6502 Programmieren - Teil 6#

von Reinhard Wilde#

Hallo Fans. Ich begrüße Euch nach kurzer Pause zum neuen Teil des Maschinensprache-Lehrgangs. Heute wollen wir uns weitere Maschinenbefehle anschauen.

Diesmal habe ich die Rechenbefehle "ADC" und "SBC" ausgesucht.

- ADC steht für Add with Carry bzw. addiere mit Carry-Flag.
- SBC steht für Subtract with Carry bzw. subtrahiere mit Carry-Flag.

In Zusammenhang mit diesen Befehlen müssen wir uns auch die Befehle "CLD", "SED", "CLC" und "SEC" anschauen.

- CLD steht für Clear Decimal-Flag bzw. lösche Dezimal-Flag.
- SED steht für Set Decimal-Flag bzw. setze Dezimal-Flag.
- CLC steht für Clear Carry-Flag bzw. lösche Carry-Flag.
- SEC steht für Set Carry-Flag bzw. setze Carry-Flag.

Die letzten vier Befehle manipulieren sogenannte Flags. Das sind einzelne Bits im Prozessor-Status-Register.

Ist das Dezimal-Flag gesetzt, rechnet der Prozessor nicht mehr wie üblich im Hexadezimal-System, sondern mit dem im Alltag vertrauten Dezimal-System. Die nächsthöhere Stelle wird beim Addieren also schon um eins erhöht, wenn der Wert 9 überschritten wird.

Das Carry-Flag gibt unter anderem Auskunft darüber, ob beim Rechnen der Wertebereich eines Bytes (0-255) über- bzw. unterschritten wurde.

Weitere Informationen über die Flags des Prozessor-Status-Registers findet Ihr im nächsten Artikel des Lehrganges.

Nun also zum Befehl "ADC". Aus der Befehlsbezeichnung wird man unter Umständen nicht all zu schlau. Addiert den Inhalt der angegebenen Speicherstelle oder Konstante zum Wert im ACCU. War das Carry-Flag vor dem Aufruf des Befehls gesetzt, wird das Ergebnis noch um eins erhöht. Wenn während des Addierens der Wertebereich des ACCUs (0-255) überschritten wird, ist das Carry-Flag nach der Befehlsausführung gesetzt. Dadurch können auch Werte über 255 zusammengezählt werden. Das Ergebnis der Rechnung steht nach der Ausführung des "ADC"-Befehls im ACCU.

Beispiel 1: 3+4=?

```
00010     LDA #3
00020     CLC
00030     ADC #4
00040     STA $E0
00050     RTS
```

Zunächst laden wir den ACCU mit dem Wert 3. Dann muss das Carry-Flag gelöscht werden, damit ein zufällig gesetztes Carry-Flag nicht das Ergebnis verfälscht. Der "ADC"-Befehl addiert zu dem Wert 3 den Wert 4. Das Ergebnis, das sich im ACCU befindet, legen wir zum späteren Überprüfen in der Zeropage-Speicherstelle $E0 ab.

Somit wäre unser erstes Rechenprogramm fertig. Das kurze Listing wird, wie üblich, im BIBO-Assembler eingegeben und nach dem Befehl "ASM" assembliert. Nach dem Assemblieren steht das Maschinenprogramm ab der Adresse $4000 im Speicher.

Um unser Programm zu testen, gehen wir zweckmäßigerweise mit "MON" in den Maschinenmonitor des Assemblers und starten das Programm mit "4000G". Da das Ergebnis in der Speicherstelle $E0 steht, lassen wir uns den Inhalt dieser Speicherstelle wie folgt anzeigen:

E0 <RETURN>.

Wir haben aber noch eine wesentlich komfortablere Möglichkeit das Programm zu testen, nämlich mit dem "Trace"-Kommando. Dazu geben wir statt 4000G 4000T ein. Jetzt werden uns die ausgeführten Befehle einzeln angezeigt, zusammen mit allen Registerinhalten. Die folgenden Beispielprogramme solltet Ihr auch mit dem "Trace"-Befehl testen.

Zurück zum Assembler kommt Ihr durch drücken der Tasten "Q" und "RETURN" oder "RESET".

Beispiel 2: ?+?=?

```
00010     LDA #$24
00020     STA $E0
00030     LDA #$4A
00040     STA $E1
00050 ;
00060     LDA $E0
00070     CLC
00080     ADC $E1
00090     STA $E2
00100     RTS
```

Im 2. Beispiel steht die eigentliche Rechenroutine in den Zeilen 60 bis 90. Dieses mal ist unser Programm etwas variabler. In Zeile 60 wird der ACCU mit dem Wert geladen, der in der Speicherstelle $E0 steht. In Zeile 80 zählt der Prozessor den Wert in der Speicherstelle $E1 zum Wert im ACCU hinzu. Das Ergebnis befindet sich wieder im ACCU und wird in Zeile 90 zu Testzwecken in die Speicherstelle $E2 geschrieben. Damit unsere zu addierenden Werte auch feststehen, werden diese in den Zeilen 10 bis 40 in die Speicherstellen $E0 und $E1 geschrieben.

Das Semikolon in Zeile 50 hat keinen Einfluss auf das zu assemblierende Programm. Es hat die gleiche Bedeutung wie der Befehl "REM" in BASIC. Alle eventuell dahinter stehenden Zeichen (zum Beispiel Text oder erklärende Stichworte) ignoriert der Assembler. In diesem Fall dient das Semikolon der Übersichtlichkeit des Programmlistings.

Um die Aufgabe des Carry-Flags zu demonstrieren, müssen wir größere Zahlen als 255 miteinander addieren.

Beispiel 3: $290+$380=?

```
00010     LDA #$90
00020     CLC
00030     ADC #$80
00040     STA $E0
00050     LDA #2
00060     ADC #3
00070     STA $E1
00080     RTS
```

Zuerst werden die niederwertigen Anteile beider Zahlen miteinander addiert - $90 + $80 ergibt $110. Da der ACCU aber nur 8 Bit Breite hat (eben 0-255) steht anschließend nur der niederwertige Anteil des Ergebnisses im ACCU, der Übertrag für die nächsthöhere Stelle wird daher im Carry-Flag vermerkt - die Eins.

Nun werden beide höherwertigen Anteile der Zahlen addiert, und da das Carry-Flag von der vorhergehenden Rechnung gesetzt war, noch eine eins hinzuaddiert. 2+3+1=6. Der niederwertige Ergebnisanteil wird wiederum in Speicherstelle $E0 und der höherwertige Ergebnisanteil in $E1 abgelegt.

Soviel erst mal zum "ADC"-Befehl. Der "SBC"-Befehl arbeitet eigentlich nicht anders als der "ADC"-Befehl, nur dass vom ACCU subtrahiert. Das Carry-Flag wird hier entsprechend anders herum gewertet. Sollte ein Übertrag auf die nächsthöhere Stelle stattfinden, wird dies durch ein gelöschtes Carry-Flag gekennzeichnet.

Beispiel 4: 8-3=?

```
00010     LDA #8
00020     SEC
00030     SBC #3
00040     STA $E0
00050     RTS
```

Zuerst wird wieder der ACCU mit dem Wert 8 geladen. Um das Ergebnis nicht durch ein zufällig gelöschtes Carry-Flag zu verfälschen, muss dieses gesetzt werden. Dann wird vom Inhalt des ACCUs der Wert 3 subtrahiert und das Ergebnis wiederum in Speicherstelle $E0 abgelegt.

Beispiel 5: ?-?=?

```
00010     LDA #$C6
00020     STA $E0
00030     LDA #$92
00040     STA $E1
00050 ;
00060     LDA $E0
00070     SEC
00080     SBC $E1
00090     STA $E2
00100     RTS
```

Wie in Beispiel 2 nimmt der Prozessor die beiden Werte wieder aus den Speicherstellen $E0 und $E1. Inhalt von $E0 minus dem Inhalt von $E1. Das Ergebnis wird zum überprüfen in Speicherstelle $E2 gespeichert.

Beispiel 6: 25000-8000=?

```
00010     LDA #25000
00020     SEC
00030     SBC #8000
00040     STA $E0
00050     LDA /25000
00060     SBC /8000
00070     STA $E1
00080     RTS
```

Hier lasse ich mir die nieder- und höherwertigen Anteile der in dezimaler Form angegebenen Zahlen vom Assembler ausrechnen. Das Doppelkreuz weist den Assembler an, nur den niederwertigen Anteil der Zahl hinter dem Befehl einzusetzen. Der Schrägstrich bedeutet, dass nur der höherwertige Anteil eingesetzt wird. Ich rechne hier wieder mit 16-Bit breiten Zahlen, deren Werte im Bereich von 0 bis 65535 liegen können.

Die Arbeitsweise des Maschinenprogramms ist ähnlich der des Beispiels 3, nur dass hier subtrahiert wird. Zuerst wird also der niederwertige Anteil der Zahl 25000 geladen, vorsorglich das Carry-

Flag gesetzt und der niederwertige Anteil der Zahl 8000 vom Inhalt des ACCUs abgezogen. Der auftretende Übertrag wird vom Prozessor durch ein gelöschtes Carry-Flag gekennzeichnet. Anschließend wird vom höherwertigen Teil von 25000 der höherwertige Teil der Zahl 8000 und der Übertrag abgezogen. Das gesamte Ergebnis wird wieder in den Speicherstellen $E0 und $E1 abgelegt.

Die Erklärung hört sich viel komplizierter an als das Programm in Wirklichkeit ist. Dass ich zum Zwischenspeichern immer Zeropage-Speicherstellen verwendet habe ist rein willkürlich. Ich könnte genauso gut absolute oder sogar indirekte Adressierungsarten verwenden.

Damit der Artikel nicht noch länger wird, befassen wir uns auch erst im nächsten Artikel mit dem Rechnen im Dezimal-Modus. Dass ich das Dezimal-Flag für die jetzigen Rechenbeispiele nicht gelöscht bzw. nicht gesetzt habe, liegt daran, dass der Prozessor sich normalerweise im Hexadezimal-Modus befindet und daher das Dezimal-Flag nicht extra gelöscht werden muss.

Zum einwandfreien Verständnis der Rechenbefehle solltet Ihr ruhig die Werte hinter den "LDA"- und "ADC"-Anweisungen ändern. Auch die Ergebnisse könnt Ihr in anderen Speicherstellen ablegen. In der Zeropage stehen Euch dafür die Adressen zwischen $D0 und $EF zur Verfügung und im absoluten Speicherbereich empfehle ich nur die Adressen zwischen $5000 und $5FFF zu verwenden.

Viel Spaß also bis zum nächsten Mal Euer R. Wilde.
CSM / 12.1988

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