieren ignorieren
LABEL .DE VALUE	LABEL wird der Wert VALUE zugewiesen
.SE LABEL	LABEL als Low/Highbyte abspeichern
.DS VALUE	Überspringen von VALUE Bytes
.BY	Es können Texte, Hex-, Dez– und Binärzahlen folgen
.LS	Assemblerliste ausgeben
.LC	Assemblerliste stoppen
.CT 'NAME'	Der Text 'NAME' wird geladen und die Assemblierung fortgesetzt
.EN	Ende des Textes

Mit diesen Pseudoopcodes ist ein recht komfortables Arbeiten mit dem Assembler möglich. Zusätzlich besteht die Möglichkeit der bedingten Assemblierung mit den Opcodes IEQ (If Equal), INE (If Not Equal), IPL und IMI (If Plus oder Minus). Wie auch schon in den Pseudoopcodes besteht auch in den weiteren Funktionen eine weitgehende Kompatibilität zum MAE-Assembler, wie z.B. das Kennzeichnen der Zeropage-Adressen mit einem * (*label) oder den langen Labelnamen (ca. 40 Zeichen) sowie der Behandlung von Low- und Highbyte-Ladebefehlen: LDA #L,LABEL und LDA #H,LABEL. - Es stehen dem Anwender einige Möglichkeiten der Speicheraufteilung zur Verfügung. So kann z.B. ein langes Maschinenprogramm, das sich aus mehreren Texten zusammensetzt, mit einem Offset in dem zum Betriebssystem parallel liegenden RAM abgelegt werden oder aber in den nicht genutzten Bereichen hex C000 oder 7000. Das Verschieben in den Original-Adressenbereich übernimmt der Monitor.

Die Geschwindigkeit des Assemblers läßt keine Wünsche offen. Es werden ca. 20 kByte Text in ca. 6 Sekunden zu 2 kByte Maschinenprogramm assembliert. Bis hierher kann gesagt werden, daß der Editor im Zusammenhang mit dem Assembler sehr schnell und zuverlässig gearbeitet werden kann. Es ist sehr praktisch, den erstellten Text aus dem Speicher zu assemblieren und bei Fehlern (die der Assembler in Klartext ausgibt) sofort im Editor zu verbessern und erneut zu assemblieren.

Vervollständigt wird dieses System durch einen kleinen im Editor- und Assembler-ROM untergebrachten **Maschinen-Monitor.** Dieser kann einerseits den erstellten Code aus dem Offsetbereich an die Originaladressen verschieben, er kann laden, saven, disassemblieren, MPG's starten und Memory Dumps erstellen. Sehr nützlich ist der auf der Diskette mitgelieferte **Reassembler,** der aus einem MPG einen assemblierfähigen Text im Editor erstellt. Hierzu wird bei der Initialisierung

des Editors der Reassembler geladen und später vom Monitor aus gestartet. Nach Eingabe der Start- und Endadresse und evtl. Tabellen wird ein Textfile erzeugt, das der AS–64 sofort wieder verarbeiten kann. Tabellen werden grundsätzlich als Hextabellen angelegt, im Kommentarfeld aber als ASCII dargestellt.

Zusammenfassend kann man bis hier sagen, daß die vier oben genannten Programme gut zusammen arbeiten und leicht zu bedienen sind. Der Anwender erhält ein Programmpaket, mit dem er in der Lage ist, Maschinenprogramme auf einfache und komfortable Weise zu erstellen oder mit Hilfe des Reassemblers zu modifizieren.

Nicht ganz in das Bild der o.g. Programme paßt der **SMON (Spezialmonitor)**, der ebenfalls auf der Diskette geliefert wird. Hier gibt es einige Schwierigkeiten im Zusammenhang mit dem AS-64 bzw. dem Editor. Auch der SMON wird wie der Reassembler nachgeladen und dann vom Monitor aus gestartet. Der SMON bietet eine Vielfalt von Befehlen zur Untersuchung von Maschinenprogrammen, wie z.B. suchen, tracen, verschieben, umrechnen, disassemblieren. Implementiert ist ein Diskmonitor, der gezielt auf Track und Sektor zugreift und nach evtl. Änderungen diesen Track zurückschreiben kann. Selbständig, d.h. als eigenständiges Programm - und nicht als Unterprogramm vom AS-64 -arbeitet der SMON sehr gut. Vielleicht wird dieser Mangel in Zukunft noch abgestellt.

Info: Roßmöller GmbH, Finkenweg 1, 5309 Meckenheim, T. 022 25 - 144 88.

□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O

Wolgang Masarie, 8346 Simbach

MAILMANAGING

Adressierungs- und Etikettenprogramm für MC-65

Wahrscheinlich geht es vielen Leuten so, daß die Korrespondenz immer umfangreicher und die dafür zur Verfügung stehende Zeit immer weniger wird.. Die vor einigen Jahren aufgekommenen Tabellier-Etiketten bringen eine wesentliche Erleichterung bei der Bewältigung der Post. Für ihren ökonomischen Einsatz ist aber ein spezielle Etikettendruckprogramm nötig: Man holt sich mit Stichwort die benötigte Adresse, und läßt sie ausdrucken, auch gleich in den Briefkopf. Die größte Arbeitsersparnis hat man natürlich bei Serienbriefen, jeder mit einem anderen Briefkopf. Sofern man keine Klarsichtkuverts verwendet, können die Anschriften im nächsten Arbeitsgang auf die Etiketten gedruckt werden.

Das hier besprochene Programm 'Mailmanaging' kann das alles, benötigt nicht einmal zwei Pages und läßt sich auf dem MC-65, AIM 65 und kompatiblen Computern ohne Schwierigkeiten einrichten.

Die Erstellung einer Computerkartei lohnt sich schon bei wenigen Adressen, nicht nur im Büro, sondern auch privat. Mit den Anschriften von Freunden, Firmen und Behörden kommt schon einiges zusammen. - Eine Datei soll möglichst universal verwendbar sein und auch Stichworte bzw. Klassifizierungen enthalten, mit denen man bestimmte Adressen heraussuchen oder auch Adressengruppen aussortieren lassen kann. Diese zusätzlichen Informationen sollen im allgemeinen nicht in das Adressenfeld gedruckt werden.
Das Programm Mailmaniaging druckt nur die ersten 26 Zeichen einer Adressenzeile aus. Alles was rechts davon steht, wird ignoriert. Eine Adresse soll vier Zeilen umfassen, die Postleitzahl muß ganz links in der letzten Zeile stehen. Das Programm orientiert sich an der Postleitzahl und zeigt nur Adressen an, wo sie an der richtigen Stelle steht. Die Staatenkennung muß ggfs. von Hand nachgetragen werden. Eine Kartei kann dann etwa so aussehen:

dieser Teil wird nicht gedruckt

```
.....................................................
. Firma                    . ACHT                   .
. W. Achtleitner GmbH      . COMP GEBRAUCHT         .
. Erlenstr. 42             . 088/24436              .
. 4470 MEPPEN              .                        .
. An das                   . FRAN                   .
. Franzis-Software-Service . COMP SOFT              .
. Karlstraße 37-41         .                        .
. 8000 MÜNCHEN 2           .                        .
. Herrn                    . GRAN                   .
. Willi Granhelm           . COMP                   .
. Weitzmannstraße 30       .                        .
. 3340 WOLFENBÜTTEL        .                        .
. Firma                    . HART                   .
. Hartriegel-Elektronik    . COMP ZUBEHÖR TABELLIERPAPIER .
. Kolbinger Straße 66      . 766/24464              .
. 2071 AMMERSBEK 1         .                        .
. Herrn                    . MASA                   .
. Wolfgang Masarie         . AIM  AME  MUSIK        .
. postlagernd              .                        .
. 8346 SIMBACH             .                        .
.....................................................
```

Bei einer Kartei mit einem solchen Schema haben in 32 KB Speicher etwa 250-300 Adressen Platz. Die Frage des Massenspeichers soll hier nicht behandelt werden. Bei mehr als 300 Adressen geht es ohne Floppy fast nicht mehr. Das vorliegende Programm arbeitet mit Daten, die im Textspeicher liegen, wobei es gleichgültig ist, ob Brieftexte oder Kartei zuerst abgespeichert sind.

Um das Programm so kurz wie möglich zu halten, ist es fest auf Standard-Etiketten von 89x23,4 mm eingestellt (zwei Leerzeilen zwischen jeder Adresse). Andere Größen können mit den Befehlen 'P' (Adresse Drucken) und 'N' (eine Zeile weiterrücken) verarbeitet werden. Für Serienausdrucke anderer Größen ist eine Änderung des Programms durch Einfügen von beliebig vielen JSR FREELN bei $031E möglich.

Befehle des Adressen-Ausgabeprogramms:

D = DOWN — Nächste Adresse im Text suchen und am Display anzeigen. Die Taste "SPACE" hat die gleiche Funktion, was die Bedienung vereinfacht.

E = SCREEN — Sprung zum Bildschirmeditor. Der Eintritt erfolg an der Stelle des Textspeichers, wo die zuletzt angezeigte Postadresse liegt. Dieser Befehl wird zum Ausbessern von Adressen verwendet.

F = FIND — Es kann ein Name oder eine beliebige Zeichenfolge mit bis zu 19 Zeichen eingegeben werden. Abschluß mit Return. Das Programm zeigt die Anschrift an, in der die Zeichenfolge gefunden wurde.

G = FINDALL — Genauso wie beim vorherigen Befehl wird ein Name oder ein Stichwort eingegeben. Das Programm sucht den Textspeicher nach Adressen durch, in denen die Zeichenfolge vorkommt, und druckt alle gefundenen Adressen aus.

H = HELP — Gibt einen Text zur Bedienerführung aus, in dem alle Befehle mit Stichwort angegeben sind.

I = INSERT — Der Befehl verwendet die Kopier-Routine aus dem erweiterten Texteditor. Die Anschrift kann an eine andere Stelle des Textspeichers (in einen

Brieftext) kopiert werden. "I" wirkt ähnlich wie "S", ist aber vielseitiger verwendbar.

N = LINEFEED Gibt eine Leerzeile an den Drucker aus. Diese Taste dient zum Adjustieren der Etiketten.

P = PRINT Druckt die gerade am Display angezeigte Anschrift aus und rückt zur nächsten weiter. Auch diese wird angezeigt.

Q = SCREEN Sprung zum Bildschirmeditor. Abweichend vom Befehl "E" erfolgt der Sprung an die Textstelle, die vom Cursor vor Eintritt in das Adressenprogramm angezeigt wurde. Dient zur Änderung von Brieftexten bei Serienbriefen.

S = SERIES Dieser Befehl dient zum Erstellen von Serienbriefen. Ein im Textspeicher vorliegender Brieftext wird ausgedruckt, wobei an der Stelle, die mit zwei CONTROL-F ($06) Steuerzeichen markiert ist, die gerade am Bildschirm angezeigte Adresse eingefügt wird. Der Brief wird dann weiter bis zum nächsten Auftreten von zwei CONTROL-F ausgedruckt. Danach rückt das Programm zur nächsten Postadresse, die sofort wieder für einen weiteren Brief verwendet werden kann. Der Anfang des Briefes wird durch die Position des Textcursors vor Eintritt in das Adressenprogramm markiert. Bei der Änderung von Serienbriefen mit "Q" muß der Cursor nach durchgeführter Änderung und vor Rückkehr in die Adressenausgabe an den Briefbeginn zurückgesetzt werden.

T = TOP Die erste im Textspeicher vorkommende Anschrift wird angezeigt.

U = UP Springt zur vorhergehenden Anschrift zurück und zeigt diese am Bildschirm. Man kann also den Speicher nach beiden Richtungen durchsuchen.

Z = PRINTALL Druckt die gerade angezeigte und alle folgenden Adressen aus. Auch für diesen Befehl gilt, daß nur die anhand der Postleitzahl als Adresse erkannten Textstellen ausgedruckt werden. Ein zwischen den Anschriften eingefügter erklärender Text wird ignoriert.

Die Help-Anzeige wird zur Bedienerführung zu Beginn des Programms und bei Betätigen der Taste 'H' wie folgt ausgegeben:

```
#########################################
# D=DOWN E/Q=SREEN F=FIND G=FINDALL N=LF #
# P=PRINT S=SERIES T=TOP U=UP Z=PRINTALL #
# I=COPY H=HELP  )CONTROL-F(             #
#########################################
```

Das Programm beginnt in Adresse hex 200, um es für AIM-Systeme aller Ausbaustufen nutzbar zu machen. Anstelle des gebräuchlichen 'ON GOTO' mit Code- und Sprungtabelle dient eine platzsparende Abfragekette als Verteiler. $D800 stellt den Einstieg zum Screeneditor dar. Diese Adresse muß für andere Systeme angepaßt werden. Der Bildschirmeditor ist selbst ein Unterprogramm. Wenn er mit der Taste CTRL-Q verlassen wird, kommt man zur Adressenausgabe zurück, wobei allerdings wieder bei der ersten Postanschrift begonnen wird.

Falls kein Bildschirmeditor vorhanden ist, kann man den AIM-Texteditor mit JMP FA7B erreichen. Auch in diesem Fall ist ein Rücksprung zum Mailmanaging nicht schwer, denn die Rücksprungadressen SRCHD für den Befehl 'E' und SVNL für den Befehl 'Q' sind auf die Funktionstasten F1 und F2 gelegt.:

Das Unterprogramm COPY (DCBD) ist Bestandteil einer Erweiterung des AIM-Texteditors und wird wohl ebenfalls nicht allen MC-65- oder AIM-Betreibern zur Verfügung stehen. Dieser Programmteil kopiert einen beliebigen Textblock und fügt das Duplikat an einer anderen Stelle des Textes ein. Es ist sehr umfangreich, so daß man den Befehl 'I' auch mit einer anderen Funktion belegen kann, wenn eine Routine in ähnlicher Form schon vorhanden ist. - Die nachfolgenden drei Unterprogramme kommen beim Standard-AIM ebenfalls nicht vor, können aber leicht implementiert werden.

```
. 8143 A000    LDY #00      Diese Routine gibt einen ASCII- .
. 8145 68      PLA          String, der dem Subroutinenaufruf .
. 8146 8512    STA 12       direkt folgt, an das Display aus. .
. 8148 68      PLA          Mit $EA schließt der Text, das .
. 8149 8513    STA 13       aufrufende Programm wird hinter .
. 814B E612    INC 12       $EA weitergeführt.              .
. 814D D002    BNE 8151                                     .
. 814F E613    INC 13                                       .
. 8151 B112    LDA (12),Y                                   .
. 8153 C9EA    CMP #EA                                      .
. 8155 F005    BEQ 815C                                     .
. 8157 20BCE9  JSR E9BC                                     .
. 815A 10EF    BPL 814B                                     .
. 815C 6C1200  JMP (0012)                                   .
```

```
. F000 2C11A4  BIT A411     Das ist eine Centronics-Routine, .
. F003 1022    BPL F027     die hier anstelle des AIM-Thermo- .
. F005 48      PHA          druckerprogramms steht. Die Aus- .
. F006 48      PHA          gabe erfolgt über Port B des .
. F007 A910    LDA #10      User-VIA, CB2 ist mit Strobe, CB1 .
. F009 2C0DA0  BIT A00D     mit Acknowledge verbunden.      .
. F00C F0FB    BEQ F009                                     .
. F00E 68      PLA                                          .
. F00F 297F    AND #7F                                      .
. F011 C940    CMP #40                                      .
. F013 D002    BNE F017                                     .
. F015 A97F    LDA #7F                                      .
. F017 C918    CMP #18                                      .
. F019 D002    BNE F01D                                     .
. F01B A91B    LDA #1B                                      .
. F01D C919    CMP #19                                      .
. F01F D002    BNE F023                                     .
. F021 A91F    LDA #1F                                      .
. F023 8D00A0  STA A000                                     .
. F026 68      PLA                                          .
. F027 60      RTS                                          .
```

```
. EFDA 2096FE  JSR FE96     Auch dieses Unterprogramm ist .
. EFDD C941    CMP #41      nicht im ursprünglichen AIM-Be- .
. EFDF 9002    BCC EFE3     triebssystem enthalten. Es dient .
. EFE1 29DF    AND #DF      als Filter für Kleinbuchstaben, .
. EFE3 60      RTS          die damit ebenfalls zur Befehls- .
.                           eingabe verwendet werden können. .
```

```
0000                *=$DE
00DE LCOUNT         *=*+1           ;TEMPORARY COUNTER
00DF NOWLN          *=*+2           ;CURRENT LINE
00E1 BOTLN          *=*+2           ;LAST ACTIVE, SO FAR
00E3 TEXT           *=*+2           ;LIMITS OF BUFFER (START)
00E5 END            *=*+2           ;LIMITS OF BUFFER (END)
00E7 SAVE           *=*+4           ;TEMPORARY STORAGE
00EB SAVELN         *=*+2           ;SAVE CURRENT LINE
00ED CR             =$0D            ;CARRIAGE RETURN
00ED CT             =$06            ;CONTROL-F
00ED LNA            =26             ;NUMBER OF CHARACTERS TO PRINT OUT
00ED KEYF1          =$010C          ;USER VECTOR, F1, F2
00ED STRING         =$8143          ;OUTPUTS FOLLOWING ASCII-STRING, STOPS WITH $EA
00ED COPY           =$DCBD          ;COPY BLOCK, SUBROUTINE OF EXEDIT
00ED SCREEN         =$D800          ;SCREEN-EDITOR
00ED READ           =$EFDA          ;INPUT CHARACTER AND FILTER
00ED CRLF           =$EA13          ;CR TO DISPLAY
00ED OUTPR1         =$F005          ;PRINT CHARACTER (A CENTRONICS ROUTINE IN MY SYSTEM
00ED FIND           =$F81E          ;INPUT STRING AND FIND IT
00ED FC5            =$F849          ;FIND STRING
00ED TOPNO          =$F8BC          ;GO TO FIRST LINE
00ED PLNE           =$F727          ;DISPLAY LINE
00ED DO             =$F6F9          ;DISPLAY NEXT LINE
00ED DOWN           =$F709          ;ONE LINE DOWN, END?
00ED DOWN1          =$F713          ;ONE LINE DOWN, X=0K
00ED UP             =$F6D8          ;ONE LINE UP, TOP?
00ED UP1            =$F6E3          ;ONE LINE UP, X=OK
```

MAILMANAGING

```
00ED                *=$200
0200        A205    LDX #5
0202 LU     BDCA03  LDA TABUS,X     ;INSTALL SRCHD TO F1 AND SVNL TO F2
0205        9D0C01  STA KEYF1,X     ;F1=REENTER AFTER "E"
0208        CA      DEX             ;F2=REENTER AFTER "@"
0209        10F7    BPL LU
020B SVNL   A5DF    LDA NOWLN       ;SAVE TEXTPOINTER
020D        85EB    STA SAVELN
020F        A5E0    LDA NOWLN+1
0211        85EC    STA SAVELN+1
0213 START  A2FF    LDX #$FF        ;RESET STACK
0215        9A      TXS
0216        204803  JSR HTEXT
0219        20BCF8  JSR TOPNO       ;START AT TOP OF TEXT
021C SRCHD  2009F7  JSR DOWN        ;SEARCH NEXT ADDRESS DOWN
021F SRCH   20D402  JSR NUMBER      ;THE POSTNUMBER IS ALWAYS
0222        D0F8    BNE SRCHD       ;IN THE LAST LINE
0224 FOUND  20EA02  JSR UPLN3       ;IF FOUND, DISPLAY ADDRESS
0227        2027F7  JSR PLNE
022A L2     20F9F6  JSR DO
022D        C6DE    DEC LCOUNT
022F        10F9    BPL L2
0231 COM    20DAEF  JSR READ        ;WHAT COMMAND?
0234        2013EA  JSR CRLF
0237        C95A    CMP #'Z
0239        9008    BCC ADR1
023B        20EA02  JSR UPLN3       ;PRINT ALL ADDRESSES OF THE FILE
023E PRIALL 20F802  JSR PRIADR
0241        D0FB    BNE PRIALL
0243 ADR1   C954    CMP #'T
0245        F0CC    BEQ START       ;GO TO TOPLINE
```

```
0247         900D    BCC ADR2
0249         20D8F6  JSR UP
024C SRCHUP  20D8F6  JSR UP          ;SEARCH NEXT ADDRESS UP
024F         20D402  JSR NUMBER
0252         D0F8    BNE SRCHUP
0254         F0CE    BEQ FOUND
0256 ADR2    C951    CMP #'Q
0258         D00E    BNE ADR3
025A QUIT    A5EB    LDA SAVELN      ;RESTORE NOWLN AND
025C         85DF    STA NOWLN       ;GO TO SCREENEDITOR
025E         A5EC    LDA SAVELN+1
0260         85E0    STA NOWLN+1
0262         2000D8  JSR SCREEN      ;SCREEN IS A SUBROUTINE, RETURN WITH CNTR-Q
0265         4C0B02  JMP SVNL
0268 ADR3    9014    BCC ADR4
026A         A5EB    LDA SAVELN      ;SAVE BECOMES NEW POINTER
026C         85E7    STA SAVE
026E         A5EC    LDA SAVELN+1
0270         85E8    STA SAVE+1      ;(COMMAND R AND S)
0272         202D03  JSR CNTOUT      ;PRINT LETTER AND INSERT ADDRESS, WHEN
0275         20F502  JSR PRIAD       ;CNTR-F OCCURS TWO TIMES. PRINT UNTIL
0278         202D03  JSR CNTOUT      ;NEXT TWO CNTR-F OR TO END OF TEXT
027B JSD     4C1C02  JMP SRCHD
027E ADR4    C94E    CMP #'N
0280         D006    BNE ADR5
0282         202003  JSR FREELN      ;LINEFEED
0285         4C3102  JMP COM
0288 ADR5    9006    BCC ADR6
028A         20F502  JSR PRIAD       ;PRINT CURRENT ADDRESS, STEP TO NEXT
028D         4C1C02  JMP SRCHD       ;(COMMAND P)
0290 ADR6    C949    CMP #'I
0292         D009    BNE ADR7
0294         20BDDC  JSR COPY        ;COPY ADDRESS INTO LETTER
0297         2000D8  JSR SCREEN      ;THEN JUMP TO SCREEN-EDITOR
029A         4C5A02  JMP QUIT
029D ADR7    C947    CMP #'G
029F         D016    BNE ADR8
02A1         201EF8  JSR FIND        ;FIND ALL ADDRESSES WITH CHARACTER STRING
02A4 FA      20D402  JSR NUMBER      ;AND PRINT IT
02A7         F006    BEQ FA1
02A9         2009F7  JSR DOWN
02AC         4CA402  JMP FA
02AF FA1     20F502  JSR PRIAD
02B2         2049F8  JSR FC5
02B5         F0ED    BEQ FA
02B7 ADR8    9006    BCC ADR9
02B9         204803  JSR HTEXT       ;(MUST BE H)
02BC         4C1C02  JMP SRCHD
02BF ADR9    C945    CMP #'E
02C1         D006    BNE ADR10
02C3         2000D8  JSR SCREEN      ;GO TO SCREENEDITOR BUT
02C6         4C1C02  JMP SRCHD       ;DONT RESTORE NOWLN
02C9 ADR10   90B0    BCC JSD
02CB         201EF8  JSR FIND        ;FIND ADDRESS WITH CHARACTER STRING
02CE         2013EA  JSR CRLF
02D1         4C1F02  JMP SRCH
-------------------------------------------------------------
SUBROTINES

02D4 NUMBER  A003    LDY #3          ;IF 4 NUMBERS OCCUR AT THE LEFT MARGIN,
02D6 NUMB1   B1DF    LDA (NOWLN),Y   ;THE ZERO-FLAG IS SET
```

```
02D8        D003    BNE NUMB2
02DA        4C1302  JMP START       ;END OF TEXT
02DD NUMB2  C930    CMP #'0
02DF        9007    BCC NUMB3
02E1        C93A    CMP #':
02E3        B003    BCS NUMB3
02E5        88      DEY
02E6        10EE    BPL NUMB1
02E8 NUMB3  C8      INY             ;INY DETERMINES THE ZERO-FLAG
02E9        60      RTS

02EA UPLN3  A202    LDX #2          ;3 LINES UP
02EC        86DE    STX LCOUNT
02EE L1     20E3F6  JSR UP1
02F1        CA      DEX
02F2        10FA    BPL L1
02F4        60      RTS

02F5 PRIAD  20EA02  JSR UPLN3
02F8 PRIADR A203    LDX #3          ;PRINT ADDRESS
02FA        86DE    STX LCOUNT
02FC PRINTL A000    LDY #0          ;PRINT CURRENT LINE
02FE PR1    B1DF    LDA (NOWLN),Y
0300        D003    BNE PR2
0302        4C1302  JMP START       ;END OF TEXT
0305 PR2    2005F0  JSR OUTPR1
0308        C90D    CMP #CR
030A        F007    BEQ PR3
030C        C8      INY
030D        C01B    CPY #LNA+1      ;PRINT ONLY FIRST 26 CHARACTER OF LINE
030F        D0ED    BNE PR1
0311        A90D    LDA #CR         ;ONE CR FOR PRINTER
0313 PR3    2005F0  JSR OUTPR1
0316        2013F7  JSR DOWN1
0319        C6DE    DEC LCOUNT
031B        10DF    BPL PRINTL
031D        202003  JSR FREELN      ;TWO CR FOR PRINTER
0320 FREELN A920    LDA #' '
0322        2005F0  JSR OUTPR1
0325        A90D    LDA #CR
0327        4C05F0  JMP OUTPR1

032A NXTSER 203E03  JSR SEROUT      ;PRINT TEXT AND PROVE CHARACTERS
032D CNTOUT A000    LDY #0          ;STOP, IF TWO CNTR-F OCCUR
032F        B1E7    LDA (SAVE),Y
0331        C906    CMP #CT         ;CNTR-F ?
0333        D0F5    BNE NXTSER
0335        204103  JSR INCSAV
0338        B1E7    LDA (SAVE),Y
033A        C906    CMP #CT         ;SECOND CNTR-F ?
033C        D0EC    BNE NXTSER
033E SEROUT 2005F0  JSR OUTPR1      ;PRINT CHARACTER AND
0341 INCSAV E6E7    INC SAVE        ;INCREMENT POINTER SAVE
0343        D002    BNE CNT
0345        E6E8    INC SAVE+1
0347 CNT    60      RTS

0348 HTEXT  204381  JSR STRING      ;DISPLAY HELP-MESSAGE
034B        0D      .BYT CR,'   *MAILMANAGING*',CR
034C        2020
035D        0D
035E        443D    .BYT 'D=DOWN E/U=SREEN',' F=FIND'
```

```
036E        2046
0375        2047    .BYT ' G=FINDALL N=LF',CR,'P=PRINT'
0384        0D
0385        503D
038C        2053    .BYT ' S=SERIES T=TOP'
039B        2055    .BYT ' U=UP Z=PRINTALL',CR
03AB        0D
03AC        493D    .BYT 'I=COPY H=HELP','  )CONTROL-F(',CR,CR,$EA
03B9        2020
03C6        0D
03C7        0D
03C8        EA
03C9        60      RTS

03CA TABUS  4C1C02 JMP SRCHD
03CD        4C0B02 JMP SVNL
03D0               .END
ERRORS=0000
```

Horst Brettin, 1000 Berlin 44

Editieren der Directory

Das nachfolgende Programm gestattet es, die Reihenfolge der Einträge im Directory der CBM-Floppies zu verändern. Dazu werden zunächst die Dateinamen und ihre Zeiger in Arrays des Computers geladen und auf dem Bildschirm in der ursprünglichen Reihenfolge zur Anzeige gebracht. Hier kann jetzt editiert werden. Die aktuelle Zeile/Eintrag ist revers hinterlegt. Es gibt die Cursorbewegungen abwärts (Inhalt der Variablen in Zeile 3170), aufwärts (Zeile 3180), Home (3190) und Clear Home (3200). Eine angesteuerte Zeile kann mit 'I' mit einer spitzen Klammer kenntlich gemacht werden, daß sie zum Vertauschen vorgesehen ist. Man bewegt mit dem Cursor weiter auf die Stelle, hinter der der kenntlich gemachte Eintrag gerückt werden soll und betätigt die Taste 'T'. Sofort wird die Reihenfolge umgestellt, zunächst aber nur im Computer. Erst wenn man die Hochpfeil-Taste (Zeile 3210) betätigt, wird das Directory in der neuen Reihenfolge aber mit alten Zeigern auf Spuren und Sektoren auf die Diskette zurückgeschrieben.

Ein Editieren erspart ein oft mühsames umkopieren, denn oft möchte man entweder mehr Übersicht und Ordnung bei den Einträgen haben oder man möchte die neueste Version eines Programmes als ersten Eintrag stehen haben, damit man von der Diskette mit 'Shift/RUN' starten kann.

```
1000 REM D I R - E D I T          2.1
1010 REM ---------------
1020 REM (C) 06.07.84  BY H.BRETTIN
1030 GOTO 5000
2000 PRINT" PROGRAMM-/DATEINAMEN UNSORTIERT
2010 PRINT#15,"U1";2;DR%;SP%;SE%;SP(SN)=SP%:SE(SN)=SE%:GOSUB 9000
2020 GET#2,A$,P$:SP%=ASC(A$+NOP$)
2030 SE%=ASC(P$+NOP$)
2040 FOR I=0 TO 7
2050 PRINT#15,"B-P";2;2+I*32
2060 P$="":FORJ=1TO30:GET#2,A$:P$=P$+LEFT$(A$+NOP$,1):NEXT
2080 IF ASC(P$) THEN N$(N)=P$:N=N+1:PRINT MID$(P$,4,16)
2090 NEXT
2100 IF SP% THEN SN=SN+1:GOTO2010
2110 SE(SN+1)=255
2400 FOR J=N TO 8*(SN+1)+1:N$(J)=NOP$:NEXT
2500 N=N-1:RETURN
2999
3000 F=20:PRINT""TAB(F)" DIRECTORY EDITOR
3010 PRINT TAB(F)"(C) BY H. BRETTIN
```

```
3020 PRINT TAB(F)"I = SETZE <
3030 PRINT TAB(F)"T = TRANSFER NACH <
3040 PRINT TAB(F)"^ = EDITIEREN
3050 PRINT TAB(F+5)"BEENDEN
3090 F=-1
3100 I=0
3110 J=I-15:J=-J*(J>0)
3120 EN=J+23:IF EN>N THEN EN=N:J=EN-23:IF J<0 THEN J=0
3130 PRINT"";
3140 FORJ=JTOEN:PRINTLEFT$("",-(I=J))MID$(N$(J),4,16)MID$("< ",2+(J=F)):NEXT
3150 POKE 151,0
3160 GET A$:IF A$="" GOTO 3160
3170 IF A$="" GOTO 3330
3180 IF A$="" THEN I=I+(I>0):GOTO 3110
3190 IF A$="" GOTO 3100
3200 IF A$="" THEN I=N:GOTO 3110
3210 IF A$="^" THEN RETURN
3220 IF A$="I" THEN F=I:GOTO 3110
3230 IF A$<>"T" GOTO 3160
3240 IF I=F GOTO 3110
3250 A$=N$(I)
3260 IF I>F GOTO 3290
3270 FOR J=I TO F-1:N$(J)=N$(J+1):NEXT
3280 N$(F)=A$:GOTO 3110
3290 IF I=F+1 GOTO 3110
3300 F=F+1
3310 FOR J=I-1 TO F STEP-1:N$(J+1)=N$(J):NEXT
3320 N$(F)=A$
3330 I=I-(I<N):GOTO 3110
3999 :
4000 N=0:FORJ=0TOSN:PRINT#15,"B-P";2;0:PRINT#2,CHR$(SP(J+1));CHR$(SE(J+1))
4010 FORI=0TO7:PRINT#15,"B-P";2;2+32*I:PRINT#2,N$(N);:N=N+1
4020 NEXT:PRINT#15,"U2";2;DR%;SP(J);SE(J):GOSUB 9000
4030 NEXT J
4040 RETURN
5000 DIM A$,J,P$,I,N$(224),SP(29),SE(29):NOP$=CHR$(0)
5010 PRINT "    EDITIEREN DER DIRECTORY "
5030 PRINT"     0  FLOPPY 3040/4040/1540/1541 (ETC)"
5040 PRINT"     1  FLOPPY 8050/8250
5050 GET A$:IF A$=""THEN5050
5060 F=ASC(A$)-48:IF F<F*F THEN 5050
5070 PRINT"     DRIVE ? ";
5080 GET A$:IF A$=""THEN5080
5090 DR%=ASC( A$)-48:IFDR%<DR%^2THEN5080
5100 PRINTDR%:N=0:SN=0:SP%=18:SE%=1:IF F THEN SP%=39
5110 OPEN15,8,15,"I"+STR$(DR%):GOSUB 9000
5120 OPEN2,8,2,"#":GOSUB 9000
5130 GOSUB 2000:REM DIR LESEN
5140 GOSUB 3000:REM  "  EDITIEREN
5150 GOSUB 4000:REM  "  SCHREIBEN
5160 CLOSE 2:CLOSE 15
5170 END
9000 INPUT#15,F,F$,SP,SE:IF F=0 THEN RETURN
9010 PRINT F;F$;SP;SE
9020 CLOSE 15
READY.
```

RLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRL

Kleinanzeigen

Ersatzteile für AIM 65: Thermodrucker St. DM 100,--, Displayplatine DM 200,--, voll bestückt, solange Vorrat reicht. FM-Elektronik, Brüß, Tel. 030 - 781 40 11.

RM 65-Module gesucht. Radeloff, ab 19 h Tel. 04101 - 20 88 93.

SPS-Manual für AIM 65 (engl.): Regge, Tel. 0421 - 71 114.

Martin Albrecht, 4350 Recklinghausen

Fehler im ASSMOS 1.0

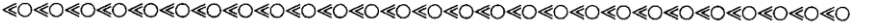
Herr Wolfgang Masarie aus Simbach hat freundlicherweise darauf hingewiesen, daß der ASSMOS 1.0 aus Heft Nr. 36 den neuen Befehl JMP (Indirekt,x) nicht korrekt verarbeitet, wenn die indirekte Adresse beim ersten Assemblerdurchlauf noch nicht definiert ist, wenn sie also erst hinter dem JMP-Befehl erklärt wird. In diesem Fall wird eine Zeropage-Adresse erzeugt. Die Korrektur erfordert einige Änderungen, die im nachfolgenden Listing aufgeführt sind. Die dabei entstehenden Adreßverschiebungen machen den ASSMOS noch unübersichtlicher. Deshalb sei darauf hingewiesen, daß dieser Assembler auch als EPROM über die Autoren erhältlich ist: Dipl.-Ing. Rüdiger Wollenberg, Stockumer Str. 234, 4600 Dortmund, Tel. 0231 - 75 34 77.

```
0000          ;ASSMOS 1.0 Korrektur: verarbeitet jetzt JMP (IND,X)
0000          ;korrekt, wenn die Adresse 'IND' hinter dem 'JMP'-Befehl
0000          ;definiert wurde.

D557                  .OPT LIS
D557 H40      A50F    LDA JOPBAS
D559          C94C    CMP #76         ;JMP ?
D55B          F004    BEQ H40Z        ;JMP, dann normal weiter
D55D          A51D    LDA JNOPV
D55F          D06D    BNE H400H       ;undef. Symbol, Sonderbehandlung
D561 H40Z     A510    LDA JOPTEM

D5C8                  .OPT LIS
D5C8 H400     A902    LDA #2
D5CA          8530    STA JOPLEN
D5CC          A527    LDA IEXP
D5CE H400H    D013    BNE H41         ;neue Einsprungmarke f. undef. Symbol
D5D0          A901    LDA #1
D5D2 JMP2B    8530    STA JOPLEN      ;Einsprung für JMP

D5EE                  .OPT LIS
D5EE H45      A8      TAY
D5EF          88      DEY
D5F0          B958DE LDA KLUDG-1,Y
D5F3          ;Eintrag in KLUDG = 0 bedeutet unerlaubten Operanden
D5F3          F007    BEQ H49
D5F5          6510    ADC JOPTEM      ;immer C=0, CLC entfernt (1 Byte Platz)
D5F7          20E4D8 JSR KLTBLU
D5FA          1029    BPL H46
D5FC H49      A51D    LDA JNOPV
D5FE          F015    BEQ H48
D600          A530    LDA JOPLEN      ;JOPLEN in Accu
D602          C902    CMP #2
D604          F005    BEQ H206        ;wenn JOPLEN = 2
D606          A918    LDA #$18
D608 H205     4C7DD6 JMP H900A
D60B          ;hier 2 Bytes Platz: ersetze LDA #1, STA JOPLEN durch
D60B          ;DEC JOPLEN !
D60B H206     C630    DEC JOPLEN      ;JOPLEN auf 1
```



Aus der Branche

Single-Chip Microcomputer Data von Motorola: Dieses etwa 1200seitige Handbuch für den Designer enthält alle Angaben für den Aufbau von 1 Chip-Systemen samt Peripherie mit M3870, M6801, M6804 und M6805 Familien-Bausteinen. Es sind jeweils die kompletten Datenblätter enthalten, die alleine 1060 Seiten füllen. Das Buch ist 1984 erschienen und trägt die Nummer B046. - Das MC68020 32-Bit Microprocessor User's Manual wurde als Neuerscheinung im Heftinneren erwähnt.

VALVO: Mikroprozessor-System S68000 - Datenbuch. Das in englischer Sprache abgefaßte Buch enthält im wensentlichen die Datenblätter zu den integrierten Schaltungen der 68xxx-Familie, ferner Modulbeschreibungen für VMEbus-Karten und für das Entwicklungssystem PMDS. Verlag Boysen + Maasch, Hamburg, ISBN3-87095-175-3.

VALVO: Der Herausgeber erhielt zahlreiche Pressemitteilungen für Asynchrone Datenübertragungsschaltung SCN 2641 (Mitt. 4058) für bis 1 Mbps/DRAM-Controller N2964B (Mitt. 4057) für Speicher mit 16 und 64K DRAMs/Schaltungen NE 5080 und 5081 für schnelle FSK-Modems bis 2 Mbaud (Mitt. 4062)/2-Chip Videotext-Decoder für Computer Controlled Teletext (Mitt. 4084).

Commodore wird dem Vernehmen nach seinen IBM-kompatiblen PC ab 26. Jan. 1985 an seine Händler ausliefern. Preis mit 256KB Arbeitsspeicher, Grafik, Tastatur und 2x 340 KB Laufwerken DM 4.995,-- + MWSt.. Auch eine Druckerschnittstelle ist inbegriffen.

AIM 65 Mikrocomputer und RM 65-Module werden für den Alleinvertrieb in Europa künftig von der Fa. System Elektronik Ihlemann GmbH in Braunschweig hergestellt werden.

Bücher

Stanka,S. und Lösch,S.: Die C-Sprache. te-wi-Verlag, München 1984, 288 Seiten, ISBN 3-921 803-28-4, DM 59,--. Das Buch ist als Lehrbuch für den Einsteiger und mit seinem Kapitel über die verschiedenen Bibliotheksfunktionen als Nachschlagewerk konzipiert. Es hat nach der Einleitung folgende Hauptkapitel: Architektur von C-Programmen, Lesen, Schreiben, Verstehen, Funktionen, Programme - keine 'black box', Schützen, Suchen, Kontrollieren und schließlich Bibliotheksfunktionen.Im Text finden wir viele Schemazeichnungen und Ablaufdiagramme, Beschreibungen der wichtigsten Systemfunktionen und Hinweise auf die Anwendungsmöglichkeiten der C-Logik. Der Inhalt des Buches ist drucktechnisch sorgfältig aufgemacht und der Text ist knapp und klar. Das Buch darf damit als Arbeitsbasis für die immer wichter werdende Sprache C empfohlen werden.

Duff, Charles B.: MACINTOSH Anwenderhandbuch. Verlag McGraw-Hill, Hamburg 1984, 156 S., ISBN 3-89028-002-6, DM 39,80. Der Autor war Mitglied im ersten Entwicklungsteam für diesen neuen und erfolgreichen Rechner. In den ersten Kapitelnn wird die Hantierung des Macintosh ausführlich abgehandelt und auch die Hardware erklärt. Anwendungsmöglichkeiten und Beispiele werden gegeben für MacWrite, MacPaint und Macplan, auch für Datenfernübertragung. Im Anhang findet man auch einen detaillierten Vergleich mit dem IBM-PC. Das Buch enthält zahlreiche Schemazeichnungen und Abbildungen, die der Ausgabe auf dem Bildschirm entsprechen, so daß ohne überflüssige Worte eine 'gerätenahe' Einführung und Übersicht gegeben wird, die die Urteilsbildung vor einer Entscheidung verbessert und über die erste Inbetriebnahme hinweghilft.

Schumny, H. (Hrsg.): Mikrocomputer Jahrbuch 1985 - Tendenzen, Anwendungen, Software, Daten. Vieweg-Verlag, Wiesbaden 1984, 366 S. ISBN 3-528-04315-6, DM 38,--. Im Oktober erschien das sechste dieser Jahrbücher mit einem Fachteil von 30 Aufsätzen, 20 Programmen, einer Datensammlung für 2000 Produkte und mit 1360 Adressen aus der Branche. Die Fülle des Materials läßt eine Aufzählung nicht zu, man sollte im Inhaltsverzeichnis und im Text blättern, um die interessierenden Beiträge zu Taschenrechnern, PC-Software-Themen, Mikrocomputern und Programmen zu prüfen. Auf jeden Fall gibt das Buch von Jahr zu Jahr wieder Orientierung zu aktuellen Themen, Hilfen und eine einmalige Datensammlung.

Spiel und Spaß mit dem Commodore 64 und **Computer für Kinder** sind zwei neue Titel bei McGraw-Hill (ISBN 3-89028-014-5, 172 Seiten, DM 29,80) und bei te-wi (ISBN 3-9211803-40-3, 92 S. DM 29,50). Ersteres enthält 35 komplett gelistete Programme, das andere wendet sich an interessierte 8-13 jährige, hat die Form einer Fibel und soll wohl eine Marktlücke schließen.

Computer Programmverzeichnis 84/85. Video Partner Verlag, Hamburg 1984, 112 S., ISBN 3-923220-10-3, DM 14,80. Für Heimcomputer und Video-Spiele sind mehr als 2700 Programme gelistet, und zwar in einer Unterteilung nach Rechnertypen. Es sind nicht nur Spiele genannt, sondern auch ernsthafte Programme. Jeder Eintrag ist kurz mit einer Programmbeschreibung und der Lieferquelle versehen, so daß man hier bei Bedarf eine nützliche Übersicht findet.

Zum IBM-PC waren hier drei verlagsneue Zugänge zu verzeichnen: **IBM-PC COMPACT** ist eine Faltkarte, die das Arbeiten am Computer erleichtert, sie enthält alle BASIC-Befehle, Funktionen und Variablen auf einen Blick (DM 9,80 ISBN3-89028-007-2, bei McGraw Hill, Hamburg). Aus diesem Verlag kommt ebenfalls das **'IBMPersonal Computer Anwenderhandbuch'** (3-89028-011-0, 370 S. DM 57,--) aus der Feder von Lon Poole und vermittelt Grundwissen zur Hantierung und Programmierung in BASIC. In seinem Aufbau und in seinen Darstellungen knüpft es an ähnliche Bücher aus diesem Verlag an, z.B. für CBM, C-64 usw. - Der te-wi Verlag brachte das **'IBM-PC Handbuch'** heruas (ISBN 3-921 803-22-5, 416 S. DM 59,--). Auch diese Übersetzung aus dem Amerikanischen dient vor allem der ausführlichen Einführung.

Editorial

Die innenpolitischen Ereignisse des auslaufenden Jahres haben sehr enttäuscht. Gemeint sind die Verwicklung von Parteien und prominenten Politikern in die Spendenaffäre, die unterbliebenen Maßnahmen, die man heute schon für den Schutz der Umwelt hätte anfassen können und die Arbeitslosigkeit, die sich zahlenmäßig etwa in der Höhe des Vorjahres hält, die tatsächlich aber zugenommen hat, wenn man an die Arbeitsbeschaffungsmaßnahmen und an die Tatsache denkt, daß einige hunderttausend Ausländer mit viel Geld repatriert wurden, die sonst die Zahl der Arbeitssuchenden noch höher hätten ausfallen lassen. Man muß wohl das Gesundbeten und Verharmlosen von den Tatsachen unterscheiden.

Gleichwohl: Der Herausgeber wünscht den Lesern und ihren Familien ein friedliches Neues Jahr, im Inneren wie auch im Äußeren! Und natürlich auch Gesundheit und Erfolg. Er darf das mit dem Dank für die hohe Lesertreue verbinden und die uns alle weiterführende Mitarbeit der vielen freien Autoren.

Beiträge zu dieser Zeitschrift sind auch 1985 sehr willkommen. Die Einsatzgebiete der Computer und ihre Programme nehmen so rapide zu, daß schon viele dazu beitragen müssen, um einen halbwegs guten Überblick über die Möglichkeiten und Lösungen zu behalten. - Dem Überblick des Lesers ist das Inhaltsverzeichnis in der Heftmitte zugedacht, das gleich drei Jahrgänge abdeckt, um möglichst das mühsame Nachschlagen zu vermeiden. Fotokopieren Sie es ggfs. und geben es auch empfehlend an andere Interessierte weiter!

Das neue Heft ist einmal wieder an der vordersten Front. Mit dem 68020 von Motorola dämmern Computer herauf, die enorm leistungsfähig sind. Wegen der Dauer des Designs für Hard- und Software und auch wegen des noch hohen Preises der Chips wird es sicher noch einige Jahre dauern, bis wir die nächste Computergeneration in Betrieb haben.

Die Prozessoren 65802 und 65816 kommen zwar etwas spät in die 65xxx-Familie. Sie sind in diesem Heft bereits ausführlich dokumentiert, ehe sie endlich 1985 am Markt erscheinen. - Es wird sich nach der Betrachtung der hier abgedruckten Beispiele und Beschreibungen für die große Gemeinde der 65xx-Betreiber lohnen, die Umrüstungsmöglichkeit zu prüfen, um damit in eine bessere Programmierumgebung zu gelangen. Neue CPU's sind prinzipiell für Sprachen und Maschinenprogrammierung besser geeignet, weil man das von der Architektur her unterstützt. Sie haben für den 65xx-Betreiber zweifellos auch den Vorteil, daß er mit seinen erworbenen Kenntnissen weiterarbeiten kann, ohne total umlernen zu müssen. - Der Artikel über die neuen 65xxx sollte unbedingt noch in das vorliegende Heft. Und da ging es dem Herausgeber nicht anders und nicht besser als vielen Lesern: Der dynamische Erweiterungsspeicher des AIM 65 wurde so unzuverlässig, daß er ausgemustert werden mußte. Eine Reparatur in sinnvoller Zeit war nicht möglich, so wurde auf CMOS statische RAMS umgerüstet. Bei den Signalen des AIM hatte das auch Anfangsschwierigkeiten mit den vorgesehenen Busverstärkern, die dann schließlich fortgelassen wurden. In der Summe ging leider viel Zeit verloren.

Im neuen Jahr werden wir uns weiter um die 658xx und ihre Möglichkeiten kümmern, vielleicht auch schon über einen entsprechenden Computer berichten können, ferner jetzt mehr auch um 68xxx, die im Kommen sind und werden auch einen Blick auf NS 32xxx werfen, eine auch sehr interessante Familie.

Peter Arps, 2000 Hamburg 73

Einkommensteuer-Berechnung für 1984

An diesem Programm, das zuletzt im Dezember 1983 (Heft 34) ergänzt wurde, sind diesmal nur wenige Änderungen vorzunehmen:

1. Aenderung der Bemessungsgrundlage der Rentenversicherung von 60.000 DM auf 62.400 DM

```
 740 IF X>62400 THEN X=62400
 850 DEF FNY(Y)=-(Y*(Y=<62400)+62400*(Y>62400))
3850 X=AL(I)-600:IF X>62400 THEN X=62400
3860 Y=AL(J)-600:IF Y>62400 THEN Y=62400
```

2. Beruecksichtigung der geaenderten Freibetraege fuer die Ausbildung (1200/2100/900 DM statt 2400/4200/1800 DM)

```
1510 IF X$="N" THEN Y=900-Z:GOTO 1560
1540 Y=1200-Z:GOTO 1560
1550 Y=2100-Z
```

3. Anpassung der Jahreszahlen

```
1780 PRINT"Verlust-Vortrag/-Ruecktrag 1979-1986":GOSUB 2160
2740  INPUT"2.1.1920 geboren (j=ja/n=nein) n(cl cl cl)";X$:IF
X$="N" THEN 2770
3280  X=X(14):IF X>0 THEN PRINT#4:X$="-  Verlust-Vor/Ruecktr.
1979-1986": GOSUB 2930
```

Peter Engels, 5308 Rheinbach

C-64 Disk-Monitor

Das Programm entstand in Anlehnung an den Disk-Monitor, der im Data Becker Floppybuch veröffentlicht wurde. Dieser läßt noch einige Wünsche offen. Die jetzige Version ist als BASIC-Programm ladbar und kann mit RUN gestartet werden. Es wurden noch einige Zusatzfunktionen wie z.B. Druckerausgabe und Cataloganzeige verwirklicht..

Nach dem Starten des Programms mit RUN erscheint nach der Programm-Meldung ein Prompt (>), und es wird die Eingabe eines Befehls mit evtl. folgender Wertangabe erwartet. Der Wert wird durch ein Space vom Befehl getrennt. Er ist grundsätzlich hexadezimal einzugeben. Die gesamte Eingabe wird mit Return abgeschlossen. Bei einem erkannten Fehler wird in die Eingabeschleife zurückgesprungen. Es stehen folgende 12 Befehle zur Verfügung, in denen die Parameter folgende Bedeutung haben:

(TT =Track SS =Sektor XX =Byte YY =Byte (CR) =Return)

R TT SS (CR)	Einlesen des Sektors SS vom Track TT in C-64 Buffer. Bei erfolgtem Lesen werden die Bytes hex 00-7F dargestellt.
W TT SS (CR)	Schreibt Buffer auf Track/Sektor.
@ (CR)	Schreiben des Disketten-Fehlerkanals.
@ TEXT (CR)	Sendet TEXT auf Kommandokanal 15 an Floppy.
: XX YY	Ändern des Bufferinhaltes ab Adresse XX. Durch Cursortasten kann der komplette angezeigte Bereich wie beim Monitor editiert werden.
$ (CR)	Ausgabe des Disk-Catalogs auf den Schirm. Mit Space kann die Ausgabe angehalten werden bzw. fortgeführt werden.
N (CR)	Einlesen des nächsten Sektors, der zum momentan bearbeiteten File gehört. Ist der letzte Sektor bereits gelesen, erfolgt die Meldung FILE-ENDE.
M XX YY (CR)	Ausgabe der Bytes von XX bis YY.
1 (CR)	Anzeige der Bytes hex 00 bis 7F.
2 (CR)	Anzeige der Bytes hex 80 bis FF.
I (CR)	Anzeige des zuletzt gelesenen bzw. beschriebenen Track/Sektors in dezimal.

D XX (CR)	Ausgabegerät wird auf XX gesetzt. Somit kann der Katalog bzw. der Sektor auf dem Drucker festgehalten werden.
D (CR)	Der D-Befehl ohne Parameter setzt die Ausgabe auf den Schirm zurück.
X (CR)	Rückkehr zu BASIC.

Software

Die Eingaberoutine wartet auf die Eingabe von Return und sucht dann das zuerst eingegebene Zeichen in der Tabelle NCMDS. Wird das Zeichen als Befehl erkannt, dann wird die Befehlsadresse-1 aus der Tabelle ADRH gelesen und auf den Stack gelegt. Durch ein nachfolgendes RTS wird die Befehlsroutine dann angesprungen.

Die Lese- bzw. Schreibbefehle werden mittels U1 bzw. U2-Befehl an die Floppy gesendet. Beim Lesen eines Sektors werden die Bytes im C-64 RAM ab Buffer gelegt, wo sie ebenfalls editiert werden können. Beim Schreiben eines Sektors wird der Buffer dann wieder auf die Diskette gebracht.

Der Catalog-Befehl eröffnet das Directory-File auf der Floppy und gibt dann Zeichen für Zeichen auf den Schirm aus. Der D-Befehl funktioniert, wie es der CMD-Befehl in BASIC macht, d.h. alle Ausgaben vom Rechner gehen nun auf das angesprochene Gerät. Somit ist es leicht möglich, die Verteilung eines Files auf der Diskette mit dem Drucker festzuhalten. - Der N-Befehl benutzt die Sektorbytes 0 und 1 dazu, den nächsten Sektor aufzufinden, der zum File gehört: Byte 0 gibt den folgenden Track und Byte 1 den Sektor an. Beim letzten Sektor hat Byte 0 den Wert 00, und das Programm gibt die Meldung FILE-ENDE aus.

Beim @-Befehl wird der Kommandokanal 15 der Floppy geöffnet und der folgende Text an die Floppy gesendet. Somit ist es z.B. möglich, Befehle wie SCRATCH, VALIDATE oder NEW an die Floppy zu senden. - Mit dem Disk-Monitor ist es nun möglich, den Namen der Diskette oder deren ID zu ändern oder einfach nachzusehen, wie ein Assembler oder Textprogramm seine Files auf Disk ablegt.

```
TRACK/SECTOR (DEZ) : 18 01

>:00  00 FF 82 11 00 44 49 53 ..... DIS
>:08  4B 2D 4D 4F 4E 20 53 4F K-MON SO
>:10  55 52 43 45 A0 00 00 00 URCE ....
>:18  00 00 00 00 00 00 23 00 ......#.
>:20  00 00 82 13 00 44 49 53 ..... DIS
>:28  4B 2D 4D 4F 4E 20 58 52 K-MON XR
>:30  4F 47 52 41 4D 00 00 00 OGRAM ...
>:38  00 00 00 00 00 00 07 00 ........
>:40  00 00 00 00 00 00 00 00 ........
>:48  00 00 00 00 00 00 00 00 ........
>:50  00 00 00 00 00 00 00 00 ........
>:58  00 00 00 00 00 00 00 00 ........
>:60  00 00 00 00 00 00 00 00 ........
>:68  00 00 00 00 00 00 00 00 ........
>:70  00 00 00 00 00 00 00 00 ........
>:78  00 00 00 00 00 00 00 00 ........
>
```

```
0010                ; DISK MONITOR V2.4
0020                ; C-64 BASICVERSION
0030                ; BY P. ENGELS
0040 ;
0050                .BA $0801
0060                .MC $0801
0070                .OS
0080 ;
0090 CLSFIL         .DE $F642
0100 GET            .DE $FFE4
0110 INPUT          .DE $FFCF
0120 TALK           .DE $FFB4
0130 SECTALK        .DE $FF96
0140 IECIN          .DE $FFA5
0150 UNTALK         .DE $FFAB
```

```
                    0160 LISTEN     .DE $FFB1
                    0170 SECLIST    .DE $FF93
                    0180 IECOUT     .DE $FFA8
                    0190 UNLIST     .DE $FFAE
                    0200 BASOUT     .DE $FFD2
                    0210 OPEN       .DE $FFC0
                    0220 CLOSE      .DE $FFC3
                    0230 SETPAR     .DE $FFBA
                    0240 SETNAM     .DE $FFBD
                    0250 SENDNAM    .DE $F3D5
                    0260 CHKIN      .DE $FFC6
                    0270 CKOUT      .DE $FFC9
                    0280 CLRCH      .DE $FFCC
                    0290 CLALL      .DE $FFE7
                    0300 READY      .DE $E37B
                    0310 LNPRT      .DE $BDCD
                    0320 STATUS     .DE $90
                    0330 SA         .DE $B9
                    0340 FA         .DE $BA
                    0350 FNADR      .DE $BB
                    0360 FNLEN      .DE $B7
                    0370 TMPC       .DE $97
                    0380 QUOTFLG    .DE $D4
                    0390 ;
                    0400 PROMPT     .DE $3E             ; " > "
                    0410 NCMDS      .DE 12              ; ANZAHL BEFEHLE
                    0420 CR         .DE $0D
                    0430 QUOTE      .DE $22
                    0440 COUNT      .DE 8
                    0450 BUFF       .DE $200
                    0460 SAVX       .DE BUFF+1
                    0470 WRAP       .DE BUFF+2
                    0480 BAD        .DE BUFF+3
                    0490 VON        .DE BUFF+4
                    0500 BIS        .DE BUFF+5
                    0510 PRINTDEV   .DE BUFF+6
                    0520 ;
0801- 20 08         0530            .SI NEXTLINE        ; NAECHSTE BASICZEILE
0803- C0 07         0540            .BY $C0 $07 ; ZEILENNR.
0805- 9E            0550            .BY $9E             ; SYS-TOKEN
0806- 20 28 32      0560            .BY ' (2120) : '
0809- 31 32 30
080C- 29 20 3A
080F- 20
0810- 43 2D 36      0570            .BY 'C-64 DISKMONITOR
0813- 34 20 44
0816- 49 53 4B
0819- 4D 4F 4E
081C- 49 54 4F
081F- 52
0820- 00 00         0580 NEXTLINE   .BY 00 00
                    0590 ;
                    0600            .BA $0848
                    0610            .MC $0848
                    0620 ;
                    0630 ;
0848- A2 00         0640 INIT       LDX #0
084A- BD 3F 08      0650 MSGOUT     LDA MESSAGE,X
084D- F0 06         0660            BEQ CLEAR
084F- 20 D2 FF      0670            JSR BASOUT
0852- E8            0680            INX
0853- D0 F5         0690            BNE MSGOUT
                    0700 ;
0855- A9 00         0710 CLEAR      LDA #0
0857- AA            0720            TAX
0858- 9D 8F 0C      0730 CL1        STA BUFFER,X
085B- E8            0740            INX
085C- D0 FA         0750            BNE CL1
                    0760 ;
085E- A2 0D         0770 START      LDX #CR
0860- A9 3E         0780            LDA #PROMPT
0862- 20 35 09      0790            JSR WRTWO
```

```
0865- A9 00     0800            LDA #0
0867- 8D 02 02  0810            STA WRAP
086A- 20 7D 09  0820 ST1        JSR RDOC            ; EINGABEZEILE LESEN
086D- C9 3E     0830            CMP #PROMPT
086F- F0 F9     0840            BEQ ST1
0871- C9 14     0850            CMP #20             ; IGNORIERE BLANK
0873- F0 F5     0860            BEQ ST1
0875- A2 0B     0870 S0         LDX #NCMDS-1
0877- DD 19 0C  0880 S1         CMP CMDS,X
087A- D0 11     0890            BNE S2
087C- 8E 01 02  0900            STX SAVX
087F- 8A        0910            TXA                 ; X*2
0880- 0A        0920            ASL A
0881- AA        0930            TAX
0882- E8        0940            INX
0883- BD 25 0C  0950            LDA ADRH,X
0886- 48        0960            PHA
0887- CA        0970            DEX
0888- BD 25 0C  0980            LDA ADRH,X
088B- 48        0990            PHA
088C- 60        1000            RTS
                1010 ;
088D- CA        1020 S2         DEX
088E- 10 E7     1030            BPL S1
0890- 4C 5E 08  1040            JMP START
                1050 ;
                1060 ; UNTERPROGRAMM ZUR ANZEIGE
                1070 ; DES BUFFERINHALTS
                1080 ;
0893- 85 97     1090 DM         STA *TMPC
0895- 20 B8 08  1100 DM1        JSR SPACE
0898- B9 8F 0C  1110            LDA BUFFER,Y        ; BYTE AUS BUFFER
089B- 20 26 09  1120            JSR WRDB
089E- C8        1130            INY
089F- D0 03     1140            BNE DM2
08A1- EE 02 02  1150            INC WRAP
08A4- C6 97     1160 DM2        DEC *TMPC
08A6- D0 ED     1170            BNE DM1
08A8- 60        1180            RTS
                1190 ;
                1200 ;BYTE LESEN UND IN BUFFER ABLEGEN
                1210 ;
08A9- 20 48 09  1220 BYT        JSR RDDB
08AC- 90 03     1230            BCC BY3             ; SPACE ?
08AE- 99 8F 0C  1240            STA BUFFER,Y
08B1- C8        1250 BY3        INY
08B2- C6 97     1260            DEC *TMPC
08B4- 60        1270            RTS
                1280 ;
08B5- 20 B8 08  1290 SPAC2      JSR SPACE           ; 2 MAL SPACE
08B8- A9 20     1300 SPACE      LDA #' '    : 1 MAL SPACE

08BA- 2C        1310            .BY $2C
08BB- A9 0D     1320 CRLF       LDA #CR             ; CRLF
08BD- 4C D2 FF  1330            JMP BASOUT
                1340 ;
08C0- A0 00     1350 DSPLYM     LDY #0
08C2- 8C 04 02  1360            STY VON
08C5- 88        1370            DEY
08C6- 8C 05 02  1380            STY BIS
08C9- 20 CF FF  1390            JSR INPUT
08CC- C9 0D     1400            CMP #CR
08CE- F0 17     1410            BEQ DSP1
08D0- 20 48 09  1420            JSR RDDB            ; START-ADR  LESEN
08D3- 90 12     1430            BCC DSP1
08D5- 8D 04 02  1440            STA VON
08D8- 20 CF FF  1450            JSR INPUT
08DB- C9 0D     1460            CMP #CR
08DD- F0 08     1470            BEQ DSP1
08DF- 20 48 09  1480            JSR RDDB            ; END-ADR  LESEN
08E2- 90 03     1490            BCC DSP1
08E4- 8D 05 02  1500            STA BIS
```

Der Abdruck des Programmes wird im Folgeheft fortgesetzt.

Assembler

R65C02-Assembler

für AIM 65 und kompatible Systeme. 2-Pass-Assembler für den kompletten (!) Befehlssatz mit zusätzlichem Komfort. Assemblerliste formatiert, Assemblierung mit Offset für virtuelle Speicherbereiche. Der Assembler läuft auf der herkömmlichen 6502. - 2 EPROMs für Speicherbereiche Ihrer Wahl DM 195,--

6805/68705 Cross-Assembler

für AIM 65 und kompatible Systeme. 2-Pass-Assembler mit allem gewohnten Komfort und 6502-Syntax. Erzeugt aus den Mnemonics von Motorola 6805-Code. 2 EPROMs wie vor DM 280,--

6805/68705-Assembler unter FORTH

wie in Heft 35 beschrieben: 1-Pass-Assembler mit Verarbeitung von Symbolen und Labeln. Codeablage für virtuelle Speicherräume. 2 EPROMs für AIM 65 DM 100,--

Mathe-ROM für 6502

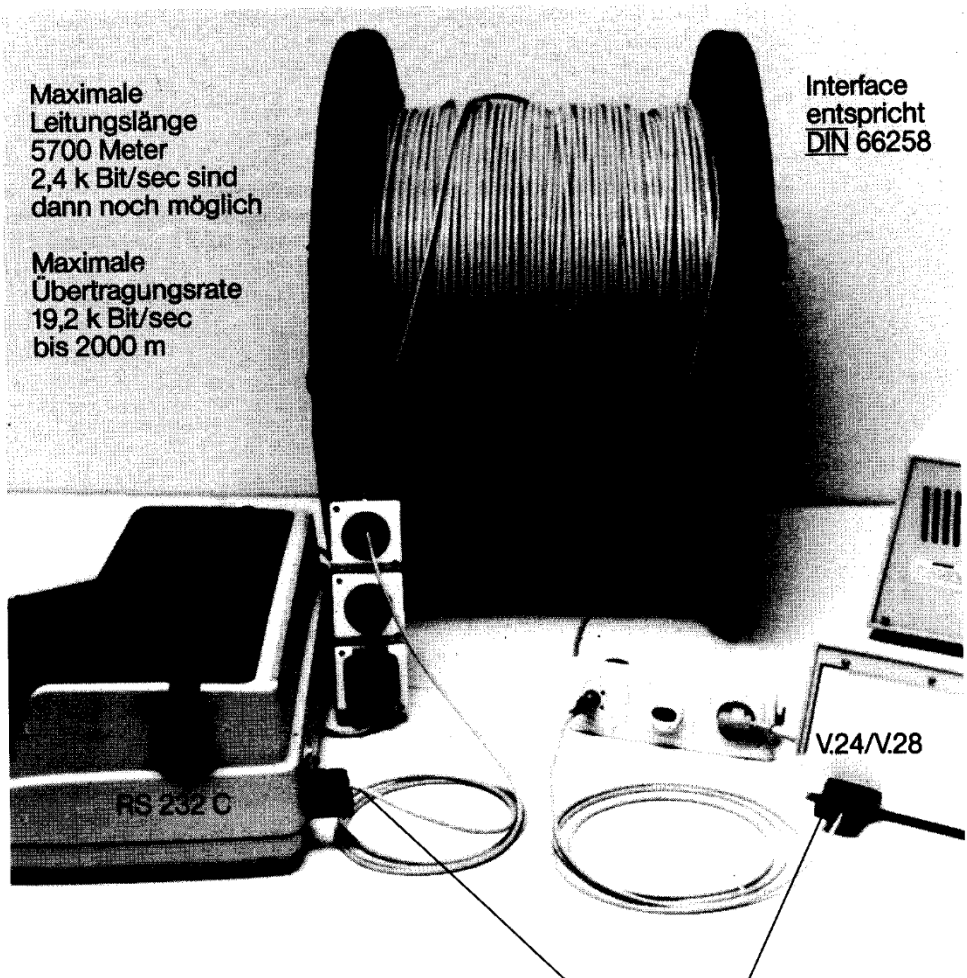
Implementierung nach Peter Rix (s. Hefte 28/29). Fließkommaarithmetik und höhere mathematische Funktionen wie in Microsoft-BASIC für AIM-65-FORTH und für jedes 6502-Assemblerprogramm (20 S. Dokumentation mit Einsprungspunkten und Argumenten), für Sockel $D000 DM 124,30

Assembler für 65802/816, R65C02 und 6502

ab Mitte Januar 1985 lieferbar. 2 Pass Assembler mit Sprungleiste für E/A, geschrieben im 6502 Code, damit auf vorhandenen Computern einsetzbar. Generierbar für beliebige Adreßbereiche, auch für die Variablen des Assemblers. 2 EPROMs. Preis und Daten auf Anfrage.

Roland Löhr, Hansdorfer Str. 4, D-2070 Ahrensburg
Tel.: 04 102 - 55 816

Suchen Sie eine preiswerte Alternative zur Dezentralisierung Ihrer Datenverarbeitung? ... dann ist 20 mA Current Loop die Lösung für Ihr lokales Punkt zu Punkt Netzwerk!



V.24/V.28 oder RS 232 C ↔ 20 mA Konverter
eingebaut in einem serienmäßigen Subminiatur „D" Steckergehäuse

Wir beraten Sie und planen Ihr lokales Netzwerk!
Fordern Sie bitte weiteres Informationsmaterial an.

DBGM: G 8331 081.9
G 8336 080.8

INGENIEURBÜRO
STECKER

5000 Köln 60 (Niehl)
Postfach 60 07 66
Delmenhorster Str. 20
Tel. (02 21) 712 4018

C64/IEEE-488 Steckmodul

Dieser ausgereifte, weltweit erprobte IEEE-488-Modul eröffnet dem Commodore 64 über seinen parallelen Ausgang ungeahnte Einsatzmöglichkeiten wie:
große, IEEE-kompatible CBM-Peripherie am C-64, simultanen (seriell – VC/paralleler – IEEE) Datenverkehr. Konfliktfreie, speicherverschiebliche Modulsoftware. Im Einsatz beispielsweise **in Schulen** ermöglicht der IEEE-488-Steckmodul problemlose Mehrbenutzersysteme am IEC-Bus wie auch durch die rationell genutzte Peripherie: z. B. zahlreiche Computer an einer Doppelfloppy.
In der Industrie bietet der IEEE-488-Steckmodul die Möglichkeit für preisgünstige IEC-Meß-/Steuersysteme mit dem Commodore 64 als Controller. Zu diesem Modul wird ein **Betriebshandbuch** geliefert, in dem Beschreibungen zu fast sämtlichen Anwendungsfällen mit Programmbeispielen, Belegungstabellen, Angaben zum erforderlichen Kabel- und Steckermaterial, Literatur etc. aufgeführt sind. Zusätzlich können zum IEEE-488-Steckmodul **Anwendungshilfen** wie u. a. Disketten mit Lesekennzeichen, Utility-Disketten usw. bezogen werden.

IEEE-Steckmodul für Commodore 64
einschließlich Betriebshandbuch DM 239,— inkl. MwSt.

te-wi

te-wi Verlag GmbH
Theo-Prosel-Weg 1
8000 München 40

MICRO MAG

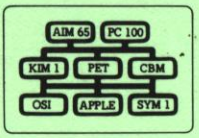
HERAUSGEBER/EDITOR:
DIPL.-VOLKSWIRT ROLAND LÖHR
HANSDORFER STRASSE 4
D-2070 AHRENSBURG
☎ (04102) 5 58 16

Bezugsbedingungen: Abonnement ab laufender Ausgabe für 6 Hefte DM 54,-- im Inland, bzw. DM 59,-- im Ausland (surface mail). Luftpostzustellung auf Anfrage. Abonnements laufen bis auf Widerruf mit Kündigungsmöglichkeit bis zu 4 Wochen vor deren Ablauf. - Nachliefermöglichkeiten siehe unten.

Private Besteller werden um Überweisung oder Scheck (auch Auslandsschecks) zusammen mit Bestellungen gebeten. Konto Roland Löhr, Nr. 654 70-202 Postgiroamt Hamburg, BLZ 200 100 20.

Leser-Service des Herausgebers

Das Buch 7-13 des 65.. MICRO MAG



340 Seiten, DM 42,--

Nachliefermöglichkeiten:

Vergriffen sind 'Das Buch 1-6 des 65xx MICRO MAG' sowie die Hefte 1-13 der Zeitschrift. Es erfolgt keine Neuauflage.

Lieferbar: 'Das Buch 7-13 des 65xx MICRO MAG' zu DM 42,-- sowie die Hefte 14-39 zu DM 7,80/St. (ab 10 St. in einer Sendung: DM 6,--/St.).

FORTH User's Manual
Rockwell-Handbuch für das FIG-FORTH des AIM 65. Mit der Erläuterung des Befehlssatzes auch für andere FORTH-Betreiber geeignet. Ca. 300 S., engl. DM 30,--

Mathe-ROM nach P. Rix für FORTH des AIM + Assemblerprog. m. Doku DM 124,30

Vorstehende Preise inkl. MWSt, zuzüglich DM 2,50/Sendung + ggfs. NN + DM 2,--

Assembler für R65C02, MC6805 und 6809: Siehe im Heftinneren.