6502 Programmieren - Teil 5#

von R. Wilde#

Nachdem ich vom letzten Artikel des Lehrganges von Euch keinerlei Reaktionen mehr erhielt, gehe ich diesmal etwas geknickt an diesen Beitrag heran. Allerdings besteht ja auch die Möglichkeit, dass der letzte Teil keine Fragen offen ließ und alle rundum zufrieden stellte.

Im letzten Artikel ging es ja endlich mit der Praxis los. Dort zeigte ich, wie man eine Speicherseite (Page) mit einem beliebigen Wert füllen kann. Wozu man das gebrauchen kann, will ich Euch heute zeigen. Wie angekündigt, werden wir die Routine vom letzten Mal etwas erweitern, um mehrere Speicherseiten zu löschen. Da wir das nicht ohne praktische Anwendung machen wollen, dachte ich mir, dass wir ein Programm entwickeln, mit dem wir den Bildschirminhalt löschen.

Dazu sind allerdings auch schon bestimmte Kenntnisse nötig. Zum Beispiel müssen wir wissen, wo der Bildschirmspeicher im Augenblick liegt und wie lang dieser ist. Problemorientiert werde ich also an geeigneter Stelle im Lehrgang Informationen zu bestimmten Speicherstellen und deren Funktion geben. Eine Übersicht über alle wichtigen Speicherstellen wird sicherlich auch in einer der nächsten Ausgaben des Magazins zu finden sein.

Die Größe des Bildschirmspeichers können wir ja recht einfach ausrechnen. Ich gehe von einem GRAPHICS 0 Bildschirm aus. Hier haben wir 24 Zeilen und jede Zeile hat 40 Zeichen. Der Bildschirminhalt besteht also aus 24 mal 40 = 960 Zeichen. 960 ist in Hexadezimal $3C0. Wir müssen also knapp 4 Pages löschen (3 Pages und $C0 = 192 Speicherstellen). Der Einfachheit halber löschen wir erst einmal 4 komplette Pages, da ja normalerweise hinter dem Bildschirmspeicher kein freier Speicher mehr liegt, sondern das ROM des XL's anfängt, stört das nicht weiter.

Zuerst wollen wir mal davon ausgehen, dass der Bildschirmspeicher bei der Adresse $BC40 beginnt. Das ist nämlich der Fall, wenn wir den BIBO-Assembler von der Diskette geladen haben.

Die Routine, mit der wir eine Page gelöscht, sprich, mit Nullen gefüllt haben, sah folgendermaßen aus:

```
00010        LDX #0
00020        LDA #0
00030 LOOP   STA $5000,X
00040        INX
00050        BNE LOOP
00060        RTS
```

Die Zeilennummern habe ich hier 5-stellig geschrieben, weil wir gleich mehr als 9 Zeilen benötigen werden. Außerdem stellt der Editor im BIBO-Assembler die Zeilennummern auch 5-stellig dar, wenn man die AUTO-Nummerierung benutzt (nach der RETURN-Taste die TAB-Taste drücken).

Um nun mehrere Pages zu löschen, brauchen wir eigentlich nur den STA-Befehl mehrfach in die Schleife einsetzen - selbstverständlich mit den Adressen der folgenden Pages. Bei 4 Pages (Speicherseiten) sähe das dann so aus:

```
00010 S      LDX #0
00020        LDA #0
00030 LOOP   STA $5000,X
00040        STA $5100,X
00050        STA $5200,X
00060        STA $5300,X
00070        INX
00080        BNE LOOP
```

```
00090        RTS
```

Also recht einfach. Da wir aber nicht den Speicherbereich von $5000 bis $53FF löschen wollen, sondern den Bildschirmspeicher, müssen wir nur die entsprechenden Adressen einsetzen.

```
00010 S      LDX #0
00020        LDA #0
00030 LOOP   STA $BC40,X
00040        STA $BD40,X
00050        STA $DE40,X
00060        STA $BF40,X
00070        INX
00080        BNE LOOP
00090        RTS
```

Damit wäre unser erstes Programm fertig, mit dem wir den Bildschirmspeicher löschen können. Da der Bildschirmspeicher aber oft an anderer Stelle im Speicher liegt, müssen wir unser Programm variabler gestalten.

Damit der Programmierer auch weiß, wo der Bildschirmspeicher liegt, legt das Betriebssystem die Anfangsadresse in bestimmten Speicherstellen ab (SAVMSC). Diese Speicherstellen liegen bei Adresse $58 und $59 (Dezimal 88 und 89). In der Speicherstelle $58 liegt das LOW-Byte der Anfangsadresse und in $59 das HIGH-Byte. Es werden ja zwei Werte benötigt, um die Adresse festlegen zu können.

Um die Anfangsadresse des Bildschirmspeichers für unser Programm zu übernehmen, kopieren wir diese in zwei Zeropage-Speicherstellen, die für eigene Programme ?frei? benutzbar sind, nämlich $E0 und $E1. Diese beiden Speicherstellen benutzen wir als indirekten Zeiger auf den Bereich des Bildschirmspeichers.

Allerdings brauchen wir für den STA-Befehl eine neue Adressierungsart, die indirekte. Schauen wir uns erst mal die Routine an:

```
00010 S      LDA $58
00020        STA $E0
00030        LDA $59
00040        STA $E1
00050        LDX #4
00060        LDY #0
00070        LDA #0
00080 LOOP   STA ($E0),Y
00090        INY
00100        BNE LOOP
00110        INC $E1
00120        DEX
00130        BNE LOOP
00140        RTS
```

Wie im dritten Teil des Lehrganges beschrieben, bezieht sich die Adressierungsart beim STA-Befehl jetzt nicht auf die Speicherstelle hinter dem Befehl, sondern auf die Speicherstelle, deren Adresse in der angegebenen Speicherstelle und der darauf folgenden abgelegt ist (Adresse in LOW-HIGH-Byte Format). In diese beiden Speicherstellen haben wir ja die Adresse des Bildschirmspeichers kopiert, also Adresse $BC40. Außerdem wird der Inhalt des Y-Registers zu der Adresse zugezählt.

Beim ersten Schleifendurchlauf wird also der Wert im ACCU in die Speicherstelle $BC40 geschrieben, Adresse $BC40 + Inhalt des Y-Registers (0).

Dann wird der Inhalt des Y-Registers um 1 herauf gezählt, und da das Ergebnis ungleich 0 ist, wird zu der Stelle verzweigt, an der der Label "LOOP" steht. Dann wird eine 0 in die Speicherstelle $BC41 geschrieben, $BC40 + Inhalt des Y-Registers (1) - ergibt $BC41. Das wiederholt sich jetzt solange, bis das Ergebnis des INY-Befehls (Increment Y-Registers) eine Null ist - 255+1.

Jetzt kommt eine wichtige Stelle. Es wird nämlich der Inhalt der Speicherstelle $E1 um 1 herauf gezählt (INC $E1). Danach zeigt unser Vektor in $E0/E1 auf die Adresse $BD40, also eine Page höher als am Anfang. Nun wird unser zweiter Schleifenzähler, das X-Register um eins herabgezählt (DEX), und solange das Ergebnis dieser Subtraktion nicht 0 ist, wird wieder nach der Stelle mit dem Label "LOOP" verzweigt.

Da das X-Register zum Anfang ja mit 4 geladen wurde, wird die äußere Schleife auch 4 mal durchlaufen und dabei die Speicherstelle $E1 jedes Mal um eins herauf gezählt.

Wie beim ersten kompletten Programm werden allerdings auch in diesem Beispielprogramm komplette 4 Pages gelöscht, also wieder von Adresse $BC40 bis $C03F. Sollte der Bildschirmspeicher nicht wie üblich direkt unter dem ROM liegen, könnten also wichtige Daten gelöscht werden. Um das zu verhindern, müssen wir unser Programm noch einmal etwas umstellen:

```
00010 S      LDA $59
00020        STA $E1
00030        LDA #0
00040        STA $E0
00050        LDY $58
00060        LDX #4
00070 LOOP   STA ($E0),Y
00080        INY
00090        BNE LOOP
00100        INC $E1
00110        DEX
00120        BNE LOOP
00130        RTS
```

Hier kopiere ich jetzt nur das HIGH-Byte der Bildschirmanfangsadresse in unseren Indirekt-Vektor und setze das LOW-Byte auf 0. Damit hätten wir z.B. die Adresse $BC00. Nun lade ich das Y-Register mit der LOW-Adresse des Bildschirmanfangs, das wäre $40. Zusammen ergibt das $BC40. Ab jetzt sieht unser Programm wieder wie vorher aus. Jetzt wird also erst ab Adresse $BC40 gelöscht und bereits bei Adresse $BCFF+1 wird auch die HIGH-Adresse erhöht und der Schleifenzähler (X-Register) um eins herabgezählt.

Damit löscht unsere Routine wirklich nur noch den Bildschirmbereich und ist trotzdem variabel, falls der GRAPHICS 0 Bildschirm woanders als direkt unterhalb des ROM-Bereichs liegt.

Übrigens: Falls Ihr Euch wundert, wofür ich am Anfang des jeweiligen Listings ein großes "S" geschrieben habe. Das soll nach dem assemblieren das Starten des Maschinenprogrammes erleichtern. Also, nach ASM braucht Ihr jetzt nicht erst RUN $4000 eingeben, sondern nur RUN S. Damit wird das Maschinenprogramm gestartet. "S" ist hier ein Label, ab dem das Programm gestartet werden kann.

Unsere Aufgabe für diesen Artikel wäre also auch gelöst und damit befinden wir uns am Ende dieses Teils. Was wir das nächste Mal machen, weiß ich noch nicht. Also lasst Euch überraschen.

Bis zum nächsten Mal Euer R. Wilde
CSM / 10.1988

---

einzeln veröffentlicht werden, bzw. anschließend als Zusammenzug als ABBUC-Buch "6502 Programmieren" erscheinen.
Koordination: Volkert Barr (volkert@nivoba.de)
Version 1.1 / 2011-01-15