Assembler im Test

Teil 1

Mit dem AS-64 von Roßmöller erhält man nicht nur einen Assembler, sondern ein ganzes Assemblerpaket: Eine EPROM-Karte enthält in 16 KByte den Editor, den Assembler und einen einfachen Monitor. Dazu gibt es eine Diskette mit weiteren Hilfsprogrammen: Einen Re-Assembler, einen Monitor mit Disk-Monitor und einen Demo-Quelltext. Damit gehört AS-64 sowohl vom Umfang, allerdings auch vom Preis her zu den »besseren« Assemblersystemen für den C 64.

Fast ein Textsystem: Der Editor

Der Text wird so editiert wie man es von Textprogrammen her gewöhnt ist: Man schreibt einfach drauflos. Eine (abschaltbare) Tabulatorfunktion sorgt dafür, daß der Text schon beim Schreiben formatiert ausgegeben wird, das heißt Labels stehen am Anfang der Zeile, die Befehle beginnen in Spalte 12. Die zur Formatierung notwendigen Leerzeichen werden allerdings nicht mit abgespeichert, sondern nur bei der Ausgabe auf Bildschirm oder Drucker eingefügt. Dies spart natürlich enorm Speicherplatz. Änderungen erfolgen durch einfaches Überschreiben des alten Textes.

AS-64: auf Modul und Diskette

Ansonsten gibt es alles, was man von Textverarbeitungen her kennt: »Cursor up« und »Cursor down« ermöglichen zeilenweises, »Home« und F1 seitenweises »Blättern« vorwärts und rückwärts. Mit »SHIFT Home« gelangt man an den Anfang des Textes, mit F2 ans Ende. Mit F3 wird eine ganze Zeile eingefügt, mit F4 gelöscht. Die Tasten F7 und F8 erfüllen eine Spezialfunktion für Labels. Mit F7 werden sechs Leerzeichen eingefügt, mit F8 kann man von der Cursorposition bis zum linken Rand löschen. Damit lassen sich bequem zusätzliche Labels einbauen oder löschen. Wie der Assembler ist auch der Editor auf maximale Geschwindigkeit ausgelegt. Der Quelltext wird als Bildschirmcode im Speicher abgelegt, so entfällt eine dauernde Umrechnung in ASCII. Die damit erreichte Geschwindigkeit ist enorm. 30 KByte, das entspricht zirka 35 A4 Seiten Quelltext, kann man mit F1 in weniger als 10 (!) Sekunden durchblättern. Ich kenne kein Textsystem, mit dem das möglich wäre.

Natürlich hat dies seinen Preis: Schreibt man bei einer Änderung nämlich über ein Zeilenende hinaus, ohne vorher mit »Insert« die erforderliche Anzahl Leerzeichen einzufügen, springt der Editor in die nächste Zeile und überschreibt den dort stehenden Text. Eingewöhnung ist also nötig. Besser wäre hier eine Art »Insert-Modus«, mit dem man automatisch Text einfügen könnte.

Mit »CTRL-Z« werden Zeilen oder Textteile markiert, die anschließend verschoben, kopiert, gedruckt oder gelöscht werden können. Mit »CTRL-*« lassen sie sich auch auf Kassette oder Floppy abspeichern, nützlich, wenn man sich eine Bibliothek häufig benutzter Unterprogramme anlegen will.

Alle weiteren Editorkommandos entsprechen den Funktionen komfortabler Textsysteme. Neben einem »FIND«-Modus, der beliebige Zeichenketten sucht, gibt es eine »EDIT«-Funktion. Damit lassen sich ganze Worte durch andere ersetzen (zum Beispiel Label durch *Label, wenn man ein Label nachträglich in die Zeropage verlegen möchte.) Auch hier ist auf extreme Geschwindigkeit geachtet worden.

Besondere Erwähnung verdient auch der Umgang mit Peripherie-Geräten: Ein am User-Port angeschlossener Drucker wird automatisch erkannt und über eine eingebaute Centronics-Schnittstelle angesprochen. Commodore-Drucker werden wie üblich über die serielle Schnittstelle versorgt.

Als externe Speicher können Floppy oder Datasette angeschlossen werden. Beim Umgang mit der Floppy steht selbstverständlich eine Directory-Funktion zur Verfügung, ebenso können Kommandos an die Floppy ausgegeben werden. Eine Spezialität ist das »Backup«: Damit wird auf der Diskette eine Sicherheitskopie des letzten Textes erzeugt und gleichzeitig der neue gespeichert. So hat man immer die beiden (!) letzten Versionen zur Verfügung.

Der Assembler

Er kann erst einmal alles, was man von einem guten Assembler erwartet: Labels mit maximal elf Zeichen werden verarbeitet, Operand-Eingaben können dezimal, hexadezimal, binär und auch als ASCII-Zeichen erfolgen. Allerdings muß die Zeropage-Adressierung dem Assembler mit einem »*« vor der Adresse angezeigt werden, sonst wird sie als absolute Adressierung, also als 3-Byte-Befehl, assembliert.

Daneben gibt es eine Reihe von Pseudo-Opcodes, die Anweisungen an den Assembler enthalten. So kann man zum Beispiel »auf Probe« assemblieren, ohne den erzeugten Maschinencode (Objektcode) gleich in den Speicher zu schreiben. Man ist dadurch zumindest vor Syntax-Fehlern sicher und überschreibt nicht aus Versehen den Quelltext. Mit .LS und .LC können beliebige Teile des Programms während der Assemblierung auf Bildschirm oder Drucker aufgelistet werden. Ein besonders bequemes Kommando: Die .BY-Anweisung gestattet nicht nur die Eingabe von einzelnen Bytes — etwa einer Tabelle —, sondern vor allem auch die bequeme Eingabe von Texten. Diese werden einfach als Buchstabenfolgen abgelegt und beim Assemblieren in die zugehörigen ASCII-Werte umgerechnet. Damit entfällt das lästige Wälzen von ASCII-Tabellen.

Weitere Pseudo-Opcodes unterstützen dies: So ist es möglich, bei der Assemblierung bestimmte Teile des Quelltextes zu überspringen (bedingte Assemblierung). Man kann damit ein und dasselbe Quellprogramm für verschiedene Rechnertypen benutzen. Bei der Adreßeingabe sind nicht nur Labels, sondern auch noch umfangreiche Re-

Wer in Maschinensprache programmieren möchte, benötigt einen Assembler. Und da gerade aus dem C 64 mit Maschinensprache sehr viel herausgeholt werden kann, haben wir die interessantesten Assembler für Sie getestet. Sie lassen sich in zwei Preisgruppen einteilen, die unter und die ab 100 Mark. Im ersten Teil ist die obere Preisgruppe mit AS-64, MAE, T.EX.AS und ASSI/M an der Reihe.

chenoperationen gestattet. Es stehen die vier Grundrechenarten sowie logische Verknüpfungen (AND, OR, EOR) zur Verfügung. Damit läßt sich ein Quelltext sehr flexibel halten, nur wenige Adressen müssen fest vorgegeben werden, der Assembler selbst rechnet den Rest aus und paßt ihn automatisch an. Im Normalfall genügt dann die Änderung der Programmstartadresse (.BA) und das Programm läuft in einem anderen Speicherbereich. Alle bisher genannten Funktionen sind auch schon früher in guten Assemblersystemen vorhanden gewesen. Auch die Syntax der Befehle dürfte jedem, der mit einem MAE, ASTEX oder ähnlichen Assembler für den 6502 bereits gearbeitet hat, geläufig sein. Zwei Merkmale aber sind es, die AS-64 herausheben: Das eine und zweifellos wichtigste ist die fast unglaubliche Geschwindigkeit. Zwei Beispiele:

Einen 14 KByte langen Quelltext (Super Copy) assembliert AS-64 in fünf (!) Sekunden. Der bisher schnellste mir bekannte Assembler (MAE) braucht dafür immerhin 65 Sekunden. Aus einem 29 KByte langen Quelltext macht AS-64 in 11 Sekunden zirka 5 KByte Objektcode.

Erhöhter Zeitbedarf ergibt sich nur dann, wenn die Quelltexte länger als 30 KByte werden. Dann müssen nämlich Teile von Diskette nachgeladen werden. Auch diese Art der Assemblierung unterstützt AS-64: Wird ein Quelltext nicht mit ».EN«, sondern mit »CT "name"« abgeschlossen, wird automatisch das Quelltextteil »name« von Diskette geholt und assembliert. Damit ist die Verarbeitung von beliebig langen Maschinenprogrammen möglich. Allerdings dürfte das nur in den seltensten Fällen nötig sein. Durch geschickte Speicherverwaltung sind bei AS-64 nämlich Quelltexte bis zu 30 KByte Länge möglich. Beim Assemblieren wird der Editor abgeschaltet, so daß im darunterliegenden RAM die Labeltabelle angelegt werden kann. Der erzeugte Objectcode kann mit dem Pseudo-Opcode ».MC« zum Beispiel nach $E000 unter das Betriebssystem gelegt und von dort später mit einer Spezialfunktion des Monitors wieder an die richtige Adresse verschoben werden. Dies ist das zweite große Plus von AS-64.

Fehlermeldungen bei der Assemblierung werden übrigens im Klartext und auf Deutsch ausgegeben (Sprung zu weit, Label doppelt etc.). Der Assembler stoppt dann die weitere Assemblierung. Allerdings erhält man keinen Hinweis darauf, an welcher Stelle im Quelltext der Fehler aufgetreten ist, vielmehr ist man auf die »FIND«-Funktion des Editors angewiesen. Angenehmer wäre es, wenn bei einem Fehler automatisch der Editor angesprungen und die fehlerhafte Zeile markiert würde.

Apropos Programmabsturz: Mir ist es noch nie gelungen, auch nur das kleinste Maschinenprogramm auf Anhieb so zu schreiben, daß es sich nicht beim ersten Probelauf »verabschiedet«. Bisher zog das eine recht umständliche Prozedur nach sich. RESET drücken, Assembler neu laden, Quelltext neu laden, neu probieren. Mit AS-64 sieht das anders aus. Da sich das Programm im EPROM befindet, kann man es auch bei noch so »geschickter« Programmierung nicht zerlegen. Nach RESET befinde ich mich wieder im Anfangs-Menü und kann einen »Softstart« durchführen. In den meisten Fällen ist dann der Quelltext sogar noch vorhanden, und es kann sofort weitergehen.

Der Monitor

Ein kleiner Monitor ist direkt ins System integriert. Seine Funktionen erlauben Disassemblieren und Verschieben des Objektcodes sowie das Laden und Abspeichern fertiger Programme. Braucht man mehr, zum Beispiel Such-Funktionen oder einen Trace-Modus, muß man den »großen« Monitor von Diskette laden. Ebenfalls auf der Diskette befindet sich ein Re-Assembler, mit dem man fertige Maschinenprogramme wieder in editierfähigen Quelltext zurückverwandeln kann. Auch Labels werden wieder gesetzt, aber natürlich nicht mit sinnvollen Namen versehen. Mit dem Re-Assembler kann der Anwender nach Belieben Maschinenprogramme seinen eigenen Wünschen anpassen, was bisher nur bei Basic-Programmen möglich erschien.

Makros sind aus Speicherplatzgründen nicht im EPROM untergebracht, sondern müssen ebenfalls bei Bedarf von Diskette geladen werden. Mir lag dieser Programmteil leider noch nicht vor, er soll aber bei Erscheinen dieser Ausgabe vorhanden sein. Außerdem kündigt der Hersteller an, daß eine erweiterte Version (mit 32 KByte EPROM) geplant ist, in der dann auch Makros und ein leistungsfähiger Monitor integriert sein werden. Der Besitzer von AS-64 kann später seine Version aufrüsten lassen.

Fazit

AS-64 ist ein sehr guter Assembler. Zwar gibt es noch Schwachpunkte (Zeropage-Adressierung, Fehlerquelle schlecht zu finden, kein Insert-Modus), aber das sollte noch zu beheben sein. Auf die noch fehlenden Makros wurde bereits hingewiesen.

Auch die mitgelieferte »Dokumentation« (10 Seiten, DIN A5) ist ein Schwachpunkt. Es wird vorausgesetzt, daß der Benutzer schon Erfahrungen mit vergleichbaren Assemblern besitzt und sie bedienen kann. Ein Anfänger wird völlig alleine gelassen.

Noch ein Wort zum Service: Der Hersteller bietet an, daß man »gegen Einsendung des Moduls mit einem frankierten Rückumschlag kostenlos (!) die aktuellste Version von AS-64 abrufen« kann. Leider ist solcher Service in der Branche noch lange nicht selbstverständlich.

In puncto Geschwindigkeit und Bequemlichkeit beim Assemblieren bleiben kaum Wünsche offen. Setzt man die Kosten (295 Mark) zur Leistung ins Verhältnis, schneidet AS-64 immer noch sehr gut ab. Immerhin erhält man Hardware im Werte von zirka 100 Mark. Das eigentliche Programmpaket ist mit knapp 200 Mark seinen Preis wert. Natürlich gibt es billigere Systeme, kaum aber preiswertere.

(D. Weineck/gk)

Bezugsquelle:
Roßmöller GmbH, Finkenweg 1,
5309 Meckenheim

MAE — ein bewährter Oldtimer

Schon 1978 konnte man den MAE von SM-Software für die damaligen Commodore-Computer kaufen. Seit einiger Zeit ist auch eine C 64 Version auf Diskette erhältlich.

MAE ist eine Abkürzung für Makro-Assembler/Editor. Wie der Name schon sagt, wird ein spezielles Editor-Programm zum Erstellen des Quelltextes benötigt. Dieser Editor arbeitet ähnlich dem Basic-Editor. Es müssen einzelne Zeilen eingegeben werden, die jeweils eine vierstellige Zeilennummer tragen. Diese Zeilennummern bestimmen dann die Reihenfolge der einzelnen Anweisungen.

Natürlich gibt es hier gegenüber dem normalen Basic-Editor erweiterte Möglichkeiten. So ist beispielsweise der Speicherbereich, in dem sich der Quelltext befindet, beliebig vom Benutzer festlegbar. Dies gilt ebenso für den Speicherbereich der Symboltabelle.

Ein eingegebener Quelltext läßt sich beliebig im Quelltextspeicher verschieben, um nachträglich die Reihenfolge von Zeilen zu ändern. Die Zeilennummern können auch, ähnlich einem Renumber, verändert, oder bei der Eingabe automatisch erzeugt werden.

Sehr komfortabel ist hier der Such- und Ersetzbefehl gehalten. Er ermöglichst nicht nur die Angabe von Zeilenbereichen, in denen gesucht werden soll, sondern es kann auch mit Jokerzeichen gesucht werden. Jede gefundene Zeile kann gelistet und dann auf Wunsch verändert oder gelöscht werden.

Die Symboltabelle kann jederzeit aufgelistet, leider aber nicht gespeichert oder geladen werden. Das Floppy-Laufwerk wird mit einigen Befehlen unterstützt, Kassettenbetrieb ist nicht vorgesehen.

Vom Editor kann auch ein Listing des gesamten Quelltextes oder einzelner Zeilennummernbereiche zu einem Drucker gesandt werden.

Der Quelltext kann jederzeit über einen weiteren Befehl formatiert werden. Dabei werden Labels, Opcodes und Kommentare in schön lesbarer Form ausgegeben. Gleichzeitig ist die maximale Länge eines Labels einstellbar.

Assembler-Grundfunktionen

Der eigentliche Assembler arbeitet wahlweise in zwei oder drei Durchgängen (Passes). Im dritten Pass wird ein relatives Lademodul erzeugt, auf das ich noch zu sprechen kommen werde.

Labels können beim MAE maximal 31 Zeichen lang sein. Dies dürfte wohl eindeutige Labelbezeichnungen auch bei sehr langen Programmen ermöglichen. Leider sind die Rechenmöglichkeiten im Quelltext stark eingeschränkt: Es stehen nur Addition und Subtraktion zur Verfügung.

Dem Programmierer steht eine große Zahl von Pseudo-Opcodes zur Verfügung, darunter auch einige, die den Ausdruck eines formatierten Assemblerlistings in Pass 2 steuern. Labels können entweder im Quelltext selbst vor der ihr zugeteilten Anweisung stehen, es ist allerdings auch möglich, Label als intern und extern durch einfache Ausdrücke zu definieren. Externe Label bezeichnen normalerweise Adressen außerhalb, interne Adressen innerhalb des Objektcodes.

Besonders komfortabel ist, daß Labelwerte auch während des ersten Passes eingegeben werden können, so daß man einen Quelltext durch solche Eingaben für verschiedene Speicherbereiche oder Computer nutzen kann.

Auch eine bedingte Assemblierung ist implementiert. Bedingte Assemblierung bedeutet nichts anderes, als daß ein Quelltextblock nur unter bestimmten Bedingungen assembliert wird. Somit ist es leicht, den Objektcode während des Assembliervorganges an bestimmte Spezifikationen anzupassen.

Ein Quelltext kann auch auf seine Syntax getestet werden, während des Assembliervorgangs wird der Objektcode dann nicht gespeichert. Genauso gut ist es möglich, den Objektcode direkt auf Diskette zu leiten, so daß er keinerlei Speicherplatz im Computer benötigt.

Das Modulkonzept

Der MAE geht beim Assemblieren längerer Quelltexte anders vor, als andere Assembler. Wollen Sie einen Quelltext, der länger als der Arbeitsspeicher ist, assemblieren, so können Sie ihn nicht, wie üblich, in mehrere Teile teilen, die sich nacheinander aufrufen. Beim MAE müssen Sie ein Kontroll-Modul definieren, das stets im Speicher vorhanden ist. Dieses lädt dann nacheinander die Quelltextteile in den Speicher und assembliert diese. Das Kontrollmodul besteht ebenfalls aus ganz normalem Quelltext, sollte jedoch zweckmäßigerweise nur die Aufrufe der einzelnen Quelltextteile beinhalten. Dieses Modulkonzept hat den Vorteil, daß Sie einen in mehreren Programmen benötigten Quelltext nur einmal schreiben müssen und dann jedesmal über das Kontrollmodul abrufen können.

Makros

Wichtig bei einem Makroassembler ist natürlich die Makro-Behandlung. Makros dürfen beim MAE beliebig viele Übergabeparameter haben, die dann in die Makrosequenz eingesetzt werden. Auch makrointerne Label sind möglich. Diese müssen mit drei Punkten gekennzeichnet sein. An deren Stelle setzt der MAE beim Assemblieren dann eine Nummer, die von Aufruf zu Aufruf wechselt. So kann es nicht zu einem Fehler aufgrund doppelter Makros kommen. Makros dürfen hier bis zu 32mal ineinander verschachtelt werden.

Sollten Sie einige Makros in großen Programmen ständig benötigen, so müssen Sie ein globales Makro-Modul aufbauen. Dieses steht dann, ähnlich dem Kontrollmodul, ständig im Speicher, so daß jeder Teil des Quelltextes auf das Makro zugreifen kann.

Relative Lademodule

Eine sehr nützliche Einrichtung ist der mitgelieferte Relativlader. Möchte ich einen Quelltext in einem Bereich erzeugen, der schon anderweitig belegt ist, kann ich ihn in einen anderen Speicherbereich assemblieren und dann den dritten Durchlauf starten, der ein relatives Lademodul auf Diskette erzeugt. Dieses Modul kann dann mit einem zusätzlichen Programm, dem Relativlader, an jede beliebige Stelle im Speicher geladen werden. Dabei sind die Speicherbereiche für Programm und Daten frei wählbar. Danach steht das Programm ablauffähig im Speicher und kann absolut gespeichert werden. Der Speichervorgang wird aber vom MAE nicht unterstützt. Dieser Relativlader wird sowohl ablauffähig, als auch selber als relatives Lademodul mitgeliefert. Damit kann der Relativlader an jeder beliebigen Stelle im Speicher stehen. Mit dem MAE ist es nicht möglich, Objektcode direkt zur Diskette zu leiten. Will man längere Objektcodes, die man im Speicher nicht mehr unterbringen kann, erzeugen, muß man einen sehr unkonventionellen Weg gehen. Sie müssen sich Schnittstellen zwischen den einzelnen Programmen definieren, das sind Adressen, über die Daten ausgetauscht werden. Zwei zu verbindende Teilprogramme werden dann durch zweimaligen Aufruf des Relativladers zusammengebunden.

Dokumentation und Sonstiges

Das mitgelieferte Handbuch ist sehr unübersichtlich und unklar. Möchte man zum Beispiel Informationen über den Relativlader haben, stellt man fest, daß dieser an drei verschiedenen Stellen im Handbuch erklärt wird, so daß man ständig vor- und zurückblättern muß. Ebenso verhält es sich mit fast allen Themenbereichen. Das Handbuch einmal in Ruhe durchzulesen ist deswegen kaum möglich.

Schade finde ich es auch, daß bei Fehlermeldungen immer nur Codezahlen und keine Klartexte erscheinen.

Teilweise ist die Bedienung etwas umständlich geraten. Anfängern wird es wohl schwerfallen, sich mit MAE zurechtzufinden, insbesondere mit Blick auf das Handbuch. Aber MAE soll ja schließlich ein Handwerkszeug für professionelle Programmierer sein. Die werden allerdings einige Möglichkeiten anderer Assembler vermissen. Insbesondere stört da die Tatsache, daß man sich zusätzlich zum MAE (Kostenpunkt zirka 140 Mark) noch einen guten Monitor zulegen muß, da man in der Testphase vom MAE allein gelassen wird.

(Boris Schneider/gk)

Bezugsquelle:
SM Software AG, Scherbaumstr. 33, 8000 München 83

T.EX.AS. — mit guter Dokumentation

Ein weiteres Produkt der »Extended« Serie von Interface Age ist der Terminal Extended Assembler, kurz T.EX.AS.

Der T.EX.AS. erfüllt mit einem Programm gleich mehrere Funktionen. Es ist sowohl ein Monitor, ein Direktassembler, ein Reassembler wie natürlich auch ein Makroassembler.

Ein Editorprogramm ist nicht enthalten. Bei T.EX.AS. müssen Quelltexte über den Basic-Editor eingegeben werden. Interface Age schlägt zwar vor, Exbasic Level II zur komfortablen Quelltexteingabe zu verwenden, dies würde aber den Kaufpreis (298 Mark für T.EX.AS.) mehr als verdoppeln und diesen damit wohl über die Kaufkraft vieler Privatanwender stellen.

Der Assembler

In diesem Assembler dürfen Labels bis zu 16 Zeichen haben. Berechnungen sind leider nur auf Addition und Subtraktion beschränkt, was ein nicht unwesentliches Manko dieses Assemblers ist.

Auch ansonsten ist T.EX.AS. nicht gerade mit Pseudo-Opcodes gesegnet. Weder ist eine bedingte Assemblierung vorhanden, noch können Labels umdefiniert werden, was bei Assemblern dieser Preisklasse Standard sein sollte.

Symboltabelle und Quelltext können im zweiten Pass ausgedruckt werden. Der Quelltext wird dabei aber weder formatiert, noch sonst irgendwie verändert, so daß Sie ihn einfacher von Basic aus listen können. Es ist allerdings möglich, den erzeugten Objektcode direkt auf Diskette abzulegen, oder ihn gar nicht auszugeben; dann wird nur die Syntax des Quelltextes überprüft.

Das Einbinden schon geschriebener Quelltextteile ist sehr einfach gelöst. Wird ein solcher Teil benötigt, kann er während des Assembliervorganges nachgeladen und eingefügt werden. Auch hier ist eine modulartige Programmierung wie beim MAE denkbar.

Makros

Auch bei den Makros hapert es ein wenig. Es ist zwar eine beliebige Verschachtelung von Makros erlaubt, (bis auf Selbstaufrufe), aber es gibt keine Möglichkeit, in einem Makro lokale Label zu definieren, ohne sie nicht als Parameter zu übergeben. Also schon eine einfache Schleife in einem Makro, die über einen Labelsprung erfolgen soll, ist äußerst kompliziert und unübersichtlich zu programmieren. Jedes Makro mit internen Labeln dürfte nur einmal aufgerufen werden, soll kein LABEL DECLARED TWICE-Fehler auftreten. In diesen Fällen sollte man bei T.EX.AS. lieber auf Makros verzichten. Dafür ist der Aufbau einer Makrobibliothek sehr einfach, weil jedes Makro als Einzelfile auf der Diskette stehen und bei Bedarf aufgerufen werden kann.

Monitorähnliche Funktionen

In diesem Bereich zeigt T.EX.AS. seine Stärken, da es sich nicht um einen starren Monitor, sondern um eine Art Editiersystem für Objektcode (nicht Quelltext) handelt. Insbesondere die Analyse fremder Programme wird extrem erleichtert.

Als allererstes fällt auf, daß der Bildschirm in zwei Fenster, links und rechts, aufgeteilt ist. Den Cursor kann man mit der Control-Taste vom einen ins andere Fenster springen lassen. Dieses Fensterkonzept ist eine nicht zu unterschätzende Hilfe bei der Programmanalyse. Was machen Sie normalerweise, wenn Sie beim Disassemblieren eines Programmes den Sprung JSR$3000 entdecken? Sie müßten nachschauen, was dort abläuft, um dann später die Stelle mit dem Sprungbefehl wiederzufinden. Beim Disassemblieren mit T.EX.AS. genügt ein Druck auf F6, und schon wird im anderen Bildschirmfenster ab $3000 disassembliert, während im ersten der Programmteil, den Sie gerade untersuchen, stehenbleibt.

Natürlich lassen sich die zwei Fenster auch anders nutzen, zum Beispiel Hexdump im einen und ASCII-Dump im anderen Fenster oder ähnliches.

Selbstverständlich kann in beiden Fenstern unabhängig nach oben und unten gescrollt werden. Aber was bietet T.EX.AS. denn außer diesen Fenstern? Den schon angeführten Dump-Möglichkeiten muß noch die des Adressen-Dumps angefügt werden. Alle Dumps können auch zum Drucker hin erfolgen. Hier wird die Workspace-Konzeption notwendig. Für die meisten Befehle lassen sich Arbeitsbereiche angeben, die in einer Tabelle zwischengespeichert werden. Soll zum Beispiel mehrmals ein bestimmter Bereich disassembliert werden, reicht eine einmalige Definition. Bei anderen Befehlen ist diese Bereichsfestlegung allerdings sinnvoller, zum Beispiel beim Speichern von Bereichen auf Disk. Wenn Sie kurz hintereinander öfters denselben Bereich speichern wollen, weil Sie ihn gerade bearbeiten, müssen die entsprechenden Adressen nicht jedesmal angegeben werden.

Zusätzlich zum Makroassembler ist auch ein Direktassembler vorhanden. Dieser ist insbesondere in der Testphase nützlich. Er kann auch in beschränktem Maße mit Labels arbeiten.

T.EX.AS arbeitet üblicherweise im Dezimalsystem, da man der Meinung war, daß dies einfacher für den Benutzer sei. Unverbesserliche können natürlich auch alle Ein- und Ausgaben ins Hexadezimale umschalten.

Zwei noch erwähnenswerte Kommandos sind der Verschiebe- und der Suchbefehl. Zusätzlich zum normalen Verschieben von Speicherblöcken können Programme einfach an neue Adressen angepaßt werden.

Der Suchbefehl läßt kaum Wünsche offen. Es ist zum Beispiel möglich, im Objektcode alle Befehle zu suchen, die etwas in den Bereich von $3459 bis $7777 schreiben, sich alle indirekten Sprünge heraussuchen lassen, und so weiter. Beim Su-

chen von Kommandos kann der Suchbereich und der Bereich, auf den sich die gesuchten Befehle beziehen, angegeben werden. Außerdem sind Joker für den Befehlscode, das Argument und die Adressierungsart möglich. Leider ist diese Suchroutine nicht auch für den Quelltext verfügbar.

Dem Benutzer stehen zehn adreßdefinierbare Breakpoints zur Verfügung, bei deren Erreichen ein T.EX.AS. Warmstart durchgeführt wird. Dadurch können einzelne Informationen (Akkuinhalt und ähnliches) verlorengehen.

Nur kurz angesprochen werden soll hier der Reassembler, der aus Objektcode wieder Quellcode machen soll. Dieser Programmteil ist mit absoluter Vorsicht zu genießen. Mir ist T.EX.AS. zweimal abgestürzt, als er einen OUT OF MEMORY ERROR gab. Außerdem ist dieser Reassembler außerst unpraktisch. Eine Redefinition von Labels ist nicht vorgesehen.

Trifft der Reassembler auf einen nicht als Opcode definierten Wert, so wird im Quelltext nicht etwa das entsprechende Byte, sondern einfach zwei Fragezeichen ausgegeben. All diese kleinen Macken machen den Reassembler praktisch nutzlos.

Dokumentation

T.EX.AS. ist der wohl einzige Assembler, der auch mit einem, auf ihn abgestimmten, Assemblerlehrbuch ausgeliefert wird. Es ist ziemlich ausführlich und fängt wirklich bei Null an. Auch die Anleitung selbst ist nicht schlecht. An manchen Stellen, die Anfängern Schwierigkeiten bereiten könnten, wird auf die entsprechenden Kapitel im anderen Buch verwiesen. Wenn ich ein ehrliches Fazit ziehen soll, muß ich sagen, daß T.EX.AS zu teuer ist. Mit 298 Mark ist T.EX.AS. zwar ein gutes Lehrsystem, aber professionelle Programmierer, und solche die es werden wollen, werden bei T.EX.AS. einiges vermissen. (Boris Schneider/gk)

Bezugsquelle:
Interface Age Verlag GmbH, Vohburgerstr. 1, 8000 München 21, Preis: 298 Mark

ASSI/M

Mit allen Wassern gewaschen ist der ASSI von D. Zabel.

ASSI/M ist ein ganzes Programmpaket, bestehend aus einem Editor (FSE), dem Assembler (ASM) und einem Monitor (DEMON). Alle drei Programme werden gemeinsam auf einer Diskette geliefert. Es gibt auch eine Version die alle drei Programme vereinigt, dafür aber enorm viel Speicherbedarf hat. Durch die konsequente Aufteilung in Editor, Assembler und Monitor konnte jedes einzelne Programm sehr umfangreich geschrieben werden, ohne daß bei ASSI weniger freier Speicherplatz als bei anderen Assemblern zur Verfügung steht.

Der Editor (FSE)

FSE steht für Full-Screen-Editor, er arbeitet also bildschirmorientiert und ohne Zeilennummern. Aufgrund seines umfangreichen Befehlssatzes gleicht er schon fast einem Textverarbeitungsprogramm. Die vorhandenen Befehle sind jedoch immer am Ziel orientiert, Programme perfekt schreiben und editieren zu können.

Fast der ganze Bildschirm steht zur Eingabe des Quelltextes zur Verfügung. Unten wird er durch eine Statuszeile begrenzt, die stets die Position des Cursors, den Arbeitsmodus und den noch freien Speicherplatz angibt. Nach oben und unten kann mit sehr hoher Geschwindigkeit gescrollt werden. Angefangen beim einfachen Löschen und Einfügen von Buchstaben, Wörtern, Zeilen und ganzen Textteilen, reicht der Befehlssatz über freisetzbare Tabulatoren bis hin zum komfortablen Such- und Ersetzbefehl. Man kann die Suche auf bestimmte Zeilen oder Spalten beschränken, Groß- und Kleinschreibung ignorieren, und Joker beim Suchen und beim Ersetzen (!) angeben.

Der FSE merkt sich automatisch die Stelle, an der die letzte Veränderung im Text vorgenommen wurde. So können Sie schnell mal etwas nachschauen und später mit einem Tastendruck zum Ausgangspunkt zurückkehren. Daß Diskette, Kassette und Drucker voll unterstützt werden, ist da schon fast selbstverständlich. Die Funktionstasten sind belegt, es gibt vom Benutzer frei definierbare Tasten und, und, und...

Es würde den Rahmen dieses Tests sprengen, alle Möglichkeiten, die der FSE bietet, aufzuzählen.

Der Assembler (ASM)

Der leistungsfähige Editor läßt natürlich auf einen ebenso leistungsfähigen Assembler hoffen. Und diese Hoffnungen werden vom ASM voll erfüllt.

Der Quelltext wird entweder von Kassette oder Diskette während des Assemblierens gelesen. Bei Kassettenbetrieb muß deswegen mindestens einmal zurückgespult werden. Dadurch ist allerdings auch eine ziemlich lange Assemblierzeit bedingt. Dafür ist wiederum sehr viel Speicher für den Objektcode und die Symboltabelle verfügbar.

Labels dürfen bis zu 250 Zeichen lang sein. Außerdem gibt es eine Vielzahl von Rechenfunktionen: Alle vier Grundrechenarten, Bitschiebeoperationen, logische Funktionen (AND, OR,XOR,NOT) und logische Vergleiche dürfen beliebig miteinander kombiniert werden. Die Hierarchie wird durch Klammern festgelegt. Angenommen werden Zahlen im Hexadezimal-, Dezimal, Oktal-, und Binärformat sowie ASCII-Zeichen.

Als besonders großer Vorzug muß hier das Block-Konzept erwähnt werden. Teile des Quelltextes können jederzeit als Block definiert werden, und alle in ihnen definierten Labels sind dann lokal. Zum Beispiel können zwei verschiedene Programmteile das Label LOOP verwenden, wenn die eine Sequenz als Block definiert wurde und dort die eine Definition von LOOP erfolgte. Somit kann man jederzeit Quelltext aus anderen Programmen übernehmen und braucht sich keine Sorgen um etwaige doppelt definierte Label zu machen. Auch der Aufbau einer Unterprogrammbibliothek ist kein Problem. Denn neben dem einfachen Verketten von Quelltextfiles, kann man Unterprogramme, die auf Diskette vorhanden sind, mit einem Pseudo-Opcode einbinden. Sie werden dann zur Assemblierung nachgeladen. Die Ausgabe von Listings während des zweiten Pass verläuft hier ähnlich wie bei anderen Assemblern, wahlweise auf Drucker, Bildschirm, Floppy oder mehreren Ausgabegeräten gleichzeitig. Der Objektcode kann entweder beliebig im Speicher plaziert, oder aber auch direkt auf Diskette geschrieben werden.

Natürlich ist auch ein bedingter Assembliermodus vorhanden. Ebenso können beim ASSI Eingaben von Labels während des ersten Pass gemacht werden.

Makros

Der ASM beherrscht natürlich auch Makros. Bei allen Makroaufrufen wird automatisch ein Block um das Makro herum gelegt. So sind die in einem Makro definierten Parameter wieder lokal, das heißt unabhängig von Parametern anderer Programmteile universell verwendbar. Makros dürfen andere Makros oder auch sich selbst aufrufen; letzteres ist jedoch nur bei bedingter Assemblierung sinnvoll, da sonst der Speicher überläuft. Eine Beson-

derheit ergibt sich noch bei den Übergabeparametern. Neben normalen Ausdrücken können auch Strings oder Befehle übergeben werden, die dann in das Makro eingebaut werden.

Und weil die Definition von Makro-Bibliotheken genauso einfach geht, wie die schon oben erwähnte von Unterprogrammen, werden zwei Makrobibliotheken mitgeliefert. Die eine enthält 17 Makros, die hauptsächlich der 16-Bit-Arithmetik dienen, sowie einen User-Stack definieren, auf dem, ähnlich dem normalen Stack, Daten nach dem First in/Last out Prinzip gespeichert werden können.

Die andere ist fast schon sensationell zu nennen, denn sie ermöglicht strukturierte Assemblerprogrammierung. Es stehen dann Befehle wie IF..THEN..ELSE..,REPEAT..EXIT, REPEAT..UNTIL und ähnliches in Assembler zur Verfügung. Da diese Bibliotheken jederzeit erweitert werden können, bilden sie einen idealen Grundstock für jeden Assemblerprogrammierer.

Der Monitor (DEMON)

Und auch das dritte Programm fügt sich nahtlos in das System ein. DEMON ist ein sehr komfortabler Maschinensprachemonitor. Er ist ideal zum Austesten von Programmen geeignet. Schwerpunkte sind die hervorragende Breakpointbehandlung und die Trace-Modi.

Beim Austesten eines Programmes können bis zu fünf normale Breakpoints gesetzt werden. Gelangt der Programmcounter an einen Breakpoint, so wird automatisch in den Einzelschrittmodus des DEMON gesprungen, wo Sie sich den weiteren Verlauf des Programms Schritt für Schritt anschauen können. Zusätzlich ist ein User-Breakpoint definierbar. Sie stellen zum Beispiel den Fehler fest, daß Ihr Programm bestimmte Speicherstellen überschreibt, die wichtig sind. Nun schreiben Sie ein kleines User-Breakpointprogramm, das nach Abarbeitung jedes Befehls angesprungen wird und diese Speicherstellen überwacht. So können Sie in wenigen Minuten Fehler ausfindig machen, an denen Sie sonst Stunden herumknobeln würden. Auch ganz verrückte Abfragen wie: »Wann ist die Summe von X und Y Register kleiner als $93 ?« sind als Userbreakpoint realisierbar. Schon kurz angesprochen wurde der Einzelschrittmodus. Außer dem normalen Abarbeiten und Anzeigen jedes Befehls ist es möglich, Sprünge, die über eine bestimmte Adresse hinaus gehen, direkt ausführen zu lassen. Das spart ernorm Zeit, wenn viele ROM-Routinen verwendet werden.

Natürlich sind sämtliche Monitor-Standardfunktionen wie Hex-Dumps, Disassemblieren, Suchen von Bytes und Zeichenketten, Laden und Speichern, Vergleichen und Verschieben von Speicherbereichen (auch mit Adreßangleichung bei Programmen), ein Mini-Assembler, Registeranzeige etc. integriert.

Außerdem können Rechnungen im hexadezimalen, dezimalen und binären Zahlensystem vorgenommen werden, die fast so komfortabel wie im ASM sind. Ein spezieller Suchbefehl findet alle die Zeropage betreffenden Befehle. Damit können auch die vier von DEMON benötigten Zeropageadressen beliebig verändert werden. Überhaupt ist DEMON im Speicher frei verschieblich, so daß Sie niemals mit einem auszutestenden Programm in Konflikt kommen können.

Alles in allem läßt auch DEMON keine Wünsche offen.

Dokumentation

Das Handbuch macht auf mich einen sehr guten Eindruck, da alle Befehle in einer vernünftigen Reihenfolge leicht verständlich und mit einigen Beispielen erklärt werden. Einzig beim FSE wird die logische Reihenfolge durch einen Anhang, der die Erweiterungen mancher Funktionen nachträglich beschreibt, durchbrochen. Allerdings werden auch hier Assembler-Kenntnisse vorausgesetzt. Zusätzlich werden zu allen drei Programmen alle wichtigen Einsprungpunkte und Speicher beschreiben, so daß eine Modifikation durch den Benutzer sehr leicht möglich ist. Die Transparenz der drei Programme, gekoppelt mit ihrer Befehlsvielfalt und Nützlichkeit, machen das Programmpaket ASSI (Preis: 220 Mark) zum fast idealen Assembler. Einzig und allein die geringe Arbeitsgeschwindigkeit kann ein Kritikpunkt sein.

(Boris Schneider/gk)

Bezugsquelle:
Dirk Zabel, Stresemannstr. 50, 1000 Berlin 61

	AS-64	ASSI	MAE	T.EX.AS.
Editor	eigener Full-Screen-Editor	eigener Full-Screen-Editor	eigener Zeileneditor	Basic-Editor
Labels	max. 11 Zeichen	max. 250 Zeichen auch redefinierbare Labels (Variablen)	max. 79 Zeichen	max. 16 Zeichen
Monitor	je nach Version nachladbar oder resident	DEMON wird mitgeliefert	—	viele monitorähnliche Funktionen
Re-Assembler	wird mitgeliefert	—	—	eingebaut
Quelltext Verkettung/Einbindung	Verkettung	Verkettung oder Einbindung	Einbindung über Steuermodul	Einbindung
Objektcode direkt auf Diskette	nein	ja	nein, aber relative Lademodule	ja
Bedingte Assemblierung möglich?	ja	ja	ja	nein
Formatierung des Quelltextes	automatisch bei der Eingabe	bei Ausgabe in Pass 2 auch form. Eingabe möglich	bei Ausgabe in Pass 2	nur durch formatierte Eingabe
Makrobibliothek möglich?	noch nicht vorhanden	ja, zwei werden mitgeliefert	ja, mit globalem Makromodul	Makros direkt von Disk abrufbar
Besonderheiten	EPROM-Modul mit Diskette	Möglichkeit der Blockstrukturierung	Relativ-Lader wird mitgeliefert	zwei unabhängige Bildschirmfenster
Preis	298 Mark	220 Mark	140 Mark	298 Mark