65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

#1 (Neuauflage) JUNI 1978

ZUM ANFANG

Sie halten die erste Nummer eines Journals in Händen, das sich schwerpunktmäßig mit einer Mikroprozessorfamilie befaßt, den 65xx Mikros von MOS-Technology, Rockwell, Synertek und den daraus aufgebauten höher entwickelten Systemen, wie KIM, AIM 65, PET, ALPHA usw. sowie mit den vielen Weiterentwicklungen, die wir in den nächsten Jahren erwarten dürfen.

Über das Vordringen der Mikros ist viel geschrieben worden. Dem ist im Moment nichts hinzuzufügen. Stattdessen fangen wir da an, wo es am meisten kneift, bei der Software. Der Leser wird in vielen langen Stunden an seinem System erfahren haben, wie wertvoll lauffähige Programme eigentlich sind und wieviel Zeit in ihre Entwicklung investiert werden muß. Demgegenüber ist die Hardwareseite, über die wir auch berichten, mit einem Kauf im allgemeinen bequemer abzudecken.

INHALTSVERZEICHNIS

Eine ganze Generation von Digitalelektronikern und Hobbyisten muß sich auf das neue "Wie" einstellen, auf das Programmieren. Die 65xx-Systeme sind für die Einarbeitung bestens geeignet, zumal für sie weltweit ein sehr offener Informationsaustausch besteht.

Im 65xx MICRO MAG finden Sie einen zuverlässigen Begleiter, der Sie auf den vielen Schritten in die Zukunft mit didaktisch dargebrachter Information unterstützt.

Die Erfahrung zeigt, daß der Betreiber eines Systems vor allem für dieses umfassende Unterrichtung wünscht. Die in unserem Sprachraum noch vorhandene Zersplitterung des Informationsangebotes ist enttäuschend. Man kann nicht damit zufrieden sein, wenn in einer Zeitschrift pro Jahr nur ein bis drei Beiträge erscheinen, die das eigene System betreffen. Beim Aufschlagen dieser ersten Nummer von 65xx MICRO MAG werden Sie feststellen, daß das nicht so sein muß. Wenn sie auch allein vom Herausgeber gestaltet wurde, so liegt das an Termingründen bei den befreundeten Autoren, die sich dafür im August um so mehr zu Worte melden werden.

KIM ist derzeit noch das am weitesten verbreitete 65xx-System. Es verdient daher im Moment noch die stärkste Beachtung. Aber schon in wenigen Monaten dürften weitere höher entwickelte Systeme einen größeren Freundeskreis gefunden haben. Die redaktionelle Arbeit wird das berücksichtigen und neben dem Maschinen-Assembler zunehmend Darstellungen in BASIC bringen.

Information steht bei den weit geplanten redaktionellen Zielen im Vordergrund. Im Anfang dürfte die Aufmachung als Manuskript-Druck diesem Zweck am besten dienen.

Der Leser möge im Laufe der Zeit die Zuversicht gewinnen, daß man aus einem klaren Gedankengang eigentlich jedes erforderliche Programm entwickeln kann. Die Programmierhilfen und Dienstprogramme, die ihm dabei zur Verfügung stehen, werden immer besser, nachdem in gemeinsamer Arbeit und Diskussion immer mehr Grundprobleme gelöst sind.

Damit ist ein weiterer wichtiger Punkt angesprochen: Ein Journal wie 65xx MICRO MAG braucht das Feedback aus der Leserschaft, den Kommentar, die Wünsche und die Fragen sowie natürlich auch die Mitarbeit freier Autoren. Bei mehr als 7000 in Europa verbreiteten KIM-Systemen ist klar, daß die Grundfragen mindestens ebenso oft individuell angegangen wurden, weil Information fehlte. In vielen Fällen ist das sicher lehrreich gewesen, nicht aber wirtschaftlich. Und man darf weiterhin mit Bestimmtheit annehmen, daß es einige tausend ausgekügelte Programme schon gibt und natürlich auch viele interessante Anwenderberichte, die zum Stolz der Autoren gereichen könnten, wenn sie in diesem Journal abgedruckt werden. Wie fruchtbar kann es sein, Diskussionspartner zu finden, die sich mit den gleichen Problemen beschäftigen und die eigene Arbeit anregen! Also bitte nicht hinter'm Berg halten.

VON HIER AUS STEHEN INFORMATIONSBLÄTTER FÜR AUTOREN ZUR VERFÜGUNG.

REDAKTIONS- UND ANZEIGENSCHLUSS FÜR NR. 2 IST DIENSTAG, DER 15. AUG.1978.

Empfehlen Sie bitte das 65xx MICRO MAG weiter!

Verschieblich laden und modifizieren:

RELOCATE AFTER LOAD

E: Utilities LAMID and RALOAD allow flexible programming and automatical adjustment of absolute addressing (with tables included) after relocational load of programs if only a very simple 'syntax' is applied, thus facilitating program exchange and usage of libraries. Length-operators = pseudo-opcodes are introduced.

Von diesem ersten Dienstprogramm hofft der Herausgeber, daß es für den Programmaustausch in der Lesergemeinde eines der nützlichsten sein möge: Auf Cassette gespeicherte Programme können gezielt mit ihrer ID verschieblich geladen und sofort lauffähig gemacht werden. Natürlich kann man diesen Lösungsweg nicht nur auf Einspeicherungen vom Lochstreifen oder von der Magnetplatte her, sondern auch für die Eingabe von der Tastatur her anwenden.

Welche Plage hatte der Programmierer bisher, wenn er sich aus lauffähigen Teilprogrammen seiner 'program library' ein größeres Programm zusammenschweißen wollte! Wenn er mit ID = FF verschieblich lud, erwischte er möglicherweise ein falsches Programm, war es das richtige, dann taugte die Programmdokumentation nichts mehr, weil sich alle Adressen geändert hatten. Wer vorsichtig operierte, lud das Programm an seinen alten Platz, änderte es mit den Dokumenten vor Augen von Hand oder mit Jim Butterfields RELOCATE ab und transportierte es dann an die neue Stelle.

Aber: Viel war zu beachten und zu zerstören. Tabellenwerte blieben ein Fremdkörper, die Maschine hätte sie zu schnell als Instruktion gelesen. Und oft genug war der Speicherbereich durch den Programmaufbau bereits belegt.

RELOCATE AFTER LOAD macht Schluß damit und ermöglicht es sogar dem Tüftler, Riesenprogramme mit kleinstem Speicher abzufahren: Er hält ein Stamm- und Verwaltungsprogramm neben dieser utility (=Dienstprogramm) im Speicher resident und zieht sich benötigte Teilprogramme von seiner Bandbibliothek in einen Überlagerungsbereich nach.

Der größte Nutzen dieser Überlegungen mag aber darin liegen, daß jeder Autor seine Programme in diesem Journal so zum Abdruck bringen kann, wie sie bei ihm geschrieben und dokumentiert sind. Der Leser kann mit der utility die Verschiebung in seinem Speicher dann mühelos selbst vollziehen.

Das Dienstprogramm hat zwei unabhängige Hauptteile: LAMID als Laderoutine (an Stelle der KIM-Routine ab 1873) und das eigentliche Umrechungsprogramm ROLOAD, das durch Unterprogramme gestützt wird. Bei den Unterprogrammen zur einfachen Veränderung von Adreßpointern und bei LBEST habe ich unter Anpassung auf eine Arbeit von Dan Fylstra zurückgegriffen (BYTE 2/78, S. 68). Vereinfachungen wären hier mit MACROS möglich.

Die Unterprogramme ADJUST und LBEST erkennen und verwenden Längenoperatoren, hier LOP genannt. Bekanntlich sind in den 65xx-CPUs eine Reihe von möglichen Befehlscodes noch nicht installiert, z.B. die 64 Kombinationen, die auf x3, x7, xB und xF enden. Ihre Identifikation ist einfach, weil sie im Gegensatz zu allen anderen in den beiden niedersten Bits eine 1 haben.

Binäres Grundmuster dieser LOPs ist also xxxx xx11. Die vorderen 6 Bits werden als eine Binärzahl zwischen dezimal 0 und 63 interpretiert, um Programmteile, speziell also Konstanten, Tabellen und andere Operatoren überspringen zu können. Wie nützlich eine solche Interpretation der LOPs ist, werden auch weitere Programme zeigen. Es können also Tabellen und Konstanten mitten in Programmen auftauchen, ohne daß die Umrechnung gestört wird. Der Programmierer muß natürlich sicherstellen, daß seine Programme Tabellen als solche behandeln.

Tabelle: Längenbestimmung durch LOPs (Pseudo-Ops)

Hexadezimalcode	03	07	0B	0F	13	17	1B	1F	usw bis	F3	F7	FB	FF
Dezimal	0	1	2	3	4	5	6	7		60	61	62	3

Anmerkungen: a) Das markante FF wird für andere Zwecke als Länge 3 interpretiert. b) Der LOP ist als 1 Byte in die gesamte Bytezahl einer Tabelle einzurechnen.

Ein LOP kann daher sich selbst und bis zu 61 weitere Bytes aus einer Adressenumrechung ausschließen. Bei längeren Tabellen müsste dahinter sofort ein nächster Längenoperator folgen. Nicht schön, also enthält der Programmzweig TABADJ eine Vorkehrung, die davon frei macht:

Warum sollte man nicht am Schluß eines Programmes in zwei Bytes die absolute Basisadresse angeben, in der eine beliebig lange Tabelle beginnt? Gesagt, getan, nur muß dann RALOAD erkennen können, daß ein solcher Fall vorliegt. Das geschieht so: Immer wenn das letzte Byte des frisch geladenen Programmes ein "EA" ist (=NOP), merkt RALOAD, daß keine Tabelle angehängt ist. Der Gebrauch von einem oder von mehreren davorliegenden LOPs ist damit nicht ausgeschlossen. Im übrigen beißt das angehängte NOP keine Programmausführung.

Ist das letzte Byte ungleich EA, dann wird ungefragt eine Adressenumrechnung vorgenommen (ältere Programme also redigieren!). Die neue Anfangsadresse der Tabelle (Tabellenbasisadresse) wird dem Programm in seine beiden letzten Bytes geschrieben, damit es auch nach einer weiteren Abspeicherung auf Band verschieblich bleibt. RALOAD wird angewiesen, mit der Adressenumrechung vor der Tabelle aufzuhören. Auf diese Weise "weiß" das Programm auch an der neuen Stelle immer, wo seine Tabelle beginnt. Beliebige Weiterverschiebungen via Programmladen sind möglich.

RALOAD läßt daher folgende Programmstrukturen wahlweise zu:

a) INSTR, LOP, TAB, EA

b) INSTR, LOP, TAB1, LOP, TAB2, ..., EA

c) INSTR, LOP, TAB1, INSTR, LOP, TAB2, ... EA

d) INSTR, TAB, Tabellenbasisadresse

e) INSTR, LOP, TAB1, ... INSTR, TAB2, Tabellenbasisadresse2

Weitere Mischformen sind möglich.

Diese Verschiebesyntax ist so einfach, läßt soviele Freiheiten und gibt soviele Gestaltungsmöglichkeiten, daß man seinen Programmbestand mühelos darauf anpassen kann, und natürlich auch Programmbeiträge zu diesem Journal.

Also: EA als letztes Byte schreiben oder
Tabellenbasisadresse (TABADL, TABADH).
LOPs nach Bedarf einsetzen.

Syntax

Ein Hinweis: RALOAD ändert nur absolute und absolut-indizierte Adressen, ferner absolute Sprungziele. Ein Grundsatzartikel wird zeigen, warum die Zeropage eine Besonderheit darstellt.

Das Programm kann nicht erkennen, wenn Sie Adressen als Konstante laden und zur Programm-Modifikation in Instruktionen einsetzen. Um solche Eigenheiten zu erkennen, müßte die Interpretation von Instruktionen wesentlich weiter getrieben werden.

Natürlich sind LAMID und RALOAD verschieblich geschrieben.
Wenn Ihnen als Anwender die Speicherplätze ab 0200 zu "wertvoll" sind, können Sie den ersten Teil des Programmes von Hand ändern und z.B. ab 0100 laden, die letzte subroutine LBEST nach 1780 ff.

Nun im einzelnen:

LAMID - Laden verschieblich mit ID

Will man mit dem KIM-Betriebssystem verschieblich laden, so gibt man "Blindfluganweisung": FF als ID nach 17F9 und neue Startadresse nach SAL, SAH in 17F5/17F6. Für LAMID gibt man die wahre Identität in's ID und startet in 0200.

Die KIM-Routinen sind für anderweitige Verwendung bekanntlich wenig flexibel angelegt. Aber da gibt es den Volatile Execution Block (VEB) in 17EC bis 17F1. In diese RAM-Zellen stellt KIM je nach BEdarf Instruktionen ein. LAMID manipuliert zunächst ein RTS dorthin, so daß die KIM-Routine ab 1875 als Unterprogramm aufgerufen werden kann. Nachdem das ID und das erste Programmbyte gefunden sind, wird die alte Startadresse des Programms für Umrechnungszwecke gerettet und die neue erwünschte zusammen mit STA=8D nach VEB gebracht. Der Rest ist wie gewohnt: KIM kehrt mit 0000 oder mit FFFF bei Fehler zurück.

Die Zeit für den Austausch ist knapp. Man kann LAMID durch Zeropage-adressierung bei den Transporten beschleunigen.

R A L O A D - Relocate after LOADT

Im Hauptprogramm (START 0227) wird zunächst das Inkrement der Verschiebung berechnet. Sodann wird das letzte Byte auf "EA" als Merker geprüft. Bei Erkennen einer Tabellenabsisadresse erfolgt deren Umrechnung auf den neuen wahren Wert (TABADJ). SCHLEI rastert zusammen mit den Unterprogrammen nun Instruktion für Instruktion nach der Längentabelle ab. Bei 3-byte-langen Instruktionen und längeren werden in ADJUST zunächst die LOPs für die Bearbeitung ausgeschieden. Echte Instruktionen mit absoluter Adressierung werden nur dann umgerechnet, wenn sie sich auf den neuen Programmspeicherplatz (einschließlich Tabellenraum!) beziehen. Adressierungen mit Ziel unterhalb Programmanfang und oberhalb Tabellenende werden also nicht umgerechnet.

Beim Zusammenschweißen von Teilprogrammen zu einem größeren Programm kann ein angehängtes EA immer durch das erste Byte des nächstfolgenden Segmentes überschrieben werden. Das gilt auch für die beiden Bytes einer Tabellenbasisadresse. Im letzteren Fall ist das entstehende neue Programm aber nicht mit RALOAD verschieblich, weil keine Programmstruktur vorliegt, wie sie oben unter a - e skizziert wurde. Soll das neue Programm verschieblich sein, so ist für Tabellen etc. immer die Verwendung von Längenoperatoren zu empfehlen. Oder man verkettet so, daß das Segment mit Tabellenbasisadresse immer an den Schluß kommt.

Laderoutine

```
0200  A9 60     LAMID   LDA #$ 60       Return-Opcode für VEB
0202  20 75 18          JSR LOADT+2     Einsprung in die KIM-Routine
0205  48                PHA             Akku retten
0206  AD ED 17          LDA VEB+1       Die dem Kennsatz entnommene alte
0209  8D F7 17          STA EAL         Startadresse nach EAL, EAH retten
020C  AD EE 17          LDA VEB+2
020F  8D F8 17          STA EAH
0212  A9 8D             LDA #$ 8D       Opcode für STA
0214  8D EC 17          STA VEB         nach VEB abspeichern
0217  AD F5 17          LDA SAL         und nun die gewünschte neue
021A  8D ED 17          STA VEB+1       Startadresse hinzufügen
021D  AD F6 17          LDA SAH
0220  8D EE 17          STA VEB+2
0223  68                PLA             Akku zurück zur Abspeicherung in ...
0224  4C EC 17          JMP VEB         entspricht CARD 257 im KIM-Monitor
```

Umrechnungsprogramm

```
0227  D8        RALOAD  CLD             Subtraktionsvorbereitung
0228  38                SEC
0229  AD F5 17          LDA SAL         Umspeicherung für die Unterprogramme
022C  85 E0             STA ANFADL
022E  ED F7 17          SBC EAL         Inkrement,L der Verschiebung berechnen
0231  85 EC             STA INCL
0233  AD F6 17          LDA SAH         dito für high order
0236  85 E1             STA ANFADH
0238  ED F8 17          SBC EAH
023B  85 ED             STA INCH
023D  20 BD 02          JSR BEGINN      Laufende Adresse = Beginnadresse
0240  AD ED 17          LDA VEB+1       Endadresse+1 des Ladevorganges
0243  85 E2             STA ENDADL      umspeichern
0245  AD EE 17          LDA VEB+2
0248  85 E3             STA ENDADH

024A  A9 01             LDA #$ 01       LAENGE = 1
024C  85 E8             STA LAENGE
024E  20 DC 02          JSR RUECK       ... und zurücksetzen
0251  A0 00             LDY #$ 00
0253  B1 E2             LDA (ENDADL),Y  Hole letztes Byte und
0255  C9 EA             CMP #$ EA       prüfe, ob EA als Merkmal vorhanden
0257  F0 17             BEQ GETOP       Oberspringe, wenn ja.
```

Umrechnung einer evtl. vorhandenen Tabellen-Basisadresse

```
0259  48        TABADJ  PHA             Rette Akku mit dem letzten Byte
025A  20 DC 02          JSR RUECK       nochmals 1 zurücksetzen
025D  18                CLC             möglichen Übertrag löschen
025E  A5 EC             LDA INCL        Verschiebung zur früheren Tabellenbasis
0260  71 E2             ADC (ENDADL),Y            addieren und das
0262  91 E2             STA (ENDADL),Y  Ergebnis als neue Tabellenbasis einsetze
0264  C8                INY
0265  AA                TAX             Ergebnis festhalten
0266  68                PLA             Hole Tabellenbasis, high order
0267  65 ED             ADC INCL        und rechne um, wie für low order
0269  91 E2             STA (ENDADL),Y
026B  85 E3             STA ENDADH
026D  8A                TXA             Zurückholen low order
026E  85 E2             STA ENDADL      und für die Unterprogramme
                                        abspeichern
```

Hauptschleife zur Durchprüfung des verschobenen Programms

```
0270  C6 E8     GETOP   DEC LAENGE      LAENGE = 0 für erstes VORAN
0272  20 C6 02  SCHLEI  JSR VORAN       eine Instruktionslänge weitersetzen
0275  F0 0D             BEQ OUT1        Ausstieg, wenn Endadresse erreicht
0277  20 E8 02          JSR LBEST       sonst Längenbestimmung
027A  C0 03             CPY #$ 03       Länge 3 oder mehr Bytes?
027C  90 F4             BCC SCHLEI      nächste Instruktion, wenn nein
027E  20 87 02          JSR ADJUST      sonst prüfen auf LOP, ggfs. umrechnen
0281  4C 72 02          JMP SCHLEI      weiter, in der Schleife
0284  4C 4F 1C  OUT1     JMP KIM         Ausstieg zum Monitor.
```

Unterprogramm zur Prüfung auf LOP, ggfs. Umrechnung der Adressen

```
0287  A0 00     ADJUST  LDY #$ 00       Erstes Byte der Instruktion per
0289  B1 E4             LDA (LAUADL),Y  Pointeradresse holen
028B  29 03             AND #$ 03       Längenoperator? Unerwünschte Bits
028D  49 03             EOR #$ 03       entfernen
028F  D0 01             BNE ADJGO       NEIN: Oberspringe
0291  60                RTS             Zurück in die Hauptschleife
```

S E R V I C E F O R D E N L E S E R

Als Abonnent haben Sie die Möglichkeit, einmal jährlich eine private Kleinanzeige im 65xx MICRO MAG abdrucken zu lassen, und zwar gratis. Sie darf bis zu 4 Schreibmaschinenzeilen lang sein. Da Chiffren nicht geplant sind, sollte Sie Ihre volle Anschrift enthalten.-
Eine hervorragende Gelegenheit für die Kontaktaufnahme in Ihrem Gebiet, um Material zu suchen oder anzubieten.

Telefonische Anrufe bei Autoren, Inserenten und beim Herausgeber bitte in bürgerliche Stunden legen! Bei brieflichen Rückfragen bitte adressierten Freiumschlag beilegen.

Folgendes Angebot gilt versuchsweise: Alle in diesem Heft enthaltenen P... ...innen Sie gegen Vorauskasse auf Tonbar... ...aus erhalten. Aufzeichnung im sch... ...IAPE. DM 13,- ersparen stundenlanges Eintast... ... Abstimmen.

Service eingestellt

Gedruckte und sorgfältig e... ...fene Programmierbögen sind eine große Hil... ...e Aufzeichnung eigener Arbeiten. DM ... 200 Blatt frei Haus. DIN A4, einseitig be...uckt.

Service eingestellt

```
0292  C8          ADJGO   INY             Für folgendes Byte: Y = 1
0293  18                  CLC
0294  B1 E4               LDA (LAUADL),Y  Holen ADL der Instruktion
0296  65 EC               ADC INCL        Verschiebung addieren
0298  AA                  TAX             und in X retten.
0299  C8                  INY             Y = 2
029A  B1 E4               LDA(LAUADL),Y   ADH + Verschiebung nach Akku
029C  65 ED               ADC INCH

029E  C5 E1               CMP ANFADH      Abbruch, wenn Ergebnis unterhalb
02A0  90 0D               BCC OUT         des neuen Startpunktes liegt
02A2  D0 04               BNE HIGB        aber noch Obergrenze prüfen
02A4  E4 E0               CPX ANFADL      ADH war gleich, nun auch noch ADL?
02A6  90 07               BCC OUT         nein, bleibt unterhalb

02A8  CD EE 17  HIGB      CMP VEB+2       Nun high order vergleichen mit dem Ende+1
02AB  90 08               BCC GOGO        umrechnen, weil innerhalb
02AD  F0 01               BEQ GO          high order gleich, nun auch low order pr.
02AF  60          OUT     RTS             Abbruch

02B0  EC ED 17  GO        CPX VEB+1       low order des Programmendes+1
02B3  B0 FA               BCS OUT         Abbruch,wenn gleich oder höher VEB+1
02B5  91 E4               STA (LAUADL),Y  die im Akku und in X bewahrte neue
02B7  88                  DEY             Adresse wird in die Instruktion
02B8  8A                  TXA             eingesetzt
02B9  91 E4               STA (LAUADL),Y
02BB  60 EA               RTS             (ein vergessener Merker)

                  Unterprogramm BEGINN

02BD  A5 E0       BEGINN  LDA ANFADL      Setze laufende Adresse = Anfangsadresse
02BF  85 E4               STA LAUADL
02C1  A5 E1               LDA ANFADH
02C3  85 E5               STA LAUADH
02C5  60                  RTS

                  Unterprogramm VORAN

02C6  18          VORAN   CLC
02C7  A5 E4               LDA LAUADL      Addiere Instruktionslänge zur
02C9  65 E8               ADC LAENGE      laufenden Adresse
02CB  85 E4               STA LAUADL
02CD  A5 E5               LDA LAUADH
02CF  69 00               ADC # $ 00      ggfs. Obertrag zur high order
02D1  85 E5               STA LAUADH
02D3  C5 E3               CMP ENDADH      Ist die Endadresse erreicht?
02D5  30 04               BMI OUT4
02D7  A5 E4               LDA LAUADL
02D9  C5 E2               CMP ENDADL
02DB  60          OUT4    RTS             Statusprüfung im Aufrufprogramm

                  Unterprogramm RUECK

02DC  38          RUECK   SEC             Subtraktionsvorbereitung
02DD  A5 E2               LDA ENDADL      subtrahiere Instruktionslänge von
02DF  E5 E8               SBC LAENGE      Endadresse = neue Endadresse
02E1  85 E2               STA ENDADL
02E3  B0 02               BCS OUT5        ohne 'borrow'?
02E5  C6 E3               DEC ENDADH      nein: vermindere high order
02E7  60                  RTS
```

Unterprogramm LBEST

```
02E8  A0 00     LBEST   LDY #$00        Für indirekte Adressierung
02EA  B1 E4             LDA (LAUADL),Y  Hole Instruktionsbyte
02EC  48                PHA             auf den Stack
02ED  C8                INY             Angenommene Länge = 1 in Y
02EE  C9 00             CMP #$00        ein BRK?
02F0  F0 24             BEQ BEST        falls zutreffend ...
02F2  C9 40             CMP #$40        ein RTI?
02F4  F0 20             BEQ BEST
02F6  C9 60             CMP #$60        ein RTS?
02F8  F0 1C             BEQ BEST
02FA  A0 03             LDY #$03        Y = 3 als Länge für die nächsten Checks
02FC  C9 FF             CMP #$FF        FF als Assemblercode?
02FE  F0 16             BEQ BEST
0300  C9 20             CMP #$20        ein JSR?
0302  F0 12             BEQ BEST
0304  29 1F             AND #$1F        nur noch letzte 5 Bits prüfen
0306  C9 19             CMP #$19        Adressierungsart absolut, Y?
0308  F0 0C             BEQ BEST
030A  29 0F             AND #$0F        nur noch letzte 4 Bits
030C  AA                TAX             übertragen zur Tabellenindexierung
030D  BC 25 03          LDY LAENTB,X    Länge nach Tabelle
0310  29 03             AND #$03        Beschneidung auf die letzten 2 Bits
0312  49 03             EOR #$03
0314  F0 04             BEQ LENCOD      wenn ein LOP vorliegt
0316  68        BEST    PLA             Instruktionsbyte zurück
0317  84 E8     OUT2    STY LAENGE      Länge nun bekannt
0319  60                RTS
031A  68        LENCOD  PLA
031B  C9 FF             CMP #$FF        Sonderzeichen?
031D  F0 FA             BEQ OUT2        Zuweisung Länge 3
031F  4A 4A             LSR ,LSR        Längeninformation isolieren
0321  A8                TAY             und nach Y
0322  D0 F3             BEQ OUT2        branch always
0324  47                LOP             für 17 Bytes, einschl. LOP
0325  02 02 02  LAENTB   .BYTE
0328  01 02 02
032B  02 01
032D  01 02 01
0330  01 03 03
0333  03 03
0335  EA                NOP             Merker am Programmende
```

Benutzung der Zeropage

```
00E0    ANFADL
00E1    ANFADH
00E2    ENDADL
00E3    ENDADH
00E4    LAUADL
00E5    LAUADH
00E8    LAENGE
00EC    INCL
00ED    INCH
```

Commodore's PET-Computer hat viele Erwartungen geweckt. Erste ausführliche Anwenderberichte finden sich in:
H. Feichtinger 'PET, der Wunderknabe', FUNKSCHAU 12/1978, Seite 549/550;
Dan Fylstra 'User's Report: The PET 2001', BYTE 3/1978, Seiten 114-127;
Ralph Wells 'PET's First Report Card', KILOBAUD 5/1978, Seiten 22-30.

Weitere Hinweise zum Thema Verschieblichkeit/Relocation

RELOCATE, von Jim Butterfield, First Book of KIM, Argonne, Ill. 1977. Dienstprogramm, das vor allem zum Einfügen oder Herausschneiden von Instruktionen bei der Editierung geeignet ist und das dabei auch die Sprungweite bei Verzweigungen (branches) berücksichtigt. Tabellen innerhalb einer Instruktionsfolge können nicht automatisch übersprungen werden. Die Verschiebeparameter sind von Hand einzugeben. Nach Programmausführung sind die Dienste eines MOVE-Programmes erforderlich.

In Heft 1/78, Seite 60,und Heft 3/78, Seite 24, von KILOBAUD beschreibt Dr. Michael Wingfield HARDWARE PROGRAM RELOCATION. Der erste Teil geht auf die Lösung bei den großen Computern ein. Es folgt eine TTL-Schaltung, mit der ein Basisregister realisiert wird. Dieses Basisregister verändert die auf dem Adreßbus ankommenden Adressen additiv zu verschobenen Adressen. Der zweite Artikel geht auf dessen Installation an seinem 6800 ein, bringt Überlegungen zur Speicherbelegung und bringt die benötigten Hilfsprogramme.

PROGRAMME ZUM BANDSCHREIBEN UND BANDLADEN

Das erwähnte FIRST BOOK OF KIM dürfte unseren Lesern bekannt sein. Wenn nicht: Es wird von zahlreichen Elektronik-Händlern angeboten. Eine lohnende Lektüre.

Hinsichtlich des mit dem KIM-Monitor doch sehr langweiligen Umganges mit der Cassettenspeicherung sei besonders auf den dort enthaltenen HYPERTAPE von Jim Butterfield hingewiesen, der 1k RAM in 21 Sekunden zu schreiben und zu laden gestattet. Dieses Programm wurde in der Zeitschrift KILOBAUD, Heft 11/1977, Seite 66 nachgedruckt und mit sehr guten Kommentaren versehen. Nach den Erfahrungen des Herausgebers gibt es mit HYPERTAPE seit etwa einem Jahr fast keine Fehler mehr beim Schreiben und Laden, wenn am Cassettenrecorder die Tonblende auf hell gestellt ist und wenn fast volle Aussteuerung gefahren wird. Verwendet werden hier die billigsten Cassetten aus dem bekannten Fürther Versandhaus. Über das Alterungsverhalten dieser Cassetten kann noch nicht berichtet werden. Vielleicht berichten Leser einmal über ihre entsprechenden Erfahrungen.

Eine Verdoppelung der Lade- und Schreibgeschwindigkeit ist mit den Programmen SUPERLOAD und SUPERDUMP des amerikanischen Astronomen John Oliver möglich: etwa 12 Sekunden für 1k. Diese wurden in den KIM USER NOTES veröffentlicht, Heft 7/8, Seite 19. Dieses Journal wird herausgegeben von Eric C. Rehnke, 109 Centre Avenue, USA W. Norriton PA. 19401.

Der Abdruck von John Olivers Programmen erfolgte zunächst fehlerhaft. Berichtigung erfolgte in Heft 10/11, Seite 21. SUPERDUMP und SUPERLOAD gestatten es, mit beweglichen Eingabepuffern zu arbeiten, ähnlich wie im LAMID des Herausgebers. Sie können zudem als Unterprogramm aufgerufen werden, was kontinuierliches Arbeiten ermöglicht. Der Herausgeber wird sich bemühen, dieses Programm nachzudrucken, und zwar möglichst in einer Überarbeitung, die auch die Dienste der in diesem Heft gebotenen utilities erhält.

RELOCATE PROGRAMS WITH HEADER

E: Another possibility for variable LOADT & relocation using a header-table - to be erased later - to render parms.

Es folgt eine andere Möglichkeit, verschiebliches Laden und lauffähige Umrechnung zu bewirken: Das Programm erhält eine vorangestellte Tabelle, die u.a. die Plätze der umzustellenden high order-Adressen bezeichnet. Die low order Bytes sollten dabei auf einen X-indizierten Anfangspunkt bezogen sein. Das Dienstprogramm wertet die Tabelle aus und tilgt sie hernach durch Überschreiben mit dem nach vorne herangezogenen umgerechneten Hauptprogramm.

Es ist eine Schwäche dieser utility, daß man manuell eine Tabelle aufbauen muß, daß die Länge des Hauptprogramms auf eine Seite beschränkt sein muß und daß zunächst immer eine vergrößerte Länge durch den header entsteht.

Gleichwohl mag der Lösungsweg für andere Anwendungen interessant sein: Übergabe einer veränderlichen Zahl von Parametern nach feststehendem 'Protokoll' an ein anderes Programm, restloses Löschen der Parameterliste nach deren Verwertung, etwas MOVE, 'Einfangen' des sonst nicht zugänglichen Programmzählerstandes (PC) und schließlich die Anwendung im nachfolgenden Timerprogramm als Beispiel, das auch den header mit Erläuterungen enthält. Das Programm wurde vor längerer Zeit schon in Englisch kommentiert, was hierzulande nicht stören dürfte.

Manual setup: ID = FF to 17F9, RAM-destination wanted to SAL/SAH, 17F5/F6.

Zeropage used: EC = WORKA
ED = PNTR2L
EE = PNTR2H
FA = POINTL
FB = POINTH

START loading as usual in 1873.
Second START in first new location of header.

First part of utility is a subroutine jumped at from program header. Program has an error-exit with break if relocation surpasses the page limit found in location 17AE (04 for KIM-1 users).

Mit einer Art 'Indian Rope Trick' kann man einem Programm mitteilen, an welcher Stelle es sich im RAM befindet. Man ruft ein JSR wie im header des Timerprogrammes oder wie beim Einsatz des MRA und findet den Programmzähler auf dem Stack. Er zeigt auf das 3. Byte der JSR-Instruktion (s. Handbücher). Durch Addition oder Subtraktion kann man daraus jede benötigte Basisadresse gewinnen.

Adreßkonstante in Programmen, wie z.B. in RNDSCR oder im besprochenen SWEET16 stellen ein Hindernis für die Verschieblichkeit dar, wenn sie von außen eingefügt sind und nicht während des Programmablaufes erzeugt werden.

Der Herausgeber möchte die Leser hiermit anregen, der Erzeugung von veränderlichen Adreß-'Konstanten' aus dem Programm heraus nachzugehen und zu berichten. Dabei kann vorgenannter JSR-Trick nützlich sein. Schön wäre dann auch eine anschließende Umrechnungsmöglichkeit für Adreßtabellen, wie in SWEET16 enthalten.

RELOCATE PROGRAMS WITH HEADER

```
                    ; subroutine, get PC as of JSR-Call from stack, restore stack pointer
1780  68            ENTRY   PLA SAL +2      get PC from stack
1781  AA                    TAX SAL+2       save A in X
1782  68                    PLA SAH         get PC from stack
1783  48                    PHA             restore stack pointer SP first time
1784  85 FB                 STA POINTH      save SAH
1786  8A                    TXA SAL+2       fetch from stack
1787  48                    PHA             restore SP completely
1788  60                    RTS             return to pgm header

                    ; second entry. Adjust to SAL & SAH of JSR call
1789  38            ADJUST  SEC             set carry
178A  E9 02                 SBC #02         subtract 2 from SAL+2
178C  85 FA                 STA POINTL      save desired SAL of user pgm
178E  48                    PHA SAL         save on stack
178F  B0 02                 BCS             no page crossing? Skip next instruction
1791  C6 FB                 DEC POINTH      adjust SAH

                    ; initiate pointer for offset-address of main pgm block
1793  A0 06          INPTR   LDY #-$ 06      load relative position of offset N in pgm heade
1795  18                    CLC             clear carry
1796  71 FA                 ADC SAL         add offset N to SAL
1798  85 ED                 STA PNTR2L      store SAL+N to pointer
179A  AA                    TAX             A to X
179B  A5 FB                 LDA POINTH
179D  69 00                 ADC # $ 00      add carry if there was one in 1796
179F  85 EE                 STA PNTR2H      store ADH of main block to pointer
17A1  8D D0 17    INSTH     STA STORE+1     insert as ADH to instruction

                    ; check RAM boundary
17A4  C8            OFLCHK  INY             advance to relative position of M in header
17A5  18                    CLC             clear flag
17A6  8A                    TXA SAL+N       ADL of main block to A
17A7  71 FA                 ADC M           add length of main block
17A9  A9 00                 LDA # 00        clear A
17AB  65 EE                 ADC PNTR2H      add the carry of last addition, if there was o
17AD  C9 04                 CMP # 04        surpassing the page boarder of 04?
17AF  10 0B                 BPL             if yes: take next zeroes as a BRK opcode
                                            and stop in 17BC, showing 17BE

17B1  C8            INSX    INY             to relative loc.of X-Address in hdr
17B2  B1 FA                 LDA (POINTL),Y  Get X-address for main program
17B4  A8                    TAY             transfer to Y
17B5  68                    PLA             get back SAL
17B6  91 ED                 STA (PONIT2L),Y and insert into LDX-instruction

                    ; check for addressing mode
17B8  A0 09         CKMODE  LDY # $ 09      relative position of AM (adressing mode) in hdr
17BA  18                    CLC
17BB  A9 00                 LDA # 00
17BD  D1 FA                 CMP(POINTL),Y   if positive: Zero Page, X, other: Absolute, X
17BF  10 14                 BPL             skip modification

                    ; modify High addresses in main program block
17C1  C8            MODIFY  INY             advance to relative address of P
17C2  B1 FA                 LDA P           get P from header
17C4  85 EC                 STA WORKA       store P
```

```
17C6  C8          LOOP1   INY              advance in the table of addresses to be modified,
                                           starting in the first place and incrementing
17C7  B1 FA               LDA(POINTL),Y
17C9  8D CF 17    INSTL   STA STORE+1      insert the required adress high
17CC  A5 FB               LDA POINTH       get ADH of main pgm block
17CE  9D xx xx    STORE   STA              modify ADH's in main pgm block, adresses xx
                                           supplied from INSTL and INSTH
17D1  C6 EC               DEC WORKA        one off P
17D3  D0 F1               BNE LOOP1        table done?

                  ; relocate, erasing header
17D5  A0 07       RELOC   LDY # $ 07       load relative address of M in header
17D7  B1 FA               LDA(POINTL),Y    get M
17D9  AA                  TAX              set up X as the length counter
17DA  A0 00               LDY # $ 00       load with starting offset
17DC  B1 ED       LOOP2   LDA(PNTR2L),Y    get byte of main block
17DE  91 FA               STA(POINTL),Y    store byte, erasing header byte
17E0  C8                  INY              advance to next position
17E1  CA                  DEX              one off M
17E2  D0 F8               BNE LOOP2        relocation done?
17E4  4C 4F 1C            JMP KIM          control to KIM. Uff, here is the end of
                                           useable RAM.
```

13

On your controls you now will find the first address (SAL+SAH) of the relocated main program, ready for a fast "GO" or the entry of initial constants.

Catching and using the program counter (PC) from a JSR in this tape-loading case could have been done easier by taking 17F5 and 17F6 as a pointer. Nevertheless above approach may be useful in other circumstances since there is no direct access to PC. And it shows the exchange of parameters via stack and a header list from one program to another external program. ●

PROGRAM HEADER FOR RELOCATION

This header will be erased in LOOP2 of utility when main program block is moved upwards.

```
Rel.position Instruction Label   Mnemon    Comment
within hdr.

0000         20 80 17    START1  JSR ENTRY Put PC on stack and return
0003         4C 89 17            JMP ADJUST gives control to utility
0006         xx          N       .BYTE     Offset of main program block relative
                                           to START1 (hex count)
0007         xx          M       .BYTE     hex length of main pgm block, 1 page max.
0008         xx          XADDR   .BYTE     relative position of immediate operand
                                           to be loaded to X within main pgm block
0009         xx          AM      .BYTE     Addressing mode of main pgm. Insert
                                           00 for zeropage,X - 01 for absolute,X.
                                           End of header, if you inserted 00 here.

000A         xx          P       .BYTE     hex length of subsequent table of parms
000B         xx          TABLE   .BYTE     relative position of first ADH-byte
...          ...                 ...       to be modified. Position calculated
                                           relative to begin of main pgm block.
                                           Not so nice, do a lot of counting.
```

●

UNIVERSAL - TIMER

E: This interrupt driven timer has a header for relocation purposes. Timing between 52 microseconds up to 33 years. Intermediate results. X-indexed addressing and program modification are applied extensively.

Es handelt sich in mehrfacher Hinsicht um ein Demonstrationsprogramm. Einerseits bemühte sich der Verfasser um konsequente X-indizierte Adressierung, um zusammen mit der utility RELOCATE PROGRAMS WITH HEADER Verschieblichkeit zu gewährleisten. Dazu gehört die Anwendung des Vorprogrammes (header).

Zweitens handelt es sich bei MAINPR um ein Interruptprogramm, das also "im Hintergrund" laufen könnte (hier nicht realisiert, stattdessen eine Warteschleife). Der erste Teil ab START2 dient daher nur der Initialisierung mit den von Hand in die ersten 6 Zellen eingegebenen Parametern und dem Transport des Interrupt-Vektors.

Drittens schließlich wird Programm-Modifikation durch Veränderung der Befehlsadressen ausprobiert. Für den praktischen Einsatz wird man andere Programmiertechniken wählen. Weiterhin dürfte der richtige Einsatz des Timers nach diesem Beispiel durchsichtiger als in den Handbüchern sein (besonders wird auf die ausführlichere Darstellung im englischen KIM-1 User Manual hingewiesen).

Das Timing ist von etwa 52 Mikrosekunden bis 33 Jahre einstellbar. Jeder Interrupt erzeugt einen Impuls an PBØ. Weitere einstellbare Impulse als Zwischenergebnis werden an PB2, 3 und 4 erzeugt, wenn immer die Zählung in einem der Hochzählregister (in der zeropage) mit den entsp. Vorgaberegistern am Programmanfang übereinstimmt.

Achtung: PB7 am Applikationsstecker muß mit $\overline{IRQ}$ auf dem Expansionsstecker verbunden sein, um Interrupt zu ermöglichen.

```
rel.             HEADER
0000  20 18 17   START1  JSR ENTRY
0003  4C 89 17           JMP ADJUST
0006  30         N       .BYTE          Offset
0007  B0         M       .BYTE          Länge Hauptpr., hex
0008  07         XADDR   .BYTE          Immediate operand an Platz 07
0009  01         AM      .BYTE          Absolut,X-Adressierung
000A  17         P       .BYTE          Länge, hex, nachfolgender Tabelle
000B  25 27 2F   TABLE   .BYTE
000E  32 35 38
0011  3C 4E 56
0014  5F 64 67
0017  6A 71 76
001A  79 89 8C
001D  94 9A 9F
0020  A2 A8
```

Bei diesem Coding wird vorausgesetzt, daß der Header in Loc. 0200 beginnt, das Hauptprogramm zunächst um den Offset von 30 (hex) verschoben niedergelegt ist, also ab Loc. 023o. Das Hauptprogramm ist so geschrieben, daß sich seine Adressen auf den späteren Startpunkt 0200 beziehen.

UNIVERSAL-TIMER

```
0200  xx          TV       .KEYIN          Timervorgabe
0201  xx          TG       .KEYIN          Timergeschwindigkeit C,D,E oder F
0202  xx          ZZLL     .KEYIN          Zählziel, niedrigstes Byte
0203  xx          ZZLH     .KEYIN
0204  xx          ZZHL     .KEYIN
0205  xx          ZZHH     .KEYIN          höchstes Byte

0206  A2 xx       START2   LDX # $ xx      wird vom utility geladen. Hier steht
                                           z.B. 00, wenn das Programm in
                                           0200 oder 0300 beginnt.
0208  A9 3D                LDA # $ 3D      PBØ, 2, 3, 4 und 5 als Ausgabe
020A  8D 03 17             STA PBDD        initialisieren
020D  A9 00                LDA # $ 00      Ausgaberegister auf 0
020F  8D 02 17             STA PBD
0212  A0 04                LDY # $ 04      Zählvorgabe für Löschen
0214  99 EA 00   LOOP1     STA ZRLL-1,Y    der Hochzählregister
0217  88                   DEY
0218  DØ FA                BNE LOOP1

021A  18                   CLC
021B  8A                   TXA             lade die Basis, low order
021C  69 44                ADC # $ 44      das ist die ADL des MAINPR
021E  8D FE 17             STA IRQL        als Interruptvektor
0221  90 03                BCC ADHIRQ      kein Überlauf
0223  FE 27 02             INC ADHIRQ+1,X
0226  A9 xx      ADHIRQ   LDA # $ xx      Der benötigte Wert ist vom
                                           Dienstprogramm hier eingetragen
                                           worden: Tabelleneintrag mit '27'
0228  8D FF 17             STA IRQH        als Interrupt-Vektor
02BB  A9 FE                LDA # $ FE      Initialisieren der Warteschleife
022D  9D 41 02             STA LOOP2+1,X
0230  9D 43 02             STA LOOP2+3,X
0233  BD 01 02             LDA TG,X        Hole Timergeschwindigkeit
0236  9D A4 02             STA FRESHT,X    setze als Parameter ein
0239  A8                   TAY             nimm ihn auch für Indizierung
023A  BD 00 02             LDA TV,X        Hole die Timervorgabe
023D  99 00 17             STA PAD,Y       und bringe sie in den Timer an
                                           Zellen 170C,ØD, 0E oder 0F.
0240  D0 FE      LOOP2    BNE LOOP2
0242  F0 FE                BEQ LOOP2+2     Warteschleife mit doppeltem Boden.
                                           Eleganter ist ein Coding  4C 40 02,
                                           das allerdings nicht indiziert
                                           werden kann.
```

An dieser Stelle ist das Initialisierungsprogramm beendet, der Timer ist gestartet und wird sich gelegentlich mit Interrupt bemerkbar machen, er arbeitet im 'background'. Im Vordergrund könnte jetzt zu weiteren Routinen gesprungen werden.

INTERRUPTROUTINE

```
0244  EE 02 17   MAINPR   INC PBD          bei jedem Interrupt wird zunächst
0247  CE 02 17            DEC PBD          ein Impuls an PBØ erzeugt.
024A  A9 EF                LDA # $ EF      initialisiere die benötigte
024C  9D 58 02             STA INCINS+1    zeropage-Adresse im Befehl
```

```
024F  A0 04          LDY #$04      Zählervorgabe
0251  88       LOOP3 DEY           die Abfage steht hier am Anfang
0252  30 07         BMI VERGL
0254  DE 58 02       DEC INCINS+1,X Vermindere Registeradresse im
                                    Folgebefehl
0257  E6 EF    INCINS INC ZRLL+1   beginnend mit ZRLL werden die
                                    Hochzählregister erhöht, bei
0259  F0 F6         BEQ LOOP3      Übertrag jeweils das Folgeregister.

025B  A9 EE    VERGL  LDA #$EE    initialisiere die Adressierung
025D  9D 6E 02        STA LOOP4+1,X
0260  A9 02           LDA #$02     dito
0262  9D 70 02        STA VGL+1,X
0265  9D 7B 02        STA LAD+1,X
0268  9D 83 02        STA IMPU+1,X
026B  A0 04           LDY #$04     Zählervorgabe
026D  A5 xx    LOOP4  LDA ZR..     Zählregisteradresse von oben
026F  DD xx 02 VGL    CMP ZZ..     eingesetzt, dito Vergleichsreg.
0272  D0 2C           BNE GOBACK
0274  1E 7B 02        ASL LAD+1,X  Bit für Impulserzeugung setzen
0277  1E 83 02        ASL IMPU+1,X
027A  A9 02    LAD    LDA #$02     o2 als Anfangswert, wird mit ASL
027C  0D 02 17        ORA PBD      nach links verschoben
027F  8D 02 17        STA PBD      aufsteigende Impulsflanke erzeugen
0282  49 02    IMPU   EOR #$02     02 ebenfalls als Anfangswert, ASL
0284  8D 02 17        STA PBD      abfallende Flanke am Port
0287  DE 6E 02        DEC LOOP4+1,X Adresse dekrementieren
028A  FE 70 02        INC VGL+1    diese erhöhen
028D  88              DEY
028E  D0 DD           BNE LOOP4    Schleifenzähler = 0?
0290  A9 63    END1   LDA #$63     Neues Verzweigungsziel zum Verlassen
0292  9D 43 02        STA LOOP2+3,X der Warteschleife nach END2
0295  18              CLC
0296  69 02           ADC #$02     dito, um 2 erhöht
0298  9D 41 02        STA LOOP2+1,X
029B  A9 07           LDA #$07     Der Timer wird immer weiter
                                   Interrupts erzeugen, wenn man ihn
                                   nicht durch einen solchen Schreib-
029D  9D A4 02        STA FRESHT+1,X vorgang abstellt
02A0  BD 00 02 GOBACK LDA TV,X     Hole Timervorgabe
02A3  8D xx 17 FRESHT STA TIMER    Auffrischen bzw. Abstellen des
                                   Timers mit unterschiedlicher
02A6  40              RTI          Adressierung. Interruptende.

02A7  A9 xx           LDA #$xx     Operand wird von utility eingesetzt.
02A9  85 FB           STA POINTH   ADH des Programmanfangs
029B  86 FA           STX POINTL   dito ADL
029D  4C 4F 1C        JMP KIM      Auf dem Display erscheint die
                                   Adresse von TV für eine eventuelle
                                   neue Timervorgabe.
```

Das Studium dieses lauffähigen Programmes zeigt, daß mit nur einem Indexregister schlecht zu arbeiten ist, wenn es auf eine veränderliche Adressierung ankommt. Das Initialisieren und Verändern von Adressen in Befehlen ist zu umständlich. RALOAD ist zweifellos die bessere Lösung für die Verschieblichkeit.

Zeropage used: EB = ZRHH, EC = ZRHL, ED = ZRLH, EE = ZRLL.

DISPLAY BLINK UND ROLLROUTINEN

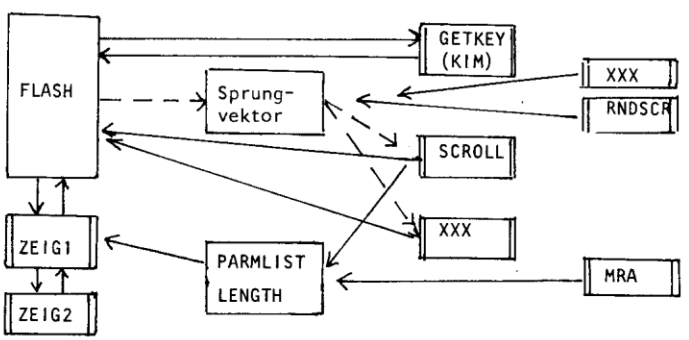
E: A cobweb of routines enables all kinds of display that LEDs can do: Continuous or blinking, scrolling output of any 7-segment character font, for results, plays and interactive job control. Of special interest is MRA, manipulating stack and rendering up to 64k of parameters to any program.

Es folgt eine Reihe von Display-Routinen für beständige, rollende oder blinkende Datenausgabe an den 6 Ziffernanzeigen des KIM. Jeder einzelne Leuchtbalken kann angesteuert werden. Auf diese Weise wird die Darstellung eines primitiven Alphabets möglich.

Routinen dieser Art sind nicht grundsätzlich neu. Im 'First Book of KIM' sind ähnliche Funktionen realisiert, allerdings immer gerade nur in der Form, wie sie für ein einzelnes Programm benötigt werden und leider auch nicht als Unterprogramme herausziehbar. Hier ist nun eine Erweiterung und Systematisierung vorgenommen, und zwar durch das Angebot, die verschiedenen Routinen je nach Bedarf miteinander zu kombinieren, sie korrespondieren miteinander. Die Abstimmung ist in der 4k-Erweiterung des Betriebssystems beim Herausgeber noch weiter geführt, als in diesem ersten Heft von 65xx MICRO MAG dargestellt werden kann.

Das System ist so aufgebaut, daß seine Bestandteile für Spiele, Disassembler, Fragestellungen etc. nach Wahl aufgerufen werden können.

FLASH ist ein Master, eine beherrschende Routine, wie folgendes Schema andeutet:



Hilfsroutinen,wie RNDSCR bringen je nach Bedarf indirekte Sprungvektoren in die Zellen 00E9/EA, die von FLASH aus angesprungen werden. Der Anwender kann hier eigene Applikationen programmieren. Er könnte z.B. die Zahl der indirekten Sprünge nach E9/EA zählen und nach Erreichen eines Ziels weiterschalten, mit der Wirkung, daß die Anzeige nur für eine gewisse Zeit blinkt usw.

MRA manipuliert den Stack um die Länge einer beliebig langen (bis 64k !) Parameterliste, die mitten in den Instruktionscodes steht, die also nicht an eine Übergabestelle gebracht werden muß. MRA legt die Anfangsadresse dieser Liste in der Zeropage EB/EC und ihre hexadezimale Länge in 00E7/E8 nieder, von wo aus ZEIG1 eine Anzeige ermöglicht.

FLASH

```
0CCD E6 EE      ENTRY   INC MEMO        Hochzählzelle für Verzögerung
0CCF A5 ED              LDA SPEED       Geschwindigkeitsvorgabe
0CD1 8D 47 17           STA CLKKT       in den Timer laden, diesen anstoßen
0CD4 D8         FLASH1  CLD
0CD5 20 6A 1F           JSR GETKEY      eine Taste gedrückt? Zeichen in A
0CD8 6C E9 00           JMP JINDL       indirekter Sprung zur Verzweigungsstelle
0CDB A5 EE      RENTRY  LDA MEMO        Wiedereintrittspunkt, Prüfen des MEMO
0CDD 29 03              AND #$03        Wert beeinflußt Blinkgeschwindigkeit
0CDF F0 03              BEQ FLASH2      wechselweise überspringen
0CE1 20 A7 0C           JSR ZEIG1       Anzeigen leuchten lassen
0CE4 2C 47 17   FLASH2  BIT CLKKT       Timer abgelaufen?
0CE7 30 E4              BMI ENTRY       wenn ja, von Anfang an
0CE9 10 E9              BPL FLASH1      wenn nein, gleiche Verzögerungs-
                        (NOP)           schleife

Zeropage used:
00EE                    MEMO
00ED                    SPEED           externe Vorgabe notwendig
00E9                    JINDL           indirektes Sprungziel, Verzweigungs-
00EA                    JINDH           stelle nach externer Vorgabe.
```

Um diese Routine auszuprobieren, geben wir die Adresse von RENTRY manuell nach JINDL/JINDH, also DB und 0C nach 00E9/EA. Niedrige Werte in SPEED bewirken ein schnelles Blinken. Voraussetzung für die Inbetriebnahme ist natürlich, daß die nachfolgenden Unterprogramme ZEIG1 und ZEIG2 installiert sind.

ZEIG2 Unterprogramm für ZEIG1

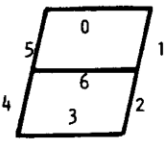
```
0C9E A0 00      ZEIG2   LDY #$00        als Konstante und f. indir. Adressierung
0CA0 8C 41 17           STY PADD        LEDs ausstellen
0CA3 8C 42 17           STY SBD
0CA6 60                 RTS
```

ZEIG1 Unterprogramm für LED-Anzeige, ähnlich SCANDS (KIM) aber ohne Codewandlung

```
0CA7 20 9E 0C   ZEIG1   JSR ZEIG 2       Lampen aus, Y = 0 bei Rückkehr
0CAA A9 7F              LDA #$7F        Aktivierung des PADD für 7 Segmente
0CAC 8D 41 17           STA PADD        als Ausgang
0CAF A2 09              LDX #$09        initalisiere f. 1. Anzeige
0CB1 8E 42 17   LOOP1   STX SBD         und aktiviere Ausgang
0CB4 B1 EB              LDA (PARML),Y   hole erstes Zeichen lt. Parameterliste
0CB6 8D 40 17           STA SAD         und bringe Segmente zum Leuchten
0CB9 98                 TYA             rette Y auf Stack
0CBA 48                 PHA
0CBB A0 7F              LDY #$7F        setze eine Verzögerung, ca. 500 cycles
0CBD 88         LOOP2   DEY             vermindere
0CBE D0 FD              BNE LOOP2       und warte, bis Y = 0
0CC0 68                 PLA             Y nun zurückholen
0CC1 A8                 TAY
0CC2 E8 E8              INX,INX         Adresse des nächsten Display einstellen
0CC4 C8                 INY             und des nächsten Parameters
0CC5 C0 06              CPY #$06        bei 06 waren alle LEDs dran
0CC7 D0 E8              BNE LOOP1       sonst nächste LED
0CC9 20 9E 0C           JSR ZEIG2       Lampen aus
0CCC 60                 RTS

Zeropage used:
00EB                    PARML           indirekte Adresse der Parameterliste
00EC                    PARMH
```

Im Gegensatz zu der KIM-Routine SCANDS nimmt ZEIG1 keine Umwandlung der im Speicher gefundenen Binärwerte mit der bekannten TABLE HEX TO 7 SEGMENT in Zellen 1FE7 ff. vor. Jedes Bit der Parameter steuert also ein Segment der LEDs direkt an. Die Zuordnung ist dabei wie folgt:



Bit Nr. ...

Die Bits im Byte:
7 6 5 4 3 2 1 0

Die Besetzung der Bits in den Parametern macht es also möglich, andere als nur hexadezimale Zeichen mit den LEDs darzustellen. Neben Balkendarstellungen für Spiele gibt es auch ein primitives Alphabet:

dunkel = 00	I = 86	q = E7	Z = C9	6 = FD
A = F7	J = 9E	r = 50	? = 53	7 = 87
b = FC	k = 75	S = ED	- = 40	8 = FF
C = B9	L = 38	t = 78	0 = 3F	9 = EF
c = 58	M = B7	U = BE	1 = 86	
d = 5E	n = D4	v = EA	2 = DB	
E = F9	O = 3F	$\overline{w}$ = 9C	3 = CF	
G = FD	o = DC	x = 94	4 = E6	
H = F6	P = F3	Y = EE	5 = ED	

Einige Buchstaben stellen sicher nur Kompromisse dar, wie z.B. v und Z oder k. Für viele Zwecke der Kommunikation mit dem Rechner reicht die Darstellung jedoch. Weil es sich um 7-Segment-Anzeigen handelt, kann man in vorstehender Tabelle natürlich immer das vorderste Bit weglassen. F7 ist also gleich mit 77.

Will man z.B. Rechenergebnisse zusammen mit solchen "Sonderzeichen" darstellen, so empfiehlt es sich, diese Werte zunächst mit der Tabelle HEX TO 7 SEGMENT in das 'Displayformat' umzuwandeln und in die Parameter zu bringen.

S C R O L L

Dieses Unterprogramm gehört ebenfalls in die Familie der Display-Routinen. Es rollt den Text von rechts nach links über die 6 LEDs. Die Geschwindigkeit hängt vom Wert in 00E6 ab. Der Einsprung nach ENTRY erfolgt von FLASH mit indirektem Sprungziel, das man entweder manuell einsetzt oder durch eine kleine Hilfsroutine, die auch noch beschrieben wird. SCROLL kann nacheinander bis zu 64k Speicherzellen durchrastern. - Die noch nicht besetzten Speicherplätze zwischen den bisher genannten Programmen werden hernach durch weitere 'utilities' besetzt.

SCROLL - Unterprogramm

```
0CFD  C6 E6   ENTRY    DEC SCRSPD     Ist das letzte Zeichen erledigt?
0CFF  F0 03            BEQ ROTATE     überspringe, wenn ja
0D01  4C DB 0C         JMP RENTRY     Bei nein: zurück nach FLASH
```

```
0D04  E6 E4    ROTATE   INC SCRCL     Hochzählen des Scrollcount low
0D06  D0 02             BNE VGL       kein Oberlauf?
0D08  E6 E5             INC SCRCH      erhöhe bei Oberlauf
0D0A  A5 E7    VGL      LDA LENGL     vergleiche nun mit Zielregister,
0D0C  C5 E4             CMP SCRCL     ob Zählung komplett
0D0E  B0 07             BCS GOFL
0D10  A5 E8             LDA LENGH
0D12  C5 E5             CMP SCRCH
0D14  90 01             BCC GOFL      wenn nicht, weiter nach FLASH
0D16  60                RTS           Rückkehr in das Programm, das FLASH
                                      als Unterprogramm aufrief.
0D17  E6 EB    GOFL     INC PARML     Erhöhen der Pointeradresse
0D19  D0 02             BNE GOFL1     wenn kein Oberlauf;
0D1B  E6 EC             INC PARMH     erhöhe high order
0D1D  4C CD 0C GOFL1    JMP FLASH     rolle weiter.

Zeropage used
00EB          PARML                   Adreßpointer
00EC          PARMH
00E4          SCRCL                   scrollcount low
00E5          SCRCH                             high
00E6          SCRSPD                  scrolling speed
00E7          LENGL                   Zahl der zu rollenden Zeichen, low
00E8          LENGH                   dito, high
```

Durch manuelles Einsetzen der Parameter in die bisher genannten Zellen der Zeropage, wird man jetzt Texte rollen können. Experimentieren mit den Geschwindigkeiten ist sehr reizvoll, und Ihre Frau wird staunen, wenn KIM sie mit Namen anredet. FLASH natürlich als Unterprogramm aufrufen.

Das nachfolgende Unterprogramm RNDSCR initialisiert Zähler, Geschwindigkeit und indirekte Sprungadresse für SCROLL bzw. FLASH. Die Rollgeschwindigkeit kann verändert werden, sobald das Unterprogramm einmal durchlaufen ist.

```
                RNDSCR - Initialisiere Parameter für SCROLL

0D20  A9 FF    ENTRY    LDA # $ FF    Einsetzen der langsamsten
0D22  85 E6             STA SCRSPD    Rollgeschwindigkeit
0D24  A9 06             LDA # $ 06    6 Anzeigen sind anzusteuern
0D26  85 E4             STA SCRCL
0D28  A9 00             LDA # $ 00    Zähler auf 0
0D2A  85 E5             STA SCRCH
0D2C  A9 FD             LDA # $ FD    Sprungvektor laden. Achtung bei
                                       Programmverschiebung! Dann neue
                                       Adresse von SCROLL einsetzen.
0D2E  85 E9             STA JINDL
0D30  A9 0C             LDA # $ 0C
0D32  85 EA             STA JINDH
0D34  60                RTS
```

Zeropage used: 00E4, 00E5, 00E6, ooE9, 00EA wie in FLASH UND SCROLL. ●

M R A - MODIFY RETURN ADDRESS AFTER JSR

by Length of Parameter List (submitted thereafter), Subroutine.

Stackmanipulation um die Länge einer Parameterliste.

E: A Call JSR MRA followed by 'FF' and XX XX = LENGTHL/LENGTHH (hex) respectively will manipulate the return address on stack to: original value + (LENGHTHL/LENGTHH) + 3. The '+3' accounts for the three interpreter bytes. By this up to 64k parameter bytes may follow a Call before normal program coding is resumed. The length information is preserved in zeropage E7/E8, the address of the first parameter byte is put to EB/EC = PARML/PARMH, ready for pointer addressing. No damage is done, if 'FF' is missing: After RTS PC will return as usual.

Dieses Dienst-Unterprogramm bringt eine gewisse Automatik in die Parameterübergabe an andere Programme. Auf den Aufruf JSR MRA folgt ein Markierbyte 'FF' und dann zwei Bytes mit einer hexadezimalen Längenangabe für die Zahl der Parameterbytes. Diese drei Vorspannbytes werden dabei nicht mitgezählt. Dahinter dürfen nun bis zu 64k Parameterbytes folgen, ehe die nächsten Instruktionen folgen. M.a.W.: Mitten im Programm dürfen irgendwelche Werte stehen. MRA berechnet die Rückkehradresse so, daß diese Werte übersprungen werden. Es legt ferner die Längenangabe in LENGL/LENGH in E7/E8 in der zeropage nieder und schreibt außerdem die Adresse des ersten Parameterbytes in den Pointer PARML/PARMH in EB/EC.

An diesen definierten Zeropagestellen steht die Information für andere Programme bereit, so z.B. für SCROLL. Es geht also um Parameterübergabe nach einem feststehenden Protokoll.

Natürlich gibt es auch andere Verfahren der Parameterübergabe; wir werden weitere kennenlernen. In Handbüchern wird u.a. vorgeschlagen, im Aufrufprogramm die genaue Zahl der benötigten Werte mit PHA auf den Stack abzulegen, von wo das aufgerufene Programm sich zu gegebener Zeit mit PLA die Werte zurückholt. Diese Methode ist angängig, wenn lineare Programme miteinander korrespondieren, sobald Unterprogramme eingesetzt werden (was ja immer erstrebenswert ist), versperren die Rückkehradressen den Zugang zu den Parametern auf dem Stack, dessen Verwaltung und Manipulation schwierig wird. MRA bringt hier eine bequeme Automatik, die sich gar nicht daran stört, wenn weniger Parameter später abgerufen werden, als zunächst übergeben worden sind. Sicher braucht MRA etwas mehr Zeit, als wenn man eine Handvoll Werte an definierte Stellen des Speichers bringt - was ja auch eine Möglichkeit der Parameterübergabe ist.

MRA ist schon 'etwas Interpreter': Wenn dem Aufruf 'FF' folgt, leistet es alle Dienste, es berechnet die Rückkehradresse neu und legt die benötigten Werte in der Zeropage ab. Fehlt das 'FF', dann nimmt es an, nichts solle geschehen, es legt allerdings den Programmzähler so ab, wie er bei Aufruf des Unterprogrammes bestand. Vielleicht ergeben sich hier weitere Anwendungen.

Wie man diese Routine einsetzt, wird am Beispiel 'ERROR BLINKING' gezeigt. An dieser Stelle sei schon darauf hingewiesen, daß MRA selbst voll verschieblich ist. Dagegen werden die Aufrufprogramme mit ihrem 'FF' und der Längenangabe von RALOAD nicht 'verdaut', weil dort FF die Länge 3 zugewiesen wird. Natürlich kann man das umarbeiten.

MRA - Unterprogramm

```
0C66  BA        MRA     TSX             Stackpointer nach X

0C67  BD 01 01          LDA 0101,X      Hole PCL vom Stack
0C6A  85 EB             STA PARML       vorläufige Ablage
0C6C  BD 02 01          LDA 0102,X      hole PCH vom Stack
0C6F  85 EC             STA PARMH       vorläufige Ablage
0C71  A0 01             LDY # $ 01      für indirekte Adressierung
0C73  B1 EB             LDA (PARML),Y   hole das Byte, das dem JSR-
                                        Aufruf folgt
0C75  C9 FF             CMP # $ FF      ist es ein FF-Markierbyte?
0C77  F0 01             BEQ DOIT        wenn ja: Weiterrechnung
0C79  60                RTS             wenn nein.

0C7A  C8        DOIT    INY             für nächste Adressierung
0C7B  B1 EB             LDA (PARML),Y   hole Längenangabe low
0C7D  85 E7             STA LENGL       Parameter ablegen
0C7F  D8                CLD             Additionsvorbereitung
0C80  18                CLC
0C81  69 03             ADC # $ 03      Berücksichtigung der 3 Funktionsbytes
0C83  65 EB             ADC PARML       ergibt PCL für das RTS
0C85  9D 01 01          STA 0101,X      Ablage auf dem Stack
0C88  C8                INY             für nächste Adressierung
0C89  B1 EB             LDA (PARML),Y   hole Länge high aus Aufrufprogramm
0C8B  85 E8             STA LENGH       und lege den Parameter ab
0C8D  65 EC             ADC PARMH       plus alter PCH plus mögl. Carry
0C8F  9D 02 01          STA 0102        PCH neu auf Stack

0C92  18                CLC             den Pointer auf das erste
0C93  A5 EB             LDA PARML       Parameterbyte einstellen, d.h.
0C95  69 04             ADC # $ 04      +3 +1
0C97  85 EB             STA PARML
0C99  90 02             BCC OUT         kein Überlauf?
0C9B  E6 EC             INC PARMH
0C9D  60        OUT     RTS
```

Die Anwendung vorgenannter Routinen für blinkende Anzeige gibt nachfolgender Programmabschnitt:

```
                ERROR BLINKING

0CEB 20 xx xx           JSR xx          beliebige Hilfsroutine, die die Adresse
                                        von RENTRY in FLASH, nämlich 0CDB in
0CEE 20 66 0C           JSR MRA         den Pointer JINDL/JINDH, 00E9/EA stellt.

0CF1 FF                 .BYTE           Markierbyte zur Interpretation
0CF2 06                 .BYTE           hexadezimale Länge, low
0CF3 00                 .BYTE           dito, high
0CF4 F9 50 50           .BYTE           6 Parameter f. Anzeige 'Error '
0CF7 DC 50 00           .BYTE
0CFA 4C CD 0C           JMP FLASH
```

Zum Zwecke des Blinkens 'gerader' Einstieg in FLASH mit JMP. Zum Rollen Übergabe von mehr als 6 Parametern mit Aufruf JSR FLASH, mit einer Hilfsroutine vorher den Adressvektor von SCROLL nach JINDL/JINDH bringen. Muster:

JSR RNDSCR, JSR MRA, FF, LENGL, LENGH, PARAMETER, JSR FLASH, JMP to ...

Zur Verschieblichkeit vorstehender Display-Routinen: Wegen der Verwobenheit der Teile sollte man den von 0C66 bis 0D34 besetzten Block als eine Einheit ansehen und eine gemeinsame Verschiebung mit RALOAD machen. Dabei wird man in 0D35 ein 'EA' anfügen und die Adreßkonstante in RNDSCR von Hand ändern.

T A S T A T U R F R A G E N

Viele Benutzer von KIM beklagen sich über das Prellen der Kontakte an der Tastatur. In den KIM USER NOTES, Heft 10/11 auf Seite 19 wurde die wohl wichtigste Ursache angesprochen: Die im Schaltplan mit KB Row 1 bezeichnete Leiterbahn muß auf der Platinenführung viele unzuverlässige Kontakte an den Kontaktscheibchen überwinden. Abhilfe: Man lötet direkte Drähte von der rechts von Taste B ankommenden Leiterbahn zu den Eingängen der Tasten A, 9 und C. Keyboard vorsichtig an der Rückseite lösen, 3 unterschiedlich lange Schrauben.

Weiterhin besteht die Möglichkeit, eine zweite Tastatur über den Applikationsstecker parallel anzuschließen. Diese kann man bequem zugänglich und mit Gummifüßen versehen auf den Arbeitstisch placieren. In diesem Falle kann man KIM mit seiner Anzeige z.B. senkrecht in ein Gestell bringen.

Nach verschiedenen Versuchen hat sich die bei Völkner in Braunschweig beziehbare DIGITAST für eine solche Tastatur als sehr geeignet erwiesen, arbeitet sie doch praktisch prellfrei. Die billigere DIGITAST-mini konnte nicht entsprechend überzeugen. Vorteil auch eines deutlichen Knackgeräusches am Druckpunkt. Die Stifte an der Taste entsprechen dem 0,1 Zoll-Raster der Leiterplatten.

Den Aufbau eines Keyboards mit 64 Tasten für den KIM beschreibt Hal Chamberlin in KILOBAUD, Heft 1/1978, Seite 98. Als einziges IC benötigt er den TTL-Dekoder 74154, der Rest ist Software in den Adressen 0200 bis 035C. Erzeugt wird ASCII-Code, die Kontakte sind softwaremäßig entprellt. Dieser Artikel - wie ja auch das KIM Monitorprogramm - ist ein typisches Beispiel, wie man bei Einsatz eines Mikroprozessors Hardware durch Software ersetzt.

Anwenderbericht:

Am Max-Planck-Institut für Limnologie in Plön betreibt Herr Dr. H.-J. Krambeck mehrere KIM mit AD-Wandlern in der Gewässerforschung. Daten werden auf Cassetten gespeichert, die im Abstand von einigen Tagen ausgetauscht werden.

Zur Programmentwicklung wurde ein Crossassembler in ALGOL für PDP8 geschrieben. Über ein Interface wird der Objektcode aus dieser Maschine direkt in den KIM geladen. Anschrift über den Herausgeber.

Das Kontrastprogramm.

SWEET16: The 6502 Dream Machine.

Von Stephen Wozniak. In BYTE, Heft 11/1977, Seite 150 ff.

Der Schöpfer des BASIC für den Apple II-Computer gibt eine ausführliche Beschreibung und Listung seines Programmes SWEET16. Es handelt sich um einen Interpreter, der aus jedem 65xx-System einen in 16 Bit Breite arbeitenden 'Metaprozessor' macht. Zu SWEET16 gehören 32 neue Anweisungen, deren Ausführung mit einer geschickt aufgebauten Verzweigungstabelle für die Adressen der benötigten Unterprogramme gesteuert wird. Die zusätzlichen Anweisungen sind vorwiegend nur 1 Byte lang. Zusätzlich zu den bekannten Maschinenregistern baut er in der Zeropage 16 Stück 2-byte-breite Register auf, und zwar in den Zellen 0000 bis 001F. Das hat den Vorteil der kurzen Befehlscodes in seinem Interpreter und der schnelleren Programmausführung. Von den Registern dient das erste als 'Akku', über den alle Rechenoperationen laufen, eines als Stack Pointer für die innerhalb SWEET16 möglichen geschachtelten Unterprogrammaufrufe und eines als Programmzähler. Weitere Register dienen der Aufnahme von Vergleichsergebnissen und des Status für das letzte Ergbnis. Die übrigen 11 Register sind frei für den Anwender.

Der Aufruf der '16-Bit-Maschine' erfolgt so:

```
    CLD
    JMP SWEET16
    ...          (Instruktionsfolge)
    00           Rückkehr zur 65xx-Programmierung
```

Die möglichen Anweisungen umfassen Setzen, Laden und Speichern von Registern, Register erhöhen oder dekrementieren, Addition, Subtraktion und Vergleich. Bei den Nicht-Registeroperationen gibt es 8 Verzweigungsmöglichkeiten in Abhängigkeit von Ergebnissen, einen unbedingten Sprung, Eintritt ins Unterprogramm und Rückkehr von dort.

Die Anweisungen erfassen entweder 1 oder 2 Byte, haben direkte oder indirekte Adressierung und auch eine POP-Automatik: Entweder wird indirekt über die angesprochene Registeradresse geladen oder gespeichert, ehe diese um 1 verändert wird oder die Veränderung erfolgt zunächst und erst danachder Datentransport.

SWEET16 ist die Arbeit eines Könners auf seiner Maschine. Es lohnt sich, das Programm nachzuvollziehen, zumal es sparsames Coding und merkenswerte Tricks enthält.

Bei der Aufbereitung des Programm-Listings für den Artikel haben sich leider Fehler eingeschlichen, nachdem der Programmanfang so rund auf 0800 gelegt wurde. SWEET16 arbeitet nur dann, wenn alle mit der Tabelle angesprungenen Unterprogramme in einer Seite (page) anfangen, also z.B. in page 09.

Diese Forderung wird erfüllt, wenn man das Programm bei etwa 0887 anfangen läßt, einige absolute Adressen entsprechend ändert und vor allem auch die Tabelle der niedrigen Sprungadressen.Programmzeile 00016 und 00028 sind auf die Nummer der verwendeten page zu ändern. Ein typisches Problem der Verschiebung bei Adreßkonstanten.

65xx MICRO MAG wird Ihnen in einem späteren Heft zeigen, wie man mit Makros wahlweise bis zur Breite einer page arbeiten kann.

H E A D H U N T E R

Etikettensuche beim Laden von der Bandcassette

E:Programs or data on cassette may be preceeded by an identifying header label of any length (not only one ID-byte). This utility either loads record - after full identity of label to contents of a table has been checked - or discards it, then searching continuously next labels. Can be used to check tape dump against memory contents.

Für die Identifizierung von auf Magnetband gespeicherter Information steht im KIM-Betriebssystem nur das 1-byte-lange ID in Zelle 17F9 zur Verfügung. Eine Beschränkung auf 256 Möglichkeiten. Wie aber, wenn man einen Datensatz (oder ein Programm) unter vielen hunderten finden will, der eigentlich "Otto Meyer" heißt, im Gegensatz z.B. zu "Otto Mair"? Oder wenn man alle Datensätze finden will, die mit "Otto " beginnen? Bisher war hier viel Bedienung des Systems von Hand notwendig. HEADHUNTER macht das im Dauerlauf.

Einem Programm oder einem Datensatz kann man vor der Abspeicherung ein beliebig langes Etikett voranstellen (sog. header label), das eine eindeutige Identifizierung erlaubt. Dieses wird zusammen mit den anderen Daten/Programmabschnitten wie üblich auf Band abgespeichert.

Vor dem Laden schreibt man eine Tabelle mit dem Suchbegriff in beliebige Zellen des RAM und hängt das Sonderzeichen FF als Abschluß daran. Das Programm geht die Tabelle durch, bis es ein solches FF findet. Damit darf also die Tabellenlänge gegenüber den Etiketten der Bandsätze verkürzt sein (verkürzter Suchbegriff).

Die ersten 4 Befehle des Programmes stellen die Adresse der Tabelle in einen Pointer. Der Rest ist einfach.

Ein Hinweis: Bei dieser Aufschreibung wird das Etikett nicht wieder in das RAM geladen.Bei Identität entspricht das erste geladene Byte in seiner relativen Stellung dem FF in der Tabelle, man sollte solche Wirkungen im Voraus bedenken. Es besteht keine Schwierigkeit, auch das Etikett beim Laden in den Speicher zu übernehmen oder die Etikettenlänge zu normieren. Bei verkürztem Suchbegriff will man den nachfolgenden Datenteil sicher auch an definierte Stellen bringen - eine Frage der in VEB+1,2 einzustellenden Adresse.

Mit HEADHUNTER können sie natürlich auch prüfen, ob eine Abspeicherung auf Cassette dem Inhalt des Speichers entspricht. Dabei wird die gesamte Abspeicherung als Suchbegriff (Tabelle) angesehen. Die Adresse des Programmanfanges wird mit den ersten 4 Befehlen in den Pointer geladen, die Prüfung auf FF in Adressen 179B/9C ist zu unterbinden. (NOP)

HEADHUNTER

```
1780  A9 xx      START   LDA # $ XX      Anfangsadresse der Tabelle in den
1782  85 FA              STA POINTL      Pointer laden
1784  A9 xx              LDA # $ XX
1786  85 FB              STA POINTH
1788  A9 60              LDA # $ 60      RTS-opcode für VEB
178A  20 75 18           JSR LOADT+2     Laderoutine im Monitor
178D  20 EA 19   BACK    JSR INCVEB      erhöhe Ladeadresse
1790  A0 00              LDY # $ 00      für indirekte Adressierung
1792  D1 FA              CMP (POINTL),Y  A mit Tabellenwert vergleichen
1794  D0 EA              BNE START       Abbruch bei der ersten
                                         Ungleichheit
1796  20 63 1F           JSR INCPT       Pointer+1 für Folgebyteprüfung
1799  A9 FF              LDA # $ FF      Prüfe auf Begrenzungszeichen der
179B  D1 FA              CMP (POINTL),Y  Tabelle
179D  D0 08              BNE WPRUEF      überspringe, wenn kein FF
179F  A9 8D              LDA # $ 8D      opcode für STA nach VEB bringen
17A1  8D EC 17           STA VEB
17A4  4C F8 18           JMP LOADT7      weiter in der bekannten
                                         Laderoutine
17A7  20 F8 18   WPRUEF  JSR LOADT7      Prüfung der folgenden von Band
                                         kommenden Zeichen ohne Abspeicherung
17AA  4C 8D 17           JMP BACK        weiter in der Schleife
17AD  EA                 NOP END         Kennzeichner für RALOAD
```

Usage of zeropage: 00 FA = POINTL 00FB = POINTH
Program is fully relocateable.

HEADHUNTER fordert für den Lesevorgang ID = 00 oder = FF in Zelle 17F9. Das Programm akzeptiert HYPERTAPE.

S T A T I S T I C I A N

E: Data records on cassette tape can be searched nonstop for a match to an array of preset tokens. Full matches are counted.

Eine häufige Fragestellung der Statistik lautet: Wie oft kommt eine Kombination von Eigenschaften in einer Menge, in einem statistischen Universum vor? In der kaufmännischen und in der Bevölkerungsstatistik sowie in den Naturwissenschaften möchte man dabei an die vorhandenen Aufzeichnungen, die Dateien, Datensätze, je nach Untersuchungsgegenstand unterschiedliche Fragestellungen richten können. Die zu untersuchenden Merkmalskombinationen sind also variabel.

Mit dem kleinen Programm STATISTICIAN können Sie mühelos das Zutreffen beliebiger interessierender Merkmalskombinationen in einer Datei zählen. Wie im vorher präsentierten HEADHUNTER wird auch hier während des Lesens eines Magnetbandes mit der abgewandelten KIM-Routine LOADT ein laufender Vergleich durchgeführt.

Treffen alle binär oder hexadezimal codierten Eigenschaften zu, dann wird ein Zähler erhöht. Sobald nur ein einziges Byte zwischen Vergleichstabelle und Datensatz nicht übereinstimmt, wird der Zähler nicht angesprochen. Es gibt aber folgende Ausnahme zur Steuerung der Paarigkeitsprüfung:

> Merkmalsbytes, die man aus der Paarigkeitsprüfung ausklammern möchte, werden in der Tabelle mit 7F = 'DELETE' eingesetzt. Im zu überprüfenden Datensatz darf dann an dieser Stelle vorkommen, was binär sonst möglich ist. STATISTICIAN ignoriert das.

Ein Laden der Datensätze findet nicht statt. Das Programm läuft nonstop über alle Datensätze hinweg, bis es auf einen die Datei abschließenden FILE SEPARATOR trifft. Das ist ein Datensatz mit dem einzigen hexadezimalen Byte '1C' (ASCII = FS). Wenn dieses 'reservierte'Zeichen auftritt, geht die Kontrolle an KIM zurück, der hexadezimale Zählerstand wird angezeigt.

Die zu überprüfenden Datensätze dürfen in beliebigem Code angelegt sein. Vielleicht nimmt man ASCII oder rein binär. Zu beachten ist nur, das bei der vorliegenden Programmaufschreibung folgende hexadezimale Kombinationen als Steuerzeichen reserviert sind:

1C = FILE SEPARATOR
7F = DELETE, ein Byte wird aus der Prüfung ausgeklammert
2F = '/' Schrägstrich, Ende eines Datensatzes (KIM-Monitor)
FF = Ende der Vergleichstabelle.

Die Datensätze dürfen beliebig lang sein. Man kann also tausende von Eigenschaften prüfen. Die Vergleichstabelle mit den Kriterien wird an beliebiger Stelle des RAM angelegt. Ihre Basisadresse wird in der 4. bis 7. Programmzeile in den Pointer geladen. Die Tabelle darf kürzer als die zu prüfenden Datensätze sein, sie wird durch ein 'FF' begrenzt. - Ist die Tabelle länger als die Datensätze, so wird nie Übereinstimmung festgestellt.

> In die ID-Zelle 17F9 muß 00 gegeben werden, andernfalls strandet der Programmablauf.

Man kann das Programm natürlich modifizieren und bei z.B. 5 untersuchten Merkmalen feststellen, wie oft 0, 1, 2, 3, 4 oder 5 Merkmale zutrafen. Ein schöner Fall für indizierte Zähleradressierung in Abhängigkeit vom Zählergebnis.

STATISTICIAN akzeptiert schnellen HYPERTAPE in der Dateneingabe.

```
                    STATISTICIAN

1780  A9 00    START     LDA # $ 00     Lösche Zähler auf 00
1782  85 E0              STA COUNTL
1784  85 E1              STA COUNTH
1786  A9 xx    INITPT    LDA # $ xx     Lade den Vektor der Tabellenbasis-
1788  85 FA              STA POINTL     adresse in den Pointer
178A  A9 xx              LDA # $ xx
178C  85 FB              STA POINTH
178E  A9 60              LDA # $ 60     RTS-Opcode für den VEB
1790  20 75 18           JSR LOADT+2    KIM-Laderoutine
1793  C9 1C    WITHCH    CMP # $ 1C     folgt ein FILE SEPARATOR?
1795  F0 3B              BEQ ENDFLE     wenn ja: Abbruch des Vergleiches
```

```
1797  C9 2F           CMP # $ 2F     / (Schrägstrich) als Satzende-Marke?
1799  F0 EB           BEQ INITPT     wenn ja: hole nächsten Datensatz
179B  A0 00           LDY # $ 00     Für indirekte Adressierung
179D  D1 FA           CMP (POINTL),Y Vergleiche mit Tabellenwert
179F  D0 29           BNE DELETE     noch prüfen, ob DELETE in der Tabelle
17A1  20 63 1F MATCH  JSR INCPT      Pointer um 1 erhöhen (KIM)
17A4  B1 FA           LDA (POINTL),Y Hole nächsten Tabellenwert, folgt
17A6  C9 FF           CMP # $ FF     Ende der Tabelle?
17A8  F0 17           BEQ INCCTR     wenn ja: Zähler erhöhen
17AA  A2 02           LDX # $ 02     Die nun folgenden Anweisungen
17AC  20 24 1A READCH JSR RDCHT      entsprechen den Zeilen 247 bis 254
17AF  C9 2F           CMP # $ 2F     im KIM-Monitor
17B1  F0 D3           BEQ INITPT     Bei Satzende: Hole nächsten Satz
17B3  20 00 1A        JSR PACKT
17B6  F0 03           BEQ            skip
17B8  4C 29 19        JMP LOADT9     Fehleranzeige bei nicht-hex
17BB  CA              DEX
17BC  D0 EE           BNE READCH     Weiterlesen

17BE  4C 93 17        JMP WITHCH     nun Zeichenprüfung

17C1  E6 E0    INCCTR INC COUNTL     erhöhe Zählerstand
17C3  D0 02           BNE NEXT       kein Überlauf?
17C5  E6 E1           INC COUNTH     erhöhe high order
17C7  4C 86 17 NEXT   JMP INITPT     nächsten Datensatz prüfen

17CA  A9 7F    DELETE LDA # $ 7F     Lade das DELETE-Zeichen
17CC  D1 FA           CMP (POINTL),Y Ist es auch in der Tabelle?
17CE  F0 D1           BEQ MATCH      dann behandeln wie paarig
17D0  D0 B4           BNE INITPT     sonst Sprung zum nächsten Datensatz

17D2  A5 E0    ENDFLE LDA COUNTL     umspeichern des Zählers in die
17D4  85 FA           STA POINTL     KIM-Anzeige
17D6  A5 E1           LDA COUNTH
17D8  85 FB           STA POINTH
17DA  4C 4F 1C        JMP START      Anzeige durch KIM-Monitor
17DD  EA              NOP            RALOAD-Byte
```

Zeropage used:

```
00E0                 COUNTL
00E1                 COUNTH
00FA                 POINTL
00FB                 POINTH
```

FILE SEPARATOR als letzten Datensatz nicht vergessen!

Eine Kombination der in HEADHUNTER und STATISTICIAN aufgezeigten Lösungswege würde es erlauben, jeweils nur solche Datensätze einzuspeichern, die einer Mindestzahl von Merkmalsübereinstimmungen entsprechen. Weiterhin wäre lösbar, eine solche Einspeicherung dann zu überschreiben, wenn ein Datensatz mit einer noch höheren Übereinstimmung gefunden wird. Und schließlich könnte man während des Lesevorganges ein Verzeichnis aller Datensätze anlegen, die den gesuchten Merkmalen entsprechen.

Ebenso könnte man die Bearbeitung einer Postliste vornehmen. ●

Neue Produkte

A I M 6 5

Ein preiswertes Entwicklungssystem von Rockwell.

AIM 65 (Advanced Interface Module) wurde von Rockwell jetzt in Genf vorgestellt: R6502-based Mikrocomputer mit alphanumerischem Thermodrucker und alphanumerischer Leuchtanzeige,jeweils 2o Zeichen breit, 64 ASCII-Zeichen darstellbar, 4k-Betriebssystem und 1k oder 4k RAM auf einer Karte. Eine über Kabel angeschlossene zweite Karte enthält eine Tastatur mit 54 Tasten für alphabetische, numerische und Spezialfunktionen.

Die Auslieferung in Europa soll im September/Oktober 1978 beginnen. Hinsichtlich des Preises darf man etwa das Eineinhalbfache des KIM-1 annehmen. Das System wird getestet und funktionsfähig geliefert. Die benötigten Spannungen von +5 V, +12 V und +24 V sind wie üblich vom Benutzer vorzuhalten.

Mit AIM 65 bzw. AIM 65-2 (4k RAM auf der Karte) werden die Lern- und Entwicklungssysteme, bzw. die für viele Zwecke direkt einsetzbaren Mikrocomputer wieder einen Schritt leistungsfähiger und preiswerter.

Die Tastatur gleicht in Größe und Anordnung der eines Terminals. Sie hat Tasten für 26 Buchstaben, 10 Ziffern, 22 Sonderzeichen, 8 Kontrollfunktionen und für 3 Funktionen, die der Benutzer festlegen kann.

Interessant ist natürlich der Thermodrucker. Er arbeitet von der Rolle preiswerten, wärmeempfindlichen Papieres. 64 Zeichen können in 5x7-Matrix dargestellt werden, pro Minute können 90 Zeilen ausgegeben werden, à 20 Zeichen.

Die Leuchtanzeige hat ebenfalls 20 Zeichen Breite und stellt den gleichen Zeichenvorrat deutlich lesbar dar.

In 4k ROM ist das Betriebssystem untergebracht; es unterstützt nicht nur Tastatur, Anzeige und Drucker sowie TTY-Ein- und Ausgabe, sondern auch den Anschluß von bis zu zwei Standard-Cassettenrekordern für die Abspeicherung oder für das Laden von Programmen (hierbei ASCII KIM-kompatibel oder auch binär). Auf der CPU-Karte sind ferner Reservesockel vorhanden, mit denen die Festwertspeicher auf bis zu 16k Bytes erweitert werden können, sei es durch EPROMs des Benutzers, sei es durch Aufstecken der Rockwell-ROMs mit 4k Assembler/Text Editor oder mit 8k BASIC Interpreter.

Das normale 4k-Betriebssystem mit seinem Debug/Monitor kann Registerinhalte oder Speicherzellen anzeigen oder ändern, es kann durch Setzen von Breakpoints in den TRACE-Mode zur abschnittsweisen Kontrolle des Programmablaufes gebracht werden.25 verschiedene Anweisungen werden mit einem Tastendruck ausgeführt, Fehler werden gemeldet. Darin sind auch die Befehle für I/O über Cassettenrekorder oder TTY enthalten.

AIM 65 hat 2 Stück 44-Pin-Stecker, die mit denen des KIM voll kompatibel sind. Der Anwendungsstecker ist für ASR 33 TTY und für 2 Cassettenrekorder ausgelegt. Er enthält ferner 2x8 bidirektionale Ports (TTL-kompatibel) für den Verkehr mit der Außenwelt, ferner 2x2 Handshake-Ports. Alles realisiert durch das R6522 VIA (Versatile Interface Adapter), das auch 2 Stück 16 Bit breite Zähler/Timer für den Anwender voll programmierbar zur Verfügung stellt.

Der Expansionsstecker erlaubt mit seinem vollen Zugriff auf Adreß-, Daten- und Kontrollbus vielfältige Erweiterungen.

Herz des Systems ist die bekannte R6502 CPU mit 1 MHz Taktfrequenz, ihren bekannten leistungsfähigen 13 Adressierungsarten und der Adressierfähigkeit für 64k Speicheradressen.

Zusammen mit den Handbüchern für den Anwender, für die Programmierung und für die Hardware verspricht AIM 65 ein echter Renner für Rockwell zu werden. So früh als möglich sollen Sie in diesem Journal einen ersten Anwenderbericht finden.

Nach den hier vorliegenden Ankündigungen darf der Anwender von Rockwell weitere interessante Bausteine für die kompatible 65xx-Familie erwarten. Vorgestellt ist schon das große Entwicklungssystem SYSTEM 65 mit 2 Floppies. Es folgen weitere intelligente Interface-Chips, Weiterentwicklungen bei Speichern (Bubble), 1-Chip-Mikroprozessoren.

KILOBAUD 11/77, S. 84 J.Blankenship

Expand Your KIM with Altair Bus Devices

Übersichtsartikel für Erweiterung des KIM-1 mit Tastatur, Bildschirmdisplay, Drucker und RAM-Speichern

65xx MICRO MAG

KILOBAUD 12/77, S. 36, J. Blankenship

Expand Your KIM, Part 2

Aufbau eines Gehäuses und Schaffung der Anschlüsse für S-100-Bus.

65xx MICRO MAG

KILOBAUD 12/77, S. 94, Don Rindsberg

Here's HUEY, super calculator for 6502

Rechenroutinen für höhere Funktionen, 13 Stellen Genauigkeit. 10 pages Coding, leider ohne Kommentare.

65xx MICRO MAG

KILOBAUD 1/78, S. 36, John Eaton

Growing with KIM

Bustreiberkarte für KIM-Erweiterung. Anschlußpläne, Bestückung und Layout.

65xx MICRO MAG

KILOBAUD 1/78, S. 98, H. Chamberlin

Software Keyboard Interface

64-Tasten-Keyboard für den KIM unter Benutzung nur eines SN 74154-Dekoders. Umfangreiches Programm zur Tastatursteuerung. Besprochen in diesem Heft.

65xx MICRO MAG

KILOBAUD 2/78, S. 68, J.Blankenship

Expand Your KIM, Part 3

Aufbau der Bustreiberkarte und des Dekoders für 4k RAM.

65xx MICRO MAG

KILOBAUD 3/78, S. 84, J.Blankenship

Expand Your KIM, Part 4

Aufbau eines billigen TTY mit SWTP-Tastatur und PR-40-Drucker. Anschluß einer herausgezogenen KIM-Tastatur und eines erweiterten Displays.

65xx MICRO MAG

KILOBAUD 5/78, S. 60, J.Blankenship

Expand Your KIM, Part 5

Anschluß eines A/D-Konverters für joysticks (Potis) und von Berührungstasten.

65xx MICRO MAG

65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

DIPL.-VOLKSWIRT
ROLAND LÖHR
DATENVERARBEITUNGSFACHMANN

HERAUSGEBER

HANSDORFER STRASSE 4
2070 AHRENSBURG
☎ (04102) 55 816

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

Bezugsbedingungen: Abonnement für 6 Ausgaben im Inland DM 40,-, einschl. Versandkosten, Porto, Umsatzsteuer. Ausland DM 46,- (surface). Firmen erhalten Rechnung. Richten Sie Ihre Überweisung/Euroscheck an:

ROLAND LÖHR, Konto 20/01121, Vereins- und Westbank, BLZ 200 300 00.

Informationsblätter für Autoren, Werbetreibende und Distributoren stehen zur Verfügung.

Redaktions- und Anzeigenschluß für Nr. 2 ist der 15. Aug.1978.

★

In den nächsten Heften finden Sie u.a.:

Grundsatzartikel zur Programmierung, zur Tabellenverarbeitung - Ein System von Rechenroutinen - Weitere Programme zur Auswertung von Daten auf Cassetten - Interaktive Programmierung - ROLDIS, ein Disassembler mit rollender Anzeige - Leistungsfähige MAKROS für die adressen- und opcodefreie Verarbeitung von arrays - Anwenderschaltungen - Weitere Mitglieder der 65xx-Familie - Umfangreiche Literaturquellen - Anwenderberichte für AIM 65 und PET.

★

Weitere Literaturhinweise

MICRO, The 6502 Journal 5/78, S. 5

LIFE For Your PET, by Dr. Frank Covitz.

Programm für das beliebte Spiel in Assemblercode. 1,5 Pages. Geeignet auch für andere 65xx-Systeme mit TV-Display

BYTE 11/77, S. 150, St. Wozniak

SWEET16, The 6502 Dream Machine

Interpreter-Routinen zur Simulation eines 16-Bit-Mikroprozessors mit 16 Registern. Programmlistung.

KIM USER NOTES 10+11,S.3, Lew Edwards

KIM D-Bug

Ein Tracerprogramm zur abschnittsweisen Programmüberprüfung mit Registeranzeige und -veränderbarkeit.

KIM USER NOTES 10+11,S.6, James Wood

RPN Calculator Interface to KIM

Anschluß des Number Chrunchers MM 57109 von NS an ein erweitertes PIA. Mit Programm.