Dr. Wolfgang Ruge

Betrachtungen zum 6502-Microsoft-Basic

Der in vielen Computern der unteren Preisklasse (AIM-65, PC-100, CBM usw.) verwendete Mikroprozessor 6502 weist einen Befehlssatz auf, der der Zero-Page (das ist der Adreßbereich 0000...00FF) eine bevorzugte Stellung bezüglich Speicherplatz und Verarbeitungsgeschwindigkeit einräumt. Jeder Programmierer ist deshalb bestrebt, möglichst viele Teile seiner Programme über die Zero-Page abzuwickeln. Andererseits werden auch vom Hersteller für die wichtigsten Systemteile sehr viele Speicherplätze in der Zero-Page benötigt. Anhand des Basic-Interpreters in AIM-65 und PC-100 kann hier gezeigt werden, daß das dazu führt, daß dem Anwender auf der Zero-Page kein Platz mehr zur Verfügung steht. Eine Abhilfe ist aber möglich.

Das Microsoft-Basic belegt auf der Zero-Page beim AIM-65 die Speicherplätze 0000...00DB. In diesem Bereich liegt auch etwas unglücklich ab der Adresse 0016 der Eingabepuffer mit allein 72 Byte Länge. Der Editor benötigt die Speicherplätze 00DF...00FE, so daß dem Anwender auf der Zero-Page lediglich die Adressen DC, DD, DE und FF voll zur Verfügung stehen, wenn er mit Basic und Editor arbeiten will. Beides wird er zwar nie tun, aber bei der Verwendung eines Video-Interfaces, das allen Programmen uneingeschränkt zur Verfügung stehen muß, gibt es doch erhebliche Komplikationen.
Ohne daß die Funktion des Microsoft-Basic beeinträchtigt wird, gibt es dennoch die Möglichkeit, etwas Platz auf der Zero-Page zu bekommen.
Doch zuerst eine Bemerkung über das Abarbeiten von Basic-Programmen, die nicht an der Standardadresse hex 0211 des AIM-65 bzw. PC-100 starten.

Basic-Programme im EPROM

Das Microsoft-Basic enthält einige Vektoren, die auf den Beginn des Programmbereiches, den Beginn der Variablen, usw. weisen.
An den Speicherstellen 0073 und 0074 steht der Vektor, der auf den Beginn des Programmbereiches weist. Wird dieser Vektor nach dem Initialisieren über die Taste ⟨5⟩ auf eine andere Speicherstelle eingestellt, so ist es möglich, Programme auch von dieser Stelle aus (zum Beispiel aus einem ROM) starten zu lassen (alle anderen Vektoren müssen weiterhin ab 0211 starten!).
Erfolgt der Start von einer RAM-Adresse (zum Beispiel ab 0F00), so ist darauf zu achten, daß die Variablen und Arrays, deren Speicherbereiche dynamisch verwaltet werden, das Basic-Programm nicht überschreiben. Der Platz für die Variablen, die Strings und die Arrays ist abhängig von der Zahl und der Länge der Einträge. Das Basic stellt immer nur so viel Speicher für diese Bereiche zur Verfügung, wie tatsächlich benötigt wird. Kommt ein neuer oder längerer Eintrag hinzu, verschiebt sich der entsprechende Bereich um diese Länge. Das kann nun dazu führen, daß bei entsprechender Ausnutzung der belegte Speicher immer weiter wächst, bis entweder kein RAM mehr vorhanden ist oder Daten- bzw. Programmbereiche überschrieben werden.

Arbeitsweise eines Interpreters

Zuerst einige erläuternde Bemerkungen über die Arbeitsweise eines Interpreters. Im Gegensatz zu einem Compiler, der das in einer höheren Programmiersprache geschriebene Programm in ein Maschinenprogramm umsetzt, arbeitet der Interpreter mit dem im Klartext gespeicherten Programm. Er liest zeichenweise und versucht dabei das Gelesene zu „interpretieren". Dabei prüft er zum Beispiel, ob das Zeichen zu einer Anweisung, einem direkten Kommando, einer Variablen, einer Statementnummer usw. gehört. Der Interpreter ruft dann ein entsprechendes Unterprogramm auf, das die weitere Verarbeitung übernimmt.
Diese Arbeitsweise führt dazu, daß ein Interpreter das Programm sofort starten und ausführen kann. Stößt er während der Verarbeitung auf einen Fehler, bricht er das Programm ab.
Einige Basic-Interpreter versuchen Platz durch Komprimieren ganzer Buchstabenfolgen zu schaffen. So legt der Microsoft-Basic-Interpreter während der Programmeingabe die Zeilen zuerst im Eingabe-Puffer ab. Nach dem Ende einer Eingabe, die er über die Return-Taste erkennt, wird die Programmzeile zeichenweise aus dem Eingabepuffer in den Programmbereich übertragen. Dabei prüft der Interpreter gegen eine Tabelle, ob es sich bei der Eingabe um ein reserviertes Wort (Anweisung, Kommando, Formel usw.) handelt. Trifft das zu, ersetzt der Interpreter die Zeichenfolge durch eine 1-Byte-Konstante (Token). Diese schreibt er dann anstelle des Klartextes in die Programmzeile.

In diesem Byte ist das Bit 7 gesetzt, so daß während der Ausführung alle Befehle sofort an diesem gesetzten Bit zu erkennen sind. Der Interpreter spart dadurch viel Platz, muß jedoch bei der Eingabe (LOAD) oder Ausgabe (SAVE) auf Kassette diese Konstante vorher wieder in den ursprünglichen Klartext umwandeln, damit die Basic-Programme auch auf anderen Rechnern ablauffähig bleiben. Das kostet natürlich Verarbeitungszeit, da die entsprechenden Tabellen unter Umständen vollständig zu durchsuchen sind. Das führt dazu, daß die Blocklücken auf der Kassette erheblich größer sein müssen.
Diese Technik bedeutet aber auch, daß im Basic-Interpreter nur maximal 128 Befehle zu implementieren sind.

Lesen des Programmes

Der Interpreter liest während der Verarbeitung die Zeichen aus dem Programmspeicher mit einem Unterprogramm, das im RAM-Bereich auf der Zero-Page abge-

legt ist. Während des Einlaufes mit der Taste ⟨5⟩ wird dieses Unterprogramm a dem ROM-Bereich zusammen mit den Konstanten für den Zufallszahlengenerator in die Zero-Page ab der Adresse 00BF geladen.

Die Leseroutine, die der Interpreter über
JSR 00BF oder
JSR 00C5
anspricht, lautet:

```
00BF LESEN1 INC  LESEN2+1
            BNE  LESEN2
            INC  LESEN2+2
00C5 LESEN2 LDA  ADRESS;  Lesen
            CMP  #':'
            BEQ  LEXIT
            CMP  #' '
            BCS  LESEN1
            SEC
            SBC  #'0'
            SEC
            SBC  #$0D
00D6 LEXIT  RTS
```

Der zentrale Befehl dieses Unterprogramms ist der Befehl LDA ADRESS. Da er im RAM untergebracht ist, läßt sich durch Verändern der Speicherplätze 00C6 und 00C7 jederzeit ein anderer Speicherplatz ansprechen. Auf diese Weise arbeitet der Interpreter den Programmspeicher (das Programm) vollständig ab.
Das Unterprogramm führt beim Lesen verschiedene Funktionen aus:

1. Es liest das nächste Zeichen, indem der Programm-Pointer vor dem Lesen um 1 erhöht wird (Einsprung 00BF: LESEN1).

2. Es liest das alte Zeichen (nochmals) (Einsprung 00C5: LESEN2).

3. Es überliest Blanks (1. Vergleichsbefehl), so daß die Programme durch Einfügen von Blanks übersichtlicher zu gestalten sind, ohne daß deren Funktion beeinträchtigt wird (aber: Blanks kosten auch Speicherplatz!).

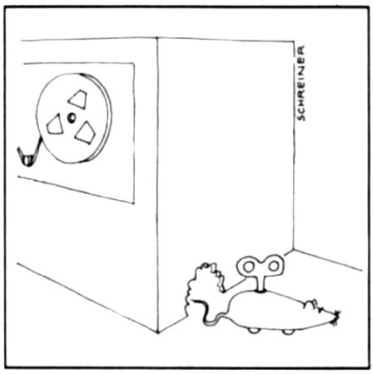
4. Bei den Ziffern wird das Carry-Flag gelöscht.

5. Am Ende einer Programmzeile oder einer Anweisung (":") ist der Akku gelöscht.

Beim genaueren Hinsehen ist zu erkennen, daß Microsoft hier eine sehr unglückliche Lösung gewählt hat: es ist nicht erforderlich, das gesamte Unterprogramm im RAM abzuspeichern. Da dort lediglich die Adressen 00C6...00C7 variiert werden, genügt es, nur den Befehl

LDA ADRESS

im RAM zu halten und den Rest aus dem ROM heraus zu bearbeiten.
Wenn in der Leseroutine der Vergleich auf Leerstellen mit dem Vergleich auf das Ende einer Anweisung (":") vertauscht wird, ergibt sich eine einfache Möglichkeit, Platz auf der Zero-Page zu bekommen. Nach dem Laden von Basic über die Taste ⟨5⟩ kehrt man sofort wie-



der in den Monitor zurück und gibt hinter dem Vergleich auf Blanks per Hand JMP CE8F ein:

```
⟨M⟩ = 00C8  C9 3A B0  0A
⟨/⟩ = 00C8  C9 20 F0  F3
⟨M⟩ = 00CC  C9 20 F0  06
⟨/⟩ = 00CC  4C 8F CE
```

Damit prüft es den Rest im ROM, und dem Benutzer stehen die acht Speicherplätze 00CF...00D6 auf der Zero-Page zur freien Verfügung.
Die Leseroutine lautet dann wie folgt:

```
LESEN1 INC  LESEN2+1
       BNE  LESEN2
       INC  LESEN2+2
LESEN2 LDA  ADRESS
       CMP  #' '          ; Blanks ?
       BEQ  LESEN1        ; ja
       JMP  $CE8F         ; weiter im
                            ROM
```

Ein Wiedereinlauf ist jetzt nur noch über die Taste ⟨6⟩ möglich, da über die Taste ⟨5⟩ jedesmal das vollständige Unterprogramm wieder in das RAM geladen und damit die Korrektur überschrieben wird. Die freiwerdenden Speicherplätze lassen sich deshalb in dieser Form leider nicht für allgemeine Systemzwecke einsetzen.

Basic-Initialisierung und RAM-Test

Beim Initialisieren soll der Interpreter prüfen, ob der vom Benutzer angegebene RAM-Speicher auch zusammenhängend zur Verfügung steht.
Zuerst fragt er nach dem Einlauf über die Taste ⟨5⟩ (Meldung „MEMORY SIZE ?"), wieviel Speicherplatz der Benutzer dem Basic zur Verfügung stellen will.
Wird eine Zahl eingegeben, übernimmt das Basic diesen Wert ohne (!) RAM-Kontrolle. Eine zu großzügige Speicherplatzvergabe (zum Beispiel 20 000 Bytes Eingabe bei einem vorhanden Speicher von nur 4 KByte) führt deshalb nicht zum Abbruch.
Wird jedoch nur die Return-Taste gedrückt, stellt Basic den gesamten ab 0211 vorhandenen und zusammenhängenden RAM-Speicher zur Verfügung.
Dazu führt Basic einen RAM-Test aus: In alle Speicherstellen schreibt es hex 55 ein. Dieser Wert wird dann im Speicher sofort um 1 Stelle nach links verschoben, so daß nach dem RAM-Test im gesamten Speicher die Konstante AA steht. (Besser wäre hier eine zweimalige „Exklusiv-Oder-Verknüpfung", da sich der Speicher danach wieder im ursprünglichen Zustand befindet.)
Dieser Test wird bei der ersten Nicht-RAM-Speicherstelle abgebrochen, spätens jedoch bei der Adresse A000 (dieser Test erfolgt im ROM bei CF18 = A0).
Beim versehentlichen Auslösen der Funktion der Taste ⟨5⟩ können dadurch ganze Basic-Programme verlorengehen. Da über den Befehl „NEW" jederzeit ein bestehendes Programm zu löschen ist, ist nicht einzusehen, warum die Funktion der Taste ⟨5⟩ nach der Erstinitialisierung noch vorhanden sein muß.
Ein erneuter Einlauf mit Löschen ließe sich durch einfaches Abfragen zu Beginn des Initialisierens vermeiden. Außerdem ergäbe sich bei einer Korrektur der Vorteil, daß die Laderoutine die freiwerdenden Speicherplätze auf der Zero-Page nicht mehr beim Initialisieren überschreibt.
Auf den Speicherplätzen 0000...0002 legt der Basic-Interpreter unmittelbar nach dem Start mit der Taste ⟨5⟩ einen Sprung auf eine Fehlermeldung ab, der nach erfolgter Initialisierung auf die Adresse B900 geändert wird. Über diese Speicherstellen läßt sich der Zustand des Systems jederzeit überprüfen: Enthalten die Adressen 0001 und 0002 den Wert 00B9, so wurde Basic bereits initialisiert.