# Von Basic zu Assembler

# (Teil 12)

**Folge 11 hat uns die Verarbeitung von Tabellen in Assemblerprogrammen nähergebracht. Diesmal wenden wir uns besonderen Tabellen zu, nämlich den Variablentabellen, die der Basic-Interpreter anlegt.**

Wozu das, werden Sie vielleicht fragen, wir programmieren doch in Assembler? Nun, es sei keinem verwehrt, sich das Leben unnötig schwer zu machen! Wer aber ökonomisch programmieren möchte, dem lege ich nicht nur die Routinen des Basic-Interpreters, sondern auch den problemlosen Umgang mit Variablen durch diesen Interpreter ans Herz.

## Kooperation von Basic und Assembler

Nehmen wir einmal an, wir schreiben ein Assemblerprogramm, das alle Variablentypen und auch Arrays benötigt, und diese während des Programmlaufes erst erhält (durch manuelle Eingabe, von Diskette etc.). Was hätten wir zu programmieren? Handelt es sich nicht nur um ganz wenige Werte (für die braucht man keinen großen Aufwand zu treiben), dann muß eine Routine geschrieben werden, die die Abfrage durchführt (beispielsweise mit einer Aufforderung an den Benutzer, den Wert nun einzutippen). Weiterhin muß nun der Typ erkannt werden, denn beispielsweise können Integerzahlen viel einfacher und schneller verarbeitet werden als Fließkommazahlen und wenn man Boolesche Variable auch noch zuläßt, ist wieder eine andere Behandlung angesagt — von Strings oder Arrays der verschiedenen Typen sowie Funktionsdefinitionen ganz zu schweigen. Damit aber noch nicht genug! Die eingegebenen Werte müssen irgendwo so sinnvoll abgelegt werden, daß sie im richtigen Format jederzeit schnell wiedergefunden werden können, Fehler müssen aufgefangen und eventuelle Ausgabemöglichkeiten eingeplant werden: Eine wahre Herkulesaufgabe!

Wie leicht haben wir es da in Basic, wo all dies der Interpreter mit seinen Routinen für uns erledigt. Außer in wenigen Spezialfällen verfahre ich daher meistens so: Ein Basic-Rahmen-Programm erledigt die Annahme und Organisation (fast) aller Variablen und Arrays. Aus diesem Programm wird dann in das Assemblerprogramm geschaltet, das mit den eingegangenen Werten arbeitet. Auf diese Weise spielt sich der von der Geschwindigkeit her kritische Teil eines Programms in der schnellen Maschinensprache ab, der von daher aber unkritische Teil der Variablenorganisation (häufig dreht es sich ja um einen interaktiven Teil) im Rahmen des Basic und höchst einfach. Um so arbeiten zu können, müssen wir mehr über die Variablentabellen wissen und auch über die Routinen, die der Interpreter zum Zugriff darauf anbietet.

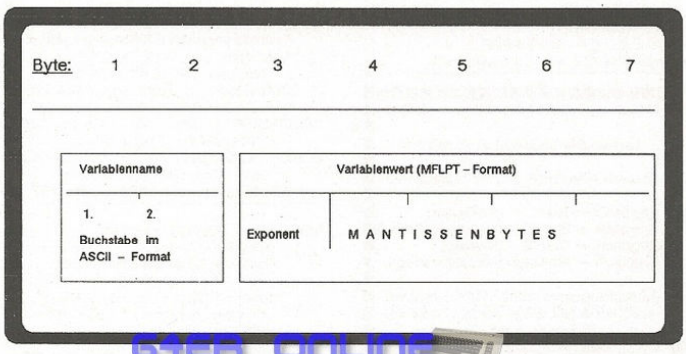




**Bild 1. Auf diese Art und Weise wird eine Fließkommavariable in die Variablentabelle eingetragen. Byte 1 und 2 enthalten den Namen**

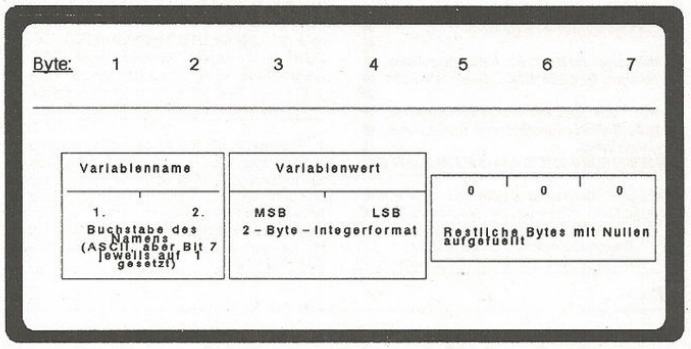


**Bild 2. Dies ist das Format eines Integervariablen-Eintrages in die Variablentabelle. Byte 5, 6 und 7 bleiben ungenutzt.**

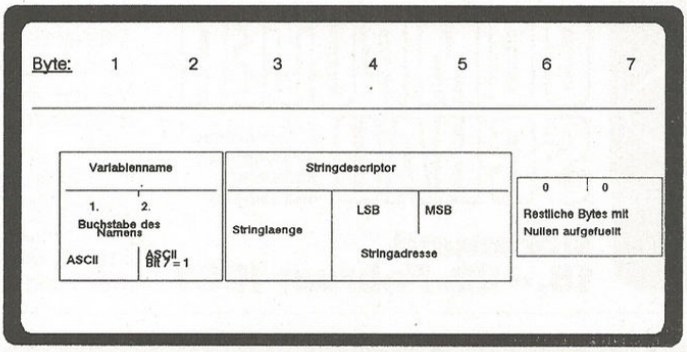


**Bild 3. Ein String erzeugt diesen Eintrag in die Variablentabelle**

## Variablentypen des Basic

Sehen wir uns zunächst einmal die verschiedenen Arten von Variablen des Basic an: Eine erste grobe Unterteilung liefert zwei Sorten von Variablen. Man findet nämlich sogenannte indizierte und nichtindizierte. Die indizierten sind solche, die in einem Zusammenhang mit anderen indizierten in einer bestimmten Ordnung, dem Feld oder Array stehen und die durch einen Index voneinander unterschieden werden (beispielsweise A(2), A(7) und so fort). Ihnen werden wir uns in der nächsten Folge widmen. Es bleiben also die Variablen ohne Index, von denen wir in der durch den Interpreter angelegten Variablentabelle vier Sorten finden. Jede Sorte beansprucht einen sieben Byte langen Eintrag in der Tabelle.

Am häufigsten verwendet der Basic-Programmierer (und der Assemblerspezialist wohl auch) die Fließkommavariable. Was man darunter zu verstehen hat, haben Sie in der Folge 10 (64'er 1/87) erfahren. Im Basic-Programmtext tauchen diese Variablen ohne weitere Kennung auf, beispielsweise als A, A1, CD und so fort. Bild 1 zeigt Ihnen den Aufbau eines solchen Fließkomma-Variablen-Eintrages in die Variablentabelle. Die beiden ersten Byte enthalten den Namen (im ASCII-Format), die restlichen fünf Byte den Wert der Variablen im MFLPT-Format.

Integervariable (also ganze Zahlen, die sich in zwei Byte ausdrücken lassen) werden im Basic-Text durch das %-Zeichen markiert. In Bild 2 sehen Sie den Unterschied zur Fließkommavariablen beim Eintrag in die Variablentabelle. Auch hier geben die beiden ersten Byte den Namen der Variablen wieder, dabei findet zwar das ASCII-Format Anwendung, aber bei beiden Byte ist als Kennung noch Bit 7 gesetzt. Die Bytes 3 und 4 enthalten den 2-Byte-Variablenwert in der Reihenfolge MSB/LSB. Die restlichen Byte sind mit Nullen gefüllt, sie bleiben unbenutzt.

Wie Sie sicher wissen, sind Stringvariablen durch das $-Zeichen markiert. Ihr Eintrag in die Variablentabelle ist etwas komplexer als die beiden bisher betrachteten Typen, siehe Bild 3. Die beiden ersten Byte enthalten wieder den Variablennamen, wobei im zweiten Byte das Bit 7 gesetzt ist (zur Kennzeichnung des Typs). In den drei folgenden Byte findet sich der sogenannte Stringdescriptor (zu

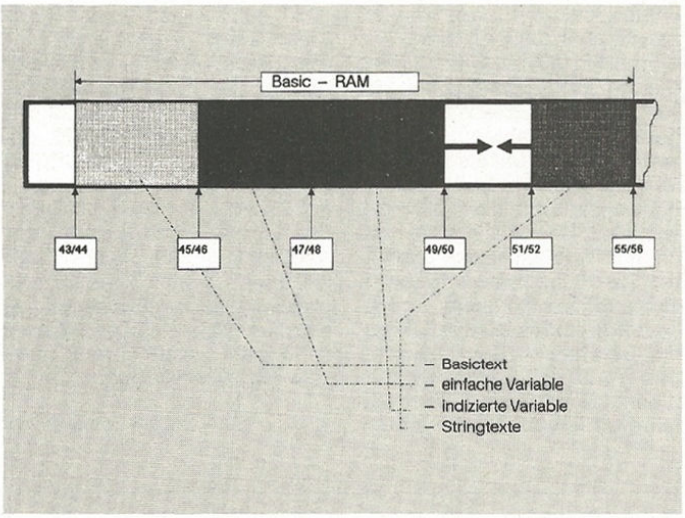


**Bild 5. Die Organisation des Basic-RAM im C 64**

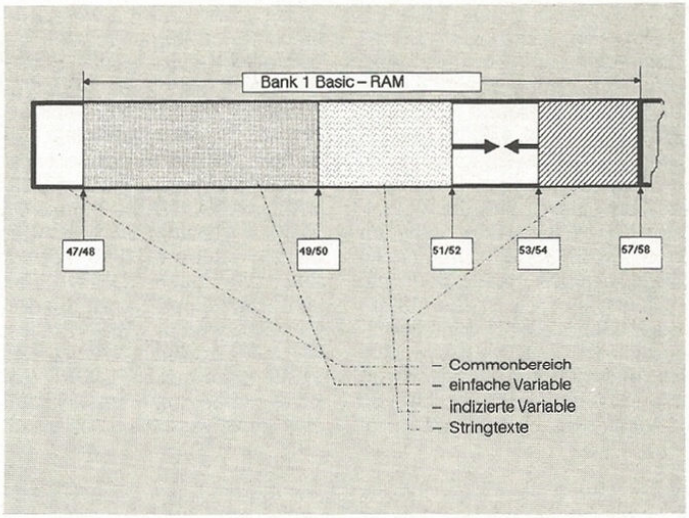


**Bild 6. Die Organisation des Basic-RAM in der Bank 1 des C 128**

deutsch »Stringbeschreiber«). Byte 3 (das erste Byte des Descriptors) enthält die Stringlänge, die Bytes 4 und 5 die Startadresse des Textes im normalen 2-Byte-Format. Die restlichen beiden Bytes sind unbenutzt und in ihnen steht der Wert 0. In diesem Variableneintrag liegt nur eine genauere Beschreibung der Variablen! Wo also ist der Text und wie sieht er aus?

Vom oberen Ende des Basic-RAM an abwärts sind die Stringtexte zu finden. Beim C 64 also ab $A000, beim C 128 in Bank 1 von $FF00 an. Wir sehen uns diese Aufteilungen gleich noch detaillierter an. Der Zeiger im Stringdescriptor weist genau auf das erste ASCII-Zeichen des hier gespeicherten Textes. Für den C 64 ist damit schon alles geklärt. Der C 128 aber birgt noch eine kleine Besonderheit, die die sogenannte »Garbage Collection« beschleunigt (darunter versteht man das Aufräumen von nicht mehr gebrauchten Stringtexten): Nach dem eigentlichen Text findet sich hier noch ein 2-Byte-Zeiger, der auf den Stringdescriptor weist (manchmal wird er »Codedescriptor« genannt, vermutlich deshalb, weil er sich an den Text im ASCII-Code anschließt).

Ein Außenseiter macht sich in der Variablentabelle als vierter »Variablentyp« breit: Die benutzerdefinierte Funktion. Bild 4 zeigt Ihnen solch einen Eintrag. Außer den beiden Namen-Byte zu Beginn (im ersten davon ist das Bit 7 gesetzt) finden wir hier zwei Zeiger. Der erste davon (Byte 3 und 4) weist auf die Funktionsvorschrift im Basic-Text. Der zweite Vektor enthält die Startadresse der Funktionsvariablen in der Variablentabelle (beispielsweise X aus der Funktionsdefinition DEF FN AB(X) = ...). Das letzte Byte ist unbenutzt.

Wir haben nun noch zwei Fragen zu klären: Wo befindet sich

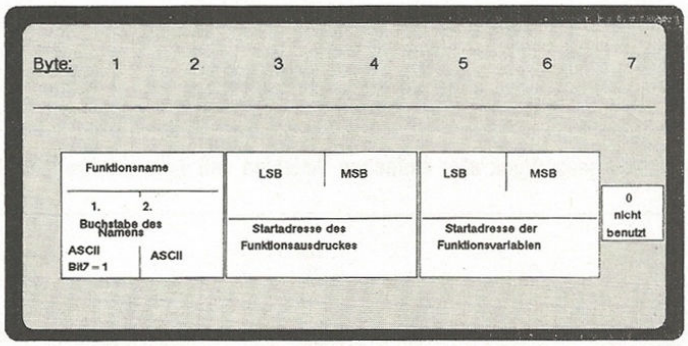


**Bild 4. Auch das findet sich in der Variablentabelle: Eine durch den Benutzer definierte Funktion.**

diese Tabelle und wie kann man sie benutzen? Den Ort der Variablentabelle in absoluten Adressen anzugeben ist unnötig (und beim C 64 auch nicht so ohne weiteres möglich). Dort nämlich schließt sie sich nahtlos an den Basic-Programmtext an. Beim C 128 verhält sich das einfacher: Da liegt sie in der Bank 1 gleich oberhalb der Common-Area, also ab $400.

Den genauen Anfang und die ganze Organisation all dieser Tabellen (außer mit der Variablentabelle haben wir es noch mit den Arrays und den Stringtexten zu tun) erfährt man am besten aus einer Reihe von Vektoren, die sich in der Zeropage befinden. Die Bilder 5 und 6 illustrieren die Zusammenhänge für den C 64 und für den C 128.

VARTAB heißt der erste dieser Zeiger (C 64: 45/46, C 128: 47/48). Er weist auf den Anfang der Variablentabelle und somit im allgemeinen beim C 64 auch auf das Programmtextende, beim C 128 dagegen auf $400 in Bank 1. Das Ende der Tabelle mit den einfachen Variablen erfährt man durch den nächsten Zeiger, der ARYTAB genannt wird (C 64: 47/48, C 128: 49/50) und gleichzeitig auch den Anfang der Array-Tabelle verrät. STREND ist der Name des Zeigers auf das Ende der Tabelle der indizierten Variablen (C 64: 49/50, C 128: 51/52). Wenn man es genaunimmt, so ist die in STREND gespeicherte Adresse ein Byte höher als dieses Ende.

Beide bisher genannten Tabellen wachsen zu immer höheren Adressen beim Hinzufügen weiterer Variablen oder Arrays. Anders herum — wie schon oben erwähnt — verhält sich das mit der Tabelle der Stringtexte. Diese fangen beim C 64 normalerweise direkt unterhalb des Basic-Interpreters an, also an der Adresse $9FFF, und dorthin weist der Zeiger MEMSIZ (55/56). Der C 128 stellt uns wesentlich mehr Platz zur Verfügung. Hier beginnen die Stringtexte direkt unterhalb der MMU-Register (genaugenommen unterhalb des CR = Konfigurationsregister) bei $FEFF in Bank 1. Auch hier hilft uns wieder ein Zeiger, der MAX-MEM-1 heißt und in 57/58 zu finden ist. Die aktuelle Textfront schiebt sich von String zu String immer weiter abwärts. Ihre Adresse kann man aus dem Vektor FRETOP (C 64: 51/52, C 128: 53/54) erfahren.

Beide Fronten schieben sich so im Verlauf eines Programms (wenn ständig neue Texte hinzukommen) aufeinander zu, bis sie sich irgendwann einmal berühren. Genaugenommen wird vor jedem Speichern eines neuen Textes geprüft, ob er noch in den verbleibenden Speicherabstand zwischen FRETOP und STREND paßt. Ist das einmal nicht mehr der Fall, dann findet die Beseitigung von Stringtextmüll — die vorhin schon erwähnte Garbage Collection — statt, bei der Texte ohne Descriptor entfernt und die gültigen Texte säuberlich nach oben gestapelt werden. Reicht auch diese Maßnahme nicht mehr aus, dann meldet sich der Interpreter mit einer Fehlerbotschaft: Out of Memory Error.

Wir können nun auf die Variablentabelle zugreifen wie auf jede andere Tabelle, die Elemente zu je sieben Byte enthält. Das beigefügte Programm DUMP (Listing 1) zeigt Ihnen das für den C 64, indem es eine Liste aller definierten Variablen eines Basic-Programms und ihrer Inhalte ausgibt. Durch SYS 49152 wird diese Ausgabe gestartet. In Listing 2 finden Sie den ausführlich dokumentierten Quelltext, so daß Sie auch schnell erkennen, daß das Programm noch verbessert werden kann. Für den C 128 ist es nicht so ohne weiteres umzuschreiben, denn hier treten wieder allerlei Bank-Probleme auf.

Drei Interpreter-Routinen wurden im Programm verwendet, die mit der Ausgabe der Variableninhalte auf den Bildschirm zu tun haben: NUMDON druckt den FAC-Inhalt (hier also eine Fließkommazahl) auf den Bildschirm. Diese Routine wird durch JSR $AABC (dezimal 43708) angesteuert. OUTSTR dient zur Stringausgabe auf den Bildschirm. Dazu muß der Textstart im Vektor INDEX (das ist

$22/23 oder dezimal 34/35) enthalten sein und die Länge des Stringtextes im X-Register. Sind diese Bedingungen erfüllt, dann beginnt die Ausgabe durch JSR $AB25 (dezimal 43813). LINPRT zeigt eine 2-Byte-Integerzahl auf dem Bildschirm an. Dazu muß das LSB dieser Zahl im X-Register, das MSB im Akku enthalten sein. JSR $BDCD (dezimal 48589) führt dann den Ausdruck durch.

Häufiger noch ist die Aufgabenstellung, eine bestimmte Variable zu suchen. Haben wir also im Aufrufprogramm eine Variable A1 definiert, dann sollte es möglich sein, im Assemblerprogramm unter Angabe dieses Namens einen Zeiger auf den Variablenwert zu erhalten. Genau das leistet die Routine ORDVAR (C 64: $B0E7 = dezimal 45287, C 128: $7B0B = dezimal 31499). Dazu trägt man den Variablennamen in die Speicherstellen VARNAM und VARNAM+1 ein (C 64: $45/6 = dezimal 69/70, C 128: $47/8 = dezimal 71/2), springt dann die Routine ORDVAR an und erhält einen Zeiger auf den Variablenwert in VARPNT (C 64: $47/8 = dezimal 71/2, C 128: $49/A = dezimal 73/4). ORDVAR ist so entgegenkommend, daß eine neue Variable automatisch eingerichtet wird, wenn die benannte noch nicht existiert.

Im Basic 7.0 des C 128 gibt es die POINTER-Funktion, mit deren Hilfe die Adresse einer beliebigen Variablen erfahren werden kann. Mittels ORDVAR läßt sich diese Funktion auch relativ einfach für den C 64 entwickeln. Ihre Kenntnisse reichen dazu allemal aus: Probieren Sie doch, diese Aufgabe zu lösen. Eine mögliche Vorgehensweise wäre es, einen Aufruf der eigenen Pointer-Routine in der Form SYS Adresse,Variablenname + Kennung durchzuführen. Hierzu müßte dann der Text gelesen und in VARNAM gespeichert werden, wobei die entsprechenden Bit 7 zu setzen wären. Ein Aufruf von ORDVAR liefert dann den Zeiger auf den Variablenwert in VARPNT, der beispielsweise durch LINPRT ausgegeben werden könnte.

Es bleibt nun noch die Aufgabe, uns die andere durch den Basic-Interpreter eingerichtete Tabelle — nämlich die der Arrays — genauer anzusehen. Das wird das Thema in der nächsten Folge sein und damit auch unseren Kurs beschließen.

(Heimo Ponnath/pd)

```
Name : dump c 64          c000 c0c9
-------------------------------------
c000 : a5 2d 85 fb a5 2e 85 fc  f8
c008 : a9 0d 20 d2 ff a0 00 b1  03
c010 : fb 08 29 7f 20 d2 ff c8  74
c018 : b1 fb 08 29 7f d0 02 a9  c8
c020 : 20 20 d2 ff 28 30 0e 28  91
c028 : 30 03 4c 83 c0 a9 21 20  7b
c030 : d2 ff 4c a9 c0 28 30 28  a8
c038 : a9 24 20 d2 ff a9 20 20  64
c040 : d2 ff a9 3d 20 d2 ff a9  10
c048 : 20 20 d2 ff c8 b1 fb aa  8c
c050 : c8 b1 fb 85 22 c8 b1 fb  c7
c058 : 85 23 20 25 ab 4c a9 c0  61
c060 : a9 25 20 d2 ff a9 20 20  0c
c068 : d2 ff a9 3d 20 d2 ff a9  38
c070 : 20 20 d2 ff c8 b1 fb 48  ef
c078 : c8 b1 fb aa 68 20 cd bd  a7
c080 : 4c a9 c0 a9 20 20 d2 ff  54
c088 : a9 20 20 d2 ff a9 3d 20  26
c090 : d2 ff a9 20 20 d2 ff 18  99
c098 : c8 98 65 fb 48 a5 fc 69  fe
c0a0 : 00 a8 68 20 a2 bb 20 bc  14
c0a8 : aa a9 0d 20 d2 ff 18 a5  47
c0b0 : fb 69 07 85 fb a5 fc 69  86
c0b8 : 00 85 fc a5 fb c5 2f a5  64
c0c0 : fc e5 30 b0 03 4c 0d c0  19
c0c8 : 60 70 70 70 70 70 70 70  b8
```

**Listing 1. DUMP C 64 erzeugt einen Bildschirmausdruck aller einfachen Variablen und ihrer aktuellen Werte. Bitte mit dem MSE eingeben.**

```
10  -;*******************************
20  -;*                             *
30  -;*           d u m p           *
40  -;*                             *
50  -;*  programm zum listen aller  *
60  -;*  variablen und ihrer werte  *
70  -;*                             *
80  -;* heimo ponnath  hamburg 1986 *
90  -;*                             *
100 -;*******************************
110 -           .ba $c000
120 -;
130 -;--- verwendete labels ---------
140 -;
150 -           .eq index=$22  ;zeiger fuer outstring
160 -           .eq vartab=$2d ;start der variablentabelle
170 -           .eq arytab=$2f ;ende der variablentabelle
180 -           .eq help=$fb   ;hilfszeiger
190 -;
200 -           .eq numdon=$aabc;fac ausgeben
210 -           .eq outstr=$ab25;string ausgeben
220 -           .eq movfm=$bba2;laedt fac aus speicher
230 -           .eq linprt=$bdcd;integer ausgeben
240 -           .eq chrout=$ffd2;akkuinhalt ausgeben
250 -;
260 -;--- das programm ---------------
270 -;
280 -init       lda vartab      ;variablentabelle start
290 -           sta help        ;uebertragen
300 -           lda vartab+1
310 -           sta help+1
320 -           lda #$0d        ;carriage return
330 -           jsr chrout      ;ausgeben
340 -hole       ldy #$00        ;offset auf null
350 -           lda (help),y    ;erstes namenszeichen holen
360 -           php             ;status merken
370 -           and #$7f        ;loeschen von bit 7
380 -           jsr chrout      ;zeichen ausgeben
390 -           iny
400 -           lda (help),y    ;zweites namenszeichen holen
410 -           php             ;wieder status merken
420 -           and #$7f        ;und bit 7 loeschen
430 -           bne ausg        ;2.zeichen existiert
440 -           lda #$20        ;leerzeichen
450 -ausg       jsr chrout      ;ausgeben
460 -           plp             ;2.status zurueckholen
470 -           bmi test        ;integer oder string
480 -;sonst funktion oder fliesskomma
490 -           plp             ;1.status zurueckholen
500 -           bmi funktion    ;funktion liegt vor
510 -           jmp float       ;fliesskommavariable liegt vor
520 -funktion   lda #$21        ;ascii fuer !
530 -           jsr chrout      ;ausgeben
540 -           jmp rest
550 -test       plp             ;1.status zurueckholen
560 -           bmi integer     ;beide bit 7 gesetzt = integervariable
570 -;stringvariable liegt vor
580 -string     lda #$24        ;$-zeichen
590 -           jsr chrout      ;ausgeben
600 -           lda #$20        ;leerzeichen
610 -           jsr chrout
620 -           lda #$3d        ;=-zeichen
630 -           jsr chrout
640 -           lda #$20        ;leerzeichen
650 -           jsr chrout
660 -           iny             ;offset auf stringlaenge richten
670 -           lda (help),y    ;laenge laden
680 -           tax             ;und merken
690 -           iny             ;offset auf stringadresse richten
700 -           lda (help),y    ;lsb adresse
710 -           sta index
720 -           iny
730 -           lda (help),y    ;msb adresse
740 -           sta index+1
750 -           jsr outstr      ;string ausgeben
760 -           jmp rest
770 -;
780 -integer    lda #$25        ;%-zeichen
790 -           jsr chrout      ;ausgeben
800 -           lda #$20        ;leerzeichen
810 -           jsr chrout
820 -           lda #$3d        ;=-zeichen
830 -           jsr chrout
840 -           lda #$20
850 -           jsr chrout
860 -           iny             ;offset auf msb richten
870 -           lda (help),y    ;msb laden
880 -           pha             ;und merken
890 -           iny
900 -           lda (help),y    ;lsb laden
910 -           tax             ;und merken
920 -           pla             ;msb in akku
930 -           jsr linprt      ;integerzahl ausgeben
940 -           jmp rest
950 -;
960 -float      lda #$20        ;leerzeichen
970 -           jsr chrout      ;ausgeben
980 -           lda #$20        ;noch ein leerzeichen
990 -           jsr chrout
1000 -          lda #$3d        ;=-zeichen
1010 -          jsr chrout
1020 -          lda #$20        ;und noch ein leerzeichen
1030 -          jsr chrout
1040 -          clc             ;addition vorbereiten
1050 -          iny             ;offset auf erstes wertebyte richten
1060 -          tya             ;und in akku schieben
1070 -          adc help        ;ergibt lsb
1080 -          pha             ;merken
1090 -          lda help+1      ;msb
1100 -          adc #$00        ;eventuell carry addieren
1110 -          tay             ;msb merken
1120 -          pla             ;lsb zurueckholen
1130 -          jsr movfm       ;fac mit variablenwert laden
1140 -          jsr numdon      ;fac ausgeben
1150 -rest      lda #$0d        ;carriage return
1160 -          jsr chrout
1170 -          clc             ;addition vorbereiten
1180 -          lda help        ;lsb
1190 -          adc #$07        ;auf naechste variable
1200 -          sta help
1210 -          lda help+1      ;msb
1220 -          adc #$00        ;eventuell carry addieren
1230 -          sta help+1
1240 -;
1250 -          lda help        ;vergleich, ob ende
1260 -          cmp arytab      ;der variablentabelle
1270 -          lda help+1
1280 -          sbc arytab+1
1290 -          bcs ende
1300 -          jmp hole        ;naechste variable
1310 -ende      rts
1320 -;
```

**Listing 2. Hypra-Ass-Quelltext von DUMP C 64**