Wichtige Makros zum Assembler Hypra-Ass

Der beschränkte Befehlssatz des 6502-Prozessors macht Maschinenprogramme unübersichtlich und fehleranfällig. Wir zeigen Ihnen hier, wie sich mit Hypra-Ass der Maschinen-Befehlssatz durch Makros erweitern läßt. Aber nicht nur das ist möglich, Sie können sich auch Ihre eigene, ganz persönliche Sprache basteln.

Besonders dem Maschinensprache-Anfänger dürfte das Wort »Makro« gänzlich unbekannt sein. Denn weder das im C 64 implementierte noch irgendein anderes Basic kennt die Definition eines Makros. Das ist wahrscheinlich auch der Grund dafür, warum Makros nur selten angewendet werden. Sie spielen jedoch gerade beim 6502- beziehungsweise 6510-Prozessor eine wichtige Rolle. Durch Makros läßt sich nämlich der bescheiden ausgefallene Befehlssatz des Prozessors um wichtige Befehle erweitern. Es werden durch Makros prinzipiell keine neuen Maschinenbefehle geschaffen, sind aber Makros einmal definiert, lassen sie sich aufrufen wie ganz normale Maschinenbefehle. Was sind nun eigentlich Makros? Dies soll an einem kleinen Beispiel erklärt werden.

Angenommen, Sie möchten in einem Maschinenprogramm 20 verschiedene 16-Bit Adressen inkrementieren, dann müßte Ihr Programm zwangsläufig zwanzigmal folgende Befehlsfolge enthalten:

```
        INC ADRESSE
        BNE LBL
        INC ADRESSE+1
LBL
```

Diese 20 Befehlsfolgen machen aber das Programm unübersichtlich und vor allen Dingen fehleranfällig. Genau so gut ließe sich auch ein Makro mit dem Namen »INCW (adresse)« für INCWord definieren, das dann anstelle der Befehlsfolgen 20mal im Quelltext erscheint. Das Makro selbst würde wie folgt aussehen:

```
10 -.MA INCW (ADRESSE)
20 -      INC ADRESSE
30 -      BNE LBL
40 -      INC ADRESSE+1
50 -LBL
60 -.RT
```

Aufgerufen wird das Makro im Quelltext nun durch den neuen Befehl »... INCW (adresse)«.
Gefolgt von dem Makronamen und in Klammern den Übergabeparametern, die durch Kommata getrennt werden, leitet der ».MA«-Pseudo-Opcode die Definition eines Makros ein. Dies geschieht in Zeile 10. Der Pseudo-Opcode ».RT« in Zeile 60 schließt die Definition des Makros ab. Alle im Makro stehenden Label sind lokal. Das heißt, daß dem Programm außerhalb des Makros die internen Label unbekannt sind. Würde dies nicht so sein, dann würde der Assembler den zweiten Makroaufruf mit der Fehlermeldung »label twice error« ahnden. Was macht der Assembler, wenn er auf einen Makroaufruf stößt? Er assembliert in den Objektcode, wie man das erzeugte Maschinenprogramm auch nennt, die Befehlsfolgen, die im Makro definiert wurden. Das heißt, daß letztendlich im erzeugten Maschinenprogramm wieder zwanzigmal, um bei dem Beispiel zu bleiben, die oben stehenden Befehlsfolgen auftauchen.
Im Listing sind die wichtigsten Makros aufgeführt. Neben den »Befehlserweiterungen« ist noch eine interessante Gruppe von Makros definiert worden, die die strukturierte Programmierung durch »Repeat... Until«- und »While... Endwhile«-Schleifen unterstützt. Zu beachten ist jedoch, daß die Schleifen nicht verschachtelt werden dürfen. Schleifenkonstruktionen wie

```
REPEAT
   .
   .
    REPEAT
      .
      .
    UNTIL
UNTIL
```

sind veboten. Die einzelnen Makros haben folgende Wirkung:
TXY: Das Y-Register wird mit dem Inhalt des X-Registers geladen.
TYX: Das X-Register wird mit dem Inhalt des Y-Registers geladen.
PHX: Das X-Register wird auf dem Stack abgelegt.
PHY: Das Y-Register wird auf dem Stack abgelegt.
PLX: Das X-Register wird vom Stack geholt.
PLY: Das Y-Register wird vom Stack geholt.
Die folgenden vier Makros definieren einen Userstack, der an eine beliebige Stelle gelegt werden kann. Dazu muß im Hauptprogramm eine globale Variable mit dem Namen »USER« in der Zeropage angelegt werden. Anschließend muß in die Adresse, die die Variable repräsentiert, die Startadresse des Stacks geschrieben werden. Das könnte so aussehen:

```
10 -.GL USER = 3
20 -      LDA #0          ;LO-BYTE
STARTADRESSE USERSTACK
30 -      STA USER
40 -      LDA #$C0        ;HI-BYTE
STARTADRESSE USERSTACK
50 -      STA USER+1
```

Hier wurde ein Userstack angelegt, der bei Adresse $C000 beginnt. Der Stackpointer, also der Zeiger, der auf die aktuelle Stackadresse zeigt, steht in der Zeropage in den Speicherzellen 3 und 4.
PUSHA: Der Inhalt des Akkumulators wird auf dem Userstack abgelegt.
PUSHAY: Der Inhalt des Akkumulators und der Inhalt des Y-Registers werden auf dem Userstack abgelegt.
PULLA: Der Akkumulator wird vom Userstack geholt.
PULLAY: Der Akkumulator und das Y-Register werden vom Userstack geholt.
ADW (adresse): 16-Bit Addition. Der Inhalt einer beliebigen Adresse wird zum Inhalt des Akkumulators (Low-Byte) und zum Inhalt des Y-Registers (High-Byte) addiert. Das Ergebnis steht anschließend im Akkumulator (Low-Byte) und im Y-Register (High-Byte).
ADMW (adr1,adr2,summe): 16-Bit Addition. Der Inhalt von adr1 und adr1+1 wird zum Inhalt der Adresse adr2 und adr2+1 addiert und das Ergebnis in der Adresse summe und summe+1 abgelegt.
SBCW (adresse): 16-Bit Subtraktion. Der Inhalt von adresse und adresse+1 wird vom Inhalt des Akkumulators (Low-Byte) und vom Inhalt des Y-Registers (High-Byte) abgezogen. Das

Ergebnis steht anschließend im Akkumulator (Low-Byte) und im Y-Register (High-Byte).

SBCMW (adr1,adr2,diff): 16-Bit Subtraktion. Vom Inhalt adr1 und adr1+1 wird der Inhalt von adr2 und adr2+1 abgezogen. Das Ergebnis wird in der Adresse diff und diff+1 abgelegt.

INCW (adresse): Der Inhalt von adresse und adresse+1 wird inkrementiert. Das Ergebnis steht in adresse und adresse+1.

DECW (adresse): Der Inhalt von adresse und adresse+1 wird dekrementiert. Das Ergebnis steht in adresse und adresse+1.

LDAY (adresse): Der Akkumulator wird mit dem Inhalt von adresse und das Y-Register mit dem Inhalt von adresse+1 geladen.

STAY (adresse): Der Inhalt des Akkumulators wird nach adresse und der Inhalt des Y-Registers nach adresse+1 geschrieben.

LDAYI (wert): Der Akkumulator und das Y-Register wird mit »wert« unmittelbar geladen. Dabei steht das Low-Byte im Akkumulator und das High-Byte im Y-Register.

Die folgenden Makros unterstützen die strukturierte Programmierung.

REPEAT, EXITREPEAT, UNTIL (übergabe,bedingung): Die Schleife wird so lange fortgesetzt, bis die Speicherzelle »übergabe« den Wert »bedingung« enthält. Beispiel:

```
10 -            LDX #255
20 -            ... REPEAT
30 -            DEX
40 -            STX $FB
50 -            ... UNTIL ($FB,0)
```

Das X-Register wird solange dekrementiert, bis es den Wert »0« enthält.

WHILE (übergabe,bedingung) , EXITWHILE, ENDWHILE: Die Schleife wird solange fortgesetzt, bis der Inhalt der Speicherzelle »übergabe« gleich »bedingung« ist. Beispiel:

```
10 -            LDX #255
20 -            ... WHILE ($FB,0)
30 -            DEX
40 -            STX $FB
50 -            ... ENDWHILE
```

Solange der Inhalt der Speicherzelle $FB ungleich Null ist, wird das X-Register dekrementiert.

(ah)

```
READY.

10   -;*********************************
20   -;*   WEITERE VERSCHIEBEBEFEHLE   *
30   -;*********************************
40   -;
50   -; X -> Y
60   -.MA TXY
70   -            PHA
80   -            TXA
90   -            TAY
100  -            PLA
110  -.RT
120  -;
130  -; Y -> X
140  -.MA TYX
150  -            PHA
160  -            TYA
170  -            TAX
180  -            PLA
190  -.RT
200  -;
210  -; X-REGISTER AUF DEN STACK
220  -.MA PHX
230  -.EQ RETTEN = $FC
240  -            STA RETTEN
250  -            TXA
260  -            PHA
270  -            LDA RETTEN
280  -.RT
290  -;
300  -; Y-REGISTER AUF DEN STACK
310  -.MA PHY
320  -.EQ RETTEN = $FC
330  -            STA RETTEN
340  -            TYA
350  -            PHA
360  -            LDA RETTEN
370  -.RT
380  -;
390  -; X-REGISTER VOM STACK HOLEN
400  -.MA PLX
410  -.EQ RETTEN = $FC
420  -            STA RETTEN
430  -            PLA
440  -            TAX
450  -            LDA RETTEN
460  -.RT
470  -;
480  -; Y-REGISTER VOM STACK HOLEN
490  -.MA PLY
500  -.EQ RETTEN = $FC
510  -            STA RETTEN
520  -            PLA
530  -            TAY
540  -            LDA RETTEN
550  -.RT
560  -;
570  -; DEN AKKU AUF USERSTACK
580  -.MA PUSHA
590  -            ... PHY
600  -            LDY #0
610  -            STA (USER),Y
620  -            ... DECW(USER)
630  -            ... PLY
640  -.RT
650  -;
660  -;AKKU UND Y-REGISTER AUF USERSTACK
670  -.MA PUSHAY
680  -            PHA
690  -            TYA
700  -            ... PUSHA
710  -            PLA
720  -            ... PUSHA
730  -.RT
740  -;
750  -; AKKU VON USERSTACK
760  -.MA PULLA
770  -            ... INCW(USER)
780  -            ... PHY
790  -            LDY #0
800  -            LDA (USER),Y
810  -            ... PLY
820  -.RT
830  -;
840  -;AKKU UND Y-REGISTER VON USERSTACK
850  -.MA PULLAY
860  -            ... PULLA
870  -            TAY
880  -            ... PULLA
890  -.RT
900  -;
910  -;*********************************
920  -;*       16-BIT BEFEHLE            *
930  -;*********************************
940  -;
950  -; A/Y + ADRESSE = A/Y AKKU=LO
960  -.MA ADW (ADRESSE)
970  -            CLC
980  -            ADC ADRESSE
990  -            PHA
1000 -            TYA
1010 -            ADC ADRESSE+1
1020 -            TAY
1030 -            PLA
1040 -.RT
1050 -;
1060 -; ADIERE ADR1 + ADR2 = SUMME
1070 -.MA ADMW (ADR1,ADR2,SUMME)
1080 -            PHA
1090 -            CLC
1100 -            LDA ADR1
1110 -            ADC ADR2
1120 -            STA SUMME
1130 -            LDA ADR1+1
1140 -            ADC ADR2+1
1150 -            STA SUMME+1
1160 -            PLA
1170 -.RT
1180 -;
1190 -; A/Y - ADRESSE = A/Y AKKU=LO
1200 -.MA SBCW (ADRESS)
1210 -            SEC
1220 -            SBC ADRESSE
1230 -            PHA
1240 -            TYA
1250 -            SBC ADRESSE+1
1260 -            TAY
1270 -            PLA
1280 -.RT
1290 -;
1300 -; ADR1 - ADR2 = DIFFERENZ
1310 -.MA SBCMW (ADR1,ADR2,DIFF)
1320 -            PHA
1330 -            SEC
1340 -            LDA ADR1
1350 -            SBC ADR2
1360 -            STA DIFF
1370 -            LDA ADR1+1
1380 -            SBC ADR2+1
1390 -            STA DIFF+1
1400 -            PLA
1410 -.RT
1420 -;
1430 -; ADRESSE = ADRESSE + 1
1440 -.MA INCW (ADRESSE)
1450 -            PHA
1460 -            LDA ADRESSE
1470 -            CLC
1480 -            ADC #1
1490 -            STA ADRESSE
1500 -            LDA ADRESSE+1
1510 -            ADC #0
1520 -            STA ADRESSE+1
1530 -            PLA
1540 -.RT
1550 -;
1560 -; ADRESSE = ADRESSE - 1
1570 -.MA DECW (ADRESSE)
1580 -            PHA
1590 -            LDA ADRESSE
1600 -            SEC
1610 -            SBC #1
1620 -            STA ADRESSE
1630 -            LDA ADRESSE+1
1640 -            SBC #0
1650 -            STA ADRESSE+1
1660 -            PLA
1670 -.RT
1680 -;
1690 -; ADRESSE -> A/Y AKKU=LO
1700 -.MA LDAY (ADRESSE)
1710 -            LDY ADRESSE+1
1720 -            LDA ADRESSE
1730 -.RT
1740 -;
1750 -; A/Y -> ADRESSE AKKU=LO
1760 -.MA STAY (ADRESSE)
1770 -            STA ADRESSE
1780 -            STY ADRESSE+1
1790 -.RT
1800 -;
1810 -; WERT=16BIT -> A/Y
1820 -.MA LDAYI (WERT)
1830 -            LDA #<(WERT)
1840 -            LDY #>(WERT)
1850 -.RT
1860 -;*********************************
1870 -;*  BEFEHLE ZUR STRUKTURIERTEN   *
1880 -;*       PROGRAMMIERUNG          *
1890 -;*********************************
1900 -;
1910 -.MA REPEAT
1920 -ACE1        .GL ACE0=ACE1
1930 -.RT
1940 -;
1950 -.MA EXITREPEAT
1960 -            JMP BCE0
1970 -.RT
1980 -;
1990 -.MA UNTIL (UEBERGABE,BEDINGUNG)
2000 -            PHA
2010 -            LDA UEBERGABE
2020 -            CMP #BEDINGUNG
2030 -            BEQ LBL1
2040 -            PLA
2050 -            JMP ACE0
2060 -LBL1        PLA
2070 -.GL BCE0=LBL1
2080 -.RT
2090 -;
2100 -.MA WHILE (UEBERGABE,BEDINGUNG)
2110 -CCE0        .GL CCE0=CCE0
2120 -            PHA
2130 -            LDA UEBERGABE
2140 -            CMP #BEDINGUNG
2150 -            BNE LBL1
2160 -            JMP CCE1
2170 -LBL1        PLA
2180 -.RT
2190 -;
2200 -.MA EXITWHILE
2210 -            PHA
2220 -            JMP CCE1
2230 -.RT
2240 -;
2250 -.MA ENDWHILE
2260 -            JMP CCE0
2270 -CCE1        .GL CCE1=CCE1
2280 -            PLA
2290 -.RT
```

Listing. Die wichtigsten Makros zum Assembler »Hypra-Ass«