Assembler-Bedienung leichtgemacht

Teil 2

Diesmal möchte ich Ihnen zeigen, wie Sie Ihre Quelltexte übersichtlicher gestalten können. Dadurch sinkt die Fehleranfälligkeit und der »Frust« bei der Programmgestaltung.

Im ersten Teil dieses Artikels wurde Grundsätzliches zur Bedienung eines Assemblers gesagt und die wichtigsten Pseudo-Opcodes behandelt. Mit diesen Pseudo-Opcodes konnte schon jeder Quelltext assembliert werden. Was jedoch noch nicht funktionierte, waren sämtliche Ausgabefunktionen, die Behandlung von Makros und Sinn und Anwendung der bedingten Assemblierung. Mit dieser Thematik wollen wir uns dieses Mal beschäftigen, so daß jeder vernünftig mit einem Assembler, speziell Hypra-Ass, arbeiten kann. Wenden wir uns zunächst den Ausgabebefehlen zu, um die Behandlung der Pseudo-Opcodes abzuschließen.

.LI lfn,dn,ba: Sendet ein formatiertes Assembler-Listing unter der Filenummer »lfn« an das Gerät »dn« mit der Sekundäradresse »ba«. Es ist jedoch zu beachten, daß das Assembler-Listing erst nach dem ».LI«-Pseudo ausgegeben wird. Wollen Sie ein komplettes Assembler-Listing haben, muß der ».LI«-Pseudo zwangsläufig in der ersten Zeile des Quelltextes stehen. Die Parameter hinter dem ».LI«-Pseudo-Opcode entsprechen denen des OPEN-Befehls. So ist es auch möglich, mit ».LI 2,8,2,"test,U,W"« ein Assembler-Listing auf eine User-Datei umzuleiten. Um den ».LI«-Pseudo-Opcode in den Griff zu bekommen, fügen Sie in den Beispiel-Quelltext (Listing 1) aus der letzten Ausgabe folgende Zeile ein:

```
10 -.LI 1,3
```

Diese Zeile beziehungsweise der .LI-Befehl gibt das Assembler-Listing auf dem Bildschirm aus. Da keine Sekundäradresse angegeben wurde, setzt der Computer automatisch eine »0« an diese Stelle. Würde man statt der »3« als Geräteadresse eine »4« einsetzten, würde das Assemblerlisting auf einen angeschlossenen Drucker ausgegeben. Auch das Umlenken auf die Datasette mit ».LI 1,1,2,"name"« wäre denkbar.

.SY lfn,dn,ba: Sendet eine sortierte Liste aller im Quelltext auftretenden Label und Variablen vom Typ ».EQ« (Symboltabelle) unter der Filenummer »lfn« an das Gerät »dn« mit der Sekundäradresse »ba«. Weiterhin gilt das gleiche wie beim ».LI«-Pseudo, bis auf die Lage des ».SY«-Pseudos. Der nämlich kann an einer beliebigen Stelle im Quelltext stehen. In diesem Zusammenhang sei noch eine weitere interessante Möglichkeit erwähnt. Möchten Sie zum Bei-

spiel die absolute Adresse eines beliebigen Labels oder den Wert einer Variablen wissen, brauchen Sie nicht nach dem Assemblieren mit einem Monitor (zum Beispiel SMON) das erzeugte Maschinenprogramm zu durchforsten. Geben Sie ganz einfach ein »PRINT name <RETURN>«. Sie erhalten schlagartig die Adresse des Labels »name« oder den Wert der Variablen. Die ausgegebenen Adressen haben zwar dezimales Format, aber mit einem Monitor läßt sich dieses schnell in das entsprechende hexadezimale Format umrechnen.

Leider hat sich in diesem Programmteil noch ein kleiner Fehler eingeschlichen. Dieser Fehler tritt extrem selten auf. Wird versucht mit dem Editorbefehl »/!!« eine sortierte Symboltabelle auszugeben, stürzt Hypra-Ass ab, wenn diese genau 36, 73, 109 (und so weiter) Label oder Variablen enthält. Der Fehler liegt in den Speicherzellen $1EB8 bis $1EBB. Hier wurden zwei Branch-Befehle vertauscht. Es muß richtig lauten:

1EB8 90 D0 BCC 1EA8
1EBA D0 04 BNE 1EC0

Diese Änderung kann direkt mit einem Maschinensprachemonitor (zum Beispiel SMON) in die entsprechenden Speicherstellen geschrieben werden. Anschließend läßt sich Hypra-Ass vom SMON aus mit »S"HYPRA-ASS"0801 1FD8« speichern.

.OB "name,P,W": Sendet den erzeugten Maschinencode zum Floppy-Laufwerk und speichert ihn dort unter dem Namen »name«. Dazu muß dieser Pseudo in der ersten beziehungsweise zweiten Zeile (nach dem ».LI«-Pseudo) stehen. Ich möchte noch darauf hinweisen, daß es unmöglich ist, ein formatiertes Assembler-Listing und gleichzeitig den erzeugten Maschinencode zum Floppy-Laufwerk zu senden. Denn dazu müßten zwei Files zum Schreiben geöffnet werden, und das geht nicht. Wurde der ».OB«-Pseudo richtig eingegeben, kann das erzeugte Maschinenprogramm direkt mit »LOAD "name",8,1« geladen und mit dem SYS-Befehl gestartet werden. Interessant ist bei diesem Pseudo, daß sich auch Maschinenprogramme erzeugen lassen, die bei $801 beginnen, also in dem Bereich liegen, in dem Hypra-Ass steht. Wird ein solches Programm geladen, ist natürlich klar, daß Hypra-Ass dabei zerstört wird. Zu dem ».OB«-Pseudo gehört unmittelbar ein weiterer Pseudo ».EN«. Dieser muß am Schluß des Quelltextes stehen und zwar ohne jegliche Parameter. Durch ihn wird das mit dem ».OB« geöffnete File geschlossen. Fehlt er, wird das durch ein Sternchen vor dem Filetyp im Directory gekennzeichnet. Übrigens muß der .EN-Pseudo am absoluten Ende des Quelltextes stehen. Daß heißt, sollte der Quelltext aus mehreren Teilen bestehen, die jeweils mit dem ».AP«-Pseudo, zu dem wir noch kommen, verkettet werden, muß der .EN-Pseudo in der letzten Zeile des letzten Quelltextteils stehen. Um auch diese beiden Pseudos in den Griff zu bekommen, fügen Sie bitte in den Beispiel-Quelltext (letzte Ausgabe) folgende Zeilen ein:

10 -.OB "BEISPIEL,P,W"
620 -.EN

Nach dem Assemblieren listet der Editorbefehl »/I« das Directory. Keine Angst, der Quelltext geht dabei nicht verloren. Sie werden feststellen, daß sich das Beispielprogramm »BEISPIEL« wie zu erwarten auf der Diskette befindet. Dieses kann nun mit LOAD"BEISPIEL",8,1 geladen und durch SYS 9*4096 aktiviert werden. Bisher existiert leider noch keine Option, den Objektcode zur Datasette zu schicken.

.AP "name": Dieser Pseudo-Opcode dient zum Verketten von einzelnen Quelltextteilen. Häufig kommt es vor, daß ein Quelltext nicht mehr in den zur Verfügung stehendenden, freien Speicherplatz paßt. In diesem Fall läßt sich der Quelltext splitten. Natürlich sollte man darauf achten, daß der Quelltext nicht willkürlich zerhackt wird. Zusammenhängende Programmteile sollten schon als ein Quelltextteil gespeichert werden. Um spätere Verwirrung zu vermeiden, sollte auch der Name, der dem Quelltextteil zugeordnet wird, einigermaßen sinnvoll sein. Damit Sie auch diesen Befehl kennenlernen, ist das Beispiel-Programm (letzte Ausgabe) unter Hypra-Ass zu laden. Anschließend löschen Sie mit dem Editorbefehl »/D 0-360« den ersten Teil des Quelltextes und speichern die verbleibenden Zeilen unter dem Namen »TEIL2.SRC« auf Diskette. Nun ist der Beispiel-Quelltext ein zweites Mal zu laden. Dieses Mal löschen Sie mit dem Editorbefehl »/D 360-« den letzten Teil des Quelltextes. Bevor die verbleibenden Zeilen unter dem Namen »TEIL1.SRC« gespeichert werden, ist noch folgende Zeile erforderlich:

360 -.AP "TEIL2.SRC"

Diese Zeile teilt dem Assembler mit, daß noch Quelltext folgt. Befindet sich der 1. Teil auf der Diskette und im Speicher, kann der Assembler mit RUN gestartet werden. Nun wird zuerst Pass 1 abgearbeitet, und zwar nur im ersten Quelltextteil. Anschließend lädt der Assembler den zweiten Quelltextteil nach, bearbeitet auch hier Pass1, lädt danach wieder den 1. Quelltextteil, erzeugt von diesem Teil das Maschinenprogramm und lädt schließlich den zweiten Quelltextteil, um auch von diesem das Maschinenprogramm zu erzeugen. Haben Sie den oben stehenden ».OB«-Pseudo wie beschrieben eingefügt, werden beide Maschinenprogrammteile natürlich unter einem Namen nämlich »Beispiel« auf Diskette gespeichert. Ist das nicht der Fall, wird der erste Maschinenprogrammteil nach $9000 gelegt und der zweite unmittelbar an den ersten gehängt.

Das ist schon alles, was zu den »normalen« Pseudo-Opcodes zu sagen wäre. Hypra-Ass enthält aber noch einige Pseudo-Opcodes, die Makros definieren, aufrufen und bedingt assemblieren. Damit Sie auch diese sicherlich sinnvollen Pseudo-Opcodes beziehungsweise Steueranweisungen an den Assembler sinnvoll einsetzen können, möchte ich darauf näher eingehen.

Der Umgang mit Makros

Die meisten Hobby-Programmierer und besonders Maschinenspracheanfänger vermeiden den Gebrauch von Makros. Der Grund dafür wird wohl der sein, daß diese Option eben nur in Maschinensprache existiert und andere Sprachen, wie zum Beispiel Basic, diesen Begriff nicht kennen. Dabei ist der Gebrauch von Makros jedem Programmierer, egal ob Anfänger, Fortgeschrittener oder Profi zu empfehlen. Denn durch sie werden Maschinenprogramme beziehungsweise die dazugehörigen Quelltexte übersichtlicher und dadurch bedingt sinkt natürlich auch die Fehleranfälligkeit der erstellten Programme. Um aber mit Makros arbeiten zu können, muß man erst einmal wissen, was das überhaupt ist. Ein Makro läßt sich grob gesehen mit einem Unterprogramm vergleichen, dem ein Name sprich Label zugewiesen wird. Das Unterprogramm selbst wird bekanntlich mit einem RTS abgeschlossen und läßt sich vom Hauptprogramm aus mit dem Maschinenbefehl »JSR name« aufrufen. Der Unterschied zwischen Unterprogramm und Makro ist nun der, daß nicht zu einem definierten Unterprogramm verzweigt, sondern das Unterprogramm selbst an die

Stelle des Aufrufs assembliert wird. Ein Makro hat allerdings gegenüber einem Unterprogramm gewaltige Vorteile. So sind alle im Makro auftretenden Label und dort definierte Variable lokal. Das heißt, daß dem Hauptprogramm die im Makro auftretenden Label und Variablen unbekannt sind. Daraus folgt, daß Label und Variable in Makros und im Hauptprogramm identische Namen haben dürfen. Ein weiterer Vorteil gegenüber Unterprogrammen ist der, daß in einem Makro Übergabeparameter definiert werden können, die eine Schnittstelle zum Hauptprogramm bilden. Außerdem wird das Programm durch intensive Anwendung von Makros wesentlich schneller, da sämtliche JSR-Befehle, die sehr viel Zeit in Anspruch nehmen, entfallen.

Zur Definition eines Makros existieren bei Hypra-Ass zwei Pseudo-Opcodes.

.MA name (par1,par2,par3): Durch den ».MA«-Pseudo wird dem Assembler mitgeteilt, daß die nachfolgenden Maschinenbefehle ein Makro mit dem Namen »name« definieren. Die in Klammern eingefaßten Übergabeparameter »par1, par2, par3«, zu denen ich später noch einiges sagen werde, müssen durch Kommata getrennt werden. Bei den Übergabeparametern selbst kann es sich entweder um Label, Variablen oder um absolute 16-Bit-Adressen oder Werte handeln.

.RT: Dieser Pseudo-Opcode schließt eine Makrodefinition ab. Er kann direkt mit dem RTS-Befehl verglichen werden, der ein Unterprogramm beendet und zurück ins Hauptprogramm beziehungsweise in ein übergeordnetes Unterprogramm verzweigt. Im Gegensatz zum RTS-Befehl findet bei dem .RT-Pseudo keine Programmverzweigung statt, sondern der Assembler selbst verzweigt, ähnlich wie beim RTS-Befehl, an die Stelle des Quelltextes, an der das Makro aufgerufen wurde.

Wie ein Makro aufgerufen wird, zeigt der nun folgende Pseudo-Opcode.

...name (par1,par2,par3): Dieser Pseudo-Opcode, der letztendlich aus zwei Punkten besteht (der erste Punkt leitet ja bekanntlich einen Pseudo-Opcode ein), ruft ein Makro mit dem Namen »name« auf und übergibt die Parameter »par1, par2, par3«, die genauso wie bei der Makrodefinition jeweils durch Kommata getrennt und in Klammern eingefaßt sein müssen. Und jetzt kommt der springende Punkt. Bei der Definition des Makros beziehen sich die Übergabeparameter auf die im Makro definierten Label und Variablen. Bei einem Aufruf beziehen sich diese Übergabeparameter aber auf die im Hauptprogramm definierten Label und Variablen.

Um ein Gefühl und ein Verständnis für die Übergabeparameter zu bekommen, soll kurz erklärt werden, wie der Assembler die Parameter behandelt. Angenommen, in einer Makrodefinition wird ein Parameter »par1« benuzt und im Makroaufruf selbst steht an seiner Stelle nicht par1 sondern »BELIEBIG«, dann setzt der Assembler diese beiden Ausdrücke gleich. Das heißt er führt folgende Operation aus:
par1 = BELIEBIG

Daraus folgt, daß beide Variablen nach dem Makroaufruf den gleichen Wert enthalten. Als Beispiel möchte ich die 16-Bit-Addition hernehmen. Das Makro soll aber so gestaltet werden, daß kein Arbeitsregister (A,X,Y) zerstört wird.

```
10 —.MA ADDW (ADR1,ADR2,SUMME)
20 —        PHA
30 —        LDA     ADR1
40 —        CLC
50 —        ADC     ADR2
60 —        STA     SUMME
70 —        LDA     ADR1+1
80 —        ADC     ADR2+1
90 —        STA     SUMME+1
100 —       PLA
110 —.RT
```

In diesem Makro wird also zum Inhalt einer Adresse2 und Adresse2 +1 der Inhalt einer Adressel und Adressel+1 addiert und das Ergebnis in der Adresse SUMME und SUMME+1 gespeichert. Um das Makro aufzurufen, ist das Programm wie folgt zu ergänzen:

Dieses kleine Programm addiert die beiden Zahlen $25E5 und $43A8 und gibt das Ergebnis im dezimalen Format auf dem Bildschirm aus, das in diesem Fall 27021 ist. Um das zu testen, sind die Zeilen 10 bis 270 abzutippen, nachdem Hypra-Ass geladen und gestartet wurde. Ist der Quelltext assembliert, müßte, wenn kein Fehler aufgetreten ist, mit SYS 9*4096 das Ergebnis »27021« auf dem Bildschirm erscheinen. Um die Wirkung und die Arbeitsweise von Makros zu verstehen, spielen Sie mit dem Quelltext ruhig ein wenig herum. Ändern Sie einfach die Label und Zahlen ab, kaputtmachen können Sie nichts.

Hypra-Ass enthält noch eine weitere, sehr angenehme Eigenschaft. Er kann nämlich bedingt assemblieren. Was das ist und was man damit machen kann, soll hier detalliert beschrieben werden. Denn gerade im Zusammenhang mit Makros spielt die bedingte Assemblierung eine wichtige Rolle. Mit ihrer Hilfe kann bei einem Makroaufruf ein weiterer Parameter übergeben werden, der bestimmt, ob gewisse Quelltextteile innerhalb des Makros ins Hauptprogramm assembliert werden sollen oder nicht. Dadurch läßt sich in Abhängigkeit des Übergabeparameters ein Makro definieren, das verschiedene Aufgaben erfüllt. Fangen wir zunächst mit einem kleinen Beispiel an, damit Sie die Wirkung der bedingten Assemblierung innerhalb eines Makros kennenlernen. Dazu soll wieder die obenstehende 16-Bit-Addition herhalten, in die wir eine bedingte ».IF«-Abfrage einbauen wollen. Mit dieser bedingten ».IF«-Abfrage soll nun bestimmt werden, ob der Akku bei einem Makroaufruf erhalten bleiben oder zerstört werden soll, beziehungsweise ob das erzeugte Maschinenprogramm den »PHA«- und den »PLA«-Befehl enthalten soll oder nicht. Dazu sind zunächst die Zeilen 10 und 240 abzuändern, denn wir benötigen ja noch einen weiteren Übergabeparameter. Nennen wir diesen Übergabeparameter »RETTEN«. Dann sind die beiden Zeilen wie folgt zu ändern:
10 -.MA ADW (ADR1,ADR2,SUMME, RETTEN)
240 - ... ADW (SUMMAND1, SUMMAND2,ERGEBNIS,x)

Das »x« ist zu ersetzen durch eine »1« für Akku retten oder für eine beliebige andere Zahl für Akku zerstören. Als nächstes ist noch die bedingte ».IF«-Abfrage in den Quelltext einzubauen. Bevor das geschieht noch einiges zur ».IF«-Abfrage selbst, die etwas anders funktioniert, als Sie es vom Basic her ge-

Fortsetzung auf Seite 173

```
120 —.BA $9000
130 —.EQ SUMMAND1  = $FB
140 —.EQ SUMMAND2  = $FD
150 —.EQ ERGEBNIS  = 2
160 —       LDA    #$E5
170 —       LDX    #$25
180 —       STA    SUMMAND1
190 —       STX    SUMMAND1+1
200 —       LDA    #$A8
210 —       LDX    #$43
220 —       STA    SUMMAND2
230 —       STX    SUMMAND2+1
240 —       ...    ADDW (SUMMAND1,
                   SUMMAND2,
                   ERGEBNIS)
250 —       LDX    ERGEBNIS
260 —       LDA    ERGEBNIS+1
270 —       JMP    $BDCD ;AUSGABE
                   AUF BILDSCHIRM
```

Fortsetzung von Seite 170

wohnt sind. Trifft das Basic-Programm auf eine Zeile, in der eine »IF«-Abfrage steht, wird zunächst überprüft, ob diese »IF«-Abfrage wahr oder falsch ist. Ist sie falsch, wird direkt zur nächsten Zeile verzweigt. Das ist bei einem Maschinenprogramm beziehungsweise dem dazugehörigen Quelltext nicht möglich, denn es darf ja pro Zeile immer nur ein Maschinenbefehl stehen. Deshalb muß bei der bedingten ».IF«-Abfrage noch ein weiterer Pseudo-Opcode vorhanden sein, der dem Assembler das Ende dieser ».IF«-Abfrage mitteilt. Dieses ist der ».EI«-Pseudo für »ENDIF«, der eine ».IF«-Abfrage abschließt. Bei Hypra-Ass existiert noch ein dritter Pseudo im Zusammenhang mit ».IF«-Abfragen der ».EL«-Pseudo für »ELSE«. Die Struktur innerhalb eines Programms sieht dann so aus:

```
.IF A !=! 1
.
.EL
.
.EI
```

Es handelt sich hier um eine »entweder oder«-Assemblierung. Entweder wird der Quelltextteil zwischen ».IF« und ».EL« (A = 1) oder zwischen ».EL« und ».EI« (A < > 1); in das Maschinenprogramm assembliert.

Doch nun wieder zurück zum Beispiel. Haben Sie die Zeilen 10 und 240 geändert, kann die ».IF«-Abfrage einbaut werden. Dazu sind folgende Zeilen zu ergänzen:

```
15 -.IF RETTEN !=! 1
25 -.EI
95 -.IF RETTEN !=! 1
105 -.EI
```

Bei RETTEN darf es sich nur um eine Variable handeln und nicht um eine Adresse oder Label.

Hypra-Ass enthält noch einige andere Pseudo-Opcodes zur bedingten Assemblierung:

.ON ausdruck,sprungziel: Ist der Ausdruck hinter .ON wahr, wird zur Zeile »sprungziel« verzweigt. Beispiel:

```
100 -.ON A !=! 1, 200
```

Ist A = 1, wird der Quelltextteil, der zwischen der Zeilennummer 100 und 199 steht ignoriert, also nicht assembliert.

.GO sprungziel: Unbedingter Sprung zur Zeile »sprungziel«. Beispiel:

```
100 -.GO 200
```

Der Quelltext, der zwischen der Zeilennummer 100 und 199 steht, wird auf keinen Fall assembliert. Daraus folgt natürlich, daß dieser bedingte Pseudo-Opcode nur in Verbindung mit einer ».IF«-Abfrage sinnvoll ist. Denn was nutzt ein Quelltext, der aufgrund eines unbedingten Sprungs generell ignoriert wird.

Ein kleines Beispiel soll den Sinn und Zweck dieses Pseudos verdeutlichen. Dazu wollen wir aus dem Quelltext heraus einen bestimmten Speicherbereich mit dem Maschinenbefehl »NOP« füllen.

```
10  —.BA $9000
20  —.EQ A = 0
30  —.IF A !<! 255
40  —                    NOP
50  —.EQ A = A + 1
60  —.GO 30
70  —.EI
80  —                    RTS
```

Beim Gebrauch von ».GO« und ».ON« ist anzumerken, daß im Falle eines RENUMBERs mit dem Editorbefehl »/N« die Sprungziele nicht angepaßt werden.

Was nun die bedingte Assemblierung leistet, soll an einem Makro, das einen »JSR«-Befehl und einen »JMP«-Befehl simuliert, der sämtliche Adressierungsarten beherrscht, gezeigt werden (Listing 1). Dabei wird ebenfalls ein Parameter übergeben, der die Adressierungsart bestimmt. Das Makro läßt sich vom Hauptprogramm mit dem Befehl »... JSR (adresse,adressierungsart)« aufrufen. Der »JMP«-Befehl wird genauso angewendet wie der »JSR«-Befehl. Adressierungsart bedeutet:

1 = X-indiziert; 2 = Y-indiziert; 3 = indiziert, indirekt; 4 = indirekt, indiziert

Was nun tatsächlich in das Maschinenprogramm assembliert wird, zeigt für den Fall, daß die Adressierungsart gleich 1 ist (also X-indiziert) Listing 2 und für den Fall, daß die Adressierungsart gleich 4 ist (also indirekt indiziert) Listing 3.

Sollten Sie noch irgendwelche Fragen zur Bedienung eines Assemblers haben, würden wir uns freuen, Ihnen bei der Lösung Ihrer Probleme zu helfen. Wir sind bestrebt, alle schriftlichen Anfragen zu beantworten. (ah)

```
30   -.MA JMP (ADRESSE,XY)
40   -.EQ HI = $FB
50   -.EQ LO = $FA
60   -.GO 131
70   -.MA JSR (ADRESSE,XY)
80   -.EQ HI = $FB
90   -.EQ LO = $FA
100  -            LDA #>(RUECKSP)
110  -            PHA
120  -            LDA #<(RUECKSP)
130  -            PHA
131  -.ON  (XY = 1)!O!(XY = 2) , 270
140  -.IF (XY = 1)
150  -            LDA ADRESSE+1,X
160  -            STA HI
170  -            LDA ADRESSE,X
180  -            STA LO
190  -.EI
200  -.IF (XY = 2)
210  -            LDA ADRESSE+1,Y
220  -            STA HI
230  -            LDA ADRESSE,Y
240  -            STA LO
250  -.EI
251  -.GO 530
270  -            INC ADRESSE
280  -            BNE LBL1
290  -            INC ADRESSE+1
310  -.IF (XY = 3)
320  -LBL1        LDA (ADRESSE,X)
330  -.EI
340  -.IF (XY = 4)
350  -LBL1        LDA (ADRESSE),Y
360  -.EI
380  -            STA HI
390  -            LDA ADRESSE
400  -            BNE LBL2
410  -            DEC ADRESSE+1
420  -LBL2        DEC ADRESSE
440  -.IF (XY = 3)
450  -            LDA (ADRESSE,X)
460  -.EI
470  -.IF (XY = 4)
480  -            LDA (ADRESSE),Y
490  -.EI
500  -.IF (XY=3)!O!(XY=4)
510  -            STA LO
520  -.EI
530  -            BNE LBL3
540  -            DEC HI
550  -LBL3        DEC LO
560  -            LDA HI
570  -            PHA
580  -            LDA LO
590  -            PHA
600  -RUECKSP     RTS
610  -.RT
```

Listing 1. Dieses Makro simuliert einen JSR-Befehl mit sämtlichen Adressierungsarten.

```
,9000  A9 90     LDA #90
,9002  48        PHA
,9003  A9 1C     LDA #1C
,9005  48        PHA
,9006  BD 01 C0  LDA C001,X
,9009  85 FB     STA   FB
,900B  BD 00 C0  LDA C000,X
,900E  85 FA     STA   FA
,9010  D0 02     BNE 9014
,9012  C6 FB     DEC   FB
,9014  C6 FA     DEC   FA
,9016  A5 FB     LDA   FB
,9018  48        PHA
,9019  A5 FA     LDA   FA
,901B  48        PHA
,901C  60        RTS
```

Listing 2. Eine »1« als Adressierungsart generiert dieses Maschinenprogramm. (Disassemblerausdruck SMON)

```
,9000  A9 90     LDA #90
,9002  48        PHA
,9003  A9 28     LDA #28
,9005  48        PHA
,9006  E6 02     INC   02
,9008  D0 02     BNE 900C
,900A  E6 03     INC   03
,900C  B1 02     LDA  (02),Y
,900E  85 FB     STA   FB
,9010  A5 02     LDA   02
,9012  D0 02     BNE 9016
,9014  C6 03     DEC   03
,9016  C6 02     DEC   02
,9018  B1 02     LDA  (02),Y
,901A  85 FA     STA   FA
,901C  D0 02     BNE 9020
,901E  C6 FB     DEC   FB
,9020  C6 FA     DEC   FA
,9022  A5 FB     LDA   FB
,9024  48        PHA
,9025  A5 FA     LDA   FA
,9027  48        PHA
,9028  60        RTS
```

Listing 3. Eine »4« als Adressierungsart generiert dieses Maschinenprogramm. (Disassemblerausdruck SMON)