MEMORY MAP

Nur zwei Adressenpaare wollen wir diesmal behandeln. Die haben es aber in sich.

Das letzte Mal haben wir den Zeiger in den Speicherzellen 43-44 ($2B—$2C) kennengelernt, der den Anfang des Speicherbereichs für Basic-Programme angibt.

Die anschließenden Speicherzellen-Paare bis 55-56 ($37—$38) zeigen auf weitere für Basic-Programme wichtige Speicherbereiche, die deswegen gemeinsam betrachtet werden sollten. Bild 1 stellt den Zusammenhang grafisch dar. Dabei ist wichtig zu wissen, daß ein Basic-Programm während des Eintippens oder Einladens von Disk beziehungsweise Kassette in den ersten Block kommt. Während des Programmlaufs werden alle normalen Variablen in den Block geschrieben, alle Felder (Arrays) in den zweiten Block und schließlich der Text der Zeichenketten (Strings) sozusagen rückwärts vom Ende des Arbeitsspeichers (Block). Je nach Größe des Programms und nach Anzahl der Variablen wandern die Blockgrenzen nach oben beziehungsweise die von Block 4 nach unten. Wenn sie sich treffen beziehungsweise überschneiden gibt es »OUT OF MEMORY«.

Diese Blockbewegung ist in Bild 1 durch die Pfeile dargestellt.

Ich bin mir bewußt, daß gerade dieses Thema in den letzten Ausgaben des 64'er sowohl im Kurs »Der gläserne VC 20« als auch im Assembler-Kurs behandelt worden ist. Ich bin aber der Meinung, daß man es gar nicht oft genug erklären kann und nehme daher eine gewisse Wiederholung in Kauf.

Adresse 45-46 ($2D-$2E)
Zeiger auf die Anfangsadresse des Speicherbereichs für Variable

Dieser Zeiger, in der Low/High-Byte-Darstellung, gibt dem Basic-Interpreter an, ab welcher Speicherzelle die Variablen eines Basic-Programms gespeichert sind. Da die Variablen direkt an das Basic-Programm anschließen, zeigt dieser Zeiger natürlich gleichzeitig auf das Ende des Basic-Programms.

Es muß betont werden, daß es sich nur um den Bereich der »normalen« Variablen handelt, also nicht um Felder (Arrays). Anders als der Zeiger in 43-44, der auf fest definierte Speicherzellen zeigt, liegt der Zeiger für den Variablen-Beginn nicht fest. Je nach Länge des Programms wandert er nach oben.

Sobald ein Programm eingetippt oder aus einem externen Speicher (Diskette, Kassette) eingelesen ist, wird der Zeiger in 45-46 durch RUN auf ein Byte hinter das Programmende gesetzt und alle Variablen werden in der Reihenfolge ihres Auftretens gespeichert. Da normalerweise die Länge eines Basic-Programms während des Ablaufs konstant bleibt, werden die Variablen in ihrer Position auch nicht gestört.

Das bedeutet, daß sie sowohl vom Programm als auch vom Programmierer nach einer Unterbrechung abgefragt werden können. Nur wenn das Programm modifiziert wird, wandert der Zeiger zusammen mit den Variablen entsprechend weiter.

Den selben Effekt wie das oben erwähnte RUN haben übrigens auch die Befehle NEW, CLR und LOAD. Eine Ausnahme bildet das LOAD innerhalb eines Programms, welches den Zeiger nicht zurücksetzt. Dadurch wird ein Aneinanderhängen von mehreren Programmen samt Variablen-Weiterverwendung unter bestimmten Voraussetzungen ermöglicht.

Die Bearbeitung der Variablen durch das Basic-Programm und die daraus resultierenden Kochrezepte für den Programmierer sind in Tabelle 1 »Zwei Regeln für normale Variable« separat erläutert.

Die verschiedenen Typen der Variablen und ihre Darstellung im Speicher finden Sie im 64'er, Ausgabe 10/84, Seite 157 und noch ausführlicher in Ausgabe 11/84, Seite 124 dargestellt und erklärt.

Für diejenigen Leser, welche kein Monitor- beziehungsweise Disassembler-Programm haben oder benutzen können, ist in Tabelle 2 »Darstellung der normalen Variablen« eine kleine Anleitung gegeben, wie sie die Variablendarstellung mittels Basic anschauen können.

Adresse 47-48 ($2F-$30)
Zeiger auf die Anfangsadresse des Speicherbereichs für Feld (Arrays)

Die Darstellung und Behandlung der Felder werden in der nächsten Folge in gleicher Art und Weise behandelt, wie die normalen Variablen.

(Dr. H. Hauck/aa)

Zwei Regeln für »normale« Variable

Alle Daten, die in einem Basic-Programm nicht in Form von READ-DATA-Anweisungen vorkommen, werden als »Variable« unmittelbar nach dem Basic-Programm abgespeichert. Wir unterscheiden dabei zwei Typen:
— normale Variable
— Felder (Arrays)

Wir betrachten hier nur die »normalen« Variablen.

Sie erscheinen in dem Speicherbereich, dessen Beginn durch den Zeiger in den Zellen 45-46 und dessen Ende durch den Zeiger in 47-48 angegeben wird, in derselben Reihenfolge, in welcher sie während des Ablaufes des Basic-Programms auftreten. Wenn Basic dann auf eine der bereits definierten (und abgespeicherten) Variablen zurückgreifen soll, muß es den gesamten Variablenbereich von Anfang an absuchen, bis es den Namen der gesuchten Variablen gefunden hat. Wenn diese Variable ganz am Ende des Bereiches steht, kann dieser Suchprozeß recht lange dauern.

Regel 1:
Häufig vorkommende Variable sollen am Anfang des Variablenbereichs stehen. Das wird dadurch erreicht, daß sie als erste Variable in einem Programm »definiert« werden. Falls sie erst später im Programm gebraucht werden (aber dann häufig), werden sie trotzdem am Anfang des Programms angegeben, notfalls mit einem beliebigen Wert, der später dann keine Rolle mehr spielt und ersetzt wird. Man nennt das einen »Dummy«-Wert.

Die Felder-Variablen stehen direkt nach den »normalen« Variablen. Auch hier kann der gewiefte Programmierer Gutes tun. Wenn nämlich nach einer Definition eines Feldes später im Programm noch normale Variable dazukommen, ist natürlich zuerst kein Platz für sie da. Das Betriebssystem des Computers muß erst alle Felder-Variablen weiterschieben, bevor die Neuankömmlinge in dem dadurch erweiterten Variablenbereich gespeichert werden können. Auch das kostet unnötig viel Zeit.

Regel 2:
Alle normalen Variablen sollen als erste in einem Programm definiert werden. Wer also drauflos programmiert, sollte zumindest am Ende das Programm so umbauen, daß diese simple Regel erfüllt wird.

Darstellung der normalen Variablen

Die normalen Variablen kommen in drei Arten vor:
— ganzzahlige Variablen
— Gleitkomma-Variablen
— String-Variablen (Zeichenketten)

Der Unterschied zwischen den drei Typen ist in den Commodore-Handbüchern gut erklärt, und ich verzichte hier auf eine Wiederholung. Ich will vielmehr direkt zeigen, wie die Variablen im Speicher abgelegt sind.

Wir können den Speicher direkt sichtbar machen.

Einmal geht das in Maschinencode mittels eines Monitors beziehungsweise Disassemblers.

Zum anderen aber geht das auch in Basic und zwar mit folgendem Trick, den ich Th. und M.L. Beyer (MC 10/1983) abgeschaut habe. Er wird im folgenden allerdings nur für den C 64 beschrieben, die Methoden für den VC 20 zeige ich das nächste Mal. Der Inhalt der Darstellung ist jedoch bei beiden Rechnern identisch, nur die »Sichtbarmachung« ist verschieden.

Wir verlegen den Beginn des Basic-Variablenspeichers einfach auf den Beginn des Bildschirmspeichers. Auf diese Weise können wir zwar kein vernünftiges Programm laufen lassen, aber alle direkt eingegebenen Variablen-Definitionen werden sofort sichtbar, weil sie eben im Bildschirmspeicher stehen.

Wir erreichen die Verlegung des Speichers durch »Verbiegen« der Zeiger in den Zellen 45-46 und 47-48. Die Bedeutung dieser Zeiger ist ja im Artikel erklärt.

mit Wandervorschlägen Teil 4

Beim C 64 beginnt der Bildschirmspeicher ab 1024. In Low/High-Byte Darstellung ist das 0/4 (1024/256=4, Rest 0).
Geben Sie bitte direkt ein: POKE 46,4 :POKE 48,4

Das Low-Byte in 45 und 47 können wir weglassen, da es ja in beiden Fällen 0 ist. Wenn Sie jetzt den Bildschirm löschen, den Cursor ungefähr in die Mitte des Bildschirms fahren und wiederum direkt eingeben: VARIABLE = 3 und die RETURN-Taste drücken, dann erscheinen oben sieben Zeichen. Bitte schalten Sie mit der SHIFT- und Commodore-Taste auf den zweiten Zeichensatz um, jetzt können wir besser lesen. Aus anderen Kursen wissen Sie wahrscheinlich, daß Variable mit sieben Byte dargestellt werden. In der Tat sehen wir oben die ersten beiden Buchstaben des Variablennamens VA und fünf weitere Zeichen. Wir wollen aber systematisch vorgehen und uns zuerst die ganzzahligen Variablen anschauen.

Ganzzahl-Variable

Wiederholen Sie bitte den Vorgang (Löschen, Cursor auf Mitte, 2. Zeichensatz). Jetzt geben Sie eine Ganzzeit-Variable ein: VA%=3

Nach RETURN sehen wir als erstes Zeichen ein reverses V, dann ein reverses A, den Klammeraffen @, das kleine c und nochmals zwei @. Die beiden ersten Zeichen des Variablennamens (besteht er nur aus einem Zeichen, wird mit einer 0 aufgefüllt) werden mit ihrem ASCII-Code eingegeben, zu dem bei Ganzzahl-Variablen zur Kennzeichnung einer solchen die Zahl 128 addiert wird.

Schauen Sie in der ASCII-Tabelle (64'er, Ausgabe 7/84) nach: Das V hat 86, um 128 erhöht gibt das 214. Wir arbeiten hier aber im Bildschirmspeicher, der die Zahlen auf seine eigene Weise interpretiert, nämlich als Bildschirmcode. Der Bildschirmcode-Tabelle entnehmen wir das Zeichen für den Wert 214, und das ist das invertierte V. Für das A können Sie das selbst nachvollziehen. Also: In unserer Darstellung erkennen wir Ganzzahlvariable an den invertierten Zeichen des Namens. Das 3. und 4. Zeichen sind das High- und Low-Byte des Variablenwertes und zwar im Bildschirmcode. In unserem Beispiel der 3 ist das High-Byte 0, also der Klammeraffe @, das Low-Byte 3, also das c. Die restlichen drei Bytes sind mit 0 aufgefüllt. Wenn Sie mit dem Cursor auf die 3 fahren, es mit einer 5 überschreiben und RETURN drücken, verwandelt sich das c in ein e. Beim Überschreiben mit 255 erscheint als 4. Byte das Zeichen für den Bildschirmcode 255. Beim Überschreiben mit 257 ändern sich beide Bytes. Das 3. (High-)Byte springt auf a (=1), das 4. (Low-)Byte ebenfalls auf a. Nun, 1x256+1 = 257. Während, wie bewiesen, das Low-Byte von 0 bis 255 gehen kann, sind beim High-Byte nur Werte zwischen 0 und 127 zugelassen. Die Werte ab 128 signalisieren negative Zahlen. Probieren Sie es aus: 127x256+255=32767

Ein Überschreiben mit 32767 resultiert in einer Darstellung der Zeichen für den Bildschirmcode 127 und 255. Der Wert 32768 wird nicht mehr akzeptiert.

Negative Zahlen

Überschreiben Sie bitte die letzte Zahl mit 0. Wie zu erwarten war, sind Byte 4 und 5 jetzt 0 (Klammeraffe). Wenn Sie jetzt mit —1 überschreiben erscheint für beide Bytes das Zeichen mit dem Bildschirmcode 255. Bei —2 sehen wir die Zeichen mit den Code-Werten 255 und 254. Sie sehen also, daß die negativen Zahlen für ganzzahlige Variable sozusagen vom Ende der Tabelle her dargestellt werden, wobei die höchste negative Zahl wieder 32767 ist. Diese »Rückwärtszählung« ist bedingt durch die Methode der negativen Zahlendarstellung im Zweierkomplement. Der Platz und die Gelegenheit verbieten es mir, näher darauf einzugehen. Aber ich glaube, unser kleines Experiment hat Ihnen zumindest von der Darstellung her den Zusammenhang gezeigt. Im folgenden die Zusammenfassung der ganzzahligen Variablen.

1	2	3	4	5	6	7
Erstes	Zweites	High-	Low-			
Zeichen des Variablen-Namens (ASCII-Wert + 128)		Byte des Variablen-wertes		0	0	0

Gleitkomma-Variable

Ich hoffe, Sie verzeihen mir, wenn ich diese Darstellung heute überspringe. Sie ist nämlich nicht ganz leicht zu verstehen, und ich möchte sie lieber dann im Detail erklären, wenn wir zur Diskussion der Speicherzellen 97-101, nämlich des Gleitkomma-Akkumulators kommen. Da geht es in einem Stück. Als Vorgeschmack gebe ich jetzt nur die Zusammenfassung an.

1	2	3	4	5	6	7
Erstes	Zweites					
Zeichen des Variablen-Namens (ASCII-Wert)		Exponent + 129	Mantisse mit Genauigkeit von 32 Dualstellen, 1. Bit des 1. Bytes ist das Vorzeichen			

String-Variable

Zuerst ist es erforderlich, den Computer in den Anfangszustand zurückzusetzen. Wenn Sie einen RESET-Schalter haben, bitte diesen drücken, sonst aber aus- und einschalten. Wir geben nach Löschen des Bildschirms in der unteren Hälfte direkt ein:
ZX$="A" <RETURN>

Fahren Sie bitte jetzt mit dem Cursor auf das A und ändern den String um in BC. Nach RETURN verwandelt sich das a in das b, das 4. Zeichen ebenfalls. Die ersten beiden Zeichen sind also wieder der Name der Variable. Um zu kennzeichnen, daß es eine String-Variable ist, erscheint das 2. Zeichen des Namens invertiert. Wie oben entsteht es dadurch, daß zum ASCII-Code die Zahl 128 addiert wird. Diese Zahl wird aber wieder als Bildschirmcode interpretiert und entsprechend angezeigt (vergleichen Sie es mit den ASCII- und Bildschirmcode-Tabellen). Das dritte Zeichen gibt die Länge des Strings an, also im ersten Fall mit a (=1 im Bildschirmcode), im 2. Fall mit b (=2). Zeichen 4 und 5 geben als Low- und High-Byte die Adresse an, bei der begonnen wird, den Text des Strings zu speichern. Wir hatten die beiden Fälle:
1. ZX $ = "A"
Viertes Zeichen: (Bildschirmcode: 255) und 5. Zeichen: (Bildschirmcode 156) ergibt als Adresse 40959.
2. ZX$ = "BC"
Viertes Zeichen: (Bildschirmcode 253) und 5. Zeichen: (Bildschirmcode 156) ergibt als Adresse 40957.

Der Text der Zeichenketten wird am Ende des Arbeitsspeichers (40959 beim C 64) abgelegt und zwar von hinten nach vorn. Mit PRINT PEEK(40957);PEEK(40958);PEEK(40959) drucken wir den Inhalt dieser Speicherzellen aus und erhalten: 66 67 65. Im ASCII-Code ist das: B C A. Die Zusammenfassung für String-Variable sieht so aus:

1	2	3	4	5	6	7
Erstes	Zweites		Low-	High-		
Zeichen des Variablen-Namens		Anzahl der Zeichen des Strings	Byte der Adresse, ab welcher der Text des Strings abgespeichert ist		0	0
ASCII-Wert	ASCII-Wert + 128					