# 65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

# 3

OKTOBER 1978

*Zu diesem Heft*

*Dieses pünktlich erscheinende dritte Heft des auf die 65xx-Mikroprozessoren spezialisierten Journals bringt wieder eine anderswo nicht erreichte Fülle ernsthafter und didaktisch aufgebauter Programme für Systeme, wie KIM-1, VIM, AIM 65, PET und ALPHA.*

*Es ist ein kleines aber gehaltvolles Journal, das vorwiegend von engagierten Datenverarbeitungsfachleuten gestaltet wird, fern von unnötiger Aufmachung und Effekthascherei. In seiner Leserschaft, überwiegend Profis, hat es mit dieser Linie in kurzer Zeit Anerkennung als ein Fachblatt gefunden, ein Image, das für die weitere Arbeit natürlich verpflichtet - wobei selbstverständlich mehr Duplexverkehr und feed back sehr willkommen sind.*

*Neben einer Fortsetzung für die Zahlenwandlung und einem Beispiel für die Verarbeitung komplexer Tabellen im Disassembler bringt diese Ausgabe wohl überhaupt erstmals Makros für 65xx, Programmstrukturen ohne ausgefüllte Opcodes und Adressen. Es ist ein etwas abstraktes Thema, bei dem der Herausgeber hofft, daß ihm seine eigentlich noch zu knappe Darstellung gelungen ist. Die genauere Analyse zeigt, daß Herr Zimmermann mit seiner Fortsetzung für das ASP, das kaufmännische Package, ähnliche Denkansätze verfolgt.*

## INHALTSVERZEICHNIS

## TRANSSCRIBE HYPERTAPE TO QUICKDUMP

*Das leistungsfähige Paar QUICKDUMP und VERSALOAD (Heft 2) hat den Wunsch geweckt, vorhandene Programme, die in normalem KIM-Format oder in HYPER-TAPE archiviert wurden, auf das schnellere Format umzukopieren. TRANSSCRIBE bewältigt das problemlos. Der Rumpf des Programmes besteht aus Butterfield's SUPER-DUPE (siehe 'The First Book of KIM'). An entscheidender Stelle trennen sich jedoch die Wege: Wenn das Programm das Drücken einer Taste ungleich Null erkennt, werden die Parameter für QUICKDUMP geladen, das hernach als Unterprogramm abgearbeitet wird. Die Aufzeichnung erfolgt mit 1-Byte-ID. Programmbedienung sonst wie bei SUPER-DUPE.*

*Der Leser möge bei dieser Gelegenheit prüfen, ob er seine Programme durch Anwendung der RALOAD-Syntax (Heft 1) verschieblich machen will, indem er an ein Programm 'EA' oder die Tabellenbasisadresse anhängt. Achtung aber bei Adreßkonstanten im Programm! Ein Betriebssystem aus RALOAD, VERSALOAD und QUICKDUMP ist für die tägliche Entwicklungsarbeit äußerst nützlich.*

*Ds Listing für SUPER-DUPE in den Zellen 0000 bis 0072 wird hier nicht wiederholt.*

TRANSSCRIBE

```
0000-0072                          AS IN SUPER-DUPE
0073  0A          ASL
0074  F0 8A       BEQ START        IF KEY 'Ø' DEPRESSED
0076  A5 F9       LDA INH          MOVE PARAMETERS
0078  8D F9 17     STA ID
007B  A5 FA       LDA POINTL
007D  85 D6       STA OSAL
007F  A5 FB       LDA POINTH
0081  85 D7       STA OSAH
0083  A5 E0       LDA POINT
0085  8D F7 17     STA EAL
0088  A5 E1       LDA POINT+1
008A  8D F8 17     STA EAH
008D  A5 E2       LDA POINT2
008F  8D F5 17     STA SAL
0092  A5 E3       LDA POINT2+1
0094  8D F6 17     STA SAH
0097  A9 00       LDA #$ 00        PARMS FOR ENTRY1
0099  A2 01       LDX #$ 01
009B  20 0F 06     JSR ENTRY1       SUBROUTINE ENTRY INTO QUICKDUMP
009E  4C 5C 18  END JMP KIM         DISPLAY '00 00'
00A1  EA          NOP              RALOAD-BYTE
                                                          R.L.
```

**********

ERRATA - DRUCKFEHLER, HEFT 2, 65XX MICRO MAG

PAGE 12: ALTER CONTENTS OF 088F FROM D9 TO DØ IN VERSALOAD
PAGE 14: ALTER CONTENTS OF 096E FROM 0F TO 11 IN VERSALOAD

**********

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

# MAKROS FÜR 65XX

E: This first article in a series gives an introduction to macros. Subsequent editorials will cover several applications. Writing subroutines for a special program or for a resident program library generally will save storage and leads to more transparency of programs. Nevertheless subroutines may not be apt to be modified for opcodes, addresses or length. Macros offer the required flexibility for these items within preformed structures and will yield storage economies in many cases.

*Einleitung*

*Unter vorwiegend didaktischen Ordnungsgesichtspunkten geben wir eine Einführung in die leistungsfähige Makrotechnik.*

*Treten in einem Programm an verschiedenen Stellen gleiche Instruktionsfolgen auf, so wird man sie fast immer als ein Unterprogramm anlegen. Man spart bei der Programmniederschrift, spart Speicherplatz und muß weniger austesten. Ein solches Programm wird übersichtlicher und lesbarer, die Anwendung von Programm-Moduln ist sehr bequem. Nur bei zeitkritischen Abläufen wird man die Instruktionsfolgen linear ausprogrammieren.*

*Nehmen wir einmal die häufig benötigte Erhöhung eines Pointers in der Zeropage um den Wert 1. In Unterprogrammtechnik würde man schreiben:*

```
A2 XX              LDX #$ XX       LADE ADL DES POINTERS
20 YY YY           JSR UPRO
..                 ..

F6 ØØ       UPRO   INC ØØ,X        ERHÖHE ADL DES POINTERS UM 1
D0 03              BNE OUT         KEIN ÜBERLAUF?
E6                 INX             ADRESSIERUNG + 1
F6 00              INC ØØ,X
60          OUT    RTS.
```

*Der Aufruf hat jeweils 5 Bytes, das Upro 8. Bei linearer Programmierung würde das RTS fortfallen, es verblieben 7 Bytes. Sobald in einem Programm mehr als viermal irgendein Pointer erhöht wird, führt die Unterprogrammtechnik zur Einsparung von 2 Speicherstellen je Anwendung, z.B. bei fünf Aufrufen 5x5 + 8 gegen 5x7 bei linearer Programmierung. Bei langen Instruktionsfolgen kann Wirtschaftlichkeit schon beim zweiten Aufruf als Unterprogramm eintreten.*

*Man wird also überlegen, ob man eine Reihe solcher nützlichen Unterprogramme nicht einmal sorgfältig anlegt und in einer Programmbibliothek speichert. Diese 'program library' muß dann für die darauf bezugnehmenden Programme resident gehalten werden, sei es im RAM, sei es in einem PROM.*

*Ein solches Vorgehen setzt voraus, daß die Unterprogramme feststehend sind. Der aufmerksame Leser wird nun beobachtet haben, daß in der Programmiertechnik gewisse Strukturen häufig wiederkehren. So unterscheidet sich eine Additionsroutine von einem gleichwertigen Subtraktionsprogramm nur durch die Opcodes. Das Gleiche gilt für die Bearbeitung von Feldern durch die logischen Funktionen. Zwei funktionsgleiche Programme unterscheiden sich untereinander nur durch den erfaßten Adressenbereich und durch die Arbeitsbreite.*

*Wie erreicht man es nun, daß man solche Grundstrukturen nur einmal anlegen muß, um sie hernach beliebig mit Parametern für die auszuführenden Operationen, für die Adressenbereiche und für die Arbeitsbreite zu füllen? Dieses ist die Frage nach den Makros, den 'unvollständigen Strukturen', die erst mit ihrem Aufruf nach dem Willen des Programmierers zu einem bestimmten ausführbaren Programmabschnitt geformt werden.*

*Ein erstes einfaches Beispiel.*

*Nehmen wir einmal wieder die Erhöhung eines beliebigen Pointers, z.B. in 0033/34. Im Hauptprogramm erfolgt ein Makroaufruf wie folgt:*

```
20 XX XX    CALL    JSR MCDEMO
1B                  .LOP            LÄNGENOPERATOR FÜR 2 BYTES
33                  .BYTE           BEZEICHNUNG DES POINTERS LOW
..                  ...
```

*Das aufgerufene Unterprogramm laute wie folgt:*

```
0400  20 D3 03    MCDEMO  JSR ADRETA       ADJUST RETURN ADDRESS ON STACK
0403  BE 15 04    TRANS   LDX TAB-1,Y      GET ADDRESSER
0406  B1 D1                LDA (CAPL),Y    GET PARM FROM CALLER
0408  9D 0B 04              STA EXCUTE,X    INSERT INTO INSTRUCTION

040B  A2 &         EXCUTE  LDX #& &BYTE    BYTE FROM CALLER
040D  F6 ØØ                INC ØØ,X        INCREMENT LOCATION ADDRESSED BY X
040F  D0 03                BNE OUT         SKIP ON NO OVERFLOW
0411  E8                   INX             NEXT ADDRESSING
0412  F6 ØØ                INC ØØ,X        INCREMENT HI ORDER
0414  60           OUT     RTS
0415  0B                   .LOP            BYTE FOR RALOAD
0416  01           TAB     .BYTE           RELATIVE ADDRESS OF & LOCATION
0417  EA                   NOP             RALOAD-BYTE
```

*Damit nun MCDEMO in seinem Abschnitt EXCUTE bis OUT zu dem gleichen Ergebnis führt, wie das oben angeführte Unterprogramm, ist folgendes zu fordern: Das Upro ADRETA muß den verwendeten Pointer CAPL/H in OOD1/D2 auf die Adresse des LOP in CALL einstellen (CALL+3) und es muß mit Y = 1 zurückkehren. Dann kann in der zweiten Instruktion des Makro das X-Register mit dem Addresser und der Akku mit dem Byte (33) aus dem Caller geladen werden.*

*Mit der vierten Instruktion wird nun der Akkuinhalt zur Programm-Modifikation in das Programmsegment EXCUTE abgelegt. Die gewählte Zelle ist mit X indiziert, bezogen auf den Adreßpegel EXCUTE = OO.*

*An diesem Makro bemerken wir folgendes:*

*a) Das Makro fordert eine Adreßänderung auf dem Stack um die Länge der Parameterliste des Callers (ADRETA),*

*b) Es folgt ein Transport der Parameter vom Caller nach EXCUTE,*

*c) Eine Grundstruktur wird in EXCUTE vorgehalten. An den zu füllenden Stellen stehen dort 'Platzhalter': & (engl. Ampersand),*

*d) Ein Längenoperator LOP und ein NOP werden für Verschieblichkeit unter RALOAD vorgehalten (siehe Heft 1 des Journals),*

*e) Am Schluß steht eine Tabelle mit den relativen Adressen der Platzhalter.*

*An diesem speziellen Beispiel fällt weiterhin auf, daß der Aufwand für eine ganz einfache Instruktionsfolge viel zu groß ist. Das Laden von X mit dem Parameter ist umständlich. - Eine Ökonomie der Makroverwendung ergibt sich offensichtlich nur, wenn wenige Parameter in eine längere Grundstruktur eingefügt werden sollen. Wir werden dafür viele Beispiele haben.*

*Stackmanipulation für den Caller - ADRETA.*

*Zunächst wird jedoch das Hilfsprogramm (ADjust RETurn Adress) beschrieben, das die Rückkehradresse auf dem Stack nach CALL um die Länge der Parameterliste+1 (für den LOP) erhöht, das Y mit dieser Länge lädt und das einen Pointer auf die Basisadresse-1 der Parameterliste einstellt. Der Anfang des vollverschieblichen ADRETA wurde willkürlich auf 03 D3 gelegt, um für die späteren Makros einen glatten und übersichtlichen Anfang bei 0400 zu finden. Die Dienste des Unterprogrammes ähneln denen des MRA in Heft 1 dieses Journals.*

```
                        ADRETA (SUBROUTINE)

03D3  BA          ENTRY  TSX             STACK POINTER TO X
03D4  BD 03 01           LDA 0103,X      GET ADL FROM STACK
03D7  85 D1              STA CAPL        INTERIM POINTER
03D9  BD 04 01           LDA 0104,X      GET ADH FROM STACK
03DC  85 D2              STA CAPH        INTERIM HI ORDER
03DE  A0 01              LDY #$ 01       FOR INDIRECT ADDRESSING
03E0  B1 D1              LDA (CAPL),Y    GET LOP FROM CALLER
03E2  4A 4A              LSR, LSR        CUT OFF UNWANTED BITS
03E4  85 DC              STA LEN         SAVE FOR OTHER USE
03E6  A8                 TAY             SAVE IN Y TOO
03E7  18                 CLC             PREPARE ADC
03E8  65 D1              ADC CAPL        ADL+ LENGTH
03EA  9D 03 01           STA 0103,X      NEW RETURN ADDRESS FOR STACK
03ED  A5 D2              LDA CAPH        GET OLD ADH
03EF  69 00              ADC #$ 00       ADD CARRY STATUS
03F1  9D 04 01           STA 0104,X      NEW ADH FOR STACK
03F4  A5 D1              LDA CAPL        ADVANCE POINTER TO ADDRESS OF
03F6  69 01              ADC #$ 01       LOP IN CALLER
03F8  85 D1              STA CAPL
03FA  90 02              BCC OUT
03FC  E6 D2              INC CAPH
03FD  88                 DEY             LENGTH - 1 LOP
03FF  60                 RTS
0400  (EA)               NOP             NOT NECESSARY WHEN CONTIGUOUS
                                         WITH MACROS.
```

*MACSET - Set an Array to Constant.*

*Häufig möchte man ein Feld auf einen definierten Anfangswert setzen. Anfangswert und Adressenbereich müssen einstellbar sein. MACSET arbeitet bis zur Breite einer page und läßt auch sonst noch einige Tricks zu. Zunächst jedoch das Coding.*

```
                        MACSET (SUBROUTINE)

0400  20 D3 03    ENTRY   JSR ADRETA      SET POINTER, Y, AND STACK
0403  BE 1C 04    GENRTE  LDX TAB-1,Y     GET DISPLACEMENT
0406  B1 D1              LDA (CAPL),Y    GET PARM FROM CALLER
0408  9D 0E 04          STA EXCUTE,X      STORE WITH DISPLACEMENT
040B  88                DEY               ADDRESSER & COUNTER -1
040C  D0 F5             BNE GENRTE        ALL DONE ?
040E  A0 &      EXCUTE  LDY #$ &          LOAD WITH PARM
0410  A2 00             LDX #$ 00         ØØ TO POSSIBLE ADDRESSER
0412  A9 &              LDA #$ &          LOAD WITH PARM
0414  & & &     EX1      &    & &         TO BE FILLED WITH PARMS
0417  E8                INX               ADRESSER + 1
0418  88                DEY               COUNTDOWN
0419  D0 F9             BNE EX1           FIELD COVERED ?
041B  60                RTS
041C  1B                .LOP              INDICATING 6 BYTES
041D  01        TAB     .BYTE             DISPLACEMENT OF  & LENGTH
041E  05                .BYTE                              & CONSTANT
041F  06                .BYTE                              & OPCODE
0420  07                .BYTE                              & ADDRESS OR OP
0421  08                .BYTE                              & DITTO
0422  EA                NOP               RALOAD BYTE
```

*Dieses Makro würde man wie folgt aufrufen (Beispiel):*

```
20 00 04    CALL    JSR MACSET
1B                  .LOP
08                  .BYTE           & LENGTH = 08
55                  .BYTE           & CONSTANT = 55
9D                  .BYTE           & OPCODE = 9D = STA &&,X
00                  .BYTE           & ADL = 00
02                  .BYTE           & ADH = 02
....
```

*Syntax: Diesem Beispiel entnehmen wir, daß die Parameterliste im Aufruf eine volle Entsprechung in der Tabelle des Makro hat - und haben muß, denn es handelt sich um ein sog. Stellungsmakro.*

*Die Parameter müssen in bestimmter Reihenfolge und vollständig übergeben werden.*

*Zum Aufbau des Makro: Der Abarbeitung der Hilfsroutine ADRETA folgt eine Schleife mit dem Namen GENRTE (generate). Sie 'generiert' das Makro im Segment EXCUTE aus den übergebenen Parametern. Wohin diese Parameter gelangen, wird von der Tabelle gesteuert. Sie enthält displacements. Das sind relative Adressen bezogen auf den Programmzählerstand von EXCUTE=00 (displacement = Verschiebung).*

*Weiterhin fällt auf, daß nur das Segment EXCUTE modifiziert wird. Die Teile GENRTE und TAB bleiben also unverändert und behalten daher ihr Generierungsvermögen für spätere Aufrufe.*

*Programmiertechnisch mag interessant sein, daß Y in GENRTE als Zähler und als Addresser eingesetzt wird, um den zweiten Addresser X zu laden - LDX, absolut, Y.*

*Bei der Abarbeitung dieses Beispieles würden die Zellen 0200 bis 0207 (Arbeitsbreite 08) mit der Konstante '55' gefüllt, weil der Opcode mit X indiziert wurde. Würde man den & Opcode mit '99' einsetzen, so beträfe das Setzen des Speichers die Zellen 0201-0208. Wenn man also mit Y indizieren will, sollte man die 'OO' mitlaufen lassen und ein Makro schreiben, das mit BPL nach EX1 zurück in die Schleife geht. In einem solchen Makro könnte man natürlich das LDX (A2 OO) einsparen, ebenso das INX.*

*Der Anwender ist also vollkommen frei, seine eigenen Makros je nach Zweckmäßigkeit zu schreiben. An dieser Stelle können nur Rezepte und Beispiele gegeben werden.*

*Ruft man das Makro mit & LENGTH=OO, so arbeitet es in der Breite einer vollen page.*

*Wenn man beim ersten Aufruf nicht alle geforderten 6 Parameter durch den Caller übergibt, so entstehen Zufallsergebnisse, weil in den &-Positionen unkontrollierte Werte stehen. Anders ist es bei späteren Aufrufen: Im Beispiel würden zwei neue Parameter nur Länge und Konstante betreffen, also:*

```
20 00 04    CALL    JSR MACSET
0F                  .LOP
04                  .BYTE          & LENGTH   = 04
FF                  .BYTE          & CONSTANT = FF
```

*Mit ähnlichen verkürzten späteren Aufrufen könnte man in diesem Makro z.B. nur Länge, Konstante und Opcode verändern oder zusätzlich auch eine niedrige Adresse, ohne die hohe zu tangieren. Als solche Opcodes in späteren Aufrufen ist besonders an X-indiziertes ASL, LSR, ROL, ROR in memory zu denken, bei denen der Akkuinhalt unerheblich ist. Weniger naheliegend sind z.B. INC und DEC, mit X indiziert.*

*Nach Abarbeitung der Routine ADRETA ist das Carry-Flag gelöscht. Man könnte also auch alle Bytes eines Feldes in den Akku addieren, nachdem dieser per Konstante auf OO gesetzt war. Es ergäbe sich eine primitive Kontrollsumme für den Inhalt des Feldes. Immerhin besser als gar keine.*

*Wenn man die &-Instruktion in 0414 des Makro MACSET per Parameter als eine 2-Byte-Instruktion einsetzt, z.B. für Zeropage-Adressierung oder für indirekte Pointeradressierung mit Y, dann bleibt das letzte Parameterbyte frei. Normalerweise wird man es mit 'EA' übergeben, damit kein Schade entsteht. Wie aber, wenn man es mit DEY oder INX einsetzt? Dann betrifft die Operation doch - je nachdem - nur jede zweite Zelle oder es wird nur die halbe Anzahl Operationen ausgeführt. Man könnte also z.B. in einer Anordnung nebeneinanderliegender Pointer in der Zeropage gezielt nur die niedrigen oder die hohen Adressen auf einen Anfangsert setzen oder bei indirekter Adressierung jede zweite Zelle sonstwo im Speicher.*

*Diese Überlegungen zeigen schon, daß mit dem einfachen Makro MACSET eine beachtliche Reihe von Dienstleistungen verwirklicht werden kann, so daß es in vielen Fällen zu einer Ökonomie in der Speicherverwendung führt.*

*Dem Leser werden sicher weitere Einsatzmöglichkeiten für dieses und für die späteren Makros einfallen. Es wäre schön, wenn es hierüber ein feedback für den gesamten Leserkreis gäbe.*

*Die besprochenen Makros und Hilfsroutinen beanspruchen alle Maschinenregister. Der Anwender sollte prüfen, ob er vor einem Makroaufruf seine Register retten muß.*

MCPADD - *Increment a Zeropage Pointer by Value of a 2-Byte Constant - on Default by Value of a 1-Byte Constant.*

*Das erste Makro, MCDEMO, erhöht einen Pointer in der Zeropage um den Wert 1. MCPADD ist universaler, es addiert das erste Byte einer Konstanten zum Pointer-Low, das zweite in den Pointer-High, natürlich mit Berücksichtigung eines möglichen Übertrages.*

*Die Nutzanwendung liegt auf der Hand: In der Datentechnik treten häufig Fälle auf, in denen ein Datenfeld länger als 1 page ist, also mehr als 256 Bytes umfaßt. Insbesondere ist hier auch an das Einlesen längerer geblockter Datensätze in einen Pufferbereich zu denken, die vor ihrer Verarbeitung einer Identitätsprüfung unterliegen sollen.*

*MCPADD löst hier alle Probleme elegant. Und es tritt ein neuer Begriff auf: 'on default'. - Beim Fehlen eines an sich möglichen zweiten Parameters im Makroaufruf wird stattdessen ein Ersatzwert in das Makro eingesetzt, in unserem Falle 'OO'. Es ist also möglich, das Makro von Anfang an nur mit der Adresse low des Pointers und einer 1-Byte-Konstanten für die Pointererhöhung in Betrieb zu nehmen. Das zweite Konstantenbyte ist also eine Option.*

*Eine kurze Überlegung zeigt, daß der 'default value' in das Makro eingesetzt werden muß, ehe die Generierung beginnt. Läge die Generierung später, so würde sie einen möglicherweise bereits übergebenen Parameter zerstören. Andererseits wird vor jeder Generierung die Konstante-High auf 'OO' gebracht, so daß man ihren Wert nicht von einem zum verkürzten nächsten Aufruf stehen lassen kann. Letztere Möglichkeit würde aber das Makro noch eleganter machen, es folgt daher im nächsten Schritt eine Verfeinerung.*

```
                        MCPADD (SUBROUTINE)

0400  20 D3 03  ENTRY   JSR ADRETA       UTILITY
0403  A9 00     DFAULT  LDA #$ 00        TO INSERT DEFAULT VALUE
0405  8D 20 04          STA AP+ 1
0408  BE 24 04  GEN     LDX TAB-1,Y      GET DISPLACEMENT
040B  B1 D1             LDA (CAPL),Y     GET PARM FROM CALLER
040D  9D 13 04          STA EXCUTE,X     INSERT
0410  88                DEY
0411  D0 F5             BNE GEN
0413  18        EXCUTE  CLC              IN ADVANCE TO ADC
0414  A2 &              LDX #$ &         'IMMEDIATE DOLLAR AMPERSAND' ...
0416  B5 00             LDA ØØ,X                        ...ADDRESSER
0418  69 &              ADC CONSTL       ADD LOW BYTE
041A  95 00             STA ØØ,X         AND STORE BACK
041C  E8                INX              NEXT ADDRESSING
041D  B5 00             LDA ØØ,X         HI ORDER
041F  69 &       AP     ADC CONSTH       INCREMENT FOR HI
0421  95 00             STA ØØ,X         STORE BACK
0423  60                RTS

0424  13                .LOP             LENGTH
0425  02        TAB    .BYTE            DISPLACEMENT OF & POINTER LOW
0426  06                .BYTE                           & CONST1
0427  0D                .BYTE                           & CONST2
0428  EA                NOP              RALOAD BYTE.
```

MCPINC - *Versatile Increment of a Zeropage Pointer*

*Vom vorherigen Makro unterscheidet sich MCPINC nur durch die eingeschobenen Instruktionszeilen 2 und drei. Zur Bequemlichkeit folgt das volle Coding:*

```
                    MCPINC (SUBROUTINE)

0400  20 D3 03   ENTRY   JSR ADRETA
0403  C0 02              CPY #$ 02        LESS THAN 2 PARMS ?
0405  90 05              BCC GEN          YES, SKIP
0407  A9 00       DFAULT  LDA              AS IN MCPADD
0409  8D 24 04            STA AP+1
040C  BE 28 04            LDX
040F  B1 D1               LDA
0411  9D 17 04            STA
0414  88                  DEY
0415  D0 F5               BNE GEN
0417  18         EXCUTE  CLC                    FOR MCPSUB : SEC   38
0418  A2 &
041A  B5 00
041C  69 &                                                  SBC   E9 &
041E  95 00
0420  E8
0421  B5 00
0423  69 &                                                  SBC   E9 &
0425  95 00
0427  60
0428  13                  LOP
0429  02                  TAB
042A  06
042B  0D
042C  EA
```

*Mögliche Aufrufe für dieses Makro:*

```
      20 00 04    CALL    JSR MCPINC      FÜR 2-BYTE-KONSTANTE
      13                  .LOP
      &                   .BYTE           & ADDRESSER FOR ZEROPAGE
      &                   .BYTE           & CONSTANT LOW
      &                   .BYTE           & CONSTANT HI

      20 00 04    CALL    JSR             FÜR 1-BYTE-KONSTANTE MIT DEFAULT
      0F                  .LOP            ØØ FÜR DIE ZWEITE
      &                   .BYTE
      &                   .BYTE
      20 00 04    CALL    JSR             NUR ADDRESSER WIRD GEÄNDERT, DIE
      0B                  .LOP            KONSTANTEN BLEIBEN UNVERÄNDERT
      &                   .BYTE
```

*Im rechten Teil des Programmsegmentes EXCUTE sind mit SEC und SBC diejenigen Instruktionen kenntlich gemacht, die in ein sonst gleiches Makro MCPSUB einzusetzen wären. Dieses würde einen durch Parameter bezeichneten Pointer um den Wert einer Konstanten vermindern. Ob man beide Makros in eines fassen will, indem man auch diese Opcodes übergibt, ist Geschmackssache, denn die Aufrufe werden dadurch länger.*

*Speziell nun mit der dritten Aufrufmöglichkeit ergeben sich schöne Perspektiven und eine Ökonomie für die Verarbeitung von Tabellen und für die Bearbeitung aneinandergereihter (geblockt im Speicher stehender) Datensätze: Zu Beginn der Bearbeitung stellt man eine Reihe nebeneinander liegender Pointer mit einem Transport auf die Anfangswerte ein. Bei einem kaufmännischen Beispiel z.B. jeweils auf Kunden-Nr., Namen, PLZ, Ort usw.. Nach dem Durchprüfen oder Bearbeiten des Datensatzes erhöht man alle Pointer gleichmäßig auf die Basisadressen der Felder im zweiten Datensatz usw..*

*Mit MCPINC und MCPSUB in kombiniertem Einsatz kann man auch ein sehr speicherfreundliches 'binäres Tabellenlesen' verwirklichen, eine Anwendung, die hier noch nicht entwickelt wurde, über die man aber vielleicht aus dem Leserkreis hören wird?*

*Der Anwender sollte sich die Besonderheiten dieser beiden Makros vor Augen halten: Die verkürzte dritte Aufrufmöglichkeit ändert die Direktoperanden CONST1 und CONST2 nicht. Man muß also beim Laden oder mit einem früheren Aufruf dafür sorgen, daß hier die gewünschten Werte stehen.*

*Der Herausgeber hat anfangs natürlich auch ein Makro geschrieben, das einen Pointer mit nur einer 1-Byte-Konstanten erhöht. Die dabei einzusparenden etwa 7 Bytes lohnen eigentlich nicht drum.*

## SPEICHERUNG VON MAKROS IN DIESEM DESIGN - *Aufbau einer MACLIB*

*Ehe wir noch einmal die 'Syntax' für Makros dieser Bauart zusammenfassen ein Hinweis zur Abspeicherung. Makros werden nach dem Willen des Programmierers 'generiert'. Sie können daher während dieser Aufbauphase nicht in einem Festwertspeicher stehen, sondern müssen im Schreib-/Lesespeicher resident sein, in den die übergebenen Parameter eingesetzt werden können.*

*Das schließt natürlich nicht aus, daß Aufrufe der Makros aus Festwertspeichern heraus erfolgen.*

*Wir werden Makros kennenlernen, denen ein eigener kleiner Arbeitsspeicherbereich im Anschluß an die Tabelle zugeordnet ist. Auch aus dieser Sicht ist eine Generierung im RAM notwendig.*

*Damit ist zugleich auch ein Unterschied zu den Makros der großen Maschinen angesprochen. Deren Makros werden nach der Generierung als ein Abschnitt in das Hauptprogramm eingebunden; bei uns bleiben sie Unterprogramm außerhalb der Mainline.*

*Ganz sicher ist es möglich, das Segment zwischen EXCUTE und der Instruktion RTS in ein Hauptprogramm mit Generierung zu 'linken'. Es bedarf dazu nur gründlichen Nachdenkens. Bleibt aber die Frage, ob es bei den kleinen Geräten wegen des meist zu beschränkenden Speicherplatzbedarfes darum lohnt, denn bei den meisten Anwendungen wird man die Generierungsphase des Makros nach seinem Aufruf als Subroutine nicht als zeitschädlich für die Ausführung betrachten.*

*Makros im hier beschriebenen Design sollte man dem ausführenden Programm daher in der spezifisch benötigten Zusammenstellung in Form einer Makro-Bibliothek, Macro Library, MACLIB, zur Verfügung stellen, ggfs. im übergeordneten Betriebssystem. Auch hierzu wären Überlegungen aus dem Leserkreis interessant.*

*Alle unsere Makros sind für die Dienstprogramme RALOAD und VERSALOAD verschieblich geschrieben. Der Benutzer kann sie in benötigter Auswahl für seine Anwendungen aneinander linken.*

## STRUKTURMERKMALE DER MAKROS

*Die vorstehenden Beispiele erlauben es uns, jetzt schon einmal die wesentlichen Merkmale der Makros herauszuschälen.*

*Ein Makro ist eine vorgeformte Programmstruktur mit einer Reihe zunächst noch symbolischer Parameter (&, Stellvertreter). Erst mit dem Aufruf werden nach dem Willen des Programmierers Parameter (Opcodes, Adressen und/oder Werte) in die Plätze der Stellvertreter eingesetzt. Mit dieser 'Generierung' wird das Makro zu einem ausführbaren Programmsegment.*

*Ein Makro gliedert sich in die folgenden wesentlichen Abschnitte:*

*Organisations- und Generierteil,*

*Ausführungsteil,*

*Tabellenteil,*

*Ggfs. Arbeitsspeicher.*

*Als Verständigungsbereich zwischen Caller und Makro dienen Tabellen. Eine Tabelle im Anschluß an den Aufruf teilt die aktuell einzusetzenden Parameter mit. Die Tabelle im Makro enthält relative Adressen (displacements) der Zellen, in die diese Parameter einzusetzen sind. Die Einsetzung betrifft den Ausführungsteil (und ggfs. noch seinen Arbeitsspeicher).*

*Die Organisations- und Generierfähigkeit wird durch die Benutzung des Makro nicht verbraucht. Die vorgeformte Struktur kann daher vielfältig und wiederholt genutzt werden.*

*Die bisher beschriebenen Makros sind Stellungsmakros, d.h. die Parameter müssen in festgelegter Reihenfolge übergeben werden, und zwar im allgemeinen in kompletter Anzahl. Bei geschicktem Aufbau der Verständigungsbereiche sind jedoch auch verkürzte Parameterlisten möglich. In diesem Falle bleibt ein Teil der in einer früheren Generierung eingesetzten Parameter unverändert.*

*Eine weitere Option besteht in der Verwendung von 'Default-Parametern': Vor der eigentlichen Generierung des Makros werden erwünschte Standardwerte in Plätze der symbolischen Parameter eingesetzt. Wenn dann ein Makroaufruf mit verkürzter Parametertabelle erfolgt, so befinden sich an diesen Plätzen dennoch Standardwerte, die die Ausführbarkeit gewährleisten. Ist die übergebene Parameterliste des Callers dagegen vollständig, so überschreiben ihre aktuellen Parameter die default values.*

*Es handelt sich um Stellungsmakros. Die von den größeren Maschinen her bekannten Kennwortmakros wurden hier noch nicht vorgestellt und sie sind auch noch nicht ausgearbeitet. Bei letzteren werden die Parameter mit Namen und mit Wert, und zwar in einer beliebigen Reihenfolge übergeben, also z.B. PARM13=A9, PARM5=CC ... Der Vorteil solcher Makros: Man muß sich die Reihenfolge der Übergabe nicht merken und man kann ganz gezielt und kurz an wenige Stellen einsetzen, wenn man vorher default values in die Zellen der Platzhalter brachte.*

*Es ist klar, daß Kenntwortmakros zusätzlich zur Adreßtabelle auch eine Namenstabelle enthalten müssen, um die Identifizierung der übergebenen Parameternamen zu ermöglichen.*

*Für die Anlage und Verwendung eines Makros gibt es folgende Möglichkeiten:*

*a) als Generator-Makro in einer MACLIB,*

*b) als generiertes Unterprogramm in einem Objekt-Modul. Dies setzt voraus, daß das vom Generator erzeugte Unterprogramm (EXECUTE-Teil) ggfs. unter Umrechnung an seinen neuen Standort gebracht wird.*

*c) als linearer Abschnitt eines Objektprogrammes. Dies setzt voraus, daß das RTS am Schluß des Makros fortfällt, daß transportiert und ggfs. umgerechnet wird und daß ferner die absolute Adressierung des übrigen Objektprogramms auf die Länge des Segmentes Rücksicht nimmt.*

*Die Möglichkeiten b) und c) sind hier nicht ausgeführt, die nachfolgende typische Makro-Gliederung enthält jedoch in JSR TRANSP eine Andeutung, wie man sie ausfüllen könnte.*

TYPISCHE STRUKTUR EINES STELLUNGS-MAKROS

```
ASSEM   JSR ADRETA      ERHÖHE DIE RÜCKKEHRADRESSE NACH
                        DEM CALL UM DIE LÄNGE DER PARM-
                        LIST. BASISADRESSE DER PARMS-1
                        NACH D1/D2. LÄNGE DER PARMLIST
                        [OHNE LOP] NACH Y ALS ZÄHLER
                        UND ADDRESSER.

        ORG             GGFS. DAS EINSETZEN VON DEFAULT
                        VALUES UEBERSPRINGEN

DFAULT                  EINSETZEN VON DEFAULT VALUES
                        ENTWEDER NACH EINER TABELLE ODER
                        IM EINZELVERFAHREN

GEN     LDX TABLE-1,Y   X WIRD AUS DER TABELLE ALS
                        ADDRESSER GELADEN
        LDA [D1],Y      HOLEN DES PARAMETERS VOM CALLER
                        VIA POINTER
        STA EXCUTE,X    ABLEGEN DES PARAMETERS RELATIV
                        ZUM ANFANG DER EIGENTLICHEN
                        ROUTINE
        DEY             ZÄHLER-1
        BNE GEN         SCHLEIFE

        JSR TRANSP      OPTION FÜR DEN TRANSPORT DES
                        GENERIERTEN MAKRO
        .LOP            LÄNGENOPERATOR FÜR DAS SEGMENT
                        EXCUTE BIS RTS

EXCUTE  ...             AUSFÜHRUNGSTEIL

        RTS
        .LOP            LÄNGENANGABE FÜR NACHFOLGENDE
                        TABELLE, GGFS. MIT WORKA
TABLE   .BYTE           RELATIVE ADRESSEN, DISPLACEMENTS

        ...
WORKA   XX              GGFS. ARBEITSBEREICH

        NOP             RALOAD-BYTE FÜR VERSCHIEBUNG
```

*Man beachte bei diesem Design, daß mit X als Addresser nur bis zu 256 Bytes überstrichen werden können, LOPs schließen nur bis zu 62 Bytes einer Tabelle aus einer Umrechnung bei Verschiebung aus.*

WIRD FORTGESETZT - R.L.

# A S P - Advanced Subroutine Package (Teil 2)

Michael Zimmermann, Eberstädter Str. 170, 6102 Pfungstadt

E: In the second part of his series Mr. Zimmermann describes the set of his 'simple subroutines', rendering parms by VORB, MOVE, FILL, ADD (decimal or binary), SUB, COM (decimal or binary) and the logical AND, OR and EOR routines.

Inhaltsübersicht für den 2. Teil



* * *

## 1. Einfache Unterprogramme

Diese Gruppe von Unterprogrammen enthält vorwiegend solche, in denen 2 Operanden miteinander verbunden werden. Hiervon abweichende Strukturen werden im Zusammenhang mit dem jeweiligen Modul beschrieben.

Die Struktur des Ausrufes ist in symbolischer Form:

```
JSR UPRO
.BYTE   Länge -1, Längenoperand, maximal 127(10)
.WORD   Adresse des ersten Operanden, gleichzeitig Adresse des
        Ergebnisses
.WORD   Adresse des 2. Operanden, ebenso wie die Adresse des
        1.Operanden auf die niedrigste Adresse des Feldes im
        Speicher zeigend. Sie ist die Dezimalstelle mit der
        höchsten Wertigkeit.
```

Ebenso wie die Struktur bei der Obergabe der Operanden ist auch der Ablauf der Verarbeitung bei den meisten internen Unterprogrammen identisch:

- Obernahme der Parameter mit dem entsprechenden internen Unterprogramm, gleichzeitig Laden des Y-Registers mit der Länge der Operation.
- Setzen der Anfangsbedingungen, wie z.B. setzen oder löschen von Carry und/oder Decimal-Flag.
- Durchführen der Verknüpfung mit einem Byte, dieses indirekt indiziert adressiert, Speicherstellen E3/E4 und E5/E6, Verarbeitung der Bytes von denen mit hoher Adresse und niedriger Dezimale hin zu denen mit niedriger Adresse und hoher Dezimale.
- Vermindern des Y-Registers als Operationslängenzähler.

- Bei positivem Y-Register Springen zur Verarbeitung des nächsten Byte.
- Rücksprung in aufrufendes Programm.

Aus diesem Standardablauf ergeben sich die Werte, die nach Ausführung eines derartigen Moduls in den Registern enthalten sind:

A Ergebnis der Verknüpfung des letzten Bytes

Y immer $FF_{16}$.

X nicht verändert

ST Als Ergebnis der Abfrage auf Y-Register immer negativ.

## 1.0 Internes Unterprogramm zur Parameterübernahme (VORB)

Die meisten ASP-Prozeduren verwenden Parameter. Wo diese Parameter zu finden sind (hinter dem Programmaufruf) und wie sie beschaffen sind, wurde bereits in 0.4 und 0.5 beschrieben.

Die verarbeitenden ASP-Module erwarten diese Parameter in Page-Zero-Locations. Die Übernahme der Parameter wird durch ein Unterprogramm vorgenommen, das von jedem ASP-Modul zunächst angesprungen wird.

Um dies zu verdeutlichen, folgt ein Beispiel, bei dem das Hauptprogramm an der Adresse 0000 ruft. Das ASP-Modul stehe auf 0200 und das Übernahme-Unterprogramm auf 0300

```
0000  20 00 02    Aufruf des ASP-Moduls als Upro.  Stand des PC wird auf
                  dem Stack abgespeichert.
0003  01          Es folgt Parameterliste
0004  02 03
0006  04 05
0008  ...         Weitere Instruktionen

0200  20 00 03    ASP-Modul, zunächst Aufruf des Parameterübernehme-
                  programmes.
0203  ...         Weitere Instruktionen des ASP-Moduls

0300  ...         Erste Instruktion des Parameterübernahmeprogrammes.


01FF  00          Rücksprungadresse für Hauptprogramm
01FE  02
01FD  02          Rücksprungadresse in ASP-Modul
01FC  02
01FB  ..          Stelle, auf die der Stack-Pointer weist.
```

Erste Aktion des Parameterübernahmeprogrammes ist es, die Parameteranzahl zu setzen und den Stand des Stack-Pointers in das X-Register zu übernehmen. Sodann muß die Rücksprungadresse des Hauptprogrammes, hinter der die Parameter stehen, in Page-Zero-Locations E0 und E1 übernommen werden. Die Rücksprungadresse steht auf 0103 bzw. 0104+X.

Hernach werden die 5 Parameterbytes in die auf E1 folgenden Adressen übertragen, gleichzeitig wird die Rücksprungadresse für das Hauptprogramm erhöht. Abschließend wird der erste Parameter, meist die Länge, in das Y-Register übertragen. - Hiermit ist die Paramterübernahme abgeschlossen, die Kontrolle wird an die ASP-Prozedur zurückgegeben.

```
XX00  A0 05      VORB   LDY #$ 05      SET PARAMETER COUNT
  02  BA               TSX            TRANSFER SP TO X
  03  BD 03 01         LDA 103,X      TRANSFER RETURN ADL
  06  85 E0            STA E0         TO E0
  08  BD 04 01         LDA 104,X      AND FOR HI
  0B  85 E1            STA E1
  0D  B1 E0      V10   LDA $E0&,Y     TRANSFER PARAMETERS
  0F  99 E1 00         STA 00E1,Y
  12  FE 03 01         INC 0103,X     UPDATE RETURN ADDRESS
  15  D0 03            BNE            NO PAGE CROSSING ? SKIP
  17  FE 04 01         INC 0104,X     ADJUST HI ORDER
  1A  88               DEY            ONE LESS TO GO
  1B  D0 F0            BNE V10        GET NEXT IF NOT DONE
  1D  A4 E2            LDY 00E2       LOAD LENGTH
  1F  60               RTS            RETURN.
```

## 1.1 Datenübertragung (MOVE)

Dieses Unterprogramm dient der Übertragung von Daten jeder Art in einer Länge bis zu 127 Bytes.

Veränderungen an den Daten werden nicht vorgenommen. Aufruf und Ablauf des Unterprogrammes entsprechen dem Standardaufbau.

```
XX00  20 XX XX   MOVE   JSR VORB
  03  B1 E5       M10   LDA (E5),Y     LOAD FROM
  05  91 E3             STA (E3),Y     STORE TO
  07  88               DEY            COUNT -1
  08  10 F9            BPL M10        DONE ? NO,REPEAT
  0A  60               RTS            YES, RETURN
```

## 1.2 Speicher Füllen (FILL)

Mit diesem Unterprogramm ist es möglich, einen Speicherbereich, bis 64 k, fortlaufend mit einer 1-Byte-Konstanten zu füllen. Der Aufruf ist abweichend vom Standardaufruf:

```
JSR FILL
.BYTE    Wert der eingesetzt werden soll
.WORD    Adresse, von der ab der Wert eingesetzt werden soll
.WORD    Adresse bis zu der einschließlich der Wert eingesetzt
         werden soll.
```

Der erste Operand sollte kleiner als der zweite sein. Ist dies nicht der Fall, so wird nur eine einzige Adresse, die des 1. Operanden,gefüllt.

Auch die Struktur dieses Unterprogrammes weicht vom allgemeinen Aufbau ab. Einmal erfolgt die Adressierung indiziert indirekt mit einem gelöschten X-Register, zum anderen werden die Speicherwerte E3/E4 direkt erhöht und das Ende durch Vergleich dieser mit dem Inhalt von E5/E6 festgestellt.

Nach Durchführung des Unterprogrammes enthält:

```
A  den eingesetzten Wert,
Y  den eingesetzten Wert,
X  00
ST Carry immer gesetzt.
```

```
XX00  20 XX XX    FILL    JSR VORB
  03  A2 00              LDX #$ 00        SET X
  05  A5 E2        F10   LDA E2           LOAD CHARACTER
  07  91 E3              STA (E3),Y       STORE CHARACTER
  09  E6 E3              INC E3           INCREASE POINTER
  0B  D0 02              BNE +2           SKIP IF NO PAGE CROSSING
  0D  E6 E4              INC E4           ADJUST
  0F  A5 E3              LDA E3
  11  C5 E5              CMP E5           COMPARE FINAL ADDRESS LOW
  13  A5 E4              LDA E4
  15  E5 E6              SBC E6           AND HI
  17  90 EC              BCC F10          REPEAT IF NOT DONE
  19  60                 RTS              IF THRU
```

## 1.3 Dezimale und binäre Addition (ADD)

Diese beiden Module entsprechen in Aufruf, Struktur und Ablauf den Standards. - Dezimale und binäre Arithmetik wurden in ein Unterprogramm zusammengefaßt, um Redundanzen zu vermeiden.

Zu beachten ist, daß beide Operanden in gleicher Länge verarbeitet werden. Für eine evtl. erforderliche Verarbeitung von dezimalen Operanden mit unterschiedlicher Länge ist ein eigenes Unterprogramm vorgesehen.

Ein evtl. auftretender Überlauf steht dem Benutzer im Overflow-Flag zur Verfügung, ebenso ein Carry. Das Decimal-Flag wird in jedem Falle nach Bearbetung gelöscht, ein Benutzereingriff ist hier also nicht mehr erforderlich.

Für eine binäre Operation muß das höchstwertige Bit im Längenbyte des Aufrufes gesetzt sein.

```
XX00  20 XX XX    ADD     JSR VORB         TRANSFER PARAMETERS
  03  98                  TYA              TRANSFER LENGTH
  04  30 01               BMI * +1         CHECK FIRST BIT SET
  06  F8                  SED              NO, SET DECIMAL
  07  29 7F               AND #$ 7F        CLEAR FIRST BIT
  09  A8                  TAY              TRANSFER LENGTH BACK
  0A  18                  CLC              CLEAR CARRY
  0B  B1 E3         A10   LDA (E3),Y       LOAD FIRST OP
  0D  71 E5               ADC (E5),Y       ADD SECOND OP
  0F  91 E3               STA (E3),Y       STORE IN LOCATION OF 1RST OP
  11  88                  DEY              LENGTHCOUNT -1
  12  10 F7               BPL A10          REPEAT IF NOT ALL DONE
  14  D8                  CLD              CLEAR DECIMAL
  15  60                  RTS              RETURN
```

## 1.4 Dezimale und binäre Subtraktion (SUB)

Bis auf die Art der gesetzten Anfangsbedingungen und die durchgeführte Operation entspricht das Unterprogramm zur Subtraktion dem zur Addition; auf eine gesonderte Darstellung kann aus diesem Grunde verzichtet werden.

```
XX00  20 XX XX    SUB     JSR VORB        TRANSFER PARAMETERS
  03  98                  TYA             TRANSFER LENGTH
  04  30 01               BMI *+1         CHECK FIRST BIT SET
  06  F8                  SED             NO, SET TO DECIMAL
  07  29 7F               AND #$ 7F       CLEAR FIRST BIT
  09  A8                  TAY             TRANSFER LENGTH BACK
  0A  38                  SEC             SET CARRY
  0B  B1 E3         S10   LDA (E3),Y      LOAD FIRST OP
  0D  F1 E5               SBC (E5),Y      SUBTRACT SECOND OP
  0F  91 E3               STA (E3),Y      STORE TO PLACE OF 1RST OP
  11  88                  DEY             COUNT DONE ?
  12  10 F7               BPL S10         NO, REPEAT
  14  D8                  CLD             YES, CLEAR DECIMAL MODE
  15  60                  RTS             RETURN.
```

## 1.5 Dezimaler und binärer Vergleich (COM)

Der dezimale und der binäre Vergleich entsprechen den Unterprogrammen für Addition und noch mehr für Subtraktion. Ein Unterschied besteht nur darin, daß das Ergebnis nicht zurückgespeichert wird.

Ebenso wird bei der Vorbereitung die Speicherzelle E1 gelöscht und bei der Verarbeitung mit einem Ergebnis ungleich Null überschrieben. Da E1 nach der Verarbeitung in A geladen wird, ist es möglich, auch Ergebnisse auf Gleichheit zu überprüfen. Die Abfragen nach Vergleich lauten:

BNE wenn Operand 1 gleich Operand 2

BCC wenn Operand 1 grösser Operand 2

BCS wenn Operand 1 kleiner Operand 2.

```
XX00  20 XX XX    COM     JSR VORB        TRANSFER PARAMETERS
  03  98                  TYA             TRANSFER LENGTH
  04  30 01               BMI *+1         CHECK FIRST BIT SET
  06  F8                  SED             NO, SET DECIMAL
  07  29 7F               AND #$ 7F       CLEAR FIRST BIT
  09  A8                  TAY             TRANSFER LENGTH BACK
  0A  38                  SEC             SET CARRY
  0B  A9 00               LDA #$ 00       CLEAR ZERO FLAG
  0D  85 E1               STA E1
  0F  B1 E3         C10   LDA (E3),Y      LOAD FIRST OP
  11  F1 E5               SBC (E5),Y      SUBTRACT 2ND OP
  13  F0 02               BEQ *+2         IF ZERO
  15  85 E1               STA E1          RESET ZERO FLAG
  17  88                  DEY             COUNT DONE ?
  18  10 F5               BPL C10         NO, REPEAT
  1A  D8                  CLD             YES, CLEAR DECIMAL
  1B  A5 E1               LDA E1          LOAD ZERO FLAG
  1D  60                  RTS             RETURN
```

## 1.6 Logisches UND (AND)

Mit diesem Unterprogramm wird bitweise ein logisches UND zwischen dem 1. und dem 2. Operanden gebildet, wobei das Ergebnis den ersten Operanden überschreibt. Aufruf, Struktur und Ablauf entsprechen dem Standard, ein Setzen von Anfangsbedingungen ist nicht erforderlich.

```
XX00  20 XX XX    AND     JSR VORB        TRANSFER PARAMETERS
  03  B1 E3        A10    LDA (E3),Y      LOAD FIRST OP
  05  31 E5               AND (E5),Y      AND WITH 2ND OP
  07  91 E3               STA (E3),Y      STORE TO PLACE OF 1RST OP
  09  88                  DEY             COUNT DONE ?
  0A  10 F7               BPL A10         NO, REPEAT
  0C  60                  RTS             YES, RETURN
```

## 1.7 Logisches ODER (OR)

Dieses Unterprogramm entspricht mit Ausnahme der Verknüpfung vollständig dem logischen UND. Auf eine besondere Beschreibung wird deswegen verzichtet.

```
XX00  20 XX XX    OR      JSR VORB
  03  B1 E3        010    LDA (E3),Y      LOAD FIRST OP
  05  11 E5               ORA (E5),Y      OR WITH SECOND OP
  07  91 E3               STA (E3),Y      STORE TO PLACE OF FIRST OP
  09  88                  DEY             COUNT DONE ?
  0A  10 F7               BPL 01Ø         NO, REPEAT
  0C  60                  RTS             YES, RETURN
```

## 1.8 Exklusives ODER (EOR)

Dieses Unterprogramm entspricht ebenfalls dem logischen UND, mit Ausnahme der Verknüpfung (s.o.).

```
XX00  20 XX XX    EOR     JSR VORB        TRANSFER PARAMETERS
  03  B1 E3        E10    LDA (E3),Y      LOAD FIRST OP
  05  51 E5               EOR (E5),Y      EXOR WITH 2ND OP
  07  91 E3               STA (E3),Y      STORE BACK
  09  88                  DEY             COUNT DONE ?
  0A  10 F7               BPL E10         NO, REPEAT
  0C  60                  RTS             YES, RETURN
```

*Wird fortgesetzt.*

**AIM-65**

Neuer Mikrocomputer v. Rockwell mit Thermodrucker ( alphanumerisch, 20 Zeichen/Zeile) , 20 stellige Anzeige ( 16 Segment) und vollständige Tastatur ( 69 Zeichen). Leistungsfähiges 4 K-ROM Betriebssystem. Reservesockel zur Erweiterung des Festwertspeichers bis zu 16 K. RAM-Erweiterung auf der Platine bis zu 4 K-Byte. Interface f. TTY und 2 Kassettenrecordern etc. Steckeranschlüsse KIM- 1 kompatibel.

| | | |
|---|---|---|
| AIM - 65 | mit 1 K RAM | 875,- DM ( 980,-) |
| AIM - 65 | mit 4 K RAM | 1 050,- DM (1176,-) |
| Assembler/Editor | 4 K ROM | 230,- DM ( 257,6) |
| BASIC-Interpreter | 8 K ROM | 280,- DM ( 313,6) |

**KIM-1**

| | | |
|---|---|---|
| Kompl. mit vollst. Dokumentation | 590,--DM | (660,80) |
| KIM-3 8 KByte RAM-Platine | 710,--DM | (795,20) |
| KIM-4 Mutterplatine | 329,--DM | (368,68) |
| KIMath Subroutines | 14,50DM | |
| "First Book of KIM-1" | 19,80DM | |
| "Hobby-Computer-Handbuch" 450 S. | 29,80DM | |
| Z 80-KIT. Bausatz | 770,--DM | (862,40) |
| Z 80-KIT-Videointerface Bausatz | 530,--DM | (593,60) |

(Preise in Klammern = incl. Mwst)
Verlangen Sie ausführliche Information und die jeweils aktuelle Preisliste über

*Johann Fippinger Vertrieb von Mikrocomputersystemen*

Postfach 26 01 46
Senftenberger Ring 42 d
1000 Berlin 26

# ZAHLEN-WANDLUNG (TEIL 2)

Dr. Günther Rüdiger, Ostlandring 12, 2057 Reinbek

E: In his second article Dr. Rüdiger presents two subroutines that will convert numbers from their notation in other numerative systems (based to 2, 4, 6, 8, 10, 12 or 14) to their hexadecimal equivalent and vice versa. The STODEF routine fixes for the extent of storage required for above.

*Zahlenwandlungsroutinen werden beim praktischen Einsatz immer Bestandteil eines übergeordneten Programmes sein, dem sie sich nach den speziellen Anforderungen bezüglich Stellenzahl und Umwandlungsmodus flexibel anzupassen haben. Außerdem müssen sie voll verschieblich sein.*

*Im Folgenden werden zwei sehr variable Wandlungsroutinen in Maschinencode und in der Assembler-Umsetzung für die 65xx-Mikros angegeben, Routinen, mit denen sich beliebig-stellige Hexadezimalzahlen in solche zur Basis 2, 4, 6, 8, 10, 12 oder 14 und umgekehrt umwandeln lassen.*

*Bei der Verwendung der Programme müssen zunächst die benötigten Stellenzahlen für die beiden Register festgelegt werden, woraus sich dann auch die Länge der Register ergibt. Es sei 'n' die Stellenzahl für die hexadezimale Zahl, 'm' sei die benötigte Stellenzahl zur Basis B = 2, 4, 6, 8, 10, 12 oder 14. Je nach Umwandlungsrichtung ist entweder n oder m vorgegeben. Die jeweils fehlende Größe ergibt sich aus der Beziehung:*

$$16^n = B^m; \qquad N \times LOG16 = M \times LOGB$$

*Als Logarithmusfunktion können sowohl der dekadische als auch der natürliche Logarithmus - die beide auf wissenschaftlichen Taschenrechnern zur Verfügung stehen - benutzt werden. Die errechnete Größe muß auf die nächsthöhere geradzahlige Ganzzahl aufgerundet werden, weil die Stellen in der gepackten Form vorliegen, also 1 Byte immer 2 Stellen enthält.*

*Beispiel: Es soll eine Serie von Hexadezimalzahlen mit maximal 9 Stellen (also N = 10, erhöht) in solche zur Basis 6 umgewandelt werden. Wie groß ist m?*

*10×log16 = m×log6*
*10×log 1,204 = m×0,778*
*m = 15,47, aufgerundet m = 16*

*Das Hexadezimalregister belegt also 5 Byte, das Register für die Zahl zur Basis 6 also 8 Byte.*

*Die Register werden bei den vorliegenden Routinen ab Speicherplatz 60 in der 'Zeropage' hintereinander angelegt. Die genaue Struktur wurde bereits im 1. Teil dieser Arbeit beschrieben. Demnach belegt die Hexazahl (B=6) im Beispiel die Plätze 60-67, die Hexadezimalzahl die Plätze 68-6C. Input und Outputroutinen können darauf eingerichtet werden.*

*Eine Verlegung der Register, auch auf Speicherbereiche außerhalb der Zeropage, ist ohne weiteres möglich. Dazu müssen lediglich die Festadressen auf den Speicherplätzen* 0306, 031C, 0324, 0336, 0338, 0342, 0356 *(Inhalt jeweils* $5F*) und* 0338 *(Inhalt:* $5E*) entsprechend geändert werden, außerdem die*

*indirekten Sprungweiten bei absoluter Adressierung.*

*Defintion der Programmkonstanten:*

*Aus den fixierten Werten für B, n und m müssen nun insgesamt 7 Programmkonstanten berechnet werden, was entweder manuell oder bequemer durch das Hilfsprogramm 'STODEF' (s.u.) geschehen kann. Die 7 Konstanten belegen die Speicherplätze 50 bis 56 in der Zeropage. Die Berechnung der drei sogenannten Verschiebungskonstanten findet sich schematisiert in Tabelle 1. Die basisabhängigen Größen lassen sich direkt aus Tabelle 2 ablesen.*

TABELLE 1: VERSCHIEBUNGSKONSTANTEN

| Z.P. | NAME | FORMEL | BEISPIEL (S.O.) FÜR B=6, N=10, M=16 | |
|---|---|---|---|---|
| 0050 | BITHEX | 4×N | 28 | |
| 0051 | LEN-N | M /2 | 08 | HEXADEZIMALE ZAHL |
| 0052 | LENREG | (N+M) /2 | 0D | |

TABELLE 2:BASISABHÄNGIGE GRÖSSEN

B = BASIS; N = STELLENZAHL HEXADEZIMAL
M = STELLENZAHL ZUR BASIS B

| Z.P. | NAME | FORMEL | SPEICHERINHALTE BEI B = 2 | 4 | 6 | 8 | A | C | E |
|---|---|---|---|---|---|---|---|---|---|
| 0053 | VALLSB | B / 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0054 | VALMSB | 8B | 10 | 20 | 30 | 40 | 50 | 60 | 70 |
| 0055 | ADDLSB | 8-B / 2 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| 0056 | ADDMSB | 80-8B | 70 | 60 | 50 | 40 | 30 | 20 | 10 |

*Bei Einsatz des Hilfsprogrammes 'STODEF' ist man von der Verwendung der beiden Tabellen unabhängig. Es sind dann lediglich 3 Speicherplätze vorher folgendermaßen zu belegen:*

*'m' nach 0051*
*'n' nach 0052*
*'B' nach 0053*

*alle drei Konstanten als hexadezimale Zahlen!*

*Abschließend sei hinzugefügt, daß bei Applikationen mit nichtveränderlichen Werten für n, m und B die 7 Programmkonstanten auch direkt in die Routinen eingearbeitet werden können, indem man an den entsprechenden Stellen die adressierten 'LOAD'- bzw. 'ADC'- und 'SBC'-Befehle durch IMMEDIATE-Befehle ersetzt werden.*

```
0300 D8        HEX-N  CLD            SUBROUTINE !
0301 A4 50            LDY BITHEX
0303 A6 51     LOOP7  LDX LEN-N
```

```
0305 B5 5F      LOOP8   LDA 5F,X        BYTE
0307 F0 12              BEQ ZERO
0309 48                 PHA             SAVE ON STACK
030A 29 0F              AND #$ OF       ISOLATE MSB
030C C5 53              CMP VALLSB
030E 68                 PLA             RESTORE
030F 90 03              BCC COMP1       SKIP
0311 18                 CLC
0312 65 55              ADC ADDLSB
0314 C5 54      COMP1   CMP VALMSB
0316 90 03              BCC COMP2
0318 18                 CLC
0319 65 56              ADC ADDMSB
031B 95 5F      COMP2   STA 5F,X        BYTE
031D CA         ZERO    DEX             FOR NEXT BYTE
031E D0 E5              BNE LOOP8
0320 18                 CLC
0321 A6 52              LDX LENREG
0323 36 5F      LOOP9   ROL 5F,X        REGISTER
0325 CA                 DEX
0326 D0 FB              BNE LOOP9       ROLLING DONE ?
0328 88                 DEY
0329 D0 D8              BNE LOOP7
032B 60                 RTS

0330 D8         N-HEX   CLD             SUBROUTINE !
0331 A4 50              LDY BITHEX
0333 A6 52      LOOP12  LDX LENREG
0335 06 5F              ASL AUX
0337 B5 5E      LOOP10  LDA 5E,X        PREBYT
0339 4A                 LSR
033A 76 5F              ROR 5F,X        REG
033C CA                 DEX
033D D0 F8              BNE LOOP10
033F A6 51              LDX LEN-N
0341 B5 5F      LOOP11  LDA 5F,X        BYTE
0343 F0 12              BEQ ZERO
0345 48                 PHA
0346 29 0F              AND #$ 0F
0348 C5 53              CMP VALLSB
034A 68                 PLA
034B 90 02              BCC COMP1
034D E5 55              SBC ADDLSB
034F C5 54      COMP1   CMP VALMSB
0351 90 02              BCC COMP2
0353 E5 56              SBC ADDMSB
0355 95 5F      COMP2   STA 5F,X        BYTE
0357 CA         ZERO    DEX
0358 D0 E7              BNE LOOP11
035A 88                 DEY
035B D0 D6              BNE LOOP12
035D 60         RTS

035F D8         STODEF  CLD             SUBROUTINE !
0360 A5 53              LDA BASIS
0362 0A 0A 0A           ASL,ASL,ASL     MULTIPLY BY 8
0365 85 54              STA VALMSB
```

```
0367 A9 80            LDA #$ 80
0369 38               SEC
036A E5 54            SBC VALMSB    80-8B
036C 85 56            STA ADDMSB
036E 4A 4A            LSR,LSR       DIVIDE BY 10 (HEX-#)
0370 4A 4A            LSR,LSR
0372 85 55            STA ADDLSB
0374 46 53            LSR VALLSB    B -> B/2
0376 A5 52            LDA LENREG    HEX LENGTH x 4 ...
0378 0A 0A            ASL,ASL
037A 85 50      SIZE  STA BITHEX    ... TO BITHEX
037C 46 51            LSR LEN-N     M -> M/2
037E A5 52            LDA LENREG    N -> N/2
0380 4A               LSR
0381 18               CLC
0382 65 51            ADC LEN-N     ADD M/2
03A4 85 52            STA LENREG
0386 AA         CLEAR TAX           SAVE IN X
0387 A9 00            LDA #$ 00     TO SET WHOLE REGISTER TO ØØ
0389 95 5F      CLEAR1 STA 5F,X
038B CA               DEX
038C D0 FB            BNE CLEAR1
038E 60               RTS
```

**********

## Benutzung der Timer und Interrupt

E: *The KIM timers (6530) are illuminated. How to influence speed, get a readout and how to use them with or without the interrupt option.*

*Nach Beobachtungen des Herausgebers besteht hinsichtlich der Benutzung der KIM-Timer auf den 6530-Chips noch einige Unsicherheit. Ein Grund dafür mag in der verkürzten Darstellung im deutschen Benutzerhandbuch liegen. Es folgt daher eine kurze Übersicht mit Beispielen.*

*Erster Bestandteil eines Timers ist der Frequenzteiler. Er teilt den 1 MHz-Maschinentakt (Ø2) durch 1, 8, 64 oder 1024. Welches dieser Teilverhältnisse zur Anwendung kommt, hängt von den beiden niedersten Bits auf dem Adreßbus (A0 und A1) bei der Adressierung des Timers in einem WRITE-Befehl ab.*

*Zweiter Bestandteil der Timer ist ein programmierbares 8-Bit-Register. Ein beliebig aus dem Akku oder aus den Registern X und Y zu entnehmendes Byte wird per STA, STX oder STY in dieses Zählregister geschrieben. Die Adressierung in A0 und A1 legt zugleich den Zähltakt, das Zeitverhältnis fest (s.o.). Ab Zeitpunkt des STA (STX, STY) wird der Inhalt des Zählregisters mit jedem aus dem Frequenzteiler kommenden Impuls um 1 heruntergezählt. Der anfangs in das Zählregister geladene Wert stellt also eine Vorgabe dar.*

*Beim Herunterzählen wird einmal der Wert 00 erreicht sein. Beim nächsten vom Frequenzteiler ankommenden Impuls klappt der Zähler auf 'FF' um. Damit treten weitere Veränderungen ein (s.u.).*

*Dritter Bestandteil der Timer ist eine Interrupt Control. Sie wird durch das Bit A3 beim Schreiben in den Timer beeinflußt. Ist A3=0, so hebt es die Interruptmöglichkeit auf. Mit A3=1 setzt man 'Enable Interrupt'.*

*Aber: Der Interrupt kann sich nur bemerkbar machen, wenn er über eine besondere Leitung des Peripheriebausteines entweder an den Eingang IRQ oder an NMI herangeführt wird. Diese 'besondere Leitung' ist Pin PB7 auf den 6530-Bausteinen. Beim 6530-002 ist Sie für das Cassetteninterface bereits fest belegt. Es bleibt der für den Anwender freie Baustein 6530-003 mit seinem Timer. Man führt also eine Verbindungsleitung von seinem PB7 (Applikationsstecker, Pin 15) nach IRQ oder NMI (Expansionsstecker, Pin 4 oder Pin 6) und man setzt PB7 im Datenrichtungsregister PBDD (Adr. 1703) als Eingangsleitung (das ist auch der normale Status nach RESET).*

*Also: Bit A3=1 schaltet PB7 als Signalleitung für Interrupt (aktiv low), wenn PB7 zuvor als Eingang programmiert wurde.*

*Zurück zum Zählregister: Sobald es auf FF kippt, treten folgende Veränderungen ein:*

*a) Eine Adresse im Timerbereich dient als Statusregister (ihr Bitmuster in der niedrigen Adresse: xxxx 0x11, z.B. 1707, 1747). Sein vorderstes Bit (MSB) wird von vorher 0 auf jetzt 1 gesetzt. Dieser Status kann vor und nach Ablaufen des Timers jederzeit mit der BIT-Operation abgefragt werden.*

*b) Wenn PB7 - wie beschrieben - als Interruptleitung geschaltet wurde, geht sein Signal auf '0'; Programmzähler und Status des Prozessors werden auf dem Stack gerettet, das Programm wird zu der Interruptroutine verzweigt. Deren Adressvektor mußte natürlich vorher nach 17FA/FB bzw. nach 17FE/FF gebracht worden sein.*

*c) Das Zählregister wird weiterhin heruntergezählt, jedoch nicht mehr durch das ursprünglich bestimmte Teilverhältnis verlangsamt, sondern unmittelbar mit dem 1 MHz-Takt des Ø2. Das Zählregister kann sogar wiederholt durch 00-FF laufen. Es zeigt jetzt also (als negative Zahl, in komplementärer Form) die Zeit seit Nulldurchgang an, gezählt in Maschinentakten.*

*In vielen Fällen möchte man nun sicher vor oder nach dem Nulldurchgang den Stand des Zählregisters prüfen. - Innerhalb des Timerbereiches ist dafür die Leseadresse (low) durch das Bitmuster xxx x110 definiert, also z.B. durch 1706 oder 1746. Besonderheit: a) Das Lesen dieser Adresse schaltet weitere Interruptmöglichkeit durch den Timer ab, auch wenn er zuvor mit Interruptmöglichkeit gestartet wurde. b) Vom Zählerstand, wie er zum Zeitpunkt des Lesens vorgefunden wurde (also nicht vom zuerst eingeschriebenen) wird wieder mit dem alten Frequenzteilerverhältnis heruntergetaktet. Das Statusbit wird auf 0 zurückgesetzt, PB7 wird inaktiviert.*

*Liest man dagegen die Adresse (low) mit Bitmuster xxxx 1110, Z.b. 170E, so wird weiterer Interrupt ermöglicht (auch schon während einer noch nicht abgeschlossenen Interruptberabeitung!), das Weiterzählen geschieht wie bei b).*

*Es bleibt nachzutragen: Das oben im Absatz a) erwähnte Statusbit kann dem sog. Interrupt-Polling dienen, wenn es konkurrierende Interruptmöglichkeiten im System gibt.*

*Pin PB7 sollte keinen pull-up-Widerstand haben, wenn es direkt mit NMI oder IRQ verbunden ist. PB7 ist so ausgelegt, daß er mit anderen Interruptquellen logisch ODER zusammengeschaltet werden kann (wired OR). Im englischen Manual steht dann noch beziehungslos: Wenn das nicht gewünscht wird, sollte man einen Widerstand von 5,1 k nach + 5 Volt legen. Offensichtlich also, wenn man die logischen ODER auf einen TTL-Baustein legt.*

*Will man PB7 als normale I/O-Verbindung fahren, so soll man durch Schreiben oder Lesen in xxxx 0110 (z.B. 1706) die Interruptmöglichkeit ausschalten. Diesem Ausschalten sollte man nach jedem Interruptbetrieb besondere Aufmerksamkeit widmen!*

*Die Adressen der beiden KIM-Timer:*

| | | | |
|---|---|---|---|
| 1744 | Frequenzteilung: 1 | ohne Interrupt | CLK1T |
| 1745 | " : 8 | " " | CLK8T |
| 1746 | " : 64 | " " | CLK64T |
| 1747 | " : 1024 | " " | CLKKT |
| 1747 | Statusregister für BIT-Test | | CLKRDI |
| 1746 | Adresse zum Registerlesen | | CLKRDT |

*Die entsprechenden Adressen 174C, 174D, 174E und 174F können nicht für Interrupt gebraucht werden, weil PB7 fest an das Cassetteninterface angeschlossen ist.*

*Und nun die Timermöglichkeiten des 6530-003:*

| | | | | | |
|---|---|---|---|---|---|
| 1704 | Frequenzteilung: 1 | ohne ... | 170C | dito mit | Interruptmöglichkeit |
| 1705 | " : 8 | " | 170D | " " | " |
| 1706 | " : 64 | " | 170E | " " | " |
| 1707 | " : 1024 | " | 170F | " " | " |
| 1707 | Statusregister | " | 170F | " " | " |
| 1706 | Leseadresse | " | 170E | " " | " |

*Und nun einige schematische Beispiele für den Einsatz der Timer:*

```
A9 20       TIMR1   LDA # $ 20
8D 44 17            STA CLK1T       DIVIDE ϕ2 BY 1, SCHNELLSTER TAKT
AD 46 17            LDA CLKRDT      REGISTER-LESEADRESSE
85 FA               STA POINTL      BRINGE ZUR ANZEIGE
A9 00               LDA #$ 00
85 FB               STA POINTH      00 IN DIE VORDERE ANZEIGE
4C 4F 1C            JMP KIM
```

*Vorstehendes kleines Programm arbeitet ohne Interrupt. Sofort nach dem Einschreiben in den Timer wird sein Inhalt wieder ausgelesen. Weil aber der Lesebefehl 3 Maschinentakte zur Ausführung braucht und weil ab Schreiben sofort heruntergezählt wird, findet man beim Lesen einen um 4 verminderten Wert, angezeigt wird also 001C auf den LEDs. Nimmt man in Programmzeile 2 ein höheres Teilverhältnis durch Adressierung der 1-3 folgenden Stellen, so gelangt 001F zur Anzeige: Der Timer hat sofort um 1 heruntergezählt. Die nächste Verminderung wäre nach 8/64/1024 Maschinentakten eingetreten, aber vorher schon wurde nach 3 Maschinentakten der Lesebefehl AD 46 17 ausgeführt. Legt man zwischen Einschreiben in den Timer und Herauslesen mindestens 5 weitere Maschinentakte durch irgendwelche Folgebefehle, so wird man 001E auf der Anzeige finden, wenn man 8D 45 17 adressiert hatte.*

```
A9 20       TIMR2   LDA #$ 20       BELIEBIGER ANFANGSWERT
8D 44 17            STA CLK1T       SCHNELLSTE GANGART DES TIMERS
2C 47 17    LOOP    BIT CLKRDI      IST VORDERSTES BIT GESETZT ?
10 FB               BPL LOOP        WENN NICHT: WEITER WARTEN
AD 47 17            LDA CLKRDI      BRINGE DEN STATUS ZUR ANZEIGE
85 FA               STA POINTL
AD 46 17            LDA CLKRDT      BRINGE REGISTERSTAND IN DIE
85 FB               STA POINTH      VORDERE ANZEIGE
4C 4F 1C            JMP KIM
```

*Nach Ausführung dieser Sequenz finden wir EC 80 in der Anzeige. EC als Ausdruck für die Zahl der Maschinentakte seit Umklappen des Registers bis zum Lesen von CLKRDT, die 80 zeigt an, daß das vorderste Bit im Statusregister auf 1 gesetzt wurde.*

*Wenn wir im nächsten Testprogramm zuerst den Zählerstand in CLKRDT abfragen und erst danach den Inhalt des Statusregisters konservieren, so finden wir die Anzeige F3 00, F3 für die vergangenen Maschinentakte und 00 für die Tatsache, daß das Lesen von CLKRDT das bei TIMR2 gesetzte Statusbit auf 0 zurückgesetzt hat. Es kommt also sehr auf die Reihenfolge der Abfrage an! Natürlich können wir alle diese Tests auch mit dem Adressenbereich des zweiten Timers fahren.*

```
A9 20       TIMR3   LDA #$ 20
8D 44 17            STA CLK1T
2C 47 17    LOOP    BIT CLKRDI
10 FB               BPL LOOP
AD 46 17            LDA CLKRDT      ZUERST DER REGISTERINHALT,
85 FB               STA POINTH
AD 47 17            LDA CLKRDI      DANN DEN STATUS GELESEN ...
85 FA               STA POINTL
4C 4F 1C            JMP KIM         ... SETZT DAS STATUSBIT ZURÜCK.
```

*Das folgende Beispiel TIMR4 nutzt die Interruptmöglichkeit des 6530-003, sofern die o.a. äußere Verdrahtung vorgenommen wurde. Das Vordergrundprogramm VORDER besteht nur aus einer Warteschleife. Im Anwendungsfall kann*

*hier natürlich jede andere ausgewachsene Routine betrieben werden. Das im Hintergrund 'lauernde' Interruptprogramm erhöht bei jedem Aufruf den Inhalt der Pointerzellen um 1. Auf diese Weise wird der Inhalt des Memory ab Startadresse Zelle für Zelle angezeigt, und zwar mit einer zeitlichen Verzögerung, die durch das Herunterzählen der Zelle 00 EE bewirkt ist.*

```
      A9 00       TIMR4   LDA #$ 00      VORDERGRUNDPROGRAMM
      8D 03 17            STA PBDD        SET PB7 TO INPUT
      A9 FF               LDA #$ FF
      8D 0F 17            STA 170F        WRITE TO TIMER 1024T WITH IN-
      A9 00       WAIT    LDA #$ 00       TERRUPT OPTION
      F0 FE               BEQ WAIT        BRANCH

0300  20 63 1F    INTER   JSR INCPT       INTERRUPTPROGRAMM:ERHÖHE POINTER
      20 19 1F    SHOW    JSR SCAND       DISPLAY
      C6 EE               DEC 00EE        HAVE A DELAY
      D0 F9               BNE SHOW        REPEAT SHOWING
      40                  RTI             RETURN FROM INTERRUPT
```

*Voraussetzung für das Laufen dieses Programms ist natürlich, daß der Adreßvektor der Routine INTER, in diesem Falle 00 03 nach IRQL/IRQH, nach 17FE/FF also, geladen wurde.* R.L.

**********

## MEET THE EDITOR - München, Sonnabend, den 11. November 1978

In Heft 2 wurde auf S. 35 bereits die parallel zur ELECTRONICA vom 9.-15. Nov. 1978 laufende IPE 78, Industrie und Personal Computer EXPO 78 angekündigt. Sie findet im Pschorr-Festsaal neben dem Haupteingang der ELECTRONICA statt.

Der Herausgeber steht dort für ein persönliches Gespräch am Sonnabend, den 11.11.78, von 11-16 Uhr zur Verfügung, und zwar auf dem Stand des KILOBAUD Fachzeitschriftenvertriebes Nedela.

## RAM TEST WITH RANDOM PATTERNS

E: Writing and checking regular patterns for a memory test will nor always unveil glitching changes in neighbouring cells. Random patterns are applied instead.

Ein Schreib-/Lesespeicher (RAM) mit schadhaften Bausteinen oder mit Kriechströmen auf Leiterbahnen ist ein besonderes Ärgernis, denn diese Fehler werden nicht unmittelbar sichtbar sondern nur indirekt. So mag man sich wundern, daß ein korrekt ausgetestetes Programm plötzlich 'verrückt' spielt. Man vermutet einen eigenen Bedinungsfehler oder eine sonstige Störung und lädt daher das Programm neu. Wieder verläuft es sich. Man vergleicht es danach also mühsam mit dem listing, stellt eine kleine Abweichung fest, berichtigt diese, startet neu - und hat die Störung noch immer nicht beseitigt.

Nun ist es höchste Zeit, dem RAM-Speicher zu mißtrauen. Wer das 'First Book of KIM' besitzt, wendet also Jim Butterfields 'Memory Test' zur Selbstdiagnose des Computers an. Und wenn sich dann - wie beim Herausgeber - kein Befund ergibt?

Durch Zufall wurde ich darauf aufmerksam, daß sich im 4k x8-RAM ein einzelnes Bit gelegentlich 'hinzuschiebt'. Statt eines geschriebenen 'A9' wird z.B. ein 'AB' gelesen. Schreibt man in eine solche Speicherzelle ein 'FF' oder ein '00', so entsteht keine Abweichung beim Lesen.

Der beschriebene Effekt mag nur unter ganz bestimmten Bitmustern im Speicher eintreten. Offensichtlich beeinflussen sich dann benachbarte Transistoren in den Bausteinen. Eine heimtückische Angelegenheit.

Warum fällt das nun im erwähnten Programm 'Memory Test' nicht auf? Es verwendet abwechselnd nur die Bitmuster 'FF' und '00', die es bei den höheren Durchgängen in jede dritte Stelle schreibt. Beim Vergleich werden nur diese beschriebenen Zellen wieder geprüft, Auswirkungen auf Nachbarzellen bleiben unberücksichtigt.

Nach diesen Erfahrungen lag es also nahe, auch andere Bitmuster zu schreiben und jeweils den gesamten Speicher auf mögliche Veränderungen abzufragen. Um nun jeder möglichen Willkürlichkeit vorzubeugen, wird die bekannte RAND-Routine (Butterfield) für die Gewinnung von Zufallszahlen eingesetzt. Eine Arbeits- und Vergleichstabelle wird dem Speicher aufgeprägt. Nach jedem Durchgang mit Invertierung rückt eine neue Zufallszahl in die um einen Platz nach rechts verschobene Tabelle nach. Die Tabelle ist mit der 'krummen' Zahl von 7 Plätzen angelegt, um architektonischen Regelmässigkeiten vorzubeugen.

Die Testbedingungen im Programmabschnitt RCHECK sind daher betriebsnah und wegen der Anwendung der Zufallszahlen mathematisiert. Dieser letzte Programmteil läuft in Endlosschleife, damit auch Effekte beim Warmwerden des RAM erfaßt werden können. Zu Beginn finden die regulären Muster im Abschnitt FCHECK Anwendung.

Eine Abweichung vom Sollwert stoppt den Programmablauf, die Adresse der irregulären Zelle erscheint auf dem Display, zusammen mit diesem Sollwert. Beim Betätigen irgendeiner Taste erscheint unter der angezeigten Adresse dann der Istwert. Dieses Programm entdeckt damit irreguläre Zellen, die bei Anwendung regulärer Muster verborgen bleiben. In manchen Fällen wird man auch schlechte Leiterbahnen auf der Platine einkreisen können.

Die sicher auch durch Alterungseffekte bedingten unauffälligen Speicherfehler legen es nahe, Bausteine - wenn möglich - nur auf Sockeln zu montieren.

Zur Programmtechnik: Die Bildung des Unterprogrammes MAC, das selber wieder eine Reihe von Unterprogrammen aufruft, verkürzt nicht nur die Programmniederschrift, sondern führt auch zu mehr Übersichtlichkeit.

```
                        RAM TEST WITH RANDOM PATTERNS

0000  XX          BEGIN   .BYTE           # OF FIRST PAGE TO BE CHECKED
0001  XX          END     .BYTE           # OF LAST. KEY-IN VALUES

0002  A5 00       MAIN    LDA BEGIN       INITIALIZE
0004  85 F9               STA INH
0006  A9 21               LDA #$ 21       FOR 33 OR ELSE PASSES
0008  85 9C               STA COUNT
000A  A0 00               LDY #$ 00
000C  84 F8               STY INL         ADDRESSER
000E  84 9D               STY TAB         FIRST PATTERN

0010  20 21 00    FCHECK  JSR MAC
0013  C6 9C               DEC COUNT
0015  D0 F9               BNE FCHECK      ALL PASSES ?

0017  20 73 00    RCHECK  JSR RAND        FORM A RANDOM-#
001A  A8                  TAY             SAVE IN Y
001B  20 21 00            JSR MAC
001E  4C 17 00            JMP RCHECK      ENDLESS LOOP, END OF MAIN

                        SUBROUTINES DIVISION

0021  20 90 00    MAC     JSR SHIFT
0024  20 2E 00            JSR WRICOM
0027  20 86 00            JSR INVERT
002A  20 2E 00            JSR WRICOM
002D  60                  RTS

002E  20 CC 1F    WRICOM  JSR OPEN        INL/INH TO POINTL/POINTH
0031  A0 00               LDY #$00        SET UP AS ADDRESSER
0033  A2 06               LDX #$ 06       SET UP AS COUNTER FOR 7
0035  B5 9D       WRITE   LDA TAB,X       WRITE TABLE
0037  91 FA               STA (POINTL),Y
0039  CA                  DEX             COUNTER -1
003A  10 02               BPL INCY        SKIP IF NOT  THRU
003C  A2 06               LDX #$ 06       RENEW COUNTER
003E  C8          INCY    INY             ADDRESSER+1
003F  D0 F4               BNE WRITE       IF NO PAGE CROSSING
0041  E6 FB               INC POINTH      FOR NEXT PAGE
0043  A5 01               LDA END         COMPARE FOR LAST PAGE-#
0045  C5 FB               CMP POINTH
0047  B0 EC               BCS WRITE       CONTINUE IF NOT

0049  20 CC 1F            JSR OPEN        RESET TO FIRST PAGE
004C  A2 06               LDX #$ 06       AS ABOVE
004E  B5 9D       COMPAR  LDA TAB,X
0050  D1 FA               CMP (POINTL),Y  NOW COMPARE TO TABLE
0052  F0 0E               BEQ GOON        SKIP IF OK
```

```
0054  85 F9       EXIT   STA INH          SAVE CORRECT CHARACTER
0056  84 FA              STY POINTL        SAVE CURRENT ADDRESSER
0058  68 68              PLA, PLA          ADJUST STACK
005A  20 1F 1F    SHOW   JSR SCANDS       AND SHOW
005D  F0 FB              BEQ SHOW          NO KEY DEPRESSED ?
005F  4C 4F 1C           JMP KIM           RETURN TO MONITOR

0062  CA          GOON   DEX              SAME SEQUENCE AS ABOVE
0063  10 02              BPL
0065  A2 06              LDX
0067  C8                 INY
0068  D0 E4              BNE COMPAR
006A  E6 FB              INC POINTH
006C  A5 01              LDA END
006E  C5 FB              CMP POINTH
0070  B0 DC              BCS COMPAR
0072  60                 RTS

0073  38          RAND   SEC              JIM BUTTERFIELD'S RAND ROUTINE
0074  A5 A5              LDA 00A5
0076  65 A8              ADC 00A8
0078  65 A9              ADC 00A9
007A  85 A4              STA 00A4
007C  A2 04              LDX #$ 04
007E  B5 A4       MOVE   LDA 00A4,X
0080  95 A5              STA 00A5,X
0082  CA                 DEX
0083  10 F9              BPL MOVE
0085  60                 RTS

0086  A2 06       INVERT LDX #$ 06        COUNTER FOR 7
0088  A9 FF              LDA #$ FF
008A  55 9D       VERT   EOR TAB,X        INVERT TABLE CONTENTS
008C  CA                 DEX
008D  10 FB              BPL VERT
008F  60                 RTS

0090  A2 05       SHIFT  LDX #$ 05        COUNTER FOR 6
0092  B5 9D       SHIFTI LDA TAB,X
0094  95 9E              STA TAB+1,X       DEPOSIT 1 PLACE HIGHER
0096  CA                 DEX
0097  10 F9              BPL SHIFTI
0099  84 9D              STY TAB           INSERT NEW CHARACTER TO LOWEST
009B  60                 RTS               PLACE

009C  XX          COUNT
009D  XX          TAB                      009D-00A3 TABLE OF 7 BYTES
00A4  XX          RNDWKA                   00A4-00A9 WORKING STORAGE FOR
                                                     RAND SUBROUTINE
```

R.L.

**********

KLEINANZEIGEN DER LESER:

Suche gebrauchte Teletype mit RS 232-Schnittstelle oder ähnliches, leicht an Mikrorechner anschließbares Hardcopyterminal. Albrecht Müller, DK2XJ, Görzallee 135, 1000 Berlin 45, Tel. 030-81 78 473 oder Christoph Müller, Wirfelstraße 8, 8070 Ingolstadt, Tel. 0841-51 962.

Wer hat ein Plotterprogramm für Drucker?
Klaus Schuenemann, An der Schanz 2, 5000 Köln 60, Tel. 0221-76 01 931.

## QUICKLOAD - MINI

E: Whenever VERSALOAD (No. 2 of this journal) is too long and only straight loading of Quickdumps with single-byte ID's is wanted, this program offers a handy alternative.

Für mit QUICKDUMP aufgezeichnete Programme (s. Heft 2 des Journals) bietet QUICKLOAD-MINI eine speichersparende Alternative für den reinen Ladevorgang, wenn die Bandsätze nur ID's von einem Byte tragen. Setzt man in die letzte Instruktion ein 'RTS' statt eines 'JMP', so dient QUICKLOAD auch als Unterprogramm. Abgesehen von den Formatunterschieden entspricht das Programm dem SUPERLOAD von John Oliver.

QUICKLOAD eignet sich sehr als ein mit Hypertape geschriebenes Vorprogramm zu einem mit QUICKDUMP aufgezeichneten Hauptprogramm großer Länge. Es kann sich eine erhebliche Einsparung der Gesamt-Ladezeit ergeben.

Analog kann man natürlich auch QUICKDUMP um etwa 50 Byte abmagern, wenn man auf die Verwendung von 6-Byte-Namen verzichtet.

```
                    QUICKLOAD

0000  AD F9 17   START   LDA ID          A DUPE FOR KIM-MONITOR
0003  85 D9              STA IDTEMP      PREVENTING RESTART FROM 'LOADT'
0005  A9 00      QUICKL  LDA #$ 00
0007  8D F9 17            STA ID
000A  A9 60               LDA #$ 60       RTS OPCODE FOR VEB
000C  8D EC 17            STA VEB
000F  20 8C 18            JSR LOADT+$18
0012  AA                  TAX             SAVE IN X, SHOULD BE 'ØØ'
0013  20 24 1A            JSR RDCHT
0016  AD EA 17            LDA SAVX+1      GET FULL BYTE
0019  C5 D9               CMP IDTEMP
001B  F0 06               BEQ LNBL        ON A MATCH
001D  E4 D9               CPX IDTEMP      X=Ø
001F  F0 02               BEQ LNBL        TREAT AS A MATCH
0021  D0 E2               BNE QUICKL      LOOK FOR NEXT RECORD

0023  20 24 1A            JSR RDCHT       GET NBL
0026  18                  CLC
0027  20 3E 19            JSR INTVEB+0C   OPCODE FOR VEB, CHECKSUM=0
002A  AD EA 17            LDA SAVX+1      GET FULL BYTE
002D  6D ED 17            ADC VEB+1
0030  85 E2               STA EOPL        GIVING ENDADDRESS
0032  08                  PHP             SAVE CARRY STATUS
0033  20 24 1A            JSR RDCHT       GET NBH
0036  28                  PLP             RETURN STATUS
0037  AD EA 17            LDA SAVX+1      FULL BYTE
003A  6D EE 17            ADC VEB+2
003D  85 E3               STA EOPH        GIVING ENDADDRESS HI
003F  A9 8D               LDA #$ 8D       STA-OPCODE
0041  8D EC 17            STA VEB
0044  20 24 1A   PATCH1   JSR RDCHT       MAIN LOOP
0047  AD EA 17            LDA SAVX+1      FULL 8-BIT-BYTE
004A  20 4C 19            JSR CHKT        CHECKSUM +[A]
004D  20 EC 17            JSR VEB         STORE
```

```
0050  20 EA 19          JSR INCVEB
0053  AD ED 17          LDA VEB+1       COMPARE WITH ENDADDRESS LOW
0056  C5 E2             CMP EOPL
0058  D0 EA             BNE PATCH1      GET NEXT CHARACTER
005A  AD EE 17          LDA VEB+2       AND NOW HI
005D  C5 E3             CMP EOPH
005F  D0 E3             BNE PATCH1      IF NOT THRU
0061  20 24 1A          JSR RDCHT       LOOK FOR '/'
0064  C9 2F             CMP #$ 2F
0066  F0 03             F0 03           SKIP IF OK
0068  4C 29 19          JMP LOADT9      ERROR EXIT WITH 'FF FF 1C'
006B  4C 15 19    END   JMP KIM         DISPLAY '00 00' FOR OK
006E  EA                NOP             FOR RALOAD
```

*R.L.*

**********

## THE 65XX FAMILY: PET 2001 - INFOS

*Seit September 1978 gibt es einen PET-Benutzer-Club (Anschrift: Commodore Büromaschinen GmbH., PET-Benutzer-Club, Frankfurter Straße 171-175, 6078 Neu Isenburg). Mitgliedsbeitrag DM 50,-.Mitglieder erhalten die etwa 6x jährlich erscheinenden Club-Mitteilungen und haben Zugang zur Programmbibliothek des Clubs. Dort sind auch Programme gegen mäßiges Entgelt erhältlich. Wer eigene Programme beisteuert, erhält dafür beliebige Programme des Clubs in dreifachem Umfang.*

*In Loseblattform wurden schon zahlreiche Hinweise für PET-Benutzer herausgegeben, sie betreffen in erster Linie die Bedienung und Programmierung des Gerätes.*

*Seitens der Firma Commodore ist derzeit nur das deutsche Handbuch erhältlich. Für etwa März 1979 ist die Herausgabe eines völlig neuen und ausführlichen Handbuches zum PET zu erwarten, ebenso ein Handbuch zur Programmierung. - Etwa im November ist die deutsche Übersetzung der ausführlichen Schnittstellenbeschreibung zu erwarten. Das englische Original hat auf 53 Seiten folgende Hauptabschnitte: PET Interfaces and Lines, Commands and Operations for Peripheral Devices, The IEEE-488-Bus - a short description. Dieses detaillierte Heft enthält viele Zeichnungen, Diagramme und Tabellen mit Signalbelegung der Pins, ferner eine Übersicht für die Wirkung der BASIC-Befehle auf die Peripherie.*

**********

## THE 65XX FAMILY: BUBBLE MEMORY VON ROCKWELL AUF DER ELECTRONICA

Den nichtflüchtigen Bubble-Speichern wird eine große Zukunft als Massenspeichern vorausgesagt. Als verschleißfreie Elektronikbausteine werden sie bei vielen Anwendungen eine Konkurrenz zu den Floppy-Disk Einheiten darstellen. Nach jahrelangen aufwendigen Vorbereitungen tritt nun Rockwell International mit seinen Blasenspeichern an den Markt. Die Lieferungen sollen 1979 beginnen. Die Systemfamilie *umfaßt die Bausteine* RBM256 für die Speicherung von 266500 Bits sowie das Bubble Memory Control Module RCM650, das voll kompatibel zum SYSTEM 65 und zu den 6502-Mikroprozessoren ist. Die Speicherbausteine werden zu funktionsfähigen Modulen auf Platinen zusammengefaßt. Das 1-Megabit Linear Bubble Memory Module RLM658 enthält 4 Stück RBM650, so daß zwei dieser Platinen byteweises Arbeiten erlauben. Der erwähnte Controller kann 16 dieser RLM658-Module betreiben, d.h. 2 Megabyte im Zugriff halten.

# R O L D I S - Scrolling Disassembler

E: This disassembler displays on the KIM-LEDs consecutive instruction lines with location, instruction in hex format, mnemonic opcode and mode of addressing. Apart from this the programming offers some interesting techniques: Forming of divisions and processing of 7 bulky tables. The logical path thru the program is managed by a table of low addresses which determine entry points of routines in the INTERMEDIATE DIVISION.

Für die 65xx-Systeme wurden gelegentlich schon Disassembler veröffentlicht. Einige bringen mit 2-6 hexadezimalen Ziffern (entspr. 1-3 Byte) nur die Instruktion zur Anzeige, im Wechsel dazu ggfs. auch die Adresse der zugehörigen Speicherplätze. Andere Disassembler zeigen oder drucken zusätzlich auch den mnemonischen Operationscode.

ROLDIS zeigt auch die Adressierungsart an; immerhin gibt es bei den 65xx-Mikros da ja 13 Möglichkeiten. Es handelt sich also um recht viel Information je Instruktionszeile. Die 6 LEDs des KIM reichen für deren Anzeige nicht aus. Die Texte werden daher als Laufschrift über die Anzeigen gerollt - eine praktische Anwendung der in Heft 1 dieses Journals veröffentlichten Display- und Rollroutinen.

Die Anzeige der Adressierungsart ist sicher nicht lebenswichtig, zumal man sich mit einer gut lesbaren 16x16 Felder umfassenden Tabelle für die Opcodes gut behelfen kann, der man auch Regelmäßigkeiten in der Zuweisung hexadezimaler Opcodes zu den Adressierungsarten entnimmt. Gleichwohl kann dieser Disassembler Feinheiten der Programmierung verdeutlichen und natürlich auch auf versehentlich falsch zugewiesene Adressierungsarten aufmerksam machen.

*Trotz dieser Nützlichkeit soll ROLDIS vor allem aber als ein Programmierbeispiel für die Tabellenverarbeitung und für die Herstellung eines Ausgabeformates dargestellt werden. Immerhin gehören 7 größere Tabellen zum Programm, 5 davon dienen der Programmsteuerung, 2 der Ausgabe von mnemonischem Code (3 Bytes) und von Adressierungsart (5 Bytes).*

ROLDIS ist auch aus folgenden Gründen ein lehrreiches Beispiel: Es gelang eine strenge Untergliederung des Programms in

FIRST MAINLINE DIVISION
INTERMEDIATE DIVISION
SECOND MAINLINE DIVISION
SUBROUTINE DIVISION
TABLE DIVISION.

Für viele Leser mag auch interessant sein, wie in Abhängigkeit von den in der Arbeitstabelle TABSR gefundenen Werten der Programmablauf zu diversen Einsprungspunkten der INTERMEDIATE DIVISION verzweigt wird.

ROLDIS nimmt wegen der großen Tabellen alleine fast 3 pages ein, zusammen mit den Dienstroutinen für das Rollen ist der Platzbedarf noch größer. Bei geschickter Ausnutzung des RAM im normalen KIM mag man mit dem Platz eben auskommen. Bei solcher Verschiebung und Zerlegung des Programms in neue Speicherbereiche muß man beachten, daß alle Einsprungspunkte der INTERMEDIATE DIVISION in einer page liegen müssen.

Die niedrigen Adressen dieser entry points sind in der TABSR enthalten, die ggfs. also zu ändern wäre. Die hohe Adresse, die dazugehörige page number, ist in der LISTA+2 als Adreßkonstante enthalten.

Der Disassembler wird zweckmäßig als Interruptroutine betrieben. Man bringt die Startadresse von ROLDIS als Vektor nach 17FA/FB, stellt das Display auf die Anfangsadresse einer zu untersuchenden Sequenz ein und drückt die ST-Taste, mit der ein NMI-Interrupt erzeugt wird. Der Disassembler arbeitet sich dann von der eingestellten Adresse von Instruktion zu Instruktion weiter. Unerlaubte Opcodes werden mit ERROR angezeigt, vor allem die auf 3, 7, B und F endenden. Nicht alle der verbleibenden 12 hexadezimalen Endziffern sind auch mit einer Anfangsziffer komplett besetzt, daher passieren einige unerlaubte Kombinationen unbeanstandet den Disassembler, z.B. D4, F4, DC, FC usw.. Solche Kombinationen wären mit umfangreicheren Arbeitstabellen auszufiltern. Natürlich könnte man sich in 32 Byte (= 256 bits) eine Wahrheitstabelle erlaubter Kombinationen schaffen.

Die Programmbedienung bietet folgende zusätzlichen Möglichkeiten: Durch Drücken der +-Taste kann man am Ende einer Zeile das Weiterrollen anhalten. Drückt man während des Zeilenwechsels die Taste 'B', und es liegt eine Branch oder Jump-Instruktion vor, so springt der Disassembler zu der Verzweigungsadresse. Das ist eine ganz nützliche Prüfung, ob man am richtigen Ziel anlangt.

Für die Ausgabe der mnemonischen und Adressierungsart-Texte wurden die entsprechenden Tabellen im 'Display-Format' für 7-stellige LEDs angelegt, ebenso die feststehenden Textteile in der DLINE. Letztere sammelt alle Informationen für die Ausgabe. Wer nun über ein ASCII-Gerät ausgeben möchte, sollte die die betreffenden Eintragungen auf den benötigten Code ändern.

Die Kommentare zum Programm werden jeweils den einzelnen Divisions vorangestellt.

FIRST MAINLINE DIVISION

Bei jedem Durchgang wird die LISTA aus der TABLE DIVISION mit ihren Anfangswerten in die Zeropage transportiert. Ein Teil der Parameter dient den Displayroutinen (Heft 1, S. 17 des 65xx MICRO MAG). Unter Indizierung mit Index X wird hernach der Operationscode mit AND und EOR aus den Tabellen TABA und TABE aufbereitet, bis sich in der Schleife ein Ergebnis '00' ergibt, womit Adressierungsart und Instruktionslänge erkannt sind. Das in der Schleife heruntergezählte X adressiert mit seinem erreichten Stand 3 weitere Tabellen, denen Werte für Einsprungspunkte in der INTERMEDIATE DIVISION, für den mnemonischen Textbeginn und für den Textbeginn der Adressierungsart entnommen werden. Abschluß mit indirektem Sprung in die INTERMEDIATE DIVISION.

```
                        FIRST MAINLINE DIVISION

0911  D8          START   CLD             BINARY MODE
0912  A2 0D              LDX #$ 0D        MOVE 13 PARMS TO ZEROPAGE
0914  BD 5A 0A    LOOP1   LDA LISTA-1,X
0917  95 E0              STA 00E0,X
0919  CA                 DEX
091A  D0 F8              BNE LOOP1
091C  A0 00              LDY #$ 00        FOR INDIRECT ADDRESSING
091E  B1 FA              LDA (POINTL),Y   START FROM CURRENT INSTRUCTION
0920  A8                 TAY              SAVE OPCODE IN Y
```

```
0921  A2 1E              LDX #$ 1E       FOR 30 LOOPINGS
0923  98        LOOP2    TYA             SUPPLY OPCODE
0924  3D 66 0B           AND TABA,X      FIND A MATCH FOR OPCODE
0927  5D 84 0B           EOR TABE,X
092A  F0 03              BEQ FOUND       ON MATCH
092C  CA                 DEX             ONE LESS TO GO
092D  D0 F4              BNE LOOP2       THRU ?

092F  BD C2 0B    FOUND  LDA TABOP,X     SAVE 3 VALUES FOR FURTHER
0932  85 FD              STA TOP         PROCESSING
0934  BD A3 0B           LDA TABMOD,X
0937  85 E0              STA TAMOD
0939  BD E1 0B           LDA TABSR,X     ADL OF INTERMEDIATE DIVISION
093C  85 E2              STA FDAL
093E  A2 00              LDX #$ 00       GODDBYE WITH LENGTH 00
0940  6C E2 00           JMP FDAL        JUMP INDIRECT
```

ZEROPAGE USED

```
00E0              TAMOD
00E1              LISTZP  EQU JPOLL
00E2              FDAL
00E3              FDAH
00E4              SCRCL                   FOR SCROLL
00E5              SCRCH
00E6              SCRSPD                  SCROLL SPEED
00E7              LENGL                   LENGTH OF DLINE
00E8              LENGH
00E8              LENGH
00E9  EA          JINDL                   ADDRESS OF SR SCROLL
00EA  0C          JINDH
00EB              PARML                   ADDRESS OF DLINE
00EC              PARMH
00ED              SPEED                   FLASHING SPEED
00EE              MEMO                    FOR FLASH

00F8              DAL     EQU INL
00F9              DAH     EQU INH
00FC              LOP     EQU TEMP
00FD              TOP     EQU TEMPX
```

INTERMEDIATE DIVISION

Die Adressierungsart bestimmt die Instruktionslänge. In der Reihe der hier folgenden Zwischenroutinen bestimmt der Einsprungspunkt, inwieweit X als Längenzähler erhöht werden soll (Anfangswert X=0). Für die Gruppe der Verzweigungs- und Sprungbefehle wird das Sprungziel errechnet oder übernommen, so daß später bei Betätigen der 'B'-Taste der Disassembler dorthin geführt werden kann (Ablage des Sprungzieles in DAL/DAH). Die diesem Zweck dienenden Routinen sind BRANCH und SPFUNC. Das nachfolgende ODD bereitet die sonstige absolute und Zeropage-Adressierung auf. PIMPL1 und PIMPL2 dienen entsprechend der Adressierungsart 'implied'.

Es muß weiterhin festgelegt werden, welcher mnemonische Text für die Opcodes der entsprechenden Tabelle zu entnehmen ist. Das besorgt die Subroutine FIND, die von hier aus aufgerufen wird (außer bei SPFUNC). Danach kann der Transport der Texte in der SECOND MAINLINE DIVISION erfolgen.

```
                        INTERMEDIATE DIVISION

0943  C6 E1       BRANCH  DEC JPOLL       SET A CONTROL <>0
0945  E8                 INX              ADD 1 ON LENGTH
0946  20 1D 0A           JSR FIND         ADVANCE IN TABLE OF OPCODES
0949  EA                 NOP              SORRY
094A  18                 CLC              PREPARE ADC
094B  A0 01              LDY #$ 01        FOR INDIRECT ADDRESSING
094D  B1 FA              LDA (POINTL),Y   GET SECOND BYTE OF INSTRUCTION
094F  30 14              BMI BACK         IF BRANCH AIMS BACKWARDS
0951  69 02        UP    ADC #$ 02
0953  08                 PHP              SAVE CARRY STATUS
0954  18                 CLC              ADD WITHOUT CARRY
0955  65 FA              ADC POINTL       TO CURRENT ADDRESS
0957  85 F8              STA DAL
0959  A5 FB              LDA POINTH       NOW HI ORDER
095B  69 00              ADC #$ 00        INCLUDE LAST CARRY
095D  28                 PLP              RETURN FIRST CARRY STATUS
095E  69 00              ADC #$ 00        ADD A POSSIBLE CARRY
0960  85 F9              STA DAH          DESTINATION NOW DETERMINED
0962  4C 9D 09           JMP MOVCOD+2     GOTO SECOND MAINLINE DIVISION

0965  69 02        BACK  ADC #$ 02        AS ABOVE
0967  08                 PHP
0968  18                 CLC
0969  65 FA              ADC POINTL
096B  85 F8              STA DAL
096D  A9 FF              LDA #$ FF        1 STEP BACK
096F  65 FB              ADC POINTH
0971  28                 PLP              RETURN CARRY STATUS
0972  69 00              ADC #$ 00
0974  85 F9              STA DAH
0976  4C 9D 09           JMP MOVCOD+2

0979  C6 E1        SPFUNC DEC JPOLL       SET CONTROL ≠ 0
097B  A0 02                LDY #$ 02      FOR INDIRECT ADDRESSING
097D  B1 FA        LOOP3   LDA (POINTL),Y GET NEXT BYTES OF INSTRUCTION
097F  99 F7 00             STA DAL-1,Y    AND STORE THEM AS A POSSIBLE
0982  88                   DEY            NEW DESTINATION
0983  D0 F8                BNE LOOP3
0985  E8                   INX            FOR LENGTH
0986  E8                   INX
0987  4C 9B 09     FORB    JMP MOVCOD     THIS IS THE ENTRY POINT FOR
                                          FORBIDDEN OPCODES

098A  E8           ODD     INX            ADD FOR LENGTH
098B  E8                   INX
098C  20 1D 0A             JSR FIND       GET ADDRESS OF OPCODE TEXT
098F  4C 9D 09             JMP MOVCOD+2

0992  20 21 0A     PIMPL1  JSR IMPL1      GET ADDRESS OF OPCODE TEXT
0995  4C 9D 09             JMP MOVCOD+2

0998  20 26 0A     PIMPL2  JSR IMPL2      DITTO
```

Dem vorstehenden Listing ist zu entnehmen, daß die Subroutine FIND so vielseitig angelegt wurde, daß die Adressenbestimmung für den Opcodetext bei den Adressierungsarten 'implied' auch über die entry points IMPL1 und IMPL2 erfolgen kann.

SECOND MAINLINE DIVISION

Die diversen Programmverzweigungen werden in dieser DIVISION wieder zusammengeführt. Die als Ausgabezeile dienende DLINE wird zunächst von den Resten der davorliegenden Interpretation befreit, alsdann beginnt der Transport aller Daten, z.T. unter Verwandlung in das Display-Format. Es sind dieses: Location, hexadezimale Instruktion und Texte für Mnemonics und Adressierungsart. Am Schluß dieses Abschnittes wird die Tastatur abgefragt. Ohne Tastendruck erfolgt Übergang zur nächsten Instruktion oder aber ein Tastendruck hält das Rollen an oder führt zu einer Verzweigungsinstruktion, wenn JPOLL entsprechend gesetzt war.

```
                        SECOND MAINLINE DIVISION

099B  86 FC     MOVCOD  STX LOP         SAVE LENGTH FROM X
099D  A2 06             LDX #$ 06       SET UP AS A COUNTER
099F  A9 00             LDA #$ 00       TO BLANK DLINE PARTS
09A1  9D 49 0A  LOOP4   STA DLINE+11,X
09A4  CA                DEX
09A5  D0 FA             BNE LOOP4       DO IT AGAIN ?
09A7  A2 07             LDX #$ 07       GET A DISPLACEMENT FOR DLINE
09A9  A5 FB             LDA POINTH      GET CURRENT HI ADDRESS
09AB  20 04 0A          JSR GETDIS      CONVERT TO DISPLAY CODE & STORE
09AE  A5 FA             LDA POINTL      SAME FOR LO
09B0  20 04 0A          JSR GETDIS

09B3  A2 0F     GETOPS  LDX #$ 0F       GET A DISPLACEMENT
09B5  A0 00     LOOP5   LDY #$00        FOR INDIRECT ADDRESSING
09B7  B1 FA             LDA (POINTL),Y  GET INSTRUCTION BYTES
09B9  20 04 0A          JSR GETDIS      CONVERT AND STORE
09BC  20 63 1F          JSR INCPT       ADVANCE POINTER BY 1
09BF  A9 40             LDA #$ 40       NOW WRITE A HYPHEN
09C1  9D 38 0A          STA DLINE,X
09C4  E8                INX             AND INCREMENT FOR LOCATION
09C5  C6 FC             DEC LOP         DOWNCOUNT
09C7  10 EC             BPL LOOP5       DONE ?

09C9  A4 FD             LDY TOP         GET ADDRESSER
09CB  A2 03             LDX #$ 03       3 BYTES TO MOVE
09CD  B9 67 0A  LOOPX   LDA TABMOT-1,   GET MNEMONICS
09D0  9D 4F 0A          STA DLINE+17,X  AND MOVE TO DLINE
09D3  88                DEY
09D4  CA                DEX
09D5  D0 F6             BNE LOOPX

09D7  A4 E0             LDY TAMOD       GET ADRESSER
09D9  A2 05             LDX #$ 05       5 BYTES TO MOVE
09DB  B9 67 0A  LOOPY   LDA TABMOT-1,Y  CONTAINING ADDRESSING MODE
09DE  9D 53 0A          STA DLINE+ ,X
09E1  88                DEY
09E2  CA                DEX
09E3  D0 F6             BNE LOOPY

09E5  20 CD 0C  SHOW    JSR FLASH       EXTERNAL ROUTINE FROM # 1 OF
09E8  20 6A 1F          JSR GETKEY      THIS JOURNAL. DISPLAYS & SCROLLS
09EB  C9 12             CMP #$ 12       IS '+' KEY DEPRESSED ?
09ED  D0 07             BNE CHECKB      SKIP IF NOT
```

```
09EF  A9 00           LDA #$ 00       RESTORE COMPARATOR FOR LENGTH
09F1  85 E5           STA SCRCH
09F3  4C E5 09        JMP SHOW        REPEAT SHOWING

09F6  C9 0B     CHECKB CMP #$ 0B      'B' REQUESTING A BRANCH ?
09F8  D0 07           BNE GOSTART
09FA  24 E1           BIT JPOLL       WAS IT A BRANCH ?
09FC  10 03           BPL GOSTART     SKIP IF NOT
09FE  20 CC 1F        JSR OPEN        MOVE DAL/DAH TO POINTER
0A01  4C 11 09        JMP START       DISPLAY FROM NEW LOCATION
```

SUBROUTINE DIVISION

Die SR GETDIS verwandelt Binärkode in 2-Byte Displaykode, und zwar unter Zuhilfenahme der TABLE HEX TO 7 SEGMENT im KIM-Monitor.

SR FIND isoliert aus Opcodes das MSD (most significant digit), manipuliert es ggfs. noch und multipliziert den erhaltenen Wert mit 3, zu dem dann noch der Tabellenanfangswert TOP addiert wird. Diese Bearbeitung ist notwendig, weil im vorderen Halbbyte der mnemonische Teil einer Instruktion verschlüsselt ist, während das zweite Halbbyte im allgemeinen die Adressierungsart beschreibt. Multiplikation mit 3, weil der mnemonische Code 3 Bytes umfaßt. Man kann also sagen, daß das vordere Halbbyte zur Adressenrechnung innerhalb einer Tabelle herangezogen wird.

```
                      SR GETDIS

0A04  48        ENTRY PHA             SAVE A ON STACK
0A05  4A 4A           LSR,LSR         CUT OFF LOWER 4 BITS
0A07  4A 4A           LSR,LSR
0A09  A8              TAY             TAKE AS AN ADDRESSER
0A0A  B9 E7 1F        LDA TABLE,Y     TO CONVERSION TABLE
0A0D  9D 38 0A        STA DLINE,X     AND MOVE TO DISPLAY LINE
0A10  E8              INS             ADVANCE 1 LOCATION
0A11  68              PLA             RESTORE FULL BYTE
0A12  29 0F           AND #$ 0F       CUT OFF     MSD
0A14  A8              TAY             AS ABOVE
0A15  B9 E7 1F        LDA TABLE,Y
0A18  9D 38 0A        STA DLINE,X
0A1B  E8              INX             FOR NEXT
0A1C  60              RTS

0A1D  98        FIND  TYA             Y STILL CONTAINS OPCODE
0A1E  4A              LSR             A:2
0A1F  10 06           BPL TAIL        BRANCH ALWAYS
0A21  98        IMPL1 TYA
0A22  29 7F           AND #$ 7F       CUT OFF MSB
0A24  10 01           BPL TAIL        BRANCH ALWAYS
0A26  98        IMPL2 TYA
0A27  86 FC     TAIL  STX LOP         LAY DOWN LENGTH OF OPERATION
0A29  4A 4A           LSR,LSR         GET MSD
0A2B  4A 4A           LSR,LSR
0A2D  85 E2           STA FDAL        STORE TEMP
0A2F  0A              ASL             Ax2
0A30  18              CLC
0A31  65 E2           ADC FDAL        ADD Ax1
0A33  65 FD           ADC TOP         ADD BASIS OF TABLE SECTOR
```

```
0A35  85 FD              STA TOP        THIS IS THE FINAL DISPLACEMENT
0A37  60                 RTS

0A38  00 00 00    DLINE  .BYTE          3 BLANKS IN DISPLAY LINE
0A3B  38 DC 58           .BYTE          MEANING 'LOC'
0A3E  00                 .BYTE          BLANK
0A3F  XX XX              .BYTE          ADH     OF LOCATION
0A41  XX XX              .BYTE          ADL     OF LOCATION
0A43  00                 .BYTE          BLANK
0A44  DC F3              .BYTE          MEANING "OP'
04A6  00                 .BYTE          BLANK
0A47  XX XX              .BYTE          FIRST BYTE OF INSTRUCTION
0A49  XX                 .BYTE          POSSIBLE HYPHEN
0A4A  XX XX              .BYTE          SECOND BYTE OF INSTR.
0A4C  XX                 .BYTE          POSSIBLE HYPHEN
0A4D  XX XX              .BYTE          THIRD BYTE OF INSTR.
0A4F  XX                 .BYTE          POSSIBLE HYPHEN
0A50  XX XX XX           .BYTE          MNEMONICS TEXT
0A55  00                 .BYTE          BLANK
0A54  XX XX XX           .BYTE          5 BYTES OF MODE TEXT
0A57  XX XX              .BYTE
0A59  00 00              .BYTE          2 BLANKS
```

TABLE DIVISION

LISTA enthält Anfangswerte, die bei jedem Durchgang des Disassemblers in der Zeropage erneuert werden. TABMOT enthält den Display-Text für die Adressierungsarten, im zweiten Teil den Mnemonics-Text.

```
                         TABLE DIVISION
0A5B  00 00       LISTA  .BYTE
0A5D  09 06              .BYTE          09 = PAGE# OF INTERMEDIATE DIV.
0A5F  00 FF              .BYTE
0A61  23 00              .BYTE          LENGTH
0A63  FD 0C              .BYTE          ADDRESS OF SR SCROLL
0A65  38 0A              .BYTE          ADDRESS OF DLINE
0A67  FF                 .BYTE          FLASHING SPEED

0A68              TABMOT
0A68  5E 04 50 58 78     .BYTE          MEANING "DIRCT"
0A6D  F7 FC ED DC 38     .BYTE          ABSOL
0A72  F7 FC ED 08 64     .BYTE          ABS-X
0A77  F7 FC ED 08 EE     .BYTE          ABS-Y
0A7C  DC F3 F7 FD F9     .BYTE          OPAGE
0A81  DC F3 FD 08 64     .BYTE          OPG-X
0A86  DC F3 FD 08 EE     .BYTE          OPG-Y
0A8B  86 B7 F3 38 5E     .BYTE          IMPLD
0A90  00 50 F9 38 00     .BYTE          _REL_
0A95  F3 D4 78 50 00     .BYTE          PNTR_   FOR ABSOLUTE INDIRECT
0A9A  F7 58 58 9C 00     .BYTE          ACCU_
0A9F  86 D4 5E 08 64     .BYTE          IND-X
0AA4  F3 D4 78 50 EE     .BYTE          PNTRY   FOR INDIRECT,Y
0AA9  F9 50 50 DC 50     .BYTE          ERROR
0AAE  B9 F7 38 38 00     .BYTE          CALL_   FOR "JSR"
```

```
     X0 X1 X2 X3 X4 X5 X6 X7 X8 X9 XA XB XC XD XE XF

0AB0           FC F3 38-FC B7 86-FC EA B9-FC EA ED-FC   BPL-BMI-BVC-BVS-B
0AC0 B9 B9-FC B9 ED-FC D4 F9-FC F9 3F-FC 50 75-9E ED   CC-BCS-BNE-BEQ-BRK-JS
0AD0 50-50 78 86-50 78 ED-3F 50 F7-F7 D4 5E-F9 3F 50-  R-RTI-RTS-ORA-AND-EOR
0AE0 F7 5E B9-ED 78 F7-38 5E F7-B9 B7 F3-ED FC B9-F9   ADC-STA-LDA-CMP-SBC-E
0AF0 50 50-FC 86 78-9E B7 F3-F9 50 50-ED 78 EE-38 5E   RR-BIT-JMP-ERR-STY-LD

0B00 EE-B9 F3 EE-B9 F3 64-F7 ED 38-50 3F 38-38 ED 50-  Y-CPY-CPX-ASL-ROL-LSR
0B10 50 DC 50-ED 78 64-38 5E 64-5E F9 B9-86 D4 B9-F3   ROR-STX-LDX-DEC-INC-P
0B20 F6 F3-B9 38 B9-F3 38 F3-ED F9 B9-F3 F6 F7-B9 38   HP-CLC-PLP-SEC-PHA-CL
0B30 86-F3 38 F7-ED F9 86-5E F9 EE-78 EE F7-78 F7 EE-  I-PLA-SEI-DEY-TYA-TAY
0B40 B9 38 EA-86 D4 EE-B9 38 5E-86 D4 64-ED F9 5E-78   CLV-INY-CLD-INX-SED-T
0B50 64 F7-78 64 ED-78 F7 64-78 ED 64-5E F9 64-F9 50   XA-TXS-TAX-TSX-DEX-ER
0B60 50-D4 DC F3-F9 50 50-                             R-NOP-ERR-

0B67              TABA,AND OPERATORS, 0B85 TABE, EOR OPERATORS
0B60                       FF 1F 1F 1F 1F 1F 1F FF FF
0B70 1F 1F 1F 1F 0F FF FF FF 1F 1F 9F 1F 1F 8F 9F 0F
0B80 9F 1F FF FF 03-A2 06 16 05 15 04 14 96 B6 0E 1E
0B90 0D 1D 0C BC BE 6C 11 01 80 09 19 8A 0A 08 00 10
0BA0 4C 20 03

0BA3              TABMOD, LOCATIONS (REL) OF MODE TEXT
0BA0           46 05 19 1E 19 1E 19 1E 23 23 0A 0F 0A
0BB0 0F 0A 0F 14 32 41 3C 05  05 14 28 37 28 28 2D 0A
0BC0 4B 46

0BC2              TABOP, LOCATIONS (REL) OF OPCODE TEXT BLOCKS
0BC0        8A B1 A2 A2 72 72 8A 8A AE B1 A2 A2 72 72
0BD0 8A 99 B1 90 72 72 8A 72 72 EA A2 BA 66 4E 90 69
0BE0 8A

0BE1              TABSR, LOW ADDRESSES OF INTERMEDIATE ROUTINES
0BE0     87 86 8B 8B 8B 8B 8B 8B 86 86 8A 8A 8A 8A 8A
0BF0 85 85 79 8B 8B 8B 8B 8A 92 8C 98 8C 43 79 79 87
```

THE TABLES CORRESPOND TO THE FOLLOWING ADDRESSING MODES:
ERROR, LDX #, ZEROPAGE, ZEROPAGE,X, ZEROPAGE, ZEROPAGE,X, ZEROPAGE,
ZEROPAGE,X, ZEROPAGE,Y, ZEROPAGE,Y, ABSOLUTE, ABS,X, ABS, ABS,X, ABS,
ABS,Y, ABS,Y, JMP IND, IND,Y, IND,X, IMMEDIATE, IMMEDIATE, ABS,Y,
IMPL, IMPL, IMPL, IMPL, BRANCHES, JMP, JSR, FORBIDDEN.  R.L.

**********

## FLOPPY DISK FÜR 65XX

In amerikanischen Zeitschriften wirbt die Fa. Johnson Computer (Anschrift: P.O. Box 523, USA Medina, Ohio 44256) für ihr Flexible Disk System for KIM-1. Es handelt sich dabei um das HDE Modell DM816-D1-1. Lieferung mit KIM-4 Motherboard und allen Kabeln, User Manual und FODS (File Oriented Disk System) Letzteres unterstützt multiple resident files, line numbered text entry und editing etc., ist kompatibel mit Microsoft KIM-1-BASIC und dem Aresco Assembler. Preis $ 1.995,-. Es soll auch eine preiswerte Mini Floppy geben. Näheres ist noch nicht bekannt.

Zahlreiche Anfragen beim Herausgeber deuten darauf hin, daß für die 65xx-Systeme hierzulande noch ein preiswertes Diskettensystem einschließlich Disk Operating System gesucht wird. Das Selbstentwickeln eines DOS wäre sehr aufwendig. - Der Herausgeber bittet die Leser um zweckdienliche Hinweise auf geeignete preiswerte Floppies mit DOS.

## THE 65XX FAMILY: DER MONITOR DES AIM 65

Die Auslieferung der ersten AIM 65 durch die Distributoren der Fa. Rockwell International soll Mitte/Ende Oktober 1978 beginnen. In dieses Heft konnte daher noch kein Anwenderbericht aufgenommen werden. Auch lagen bisher keine Handbücher vor, lediglich einige Fotokopien aus der Phase vor der Drucklegung. Die darin enthaltenen und über den vorliegenden Prospekt hinausreichenden Systembefehle sind für die erwartete Leistungsfähigkeit und für den Programmierkomfort des 'kleinen Entwicklungssystems' sicher schon recht aufschlußreich. Im Laufe der Zeit wird sicher noch manches nachzutragen sein, besonders auch hinsichtlich der interaktiven Führung des Benutzers durch das Betriebssystem.

MONITOR-ANWEISUNGEN:

```
ESC    RE-ENTER MONITOR
E      ENTER AND INITIALIZE MONITOR
T      RE-ENTER TEXT EDITOR
N      ENTER ASSEMBLER
5      ENTER AND INITIALIZE BASIC
6      RE-ENTER BASIC
CNTL   TOGGLE PRINTER CONTROL ON/OFF
PRINT  PRINT DISPLAY CONTENTS
LF     ADVANCE PRINTER TAPE
1      TOGGLE TAPE 1 CONTROL ON/OFF
2      TOGGLE TAPE 2 CONTROL ON/OFF
3      TAPE VERIFY
L      LOAD MEMORY
D      DUMP MEMORY
M      DISPLAY SPECIFIED MEMORY CONTENTS
SPACE  DISPLAY NEXT FOUR MEMORY CONTENTS
/      ALTER MEMORY CONTENTS
*      ALTER PROGRAM COUNTER
A      ALTER ACCUMULATOR
X      ALTER X REGISTER
Y      ALTER Y REGISTER
P      ALTER PROCESSOR STATUS
S      ALTER STACK POINTER
R      DISPLAY REGISTER CONTENTS
#      CLEAR ALL BREAKPOINTS
B      SET/CLEAR SOFTWARE BREAKPOINTS
4      TOGGLE BREAKPOINTS ENABLE ON/OFF
?      DISPLAY BREAKPOINTS
(I)    INSTRUCTION MNEMONIC ENTRY
K      DISASSEMBLE MEMORY
Z      TOGGLE INSTRUCTION TRACE ON/OFF
V      TOGGLE REGISTER TRACE ON/OFF
G      START EXECUTION AT PROGRAM COUNTER ADDRESS
H      TRACE PROGRAM COUNTER HISTORY
F1     USER FUNCTION, DITO F2,F3
```

TEXT EDITOR-ANWEISUNGEN:

```
Q      EXIT THE EDITOR AND RE-ENTER THE MONITOR
R      READ INTO TEXT BUFFER
L      LIST FROM TEXT BUFFER
I      INSERT ONE LINE
K      DELETE ONE LINE
```

```
T      MOVE THE LINE POINTER TO THE TOP
B      MOVE THE LINE POINTER TO THE BOTTOM
U      MOVE THE LINE POINTER UP ONE
D      MOVE THE LINE POINTER DOWN ONE
SPACE  DISPLAY CURRENT LINE
F      FIND A  CHARACTER STRING
C      CHANGE A CHARACTER STRING
```

**********

## THE 65XX FAMILY: KIM-1 WURDE BILLIGER

Die Spanne zwischen amerikanischen und deutschen Preisen verringert sich zusehends: Die Fa. Neumüller in Taufkirchen bietet den KIM-1 samt 3 Manuals zu jetzt DM 475,- zuzügl. 12 % MWSt an. Auch wenn es sich auf Zeit nicht um das letzte Angebot handelt, so lohnt der Selbstaufbau einer CPU-Platine mit allen Zusatzbauteinen kaum noch. (USA-Preis ab $ 219).

**********

## THE 65XX FAMILY: SYNERTEK'S VIM-1

Bei den weiterentwickelten single-board Computern sollte man auch den VIM-1 (Vertrieb Astronic, München) in Betracht ziehen. Leider liegen hier noch keine Datenblätter und Handbücher vor. Das Entwicklungssystem wird in den USA ab ca. $ 249 gehandelt. Es kann wie folgt kurz beschrieben werden: 6502 CPU, 4 kByte SUPERMON Betriebssystem, 1 kByte RAM, auf der Platine bis 4 K erweiterbar, dito freie Steckplätze für ROM/EPROM auf insges. 24 k, 50 I/O-Lines, Interface für Cassette (2 Bandgeschwindigkeiten: 8 Bytes/sec - wie KIM - 185 Bytes/sec), für TTY, Keyboard mit 28 Tasten, 6 Ziffernanzeigen.

Der angekündigte VIM-2 soll zusätzlich eine ASCII-Tastatur sowie einen HF-Modulator für den Anschluß eines normalen Fernsehers enthalten.

**********

## 65XX HARDWARE: ERWEITERUNGSPLATINEN VON GWK ELEKTRONIK

Der Herausgeber erhielt einen sorgfältig aufgemachten und aussagefähigen Prospekt der Fa. GWK Elektronik, Aachener Str. 56 in 5132 Übach-Palenberg, Tel. 02451-41911. Umfang 12 Seiten mit Schemazeichnungen für Platinen. Für den professionellen Anwender wie auch für Hobbyisten werden verschiedene Erweiterungsplatinen, pinkompatibel zum KIM-Expansion Connector, angeboten:

Motherboard, vollgepufferte Busse, 9 Steckplätze für Expansion, 3 für Applikation, voll pinkompatibel zum Expansion- bzw. Application Connector des KIM, alle ICs auf hochwertigen Steckfassungen.

RAM Board, 12 K statisch, in 4 kBlöcken auf der Platine beliebig adressierbar. EPROM Board 12 kByte, 12 Steckplätze für 2708/2758, EPROM-Programmer Board mit residentem Betriebsprogramm.

Besondere Aufmerksamkeit wurde der Störsicherheit der Adreß- und Datenbusse durch doppelte Pufferung und Abschluß gewidmet.

Der Besteller erhält mit seiner Lieferung eine ausführliche Dokumentation.

## THE 65XX FAMILY: NEUE INTELLIGENTE PERIPHERIEBAUSTEINE

Rockwell International hat weitere Chips für die 65xx-Mikroprozessorperipherie angekündigt: Der R6545 ist ein CRT-Controller (CRTC) im 40-Pin-Gehäuse, lieferbar ab Dez. 1978. Er kann 16 kSpeicherplätze adressieren und benötigt kein externes DMA (Direct Memory Access). Sein Bildschirmspeicher (Refresh RAM) kann entweder in Zeilen und Spalten oder rein binär adressiert werden. Dieser Speicher kann als unabhängiger Bereich des CRT-Controllers oder als vom Mikroprozessor adressierbarer Speicher angelegt sein.

Das Display ist voll programmierbar (Reihen-, Spalten- und Zeichenmatrix), ebenso der Cursor. Es gibt weiterhin ein Register für Lichtgriffel.

Der Chip erzeugt auch die Synchronsignale für Zeile und Bildwechsel, und zwar für 50 und 60 Herz, wahlweise mit oder ohne Zeilensprung. 5 Ausgangsleitungen geben einem Character Generator die Nummer der Rasterzeile innerhalb der Schreibzeile. - Für Blockladen in den Bildspeicher gibt es ein auf einen Anfangswert setzbares Adressenregister, das sich bei jedem Zeichentransport automatisch um 1 erhöht. Die Funktionen des Chips werden durch zahlreiche über den Datenbus geladene Registerworte gesteuert.

Für das 1. Quartal 1979 ist der R6541 angekündigt, ein Programmable Keyboard/Display Controller (PKDC), 40 Pin. Auch dieser versatile Baustein ist zur Entlastung der CPU konzipiert. Er kann entweder eine Matrix von 128 Tasten (8x16) abrastern und entprellen oder 64 statische Schalter (Sensoren) abfragen oder ein Keyboard im Strobe-Betrieb versorgen. Ein FIFO-Speicher dient der Datenpufferung.

Der für die Versorgung von LED-Displays angelegte Schaltungsteil enthält zwei 16x8bit Display-RAMs. Die Ansteuerungsmöglichkeiten mit einstellbarer Multiplexgeschwindigkeit sind vielfältig: Entweder 12 unabhängige Anzeigen mit 16 Zeichen in 8-Segment-Darstellung oder eine Anzeige mit 16 Zeichen und 16-Segment-Darstellung. Es ist auch ein gemischter Betrieb für Tastatur und Display möglich, dann können 16 Zeichen in 12-Segment-Darstellung gebracht und ein Tastenfeld von 16x8 bedient werden.

Zur Lieferung ab Dezember 1978 ist ebenfalls der R6551 angekündigt, ein Serial Communication Controller (SCC) mit Baud-Rate Generator für 15 Geschwindigkeiten zwischen 15 und 19200 Baud. Voller Duplexbetrieb zwischen Sender und Empfänger, synchron und asynchron, Erzeugung und Kontrolle von Parity. Programmierbare Wortlänge und Interruptkontrolle.

Viele leistungsfähige Anwendungen läßt auch der bereits lieferbare R6531 erwarten, der ROM-RAM I/O-Counter (RRIOC), mit 2kByte ROM, 128 Byte RAM, 15 I/O Pins, einem 8-Bit seriellen Kanal, einem vielseitig einsetzbaren 16-Bit-Counter (Timer, Pulsgenerator, Ereigniszähler, Pulsweitenmesser), I/O Handshake Control und 4 auf Impulsflanken reagierenden Interrupt-Eingängen. Soweit für das 40-Pin-Gehäuse. Bei der Ausführung mit 52 Pins stehen weitere 8 Ausgangs- und 4 Eingangsleitungen zur Verfügung.

Die erwähnten Bausteine ergänzen die Linie der bereits bekannten PIA R6520 mit seinen 2 Ports à 8 Bit, des VIA (Versatile Interface Adapter) R6522, der zusätzlich 2 programmierbare Time und ein 8-Bit serielles Schieberegister trägt und des R6532 RIOT mit seinen 16 I/O-Pins, Timer und seinem 128-Byte-RAM.

*KIM USER NOTES 11, S.2, E. Rehnke*

*KIMSI vs. KIM-4*

*Vergleich des S-100 Motherboards mit dem KIM-Motherboard.*

*KIM USER NOTES, Nr. 11, div.*

*TVT 6*

*Dieses Heft bringt schwerpunktmäßig eine Besprechung des KIM-TV-Displays in Einfachbauweise. Programme und Aufbaugesichtspunkte.*

*MICRO 6/78, S. 5, Charles R. Husbands*

*Design of a PET-TTY-Interface*

*Schaltung und Software f. Anschluß eines Fernschreibers zum Listen von Programmabschnitten, die auf dem Bildschirm angezeigt sind.*

*MICRO 6/78, S. 17, Michael McCann*

*A Simple 6502 Assembler for the PET*

*BASIC-Programm z. Eingabe von Source-Code mit zeilenweiser Generierung von Maschinencode. Keine symbolischen Adressen und Namen möglich.*

*MICRO 6/78, S. 25, Albert Gaspar*

*A Debugging Aid for the KIM-1*

*Kompaktes Dienstprogramm in den Zellen 1780-17E6 zur Programmentwicklung und -korrektur.*

*KILOBAUD 6/78, S. 22, Emerson Brooks*

*Taming the I/O Selectric (Part 1)*

*Beschreibung eines Interface und des Aufbaues der bekannten Kugelkopf-I/O-Schreibmaschine*

*KILOBAUD 6/78, S. 92, Dr. Mark Boyd*

*Two Systems sharing the same Bus*

*Das Interface eines 2-Prozessorsystems wird detailliert dargestellt, bezogen auf den Jupiter II, zusammen mit dem SWTP 6800.*

*KILOBAUD 7/78, S. 40, Emerson Brooks*

*Taming the I/O-Selectric (Part 2)*

*Software als Flowcharts und 6800-Mnemonics für den Betrieb der Kugelkopfmaschine als Mikroprozessorperipherie. Insbes. Codewandlung und Zeitbetrachtungen.*

*KILOBAUD 7/78, S. 60, David Swindle*

*A Sensible Expansion: Atwood Memory for Your KIM*

*Anschluß der 4k-RAM-Karten der Kathryn Atwood Enterprises.*

*KILOBAUD 7/78, S. 100, Bill Fuller*

*Compatibilty and the Altair Bus*

*Grundsatzartikel zum S-100-Bus. Benutzung der 'unused pins' durch die Anbieter der S-100-Karten. Nennung geeigneter und weniger geeigneter Ergänzungen. Kritik des Bussystems.*

*KILOBAUD 8/78, S. 36, Peter A. Stark*

*Data and Adress Busses*

*Einführungsartikel über Bustreiber- und Transceiver, Adressendekodierung.*

*KILOBAUD 8/78, S. 40, John Leslie*

*Software Debugging for Beginners*

*Anleitung zur Analyse und Korrektur fehlerhafter Programme.*

*KILOBAUD 9/78, S.22, R.M.Law und D.C. Mitchell*

*(Con)Text Editor*

*Programm in BASIC mit ausführlicher Beschreibung eines Textverarbeitungssystems.*

*KILOBAUD 9/78, S. 100, James Grina*

*Supercheap 2708 Programmer*

*Aufbau und KIM-Programm für ein sehr preiswertes EPROM-Programmiergerät, das auch als Platine bezogen werden kann.*

# GWK EURO BOARD EXPANSION SYSTEM für KIM

Es bietet sowohl dem professionellen Anwender,als auch dem Hobbyisten,die Möglichkeit,seinen KIM optimal zu erweitern.
Dies wird erreicht durch die folgenden grundlegenden Konzepte:
Doppelte Pufferung von Daten- und Adressbus bewirken größtmögliche Störsicherheit.
Die Adressdecodierung der einzelnen Komponenten erfolgt auf jeder Karte.Das heißt,jede Komponente kann beliebig adressiert werden.Es wird kein Adressraum verschenkt.
Der GWK EURO BOARD EXPANSION SYSTEM Bus und die Steckverbinder der einzelnen Komponenten sind pinkompatibel zum Expansion Connector des KIM.Daraus ergibt sich,daß einzelne Karten auch ohne das MOTHER BOARD direkt am KIM laufen.
Bisher sind folgende Komponenten lieferbar.

## MOTHER BOARD ohne Steckerleisten 325,--

Einbau in Gehäuse 19 Zoll oder direkt an KIM steckbar.Daten-Adress- und Control Bus sind voll gepuffert und dynamisch abgeschlossen.9 Steckplätze für Expansion,3 für Application, 1 mit genügend Platz für ein Netzteil.

## RAM BOARD 895.--

12 K Byte statisches Ram.In Blöcken zu je 4K Byte beliebig adressierbar.

## EPROM BOARD ohne Eproms 395,--

12K Byte für die 1K Eproms 2708 oder 2758.

## EPROM PROGRAMMER BOARD 745,-- 795,--

Mit residentem Betriebsprogramm.Softwareumschaltung für 1K-,2K-,4K Byte EPROMS xx08/58,xx16,xx32.Zwei Ausführungen: Für Eproms mit einer oder mit drei Betriebsspannungen.

## CROSS ASSEMBLER 650X - PDP11 985,--

Zusätzliches Info anfordern.

| | |
|---|---|
| EPROM TMS 2708,1K Byte,3 Vers.Spng. | 23,-- |
| EPROM TMS 2516,2K Byte,1 Versorg.Spng. | 115,-- |
| STECKERLEISTEN zum MOTHERBOARD | |
| 22 Polig für Netzteil Amphenol 143-022-01-102 | 7,12 |
| 44 Polig für Exp.u.Appl. " 225-22221-110 | 9,60 |

Alle Preise zuzüglich Versandspesen und MWST.

**GWK** TECHNISCHE ELEKTRONIK
HARDWARE SOFTWARE SYSTEMENTWICKLUNG
Aachener Str.56 · D 5132 Uebach Palenberg
Tel.: 02451 / 41911

# 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, der als Datenverarbeitungsfachmann auch freiberuflich beratend tätig ist.

COPYRIGHT 1978 BY ROLAND LÖHR. BEITRÄGE UND PROGRAMME DIENEN DEM PERSÖNLICHEN 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.

Wegen des Zusammenhanges der Hefte erhalten hinzukommende Abonnenten, wenn nicht anders gewünscht, 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. 4 IST DER 5. DEZ. 1978

*In den nächsten Heften finden Sie u.a.:*

*Weitere Grundsatzartikel für die Programmierungstechnik - Fortsetzungen für 65xx-Makros, für Advanced Subroutine Package - Anwenderschaltungen - Ausführlicher Bericht über den AIM 65 und sein Betriebssystem ....*

*BYTE 7/78, S. 12, Robert Baker*

*KIMER, A KIM-1 Timer*

*Programm für Digitaluhr und Timer mit KIM-1. Leider liegt ein schwerwiegendes Mißverständnis hinsichtlich des im Interrupt arbeitenden Timers vor.*

*BYTE 8/78*

*PASCAL*

*Diese höhere Programmiersprache ist Schwerpunktthema in diversen eingehenden Aufsätzen dieses Heftes.*

*BYTE 8/78, S. 64, Peter Nelson*

*The Number Chrunching Processor*

*Datenblattbeschreibung des MM57109 von NS, Interface für Mikroprozessor und 8080-Programm für den Betrieb.*

*BYTE 9/78, S. 58, K.-M. Chung und H. Yuen*

*A Tiny PASCAL Compiler, Part 1*

*The P-Code Interpreter. Beschreibung des maschinenunabhängigen Sprachaufbaues. Zahlreiche logische Flußdiagramme. Nachvollziehbarer Grundsatzartikel*

65xx MICRO MAG