65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

#2 (2. AUFL.) AUGUST 1978

Ein Oberblick

Das zweite Heft dieses Journals erscheint in weiterer Gestaltung, um den zahlreichen Anregungen zu entsprechen.

Die erste Ausgabe fand wegen ihres sicher bemerkenswerten Inhaltes eine überaus zustimmende Aufnahme, besonders auch in Firmen, die sich mit Mikroprozessorentwicklung befassen. Das vorliegende zweite Heft ist umfangreicher geworden und es enthält weitere wertvolle Programme und Hinweise. Wegen der Sommerpause konnten dabei noch nicht alle angestrebten Rubriken bedacht werden. Das Oktoberheft wird nach den Planungen weitere Fortschritte bringen.

Weitere Programme und Beiträge kamen auf das Journal zu, so daß bereits jetzt ein Fundus sehr nützlicher Artikel für einige weitere Ausgaben vorliegt.

INHALTSVERZEICHNIS

Gleichwohl sollte ein breiteres 65xx-Gerätespektrum als bisher überstrichen werden. Das jetzige Autorenteam kann sich bei der bereits gegebenen Fülle des Angebotes nicht mit allen bemerkenswerten neuen Geräten für die redaktionelle Arbeit ausrüsten, so daß hier weitere Autoren mit ihrem Spezialwissen für Anwendungsberichte, Interface-Vorschläge und Programme ihr Feld finden können.

Im vorliegenden Heft wurden die englischen Zusammenfassungen erweitert und auch die Kommentare in den Programmzeilen wurden vermehrt auf Englisch ausgerichtet, um die internationale Lesbarkeit zu erleichtern. Im Hinblick auf unsere deutschsprachige Leserschaft meinen wir, daß sie wegen der Geläufigkeit des Englischen damit bequem zurechtkommt, zumal der deutsche Textteil entsprechend ausführlicher gestaltet wurde. Für die Nützlichkeit internationalen Informationsaustausches spricht die Arbeit auf den folgenden Seiten.

****

HINWEISE FÜR AUTOREN

Dieses Journal wird als Manuskript gedruckt. Als Vorlagen dienen weiße DIN A4-Seiten, einseitig beschrieben. Der ausnutzbare Raum beträgt bis zu 185 mm Breite und bis zu 240 mm Höhe. Für den direkten Abdruck bestimmte Vorlagen sollten engzeilig, mit kräftigem Farbband und mit sauberen Typen geschreiben sein. Die besten Vorlagen entstehen auf elektrischer Kugelkopfmaschine mit Plasticontrastband. Wenn solche Möglichkeiten nicht zur Verfügung stehen, sollten Manuskripte hier ins Reine geschrieben werden. Schreibfehler können mit Korrekturlack überdeckt werden. Klebemontagen von Textabschnitten und Zeichnungen sind möglich.

Für die Gestaltung, Kommentierung und Aufmachung enthält dieses Heft sicher ausreichende Vorlagen für die Möglichkeiten.

Alle Beiträge, die sich auf zusätzliche Schaltungen, Interfaces, beziehen, sollten eine Reinzeichnung dieser Schaltungen umfassen, und zwar möglichst in einer Größenanordnung, die für den direkten Abdruck geeignet ist.

Kleine Korrekturen werden hier direkt vorgenommen, größere mit dem Autor abgesprochen, im allgemeinen tritt eine englische Zusammenfassung hinzu. Natürlich können auch englische Manuskripte vorgelegt werden. Wenn nicht anders gewünscht, erscheinen Namen und Anschrift des Autors regelmäßig im Kopf des Beitrages.

Die Benutzung der Zeropage in Programmen sollte besonders erwähnt werden, ebenso die 'Bedienung' des Programmablaufes. Programmniederschriften bitte möglichst mit Hilfe einer zweiten Person in Korrektur lesen.

KIM und PET sind registrierte Warenzeichen der Fa. MOS Technology/ Commodore.

THE LOVELY COUPLE OF QUICKDUMP AND VERSALOAD

Double speed loader, relocating and linking.

By John Oliver and Roland Löhr.

E: John Oliver, Ass. Professor of Astronomy at the University of Florida, doubled the speed of writing and reading magnetic tape as compared to Jim Butterfield's HYPERTAPE by coding each byte with only 8 bits instead of two ASCII characters. Prof. Oliver gave kind permission to reprint his SUPERDUMP and SUPERLOAD in this journal. The editor nevertheless decided to put the nucleus of these into another frame in order to win new features and to combine it with other useful utilities.

Thankful acknowledgement is made to Mr. Oliver for co-authoring his grand idea which formed the basis for all of this:

By now we have writing or reading 1 kbytes in about 12 seconds, input to old locations or relocated, open end input or protection of memory by buffer, DIRECTORY, DUPE, calling records by single-byte ID or by name (header of any 6 bytes). And together with the editor's RALOAD (first issue of 65xx MICRO MAG): Relocation of programs to new address space, a linking loader with same transformation. Last, not least the option to introduce 'external parameters' aside from header.

QUICKDUMP and VERSALOAD are a dependent pair, they are not compatible with other recording formats.

* * *

Jim Butterfield's HYPERTAPE beschleunigt das Bandschreiben und -laden um den Faktor 6. Sehr nützlich ist auch sein SUPER DUPE (Kopieren von Bandcassetten) und sein DIRECTORY (Lesen von Startadresse und ID vom Band ohne zu laden).

Den nächsten großen Schritt machte John Oliver, Astronomie-Professor an der University of Florida, Williamson Hall, Gainesville FL 32611, mit seinen Programmen SUPER DUMP und SUPERLOAD, zuerst veröffentlicht in den KIM User Notes 7/8. Im KIM-Monitor und auch noch in HYPERTAPE wird jedes Zeichen zunächst in 2 ASCII-codierte Sequenzen zerlegt und gesendet. John Oliver verdoppelt die Geschwindigkeit auf etwa 1 kBytes in 12 Sekunden, indem er je Byte nur einmal 8 Bits sendet; eine klare logische Konsequenz, auf die jemand erst kommen mußte.

Wie beim KIM, so wird auch hier das einzelne bit durch verschiedene und verschieden lange Frequenzen auf dem Magnetband abgebildet. (Zur Erläuterung: s. KILOBAUD, Heft 11/77, S. 66 ff.). Soweit als möglich verwenden beide Autoren Unterprogramme des KIM-Monitors und trixen sie z.T. aus.

John Oliver erteilte diesem Journal freundlichst Nachdruckrechte seiner Programme. Wir haben ihm dafür herzlich zu danken, denn sein Brief ermutigte in mehr als einwöchiger intensiver Arbeit eigene Weiterentwicklungen, die hier als QUICKDUMP und VERSALOAD präsentiert werden.

Um das Herzstück der utilities SUPERDUMP und SUPERLOAD baute der Herausgeber einen Rahmen, der sicher fortschrittlich ist:

VERSALOAD ist nicht nur ein reines Ladeprogramm, sondern zugleich auch Kopierprogramm (wie SUPER DUPE), Inhaltsverzeichnis (wie DIRECTORY), Etikettensucher (wie HEADHUNTER des Herausgebers) und es ist neben anderem auch ein

linking and/or relocating program loader.

Diese Eigenschaft dürfte die schönste und bequemste von allen sein. VERSALOAD nutzt die Dienste des in Heft 1 des 65xx MICRO MAG abgedruckte RALOAD (Verschiebung und Umrechnung): Programmsegmente werden jeweils ab nächstfolgender freier Speicherzelle geladen und für den neuen Adressenraum umgerechnet (zur notwendigen 'Syntax' s. Heft 1).

Nützlich ist auch die mögliche Festlegung eines Eingabepuffers. Ein Speicherbereich kann ab einer in Loc. 17F7/F8 festgelegten Anfangsadresse gegen unbeabsichtigtes Überschreiben durch das einzulesende Programm geschützt werden.

Programme oder Datensätze können wahlweise mit 1-Byte-ID oder mit einem Standard-Label (Name von 6 Bytes) versehen sein. VERSALOAD erkennt die gewählte Form der Identifizierung und sucht entsprechend nac Gleichheit. Es erkennt auch, ob ein Bandsatz darüber hinaus zusätzliche Bytes (bis 249) als Option mit sich führt, um z.B. Namen und Speicherplatz benötigter oder abzugebender 'externer Parameter' zu beschreiben.

QUICKDUMP und VERSALOAD sind als Unterprogramme geschrieben. Sie ermöglichen kontinuierliches, von der Maschine her gesteuertes Arbeiten (JSR ENTRY mit Paramtern in A bzw. in X). Bei Aufruf mit einer der Kopfzeilen kann jedoch ebenso einmaliges Abarbeiten mit Rückkehr zum KIM-Monitor bewirkt werden. Das Datenformat beim Schreiben und das Blockdiagramm des Leseprogrammes sind auf den folgenden Seiten abgedruckt.

Dieses Programm-Paar ist mit anderen Aufzeichungsformen nicht kompatibel

In der Summe haben wir jetzt Dienstleistungen zur Hand, die schon denen einer größeren Datenverarbeitung ähneln. Insbesondere sei auch auf Möglichkeiten des Overlay hingewiesen: Bei begrenztem Speicher können lange Programme in Segmente zerlegt werden, die bei Bedarf in einen Puffer (Overlaybereich) nachgezogen werden.

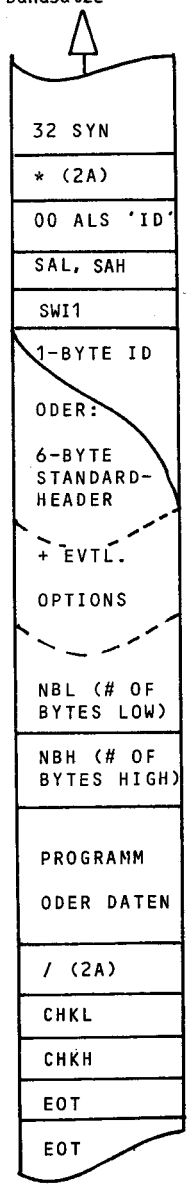
* * *

```
                        QUICKDUMP

0600   A9 00      STARTA  LDA #$ 00       CALL WITH NO HEADER
0602   AA                 TAX             00 TO X FOR SWITCHING
0603   F0 04              BEQ GOSUB       BRANCH ALWAYS
0605   A9 05      STARTB  LDA #$ 05       CALL WITH 6-BYTE HEADER
0607   A2 00              LDX #$ 00       FOR SWI2
0609   20 0F 06   GOSUB   JSR ENTRY1      CALL MAIN PGM AS A SUBROUTINE
060C   4C 4F 1C           JMP KIM         RETURN TO MONITOR

                        SUBROUTINE (MAIN)

060F   85 D4      ENTRY1  STA SWI1        SAVE PARAMETERS
0611   86 D5              STX SWI2        FROM START
0613   A2 04              LDX #$ 04       COUNTER FOR TRANSPORT
```

```
0615  BD 29 07  STOTAB  LDA TAB-1,X    PUT TRAILING TABLE TO
0618  95 CF             STA NPUL-1,X   ZEROPAGE
061A  CA                 DEX
061B  D0 F8              BNE STOTAB     DONE?

061D  A9 AD      SUPERD  LDA #$ AD      "STA"-OPCODE FOR VEB. MAIN BODY
061F  8D EC 17           STA VEB        OF JOHN OLIVERS PROGRAM
0622  20 32 19           JSR INTVEB     INITIALIZE VEB
0625  A9 27               LDA #$ 27
0627  85 CC              STA GANG       SBD OUTPUT WORD
0629  A9 BF              LDA #$ BF      OPEN CHANNELS
062B  8D 43 17           STA SBD
062E  A9 20              LDA #$ 20      SEND 32 SYNC CHARACTERS
0630  85 CD              STA TIC        SAVE CHAR COUNT
0632  A9 16              LDA #$ 16      ....SYNC....
0634  48        HIC1    PHA            SAVE THIS CHARACTER
0635  20 F2 06           JSR OUTCHT     SEND
0638  68                 PLA            RESTORE CHARACTER
0639  C6 CD              DEC TIC        REDUCE COUNTER
063B  D0 F7              BNE HIC1       FINISHED?

063D  A9 2A              LDA #$ 2A      TO SEND "*"
063F  20 F2 06           JSR OUTCHT

0642  A9 00              LDA #$ 00      DUMMY-ID
0644  20 D0 06           JSR OUTBT      SEND 2 ASCII
0647  A5 D5              LDA SWI2       DOES VERSALOAD REQUEST A DUPE?
0649  D0 0F              BNE DUP1       YES
064B  AD F5 17           LDA SAL        TAKE ORIGINAL VALUES
064E  20 D0 06           JSR OUTBT      AND SEND
0651  AD F6 17           LDA SAH
0654  20 D0 06           JSR OUTBT
0657  4C 64 06           JMP SENDID     SKIP

065A  A5 D6      DUP1    LDA OSAL       TAKE INSTEAD AND SEND,
065C  20 D0 06           JSR OUTBT      VALUES SUPPLIED BY VERSALOAD
065F  A5 D7              LAD OSAH
0661  20 D0 06           JSR OUTBT

0664  A5 D4      SENDID  LDA SWI1       SWI1 = 00 OR 05 OR OPTIONAL
0666  20 D0 06           JSR OUTBT      LENGTH OF TABLE [EXTERNAL PARMS]
0669  A5 D4              LDA SWI1       DECIDE TO SEND 1-BYTE ID OR
066B  D0 09              BNE SENDTB     A HEADER

066D  AD F9 17           LDA ID         SEND 1-BYTE ID
0670  20 F2 06           JSR OUTCHT
0673  4C 8A 06           JMP NUMBOB     SKIP ALWAYS

0676  A9 00      SENDTB  LDA #$ 00      SWI2 INSTALLED NOW AS COUNTER
0678  85 D5              STA SWI2
067A  A6 D5      LOSWI   LDX SWI2
067C  BD 80 17           LDA 1780,X     HEADER TABLE HERE INSTALLED
067F  20 F2 06           JSR OUTCHT     AND SENT
0682  E6 D5              INC SWI2       UPCOUNT...
0684  A5 D4              LDA SWI1       TO BE COMPARED
0686  C5 D5              CMP SWI2
0688  B0 F0              BCS LOSWI      IF SWI1 ≥ SWI2
```

```
068A  38        NUMBOB  SEC             PREPARE SBC TO CALCULATE
068B  AD F7 17          LDA EAL         NUMBER OF BYTES
068E  ED F5 17          SBC SAL         GIVING NBL
0691  08                PHP             SAVE CARRY STATUS
0692  20 F2 06          JSR OUTCHT      SEND NBL

0695  28                PLP             RESTORE CARRY STATUS
0696  AD F8 17          LDA EAH
0699  ED F6 17          SBC SAH         GIVING NBH
069C  20 F2 06          JSR OUTCHT      & SEND
069F  20 43 19          JSR INTVEB+17   RESET CHECKSUM TO 00

06A2  20 EC 17  SUPDP1  JSR VEB         GET BYTE ADDRESSED BY VEB...
06A5  20 EF 06          JSR OUTCHC      ...AND SEND IT
06A8  20 EA 19          JSR INCVEB      ADDRESSING + 1 FOR NEXT BYTE
06AB  AD ED 17          LDA VEB+1       ARE WE AT ...
06AE  CD F7 17          CMP EAL         ... END ADDRESS?
06B1  AD EE 17          LDA VEB+2
06B4  ED F8 17          SBC EAH
06B7  90 E9             BCC SUPDP1      NOT FINISHED, GET MORE
06B9  A9 2F             LDA #$ 2F       SEND "/"
06BB  20 F2 06          JSR OUTCHT

06BE  AD E7 17          LDA CHKL        SEND CHECKSUM
06C1  20 D0 06          JSR OUTBT
06C4  AD E8 17          LDA CHKH
06C7  20 D0 06          JSR OUTBT
06CA  A9 04             LDA #$ 04       EOT CHARACTER
06CC  20 F2 06          JSR OUTCHT
06CF  60                RTS             GOBACK

                SUBROUTINES DIVISION
06D0  48        OUTBT   PHA             HEX OUTPUT ROUTINE: SAVE BYTE
06D1  4A 4A             LSR, LSR        ISOLATE MSD
06D3  4A 4A             LSR, LSR
06D5  20 E3 06          JSR HEXTA       & WRITE AS ASCII
06D8  20 F2 06          JSR OUTCHT
06DB  68                PLA             RESTORE BYTE
06DC  20 E3 06          JSR HEXTA       GET 4 LSB AND WRITE AS ASCII
06DF  20 F2 06          JSR OUTCHT
06E2  60                RTS

06E3  29 0F     HEXTA   AND #$ 0F       MASK OFF 4 LSB
06E5  C9 0A             CMP #$ 0A
06E7  18                CLC
06E8  30 02             BMI HEXTA1
06EA  69 07             ADC #$ 07       A TO F
06EC  69 30     HEXTA1  ADC #$ 30       0 TO 9
06EE  60                RTS

06EF  20 4C 19  OUTCHC  JSR CHKT        CHECKSUM CALCULATION
06F2  A0 08     OUTCHT  LDY #$ 08       SET FOR 8 BITS
06F4  84 CE             STY COUNT       SAVE BIT COUNT
06F6  A0 02     TRY     LDY #$ 02       SET FOR 3 PHASES
06F8  84 CF             STY TRIB        SAVE PHASE COUNT
06FA  B6 D0     ZON     LDX NPUL,Y      # OF 1/2 CYCLES
06FC  48                PHA             SAVE CHARACTER
06FD  78        ZON1    SEI             DISABLE INTERRUPTS
```

QUICKDUMP
Aufzeichungs-
format der
Bandsätze



Weitere Hinweise zu QUICKDUMP

Schreiben eines Bandes mit 1-Byte-ID:

Startadresse	SAL/SAH nach 17F5/F6
Endadresse+1	EAL/EAH nach 17F7/F8
Identität	ID nach 17F9
Start	STARTA in 0600

Schreiben eines Bandes mit 6-Byte-Namen (Standard header):

Startadresse und Endadresse wie vor

Identität	Header nach 1780-1785 oder bei Veränderung der Adresse in LOSWI woanders.
Start	STARTB in 0605

Schreiben eines Bandes zusätzlich mit 'externen Parametern':

Startadresse, Endadresse und standard header wie vor.

LDA wirkliche Länge der in 1780 ... gespeicherten Tabelle.

```
LDX #$ 00
JSR ENTRY1
...
```

Die Aufzeichungsgeschwindigkeit kann mit den für 072A und 072C alternativ vorgesehenen Wertepaaren herabgesetzt werden. Die in TAB eingestellten Werte entsprechen x6 mit doppelter Schreibdichte. In Heft 10/11 der KIM-User Notes hatte John Oliver noch vorgeschlagen, das Wertepaar gegeneinander auszutauschen. Diese Empfehlung gilt heute nicht mehr. Sollten beim Lesen irgendwelche Schwierigkeiten auftreten, so wird stattdessen eine sorgfältige Einstellung des PLL-Potentiometers empfohlen. Dazu schreibt man eine größere Zahl kurzer Sätze auf ein Band und regelt beim Lesen solange ein, bis es eindeutig funktioniert.

ZERO PAGE USED BY QUICKDUMP & VERSALOAD

00C9	EALB	00D0	NPUL	00D6	OSAL
00CA	EALH	00D1	TIMG	00D7	OSAH
00CB	LFLG	00D2	"	00D8	CNTRL
00CD	TIC	00D3	"	00D9	IDTEMP
00CE	COUNT	00D4	SWI1	00E2	EOPL
00CF	TRIB	00D5	SWI2	00E3	EOPH

```
06FE  2C 47 17  ZON2    BIT CLKRD1      TIMER DONE?
0701  10 FB            BPL ZON2        NO, WAIT
0703  B9 D1 00          LDA TIMG,Y      GET WAIT TIME IN ...
0706  8D 44 17          STA CLK1T       MICROSECONDS FOR TIMER
0709  A5 CC             LDA GANG        FLIP OUTPUT BIT ...
070B  49 80             EOR #$ 80       BETWEEN Ø AND 1
070D  8D 42 17          STA SBD         OUTPUT BIT
0710  58                CLI             ENABLE INTERRUPTS
0711  85 CC             STA GANG        SAVE OUTPUT BIT
0713  CA                DEX             ALL CYCLES SENT ?
0714  D0 E7             BNE ZON1        NO, SEND MORE

0716  68                PLA             RESTORE CHARACTER
0717  C6 CF             DEC TRIB        ONE LESS PHASE TO GO
0719  F0 05             BEQ SETZ        AND THIS IS PHASE 3
071B  30 07             BMI ROUT        ALL PHASES DONE

071D  4A                LSR             GET BIT ...
071E  90 DA             BCC ZON                 ... IF IT IS '1'...
0720  A0 00      SETZ    LDY #$ 00       ... CHANGE TO 2400 HZ
0722  F0 D6             BEQ ZON         FORCED BRANCH
0724  C6 CE      ROUT    DEC COUNT       ONE LESS BIT TO GO
0726  D0 CE             BNE TRY
0728  60                RTS             ALL DONE

0729  17                .BYTE           LOP, LENGTH OPERATOR
072A  02          TAB     .BYTE           VALUE FOR NPUL
072B  C3                  .BYTE
072C  03                  .BYTE           VALUE FOR TIMG+1
072D  7E                  .BYTE
072E  EA                  NOP             RALOAD-BYTE

          ALTERNATIVE VALUES FOR NPUL AND TIMG+1:

072A      ×3: $04   ×2: $06    ×1: $0C   SEE ADDITIONAL REMARKS
072C          $06       $09        $12
```

* * *

VERSALOAD - SUMMARY

Program renders 6 basic reading functions (see block-diagram) which are increased by the possibility to define a buffer+1 which may not be surpassed. Begin of buffer to SAL/SAH, end+1 to EAL/EAH.

Records to be read may have any of these formats:

a) single-byte ID to be compared with 17F9,
b) standard name of 6 bytes to be compared to table in 1780-85 or
c) name as in b) plus additional parameters which are read into 1786 ...

First cell to be filled on read is addressed by VEB+1,2 for LINK.., is old start address for LOADOE or LOADBU in all other cases it is the cell: begin of buffer.

VERSALOAD

Wie schon dargestellt, bietet VERSALOAD 6 grundsätzliche Dienstleistungen (Blockdiagramm auf der nächsten Seite). Welche im einzelnen ausgeführt wird, hängt vom gewählten Startpunkt ab oder von dem im Akkumulator mitgebrachten Parameter, wenn man JSR ENTRY aufruft.

Diese sechs Dienstleistungen werden durch die Buffer-Möglichkeit ergänzt: SAL/SAH in 17F5/F6 legen dann eine Anfangsadresse für das Speichern fest, EAL/EAH in 17F7/F8 das Buffer-Ende+1 (erste zu schützende Zelle). Wenn ein Ladevorgang wegen Erreichens des Schutzbereiches unvollständig abgebrochen werden mußte, so erfolgt Fehleranzeige durch den KIM-Monitor mit 'FFFF1C', zugleich wird die LFLAG auf 'FF' gesetzt. Lesefehler mit abweichender Checksum führen zur gleichen Monitor-Anzeige, die LFLAG wird aber auf 'FE' gesetzt.

Die Zahl der Dienstleistungen wird eigentlich fast verdreifacht, weil die Identitätsangabe für die zu lesenden Bandsätze verschieden sein darf:

a) Identität 1 Byte, Vergleich mit Zelle 17F9,
b) Standard-Name 6 Bytes, Vergleich mit Tabelle in 1780-85,
c) Standard-Name wie in b), zusätzlich externe Parameter, die beim Lesen in den Zellen 1786 ff. abgelegt werden.

Die Dienstleistungen im einzelnen:

LADOE, LOAD Open End. Ein Programm wird an seinem alten Speicherplatz geladen, es darf beliebig lang sein, weil ein Puffer nicht zu beachten ist.

LOADBU, LOAD only up to end of BUffer. Einspeicherung ab altem Speicherplatz aber nicht über Puffer-Ende hinaus.

DATALB, DATA Load to Buffer. Anfang und höchstmöglicher Speicherplatz sind festgelegt.

RELOE, RELocate Open End. Rechnet ein Programm sofort nach dem Laden ab festgelegter Speicheradresse auf den neuen Adressenbereich um, und zwar mit Hilfe des in Heft 1 von 65xx MICRO MAG beschriebenen RALOAD, das natürlich im Speicher resident sein muß. Ein Pufferende muß nicht berücksichtigt werden.

RELBUF, RELocate with BUffer. Wie vor, die Endadresse darf nicht überschritten werden.

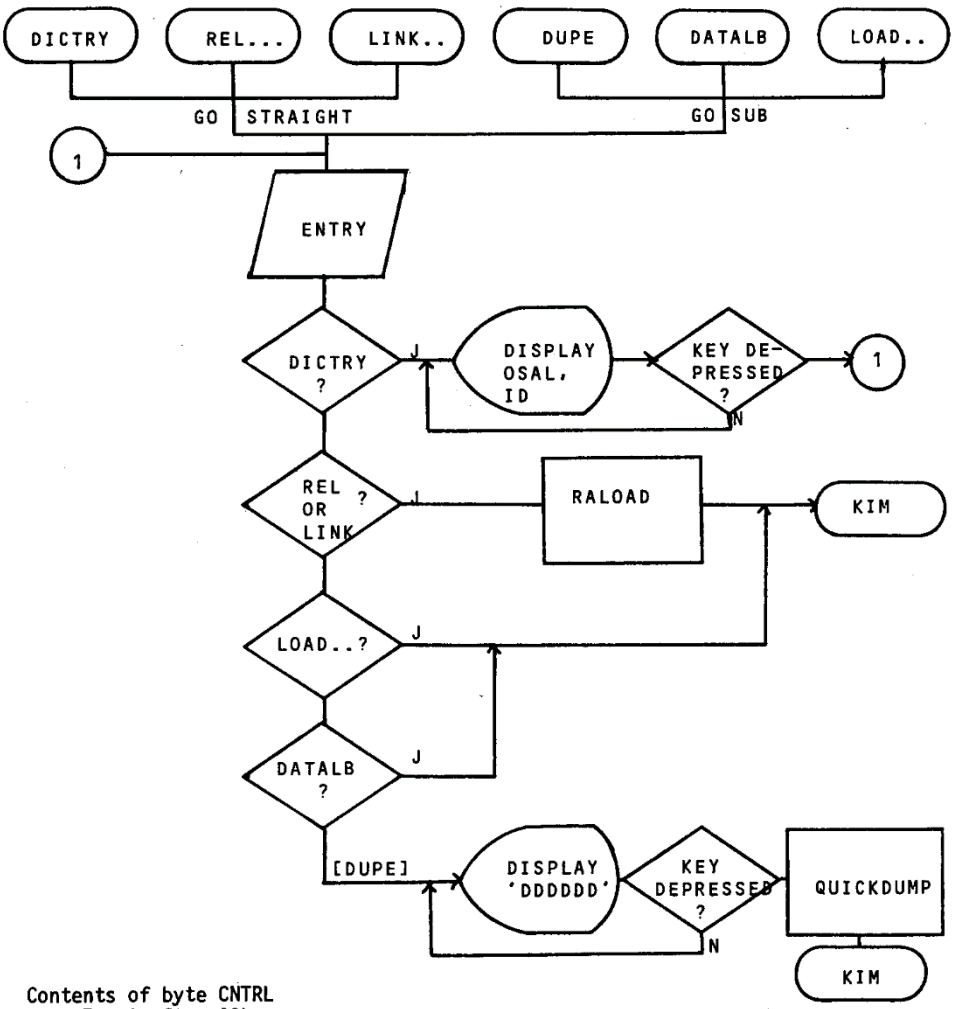
LINKOE, LINK Open End. Verwirklicht Laden und Verschweißen. Laden ab der in VEB+1,2 ausgewiesenen ersten freien Speicherstelle und Umrechnen mit RALOAD auf den neuen Adressenbereich.

LINKBU, LINK with BUffer. Entspr. LINKOE bzw. RELBUF.

DUPE entspricht den Funktionen von Jim Butterfields SUPER DUPE. Es gestattet, Programme von einem Band auf ein anderes zu kopieren. Zur Zwischenspeicherung dient ein Puffer, dessen Grenzen wie vor festzulegen sind. Ein zu kopierendes Programm muß mit seiner wirklichen Identität bzw. 00 oder mit seinem vollen Namen gesucht werden. Betätigung einer Taste löst Quickdump aus.

DICTRY lädt nichts, sondern bringt nur die alte Startadresse und ein weiteres Byte zur KIM-Anzeige. Dieses ist entweder die ID von einem BYTE oder das erste Zeichen des Namens (von 6 Zeichen). Nach Drücken einer Taste entspr. Anzeige für den nächsten Bandsatz.

PROGRAMMABLAUFPLAN VERSALOAD



Contents of byte CNTRL
on ENTRY & after ASL

	LINK	RELOCATE	BUFFER		LOADBU	DATALB	DUPE	DICTRY
LINK	RELOCATE	BUFFER		LOADBU	DATALB	DUPE	DICTRY	
BIT C	N	V	5	4	3	2	1	0

```
                        VERSALOAD
0800  A9 22     DUPE    LDA #$ 22       PARAMETER FOR CNTRL
0802  D0 1F             BNE GOSUB       BRANCH ALWAYS
0804  A9 01     DICTRY  LDA #$ 01       THESE ARE THE DIFFERENT
0806  D0 0E             BNE GOSTR       ENTRIES FOR FUNCTIONS
0808  A9 C0     LINKOE  LDA #$ C0       TO BE RENDERED ONCE
080A  D0 0A             BNE GOSTR
080C  A9 E0     LINKBU  LDA #$ E0
080E  D0 06             BNE GOSTR
0810  A9 40     RELOE   LDA #$ 40
0812  D0 02             BNE GOSTR
0814  A9 62     RELBUF  LDA #$ 62
0816  4C 29 08  GOSTR   JMP ENTRY       THIS ONE COULD HAVE BEEN
0819  A9 00     LOADOE  LDA #$ 00                DELETED.
081B  F0 06             BEQ GOSUB
081D  A9 28     LOADBU  LDA #$ 28
081F  D0 02             BNE GOSUB
0821  A9 24     DATALB  LDA #$ 24
0823  20 29 08  GOSUB   JSR ENTRY
0826  4C 4F 1C          JMP KIM         RETURN TO MONITOR

                        MAIN ROUTINE
0829  0A       ENTRY   ASL             GIVES A BETTER CNTRL, CUTS OFF
082A  85 D8             STA CNTRL       LINK BIT TO CARRY
082C  90 0C             BCC NOLINK
082E  AD ED 17  LINK    LDA VEB+1       INSERT FIRST FREE PLACE TO
0831  8D F5 17          STA SAL         SAL/SAH
0834  AD EE 17          LDA VEB+2
0837  8D F6 17          STA SAH

083A  24 D8     NOLINK  BIT CNTRL       TEST
083C  50 0A             BVC XID         SKIP IF OPEN END LOADING
083E  AD F7 17          LDA EAL         INITIALIZE END OF BUFFER
0841  85 C9             STA EALB
0843  AD F8 17          LDA EAH
0846  85 CA             STA EAHB

0848  AD F9 17  XID     LDA ID          SAVE DESIRED ID
084B  85 D9             STA IDTEMP

084D  A9 00     SUPERL  LDA #$ 00
084F  8D F9 17          STA ID          KIM'S LOADT IS DUPED TO STAY IN
0852  85 CB             STA LFLG        FLAG = 00              SUBROUTINE
0854  85 D5             STA SWI2        COUNT = 00

0856  A9 60             LDA #$ 60       'RTS' OPCODE FOR VEB
0858  8D EC 17          STA VEB
085B  20 8C 18          JSR 188C        ENTER KIM'S LOADT
085E  85 D4             STA SWI1        LENGTH OF NAME & PARMS OR JUST ID
0860  20 24 1A  NEXCHT  JSR RDCHT       READ NEXT CHARACTER
0863  AD EA 17          LDA SAVX+1      GET FULL 8 BIT BYTE
0866  A6 D8             LDX CNTRL       TEST FOR DICTRY
0868  E0 02             CPX #$ 02
086A  D0 03             BNE             SKIP IF NOT
066C  4C 7E 09          JMP SHOWDI      DISPLAY AND WAIT FOR KEY

086F  A6 D4             LDX SWI1        TEST KIND OF ID/NAME & PARMS
0871  D0 0E             BNE COMTB       SKIP IF NO SINGLE BYTE ID
```

```
0873  C5 D9        CMP IDTEMP     IS IT THE SINGLE BYTE ID?
0875  D0 04        BNE SKPD       SKIP IF NOT
0877  85 D9        STA IDTEMP     A SERVICE TO DUPE IF 17F9 WAS ØØ
0879  F0 2B        BEQ NUMBYT     BRANCH ALWAYS

087B  E4 D9        CPX IDTEMP     X CONTAINS ØØ
087D  F0 27        BEQ NUMBYT     TREAT AS A MATCH
087F  D0 CC        BNE SUPERL     TEST NEXT RECORD

0881  A6 D5  COMPTB LDX SWI2       GET CURRENT OFFSET
0883  DD 80 17      CMP 1780,X     COMPARE TO NAME IN TABLE
0886  D0 CE        BNE SUPERL     GET NEXT RECORD ON NO MATCH
0888  E6 D5        INC SWI2       OFFSET+1
088A  A9 05        LDA #$ 05      TO COMPARE FOR END OF NAME
088C  C5 D5        CMP SWI2
088E  B0 D0        BCS NEXCHT     GET TOTAL OF 6 BYTES
0890  A5 D4  EOHEAD LDA SWI1       STORE ADDITIONAL PARMS FROM
0892  C5 D5        CMP SWI2       TAPE HEADER IF SWI1 STANDS FOR
0894  90 10        BCC NUMBYT     MORE THAN 6 BYTES  * ALL DONE
0896  20 24 1A     JSR RDCHT      GET NEXT PARM
0899  AD EA 17     LDA SAVX+1     GET 8-BIT-BYTE
089C  A6 D5        LDX SWI2       GET CURRENT OFFSET
089E  9D 80 17     STA 1780,X     STORE TO MEMORY
08A1  E6 D5        INC SWI2       FOR NEXT
08A3  4C 90 08     JMP EOHEAD     AND TEST FOR END OF PARMLIST

08A6  AD ED 17  NUMBYT LDA VEB+1   SAVE OLD START ADDRESS
08A9  85 D6        STA OSAL
08AB  AD EE 17     LDA VEB+2
08AE  85 D7        STA OSAH
08B0  A9 8D        LDA #$ 8D      'STA'-OPCODE FOR VEB
08B2  8D EC 17     STA VEB        REPLACES 'RTS'

08B7  20 24 1A     JSR RDCHT      NEXT CHAR
08B8  A5 D8        LDA CNTRL      TEST
08BA  F0 04        BEQ NADJ       IT'S LOAD TO OLD START ADDRESS
08BC  C9 10        CMP #$ 10
08BE  D0 06        BNE XCHNG      IT'S LOAD TO NEW START ADDRESS
08C0  20 3E 19     JSR INTVEB+12  INITIALIZE VEB AND CHKL CHKH
08C3  4C C9 08     JMP COMPU      SKIP

08C6  20 32 19  XCHNG JSR INTVEB   INITIALIZE FULL WITH NEW START
08C9  18        COMPU CLC                                  ADDRESS
08CA  AD EA 17     LDA SAVX+1     NBL WITH 8 BITS
08CD  6D ED 17     ADC VEB+1      TO COMPUTE ENDADDRESS OF PGM
08D0  85 E2        STA EOPL       AND SAVE
08D2  08           PHP            SAVE CARRY STATUS
08D3  20 24 1A     JSR RDCHT      NEXT CHAR IS NBH
08D6  28           PLP            GET BACK CARRY STATUS
08D7  AD EA 17     LDA SAVX+1     SAME FOR # HI
08DA  6D EE 17     ADC VEB+2
08DD  85 E3        STA EOPH

08DF  20 24 1A  PATCH1 JSR RDCHT   JOHN OLIVER'S NUCLEUS
08E2  AD EA 17     LDA SAVX+1     NEXT 8 BITS FROM TAPE
08E5  20 4C 19     JSR CHKT       ADD TO CHECKSUM
08E8  20 EC 17     JSR VEB        STORE IT

08EB  20 EA 19     JSR INCVEB     INCREMENT VEB ADDRESS FOR STORE
```

```
08EE  AD ED 17          LDA VEB+1       END ADDRESS?
08F1  24 D8             BIT CNTRL
08F3  50 06             BVC PATCH3      DONT'T CARE FOR BUFFER
08F5  C5 C9             CMP EALB        BUFFER END?
08F7  D0 02             BNE PATCH3      NO
08F9  F0 04             BEQ PATCH4      MAYBE?
08FB  C5 E2     PATCH3  CMP EOPL        RECORD END?
08FD  D0 E0             BNE PATCH1      NO, GET MORE BYTES

08FF  AD EE 17  PATCH4  LDA VEB+2       SAME FOR HI
0902  24 D8             BIT CNTRL
0904  50 11             BVC PATCH5      DON'T CARE FOR BUFFER
0906  C5 CA             CMP EAHB        BUFFER END?
0908  D0 0D             BNE PATCH5      NO
090A  C5 E3             CMP EOPH        ALSO RECORD END?
090C  D0 64             BNE ERROR2      NO, ERROR EXIT
090E  AD ED 17          LDA VEB+1       LOW ORDER BYTE ALSO OK?
0911  C5 E2             CMP EOPL
0913  D0 5D             BNE ERROR2
0915  F0 04             BEQ PATCH6

0917  C5 E3     PATCH5  CMP EOPH        RECORD END?
0919  D0 C4             BNE PATCH1      NO, CONTINUE

091B  20 24 1A  PATCH6  JSR RDCHT       GET ENDING CHARACTER
091E  C9 2F             CMP #$ 2F       '/' ?
0920  D0 4E             BNE ERROR

0922  20 F3 19          JSR RDBYT       GET CHECKSUM LO
0925  CD E7 17          CMP CHKL        CHECKSUM OK ?
0928  D0 46             BNE ERROR
092A  20 F3 19          JSR RDBYT       GET CHECKSUM HI
092D  CD E8 17          CMP CHKH
0930  D0 3E             BNE ERROR

0932  A5 D8             LDA CNTRL       TEST FOR LINK AND RELOCATE
0934  10 0D             BPL BUFA        BRANCH IF NOT
0936  A5 D6             LDA OSAL        SERVICE TO RALOAD PROGRAM
0938  8D F7 17          STA EAL
093B  A5 D7             LDA OSAH
093D  8D F8 17          STA EAH
0940  4C 27 02          JMP RALOAD      TAKE CARE THAT PROGRAM
                                        ACTUALLY RESIDES HERE.

0943  29 0C     BUFA    AND #$ 0C
0945  F0 36             BEQ EXIT        IF LOADOE OR LOADBU
0947  AD ED 17          LDA VEB+1       INSERT ENDADDRESS+1 FOR
094A  8D F7 17          STA EAL         DUPE AND DATALB
094D  AD EE 17          LDA VEB+2
0950  8D F8 17          STA EAH
0953  A5 D8             LDA CNTRL       TEST
0955  C9 48             CMP #$ 48       IS IT DATALB ?
0957  F0 24             BEQ EXIT        ALL DONE FOR THIS ONE

0959  A5 D9             LDA IDTEMP      GET PARAMETERS FOR DUPE
095B  8D F9 17          STA ID
095E  A9 DD             LDA #$ DD       TO SHOW 'DD DD DD'
0960  85 F9             STA INH
```

```
0962   85 FA          STA POINTL
0964   85 FB          STA POINTH
0966   20 1F 1F SHOW1 JSR SCANDS     DISPLAY 'DD DD DD' AND WAIT
0969   F0 FB          BEQ SHOW1      NO KEY DEPRESSED ?
096B   A2 01          LDX #$ 01      PARAMETER ≠ ØØ FOR SWI2
096D   4C 11 06       JMP ENTRY1+2   TAKE CARE THAT QUICKDUMP'S
                                      ENTRY1 ACTUALLY RESIDES HERE !

0970   C6 CB    ERROR  DEC LFLG
0972   C6 CB    ERROR2 DEC LFLG
0974   24 D8          BIT CNTRL      TEST KIND OF ENTRY INTO PGM
0976   30 02          BMI SKPD       WAS STRAIGHT ENTRY, SKIP
0978   68 68    SKPD  PLA PLA        ADJUST SP
097A   4C 29 19       JMP LOADT9     KIM SHOWS 'FF FF 1C'

097D   60        EXIT  RTS

097E   85 F9    SHOWDI STA INH       DICTRY SHALL SHOW ID
0980   AD ED 17       LDA VEB+1      AND OLD START ADDRESS
0983   85 FA          STA POINTL
0985   AD EE 17       LDA VEB+2
0988   85 FB          STA POINTH
098A   20 1F 1F SHOW2 JSR SCANDS
098D   F0 FB          BEQ SHOW 2     NO KEY DEPRESSED ?
098F   4C 4D 08       JMP SUPERL     READ HEADER OF NEXT RECORD IF YE
0992   EA        END   NOP            RALOAD BYTE FOR RELOCATION
```

* * * * *

Wo soll man Arbeitsspeicher bereitstellen ?

Bei der Abfassung von Programmen tritt immer wieder ein Gewissenskonflikt auf: Soll man den Arbeitsspeicher

a) in die Zero Page legen, wie hier z.B. bei QUICKDUMP und VERSALOAD, oder aber
b) an das Ende des Programmes selbst, in die absolute Adressierung?

a) hat den Nachteil, daß die Zero Page sich sehr schnell füllt und daß ein Programm die Pointer eines anderen zerstören kann. Diese Gefahr besteht vor allem beim Einsatz von Programmen aus verschiedener Quelle.

b) ist eine eindeutige Methode, die Arbeitsspeicher zu entflechten. Es treten keine Schwierigkeiten bei der Programmverschiebung auf, wenn RALOAD und seine Formate verwendet werden. Nachteil: Solche Programme können nicht in Festwertspeicher übernommen werden.

Je nach Interessenlage wird man den Ausweg aus diesem Konflik in der Reservierung der Zero Page für Pointer und schnelle Verarbeitung und in der Bereitstellung einer anderen Page für Arbeitsspeicher suchen.

S U M M A R Y

E: A file on magnetic tape is built up from several evenly structured records and is terminated by a file-separator record. Each record may contain various fields for general information and field labels followed by amounts or quantities, the latter in BCD-code.

SUMMARY looks for a match to a field label which you put into a table, working nonstop over the whole file. On each find quantities are added into a results field. Many variations from this example could be worked out.

Heft 1 dieses Journals enthielt das Programm STATISTICIAN zur Auswertung von Dateien. Dort wurde gezählt, wie oft eine beliebige Merkmalskombination in einem auf Magnetband gespeicherten Datenbestand vorkommt. SUMMARY geht von einem ähnlichen Denkansatz aus: Es addiert alle vorgefundenen Werte in ein Zählfeld, und zwar wiederum nonstop über alle Datensätze.

Zur Verdeutlichung und Klärung der Begriffe gehen wir von einem kaufmännischen Beispiel aus. Die Kundenkonten einer Buchhaltung bilden in ihrer Gesamtheit eine Datei engl. file, die sich in eine Vielzahl einzelner Kontenblätter gliedert. Dem Einzel-Kontenblatt entspricht in der Datentechnik der Satz (engl. record). So wie ein Kontenblatt einheitlich mit Feldern für die Kontennummer, den Namen und die Anschrift sowie für Beträge usw. gegliedert ist, so enthält auch ein gespeicherter Satz entsprechende Felder. In der begrifflichen Rangordnung zwischen Datei und Satz gibt es dann noch den sog. Block. Es handelt dabei um die Zusammenfassung mehrerer logischer Sätze zu einer größeren physischen Einheit bei der Abspeicherung und Wiedereinlesung. Durch das Blocken erspart man in der kommerziellen Datentechnik viele der Starts und Stops, die sonst bei der satzweisen Aufzeichnung eintreten, die Zahl der Lücken (gaps) zwischen den Blöcken ist geringer, das Magnetband wird besser ausgenutzt.

SUMMARY gestattet es nun, die Werte oder Mengen für jeweils ein Feld in den Datensätzen über die Datei hinweg aufzuaddieren, also z.B. für "Forderungen". Dieses Programm ist für satzweise Speicherung geschrieben. Abwandlungen sind mühelos zu vollziehen.

Nun muß SUMMARY bei Beginn 'wissen', nach welchem Feld es suchen soll, es braucht ein Vergleichsfeld, das laufend mit den Etiketten verglichen wird, die am Lesekopf des Magnetbandgerätes vorbeihuschen.
Etiketten: Auch eine Buchhaltung führt man nicht mit unbedruckten Kontokarten, man hat mindestens Lineaturen und meistens kleingedruckte Feldbezeichnungen und Spaltenüberschriften. Ebenso brauchen Magnetbandsätze Etiketten für Auswertungsfelder. Man kann z.B. also ganz friedlich und in ASCII-Code das Wort "Forderungen" vor das dazugehörige Betragsfeld schreiben. Wenn man diese summieren will, schreibt man in das Vergleichsfeld ebenfalls "Forderungen", was sich von 7-Bit-ASCII auf hexadezimal so liest: 46 6F 72 64 65 72 75 6E 67 65 6E.
Und zum Abschluß des Vergleichsbegriffes fügt man ein hexadezimales "FF" an.

SUMMARY ist einfach gestaltet, um nur das Prinzip aufzuzeigen. Es setzt daher voraus, daß sich unmittelbar an das Etikett "Forderungen" das Betrags- oder Mengenfeld anschließt, und zwar in der dezimal-rechenfähigen BCD-Codierung.

Die dezimale 22 soll da also wirklich als 22 stehen und nicht als das hexadezimale Pendant 16. Die Zahlen sind weiterhin als von links nach rechts und ohne Kommas notiert angenommen. Der Betrag von DM 123,50 stehe unter Paarbildung der Ziffern wie folgt im Datensatz: 50 23 01 00 00 EF. Das "EF" ist dabei die Feldende-Marke. Das Ergebnisfeld hat hier die gleiche Breite wie das Datenfeld, beide müssen daher genug Kapazität für die Aufnahme des Gesamtergebnisses haben.

Für den aktuellen Anwendungsfall kann man SUMMARY flexibel anpassen, z.B. mehrere Betrags- und Mengenfelder gleichzeitig addieren und auch die für kaufmännische Anwendungen erforderliche Zuverlässigkeit überprüfen, indem man für jeden Datensatz, wie im KIM-Monitor, die CHECKSUM ermittelt. Man könnte sogar für jedes einzelne Betragsfeld eine Kontrollsumme bilden und diese mit den Daten zusammen abspeichern. Weitere Sicherheit läßt sich wie folgt erzielen: Es besteht ja immer die Gefahr einer Fehl-Einlesung. Nichthexadezimale Zeichen führen bei SUMMARY zwar zum vorzeitigen Programmabbruch, jedes Betragsbyte wird aber sofort auf das Ergebnisfeld addiert, auch wenn es fehlerhaft war. Es ist empfehlenswerter, die ankommenden Bytes zunächst in einen Pufferbereich zu übernehmen, bei Ende des Datensatzes dann zunächst die zugehörige Kontrollsumme zu prüfen und erst dann auf das Ergebnisfeld zu addieren.

Das Demonstrationsprogramm hat weitere Steuerungsmöglichkeiten. Mit '7F' = DELETE in der Vergleichstabelle kann ein Byte des Datenfeldes aus dem Vergleich als irrelevant ausgeklammert werden, z.B. für die Gruppenbildung. Das Programm endet, wenn ein letzter Datensatz mit dem FILE SEPARATOR '1C' erkannt wird. Der KIM-Monitor zeigt dann in endloser Schleife 'EF EF EF' an. An dieser Stelle könnte man natürlich die Ergebnisse ausdrucken.

Die reservierten Zeichen sind damit:

- *1C = File Separator*
- *2F = Satzende-Marke (KIM)*
- *EF = Feldende-Marke im Datensatz*
- *FF = Feldende-Marke in der Vergleichstabelle*
- *7F = DELETE, Auslassungs-Marke im Vergleichsfeld.*

SUMMARY akzeptiert schnellen HYPERTAPE in der Dateneingabe.

Zum Programmaufbau: Das Ergebnisfeld ist in 17C0 bis 17E6 angelegt. Es wird bei ANF zunächst auf Null gelöscht, und zwar mit dem 'hidden opcode' 9C. Dieser von mir mit STZ bezeichnete Code arbeitet absolut, mit X indiziert und speichert 00 ins RAM. Die nach BEG aufgerufenen Unterprogramme initialisieren die Pointer für das Vergleichsfeld (1780) und für das Ergebnisfeld. Es folgt eine Parameterübergabe an den Stack für Y=0 und für dezimalen Status ohne Carry. Aufruf LOADT+2 als Unterprogramm. Ab CHINA (CHaracter IN A) Prüfung auf reservierte Zeichen und Paarigkeit des Datenfeld-Etikettes mit der Vergleichstabelle. Bei Gleichheit Erhöhung des Pointers für die Tabelle (sonst zurück auf Anfangswert), Prüfung auf Ende der Vergleichstabelle, Beschaffung der Y-Zählung und des Status vom Stack, Addition und Wiederablage dieser Parameter auf dem Stack. Die Programmteile ADJ, NONHEX und ENDFLE nehmen die notwendige Berichtigung des Stackpointers beim Verlassen der Hauptroutine vor.

Die Parameterübergabe hätte natürlich nicht über den Stack erfolgen müssen, sie ist aber sehr bequem und man kann sehr entfernte Programmteile auch so mit Angaben versorgen.

Nach dieser eingehenden Erklärung in Deutsch folgen die Kommentare zum listing in Englisch - für die Leser im Ausland.

```
                    .SUMMARY
0200  A2 26     ANF     LDX #$ 26       FOR 27 LOCATIONS
0202  9C C0 17 ANF1     STZ COUNT,X     HIDDEN CODE TO STORE ØØ
0205  CA                DEX             TO RESULTS FIELD
0206  10 FA              BPL ANF1        STORE UNTIL DONE
0208  20 68 02 BEG      JSR SETPNT      SET TWO POINTERS
020B  20 71 02           JSR SETCPT
020E  A9 00              LDA #$ 00       PARAMETER Y = Ø
0210  48                 PHA             VIA STACK
0211  18                 CLC             NO-CARRY STATUS AND
0212  F8                 SED             DECIMAL MODE ...
0213  08                 PHP             TO STACK
0214  D8                 CLD             RETURN BINARY MODE

0215  A9 60              LDA #$ 60       RTS-OPCODE FOR VEB IN KIM
0217  20 75 18           JSR LOADT+2     GOSUB FOR INITIAL LOAD

021A  C9 1C     CHINA    CMP #$ 1C       TEST FOR FILE SEPARATOR
021C  F0 3A              BEQ ENDFLE      LEAVE WHEN MET
021E  C9 2F              CMP #$ 2F       TEST FOR END OF RECORD
0220  F0 73              BEQ ADJ1        ADJUST SP & GET NEXT RECORD

0222  A0 00              LDY #$ 00       FOR INDIRECT ADDRESSING
0224  D1 FA              CMP (POINTL),Y  COMPARE TO TABLE
0226  F0 0F              BEQ MATCH       IF EQUAL:SKIP
0228  B1 FA              LDA (POINTL),Y  GET CHARACTER FROM TABLE
022A  C9 7F              CMP #$ 7F       IS IT A 'DELETE' ?
022C  F0 09              BEQ MATCH       THEN TREAT AS A MATCH
022E  20 68 02           JSR SETPNT      ELSE RESET TABLE POINTER
0231  20 81 02 NEXTCH    JSR READ        & GET NEXT CHARACTER
0234  4C 1A 02           JMP CHINA       AND TEST FROM SCRATCH

0237  20 63 1F MATCH     JSR INCPT       ADVANCE TO NEXT TABLE VALUE
023A  B1 FA              LDA (POINTL),Y  AND LOAD IT TO A
023C  C9 FF              CMP #$ FF       TEST FOR END OF TABLE
023E  D0 F1              BNE NEXTCH      DO MORE TESTING IF NOT

0240  20 81 02 NEXTNR    JSR READ        GET NEXT CHARACTER FROM TAPE
0243  C9 EF              CMP #$ EF       TEST FOR END OF FIELD
0245  F0 4E              BEQ ADJ1        ADVANCE TO NEXT RECORD

0247  28                 PLP             STATUS FROM STACK
0248  AA                 TAX             SAVE CH. IN X
0249  68                 PLA             GET LAST Y-COUNT
024A  A8                 TAY             AND MOVE TO Y
024B  8A                 TXA             RESTORE A

024C  71 EC              ADC (CPT),Y     ADD TO CONTENTS OF COUNTER
024E  91 EC              STA (CPT),Y     GIVING NEW VALUE
0250  C8                 INY             ADVANCE Y FOR NEXT ADDRESSING
```

```
0251   98              TYA          AND SAVE CURRENT COUNT
0252   48              PHA          ON STACK
0253   08              PHP          SAVE POSSIBLE CARRY & MODE ON STACK
0254   D8              CLD          RETURN TO BINARY MODE
0255   4C 40 02        JMP NEXTNR   GET NEXT CHARACTER

0258   68 68   ENDFLE  PLA, PLA     ADJUST SP
025A   A9 EF           LDA #$ EF    LOAD THE DISPLAY
025C   85 F9           STA INH
025E   85 FA           STA POINTL
0260   85 FB           STA POINTH
0262   20 1F 1F SHOW   JSR SCANDS   AND SHOW "EF EF EF"
0265   4C 62 02        JMP SHOW     CONTINUOUSLY

0268   A9 80   SETPNT  LDA #$ 80    SET POINTER ADDRESS OF TABLE
026A   85 FA           STA POINTL   TO 1780
026C   A9 17           LDA #$ 17
026E   85 FB           STA POINTH
0270   60              RTS

0271   A9 C0   SETCPT  LDA #$ C0    SET POINTER ADDRESS OF COUNTER
0273   85 EC           STA CPT      TO 17C0
0275   A9 17           LDA #$ 17
0277   85 ED           STA CPTH
0279   60              RTS

027A   E6 EC   INCCPT  INC CPT      CPT = CPT + 1
027C   D0 02           BNE OUT      SKIP ON NO CARRY
027E   E6 ED           INC CPTH
0280   60       OUT    RTS

0281   A2 02   READ    LDX #$ 02    CODING CORRESPONDS TO LOADT7
0283   20 24 1A READ1  JSR READCHT  IN KIM MONITOR
0286   C9 2F           CMP #$ 2F    END OF RECORD?
0288   F0 09           BEQ ADJ      ADJUST AND GET NEXT RECORD
028A   20 00 1A        JSR PACKT
028D   D0 0B           BNE NONHEX
028F   CA              DEX
0290   D0 F1           BNE READ1    DONE?
0292   60              RTS

0293   68 68   ADJ     PLA, PLA     ADJUST SP
0295   68 68   ADJ1    PLA, PLA
0297   4C 08 02        JMP BEG      GET NEXT RECORD

029A   68 68   NONHEX  PLA, PLA     ADJUST SP
029C   68 68           PLA, PLA
029E   4C 29 19        JMP LOADT9   ERROR EXIT TO KIM MONITOR
                                    DISPLAY 'FF FF'
02A1   EA       END    NOP          RALOAD-BYTE FOR RELOCATION
ZERO PAGE USED

00F9            INH
00FA            POINTL              Load ID = 00 to 17F9.
00FB            POINTH              Last record should contain
00EC            CPT                 FILE SEPARATOR = 1C as the
00ED            CPTH                sole byte.
```

R.L.

S T R I N G H U N T E R

E: As in Programs HEADHUNTER, STATISTICIAN and SUMMARY the LOADT-Routine contained within the KIM monitor is somewhat altered in order to identify a contiguous string of characters within a tape recorded file. The latter may comprise any number of records of different format.

A table containing the requested string of characters is to be set up in advance at any place in RAM for comparision purposes. STRINGHUNTER searches the data records nonstop until it finds a full match to table then loads remainder of record in process into RAM. If a last record with the sole file separator byte '1C' is met without a prior match, KIM will return displaying 'EF EF EF'. This little program should be applied to text files or a dictionary in order to find additional information for the string in question. Single characters can be masked by '7F' = DELETE.

STRINGHUNTER ist wiederum ein Auswertungsprogramm für eine auf Magnetband gespeicherte und auf mehrere Bandsätze verteilte Datei. Die Bandladeroutine LOADT im KIM-Monitor wird in Hinsicht auf ein sparsames Coding verwertet, und zwar unter Verzicht auf eine Überprüfung der Kontrollsumme.

Das Programm sollte auf Textdateien angewandt werden, um in irgendeinem der Datensätze eine volle Entsprechung zu einer Textkette zu finden, die man zuvor in einer Vergleichstabelle niedergelegt hat. Sobald ein gleicher String gefunden ist, lädt das Programm den Rest des in Arbeit befindlichen Datensatzes in das RAM.

Die erste Einlesezelle wird dabei beim Lesen mit ID = 00 vom Vorspann des Datensatzes her bestimmt, beim Lesen mit ID = FF entspricht sie dem Vektor in SAL/SAH. Die Information steht ab dieser Speicherzelle zur weiteren Analyse oder zum Ausdruck zur Verfügung. - Man könnte das Programm auch DICTIONARY nennen, es beschafft zu einem Suchbegriff weitere Information aus den Datensätzen.

Wenn es in den Datensätzen keine Entsprechung für den Suchbegriff gibt, kehrt KIM mit der Anzeige 'EF EF EF' zurück, sobald ein letzter Datensatz mit dem FILE SEPARATOR '1C' gefunden wird.

Reservierte Zeichen wie in SUMMARY, EF nicht in Gebrauch
Reserved Characters as in SUMMARY, EF not used.

Mit dem Zeichen '7F' = DELETE in der Vergleichstabelle kann ein Zeichen des Strings 'maskiert', aus dem Vergleich ausgeklammert werden, um ähnliche Wortbildungen abzudecken, wie etwa 'MACRO' contra 'MICRO'. Ein 'FF' soll die Vergleichstabelle begrenzen. Natürlich können auch Steuerzeichen in der Textkette enthalten sein, um sich an die Feldgliederung in den Datensätzen anzupassen.

Lade mit ID = 00 oder ID = FF in 17F9.
Load with or

Zero Page used:

00FA	*POINTL*
00FB	*POINTH*
00F9	*INH*

STRINGHUNTER (ACCEPTS HYPERTAPE)

```
1780  20 DD 17 ANF     JSR SETPNT      SET POINTER TO BASE OF TABLE
1783  A9 60            LDA #$ 60       RTS-OPCODE FOR VEB
1785  20 75 18         JSR LOADT+2     KIM ROUTINE TO CHECK HEADER ON TAPE
1788  C9 1C    CHINA   CMP #$ 1C       TEST FOR END OF FILE
178A  F0 43            BEQ ENDFLE      EXIT TO KIM
178C  C9 2F            CMP #$ 2F       TEST FOR END OF RECORD
178E  F0 F0            BEQ ANF         TO NEXT RECORD
1790  AA               TAX             SAVE CHAR. IN X

1791  A0 00            LDY #$ 00       FOR INDIRECT ADDRESSING
1793  D1 FA            CMP [POINTL],Y  COMPARE TO TABLE
1795  F0 0C            BEQ MATCH
1797  B1 FA            LDA [POINTL],Y  IF ≠ LOOK FOR A MASK
1799  C9 7F            CMP #$ 7F
179B  F0 06            BEQ MATCH       TREAT AS A MATCH

179D  20 DD 17         JSR SETPNT      POINTER TO BASE ON NO MATCH
17A0  4C AC 17         JMP READ        GET NEXT CHARACTER

17A3  20 63 1F MATCH   JSR INCPT       POINTER = POINTER + 1
17A6  A9 FF            LDA #$ FF       TEST NEXT VALUE IN TABLE FOR
17A8  D1 FA            CMP [POINTL],Y  DELIMITER 'FF'
17AA  F0 1A            BEQ LADE        LOAD AFTER END OF TABLE DETECTED

17AC  A2 02    READ    LDX #$ 02       CORRESPONDS TO LOADT7  IN KIM
17AE  20 24 1A READ1   JSR RDCHT
17B1  C9 2F            CMP #$ 2F
17B3  F0 CB            BEQ ANF         NEW TURN AFTER END OF RECORD
17B5  20 00 1A         JSR PACKT
17B8  F0 03            BEQ HEX         SKIP ON HEX
17BA  4C 29 19         JMP LOADT9      DISPLAY 'FF FF' ON NON-HEX
17BD  CA               DEX
17BE  D0 EE            BNE READ1       DONE?
17C0  20 4C 19         JSR CHKT        COMPUTE CHECKSUM
17C3  4C 88 17         JMP CHINA       DO MORE COMPARISIONS


17C6  A9 8D    LADE    LDA #$ 8D       OPCODE 60 IN VEB IS REPLACED
17C8  8D EC 17         STA VEB         BY 8D = STA
17CB  8A               TXA             CHARACTER RESTORED
17CC  4C F8 18         JMP LOADT 7     LOADING BY KIM BEGINS

17CF  A9 EF    ENDFLE  LDA #$ EF       LOAD DISPLAY
17D1  85 F9            STA INH
17D3  85 FA            STA POINTL
17D5  85 FB            STA POINTH
17D7  20 1F 1F SHOW    JSR SCANDS      AND DISPLAY 'EF EF EF'
17DA  4C D7 17         JMP SHOW        ONCE MORE

17DD  A9 00    SEPNT   LDA #$ 00       SET POINTER TO ADDRESS 0100
17DF  85 FA            STA POINTL      OR ELSE
17E1  A9 01            LDA #$ 01
17E3  85 FB            STA POINTH
17E5  60               RTS
17E6  EA       END     NOP             RALOAD BYTE FOR RELOCATION
```

R.L.

A S P - Advanced Subroutine Package

By Michael Zimmermann, Eberstädter Str. 170, 6102 Pfungstadt

E: The author starts a series on his Advanced Subroutine Package. In this issue you find the introduction. Subsequent articles will contain the listings.

ASP looks like a grand design for commercial applications, rendering decimal arithmetics, logical functions, packing/unpacking, editing for output and processing of tables. Everything is managed with only a few parameters, mainly for length and addresses. Processing is done directly - not interpreter-style. Thus the user is free to choose 'his' subset of the offered subroutines as required.

Ein großer Wurf: In einer Artikelfolge veröffentlicht Michael Zimmermann sein speziell für kaufmännische Anwendungen konzipiertes Unterprogrammpaket. Die nachfolgende Inhaltsübersicht gibt bereits einen Eindruck von der Fülle der angebotenen Dienstleistungen. Lizenzvergabe nur direkt durch M. Zimmermann.

Inhaltsübersicht

0. Einleitung

0.1 Warum ASP ?

Wie die meisten der derzeit erhältlichen Mikroprozessoren sind auch diejenigen der 65xx-Systemfamilie (bislang) hardwaremäßig auf die Verarbeitung 8 Bit breiter Zeichen eingerichtet. Bei vielen Anwendungen, besonders bei den kaufmännischen, möchte man eine größere Arbeitsbreite erzielen. Prinzipiell bieten sich dafür zwei Wege an:

- Einsatz eines Taschenrechnerchips als Slave-Rechner unter der Kontrolle eines Mikrocomputers,
- Durchführung der Berechnungen über Software.

Während der erste Weg bei numerischen Daten noch durchaus gangbar erscheint, dürfte er für alphanumerische Daten indiskutabel sein; dem Verfasser sind zumindest keine Taschenrechnerchips bekannt, die Alphazeichen verarbeiten.

Der einzig erfolgversprechende Weg ist also, Operationen, die eine erhöhte Genauigkeit erfordern, über Software abzuwickeln.

Für die 65xx-Mikrocomputer wurde bereits viel Software veröffentlicht, teils in Zeitschriften, teils in Buchform. Zum einen Teil handelt es sich hierbei um Spiele und ähnlich irrelevante Programme, zum anderen sind es Routinen für eine sehr spezielle Hardware, die damit nur für einen kleinen Teil der Anwender interessant sind.

Darüber hinaus gibt es auch eine Reihe von Programmen, die insbesondere die Programmentwicklung erleichtern oder die bestimmte arithmetische Operationen zur Verfügung stellen. Sicher gibt es eine Floating-Point-Routine, aber für eine einfache dezimale Arithmetik wird nichts gleichwertiges angeboten.

Aufgabe von ASP soll es daher sein, hier eine Lücke zu schließen, sich dabei aber nicht nur auf die Arithmetik zu beschränken, sondern gleichzeitig Routinen anzubieten für Druckaufbereitung sowie Ein- und Ausgabe auf unterschiedlichen Medien.

0.2 Wie arbeitet ASP ?

Vor einer detaillierten Beschreibung der einzelnen Programme sollen die Design-Kriterien kurz dargestellt werden, um dem Benutzer die grundlegenden Gedanken von ASP näher zu bringen und ihn dadurch befähigen, bestehende Module besonderen Anforderungen anzupassen oder auch neue eigene Routinen hinzuzufügen.

Für die Bearbeitung von immer wiederkehrenden Programmabschnitten bieten sich zwei Vorgehensweisen an:

- Die Makrotechnik, bei der der wiederkehrende Programmabschnitt direkt in das Hauptprogramm eingebunden ist,
- Die Unterprogrammtechnik, bei der der wiederholt zu durchlaufende Teil aus dem Hauptprogramm ausgelagert ist und von diesem nur angesprungen wird.

Während die Makrotechnik allgemein auf Assembler und höhere Programmiersprachen beschränkt ist, sind Unterprogramme auch bei einer Systementwicklung in Maschinensprache sinnvoll einzusetzen. Darüber hinaus erfordert eine Makrotechnik, insbesondere wenn die die einzelnen Module häufig angesprochen werden, einen erhöhten Speicherbedarf, allerdings mit dem Vorteil kürzerer Ausführungszeiten.

Allgemein dürfte bei den betrachteten Maschinen der Speicherplatz ein größerer Engpaß sein als die Ausführungszeit. Aus diesem Grunde wurden sämtliche Moduln als Unterprogramme ausgelegt - eine Kombination mit der Makrotechnik (die Unterprogramme werden dann als Makros angelegt) ist jederzeit möglich

0.3 Was für ein Unterprogrammsystem ist ASP ?

Ein Unterprogrammsystem wie ASP kann generell in zwei verschiedenen Formen implementiert werden:

- Als interpretatives System,
- Als direktes Unterprogrammsystem.

Die erste Variante hat dann Vorteile, wenn mit dem Unterprogrammsystem eine eigene Programmiersprache verbunden ist oder wenn der Instruktionssatz einer anderen Maschine simuliert werden soll. Dieses ist aber nicht die eigentliche Zielsetzung von ASP.

Ein interpretatives System kann auch nur komplett installiert werden, während direkte Unterprogramme den Wünschen und Ressourcen des Anwenders viel besser angepaßt werden können.

Ebenso erspart ein direktes Unterprogrammsystem die Einführung einer Anzahl von Befehlen, wie z.B. Sprunginstruktionen, die auf der Host-Maschine sowieso implementiert sind.

0.4 Welche Parameter verwenden die ASP-Unterprogramme?

Da ASP generell Daten unterschiedlicher Länge verarbeiten soll und weil irgendwelche Feldmarken nicht unbedingt praktisch sind, muß mindestens einer der Parameter ein Längenschlüssel sein.

Aber wieviel Adreßparameter sind sinnvoll?

- Einadreßmaschinen erfordern eine größere Zahl von Instruktionen bei kurzer Instruktionslänge.
- Zweiadreßmaschinen sind ein weitverbreiteter Standard in der kommerziellen Datenverarbeitung, die Zahl der Instruktionen nimmt ab, die Länge der einzelnen Befehle zu.
- Vieladreßmaschinen sind heute unüblich und nur bei Wortmaschinen mit großer Wortlänge sinnvoll. Die einzelne Instruktion ist sehr lang und enthält meist eine Vielzahl redundanter Informationen. Der Code ist - von Sonderfällen abgesehen - lang.

Das beste Format für die ASP-Instruktionen dürfte daher aus einer Längenangabe und 2 Adressen bestehen. Es lehnt sich, und das ist durchaus beabsichtigt, an das Instruktionsformat der IBM 360 an.

Eine Reihe von Unterprogrammen wird von direkten ASP-Routinen aufgerufen, diese beziehen ihre Parameter direkt aus den Adreßpointern, die durch das direkte Unterprogramm geladen wurden.

Ebenso gibt es Unterprogramme, die für den Benutzer parameterlos sind; diese können nur direkt nach einem anderen Unterprogramm aufgerufen werden und verwenden dann dessen Adreßpointerstände.

Bei Anwendungen, die mehr Parameterangaben erfordern, so z.B. bei der Tabellenverarbeitung, wurden die Angaben auf 2 Routinen aufgeteilt.

0.5 An welcher Stelle stehen die ASP-Parameter ?

Bei den Maschinen der Serie 65xx gibt es verschiedene Möglichkeiten, Parameter an ein Unterprogramm zu übergeben:

- in den Registern,
- in festen Speicherstellen,
- in solchen Speicherstellen, die dem Unterprogrammaufruf folgen.

Die erste Möglichkeit dürfte wegen der Anzahl der Parameter nicht in Betracht kommen. Die zweite erfordert ein umständliches Laden der festen Speicherbereiche und ist damit normalerweise sehr platzraubend.

Die dritte Version dürfte hier das Optimum darstellen, es werden alle Bedürfnisse an Parameter befriedigt, gleichzeitig ist der Platzbedarf gering.

Die ASP-Aufrufe haben damit folgendes Format:

```
...
JSR   ASPUPRO    gewünschtes ASP-Unterprogramm
.BYTE LEN        Längenparameter
.WORD ADPARM1    Adreßparameter 1
.WORD ADPARM2    Adreßparameter 2
...
```

Eine Ähnlichkeit mit den SS-Befehlen der IBM 360 ist also nicht zu leugnen und durchaus beabsichtigt.

Neben diesen direkten ASP-Routinen verwendet das Package noch eine Reihe interner Unterprogramme, die im Zusammenhang mit den Modulen beschrieben werden, die sie verwenden. Hierfür gelten natürlich eigene Gesetzmäßigkeiten.

0.6 Welche Datentypen verabeitet ASP ?

ASP wurde in Hinblick auf kaufmännische Anwendungen entwickelt. Daher wurden alle Anwendungen primär auf alphanumerische Daten und Zahlen in gepackter Darstellung ausgerichtet. Bei gepackten Zahlen ist die höchstwertige Stelle an der niedersten Adresse im Speicher placiert und enthält '00' als positives bzw. '99' als negatives Vorzeichen.

Alle Daten, die von ASP verarbeitet werden, können sich frei im Speicher befinden, eine Bevorzugung einzelner Speicherbereiche z.B. als Register ist nicht gegeben.

Folgende Datentypen werden verarbeitet:

- numerisch binär
- numerisch gepackt
- alphanumerisch.

Eine Umwandlung ist nur zwischen gepackten und alphanumerischen Daten vorgesehen. Numerisch binäre Daten dienen primär der Adreßrechnung und sind nur beschränkt verwendbar.

Einzelheiten der Verwendung sind im Zusammenhang mit der Beschreibung der einzelnen Funktionen zu entnehmen.

0.7 Welche Einschränkungen bestehen für den Benutzer bei Verwendung von ASP ?

ASP bearbeitet Daten, die im Speicher sind. Aus diesem Grunde nehmen die ASP-Routinen keine Rücksicht auf die Verwendung der Register durch den Benutzer. Sollen also Registerstände über einen ASP-Aufruf hinweg erhalten bleiben, so hat der Benutzer selbst Maßnahmen zu ihrer Sicherung zu treffen.

Es werden folgende Zellen der Zero Page durch ASP benutzt:

E0 - EE als Adreßpointer,
C0 - DF als Zwischenspeicher.

Alle weiteren Speicherplätze stehen dem Benutzer unbeschränkt zur Verfügung. Für die Abspeicherung der ASP-Routinen muß der Anwender selber sorgen. Alle Routinen sind voll verschieblich, lediglich die Adressen der internen Unterprogramme sind zu ändern.

Fortsetzung folgt.

****

Entwicklung von Assemblern und Sprachcompilern

Die Verbesserung der Programmiermöglichkeiten liegt im Interesse aller Systembenutzer. Unsere Leser sind freundlich gebeten, über eigene oder ihnen zur Kenntnis gelangte Sprachentwicklung Dritter zu berichten. Ebenso interessieren Cross-Compiler, mit denen auf größeren Geräten der 65xx-Maschinencode erzeugt werden kann.

****

Zum Leserservice: Programmcassetten

Bei der Vielfalt der in Gebrauch befindlichen Rekorder traten Schwierigkeiten mit der Wiederlesbarkeit gelieferter Programme ein. An verbesserten Lösungen wird gearbeitet. Bis dahin wird der in Heft 1, Seite 7 versuchsweise angebotene Leserservice zur Lieferung von Programmcassetten unterbrochen. Geleistete Zahlungen bleiben zur Verfügung der Absender.

KIM-1 ALS STÖRUNGSANALYSATOR

Autor: Hans Häss

In Verbindung mit dem schnellen Metallpapierdrucker DS 3000 der Firma DS Elektronik, München, überwacht KIM-1 16 Meßstellen auf Pegeländerung H → L bzw. L → H. Der Drucker (Preis im Tischgehäuse etwa DM 998,-) hat eine eingebaute Quarzuhr für Datum und Uhrzeit und druckt in einer Zei noch 4 Stellen aus. Der Drucker 3000 arbeitet nach dem elektrosensitiven Druckverfahren, wobei je Zeile 20 alphanumerische Zeichen im 5 x 7 Punktraster gedruckt werden. Die Standard-Zeilenhöhe von 4,5 mm läßt sich durch Ändern eines Widerstandes variieren. Als Datenträger dient 60 mm breites Metallpaier in Rollen. Druckgeschwindigkeit 10 Zeilen/sec. Auflösung bei einer Zeile daher 100 ms.

Der Ausdruck erfolgt je nach Anwendung und EPROM entweder mit Jahreszahl und Minuten:

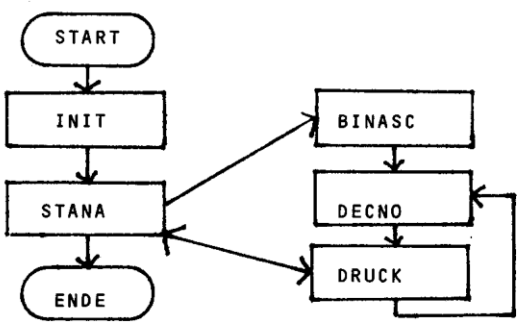
01.08.76 12:01 01 = L oder H

oder ohne Jahreszahl mit Sekunden

01.08. 12.01.16 01 = H oder L.

Bei Start werden zunächst alle Stellen ausgedruckt als Status. Dann nur noch die jeweils geänderte Stelle. Damit können Dauerversuchsprüfstände, Vacuumapparaturen o.ä. überwacht werden. Bei einer erfolgenden Abschaltung können Reihenfolge und Zeitverhalten der Abschaltung erkannt werden.

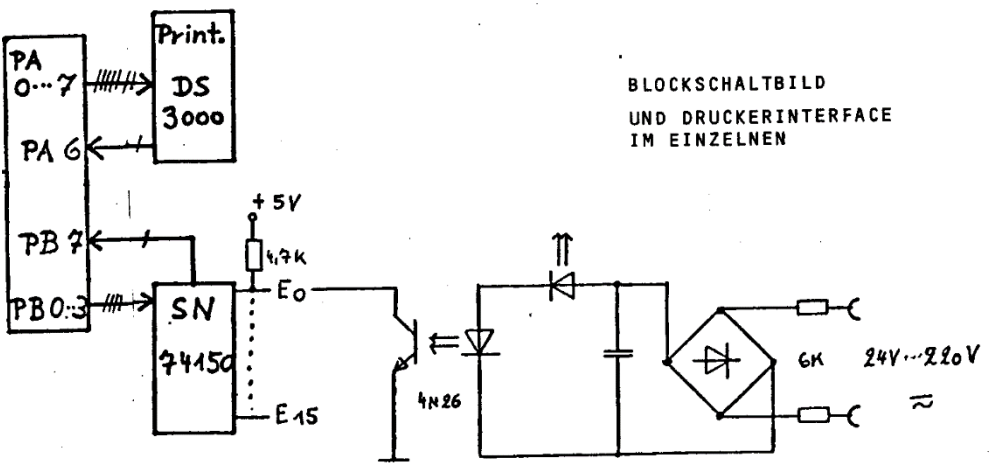
Die nachfolgenden Abbildungen bringen ein Blockbild für den Programmablauf, ein Blockschaltbild für den Druckeranschluß an KIM und das Druckerinterface im einzelnen.

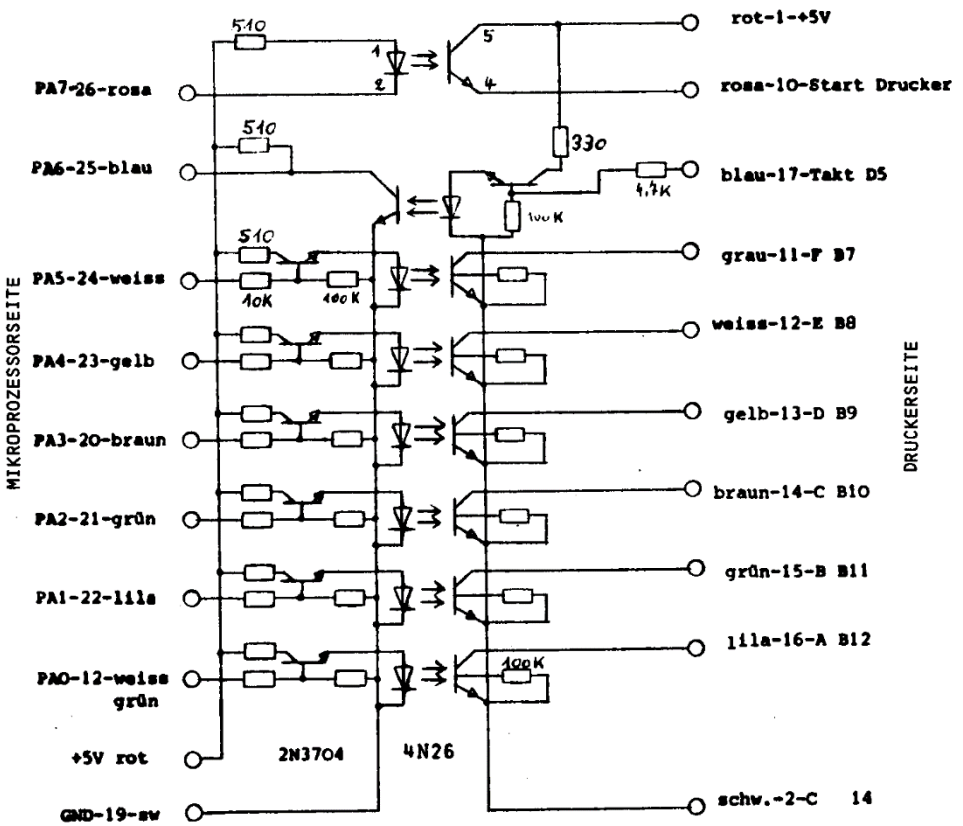



```
0200  A9 80    INIT    LDA #$ 80       SETZT BIT 7
0202  8D 00 17        STA PAD         SPERRT DRUCKER
0205  A9 BF            LDA #$ BF
0207  8D 01 17        STA PADD        A-PORT AUSGANG 7,5,4,3,1,0
020A  A9 0F            LDA #$ 0F              EINGANG 6      ZUM DRUCKER
020C  8D 03 17        STA PBDD        B-PORT AUSGANG 3,2,1,0
020F  A9 FF            LDA #$ FF              EINGANG 7,5,4  ZUM MU
0211  A2 0F            LDX #$ 0F
0213  95 00            STA 00,X        LADEN DER 16 SPEICHER MIT FF
0215  CA               DEX
```

BLOCKSCHALTBILD

UND DRUCKERINTERFACE
IM EINZELNEN





```
0216  10 FB          BPL          NACH -5
0218  D8             CLD

0219  A2 0F    STANA  LDX #$ 0F    FÜR 16 MESZSTELLEN
021B  8E 02 17       STX PBD      WÄHLT MESZSTELLE AN
021E  AD 02 17       LDA PBD      ANTWORT DER MESZSTELLE
0221  29 80          AND #$ 80    LIEST BIT 7
0223  D5 00          CMP ØØ,X
0225  D0 05          BNE          NACH +5
0227  CA             DEX
0228  10 F1          BPL          NACH -15, NÄCHSTE MESZSTELLE
022A  30 ED          BMI          NACH -19 VON VORN
022C  95 00          STA ØØ,X
022E  8A             TXA          MESZSTELLEN-NR.
022F  20 4F 02       JSR BINASC
0232  A9 02          LDA #$ 02    = '=' 3. ZEICHEN
0234  20 6D 02       JSR DRUCK
0237  B5 00          LDA ØØ,X
0239  C9 00          CMP #$ ØØ
023B  F0 0D          BEQ          NACH +13
023D  A9 33          LDA #$ 33    = 'L', 4. ZEICHEN
023F  20 6D 02 STA1  JSR DRUCK
0242  A9 80          LDA #$ 80
0244  8D 00 17       STA PAD      "STOP"-DRUCKER
0247  4C 19 02       JMP STANA
024A  A9 37          LDA #$ 37    = 'H', 4. ZEICHEN ALTERNATIV
024C  4C 3F 02       JMP STA1

024F  85 EE    BINASC STA 00EE     TEMPORÄR SPEICHERN
0251  A0 30          LDY #$ 30    = 'Ø'
0253  20 61 02       JSR DECNO
0256  A9 30          LDA #$ 30    = 'Ø'
0258  05 EE          ORA 00EE     REST BINÄRZAHL
025A  49 FF          EOR #$ FF    INVERTIERUNG
025C  4C 6D 02       JMP DRUCK
025F  85 EE          STA 00EE
0261  C8       DECNO INY
0261  18             CLC
0263  A5 EE          LDA 00EE
0265  69 F6          ADC #$ F6    MIT -10D "SUBTRAHEND"
0267  B0 F6          BCS          NACH -10
0269  88             DEY
026A  98             TYA
026B  49 FF          EOR #$ FF    INVERTIERUNG

026D  29 7F    DRUCK AND #$ 7F    LÖSCHT BIT 7
026F  8D 00 17       STA PAD      "START"-DRUCKER
0272  AD 00 17       LDA PAD
0275  29 40          AND #$ 40    LIEST BIT 6
0277  F0 F9          BEQ          NACH -7
0279  AD 00 17       LDA PAD      DRUCKER HOLT ZEICHEN
027C  29 40          AND #$ 40    LIEST BIT 6
027E  D0 F9          BNE          NACH -7
0280  60             RTS          ZEICHEN ÜBERNOMMEN
```

This program was originally published in the 'Feltron Nachrichten'. It prints out via matrix printer any change in the level of 16 measuring points, together with time and date which are supplied from a crystal clock within the printer.

ZAHLEN-WANDLUNG

. Günther Rüdiger
Ostlandring 12
2057 Reinbek

E: The author describes his algorithm for converting hexa-integers into decimal numbers and vice versa. KIM-1 acts as a table top calculator: Up to 5 hexa digits may be keyed in. On 'GO' their decimal equivalent will be displayed (6 digits). Two subroutines follow, converting up to 16 hexa digits to 20 decimal digits in memory et vice versa. Article will be continued.

Die im täglichen Leben verwendeten Zahlen verwenden die Basis 10, d.h. es gibt 10 verschiedene Zahlzeichen (0 bis 9), mit denen man auskommen muß, um alle nur denkbaren großen und kleinen Zahlen zu bilden. In der heutigen Zeit wird überall auf der Welt in den Schulen dieses System gelehrt. Man hat sich an das Dezimalsystem gewöhnt und kann leicht damit umgehen, obwohl jedes andere Zahlensystem mit einer anderen Basis, wie z.B. 2,8,12 oder 16, völlig gleichberechtigt an seine Stelle treten könnte. Vor- und Nachteile ergeben sich nur aus der Art der Anwendung, wobei Teilbarkeitseigenschaften häufig im Vordergrund stehen. Der Mensch hat der Anschaulichkeit halber die Zehn wegen seiner 10 Finger gewählt. Der Digitalrechner "braucht" als Basis die 2 wegen der beiden Spannungspegel oder Stromfluss-zustände. Der Maschinencode des KIM 1-Systems, wie vieler anderer Mikroprozessorsysteme, wird hexadezimal notiert, also die Basis 16 zugrunde gelegt, weil $2^4=16$ ist und außerdem eine Dezimalstelle sich im BCD-Code ebenfalls mit 4 Binärstellen darstellen läßt. Bei einer Reihe anderer Mikroprozessorsysteme verwendet man zur Codierung den Oktalcode (Basis 8).

Allgemein kann man feststellen, daß die Maschine bzw. das Problem ein bestimmtes Zahlensystem begünstigt und die Verwendung genau dieses Systems Vorteile bringt.

Trotzdem dürfen die menschliche Anschauung und die Anforderungen der Praxis nicht vergessen werden. Eine Umwandlung aus den diversen anderen Zahlensystemen in das Dezimalsystem und umgekehrt ist also häufig erforderlich!

Die immer wieder geforderte Gewöhnung an hexadezimale Zahlen beschränkt sich auch bei bestem Willen nur auf den Adressbereich bis 64 K oder wenig darüber hinaus. Bis zur Zahl FFFF behält der Engagierte den Überblick, darüber hinaus kommt auch er ohne Überschlagsrechnung nicht aus, wenn er sich ein Bild über die Größenordnung seiner Zahlen machen will, die hexadezimal notiert sind.

Mit keinem normalen Taschenrechner läßt sich leicht ermitteln, daß
8AC7230489E80000(16) =10Exponent19(10) oder
10Exponent16(16) = 18446744073709551616(10) ist.
Dies und noch einiges mehr liefern die Programme und Unterprogramme dieses Artikels!

Zunächst ein praktisches Programm mit bequemer Bedienung zur Einführung:

Es wandelt eine maximal 5-stellige hexadezimale Zahl, die direkt nach "Taschenrechnermanier" am Keyboard eingetastet und am Display angezeigt wird, per Knopfdruck ("GO"-Taste) in eine Dezimalzahl um. Nach dem Loslassen der "GO"-Taste wird die Anzeige wieder gelöscht und es kann sofort die nächste hexadezimale Zahl eingetippt werden. Die größte Zahl, die ohne Überlauf gewandelt wird, ist
F423F(16) = 999999(10)

Programm "HEX-DEZ-Wandlung mit Display" (PGR-Liste)

```
0200  20 30 02   BEGIN   JSR  CLEAR
0203  20 1F 1F   LOOP1   JSR  SCANDS
0206  F0 FB             BEQ  LOOP1
0208  20 6A 1F          JSR  GETKEY
020B  C9 13             CMP  #$13
020D  F0 0A             BEQ  KONV

020F  20 45 02          JSR  ROLL
0212  20 1F 1F   LOOP2   JSR  SCANDS
0215  D0 FB             BNE  LOOP2
0217  F0 EA             BEQ  LOOP1
0219  20 3A 02   KONV    JSR  CHANGE
021C  20 61 02          JSR  HEXDEZ
021F  20 57 02          JSR  MOVE
0222  20 3A 02          JSR  CHANGE
0225  20 1F 1F   LOOP3   JSR  SCANDS
0228  D0 FB             BNE  LOOP3
022A  F0 D4             BEQ  BEGIN
022C  EA                NOP
```

Subroutines

```
0230  A2 FA      CLEAR   LDX  #$FA     Speicherplätze
0232  A9 00              LDA  #$00     F6 bis FB in der
0234  95 FC      LOOP4   STA,X REG     Zeropage werden
0236  E8                INX           gelöscht!
0237  D0 FB              BNE  LOOP4
0239  60                RTS

023A  A5 FB      CHANGE  LDA  POINTH   Inhalt der Speiche
023C  48                 PHA           plätze F9 und FB
023D  A5 F9              LDA  INH      werden vertauscht!
023F  85 FB              STA  POINTH
0241  68                PLA
0242  85 F9              STA  INH
0244  60                RTS

0245  A2 04      ROLL    LDX  #$04     Inhalt der Speiche
0247  06 F9      LOOP5   ASL  INH      plätze F9 bis FB
0249  26 FA              ROL  POINTL   wird 4 Bit nach
024B  26 FB              ROL  POINTH   links verschoben
024D  CA                 DEX           und Inhalt des
024E  D0 F7              BNE  LOOP5    Akkumulators zu
0250  29 0F              AND  #$0F     F9 hinzuaddiert!
0252  05 F9              ORA  INH
0254  85 F9              STA  INH
0256  60                RTS

0257  A2 03      MOVE    LDX  #$03     Inhalt der Speiche
0259  B5 F5      LOOP6   LDA,X REGDEZ  F6 bis F8 wird
025B  95 F8              STA,X REGHEX  nach F9 bis FB
025D  CA                 DEX           übertragen!
025E  D0 F9              BNE  LOOP6
0260  60                RTS
```

```
0261  D8          HEXDEZ  CLD
0262  A0 18                LDY  #$18      nächstes Byte
0264  A2 03        LOOP7   LDX  #$03      holen und LSB und
0266  B5 F5        LOOP8   LDA,X BYTE     MSB getrennt auf
0268  F0 12                BEQ  ZERO      Größe untersuchen
026A  48                  PHA            und nach dem
026B  29 0F               AND  #$0F      Algorithmus korri-
026D  C9 05               CMP  #$05      gieren!
026F  68                  PLA
0270  90 02               BCC  COMP1
0272  69 02               ADC  #$02
0274  C9 4F        COMP1   CMP  #$4F
0276  90 02               BCC  COMP2
0278  69 2F               ADC  #$2F
027A  95 F5        COMP2   STA,X BYTE     nach Korrektur
027C  CA           ZERO    DEX            zurückspeichern!
027D  D0 E7               BNE  LOOP8
027F  A2 06               LDX  #$06
0281  36 F5        LOOP9   ROL,X REG      Verschiebung des
0283  CA                  DEX            gesamten Registers
0284  D0 FB               BNE  LOOP9     um 1 Bit nach
0286  88                  DEY            links!
0287  Do DB               BNE  LOOP7
0289  60                  RTS
                  ENDE
```

Durchlaufzeit der Unterprogramme:
"HEXDEZ": 3.8 ms (durchschnittlicher Wert, je nach Zahl)
"CLEAR" : 63µs "CHANGE": 25µs "ROLL": 95µs "MOVE": 46µs

Die eigentliche Wandlungsroutine findet sich als Unterprogramm bei Adresse 0261. Bevor diese Routine , die übrigens voll verschieblich ist, verallgemeinert wird, soll ganz kurz der Algorithmus, der dahinter steckt, genannt, wenn auch nicht im einzelnen erläutert werden:

Es handelt sich um die Methode des "shift and correct", die in schnellen Rechnern hardware-mäßig implementiert ist z.B. unter Verwendung des TTL-Bausteines 74185A, und der hier software-mäßig simuliert wird. Näheres im "The TTL-DATA-BOOK, Supplement to CC-401"von Texas Instrument, S.400. Analog wird bei der inversen Wandlung der Baustein 74184 simuliert (S.398 im gleichen Buch).

Mit Hilfe dieses Algorihmus lassen sich leicht beliebig große Zahlen zur Basis 16 in solche zur Basis 10 umwandeln und umgekehrt. Es folgen 2 Unterprogramme, die dies fuer maximal 16-stellige Hexadezimalzahlen bzw. maximal 20-stellige Dezimalzahlen bewirken. Es handelt sich dabei jeweils um ganze Zahlen, wie auch unter dem Wort "Zahl" in diesem Artikel immer nur ganze Zahlen verstanden werden sollen.

Vor dem Auslisten der Unterprogramme soll die Struktur der notwendigen Register beschrieben werden:

Register"REGDEZ" belegt die Speicherplätze C0 bis C9 in der Zeropage und nimmt in gepackter Form entweder vor (Eingabe) oder nach (Ausgabe)dem Durchlaufen der entsprechenden Routine die jeweilige Dezimalzahl auf. Register "REGHEX" belegt die Speicherplätze CA bis D1 in der Zeropage und ist im gleichen Sinne für die Hexadezimalzahl reserviert.

Vor dem Durchlaufen der gewünschten Routine muß jeweils das Zielregister leer, d.h. mit Nullen gefüllt sein. Bei der Wandlung von hexadezimal zu dezimal wird das Bitmuster bei gleichzeitiger Korrektur bitweise nach links verschoben, bei der inversen Wandlung bitweis nach rechts. Es stehen also jeweils die höchstwertigen Bits im niedrigstwertigen Speicherplatz,die niedrigstwertigen Bits im höchste Speicherplatz. Dies ist bei der Datenein- und Ausgabe zu beachten.

Subroutine "HEXDEZ" für 16 Hex- bzw. 20 Dez-Stellen

```
0300  D8        HEXDEZ  CLD
0301  A0 40               LDY   #$40
0303  A2 0A      LOOP7   LDX   #$0A
0305  B5 BF      LOOP8   LDA,X BYTE
0307  F0 12               BEQ   ZERO
0309  48                 PHA
030A  29 0F              AND   #$0F
030C  C9 05              CMP   #$05
030E  68                 PLA
030F  90 02              BCC   COMP1
0311  69 02              ADC   #$02
0313  C9 4F      COMP1   CMP   #$4F
0315  90 02              BCC   COMP2
0317  69 2F              ADC   #$2F
0319  95 BF      COMP2   STA,X BYTE
031B  CA         ZERO    DEX
031C  D0 E7              BNE   LOOP8
031E  A2 12              LDX   #$12
0320  36 BF      LOOP9   ROL,X REG
0322  CA                 DEX
0323  D0 FB              BNE   LOOP9
0325  88                 DEY
0326  D0 DB              BNE   LOOP7
0328  60                 RTS
```

Subroutine "DEZHEX" für 20 Dez- bzw. 16 Hex-Stellen

```
0329  D8        DEZHEX  CLD
032A  A0 40               LDY   #$40
032C  18                  CLC
032D  A2 EE      LOOP12   LDX   #$EE
032F  76 D2      LOOP10   ROR,X REG
0331  E8                  INX
0332  D0 FB               BNE   LOOP10
0334  A2 0A               LDX   #$0A
0336  B5 BF      LOOP11   LDA,X BYTE
0338  F0 12               BEQ   ZERO
033A  48                  PHA
033B  29 0F               AND   #$0F
033D  C9 08               CMP   #$08
033F  68                  PLA
0340  90 02               BCC   COMP1
0342  E9 03               SBC   #$03
0344  C9 7F      COMP1    CMP   #$7F
0346  90 02               BCC   COMP2
0348  E9 30               SBC   #$30
034A  95 BF      COMP2    STA,X BYTE
```

S. 34

EIN ERSTER BLICK AUF DEN AIM 65

65xx MICRO MAG brachte in Heft 1 die Ankündigung des neuen 'kleinen' Entwicklungssystems von Rockwell. Im Juli hatte Herr Michael Zimmermann Gelegenheit, dieses System erstmals zu prüfen. Wir zitieren aus seinem Bericht, soweit er zusätzliche Informationen enthält:

... Während die Hardware schon einen guten Eindruck macht, ist die Software nahezu phantastisch zu nennen. Das Betriebssystem übernimmt nicht nur die Ansteuerung der peripheren Einheiten, sondern unterstützt den Benutzer ganz entscheidend. Die Befehlseingabe erfolgt nicht als Hexa-Wert sondern in symbolischer Form; lediglich die Adressen sind hexadezimal einzugeben, wobei eine Relativierung bei Sprüngen vorgenommen wird.

Noch bequemer ist die Programmierung mit Assembler oder BASIC, die als Optionen in PROMs geliefert werden. Eine Unterstützung bei der Programmentwicklung stellt der Editor dar, der im Zusammenhang mit dem Assembler implementiert ist. Schon in der Grundausstattung ist ein Disassembler vorhanden.
Sämtliche Anweisungen für das Betriebssystem sind in bester Manier interaktiver Systeme als Dialog aufgebaut. So sind z.B. bei Ein- oder Ausgabe auf Kassette im Gegensatz zum KIM nicht bestimmte Speicherplätze zu füllen, sondern das System fragt den Benutzer nach Anfangs- und Endadresse des auszugebenden Speicherbereiches.

Vielfältig sind auch die Testhilfen. Es stehen Instruktions-Trace, Register-Trace und Program-Counter History zur Verfügung sowie vier Breakpoints.

Eine Vielzahl von Möglichkeiten ist für die Ein- und Ausgabe vorgesehen. Cassetten können sowohl im KIM-Format als auch in einem geblockten Binärformat verarbeitet werden. Und wenn dieses alles nicht ausreicht, so gibt es für den Benutzer noch die Möglichkeit, seine eigenen Ein- und Ausgaberoutinen aufzurufen.

Ebenso können 3 Benutzerfunktionen Bestandteil des Betriebssystems werden...

****

LITERATURHINWEISE

BYTE 2/78, S. 62, Dan Fylstra

SWEETS For KIM

Line Editor und Assembler für KIM. Einsprungspunkte für branches, jumps und JSR erhalten Etiketten. Listing etwa 2,5 pages. Sehr ausbaufähig für größere Speicher.

BYTE 3/78, S.18, Charles Helmers

An Apple to Byte

Anwenderbericht für den APPLE II durch den Herausgeber der Zeitschrift BYTE.

BYTE 3/78, S. 84, Steve Chiarcia

Program Your Next EROM in BASIC

Schaltplan für einen einfachen 2708-Programmierer nebst BASIC-Programm.

KILOBAUD 3/78, S. 90, M.L. Simon

Faster Erase Times
Selbstbau eines Löschgerätes für EPROMS.

ADRESSKONSTANTE VS. VERSCHIEBLICHKEIT

Das Thema wurde bereits in Heft 1 auf Seite 11 angeschnitten. Hier folgen weitere Überlegungen. Die Instruktionsfolge:

```
0200  18           CLC
0201  20 07 02     JSR 0207
0204  90 02        BCC            SKIP 2
0206  XX           .BYTE          PARAMETER
0207  60           RTS
...                ...
```

legt die Adresse des JSR+2 [0203] auf dem Stack ab, ohne den Programmablauf zu stören. Zusammen mit dem Parameter .BYTE kann eine solche Adresse z.B. einer Umrechung und anschließender Abspeicherung als Pointer in die Zeropage unterworfen werden, wenn man ein darauf ausgerichtetes Unterprogramm folgen läßt.

Ein anderes Beispiel für Adreßablage in einem Pointer:

```
0200  20 03 02     JSR 0203
0203  38           SEC            SUBTRAKTIONSVORBEREITUNG
0204  68           PLA            HOLE ADL VOM STACK
0205  E9 02        SBC #$ 02      EIN OFFSET-PARAMETER WIRD
0207  85 xx        STA PTRLO      VERARBEITET FÜR POINTERADRESSE
0209  68           PLA            HOLE ADH
020B  E9 00        SBC #$  00     BERÜCHSICHTIGE CARRY
020D  85 xx        STA PTRHI      POINTER HI
...                ...
```

Auch hier könnte man mit geringer Modifikation Parameter-Bytes auf den Aufruf folgen lassen. In vorstehendem Beispiel wurde willkürlich die Adresse 0200 in den Pointer gebracht.

Beide Arten der Adressenerzeugung sind durch ihre Generierung zum Zeitpunkt der Programmausführung unempfindlich gegen eine Verschiebung im Speicher, wenn man die Utility RALOAD einsetzt. Im Gefolge dieser Überlegungen werden Programm-Modifikationen und auch Längenbestimmungen möglich.

R.L.

****

Fortsetzung von "Zahlen-Wandlung":

```
034C    CA          ZERO    DEX
034D    DO E7               BNE    LOOP11
034F    88                  DEY
0350    DO DA               BNE    LOOP12
0352    60                  RTS
```

Es ist nun ohne Schwierigkeiten möglich, die Routinen sowohl bezüglich der Anzahl der zu verarbeitenden Stellen als auch bezüglich der Lage der beiden Register in der Zeropage den jeweiligen Bedürfnissen anzupassen. Außerdem kann man durch kleine Änderungen erreichen, daß die Wandlung von Zahlen zur Basis 16 zu solchen mit einer Basis von 2,4,6,8,10,12 oder 14 durchgeführt wird.
Die für einen universellen Anwendungsfall vorgesehenen Unterprogramme mit den angedeuteten Erweiterungen werden in der nächsten Nummer vorgestellt und erläutert.

VIDEOPROZESSOR-IC UND -KARTE

Mit dem SF.F. 96364 von Thomson-CSF als TV-Controller und wenigen weiteren Bausteinen kann aus einem handelsüblichen Fernsehgerät (CCIR-Norm, 625 Zeilen) ein preiswertes Datenterminal für die Anzeige von 16 Zeilen à 64 Zeichen aufgebaut werden. Die benötigten weiteren Schaltelemente betreffen Quarz, ein UART für die seriell-asynchrone Datenübertragung, einen quarzgesteuerten Baud-Rate-Generator, Bildschirm-RAM und Character-Generator. Anschluß entweder direkt an den Video-Eingang oder mittels UHF-Modulator an die Antennenbuchse.

Bequemer noch für den Anwender ist der Einsatz der Videoprozessor-Karte SFK 96364-00 (Preis etwa DM 580,-), die mit diesen Komponenten funktionsfähig und getestet geliefert wird. Diese hat dann parallelen ISO-7-bit-Eingang sowie Ein- und Ausgang zur seriell-asynchronen Übertragung bis 1200 Baud (RS 232) für Modem oder Mikrocomputer und einen Synchronausgang für 625 Zeilen ohne Zwischenzeile (TTL-kompatibel) sowie ein 1k x 6-bit Video-RAM.

Die Karte gibt volle Cursor-Kontrolle, Cursor blinking, Aufrollen der Textzeilen. Zeichenvorrat 64 ASCII-Zeichen in 5x7 Punktmatrix, Umwandlung von kleinen in Großbuchstaben. Zeichenbreite durch Poti einstellbar. Normale und inverse Darstellung. Versorgung einer ASCII-Tastatur und des Modulators über die Videokarte.

Der Controller-Chip allein wird in Kleinstückzahlen zu DM 82,90 angeboten (Kunststoff-Gehäuse) oder auch in keramischem Gehäuse, 28 Pin.

Datenblätter: Thomson CSF GmbH., Bereich Halbleiter, Fallstr. 42, D-8000 München 70 .

****

MESSEHINWEIS: Vom 9. bis 15. Nov. 1978 findet in München wieder die ELECTRONICA statt.

Gleichzeitig wird die erste deutsche IPE 78, INDUSTRIE und PERSONAL COMPUTER EXPO 78 im PSCHORR-Festsaaal neben dem Haupteingang der ELECTRONICA mit etwa 25 Firmen aus Europa und Amerika abgehalten. Weitere Info bei den Veranstaltern: KILOBAUD Europe uP-Literatur, Marktstr. 3 in D-7778 Markdorf und bei ABC-Computershop, Schellingstr.33 D-8000 München 40.

↑↑

KLEINANZEIGEN DER LESER:

Erfahrung mit OLIVETTI P6060 (ext. BASIC, Disketten), SR-52, TI-59/PC100A. Suche Erfahrungsaustausch über KIM-1, Z-80, Laufwerke für Disketten und Kassetten (ECMA): Runge, 1 Berlin 19, Lyckallee 21, Tel. (030) 30 42 118.

Terminal MOHAWK Data Sciences, Typ 2491, 3 Platinen, Bildschirm, Tastatur, Netzteil 220V, ohne Gehäuse zu etwa DM 700,- zu verkaufen. Darstellung von 8 Zeilen à 30 Zeichen: Rainer Köthe, DK6XN, Warnckesweg 52, 2000 Hamburg 61, Tel. 040-58 66 62.

%%%

ALS ABONNENT HABEN SIE DIE MÖGLICHKEIT, EINMAL JÄHRLICH EINE SOLCHE KLEINANZEIGE (BIS ZU 4 ZEILEN) GRATIS ABDRUCKEN ZU LASSEN. NICHTGEWERBLICHER INHALT VORAUSGESETZT.

65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

HERAUSGEBER:
DIPL.-VOLKSWIRT ROLAND LÖHR
HANSDORFER STRASSE 4
2070 AHRENSBURG
☎ (04102) 55 816

65xx MICRO MAG erscheint zweimonatlich als Manuskriptdruck. Beiträge, die nicht besonders gekennzeichnet sind, stammen vom Herausgeber.

COPYRIGHT 1978 BY ROLAND LÖHR. BEITRÄGE UND PROGRAMME DIENEN DEM PERSÖNLCHEN GEBRAUCH DES LESERS. NACHDRUCK UND GEWERBLICHE VERWENDUNG BEDÜRFEN DER VORHERIGEN SCHRIFTLICHEN GENEHMIGUNG.

BEZUGSBEDINGUNGEN: Abonnement für 6 Ausgaben im Inland DM 40,-, einschl. Versandkosten, Porto, Umsatzsteuer. Ausland DM 46,- (surface). Firmen erhalten Rechnung. Rechnungserteilung sonst nur auf Wunsch. Richten Sie bitte Ihre Überweisung/Euroscheck an:

Roland Löhr, Konto 20/01121, Vereins- und Westbank, BLZ 200 300 00.

Hinzukommende Abonnenten erhalten, wenn nicht anders vereinbart, Lieferung ab erster Ausgabe mit Laufzeit von da an. Einzelne Hefte können zu DM 7,- inkl. Porto nachbezogen werden.

Informationsblätter für Werbetreibende und Distributoren stehen zur Verfügung.

REDAKTIONS- UND ANZEIGENSCHLUSS FÜR NR. 3 IST DER 10. OKT.78.

In den nächsten Heften finden Sie u.a.:

Anschluß eines Metallpapierdruckers - Fortsetzungen für das kaufmännische Advanced Subroutine Package, für die Zahlenwandlung - Ablaufsteuerung durch 'stack riding' - Disassembler - Makros - Anwenderschaltungen - Berichte über AIM 65 und Pet ... - Grundsatzartikel.

Weitere Literaturhinweise

KILOBAUD 12/77, S. 30, Don Lancaster

TVT Hardware Design, Instruction Decoder and Screen.

Grundsatzartikel zur Zeichenerzeugung auf dem Bildschirm, mit insges. 21 Graphiken und Blockdiagrammen

KILOBAUD 1/78, S. 64, Don Lancaster

TVT Hardware Design, Low Cost Graphics

Zweiter Teil vorgen. Artikels. Vorabdruck aus dem 256 seitigen 'The Cheap Video Cookbook' (Sommer 1978 bei Sams ersch.) Speziell für 65xx und 6800.

KILOBAUD 5/78, S. 84, Adam Osborne

Number Chrunching, Two Hardware Solutions.

MM 57109 von NS und AM 9511 von Advanced Micro Devices werden als mögliche Tochterprozessoren für höhere mathemat. Funktionen besprochen. Überblick.

KILOBAUD 4/78, S. 104, D. Maciorowski

Displaying Hexadecimal

Breit angelegter Artikel mit Schaltbildern über die vielfältigen Darstellungsmöglichkeiten mit Ziffernanzeigen.

65xx MICRO MAG 65xx MICRO MAG 65xx MICRO MAG 65xx MICRO MAG