6502 Programmieren - Teil 8#

von Uwe Röder#

Hallo Freaks!

In diesem Teil unseres Kurses werde ich endlich und ausschließlich auf die Flags eingehen. Die Flags (Flaggen) dienen dazu, dem Programm gewisse Zustände des Rechners mitzuteilen, so dass das Programm genau weiß, wie die eine oder andere Operation verlaufen ist. Ohne diese Flags wäre es unmöglich Rechnungen oder Vergleiche richtig durchzuführen.

Der 6502 verfügt über 7 Flags:

Negative-, Overflow-, Break-, Decimal-, Interrupt-, Zero- und Carry-Flag;

Ich werde die Funktion der Flags nun im Einzelnen darlegen:

Das Negativ-Flag N-Flag (Vorzeichen-Flag)#

Das N-Flag entspricht in aller Regel dem Bit 7 des zuletzt bearbeiteten Bytes. Bit 7 wird in der Regel als Vorzeichen-Anzeige benutzt wenn man mit positiven und negativen Zahlen arbeiten möchte. Ist das Bit gesetzt, so ist das Byte negativ. Das N-Flag gibt in dem Fall also an, dass die Zahl negativ ist.

Auch wenn man nicht mit negativen Zahlen arbeitet, kann die Funktion des N-Flags von Nutzen sein, da es, wie gesagt, immer den Zustand von Bit 7 des zuletzt bearbeiteten Bytes angibt. Dieses Bit wird dadurch zum am leichtesten zu untersuchenden Bit.

Das N-Flag wird immer nach einem der folgenden Befehle aktualisiert:

ADC, AND, ASL, BIT, CMP, CPX, CPY, DEC,
DEX, DEY, EOR, INC, INX, INY, LDA, LDX,
LDY, LSR, ORA, PLA, PLP, ROL, ROR, RTI,
SBC, TAX, TAY, TSX, TXA, TYA

Das Overflow-Flag V-Flag (Übertrags-Flag)#

Das V-Flag gibt einen Übertrag von Bit 6 nach Bit 7 an. Dies hat wieder mit der besonderen Funktion des Bit 7 als Vorzeichen-Flag beim Rechnen mit negativen Zahlen zu tun. Wenn zum Beispiel das Ergebnis einer Rechenoperation grösser als 127 ist, so wird das Bit 7 zu der Darstellung des Ergebnisses benötigt.

Da der Prozessor ja nicht weiß, ob ich jetzt mit negativen Zahlen oder im normalen Dualsystem arbeite, setzt der Rechner das V-Flag, falls ein Übertrag von Bit 6 nach Bit 7 stattgefunden hat.

Falls ich wirklich mit durch Vorzeichen gekennzeichneten Zahlen (Zweierkomplement-Zahlen) gearbeitet habe, wird mir durch dieses Flag angezeigt, dass eventuell das Vorzeichen-Bit benutzt wurde und dass dadurch das Ergebnis verfälscht ist.

Nachfolgende Befehle haben Einfluss auf den Zustand des V-Flags: ADC, BIT, CLV, PLP, RTI, SBC

Eine Besonderheit stellt hier der BIT Befehl dar. Nach Ausführung des BIT-Befehles gibt das Overflow-Flag den Zustand des 6. Bits wieder.

Das Break-Flag B-Flag#

Das B-Flag gibt an, dass der Prozessor bei der Durcharbeitung des Programmes auf den Befehl BRK gestoßen ist.

Dies hat nichts mit der Break-Taste zu tun!!!

Ist das B-Flag durch einen BRK-Befehl gesetzt worden, so wird das Programm an der in $FFFE/ $FFFF eingetragenen Adresse fortgesetzt.

Das B-Flag kann nur durch den Befehl BRK verändert werden.

Das Dezimal-Flag D-Flag#

Wenn das D-Flag gesetzt ist, so arbeitet der Computer im Dezimal-Modus. Das bedeutet, dass 9+1 wie üblich 10 und nicht $0A ist. Das Rechenergebnis wird dann auch als 10 abgelegt. Dabei gehen vier Bits pro Ziffer drauf. Bei Berechnungen mit ADC oder SBC kann ein Byte lediglich die Werte 0 bis 99 annehmen. Ist das Ergebnis grösser als 99 so wird das Carry-Flag gesetzt.

Das D-Flag wird durch CLD, SED, PLP und RTI beeinflusst.

Das Interrupt-Flag I-Flag#

Ist das I-Flag gesetzt, so wird vom Prozessor kein Interrupt (IRQ) mehr zugelassen. Das bedeutet, dass keine I/O-Operationen mehr ausgeführt werden können, da nur POKEY und PIA einen IRQ auslösen können.

Dies ist zum Beispiel dann nützlich, wenn man Daten in den RAM Speicher unter dem Betriebssystem-ROM legen möchte, da man dazu das ROM wegschalten muss. Hätte man das ROM weggeschaltet ohne die Interrupts zu sperren, würde der Computer sich aufhängen, da die Interrupts Adressen anspringen, die im weggeschalteten ROM liegen. Ein NMI (VBI) Interrupt kann nicht gesperrt werden.

Die Befehle BRK, SEI, CLI, PLP und RTI beeinflussen dieses Flag.

Das Zero-Flag Z-Flag#

Hier kommen wir endlich zu meinem Lieblings Flag! Das Zero-Flag ist gesetzt, wenn das Ergebnis irgendeiner Operation 0 ergeben hat oder wenn bei einem Zahlenvergleich beide Zahlen gleich groß sind, also die Differenz gleich 0 ist.

Das Zero-Flag wird von folgenden Befehlen hin- und hergesetzt:

ADC, AND, ASL, BIT, CMP, CPX, CPY, DEC,
DEX, DEY, EOR, INC, INX, INY, LDA, LDX,
LDY, LSR, ORA, PLA, PLP, ROL, ROR, RTI,
SBC, TAX, TAY, TSX, TXA, TYA;

Das Carry-Flag C-Flag#

Das Carry-Flag hat nur zwei Funktionen. Die eine Funktion wurde im letzten Teil dieses Kurses besprochen als wir uns mit allen möglichen Arten des Shiftens befassten. Hierbei dient das C-Flag dazu das überschüssige oder herausgeschobene Bit aufzunehmen, damit es nicht verloren geht. Außerdem zeigt das C-Flag den Übertrag bei einer Rechenoperation an. Das heißt, wenn das Ergebnis einer Binär-Operation grösser als 255=$FF ist, wird dies durch das C-Flag angezeigt. Im Dezimal-Modus ist dies der Fall, wenn ein Ergebnis grösser als 99 ist.

Und nun wie gewohnt die Auflistung der für das C-Flag relevanten Befehle:

ADC, ASL, CLC, CMP, CPX, CPY, LSR, PLP,
ROL, ROR, RTI, SBC, SEC;

Um nun die Flags genauer zu untersuchen, gibt es zwei Möglichkeiten. Man kann über den Befehl PHP das Statusbyte, in welchem die Flags zusammengefasst sind, auf dem Stapel ablegen und mit PLA in den Akku übertragen. Jetzt kann man alle Bits genau überprüfen. Das Statusbyte setzt sich wie folgt zusammen:

Bit 0 ? Carry
Bit 1 ? Zero
Bit 2 ? Interrupt
Bit 3 ? Dezimal
Bit 4 ? Break
Bit 5 - keine Funktion
Bit 6 ? Overflow
Bit 7 ? Negativ

Für vier der Flags existieren bedingte relative Sprünge. Das heißt, dass je nach Status des zugehörigen Flags ein Sprung ausgeführt wird oder nicht. Die Sprungweite eines solchen Befehls beträgt effektiv 126 Bytes nach hinten oder 129 Bytes nach vorne.

Negative Flag#

BMI adr: Branch on minus Verzweigen falls N=1 BPL adr: Branch on plus Verzweigen falls N=0

Overflow Flag#

BVS adr: Branch on overflow set Verzweigen falls V=1 BVC adr: Branch on overflow clear Verzweigen falls V=0

Zero Flag#

BEQ adr: Branch if equal Verzweigen falls Z=1 BNE adr: Branch if not equal Verzweigen falls Z=0

Carry Flag#

BCS adr: Branch on carry set Verzweigen falls C=1 BCC adr: Branch on varry clear Verzweigen falls C=0

Auf diese Art und Weise lässt sich nach jeder Operation des Prozessors genau untersuchen ob alles wie erwünscht abgelaufen ist oder ob man Korrekturen ausführen muss.

Dazu nun zwei kurze Beispiele:

1. Inkrement einer 16 Bit Zahl

```
          INC LOW
          BNE .1
          INC HIGH
.1        RTS
```

Wenn ich eine 16 Bit Zahl um eins erhöhe, so muss ich immer wenn das Low-Byte den Wert 255 überschreitet und dadurch wieder 0 wird, das High-Byte um 1 erhöhen. Mit dem Befehl BNE .1 wird der Status des Zero-Flags untersucht. Wenn der Status gleich 0 ist, bedeutet dies, dass das Inkrementieren einen Wert ungleich 0 zum Ergebnis hatte. Dann wird der Sprung nach .1 ausgeführt

und das High-Byte nicht verändert. Ist der Status gleich 1, heißt dies dementsprechend, dass das Low-Byte die 255 überschritten hat und wieder 0 ist. Somit ist der Status des Zero-Flags 1 und der Sprung nach .1 wird nicht ausgeführt. Das Programm wird also bei INC HIGH weitergeführt, so dass in diesem Falle sowohl Low- als auch High-Byte inkrementiert werden.

2. Addition einer 8-Bit Zahl zu einer 16-Bit Zahl

```
          LDA LOW
          CLC
          ADC ZAHL
          STA LOW
          BCC .1
          INC HIGH
.1        RTS
```

Wenn die erste Addition Low + Zahl zu einem Ergebnis führt, das grösser als 255 ist, wird das Carry-Flag gesetzt. Ist dies der Fall so wird das High-Byte inkrementiert, da der Befehl BCC .1 dann nichts bewirkt.

Wäre das Ergebnis der ersten Addition kleiner oder gleich 255 gewesen und somit das Carry-Flag gleich 0, so hätte der Befehl BCC .1 einen Sprung nach .1 an das Ende der Routine bewirkt.

Dies soll soweit reichen. Im nächsten Monat werde ich dazu noch einige weitere Sachen wie z.B. die Vergleichsbefehle einführen und dann auch endlich ein Beispielprogramm abliefern.

Bis dann viel Spaß am Assembler wünscht
Uwe Röder
CSM / 3.1989

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