6502 Programmieren - Teil 10#

von Uwe Röder#

Hallo und guten Tag zum zehnten Teil unseres Kurses.

In diesem Teil werden die letzten unbekannten Befehle vorgestellt, so dass ich nächsten Monat schon damit anfangen kann, auf das eigentliche Programmieren einzugehen.

Im direkten Anschluss an den Artikel vom letzten Monat werde ich zunächst den JSR-Befehl behandeln:

Wenn man ein Programm schreibt, wird man schnell feststellen, dass gewisse Befehlsfolgen immer wieder gebraucht werden. Damit man diese aber nicht immer wieder hinschreiben und damit kostbaren Speicherplatz verschwenden muss, gibt es den JSR-Befehl.

Mit JSR ADR wird ein an der Stelle ADR beginnendes Unterprogramm aufgerufen, das mit RTS enden muss. Nach Durchlaufen des Unterprogramms wird das Programm in der Zeile nach dem JSR-Aufruf weiter fortgesetzt.

Beispiel:

```
          ...
          ...
          LDX #2
          LDY #10
          JSR SETPOS
          ...
          ...
--------------------
SETPOS    STX $55
          STY $54
          RTS
--------------------
```

Die Routine SETPOS setzt den Cursor an die Position des Bildschirmes, die durch die Koordinaten in X- und Y-Register definiert ist.

In diesem Fall wird also die Position über die beiden Register X und Y an die Routine SETPOS übergeben.

Ansonsten können Parameter auch einfach an bestimmten Stellen im Speicher abgelegt werden. Ist die Position eines solchen 'Parameterspeichers' veränderlich, könnte die Startadresse des Speichers in den Registern oder in anderen dafür vorgesehenen Speicherstellen stehen. Es gibt also viele verschiedene Möglichkeiten, Daten an solche Unterprogramme weiterzugeben.

Allerdings sollte man sich hüten dies über den STACK zu tun, da dieser als oberste zwei Bytes die Rücksprungadresse des Unterprogramms enthält. Stößt der Prozessor auf den Befehl RTS, so holt er sich die obersten zwei Bytes vom STACK, addiert 1 und springt zu der Adresse.

Hat man inzwischen irgendwelche Daten auf dem STACK abgelegt, so kann dies bei einem RTS zu unerwünschten Folgen führen.

Im Übrigen können Unterprogramme weitere Unterprogramme aufrufen. Zu beachten ist dabei eigentlich nur, dass man jedes Unterprogramm auch mit einem RTS verlässt, damit die abgelegten Adress-Bytes vom STACK genommen werden. Sonst könnte es auch wieder zu unkontrollierten Sprüngen kommen.

Ein dem RTS ähnlicher Befehl ist der RTI. RTI steht für ?Return from Interrupt? und ist der Rücksprungbefehl bei den einzelnen Interrupts. Zusätzlich zur Rücksprungadresse wird hierbei noch ein drittes Byte vom Stack geholt und in das Stackregister des Prozessors kopiert.

Bei den 'gängigen' Interrupts wie zum Beispiel VBI oder DLI, benötigt nur der DLI den RTI.
Die beiden VBIs haben eigene Rücksprungvektoren, die in aller Regel einem Abbruch mit RTI vorzuziehen sind, da diese das Retten der Registerinhalte selbst vornehmen.

Des weiteren gibt es noch einige Befehle, die uns nur indirekt betreffen, nämlich die Zusatzbefehle des 65C02, der in der SPEEDY 1050 Verwendung findet. Ich führe sie hier zum Abschluss der Vollständigkeit halber mit einer kurzen Erklärung auf.

BRA Branch always#

Dies ist ein relativer Sprungbefehl, wie z.B. die Befehle BNE oder BEQ etc. Das besondere an diesem Befehl ist, dass er nicht wie alle anderen Sprungbefehle Sprungbedingung (gesetztes Flag) braucht. Er verzweigt immer an die gewünschte Stelle.

DEA, INA#

DEA dekrementiert den Akku, INA inkrementiert den AKKU. Diese beiden Befehle sind analog zu den Befehlen INX, INY, DEX, DEY.

PHX, PHY, PLX, PLY#

Wir kennen die Befehle PHA und PLA, mit denen der Inhalt des AKKU auf dem STACK abgelegt werden kann. Mit Hilfe der vier oben angegebenen Befehle können nun die Inhalte des X- und Y-Registers direkt auf dem Stapel ohne den Umweg über den Akku abgelegt werden.

STZ adr#

Die Adresse adr wird Null gesetzt. Dies ist ein recht nützlicher Befehl, da es innerhalb eines Programms sehr häufig vorkommt, Speicherbereiche oder Adressen mit Nullen zu füllen. Mit diesem Befehl kann daher Platz gespart werden.

TRB Test and reset Bit#

Der Akku wird logisch mit der Speicherstelle verknüpft (AND). Bits die sowohl beim Akku als auch in der Speicherstelle gesetzt sind werden zurückgesetzt.

TSB Test an set Bit#

Auch hier werden Akku und Speicherstelle logisch verknüpft (OR). Bits die im Akku oder in der Speicherstelle gesetzt sind werden gesetzt.

So, Sie müssten nun alle Befehle und deren Anwendung kennen. Ich meine es wäre daher an der Zeit auf andere Probleme einzugehen, zum Beispiel wie elementare Funktionen (PRINT, POSITION...) ausgeführt werden. Ab nächsten Monat wird der ganze Kurs dann praxisorientiert laufen!

Bis dann Ihr Uwe Röder
CSM / 5.1989

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