Tips und Tricks zu Hypra-Ass

Hypra-Ass, ein Assembler der Spitzenklasse beherrscht alles, was zum Programmieren in Maschinensprache erforderlich ist. Hier wollen wir Ihnen anhand vieler Beispiele zeigen, was in ihm steckt und was er wirklich leistet.

Hypra-Ass ist einer der leistungsfähigsten Assembler die zur Zeit auf dem Markt sind. Eine seiner hervorstechendsten Eigenschaften ist der integrierte Editor mit einer formatierenden LIST-Routine. Aber gerade durch diese Eigenschaft weicht die Bedienung vom normalen Basic-Editor ab. Dadurch traten bei vielen Lesern Schwierigkeiten auf, die hier im einzelnen behandelt werden.

Im Gegensatz zum Basic-Editor kann unter Hypra-Ass eine Zeile nicht dadurch gelöscht werden, daß nur die Zeilennummer eingegeben und anschließend RETURN gedrückt wird. Bei Hypra-Ass ist unbedingt darauf zu achten, daß hinter der Zeilennummer ein Minuszeichen eingegeben wird. Drückt man nun die RETURN-Taste ist die Zeile auch verschwunden. Da aber dieses Minuszeichen hinter der Zeilennummer meistens vergessen wird, ist es empfehlenswert, nicht nur Zeilenbereiche, sondern auch einzelne Zeilen mit dem Editor-Befehl »/D zeilennummer« zu löschen, bis auf die Zeile »0«, die sich mit dem »/D«-Befehl nicht löschen läßt. In diesem Fall geben Sie bitte »0-« <RETURN> ein.

Ein kleiner Fehler tritt beim Sortieren der Symboltabelle auf. Hypra-Ass stürzt ab, wenn die Symboltabelle genau 36, 73, 109 (und so weiter) Variablen oder Label 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 unmittelbar mit einem Monitor in die entsprechenden Speicherzellen geschrieben und anschließend gespeichert werden. Sollten Sie keinen Monitor haben, dann geben Sie bitte den folgenden Quelltext ein:

```
 10 -.BA $C000
;STARTADRESSE = $C000
 20 -            LDY #0
 30 - LBL        LDA TAB,Y
;KORREKTUREN VORNEHMEN
 40 -            STA $1EB8,Y
 50 -            INY
 60 -            CPY #4
 70 -            BNE LBL
 80 -.EQ SOURCESTART = $1FD8 ;UND DIE KORRIGIERTE
 90 -.EQ NAMLEN = 12         ;VERSION SPEICHERN
100-            LDA #1
110-            LDX #8
120-            STA $FE
130-            STX $FF
140-            LDA #8
150-            JSR $FFBA
160-            LDA #NAMLEN
170-            LDX #<(NAME)
180-            LDY #>(NAME)
190-            JSR $FFBD
200-            LDA # $FE
210-            LDX #<(SOURCESTART)
220-            LDY #>(SOURCESTART)
230-            JMP $FFD8
240-;
250-NAME        .TX "HYPRA-ASS.V1"
260-TAB         .BY $90,$D0,$D0,$04
```

Nach dem Assemblieren wird mit SYS 49152 <RETURN> Hypra-Ass geändert und unter dem neuen Namen »Hypra-Ass.V1« auf Diskette gespeichert.

Der »/A«-Befehl zur automatischen Zeilennumerierung reagiert auch recht sensibel. Wird mit diesem Befehl gearbeitet, darf der Cursor mit den entsprechenden Steuertasten auf keinen Fall auf eine andere Zeile gesetzt und RETURN gedrückt werden. Sollte das versehentlich doch einmal passieren, läßt sich die so entstandene, ewas seltsam aussehende Zeile mit dem »/D«-Befehl problemlos löschen.

Diskettenbefehle können mit dem Editorbefehl »/@« zum Floppy-Laufwerk gesendet werden. Hinter den Editorbefehl werden dann die Diskettenbefehle unmittelbar angehängt. So formatiert der Befehl »/@N:NEWDISK,ND« eine neue Diskette.

Der Editor

Eine feine Sache ist auch das Arbeiten mit dem »/P«-Befehl, der dazu dient, Arbeitsseiten beziehungsweise Arbeitsbereiche anzulegen. Durch diesen Befehl, auf den sich die meisten Editor-Befehle beziehen, ist es möglich, jedem zusammenhängenden Quelltextteil (Unterprogramme oder Unterprogrammblöcke) einen Arbeitsbereich zuzuordnen. Möchte man dann in der Page 3 etwas ändern oder nachschauen, LISTet der Befehl »/3« nur diesen Bereich und nicht das komplette Listing wie bei dem »/E«-Befehl. Legt man nun die einzelnen Arbeitsbereiche gleich von vornherein so an, daß sie jeweils einen Zeilenbereich von zum Beispiel 5000 Zeilen überdecken, dürfte für die einzelnen Quelltextteile genügend Platz vorhanden sein, so daß beim Durchnumerieren der einzelnen Arbeitsbereiche keine Überlappungen auftreten können. Die Arbeitsbereiche selbst dürfen sich aber durchaus überlappen. So läßt sich zum Beispiel ein Arbeitsbereich von 0 bis 5000, ein zweiter von 10000 bis 15000 und ein dritter von 0 bis 15000 anlegen.

Bei dem Assembler selbst sind bisher keine Fehler bekannt. Deshalb möchte ich an dieser Stelle auf einige Dinge eingehen, mit denen viele Leser Schwierigkeiten hatten. Da wäre zum Beispiel das unmittelbare Erzeugen des Objektcodes auf Diskette mit dem ».OB«-Pseudo-Opcode.

Der Pseudo-Opcode ».OB "filename,P,W"« muß am Anfang des Quelltextes stehen und zwar in der ersten beziehungsweise zweiten Zeile (nach dem ».LI«-Pseudo zur Ausgabe des Assembler-Listings). In dem Zusammenhang sei auch erwähnt, daß es unmöglich ist, den Objektcode und gleichzeitig das Assembler-Listing mit dem Befehl ».LI 2,8,2,"filename,U,W"« auf Diskette zu erzeugen. Der Grund ist der, daß zwei Kanäle zum Schreiben geöffnet werden müßten und das ist nicht möglich. Zu dem ».OB«-Pseudo gehört unmittelbar ein zweiter Pseudo ».EN«, der das mit »filename« gekennzeichnete File schließt. Dazu muß dieser Pseudo am Ende des Quelltextes stehen. Sollten mit dem ».AP« mehrere Quelltexte verkettet werden, muß der ».EN«-Pseudo am Schluß des letzten Quelltextes auftauchen.

Bei der Anwendung von Makros gab es auch einige Schwierigkeiten. Wird zum Beispiel von einem Makro (Ordnung 1) zweimal ein weiteres Makro (Ordnung 2) aufgerufen, meldet Hypra-Ass einen »label twice error«, vorausgesetzt,

im Makro zweiter Ordnung befindet sich ein Label. Zum Beispiel würde folgendes Programm zu einer solchen Fehlermeldung führen:

```
10 -.BA $C000
20 -.MA MAK1
;MAKRODEFINITION 1. ORDNUNG
30 -   ... MAK2
;MAKROAUFRUF 2. ORDNUNG
40 -   ... MAK2
50 -.RT
60 -.MA   MAK2
70 -LBL   NOP
80 -.RT
90 -        ... MAK1
```

Dabei ist Mak1 das Makro 1. Ordnung und Mak2 das Makro 2. Ordnung. Alle Label in Makros zweiter oder dritter Ordnung sind untereinander global. Das heißt, daß in Makros zweiter Ordnung nur einmal das Label mit dem Namen »LBL« definiert werden dürfte.

Im Augenblick wird an einer Erweiterung gearbeitet, die diesen Mißstand beseitigt. Denn gerade beim intensiven Arbeiten mit Makros sind Makros zweiter und sogar dritter Ordnung unabdingbar. Ganz deutlich sieht man dies an dem Artikel »Wichtige Makros für Hypra-Ass« in dieser Ausgabe. Dort wurde ein Makro mit dem Namen »INCW (adresse)« definiert. Würde man die dort stehende 16-Bit-Addition ersetzen durch:

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

könnte dieses Makro von keinem anderen Makro aus zweimal aufgerufen werden, weil durch das Label »LBL« ein »label twice error« erscheinen würde.

Der gleiche Fehler erscheint natürlich auch dann, wenn ein anderes Makro aufgerufen wird, das »LBL« als Label oder Variable benutzt. Denn die Ordnungszahl, die den beiden Labeln »LBL« zugewiesen wird, ist identisch.

Bedingte Assemblierung

Auch mit der bedingten Assemblierung wissen nur die wenigsten etwas anzufangen, obwohl sie gerade im Zusammenhang mit Makros eine große Rolle spielt. Dies soll an einem kleinen Beispiel demonstriert werden:

```
10  -.MA ADW (ADR1,ADR2,SUMME,RETTEN)
20  -.IF RETTEN !=! 1              ;NUR WENN RETTEN
                                   = 1, WIRD
30  -               PHA            ;PHA IN DAS MASCHI-
                                   NENPROGRAMM
40  -.EI                           ;ASSEMBLIERT
50  -               CLC
60  -               LDA ADR1
70  -               ADC ADR2
80  -               STA SUMME
90  -               LDA ADR1+1
100 -               ADC ADR2+1
110 -               STA SUMME+1
120 -.IF RETTEN !=! 1              ;NUR WENN RETTEN
                                   = 1, WIRD
130 -               PLA            ;PLA IN DAS
                                   MASCHINENPROGRAMM
140 -.EI ;                         ASSEMBLIERT
150 -.RT
```

Dieses Makro addiert (ADR1,ADR1+1)+(ADR2,ADR2+1) und speichert das Ergebnis in den Speicherzellen (SUMME,SUMME+1). ADR1, ADR2 und SUMME können beliebige Speicherzellen oder Variablen sein. Soll der Inhalt des Akkumulators erhalten bleiben, wird für RETTEN eine 1, ansonsten eine beliebige andere Zahl eingegeben. Anhand des Übergabeparameters RETTEN erkennt der Assembler, ob das erzeugte Maschinenprogramm den Maschinenbefehl »PHA« beziehungsweise »PLA« enthalten soll oder nicht. Die Befehle zur bedingten Assemblierung zeigen also einzig und allein eine Wirkung beim Assemblieren. Durch sie wird bestimmt, welche Teile des Quelltextes im erzeugten Maschinenprogramm stehen und welche unter bestimmten Bedingungen übersprungen werden sollen. Schauen Sie sich nun noch einmal die Zeilen 20 und 120 an. Sie finden dort in der bedingten ».IF«-Abfrage den Operator »=«, der wie alle anderen Operatoren auch in Ausrufezeichen einzufassen ist. Durch dieses Ausrufezeichen erkennt Hypra-Ass, daß es sich bei dem Operator »=« um eine Rechenvorschrift aus dem Quelltext heraus handelt. Alle Operatoren können außer bei der bedingten Assemblierung zum Beispiel auch bei der unmittelbaren Adressierung angewendet werden. Ein kleines Beispiel soll die Wirkung dieser Operatoren bei der unmittelbaren Adressierung verdeutlichen:

```
20  -.EQ VARIABLE1 = 10
30  -.EQ VARIABLE2 = 20
40  -            LDA #(VARIABLE1 !0! VARIABLE2)
```

Der Akkumulator wird mit einer Zahl geladen, die mit der Variablen »VARIABLE1« und »VARIABLE2« wie im Basic geORt wird. Das Ergebnis ist folglich 30.

Viele Maschinensprache-Anfänger verwechseln die Befehle zur bedingten Assemblierung mit normalen Basic-Befehlen. Deshalb möchte ich an einem kleinen Beispiel zeigen, was nicht mit der bedingten Assemblierung funktioniert:

```
10  -.BA  $C000
20  -           INC $D020
30  -.GO  20
```

Was nicht funktioniert

Das Programm sollte die Bildschirmrahmenfarbe laufend um 1 incrementieren. Wird der Assembler jedoch gestartet, ersetzt er den Befehl ».GO 20« nicht durch den Befehl »JMP adresse«. Vielmehr versucht er den gesamten Speicher von $C000 bis unendlich mit dem Befehl »INC $D020« zu füllen, denn es fehlt jegliche Abbruchbedingung. Nur mit einer Abbruchbedingung ist der ».GO«-Befehl sinnvoll. Sollte das Maschinenprogramm zehnmal hintereinander den Befehl »INC $D020« enthalten, könnte das so aussehen:

```
10  -.LI 1,3
20  -.BA $C000
30  -.EQ A = 0
40  -.EQ A = A + 1
50  -           INC $D020
60  -.IF A !<! 11
70  -.GO 40
80  -.EI
90  -           JMP $C000
```

Der Assembler überprüft in Zeile 60, ob die Variable »A« kleiner 11 ist. Trifft das zu, wird in Zeile 70 durch den ».GO«-Befehl zur Zeile 40 verzweigt und der Befehl »INC $D020« ein weiteres Mal assembliert. Sobald »A« gleich 10 ist, verzweigt der Assembler in die Zeile 90, übersetzt den Befehl »JMP $C000« und beendet den Assembliervorgang.

Sollte Ihnen der Umgang mit Makros und der bedingten Assemblierung noch nicht klar sein, empfehle ich Ihnen den Artikel »Assemblerbedienung leicht gemacht (2)« in der 64'er Ausgabe 1/85. In diesem Artikel wird ausführlich auf den Gebrauch von Makros und eben der bedingten Assemblierung eingegangen.

(ah)