6502 Programmieren - Teil 7#

von Uwe Röder#

Sie werden jetzt sicher ziemlich erstaunt darüber sein, dass ich den Assembler-Kurs weiterführe. Reinhard Wilde, der ihn bisher führte, wird in den nächsten Monaten berufsmäßig dermaßen eingespannt werden, dass er die notwendige Zeit für den Kurs nicht mehr wird aufbringen können.

Damit aber die gesamte assemblerinteressierte Leserschaft weiterhin auf Ihre Kosten kommt, wurde ich dazu überredet den Kurs zu übernehmen. Zusätzlich hat Erwin Reuss eine Serie übernommen, in der nützliche, fertige Maschinenroutinen vorgestellt werden, die in eigene Programme eingebunden werden können. Ich denke, diese beiden Serien werden allen Maschinensprach-Anfängern und Fortgeschrittenen bei der Programmierung behilflich sein.

Zum Anfang meiner Serie möchte ich Ihnen einige Buch-Tipps geben, die Ihnen mit Sicherheit sehr hilfreich sein werden. Das Buch "Programmierung des 6502" von Rodnay Zaks (Sybex-Verlag) ist der Klassiker unter den 6502-Assembler Büchern. Dieses Buch kann ich jedem ohne Einschränkung empfehlen, da sehr detailliert auf jeden Befehl und auf viele Probleme eingegangen wird. Dieses Buch dürfte übrigens in den meisten größeren Leih-Büchereien erhältlich sein, so dass man sich die Anschaffungskosten sparen kann.

Das nächste Buch, das ich Ihnen kurz vorstellen möchte, ist "Der ATARI Assembler" von Don und Kurt Inman (Idea-Verlag). Dieses Buch ist gedacht für BASIC-Aussteiger und Assembler-Einsteiger. Der Nachteil dieses Buches ist, dass es sich sehr stark am alten Atari-Assembler-Modul orientiert, das ich beispielsweise gar nicht kenne. Man sollte also seinen Assembler/Monitor schon recht gut kennen, um die Kenntnisse dieses Buches in die Praxis umsetzen zu können.

Weiter möchte ich noch darauf hinweisen, dass ein Assembler-Programmierer seinen Rechner sehr viel besser kennen muss als ein Basic-User, da man solche Befehle wie z.B. PRINT selber definieren muss. Dazu müssen aber Einsprünge in das Betriebssystem bekannt sein. Daher ist es ratsam, sich zusätzlich noch genau über den Rechner zu informieren. Das "Atari Profibuch" bietet hierzu genügend Informationen.

Für alle, die bis jetzt noch nicht eingeschlafen sind, möchte ich dann endlich konkret werden.

Da in der letzten Folge auf das Rechnen mit Zahlen eingegangen worden ist, möchte ich dort anknüpfen und nun einige weitere Befehle zum "Bearbeiten" von Zahlen vorstellen: INC, INX, INY, DEC, DEX, DEY,
ASL, LSR, ROL, ROR,
AND, ORA, EOR

Die ersten drei Befehle INC, INX und INY dienen dem Inkrementieren von Speicherstellen und X- bzw. Y-Register. Inkrementieren bedeutet ganz einfach erhöhen um den Wert 1.

Der INC-Befehl bezieht sich hierbei aus schließlich auf Speicherstellen und kann in zwei Formen verwendet werden:

INC ADR#

Der Wert der Adresse ADR wird um einserhöht und wieder in ADR abgelegt. Der ACCU, das X- und Y-Register werden sowohl hier als auch bei der gleich folgenden Adressierungsart nicht verändert. Je nach Ergebnis wird das Negativ- oder das Zero- Flag gesetzt.

Das Negativ-Flag wird immer dann gesetzt, wenn das Ergebnis größer als $7F=127 ist, das Zero-Flag, wenn das Ergebnis gleich 0 ist.

INC ADR,X#

Hier passiert im Grunde dasselbe wie bei INC ADR, nur dass die Adresse, deren Inhalt inkrementiert wird, ADR+X ist. Das heißt, dass zu der Adressangabe ADR der Inhalt des X-Registers addiert wird, um die endgültige Adressangabe zu erhalten.

Die Befehle INX und INY inkrementieren das X- oder das Y-Register. Je nach Ergebnis werden wieder Negativ- oder Zero-Flag gesetzt. Der ACCU kann nicht auf diese Weise inkrementiert werden. Um den Inhalt des ACCU zu erhöhen oder zu vermindern muss der ADC oder der SBC Befehl benutzt werden, der in der letzten Folge des Kurses erklärt wurde.

So wie man Speicherstellen inkrementieren kann, können diese natürlich auch dekrementiert werden. Die dafür nötigen Befehle sind: DEC, DEX und DEY. Die Benutzung ist völlig identisch mit der des Inkrement-Befehles, bis auf den Unterschied, dass der Inhalt der Speicherstellen oder Register um eins vermindert werden. Damit später keine Missverständnisse auftreten, möchte ich noch einmal betonen, dass nach der Logik des INC oder DEC 255+1=0 und 0-1=255 gilt!

Diese sechs vorgestellten Befehle werden sehr häufig in allen möglichen Schleifen gebraucht und sind auch schon in Programmbeispielen dieses Kurses vorgekommen, so dass ich mir ein extra Beispiel sparen kann.

Die nächsten vier Befehle ASL, LSR, ROL, ROR bewirken eine Verschiebung der Bits in einem Byte nach links oder rechts. ASL ist die Kurzform für "arithmetic shift left", was einfach nur bedeutet, dass die Bits nach links geschoben werden. Der direkte Gegenbefehl ist LSR, "logical shift right", also nach rechts schieben.

Beispiel: ASL

```
vorher:
Bit : 7 6 5 4 3 2 1 0
Wert: 0 0 1 1 0 1 0 1 Carry-Flag: egal

nachher:
Bit : 7 6 5 4 3 2 1 0
Wert: 0 1 1 0 1 0 1 0 Carry-Flag: 0
```

Es wurden alle Bits um eine Position nach links verschoben. Bit 7, das eigentlich aus dem Byte 'herausgeschoben' wird, wird ins Carry-Flag übertragen. Bit 0 ist nach einer Verschiebung nach links immer 0.

Beispiel: LSR

```
vorher:
Bit : 7 6 5 4 3 2 1 0
Wert: 1 0 0 1 1 0 0 1 Carry-Flag: egal

nachher:
Bit : 7 6 5 4 3 2 1 0
Wert: 0 1 0 0 1 1 0 0 Carry-Flag: 1
```

Alle Bits wurden hier eine Position nach rechts verschoben. Bit 7 ist nach der Verschiebung immer 0. Das 'herausgeschobene' Bit 0 findet Aufnahme im Carry-Flag.

ASL und LSR können auf den Akkumulator, auf Adressen und X-indizierte Adressen angewendet werden:

```
ASL          LSR          (ACCU)
ASL ADR      LSR ADR      (Adresse=ADR)
```

```
ASL ADR,X   LSR ADR,X   (Adresse=ADR+X)
```

Die Befehle ROL und ROR stehen für "rotate left" und "rotate right". Rotieren steht hier für eine besondere Art des Verschiebens. Die Besonderheit besteht darin, dass Bit 0 oder 7, die je nach Verschiebungsrichtung einfach gleich 0 gesetzt wurden, nun den Status des Carry-Flags zu Anfang der Verschiebung erhalten. Die Adressierungsarten entsprechen denen von ASL und LSR.

Beispiel: ROL

```
vorher:
Bit : 7 6 5 4 3 2 1 0
Wert: 0 1 1 0 1 1 0 0 Carry-Flag: 1

nachher:
Bit : 7 6 5 4 3 2 1 0
Wert: 1 1 0 1 1 0 0 1 Carry-Flag: 0
```

Beispiel: ROR

```
vorher:
Bit : 7 6 5 4 3 2 1 0
Wert: 1 1 0 0 1 1 1 0 Carry-Flag: 1

nachher:
Bit : 7 6 5 4 3 2 1 0
Wert: 1 1 1 0 0 1 1 1 Carry-Flag: 0
```

Der Sinn in Verschiebungen überhaupt liegt darin begründet, dass sich Bytes so sehr leicht untersuchen lassen. Auch Rechenfunktionen lassen sich einfach mit den Verschiebebefehlen realisieren. Eine Verschiebung nach links (ASL, ROL) entspricht einer Multiplikation mit 2, zwei Verschiebungen eine Multiplikation mit 4, 3 = 8, 4 = 16 und so weiter. Eine Division mit den gleichen Faktoren lässt sich mit den Verschiebungen nach rechts (LSR, ROR) realisieren.

Eine weitere Möglichkeit Bytes zu verarbeiten besteht in der logischen Verknüpfung von zwei Bytes. Bei den folgenden drei Verknüpfungsarten wird ein Byte des Speichers oder eine Konstante mit dem ACCU verknüpft. Das daraus folgende Ergebnis wird im ACCU abgelegt.

AND#

```
Bit  : 7 6 5 4 3 2 1 0
Wert1: 0 1 1 0 1 1 0 0
Wert2: 1 1 0 0 1 0 1 1
       ---------------
Erg. : 0 1 0 0 1 0 0 0
```

UND-Verknüpfung:
Nur Bits, die bei beiden Werten gesetzt sind, sind auch bei dem Ergebnis gesetzt.

ORA#

```
Bit  : 7 6 5 4 3 2 1 0
Wert1: 0 0 1 1 0 1 1 0
Wert2: 0 1 0 1 0 0 1 0
       ---------------
Erg. : 0 1 1 1 0 1 1 0
```

ODER-Verknüpfung:
Alle Bits, die bei dem einen oder anderen Wert gesetzt sind, sind auch bei dem Ergebnis gesetzt.

EOR#

```
Bit  : 7 6 5 4 3 2 1 0
Wert1: 0 1 1 0 0 1 1 0
Wert2: 0 1 0 0 1 1 0 1
       --------------
Erg. : 0 0 1 0 1 0 1 1
```

EXCLUSIV-ODER-Verknüpfung:
Wenn zwei Bits gleicher Nummer den gleichen Status aufweisen (0+0 oder 1+1), so ist der Status des Bits beim Ergebnis 0. Bei unterschiedlichen Belegungen (1+0 oder 0+1), ist das resultierende Bit 1.

AND, ORA und EOR werden wie folgt angewandt:

AND ADR Absolute oder ZP-Adresse AND #$.. Direkte Adressierung AND ADR,X Absolute oder ZP-Adresse AND ADR,Y (nur für 16-Bit Adressen) AND (ZP,X) ZP-Indirekte Adressierung AND (ZP),Y

Dies alles sollte erst einmal ausreichen, um Sie einen weiteren Monat intensiv mit Ihrem Rechner zu beschäftigen.

Geben Sie bitte bei irgendwelchen Problemen nicht gleich auf, sondern probieren Sie in einem solchen Fall einfach solange herum bis es klappt. Eine Erfahrung, die man auf diese Weise gemacht hat, ist nämlich bedeutend wertvoller als reines Auswendiglernen von Fakten.

In diesem Sinne bis nächsten Monat.
Ihr Uwe Röder
CSM / 2.1989

Der Artikel entstammt der Kursreihe ?6502 Programmieren? des Compy Shop Diskettenmagazins. Die Kursreihe besteht aus 14 Kursen, die im Laufe des Jahres 2011 in unregelmäßigen Abständen einzeln veröffentlicht werden, bzw. anschließend als Zusammenzug als ABBUC-Buch ?6502 Programmieren? erscheinen.
Koordination: Volkert Barr (volkert@nivoba.de)
Version 1.1 / 2011-01-23