2 Der kombinierte Editor/Makro-Assembler

- Änderungen -

Der Editor von `GENST.PRG` bildet das Kernstück der völlig integrierten Umgebung von Devpac ST Version 2. Sie können assemblieren, debuggen und auch andere Programme, wie z.B. LinkST, aufrufen, ohne jemals den Editor verlassen zu müssen. Wenn Sie zusätzlich noch unser Saved!- Desk-Accessory besitzen, gibt es kaum einen Grund, während der Programmentwicklung zurück ins Desktop zu gehen.

2.1 Generelle Änderungen und Ergänzungen

Zuerst wird auf die einfacheren Veränderungen des Editors eingegangen. Es handelt sich hiermit hauptsächlich um neue Tastaturkürzel und -Umbelegungen.

Die Geschwindigkeit des Editors hat sich stark erhöht. Es gibt so gut wie keinen Tastaturnachlauf mehr, und das »Blättern« geht um ein Mehrfaches schneller als beim Editor der Version 1.

Zeilen können jetzt bis zu 240 Zeichen lang sein. Das Seitwärts- Scrollen ist durch Benutzung des horizontalen Scrolling-Balkens des Editorfensters möglich. Der Editor scrollt auch automatisch zur Seite, wenn man sich mit dem Cursor nahe genug am Rand befindet. Dies gilt auch für Ctrl-C.

Shift-Alt-S ist das Tastaturkürzel für »Datei speichern«. Der Editor funktioniert jetzt auch in niedriger Auflösung.

Tipp: Wenn Sie ein Tab-Zeichen suchen bzw. ersetzen wollen, geben Sie Ctrl-I in der Dialogbox ein; als Tab wird dann eine kleine Uhr erscheinen.

2.2 Blockverwaltung

Die Verwaltung von Blöcken ist um einiges erweitert worden. Markierte Blöcke werden jetzt invers dargestellt.

2.2.1 Block löschen

Ein Block wird nicht mehr mit Shift-F3, sondern mit Shift-F5 gelöscht. Der gelöschte Block ist auch nicht unwiderruflich verloren, sondern wird im Blockspeicher abgelegt. Voraussetzung ist, daß genug freier Speicher vorhanden ist.

2.2.2 Block in den Blockspeicher kopieren

Mit Shift-F4 kopieren Sie den markierten Block in den Blockspeicher. Der markierte Block bleibt jedoch an seiner ursprünglichen Stelle und wird nicht - wie mit Shift-F5 - gelöscht.

2.2.3 Block aus dem Blockspeicher kopieren

Mit F5 wird der Inhalt des Blockspeichers an die Cursorposition kopiert.

Übersicht:

F1	Block Start
F2	Block End
F4	Copy Block
F5	Paste Clipboard
Shift-F4	Copy Block to Clipboard
Shift-F5	Cut Block to Clipboard

2.2.4 Undo und Ctrl-U

Eine ähnliche Funktion haben die Tastenkombinationen Ctrl-U und Undo. Eine Zeile, die mit Ctrl-Y gelöscht wurde, kann so oft wie nötig mit Undo bzw. Ctrl-U wieder eingefügt werden. Das gleiche gilt für Text, der mit Ctrl-Q gelöscht wurde.

2.3 Voreinstellungen

Es ist jetzt möglich, zusätzlich zum Tab-Abstand auch all die Einstellungen, die mit dem Installationsprogramm gemacht wurden, über »Voreinstellungen« festzulegen:

2.3.1 Tab-Abstand

Wie bei `GENST.PRG` Version 1 ist der Standardwert 8, kann aber auf jeden Wert zwischen 2 und 16 eingestellt werden.

2.3.2 Textspeichergröße

Der Standardwert ist 60.000 Byte, kann aber auf einen Wert von 4.000 bis 990.000 Byte eingestellt werden. Dadurch wird die maximale Dateigröße bestimmt, die geladen und editiert werden kann. Man sollte darauf achten, genug Speicher für den Assembler und ggf. MonST übrig zu lassen, normalerweise werden etwa 100 KByte benötigt. Verändern der Speichergröße hat zur Folge, daß der Text, der gerade bearbeitet wird, gelöscht wird; eine Alarmbox erscheint, wenn der Text nicht abgespeichert wurde.

2.3.3 Numerischer Tastenblock

Diese Option erlaubt es, den numerischen Tastenblock der ST-Tastatur als Cursorblock zu benutzen, ähnlich wie bei PC-Kompatiblen. Standardmäßig ist er als Cursorblock konfiguriert.

2.3.4 Backups

Ohne Voreinstellung macht der Editor keine Backups. Wenn Sie in der Abfragebox »Ja« wählen, gibt der Editor der schon existierenden Version Ihrer Datei die Extension `.BAK`. Die aktuelle Datei wird dann unter dem richtigen Namen gespeichert.

2.3.5 (Automatisches) Einrücken

Diese Funktion rückt eine Zeile automatisch nach Eingabe von Return um so viel ein, wie die vorherige Zeile durch Leerzeichen bzw. Tabs eingerückt wurde.

2.3.6 Cursor

Die Standardeinstellung des Cursors ist ein Blinken, er kann jedoch auch so eingestellt werden, daß er feststeht.

2.3.7 MonST laden

Diese Option ist standardmäßig an und bewirkt, daß jedesmal, wenn `GENST.PRG` geladen wird, MonST mitgeladen wird. MonST ist so per Tastendruck ohne Diskettenzugriff sofort verfügbar. Das Ausschalten der Option bringt einen Speichergewinn von ca. 24 KByte.

Der neue Wert der Einstellung wird nur dann wirksam, wenn Sie die Voreinstellungen abspeichern und `GENST.PRG` neu starten. Es sollte beachtet werden, daß AMonST schon installiert zu haben nicht das gleiche ist, wie MonST beim Start von GenST zu laden.

2.3.8 Voreinstellungen sichern

Wenn Sie den »Abbruch«-Knopf betätigen, werden sämtliche Änderungen, die Sie in der Voreinstellungen-Dialogbox gemacht haben, ignoriert; mit »OK« werden die Änderungen wirksam. Wenn die von Ihnen gewählte Konfiguration permanent gespeichert werden soll, klicken Sie auf »Sichern«. Die Datei `GENST2.INF` wird erstellt und enthält die von Ihnen gewählten Voreinstellungen sowohl aus der Assemblieren-Box als auch aus der Voreinstellungen-Box.

2.4 Das Programm-Menü

Das Programm-Menü enthält alle Funktionen, die mit dem Assemblieren oder der Ausführung von Programmen zu tun haben.

2.4.1 Assemblieren

Um das Programm, das Sie z.Zt. editieren, zu assemblieren, wählen Sie entweder »Assemblieren« oder drücken Alt-A. Es erscheint die Assemblierungs-Dialogbox, die verschiedene Optionen enthält; hier wird nur auf die Ausgabe-Option eingegangen, die weiteren Optionen werden im Assembler-Teil besprochen.

GenST kann auf Diskette, in den Speicher, oder nirgendwohin (»Nein«) assemblieren; die »Nirgendwo«-Option ist nützlich, wenn nur ein Syntaxcheck durchgeführt werden soll. In den Speicher assemblieren ist ideal zum Ausprobieren eines Programms, da nicht auf die Diskette geschrieben wird. Wenn Sie in den Speicher assemblieren, müssen Sie die Speichermenge angeben, in die das Programm hineinassembliert wird; der Standardwert ist 20 KByte, genug für ein durchschnittlich großes Programm mit Symbolen oder ein größeres Programm ohne Symbole.

Wenn Sie die Meldung »Programmspeicher voll« erhalten, heißt das, daß die von Ihnen angegebene Speichermenge nicht ausreicht, um Ihr Programm vollständig zu assemblieren; Sie müssen also den Programmspeicher vergrößern. Natürlich hat ein größerer Programmspeicher zur Folge, daß der Assembler weniger Platz zum Arbeiten hat. Falls der Assembler mit der Meldung »ungenügender Speicher« abbricht, müssen Sie die Programm-Speichergröße verringern.

Wenn die Programm-Speichergröße mit dem für den Assembler notwendigen Platz nicht zu vereinbaren ist, müssen Sie auf Diskette assemblieren. Wenn Sie auf Diskette assemblieren, wird die Größeneinstellung des Programmspeichers ignoriert und der gesamte verfügbare Speicherplatz dem Assembler zur Verfügung gestellt. Wenn Ihr Programm auf Diskette assembliert wird, ohne daß der Assembler sich einen Namen für diese Datei bilden kann, wird der Name Ihres Programms mit `NONAME` anfangen.

Wenn Sie auf »Assemblieren« klicken oder Return drücken, wird der Assembler gestartet. Am Ende des Vorgangs wird auf einen Tastendruck gewartet, damit Sie ggf. erscheinende Meldungen in Ruhe lesen können, bevor Sie zum Editor zurückkehren. Wenn in Ihrem Programm Fehler auftreten, wird der Cursor des Editors bei der Rückkehr auf der ersten fehlerhaften Zeile Ihres Programmtextes stehen; mit Alt-J gelangen Sie dann an den nächsten Fehler.

2.4.2 Programme ausführen

Wenn Sie »Ausführen« im Programm-Menü wählen, können Sie ein in den Speicher assembliertes Programm ausführen; das Tastaturkürzel hierfür ist Alt-X. Wenn der Vorgang beendet wird, kehren Sie in den Editor zurück. Falls Ihr Programm nicht vollständig assembliert wurde, kann es nicht ausgeführt werden.

Wenn Ihr Programm abstürzt, kann es sein, daß Sie nicht wieder in den Editor gelangen und neu »booten« müssen. Es ist also ratsam, den Quelltext vor der Ausführung des Programmes abzuspeichern.

Vorsicht: Wenn Ihr Programm mit Fehlern, wie z.B. undefinierten Symbolen, vollständig assembliert wurde, können Sie es zwar ablaufen lassen, jedoch ist die Gefahr eines Absturzes vorhanden.

Wenn Sie ein Programm, egal welcher Art, vom Editor aus ausführen, kann es vorkommen, daß der Rechner am Programmanfang oder zum Programmende hin zu »hängen« scheint. Dies ist ein Fehler im GEM, über den wir keine Kontrolle haben; der Mauszeiger befindet sich dabei in der Menüleiste. Wenn Sie die Maus nach unten bewegen, müßte alles normal weiterlaufen.

2.4.3 Debuggen

Wenn Sie ein Programm, welches in den Speicher assembliert wurde, debuggen wollen, wählen Sie »Debuggen« oder Alt-D. Dadurch wird MonST aufgerufen und kann Ihr Programm, das ggf. Symbole enthält, debuggen. Wenn Sie Ctrl-C eingeben, wird Ihr Programm abgebrochen und Sie gelangen in den Editor zurück. Die Bildschirm-Initialisierung hängt vom Menüpunkt »GEM« ab, der weiter unten erläutert wird. Wenn Sie »MonST laden« in den »Voreinstellungen« nicht gewählt haben, sind weder »Debuggen« noch »MonST« wählbar.

2.4.4 MonST

Wenn Sie »MonST« wählen, oder Alt-M drücken, wird MonST aufgerufen, und zwar so, als ob Sie ihn vom Desktop aufgerufen hätten, also ohne Diskettenzugriff (so wie es auch z.B. beim Debuggen der Fall ist). Ctrl-C bringt Sie wieder in den Editor zurück. Die Bildschirm-Initialisierung hängt hier auch von GEM ab.

2.4.5 GEM

Normalerweise wird bei »Ausführen«, »Debuggen« und »MonST« eine GEM- Bildschirm-Initialisierung vorgenommen. Bei TOS-Programmen braucht man aber einen leeren Bildschirm und einen blinkenden Cursor. Wenn der Menüpunkt »GEM« mit einem Haken gekennzeichnet ist, wird der Bildschirm auf ein GEM-Programm vorbereitet, wenn nicht, wird der Bildschirm auf ein TOS-Programm vorbereitet. Die Einstellung dieser Option wird beim Abspeichern der Voreinstellungen beibehalten.

Vorsicht: Ein TOS-Programm mit einem GEM-Bildschirm ist möglich, sieht aber unschön aus. Ein GEM-Programm darf aber nie mit der TOS- Bildschirm-Initialisierung ausgeführt werden, da sonst ein Absturz sehr wahrscheinlich wird.

2.4.6 Nächster Fehler

Hier ist so gut wie keine Veränderung gegenüber der Version 1 zu erkennen. Der einzige Unterschied ist, daß bei der Rückkehr vom Assembler in den Editor der Cursor auf der ersten fehlerhaften Zeile platziert ist.

2.4.7 Anderes Programm

Mit dieser Funktion, die auch mit Alt-O aufgerufen werden kann, können Sie ein anderes Programm vom Editor aus laufen lassen; wenn das Programm beendet wird, kehren Sie in den Editor zurück. Die Hauptanwendungen dieser Funktion sind das Ausführen von assemblierten Programmen und der Aufruf des Linkers. Sowohl TOS- als auch GEM-Programme können ausgeführt werden. Voraussetzung dafür ist natürlich, daß genug Speicher vorhanden ist.

Wenn Sie »Anderes Programm« aufrufen, erscheint zunächst eine Warnung mit dem Inhalt, daß Sie Ihren Quelltext nicht gesichert haben. Danach erscheint die File-Selector-Box, in der Sie das auszuführende Programm auswählen. Wenn es ein TOS- oder ein TTP-Programm ist, werden Sie zusätzlich nach einer Kommandozeile für das Programm gefragt. Sie können in der Kommandozeile das »%« Zeichen angeben; dies wird dann durch den Namen (und nicht durch die Extension) der gerade editierten Datei ersetzt. Mit »%%« wird ein echtes »%« Zeichen in die Kommandozeile gesetzt.

Die Bildschirm-Initialisierung hängt von der Extension des Programmes ab, und nicht von der Einstellung von GEM.

2.5 Benutzer von Saved!

Wenn Sie die Path-Funktion Saved!s benutzen und Ihr System zum automatischen Laden von `GENST.PRG` mit einem Doppelklick auf die Quelldatei konfiguriert haben, gilt die Einschränkung nicht, daß sich `GENST.PRG` und Ihre Quelltexte im gleichen Ordner befinden müssen. Der Editor sucht nach der `GENST2.INF`-Datei zuerst im aktuellen Ordner (der Ordner, in dem sich die Datei befindet, auf die Sie

geklickt haben), und dann in den Ordnern des Paths. Wenn Sie die Voreinstellungen sichern, wird die Datei an der Stelle gesichert, an der sie gefunden wurde.

Sie können Saved! mit dem Tastaturkürzel Shift-Clr aufrufen. Dies funktioniert nur, wenn das Accessory auf Ihrer Bootdiskette `SAVED!.ACC` oder `SAVED.ACC` heißt.

2.6 Der Assembler GenST

Der Assembler von Devpac ST, GenST, ist für die Version 2 völlig neu geschrieben worden. Nur so ist es möglich gewesen, die gewünschten Verbesserungen zu machen. Einige der neuen Merkmale von GenST2 sind die Assemblierungs-Geschwindigkeit von bis zu 75.000 Zeilen pro Minute, Symbolsignifikanz von 127 Zeichen und die `INCBIN`-Direktive, mit der eine Binärdatei, wie z.B. eine Bildschirmgrafik, direkt in Ihr Programm eingebunden werden kann. GenST generiert Code, der ebenso mit Pascal Plus linkbar ist.

GenST gibt es in zwei Ausführungen: einmal direkt vom Editor aus aufrufbar und einmal als alleinstehendes Programm.

2.7 Den Assembler aufrufen

2.7.1 Vom Editor aus

Sie können vom Editor aus durch Wählen der »Assemblieren«-Funktion im Programm-Menü oder mit Alt-A den Assembler aufrufen.

Programmtyp

Hier wählen Sie zwischen ausführbarem, GST-linkbarem oder DRI- linkbarem Code. Die Unterschiede zwischen den drei Programmtypen werden später genau erklärt.

Groß/Klein

Hier können Sie wählen, ob bei Symbolen Unterschiede wegen der Groß- oder Klein-Schreibung gemacht werden sollen. Wenn Sie »gleich« wählen, dann sind die Symbole »Start« und »start«, vom Assembler aus gesehen, dasselbe Symbol; wenn »Verschieden« gewählt wird, sind die beiden zwei verschiedene Symbole.

Debug-Information

Wenn Sie Ihr Programm noch debuggen, ist es nützlich, daß die Symbole im fertigen Programm enthalten sind. Die Symboloptionen sind »Normal« und »Erweitert«: das HiSoft-erweiterte Debug-Format erlaubt es, bis zu 22 Zeichen lange Symbole zu verwenden, anstatt der üblichen 8 Zeichen Länge.

Listing

Sie können hiermit die Ausgabe des Listings wählen: auf dem Drucker, auf Diskette, auf der eine Datei mit gleichem Namen und der Extension `.LST` angelegt wird, oder auf den Bildschirm. Wenn Sie keine Ausgabe wollen, können Sie auch diese Option wählen.

Assemblieren

Hier können Sie die Geschwindigkeit des Assemblers bestimmen. Normalerweise sollten Sie den »Schnell«-Knopf betätigt lassen, wenn jedoch der Speicherplatz zu gering wird, sollten Sie »Langsam« wählen. Dadurch wird der Assembler gezwungen, so wenig Platz wie möglich zu belegen und den Zugriff auf Diskette zu verlangsamen. In der `.TTP`-Version wird der »Langsam«-Modus mittels der Option `-M` in der Kommandozeile eingeschaltet.

Output

Wie bereits erwähnt, können Sie hier wählen, welches Programm aus Ihrem Quelltext erzeugt wird.

- »Nein« heißt, daß nur ein Syntaxcheck gemacht wird und kein Programm erzeugt wird.
- »Speicher« heißt, daß das Programm in den Speicher assembliert wird, und so bereitsteht, um von MonST debuggt zu werden, oder auch einfach zur Ausführung gebracht wird. Es wird kein Diskettenzugriff gemacht, außer dann, wenn eine Include-Datei gebraucht wird, die meistens nur ein Mal gelesen wird; bei der Version 1 wurde eine Include-Datei immer zweimal gelesen.
- »Diskette« bedeutet, daß das Programm wie gewohnt auf Diskette erzeugt wird. Die Regeln der Namensgebung für die Dateien werden in Kürze erläutert.

Wenn Sie alle gewünschten Optionen gewählt haben, klicken Sie auf »Assemblieren« oder drücken Sie Return. Der Ablauf wurde oben schon erläutert.

2.8 Der Stand-alone-Assembler

Wenn Sie die `.TTP`-Version des Assemblers aufrufen, werden Sie nach einer Kommandozeile gefragt; die Kommandozeile hat das unten beschriebene Format. Wenn Sie keine Kommandozeile übergeben wollen, geben Sie nur Return ein, Sie kehren dann zum Desktop zurück. Am Ende des Assemblier-Vorganges wird auf einen Tastendruck gewartet. Wenn dem Assembler beim Aufruf eine Kommandozeile übergeben wurde, wird kein Tastendruck abgewartet, da angenommen wird, daß der Assembler von einem CLI oder einer Batch-Datei aus aufgerufen wurde.

2.8.1 Format der Kommandozeile

Die Kommandozeile hat das Format

```
Hauptdatei <-Optionen> [-Optionen]
```

Die Hauptdatei ist der Name der Datei, die assembliert werden soll; wenn keine Extension angegeben wurde, wird `.S` angenommen. Optionen sollte ein »-« Zeichen voranstehen. Erlaubte Optionen sind (zusammen mit den äquivalenten `OPT`-Direktiven):

B	keine Binärdatei erzeugen
C	GROSS/klein egal (`OPT C-`)
D	Debug (`OPT D+`)

L GST-linkbarer Code (`OPT L+`)

L2 DRI-linkbarer Code (`OPT L2`)

O Name der erzeugten Datei, folgt dem »O« ohne Leerschritt dazwischen

P Name der Listing-Datei, folgt dem »P« ohne Leerschritt dazwischen

Q In jedem Fall auf einen Tastendruck am Ende warten

T Der Tab-Abstand; die Zahl folgt dem »T« ohne Leerschritt;
In der `.TTP`-Version wird die Tab-Einstellung durch die
Option `-Txx` geändert, z.B. `-T10`.

X Erweiterter Debug (`OPT X+`)

Wenn keine Optionen auf der Kommandozeile angegeben werden, wird eine ausführbare Programm-Datei erzeugt, deren Name auf dem der Source-Datei basiert, allerdings ohne Listing und mit Unterscheidung zwischen Groß- und Kleinschrift.

Beispiele:

```
test -b
```

assembliert die Datei `test.s` ohne eine Datei zu erzeugen; dies wäre ein Syntaxcheck, mehr nicht.

```
test -om:test.prg -p
```

assembliert `test.s` in die Datei `m:test.prg` und erzeugt eine Listing-Datei namens `test.lst`.

```
test -l2dpprn: -t10
```

assembliert `test.s` in DRI-linkbares Format mit vollständiger Debug- Information und schickt das Listing an die parallele Schnittstelle. Wenn Sie ein Listing an die serielle Schnittstelle senden wollen, müssen Sie `AUX:` angeben. Der Tab-Abstand des Listings beträgt 10 Zeichen.

2.8.2 Dateinamen

In GenST ist geregelt, wie der Name einer Ausgabedatei kreiert wird. Der Name ist außerdem abhängig von den angegebenen Optionen, wie z.B. `-O`, und der `OUTPUT`-Direktive.

- Wenn ein Dateiname explizit angegeben wird, dann ist
  Name=angegebenerName
- Wenn die Output-Direktive nicht benutzt wurde, dann ist
  Name=Source-Name + `.PRG`, `.BIN` oder `.O`
- Wenn die Output-Direktive eine Extension angibt, dann ist
  Name=Source-Name + Extension vom Output
- Sonst ist
  Name=Name vom Output

2.9 Der Assembliervorgang

GenST ist ein Zwei-Pass-Assembler; während des ersten Passes wird der Quelltext im Speicher, ggf. auch von Diskette, verarbeitet und eine Symboltabelle gebildet. Wenn Syntaxfehler während des ersten Passes gefunden wurden, werden diese angegeben, der zweite Pass wird nicht gestartet. Während des zweiten Passes werden die mnemonischen Befehle in binäre Instruktionen umgewandelt, ein Listing kann ausgegeben - wenn erforderlich mit einer Symboltabelle - und eine Binärdatei erzeugt werden. Während des zweiten Passes gefundene Fehler und Warnungen werden angegeben.

Während des Assemblierens kann jede Bildschirm-Ausgabe mit Ctrl-S angehalten werden und mit Ctrl-Q weiterlaufen. Mit Ctrl-C kann der Assembler abgebrochen werden; die bis dahin erzeugte Datei ist aber unvollständig und sollte nicht ausgeführt werden.

2.9.1 In den Speicher assemblieren

Um die turn-around-Zeiten so gering wie möglich zu halten, kann GenST in den Speicher assemblieren, damit Ihr Programm sofort ausgeführt oder mit Hilfe von MonST sofort debuggt wird. Um dieses zu erreichen, wird der sog. Programmspeicher benutzt, dessen Größe in der Assemblierungs-Dialogbox einzustellen ist. Wenn Sie ohne Debug- Information arbeiten, kann der gesamte Programmspeicher für Ihr Programm benutzt werden; wenn Sie aber mit Debug arbeiten, wird der Programmspeicher sowohl mit Ihrem Programm als auch mit der Debug- Information gefüllt.

Ein Programm, das im Speicher ausgeführt wird, ist ein normales GEMDOS-Programm. Es sollte daher mit einem `pterm`- oder `pterm0`-Aufruf beendet werden, wie z.B.

```
clr.w   -(sp)
trap    #1
```

Programme können sich selbst modifizieren, sind aber bei einer erneuten Ausführung wieder im Urzustand.

Die Programmspeichergröße und die in der Dialogbox eingestellten Optionen des Assemblers werden beim Abspeichern der Voreinstellungen mit abgespeichert.

2.9.2 Arten der Binärdateien

Es gibt sechs verschiedene Typen von Binärdateien, die GenST produzieren kann. Sie unterscheiden sich äußerlich an den Extensions:

- `.PRG` Ein GEM-Programm, welches Fenster benutzt
- `.TOS` Ein TOS-Programm, welches keine Fenster benutzt
- `.TTP` Ein TOS-Programm, welches eine Kommandozeile braucht
- `.ACC` ein Desk-Accessory
- `.BIN` nicht ausführbare, linkbare Datei im GST-Format
- `.O` nicht ausführbare, linkbare Datei im DRI-Format

Die ersten drei Arten können vom Desktop aus mit einem Doppelklick ausgeführt werden; sie unterscheiden sich in der Vorbereitung, die das Betriebssystem macht, bevor das Programm ausgeführt wird. Bei einer `.PRG`-Datei wird der Bildschirm gelöscht und mit dem Muster des Desktops versehen; außerdem wird an der Bildschirm-Oberseite Platz für eine Menüleiste gemacht. Bei `TOS`- und `TTP`-Programmen wird der Bildschirm bis auf die Hintergrundfarbe gelöscht, ein blinkender Cursor an den oberen linken Bildschirmrand gesetzt und die Maus ausgeschaltet. Wenn Sie auf ein `TTP`-Programm doppelklicken, dann erscheint vor Programmaufruf eine Dialogbox, in der Sie eine Kommandozeile dem Programm übergeben können.

`.ACC`-Dateien sind zwar ausführbar, jedoch nicht vom Benutzer. Sie werden vom AES während des Bootvorganges oder während eines Wechsels der Bildschirm-Auflösung geladen und initialisiert.

`.BIN`- und `.O`-Dateien können nicht ausgeführt werden; sie müssen erst von einem Linker verarbeitet werden - meist zusammen mit anderen linkbaren Dateien - um zu einem vollständigen Programm zu werden. Es gibt auf dem ST zwei Standard-Formate, nämlich das GST-Format und das DRI-Format. Die Unterschiede zwischen diesen Formaten werden später erläutert.

Die einzige Ausnahme von diesen Regeln sind Programme, die im `AUTO`-Ordner ausgeführt werden. Sie müssen die Extension `.PRG` haben, obwohl sie `TOS`-Programme sein müssen.

2.10 Verschiedene Veränderungen gegenüber der Version 1

- Es gibt drei reservierte Symbole: `__LK`, `__RS` und `__G2`. Ihre Bedeutungen werden später erläutert.
- Es gibt drei neue Operatoren, die alle mit der Priorität unterhalb der Addition und der Subtraktion liegen: Gleichheit »=«, kleiner als »<« und größer als »>«.
- Oktale Konstanten können jetzt mit »@« bezeichnet werden.
- Wenn Sie die Adressierungsart »Adreßregister indirekt« mit Index benutzen, können Sie den Offset weglassen.

  ```
          move.l  (a3,d2.l),d0
  ```

  wird als

  ```
          move.l  0(a3,d2.l),d0
  ```

  assembliert.
- Die Daten- und Adreßregister können auch als `R0` bis `R15` bezeichnet werden, um Kompatibilität mit anderen Assemblern zu gewährleisten. `R0-R7` entsprechen `D0-D7`, `R8-R15` entsprechen `A0-A7`.

2.10.1 Lokale Labels

GenST2 unterstützt lokale Labels. Dies sind Labels, die einem bestimmten Teil des Sources angehören. Sie beginnen mit einem Punkt (».«) und *sind vom vorhergehenden nicht-lokalen Label abhängig*.

```
länge1  move.l  4(sp),a0
.loop   tst.b   (a0)+
        bne.s   .loop
```

```
        rts
länge2  move.l  4(sp),a0
.loop   tst.b   -(a0)
        bne.s   .loop
        rts
```

Es gibt zwei Labels namens `.loop`, von denen das erste zu `länge1`, das zweite zu `länge2` gehört.

Um Verwechslung mit Adressierungsarten zu vermeiden, sind `.W` und `.L` nicht als lokale Labels zulässig. Eine besondere Form eines lokalen Labels ist die Form `1234$`, das aus Dezimalzahlen besteht und mit »$« endet. Dadurch ist Kompatibilität mit anderen Assemblern gegeben.

2.10.2 Erweiterungen des Befehlssatzes

Der vollständige 68000er-Befehlssatz wird von GenST2 unterstützt. Es gibt aber stilistische Abkürzungen, die GenST2 auch akzeptiert:

Condition Codes

Die alternativen Condition Codes `HS` und `LQ`, äquivalent mit `CC` und `CS`, werden bei `Bcc`, `DBcc` und `Scc` unterstützt.

Branch-Befehle

Um einen kurzen Branch explizit anzugeben, benutzen Sie `Bcc.B` oder `Bcc.S`. Einen Branch mit Wort-Länge geben Sie mit `Bcc.W` an; Sie können aber auch den Optimierer die Branch-Länge wählen lassen. `Bcc.L` wird aus Kompatibilitätsgründen zu GenST1 beibehalten, obwohl es genaugenommen ein 68020-Befehl ist; beim Vorkommnis eines `Bcc.L` wird eine Warnung generiert.

Ein `BRA.S` zum darauffolgenden Befehl ist nicht erlaubt und wird, mit einer Warnung, in einen `NOP` verwandelt. Ein `BSR.S` zum darauffolgenden Befehl ist nicht erlaubt und erzeugt einen Fehler.

BTST

`BTST` ist einzigartig unter den Bit-Test-Befehlen, weil es `PC`-relative Adressiermodi unterstützt.

CLR

`CLR An` ist nicht erlaubt, benutzen Sie statt dessen `SUB.L An,An`; die Flags werden von diesem Befehl aber nicht beeinflußt.

CMP

Wenn der Source-Operand immediate ist, wird `CMPI` erzeugt, wenn der Ziel-Operand ein Adreßregister ist, wird `CMPA` benutzt. Wenn die Adressierungsarten beider Operanden post-inkrement benutzen, wird ein `CMPM` generiert.

DBcc

`DBRA` wird für `DBF` akzeptiert.

ILLEGAL

Das Wort $4AFC wird generiert.

LINK

Wenn der Wert positiv oder ungerade ist, wird eine Warnung ausgegeben.

MOVE from CCR

Dies ist ein Befehl der Prozessoren 68010 und höher. Er wird in ein »MOVE from SR« konvertiert.

MOVEQ

Wenn der Wert im Bereich 128 bis 255 inklusive ist, wird eine Warnung ausgegeben. Wenn Sie ausdrücklich ein .L angeben, wird keine Warnung erzeugt.

2.10.3 Assembler-Direktiven

INCBIN Dateiname

Mit dieser Direktive wird eine Datei vollständig in Ihr Programm miteinbezogen. Dies ist z.B. nützlich für Grafiken, die mit anderen Programmen erstellt wurden und von Ihrem Programm in binärer Form benutzt werden; es entfällt eine Konvertierung in DC-Direktiven. INCBIN startet an einer geraden Grenze und wird mit einem Nullbyte gepaddet, wenn die Datei eine ungerade Länge besitzt.

OPT

Es gibt verschiedene neue Optionen, aber auch solche, die sich gegenüber der Version 1 verändert haben.

- A Auto-PC

  Um Ausführzeit zu sparen und die Programmgröße zu reduzieren, kann die Option A+ verwendet werden. Sie erlaubt Automatik-PC- Modus, wo immer er möglich ist. Die Zeile

  ```
  MOVE.L int_in,d0
  ```

  würde zu

  ```
  MOVE.L int_in(pc),d0
  ```

  assembliert werden. Dadurch wird jedoch kein positionsunabhängiger Code erzeugt. A+ kann z.B. im Programm GEMTEST effektiv eingestellt werden. Unter Umständen wird die Option nicht beachtet, z.B. beim Lesen des absoluten Speichers, oder wenn der Ausdruck .L gebraucht wird.

- C Groß/klein-Unterscheidung und Signifikanz

  Standardmäßig unterscheidet GenST2 zwischen Groß- und Klein-Schreibung von Labels. Labels haben standardmäßig eine Signifikanz von 127 Zeichen, d.h. daß die ersten 127 Zeichen eines Labels zur Unterscheidung von anderen Labels bewertet werden. Sie können, wie bei GenST1, C- angeben, wenn

keine GROSS/klein-Unterscheidung gemacht werden soll. Sie können auch die Signifikanz verändern, indem Sie eine Dezimalzahl zwischen dem `C` und dem `+` oder `-` angeben, z.B. `C16+` um eine Signifikanz von 16 Zeichen mit einer Groß/klein-Unterscheidung einzustellen.

- `L` Linker-Modus

  Standardmäßig produziert GenST ausführbaren Code. Mit `L+` wird GST-linkbarer Code erzeugt, mit `L2` wird DRI-linkbarer Code erzeugt. Mit `L-` kann ausdrücklich ausführbarer Code erzeugt werden. Eine `OPT L-` Direktive muß in der allerersten Zeile der Quelldatei stehen.

- `O` Optimierung

  GenST2 besitzt die Fähigkeit, manche Befehle in kürzere oder schnellere zu verwandeln. Standardmäßig wird zwar keine Optimierung vorgenommen, jedoch kann jede Art von Optimierung wahlweise hinzugeschaltet werden.

  `O0+`
  : Optimiert Rückwärts-Branches wenn möglich, kann mit `O0-` wieder abgeschaltet werden

  `O1+`
  : Optimiert Adreß-Register indirekt mit Offset in Adreßregister. Wenn der Offset Null ist, kann mit `O1-` abgeschaltet werden.

    ```
    move.l  Wert(a0),d3     wird zu
    move.l  (a0),d3         wenn Wert gleich Null
    ```

  `O+`
  : schaltet alle Optimierungen an.

  `O-`
  : schaltet alle Optimierungen ab.

  `OW-`
  : schaltet die mit den Optimierungen verbundenen Warnungen aus. Kann mit `OW+` wieder eingeschaltet werden.

- `P` Positionsunabhängigkeit

  Wenn diese Option mit `P+` zugeschaltet wird, wird der erzeugte Code auf Positionsunabhängigkeit hin geprüft. Wenn ein Befehl nicht positionsunabhängig ist, wird ein Fehler erzeugt. Mit `P-` kann die Option wieder ausgeschaltet werden; die Option ist dann im Standardzustand.

- `T` Typen-Überprüfung

  GenST2 kann Programmierfehler entdecken, indem es Ausdrücke dahingehend prüft, ob sie absolut oder relativ sind. Da für manche Anwendungen und Programmierstile eine solche Meldung hinderlich sein kann, ist sie mit `T-` aus-, und mit `T+` wieder einschaltbar.

  ```
  main            bsr     init
                  lea     main(a6),a0
                  move.l  (main).w,a0
  ```

  Diese Zeilen würden normalerweise einen Fehler erzeugen, da `main` ein relativer Ausdruck ist und GenST in beiden Fällen einen absoluten Ausdruck erwartet. Falls aber der Code auf einer anderen

68000er-Maschine laufen soll, kann dies zulässig sein: also müßte `T-` angegeben werden.

- `U` Lokale Labels mit »_«

  Mir `U+` können Sie angeben, daß lokale Labels mit »_« anstatt mit einem Punkt beginnen. Diese Option existiert, um Kompatibilität mit Compilern der Firma Prospero zu gewährleisten.

- `X` Erweiterter Debug

  Dies ist eine erweiterte Version der Option `D`. Hiermit werden Symbole mit bis zu 22 Zeichen Länge anstatt maximal 8 Zeichen wie bei `D` ausgegeben.

  Zusammenfassung der Optionen:

  Alle Optionen GenST2s werden hier zusammengefaßt. Die Standardeinstellungen sind in Klammern angegeben.

`C`	Groß/Klein-Unterscheidung und Signifikanz (`C127+`)
`D`	Debug-Information (`D-`)
`L-`	ausführbarer Code (Standard)
`L+`	GST-linkbarer Code
`L2`	DRI-linkbarer Code
`M`	Makro-Expansion (`M-`)
`O`	Optimierungen (`O-`)
`P`	auf Positionsunabhängigkeit prüfen (`P-`)
`S`	Symboltabelle ins Listing (`S-`)
`T`	Typen-Überprüfung (`T+`)
`W`	Warnungen (`W+`)
`U`	Lokale Labels mit »_« (`U-`)
`X`	Erweiterter Debug (`X-`)

DCB

```
<Label> DCB.B           Anzahl,Wert
<Label> DCB.W           Anzahl,Wert
<Label> DCB.L           Anzahl,Wert
```

Diese Direktive erlaubt es, Datenblöcke bestimmter Größe mit konstantem Wert zu erzeugen. »Anzahl« gibt die Größe des Blocks an, »Wert« den sich wiederholenden Wert.

OUTPUT Dateiname

Mit dieser Direktive läßt sich der Name der Ausgabe-Datei bestimmen. Wenn der angegebene Dateiname mit einem Punkt anfängt, wird nur die Extension der erzeugten Datei beeinflußt. Der Name wird sonst nach den oben beschriebenen Regeln gebildet.

`__G2` (reserviertes Symbol)

Dieses Symbol kann mit der `IFD`-Bedingung verwendet werden, um festzustellen, ob mit GenST2 assembliert wird. Der Wert dieses Symbols ist die Version des Assemblers und immer absolut.

`REPT`

```
<Label> REPT   Ausdruck
        ENDR
```

Oft ist es nützlich, einen oder mehrere Befehle zu wiederholen. Mit `REPT` ist dies durchführbar. Die zu wiederholenden Befehle werden zwischen `REPT` und `ENDR` eingegeben. Die Anzahl der Wiederholungen wird vom angegebenen Ausdruck bestimmt. Wenn der Ausdruck Null oder negativ ist, wird kein Code generiert. REPTs können nicht verschachtelt werden.

```
        REPT   512/4           einen Sektor kopieren
        move.l (a0)+,(a1)+
        ENDR
```

Dies ist ein Beispiel, um die Wirksamkeit des `REPT` zu zeigen: so muß im Quelltext nicht 128-mal die `move.l` Zeile stehen.

Vorsicht: Man sollte keine Labels innerhalb eines REPTs definieren, da es sonst zu mehrfach definierten Labels und den dazugehörigen Fehlermeldungen kommt.

`LIST`

Zusätzlich zur `LIST`-Direktive gibt es jetzt `LIST+` und `LIST-`. Ein Zähler wird bei jedem `LIST+` um eins höher gesetzt und bei jedem `LIST-` um eins erniedrigt. Wenn der Zähler Null oder positiv ist, wird ein Listing ausgegeben; wenn der Zähler negativ ist, wird kein Listing erzeugt. Der normale Zählerwert ist -1 (kein Listing), außer dann, wenn beim Assemblerstart ein Listing angefordert wird, wodurch der Zähler auf Null gesetzt wird. So haben Sie eine flexiblere Kontrolle über Listings, besonders bei Include-Dateien. `LIST` alleine setzt den Zähler auf 0, `NOLIST` auf -1.

`SUBTTL String`

Ein »Unter«-Titel wird hiermit angegeben, er kann in Hochkommas (»'«) eingegrenzt sein. Die erste Direktive setzt den Unter-Titel der ersten Seite usw.

`FORMAT Parameter<,Parameter,...>`

Hiermit können Sie das Format einer Zeile im Listing bestimmen. Jeder Parameter entspricht einem Feld im Listing und besteht aus einer Zahl von 0 bis 2, gefolgt von einem »+« (um das Feld darzustellen) oder »-«:

0 Zeilennummer, dezimal

1 SECTION-Name bzw. -Zahl und `PC`

2 Hex-Daten in Worten, bis zu 10 Wörter (außer, wenn der Drucker breiter als 80 Zeichen ist)

`Label = Ausdruck`

Das Gleichheitszeichen kann anstatt `EQU` benutzt werden.

`Label REG Registerliste`

Mit der `REG`-Direktive können Sie einem Symbol eine Registerliste zur Verwendung mit `MOVEM` zuordnen. Somit wird die Gefahr verringert, am Anfang und am Ende einer Routine unterschiedliche Registerlisten zu haben. Mit `REG` definierte Symbole können ausschließlich mit `MOVEM` benutzt werden.

`RSSET Ausdruck`

Mit `RSSET` können Sie dem RS-Zähler einen bestimmten Wert zuordnen.

`__RS` (reserviertes Symbol)

Dieses reservierte Symbol enthält den gegenwärtigen Wert des RS-Zählers.

Bedingte Blöcke

Bedingte Blöcke können mit GenST2 bis zu 65.535 Stufen tief verschachtelt werden.

`IIF Ausdruck Instruktion`

Dies ist eine Kurzform von `IFNE` und betrifft nur eine Zeile; ein `ENDC` sollte nicht mit `IIF` benutzt werden.

2.11 Makros

Makros können jetzt bis zu 36 Parameter haben. Die Parameter beginnen wie bei GenST1 mit einem Backslash »\«. Die zulässigen Parameter- Bezeichner sind die Zahlen 1-9 für die ersten neun Parameter, und a-z oder A-Z für die restlichen 26 Parameter. \0 behält die gleiche Bedeutung wie bei GenST1.

Es gibt eine besondere Form einer Makro-Expansion. Sie können ein Symbol in eine dezimale oder hexadezimale Zahlensequenz verwandeln. Die Syntax ist `\<Symbol>` oder `\<$Symbol>`; das Symbol muß definiert und absolut sein.

Ein Makro-Aufruf kann über mehrere Zeilen gemacht werden; dies ist nützlich bei Makros mit vielen Parametern. Um als Makro erkannt zu werden, muß eine Zeile mit einem Komma beendet werden, das erste Zeichen der nächsten Zeile muß ein »&« sein, gefolgt von Leerschritt(en) oder Tab(s) und den weiteren Parametern.

Die Verschachtelung von Makro-Aufrufen ist nur durch den Arbeitsspeicher begrenzt; Rekursion ist also möglich.

Die Namen der Makros werden in einer separaten Symboltabelle gespeichert und können daher nicht mit anderen Symbolen gleichen Namens kollidieren; außerdem kann der Name des Makros mit einem Punkt beginnen.

Ein Beispiel mit numerischer Substitution:

```
vname           MACRO
                dc.b    \1,\<version>,0
                ENDM
                .
                .
version         equ     42

                vname   <'Buchstabensuchprogramm v'>
```

wird expandiert zu:

```
                dc.b    'Buchstabensuchprogramm' v','42',0
```

Beispiel eines komplexen Makro-Aufrufs:

Nehmen wir an, daß ein Programm eine komplizierte Tabellenstruktur hat, die eine variable Anzahl von Feldern beinhaltet. Ein Makro kann z.B. auch dafür geschrieben werden, daß angegebene Parameter verwendet werden:

```
Tabellen_Eintrag MACRO
                 dc.b   .end\@-*                Längenbyte
                 dc.b   \1                      immer
                 IFNC   '\2',''
                 dc.w   \2,\3                   2. und 3.
                 ENDC
                 dc.l   \4,\5,\6,\7
                 IFNC   '\8',''
                 dc.b   '\8'                    text
                 ENDC
                 dc.b   \9
.end\@           dc.b   0
                 ENDM
```

Beispiel eines Aufrufs:

```
                Tabellen_Eintrag $42,,,t1,t2,t3,t4,<Namen eingeben:>,%0110
```

Dies ist ein gutes Beispiel um zu zeigen, daß Makros das Programmieren enorm erleichtern können. In diesem Fall wird mit einer Datenstruktur gearbeitet, die aus einem Längenbyte (im Makro mit Hilfe von \@ errechnet), zwei optionalen Wörtern, vier Langwörtern, einem optionalen String, einem Byte, und einem Nullbyte besteht. Der Code, der aus diesen Beispiel resultiert, sieht so aus:

```
                dc.b   .end_001
                dc.b   $42
                dc.l   t1,t2,t3,t4
                dc.b   'Namen eingeben:'
                dc.b   %0110
.end_001        dc.b   0
```

2.12 Ausgabedatei-Formate

GenST2 ist sehr flexibel, was Dateiformate angeht. Die verschiedenen Formate, die er erzeugen kann, werden hier erklärt und bewertet. Manche Direktiven haben auch je nach Ausgabeformat andere Wirkungen.

2.12.1 Ausführbare Dateien

Diese Dateien sind sofort vom Desktop aus ausführbar. Eine solche Datei kann Symbole und Relozier-Information enthalten. Die normalen Extensionen für diese Art von Datei sind `.PRG`, `.TOS`, `.TTP` und `.ACC`.

Vorteile: Echte `BSS`-Section, Entwicklungszeit wird reduziert (da nicht gelinkt werden muß).

Nachteile: Wenn mehr als ein Programmierer am Projekt arbeitet, ist es unpraktikabel.

2.12.2 GST-linkbare Dateien

Wenn Sie an einem größeren Projekt arbeiten oder mehrere Programmierer am Projekt beteiligt sind, oder wenn Sie Maschinensprache-Routinen für eine Hochsprache schreiben, werden Sie wahrscheinlich linkbare Dateien erzeugen wollen. Das GST-Format wird von den meisten in England oder auch anderswo produzierten Hochsprachen-Systemen unterstützt. Zu diesen Sprachen gehören HiSoft Basic, Lattice C, Prospero Pascal und Prospero Fortran. Dateien im GST-Format haben meist die Extension `.BIN`.

Vorteile: Sehr flexible Möglichkeiten: importierte Labels können fast überall verwendet werden, wie z.B. auch in Ausdrücken. Libraries können vom Assembler aus erstellt, und durch die verwendete Import-Methode können Typenkonflikte erkannt werden.

Nachteile: Aufgrund des Library-Formats kann das Linken länger dauern, richtige GEMDOS-Sections werden standardmäßig nicht unterstützt (obwohl LinkST eine `BSS`-Section erzeugen kann).

2.12.3 DRI-linkbare Dateien

Dies ist das Original-Linkerformat des ST und wurde ursprünglich von Digital Research für CP/M-68K entwickelt. Es wird, meist durch Konvertierungs-Programme, vom Großteil der in den USA entwickelten Hochsprachen-Systemen unterstützt. DRI-linkbare Dateien haben meist die Extension `.O`.

Vorteile: Selektives Hinzulinken mit Libraries geht schnell, GEMDOS-Sections werden voll unterstützt.

Nachteile: Sehr große Einschränkungen bei importierten Labels, linkbare Dateien sind doppelt so groß wie ausführbare Dateien, und Symbole können maximal nur 8 Zeichen lang sein.

2.12.4 Das richtige Format

Wenn Sie mit einer Hochsprache zusammenarbeiten, haben Sie meistens keine Wahl: Sie müssen das vom Compiler unterstützte Format verwenden. Wenn Sie ausschließlich in Assembler schreiben, werden Sie wohl das ausführbare Format benutzen: die Assemblierzeit ist gering, Linken fällt weg, und Sie können direkt in den Speicher assemblieren, um die Turn-around-Zeiten so klein wie möglich zu halten. Wenn Sie ein größeres Programm schreiben, womöglich im Team, ist linkbarer Code meist am sinnvollsten. Das GST- Format ist dem DRI-Format weitaus überlegen und deshalb auch empfehlenswert.

2.13 Ausgabedatei-Direktiven

Hier werden die Direktiven erklärt, deren Wirkungen vom Ausgabeformat abhängig sind. Die Formate können auf verschiedene Weise gewählt werden: über die Kommandozeile bei `GENST2.TTP`, in der

Assemblieren-Dialogbox bei GENST2.PRG, oder mit der OPT-L-Direktive am Anfang Ihres Programms.

2.13.1 MODULE und SECTION

MODULE Modul-Name

Mit diesem Befehl wird der Anfang eines neuen Moduls festgelegt. Der Name muß in Anführungszeichen stehen, wenn er Leerschritte enthält. Diese Direktive ist nicht unumgänglich, ein Modul namens ANON_MODULE wird automatisch erzeugt.

- Ausführbar

  Diese Direktive wird ignoriert.

- DRI

  Diese Direktive wird ignoriert.

- GST

  Mit dieser Direktive können Sie Maschinensprache-Libraries erstellen. Jedes MODULE ist ein eigenständiges Segment mit eigenen Imports und Exports. Relative Labels gehören jeweils zum eigenen Modul, Sie können also zwei Labels mit demselben Namen in zwei Modulen benutzen, ohne daß sie kollidieren. Absolute Labels sind für alle MODULE global, was für Konstanten sehr nützlich ist.

SECTION Section-Name

Diese Direktive verursacht einen Wechsel zu der genannten Section. Ein Programm kann aus mehreren Sections bestehen, die im endgültigen Programm mit gleichnamigen Sections zusammengefügt werden. Wenn nicht anders angegeben, wird Ihr Programm in SECTION TEXT assembliert. Sie können mit diesem Befehl jederzeit in eine andere Section wechseln.

- Ausführbar

  Erlaubte Namen für Sections sind TEXT (für den Bereich, der das Programm enthält), DATA (für initialisierte Daten) und BSS, ein besonderer Speicherbereich, der vom GEMDOS für Ihr Programm reserviert wird. Das BSS enthält Nullen beim Programmstart und belegt keinen Platz auf der Diskette. Innerhalb des BSS darf nur die DS- Direktive verwendet werden. Wenn Sie den BSS-Bereich für globale Variablen benutzen, können Sie Platz auf der Diskette sparen.

- DRI

  Die obengenannten Regeln für ausführbare Programme gelten hier auch.

- GST

  Es gibt keine Regeln für Namen der Sections. Sections mit gleichem Namen in verschiedenen Modulen werden beim Linken zusammengefügt. Die Anordnung der Sections im endgültigen Programm hängt von der Reihenfolge ab, in der sie von Linker begegnet werden (LinkST hat die SECTION-Direktive, mit der die Reihenfolge zur Linkzeit festgelegt werden kann).

Imports und Exports

Bei beiden linkbaren Formaten ist es wichtig, Symbole importieren und exportieren zu können, was sowohl für relative Symbole wie Programmlabels als auch für absolute Symbole wie Konstanten gilt. Das GST-Format trifft zwischen den beiden Arten eine Unterscheidung, das DRI-Format dagegen nicht. Das GST-Format erlaubt dem Assembler, die Typen zu überprüfen, um Programmierfehler zu finden, die sonst vielleicht übersehen worden wären.

`XDEF Export<,Export>...`

Hiermit werden Labels angegeben, die für andere Module zugänglich gemacht werden sollen. Wenn ein exportierter Label nicht existiert, erscheint eine Fehlermeldung. Es ist nicht möglich, lokale Labels zu exportieren.

- Ausführbar

  Diese Direktive wird ignoriert.

- DRI

  Alle Symbole werden beim Exportieren auf 8 Zeichen Länge gekürzt; es ist ratsam, `OPT C8` zu verwenden.

`XREF`

```
XREF   Import<,Import>...
XREF.L Import<,Import>...
```

Hiermit werden Labels definiert, die aus anderen Modulen importiert werden können; sie müssen dort natürlich exportiert worden sein, da sonst eine Fehlermeldung erscheint. Das normale `XREF` sollte dazu verwendet werden, relative Labels zu importieren, `XREF.L` wird benutzt, um absolute Labels zu importieren. Sie können ein und dasselbe Label mehrmals importieren.

- Ausführbar

  Diese Direktive wird ignoriert.

- DRI

  Das DRI-Format unterscheidet zwar nicht zwischen absoluten und relativen Imports, man sollte sie aber trotzdem angeben, damit der Assembler die Typenüberprüfung vornehmen kann. Wenn Ihre Imports keine Typen haben, sollten Sie `OPT T-` angeben. DRI-Labels haben eine Signifikanz von nur 8 Zeichen.

- GST

  Importieren Sie nur Labels mit dem richtigen Typ; es kann sonst sein, daß die Relozier-Information des endgültigen Programms nicht stimmt.

Imports in Ausdrücken verwenden

- Ausführbar

  Es gibt keine Imports.

- DRI

  Imports können innerhalb von Ausdrücken benutzt werden, allerdings nur ein Import pro Ausdruck. Das Ergebnis eines solchen Ausdrucks muß immer Zahl + bzw. - Import sein. Imports können auch mit beliebig komplexen Ausdrücken kombiniert werden, vorausgesetzt, daß der komplexe Ausdruck vor dem Import angeführt wird:

  ```
  move.l  3+(1<<Zähler+5)+Import
  ```

- GST

  Imports können innerhalb von Ausdrücken benutzt werden, maximal zehn pro Ausdruck. Sie dürfen miteinander nur addiert oder subtrahiert werden, können aber mit beliebig komplexen Ausdrücken kombiniert werden, vorausgesetzt, daß der komplexe Ausdruck vor dem Import angeführt wird:

  ```
  move.l  3+(1<<Zähler+5)+Import1-Import2
  ```

  Wo ein Ausdruck mit einem Import benutzt werden darf, hängt vom Dateiformat ab. Die folgende Tabelle zeigt die erlaubten Kombinationen:

Ausdruck	**GST**	**DRI**	**Beispiel**
PC-Byte	J	N	`move.w import(pc,d3.w)`
			`bsr.s import`
PC-Wort	J	J*	`move.w import(pc),a0`
			`bsr import`
Byte	J	N	`move.b #import,d0`
Wort	J	J	`move.w import(a3),d0`
Langwort	J	J	`move.l import,d0`

  So lange das Symbol nicht in einer anderen Section des Programmes steht, ist dies unzulässig.

- DRI & GST

  Die Benutzung eines Symbols aus einer anderen Section muß als Import gehandhabt werden und unterliegt den obengenannten Regeln.

`COMMENT Bemerkung`

- Ausführbar

  Diese Direktive wird ignoriert.

- DRI

Diese Direktive wird ignoriert.

- GST

  Diese Direktive übernimmt die angegebene Bemerkung in die `.BIN`-Datei; zur Linkzeit wird die Bemerkung angezeigt.

`ORG Ausdruck`

Diese Direktive läßt den Assembler positionsabhängigen Code erzeugen; der PC wird auf den angegebenen Ausdruck gesetzt. Normalerweise brauchen GEMDOS-Programme kein `ORG`, auch wenn sie nicht positionsunabhängig sind; der GEMDOS-Programmlader erledigt die Relozierung des Programms. Die `ORG`-Direktive erlaubt die Erzeugung von ROM-Code oder Code, der für andere 68000er Rechner gedacht ist. Es darf mehr als ein `ORG` im Programm stehen, es wird aber kein »Padding« vorgenommen.

- Ausführbar

  Diese Direktive sollte mit höchster Vorsicht gehandhabt werden, da das erzeugte Programm wahrscheinlich nicht auf dem ST mit einem Doppelklick problemlos ausführbar ist. Die erzeugte *Datei hat am Anfang den GEMDOS-Header*, aber am Ende *keine Relozier-Information*.

- DRI

  Diese Direktive ist nicht erlaubt, die Generierung absoluten Codes ist Aufgabe des Linkers.

- GST

  Ein `ORG` wird dem Linker übergeben, der die Datei mit Nullen »paddet«.

  Vorsicht: Es ist höchst unwahrscheinlich, daß diese Direktive sinnvoll ist, wenn in den Speicher assembliert wird.

`OFFSET <Ausdruck>`

Hiermit wird die Code-Generierung in eine besondere, Assembler-interne Section umgeschaltet. Der Ausdruck, der nicht angegeben werden muß, setzt den `PC` dieser Section. Es werden keine Daten auf Diskette geschrieben, innerhalb dieser Section ist nur die `DS`-Direktive erlaubt. Labels, die innerhalb dieser Section definiert werden, sind absolut. Um einige Systemvariablen des ST zu definieren, sähe der Quelltext so aus:

```
                OFFSET $400
etv_timer       ds.l            1               wird $400 sein
etv_critic      ds.l            1               404
etv_term        ds.l            1               408
ext_extra       ds.l            5               40C
memvalid        ds.l            1               420
```

Solche Offsets gelten als »Moduls«, *nicht* als »Globals«.

`__LK` (reserviertes Symbol)

Dies ist ein reserviertes Symbol; es kann dazu benutzt werden zu erkennen, welche Code-Art erzeugt

wird. Der Wert dieses Symbols ist immer absolut und kann einen der folgenden Werte haben:

0 ausführbar

1 GST linkbar

2 DRI linkbar

DRI Debug

Normalerweise werden nur explizit geXDEFte Labels der Symboltabelle der linkbaren Datei beigefügt. Das Format erlaubt aber sog. lokale Labels, nicht zu verwechseln mit den lokalen Labels von GenST2, welche keine wahren Exports sind und deshalb nicht von anderen Modulen aus benutzt werden können. Diese Symbole werden aber der Symboltabelle des fertigen Programmes hinzugefügt. `OPT D+` gibt alle nicht exportierten Labels als DRI-lokale an.

GST-Libraries schreiben

Wenn Sie eine GST-Library mit Hilfe mehrerer Module in eine Datei schreiben, müssen Sie mit Rückwärts-Verweisen an Imports vorsichtig sein. Innerhalb einer GST-Library sollten »high-level«-Routinen zuerst kommen, »low-level«-Routinen zuletzt. Das folgende Beispiel würde nicht fehlerfrei linken:

```
                MODULE  low_level
                XDEF    low_output
low_output
                ...
                MODULE  high_level
                XDEF    high_output
                XREF    low_output
high_output
                ...
```

Der Grund dafür liegt darin, daß das zweite Modul eine Routine aus dem ersten benutzt; dies ist nicht zulässig. Die fehlerfreie Version sieht so aus:

```
                MODULE  high_level
                XDEF    high_output
                XREF    low_output
high_output
                ...
                MODULE  low_level
                XDEF    low_output
low_output
                ...
```

Beispiele für verschiedene Formate

Hier folgen Beispiele, die die Unterschiede zwischen den einzelnen Formaten verdeutlichen.

Ausführbar

```
                SECTION TEXT
start           lea     string(pc),a0
                move.l  a0,save_str
                bsr     printstring
                bra     quit
```

```
                SECTION DATA
string          dc.b    'Ihren Namen bitte: ',0
                SECTION TEXT
printstring
                move.l  a0,-(sp)
                move.w  #9,-(sp)
                trap    #1
                addq.l  #6,sp
                rts
                SECTION BSS
save_str        ds.l    1
                END
```

DRI linkbar

```
                XREF.L  quit
                SECTION TEXT
start   move.l  #string,a0
                move.l  a0,save_str
                bsr     printstring
                bra     quit
                SECTION DATA
string          dc.b    'Ihren Namen bitte: ',0
                SECTION TEXT
printstring
                move.l  a0,-(sp)
                move.w  #9,-(sp)
                trap    #1
                addq.l  #6,sp
                rts
                SECTION BSS
save_str        ds.l    1
                END
```

Beachten Sie, daß der erste Befehl nicht PC-relativ ist, *da keine PC-relativen Verweise zwischen Sections erlaubt sind*.

GST linkbar

```
                MODULE  TESTPROG
                COMMENT nicht vollständig
                XREF.L  quit
                SECTION TEXT
start           lea     string(pc),a0
                move.l  a0,save_str
                bsr     printstring
                bra     quit
                SECTION DATA
string          dc.b    'Ihren Namen bitte: ',0
                SECTION TEXT
printstring
                move.l  a0,-(sp)
                move.w  #9,-(sp)
                trap    #1
                addq.l  #6,sp
                rts
                SECTION BSS
save_str        ds.l    1
                END
```

3 LinkST, der Linker

LinkST in Devpac ST Version 2 hat sich gegenüber seinem Vorgänger nicht so stark verändert wie z.B.

GenST oder MonST. Es sind lediglich einige Optionen dazugekommen, die im folgenden beschrieben werden.

3.1 Optionen mit einem Buchstaben

Es gibt zwei neue Optionen mit einem Buchstaben:

`B`
: ein `BSS`-Segment im Sinne des GEMDOS wird generiert. Voraussetzung ist, daß es eine Section namens `BSS` gibt.

`X`
: funktioniert wie die Option `D`, nur wird anstatt dem DRI-Symbolformat das HiSoft-erweiterte Format in der Symboltabelle des erzeugten Programmes generiert.

3.2 Direktiven in Kontroll-Files

Es gibt vier neue Direktiven, die innerhalb eines Kontroll-Files verwendet werden können.

SECTION <Name>

Mit dieser Direktive wird die Reihenfolge der Sections im gelinkten Programm festgelegt. Wie in den zwei mitgelieferten Kontroll-Files sollte die Anordnung so aussehen:

```
section text
section data
```

BSS <Name>

Mit dieser Direktive benennen Sie die `SECTION`, die als `BSS`-Segment beim Linken verwendet werden soll. Für Lattice C Benutzer würde die Zeile nach den vorangegangenen `SECTION`-Direktiven

```
bss udata
```

heißen. `BSS` sollte nie mit `DATA` zusammen benutzt werden. Die Section, die angegeben wird, sollte niemals Daten ungleich Null enthalten, da sonst die Fehlermeldung »non-zero data in BSS section« erscheint.

XDEBUG

hat die gleiche Wirkung wie die Option `X`. Es wird eine Symboltabelle im HiSoft-erweiterten Format angelegt.

TRUNCATE

Es werden alle Symbole auf 8 Zeichen Länge gekürzt. Dies ist dann notwendig, wenn Maschinensprache-Routinen mit langen Labels mit Hochsprachen-Programmen mit kurzen Labels gelinkt werden.

3.3 Lattice C-Benutzer

Es wird eine neue `C.LNK`-Datei für Lattice C 3.04 mitgeliefert. Wenn Sie dieses Kontroll-File verwenden, reduzieren sich Ihre Link-Zeiten und die Größe des erzeugten Programms.

Wenn Sie die `-n`-Option des Compilers benutzen, können Sie das HiSoft-erweiterte Debug-Format beim Linken angeben; dies ist nützlich, da beim Debuggen mit MonST2 längere, und deshalb übersichtlichere Symbolnamen, benutzt werden können.

4 MonST - Der symbolische Debugger

4.1 Einleitung

Programme, die in Maschinensprache geschrieben sind, sind wesentlich anfälliger für kleine Fehler als solche, die in einer Hochsprache geschrieben sind. Es ist auch durchaus möglich, mit kleinen Fehlern den ganzen Rechner »abstürzen« zu lassen. Es gibt die verschiedensten Arten von Fehler: triviale Fehler wie ein vergessenes Nullbyte am Ende einer Zeichenkette, Durchschnittsfehler wie ein falsches Ergebnis einer Rechenroutine bis hin zum schweren Fehler, der mit einem spektakulären Absturz zutage kommt.

Um solche Fehler zu finden und zu korrigieren, gehört zum Paket Devpac ST das Programm MonST. MonST ist ein sog. symbolischer Debugger und Disassembler. Er ermöglicht es dem Programmierer, sich Programme und Speicher anzusehen, Programme Schritt für Schritt auszuführen, und die meisten Programmierfehler (wie Bus- oder Adreßfehler) ohne Schaden abzufangen. Da MonST symbolisch arbeitet, müssen Sie sich nicht mit langen Hex-Ziffern herumschlagen, sondern können sich in Ihrem Programm an Ihren eigenen Labels orientieren.

Obwohl MonST ein sog. »low-level« Debugger ist, d.h. er arbeitet auf Maschinensprachenebene, ist es auch ohne weiteres möglich, Programme, die in einer Hochsprache geschrieben worden sind, zu »debuggen«; die einzige Voraussetzung ist, daß der Hochsprachen-Compiler 68000er Maschinencode erzeugt. Es gibt viele Compiler, die die Möglichkeit bieten, Symbole im fertigen Programm zu hinterlassen. So können Sie genau erkennen, in welcher Prozedur Sie sich befinden, während Sie den erzeugten Code des Compilers auf dem Bildschirm vor sich haben. Wir haben MonST selber zum Debuggen von LinkST, welches in C geschrieben ist, verwendet. MonST und GenST sind beide vollständig in Maschinensprache geschrieben.

MonST hat seinen eigenen Bildschirmspeicher. Sie bekommen deshalb keine Probleme, wenn Sie ein Programm mit Grafiken, wie z.B. ein Spiel oder ein GEM-Programm, debuggen; das Programm hat seinen eigenen Bildschirmspeicher, MonST auch. MonST hat auch einen eigenen Bildschirmtreiber und ist deshalb nicht auf das Betriebssystem angewiesen, um seine Bildschirm-Ausgabe zu machen. Sie können so problemlos die Bildschirm-Ausgaberoutinen des AES oder des BIOS »single-steppen«, ohne daß der Debugger darunter leidet.

Auf der Devpac ST-Originaldiskette befinden sich drei Versionen von MonST: alle sind fast gleich zu bedienen. Auf die Unterschiede wird etwas später eingegangen.

4.2 Die Exceptions des 68000

MonST verwendet die Exceptions des 68000 um amok-gelaufene Programme zu stoppen und um »Single-Steppen« zu ermöglichen. Es ist deshalb ganz nützlich zu wissen, was normalerweise geschieht, wenn auf dem ST ein Exception passiert.

Es gibt verschiedene Sorten von Exceptions: die einen sind absichtliche, die anderen eindeutige Fehler. Wenn ein Exception passiert, werden auf dem `SSP` (Supervisor-Stack) bestimmte Daten hinterlassen, der Prozessor geht dann in den Supervisor-Modus über, und springt zum sog. »exception handler«. Dies sind Maschinensprachebefehle, die bei einem Exception (und sonst nie) ausgeführt werden. Wenn MonST geladen ist, fängt er manche Exceptions ab, damit kein Schaden passiert. Die verschiedenen Arten von Exceptions, was sie tun, und MonSTs Reaktion darauf sind in der folgenden Tabelle zu sehen:

Exception Nr.	Exception	Auswirkung	Auswirkung mit MonST
2	Busfehler	Bomben	abgefangen
3	Adreßfehler	Bomben	abgefangen
4	Illegaler Befehl	Bomben	abgefangen
5	Teilung durch Null	Bomben	abgefangen
6	CHK	Bomben	abgefangen
7	TRAPV	Bomben	abgefangen
8	Privilegverletzung	Bomben	abgefangen
9	Trace	Bomben	für Single-Steppen verwendet
10	line 1010 emulator	line A-Aufruf	line A-Aufruf
11	line 1111 emulator	TOS intern	TOS intern
32	trap #0	Bomben	abgefangen
33	trap #1	GEMDOS-Aufruf	GEMDOS-Aufruf
34	trap #2	AES/VDI-Aufruf	AES/VDI-Aufruf
35-44	trap #3-trap #12	Bomben	abgefangen
45	trap #13	XBIOS-Aufruf	XBIOS-Aufruf
46	trap #14	BIOS-Aufruf	BIOS-Aufruf
47	trap #15	Bomben	abgefangen

Die genauen Ursachen dieser Exceptions werden am Ende dieses Abschnitts genauer erklärt, um aber zusammenzufassen:

- Die Exceptions 2 bis 8 sind Programmierfehler und werden von MonST abgefangen.
- Exception 9 kann unter Umständen durch einen Programmierfehler hervorgerufen werden, wird aber normalerweise von MonST für das »Single-Steppen« benutzt.
- Die Exceptions 10, 11, 33, 34, 45 und 46 werden vom Betriebssystem gebraucht und werden von

MonST nicht bearbeitet.

Die restlichen Exceptions werden alle von MonST abgefangen; sie können aber durchaus von einem Programm neu definiert und verwendet werden. Der Eintrag »Bomben« in der obigen Tabelle bedeutet, daß das Betriebssystem des ST versucht, mit dem Fehler fertig zu werden und einige Bomben (bzw. kleine Atompilze) auf dem Bildschirm malt (immer so viel, wie die Exception-Nummer, z.B. zwei Bomben bei einem Busfehler), und dann versucht, das Programm abzubrechen und ins Desktop zurückzukehren. Wenn durch den Fehler wichtige Systemvariablen zerstört wurden, kann es ohne weiteres passieren, daß das System abstürzt. Es kann auch sein, daß das System von einer Exception direkt in eine andere gerät und dadurch der ganze Bildschirm sich mit Bomben füllt.

4.3 Speicherbelegung

Die interaktiven Versionen von MonST sind zum gleichen Zeitpunkt im Speicher wie das zu debuggende Programm, d.h. MonST wird geladen, es wird nach einem Programmnamen gefragt, und dann das angegebene Programm, gegebenenfalls mit Symbolen, geladen.

MonST ist zum Zeitpunkt der Entstehung dieses Schreibens ca. 23 KByte groß, es benötigt noch zusätzlich etwa 32 KByte an Arbeitsspeicher. Dies mag etwas viel erscheinen, MonST braucht aber diesen Speicher, um immer eine Bildschirmseite für sich zu behalten.

Die drei mitgelieferten MonST-Versionen heißen:

MONST2.PRG	interaktive GEM Version
MONST2.TOS	interaktive TOS Version
AMONST2.PRG	auto-residente Version

Zuerst werden die ersten beiden Versionen beschrieben, die auto-residente Version wird danach erläutert, ist aber den ersten zwei Versionen sehr ähnlich.

4.4 MonST starten

4.4.1 Vom Desktop aus

Die beiden interaktiven Versionen von MonST sind bis auf die Dateinamen identisch. Die `.PRG`-Version sollte mit GEM-Programmen verwendet werden, die `.TOS`-Version sollte mit TOS-Programmen benutzt werden. GEM- und TOS-Programme werden vom Betriebssystem anders gestartet. Weil MonST das zu debuggende Programm startet, muß es selbst auch richtig gestartet werden.

Ein GEM-Programm hat einen Hintergrund, Platz für eine Menüleiste und braucht einen Mauszeiger, TOS-Programme brauchen aber einen blinkenden Cursor, keine Maus und einen leeren Bildschirm. Sie können ein TOS-Programm mit der GEM-Version von MonST debuggen, der Bildschirminhalt kann dann aber ziemlich chaotisch aussehen.

Es ist aber keine gute Idee, ein GEM-Programm mit der TOS-Version MonSTs zu debuggen, es können große Probleme auftauchen und unter Umständen kann das System abstürzen; dies sollte auf jeden Fall

vermieden werden.

4.4.2 Vom Editor aus

Wenn GenST gestartet wird, sucht es automatisch nach `MONST2.PRG` und lädt MonST (wenn diese Option nicht anders gewählt wurde), falls er vorhanden ist. Der Debugger ist sofort vom Editor aus verfügbar.

Wenn man Alt-M drückt oder im Programm-Menü »MonST« wählt, wird MonST, wie oben beschrieben, aufgerufen, nur wesentlich schneller, da es sich schon im Speicher befindet.

Man kann aber auch Alt-D drücken oder »Debuggen« vom Programm-Menü wählen. MonST wird so auch aufgerufen, hat aber jetzt zusätzlich das zuletzt in den Speicher assemblierte Programm bereit, ggf. auch mit Symbolen.

Der anfängliche Bildschirmmodus wird durch den GEM-Eintrag im Programm-Menü gewählt. Wenn der Eintrag abgehakt ist, wird die GEM-Bildschirm-Initialisierung vorgenommen, sonst wird der Bildschirm für ein TOS-Programm vorbereitet. Die oben erwähnten Regeln bezüglich falscher Bildschirmmodi treffen hier auch zu.

4.5 Symbolisch Debuggen

Ein Hauptmerkmal von MonST ist die Fähigkeit, die Originalsymbole des Programms beim Debuggen zu benutzen. MonST unterstützt zwei verschiedene Formate von Symbolen: das Standard-DRI-Format, welches maximal 8 Zeichen lange Symbole unterstützt, und das HiSoft- erweiterte Format, welches eine Symbollänge von maximal 22 Zeichen erlaubt. Sowohl GenST als auch LinkST können beide Formate produzieren; viele andere Hochsprachen-Compiler unterstützen auch das DRI-Format. Wir versuchen z.Zt. das HiSoft- erweiterte-Format als zweiten Standard einzuführen; zum Zeitpunkt des Schreibens dieses Handbuchs unterstützt nur der FTL-Modula-2- Compiler auch das HiSoft-Format.

4.6 Dialog- und Alertboxen

MonST verwendet viele Dialog- und Alertboxen, die den GEM- Boxen entsprechen; es sind aber keine. Der Grund dafür ist einfach: MonST muß vom Betriebssystem so unabhängig wie möglich sein; nur so ist es gewährleistet, daß MonST auch unter den schwersten Bedingungen problemlos funktioniert.

Außerdem ist auf die Maus in MonST kein Zugriff vorhanden, da es richtige GEM-Boxen und Buttons nicht unterstützt. Eine MonST-Dialogbox erkennt man daran, daß »ESC um abzubrechen« über dem linken oberen Rand steht. In der Box steht der Mitteilungstext und (normalerweise) eine leere Zeile, in der sich der Cursor befindet. Man kann zu jedem Zeitpunkt mit Esc die Dialogbox abbrechen. Die Eingabe erfolgt über die Tastatur und die Cursortasten. Die Backspace- und Del-Tasten können zum Editieren der eingetippten Zeile benutzt werden, Clr löscht die eingetippte Zeile bis zum Anfang, genauso wie die Esc-Taste in GEM-Dialogboxen. Die Eingabe wird mit der Return- Taste beendet. Falls eine Zeile einen Fehler enthält, wird Return so lange ignoriert, bis die Zeile akzeptabel ist. Dialogboxen, in die mehr als eine Zeile eingegeben wird, lassen keine Zeilensprünge durch die ↑- und ↓-Tasten zu.

Eine MonST-Alertbox ist eine Box, die eine Meldung (meistens eine Fehlermeldung) anzeigt und auf

Return oder Esc wartet.

4.7 Die Anzeige am Anfang

Wenn Sie MonST über Alt-M vom Editor oder vom Desktop aus aufgerufen haben, werden Sie zuerst in einer Dialogbox nach einem Programmnamen gefragt. Wenn Sie ein Programm laden wollen, geben Sie den Programmnamen ein (die Extension ist automatisch `.PRG`, wenn nicht anders angegeben) und drücken Sie Return. Danach werden Sie nach einer Kommandozeile gefragt, die dem Programm übergeben werden soll. Wenn das Programm keine braucht, drücken Sie einfach wieder Return, und das Programm wird geladen. Wenn Sie kein Programm debuggen wollen, z.B. nur den Speicher des Rechners bearbeiten wollen, dann drücken Sie Esc, um direkt in MonST zu gelangen.

4.8 Die MonST-Anzeige

Das Anzeigenprinzip von MonST ist dem von alten Großrechnern nachempfunden: damals hatte man immer kleine flackernde Lämpchen an der Vorderseite des Rechners beobachten können um festzustellen, welche Register gerade benutzt wurden. MonST zeigt in wesentlich leserlicherer Form an, was gerade in Ihrem ST passiert. MonST generiert eine Anzeige mit den Registern, Speicherinhalten und disassemblierten Instruktionen, flackernde Lämpchen sind also überflüssig.

Die MonST-Anzeige beim Programmstart besteht aus vier Fenstern; in niedriger Auflösung sieht die Anordnung (platzbedingt) etwas anders aus.

1. Das obere Fenster zeigt die Registerinhalte zusammen mit den Daten, die an der Adresse stehen, die gleich dem Inhalt der Register sind.

2. Fenster 2 ist das Disassemblierungs-Fenster, in dem mehrere Zeilen disassemblierten Codes zu sehen sind. Die Anfangsadresse des Fensters ist normalerweise der `PC`; Das →-Zeichen ist die momentane Position des `PC`.

3. Fenster 3 zeigt einen Teil des Speichers sowohl in Hex als auch in ASCII an.

4. Das kleine Fenster ganz unten am Bildschirm enthält keine Zahl und wird hauptsächlich für Mitteilungen verwendet.

4.9 Benutzung der Fenster

MonST hat immer ein *aktuelles Fenster*. Dieses Fenster erkennt man daran, daß der Fenstertitel weiß auf schwarz, statt umgekehrt, ist. Die Tab-Taste kann dazu verwendet werden, durch die einzelnen Fenster zu gehen, man kann aber auch Alt zusammen mit der Fensternummer drücken um ein bestimmtes Fenster zum aktuellen Fenster zu machen. Das kleine Fenster unten kann nie das aktuelle Fenster werden, da es nur zur Anzeige von Mitteilungen gedacht ist.

4.10 Eingabe von Befehlen

Der MonST-Befehlssatz basiert auf einzelnen Tasten, um ein schnelles Arbeiten zu ermöglichen. Dies ist

zu Anfang natürlich etwas gewöhnungsbedürftig, macht sich aber schon nach geringer Zeit bezahlt. Benutzer von MonST1 werden viele Befehle wiedererkennen.

Vorsicht: Manche Befehle tun fast das gleiche in MonST2, sind aber nicht unbedingt identisch.

Allgemein gilt, daß *die Alt-Taste die Fenstertaste* ist. Alle Befehle, die die Alt-Taste beinhalten, haben Auswirkungen auf Fenster. Befehle können groß- oder kleingeschrieben eingegeben werden. Alle Befehle, die unter Umständen katastrophale Auswirkungen haben können, erfordern die Control-Taste. Befehle werden sofort ausgeführt, es braucht nicht nach jedem Befehl Return zusätzlich eingegeben werden. Ungültige Befehle werden ignoriert. Die Fenster, deren Daten sich verändert haben könnten, werden auch nach jedem Befehl auf den neuesten Stand gebracht.

MonST ist ein sehr funktionsstarkes und manchmal komplex erscheinendes Programm. Wir sehen natürlich ein, daß es unwahrscheinlich ist, daß jeder Benutzer jeden einzelnen Befehl auf Anhieb gebrauchen wird. Deshalb ist dieses Kapitel in zwei weitere Teile gegliedert: einmal eine Einleitung in die Benutzung MonSTs und darauf folgend einen vollständigen Referenzteil. Es ist ohne weiteres möglich, daß Anfänger und MonST-Neulinge effektiv den Debugger benutzen, auch wenn sie nur die Bedienungsanleitung gelesen haben. Lassen Sie sich aber nicht vom Referenzteil abschrecken: er enthält sehr nützliche und wichtige Informationen.

4.11 MonST - eine Probefahrt

Es ist nicht unbedingt notwendig, jedes kleine Detail von MonST zu kennen, um den Debugger »fahren« zu können. In diesem Teil werden anhand eines kleinen Beispiels die wichtigsten Funktionen MonSTs erläutert. In unserem Beispiel geben Sie das folgende kleine Programm in den Editor ein:

```
Größe           equ     20
                section text
                moveq   #0,d0
                lea     Ende(pc),a0
                moveq   #Größe-1,d1
Schleife1       move.l  d0,(a0)+
                dbf     d1,Schleife1
                moveq   #-1,d0
                lea     Ende(pc),a0
                moveq   #Größe-1,d1
Schleife2       move.l  d0,(a0)+
                dbf     d1,Schleife2
Schluß          clr.w   -(sp)
                trap    #1
                section data
Ende            dcb.l   Größe,-1
```

Das Programm tut nichts anderes als den Bereich von 80 Byte, der mit dem Hex-Wert `$FFFFFFFF` gefüllt ist, mit Nullen zu löschen und daraufhin wieder mit `$FFFFFFFF` zu füllen. Wenn dies geschehen ist, gibt das Programm die Kontrolle wieder an das Betriebssystem zurück. Assemblieren Sie das Programm in den Speicher und starten Sie MonST mit Alt-D.

Sie können auch das Programm auf Diskette assemblieren: speichern Sie die Quelltextdatei unter dem Namen `BEISPIEL.S` ab. Dann wählen Sie als Ziel für den Assembler »Diskette« und »erweiterten Debug«; wenn er fertig ist, müßten Sie die Datei `BEISPIEL.PRG` auf Ihrer Diskette finden. Wenn Sie dann Alt-M drücken, fragt MonST nach einem Dateinamen. Geben Sie »BEISPIEL« ein, die Extension `.PRG` wird automatisch hinzugefügt. Mit dieser Prozedur sind Sie am gleichen Punkt angelangt, als ob Sie

in den Speicher assembliert und Alt-D gedrückt hätten.

Sie sehen in Fenster 2 Ihr Programm in disassemblierter Form. Drücken Sie die Tab-Taste einmal, um Fenster 3 zum aktuellen Fenster zu machen. Dann drücken Sie Alt-L und geben `a0` ein. Sie haben hiermit das Fenster 3 an das Adreßregister `A0` gebunden. Die Bedeutung dieser Funktion wird später klar. Drücken Sie Alt-2, um das Disassemblierungs-Fenster wieder zum aktuellen Fenster zu machen. Drücken Sie Ctrl-Y dreimal. Sie werden nach jeder Eingabe sehen, daß der Pfeil, der den `PC`-Stand anzeigt, einen Befehl weiterrückt.

Wenn Sie nach dem ersten Ctrl-Y ins Registerfenster schauen, werden Sie sehen, daß der erste Befehl den Wert `0` in das Datenregister `D0` gesetzt hat (man sollte nie davon ausgehen, daß ein Register einen bestimmten Wert hat; es kann sein, daß `D0` schon `0` ist, dies ist aber von GEMDOS abhängig und deshalb nicht unbedingt zutreffend). Das zweite Ctrl-Y lädt die Adresse von Ende in das Adreßregister `A0`. Das Fenster 3, welches Sie an `A0` gebunden haben, zeigt jetzt auf den Speicherbereich, der »Ende« folgt. Wie Sie sehen, ist der Bereich mit `$FFFFFFFF` gefüllt. Der dritte Befehl lädt den Schleifenzähler in das Register `D1`.

Drücken Sie Ctrl-Y erneut. Sie werden sehen, daß sich im Registerfenster der Wert von `A0` um vier erhöht hat, ebenso die Startadresse von Fenster 3. Ein Fenster an ein Register binden heißt also, daß bei jedes Exception die Startadresse des Fensters auf die Adresse des Registers gesetzt wird.

Jetzt wollen wir aber nicht unbedingt jedes Mal die Schleife durchgehen. Mit Ctrl-A wird ein Breakpoint nach dem Befehl, der am `PC` ist, gesetzt, und das Programm dann laufengelassen.

Das Setzen von Breakpoints ist eine Funktion von MonST. Wenn Sie ein Breakpoint an eine Adresse setzen, wird das Programm angehalten und MonST aufgerufen, wenn der `PC` diese Adresse erreicht. Mit Ctrl-A umgehen wir das lästige Durchlaufen der Schleife, indem wir ein Breakpoint nach dem `DBF`-Befehl setzen, d.h. nachdem die Schleife zu Ende ist. Drücken Sie jetzt also Ctrl-A.

Wenn die Schleife zu Ende ist, sind Sie dazu bereit, die zweite Schleife, die wieder `$FFFFFFFF` in den Speicherbereich füllt, auszuführen. Drücken Sie jetzt Alt-3, um Fenster 3 zum aktuellen Fenster zu machen. Drücken Sie wieder Alt-L und geben Sie als Binde-Register diesmal m2 ein. Sie können nämlich ein Fenster an ein anderes Fenster binden. Die Startadressen der einzelnen Fenster sind in Speichern enthalten, die wie Register behandelt werden können.

Die Startadresse vom Fenster 2, dem Disassemblierungs-Fenster, ist im Speicher M2 enthalten. Jetzt wird nach jedem Einzelschritt (den wir mit Ctrl-Y machen) das Fenster 3 auf dem nächsten Befehl stehen; der Unterschied ist aber, daß im Fenster 3 die Befehle als Hex-Zahlen ausgegeben werden und nicht disassembliert sind. Mit Alt-2 wird wieder Fenster 2 zum aktuellen Fenster.

Da wir die zweite Schleife auch nicht Schritt für Schritt durchlaufen wollen, benutzen wir einen Breakpoint, um zum `Schluß` zu kommen. Drücken Sie Alt-B und geben Sie Schluß, den Namen des Symbols, ein. Sie werden rechts neben dem Befehl, der an der Adresse von `Schluß` steht, `[ 1]` sehen. Dies bedeutet, daß ein Breakpoint mit Wert `1` am `Schluß` gesetzt wurde. Dann benutzen wir Ctrl-R um das Programm laufen zu lassen. Wenn die Schleife fertig ist, hält das Programm an, und zwar am `Schluß`, wie wir es angegeben haben.

Wenn Sie zweimal Ctrl-Y drücken, ist das Programm fertig. Wenn Ihr Programm in den Speicher

assembliert wurde, springt MonST direkt in den Editor zurück. Wenn nicht, meldet MonST »Programmende«. Sie könnten jetzt ein weiteres Programm mit Ctrl-L laden. Wenn nicht, kommen Sie mit Ctrl-C zurück zum Editor.

Sie haben soeben MonST »fahren« gelernt. Sie können jetzt mit Einzelschritten, einfachen Breakpoints und Fenstern umgehen. Wenn Sie den ersten Teil des Kapitels verstanden haben, lesen Sie im Referenzteil weiter. Er beschreibt ausführlich alle Funktionen, die MonST zu bieten hat.

4.12 MonST - Referenzteil

4.12.1 Numerische Ausdrücke

MonST kann numerische Ausdrücke auswerten, ähnlich wie GenST. Der Hauptunterschied zu GenST liegt darin, daß MonSTs normale *Zahlenbasis in Hex-Code* ist, dezimalen Zahlen muß ein »\«-Zeichen (kein »#« wie in Version 1!) vorangesetzt werden. Es gibt keine Ausdruckstypen (absolut oder relativ), das »*«-Zeichen wird nur für die Multiplikation verwendet.

Symbole können verwendet werden, die normalerweise »case-sensitiv« sind, d.h. daß ein Unterschied zwischen kleinen und großen Buchstaben gemacht wird. Sie haben eine Signifikanz von 8 bis 22 Zeichen (je nach Symbolformat). Diese Standardeinstellungen können mit »Voreinstellungen« verändert werden.

Register werden mit ihren Namen bezeichnet, z.B. `A3` oder `D7`. Dies sind aber keine Hex-Zahlen. Um die Hex-Zahl `A0` anzugeben, muß entweder ein »$«-Zeichen der Zahl vorangesetzt werden oder die Zahl Null.

`A7` bezeichnet immer den User-Stack-Pointer. Es gibt reservierte Symbole, die nicht »case-sensitiv« sind: `TEXT`, `DATA`, `BSS`, `END`, `SP`, `SR` und `SSP`. `END` bezeichnet das erste Byte nach dem `BSS`-Segment, `SP` ist entweder der Supervisor oder der User-Stack-Pointer, abhängig vom Wert des Status-Registers.

Es gibt 10 »Speicher«, M0 bis M9, ähnlich wie bei Taschenrechnern. Diese Speicher können, wie Register, Werte zugewiesen bekommen. Die Speicher 2 bis 5 enthalten immer die Startadresse der Fenster 2 bis 5; wenn Sie einem dieser Speicher einen neuen Wert zuweisen, verändert sich auch die Anfangsadresse dieses Fensters.

Vorsicht: Weisen Sie nie M4 einen neuen Wert zu, wenn Fenster 4 als »Source«-Fenster benutzt wird.

Ausdrücke können auch Indirektionen beinhalten: die Zeichen »{« und »}« werden dafür benutzt. Indirektion kann auf einer Byte-, Wort-, oder Langwortbasis angegeben werden, wenn das »}« von einem Punkt und der gewünschten Größe gefolgt wird; der Normalwert ist Langwort. Falls der Pointer ungültig ist, entweder weil der Speicher unlesbar oder ungerade (wenn Wort- oder Langwortgröße angegeben ist) ist, ist der gesamte Ausdruck ungültig.

Z.B. wird der Ausdruck {Daten_Anfang+10}.w das Wort liefern, welches an der Adresse `Daten_Anfang+10` steht. Indirektion kann auf gleiche Art wie normale Klammern verschachtelt werden.

4.12.2 Fenstertypen

Es gibt vier verschiedene Fenstertypen. Die erlaubten Fenstertypen sind wie folgt:

Fensternummer	Erlaubte Typen
1	Registerfenster
2	Disassemblierungsfenster
3	Speicher
4	Disassemblierung, Speicher oder »Source«
5	Speicher

Registerfenster

In einem Registerfenster werden die Inhalte der Datenregister in Hex angezeigt, zusätzlich der ASCII-Wert des untersten Bytes und die Anzeige (in Hex) der ersten acht Byte, auf die der Registerinhalt zeigt. Die Adreßregister werden auch in Hex angezeigt, zusammen mit den ersten 12 Byte, auf die das Register zeigt. Wie in allen Hex-Anzeigen in MonST werden Werte in nicht lesbaren Speicherstellen mit `**` ersetzt. Das Statusregister wird sowohl in Hex als auch als »Flags« dargestellt.

Zusätzlich wird entweder `U` oder `S` angezeigt, je nachdem, ob der Rechner sich im Supervisor- oder Usermodus befindet. A7' ist der Supervisor-Stack-Pointer und wird wie ein normales Adreßregister angezeigt.

Der Wert des `PC` ist zusammen mit der Instruktion, die am `PC` ist, in der untersten Zeile des Registerfensters angezeigt. Wenn die Instruktion eine oder zwei effektive Adressen hat, werden diese zusammen mit dem Speicher, der an dieser(n) Adresse(n) ist, angezeigt.

```
TST.W $12A(A3)  ;00001FAE 0F01
```

Diese Zeile bedeutet, daß der Wert von `A3` mit `$12A` addiert die Adresse `$1FAE` ergibt. An dieser Adresse ist der Wert `$0F01`. Ein komplexeres Beispiel ist

```
MOVE.W $12A(A3),-(SP)  ;00001FAE 0F01 →0002AC08 FFFF
```

Der Ursprungsadreßmodus ist der, wie im ersten Beispiel, die Zieladresse ist dagegen `$2AC08`, die z.Zt. `$FFFF` enthält. Beachten Sie, daß die Anzeige immer so groß ist, wie gerade nötig. `MOVEM`-Daten werden als Quad-Worte (64 Bit) angezeigt. Wenn Prädekrement im Adressierungsmodus verwendet wird, wird dies bei der Adreßberechnung mit in Betracht gezogen.

In niedriger Auflösung werden keine Hex-Daten für die Datenregister und nur vier Byte als Daten bei Adreßregistern angezeigt. Es kann auch in niedriger Auflösung vorkommen, daß die Bildschirmbreite nicht groß genug ist, um eine komplexe Zeile (wie das zweite Beispiel) anzuzeigen.

Disassemblierungs-Fenster

In einem Disassemblierungs-Fenster wird Speicherinhalt in disassemblierter Form angezeigt. Ganz links auf einer Zeile wird die Adresse angezeigt, dann ggf. das Symbol an dieser Adresse, daraufhin der Befehl an dieser Adresse. Falls an dieser Adresse ein »Breakpoint« gesetzt ist, wird dessen Typ in eckigen Klammern, »[«und »]«, nach dem Befehl angezeigt. Bei Stop- Breakpoints wird angezeigt, wie oft dieser Befehl noch ausgeführt werden muß, bis der Breakpoint aktiv wird und das Programm stoppt. Bei einem Breakpoint mit einer Bedingung enthalten die eckigen Klammern ein Fragezeichen, gefolgt von der

Bedingung. Bei Zähler-Breakpoints enthalten die Klammern ein Gleichheitszeichen und den Zählerwert. Bei permanenten Breakpoints ist ein »*« in den eckigen Klammern.

Das Format der disassemblierten Instruktionen ist im Motorola- Standard, wie GenST ihn akzeptiert. Alle Text-Daten sind großgeschrieben, bis auf die Symbole, die kleine Buchstaben enthalten. Alle numerischen Daten sind in Hex bis auf die TRAP- Nummern. Vorangehende Nullen und der Hex-Bezeichner »$« werden bei Zahlen kleiner als 10 nicht angezeigt. Wenn erforderlich, werden numerische Daten mit Vorzeichen angezeigt. Die einzige Abweichung vom Motorola-Standard ist die Darstellung der Registerliste des `MOVEM` Befehls:

```
MOVEM.L D0-D3/A0-A2,-(SP)
```

wird, um Platz zu sparen, als

```
MOVEM.L D0-3/A0-2,-(SP)
```

angezeigt.

In niedriger Auflösung ersetzen Symbole die Adresse und werden nur bis zu einer maximalen Länge von acht Zeichen angezeigt.

Speicher-Fenster

In einem Speicher-Fenster wird Speicher zeilenweise im Format Hex-Adresse, Hex-Daten und ASCII-Daten angezeigt. Unlesbarer Speicher wird als »**« dargestellt. Die Anzahl der angezeigten Bytes pro Zeile hängt von der Breite des Fensters ab und kann maximal 16 Byte betragen.

Source-Fenster

Sie können im »Source«-Fenster ASCII-Dateien, wie z.B. Ihren Programm-Quelltext, ähnlich wie in einem Text-Editor darstellen. Der normale Tab-Abstand ist 8, kann aber mit dem Editieren-Fenster-Befehl (Alt-E) auf 4 umgeschaltet werden.

4.12.3 Fenster-Befehle

Die Alt-Taste ist die sog. Fenster-Taste, d.h. alle Fenster-Befehle beinhalten die Alt-Taste. Jeder dieser Befehle bezieht sich auf das aktuelle Fenster, also das Fenster, dessen Titel invers dargestellt ist. Das aktuelle Fenster kann gewählt werden, indem man entweder die Tab-Taste so oft drückt, bis das gewünschte Fenster zum aktuellen Fenster wird oder Alt zusammen mit der Nummer des gewünschten Fensters drückt.

Die meisten Fenster-Befehle funktionieren in allen Fenstern unabhängig davon, ob sie im »Zoom«-Modus sind oder nicht. Wenn ein Befehl keinen Sinn ergibt, wird er ignoriert.

Alt-A Adresse setzen

Dieser Befehl setzt die Anfangs-Adresse eines Fensters

Alt-B Breakpoint setzen

Mit diesem Befehl werden Breakpoints gesetzt; Breakpoints werden später genauer beschrieben.

Alt-E Editieren

In einem Speicherfenster kann man mit diesem Befehl im Speicher in Hex und ASCII editieren. Hex-Zahlen können mit den Tasten 0-9 und A-F eingegeben werden. Die Cursor-Tasten können dazu benutzt werden, den Cursor im Fenster zu bewegen. Mit der Tab-Taste können Sie zwischen dem Hex- und dem ASCII-Editiermodus hin- und herschalten. Der ASCII-Modus schreibt den ASCII-Wert des Tastendrucks in den Speicher. Um den Editiermodus zu verlassen, drücken Sie die Esc-Taste.

Im Registerfenster bewirkt Alt-E das gleiche wie Alt-R, nämlich Register setzen.

Im »Source«-Fenster kann mit Alt-E der Tab-Abstand zwischen 4 und 8 gewählt werden.

Alt-F Fontgröße

Dieser Befehl verändert die Fontgröße im aktuellen Fenster. In hoher Auflösung wird zwischen 16 und 8 Pixel großen Fonts gewechselt; in Farbe wird zwischen 8 und 6 Pixel großen Fonts gewechselt. Wenn die Fontgröße im Register-Fenster verändert wird, werden die Größen der anderen Fenster neu berechnet, um sie so groß wie möglich zu machen.

Alt-L Fenster binden

Mit diesem Befehl kann ein Fenster an ein Register oder an ein anderes Fenster gebunden werden. Nach einer Exception werden die Startadressen aller Fenster neu berechnet und ggf. entsprechend verändert. Um eine Bindung zu löschen, geben Sie in die Dialogbox nichts ein. Fenster 2 ist standardmäßig an den `PC` gebunden. Um ein Fenster an ein anderes zu binden, müssen Sie den entsprechenden Speicher angeben, z.B. M2 für Fenster 2.

Alt-O Auswerten

Eine Dialogbox erscheint, in der Sie einen Ausdruck eingeben. Dieser wird ausgewertet und in Hex, Dezimal und ggf. als Symbol ausgegeben.

Alt-P Drucker Dump

Der Inhalt des aktuellen Fensters wird auf dem Drucker ausgegeben. Dieser Befehl kann mit Esc abgebrochen werden.

Alt-R Register setzen

Mit diesem Befehl kann jedes Register gesetzt werden. Das Register muß angegeben werden, gefolgt von einem Gleichheitszeichen und dem Ausdruck des neuen Registerwerts. Zum Beispiel: A3=A2+4 setzt das Register `A3` auf den Wert vom Register `A2` mit vier addiert.

Sie können z.B. im »Zoom«-Modus diesen Befehl dazu verwenden, einem anderen Fenster eine neue Startadresse zu geben. Wenn Sie den »Zoom«-Modus verlassen, zeigt das Fenster auf die angegebene Adresse.

Alt-S Fenster teilen

Dieser Befehl teilt Fenster 2 in die Fenster 2 und 4 oder das Fenster 3 in die Fenster 3 und 5. Jedes neue Fenster ist vom Ursprungsfenster unabhängig. Wenn Sie Alt-S ein zweites Mal drücken, werden die zwei Fenster wieder zusammengefügt.

Dieser Befehl wird in niedriger Auflösung ignoriert.

** Alt-T Typ verändern

Dieser Befehl hat nur Wirkung in Fenster 4, welches durch die Teilung von Fenster 2 entsteht, oder wenn Sie eine ASCII-Datei laden. Der Fenstertyp wird zwischen Disassemblierung, Speicheranzeige oder »Source« geschaltet.

Alt-Z »Zoom«

Das aktuelle Fenster wird mit diesem Befehl auf die volle Bildschirmgröße gebracht. Mit Esc oder einem erneuten Alt-Z wird das Fenster wieder auf Normalgröße gebracht. Alt-Befehle sind im »Zoom«-Modus weiterhin benutzbar.

Cursor-Tasten

Die Cursor-Tasten werden in Verbindung mit dem aktuellen Fenster benutzt. Ihre Wirkung hängt vom Fenstertyp ab.

- In einem *Speicher-Fenster* verändern alle vier Cursor-Tasten die Startadresse des Fensters. Mit Shift-↑ und Shift-↓ können Sie seitenweise hoch und runter »blättern«.
- In einem *Disassemblierungs-Fenster* verändern ↑- und ↓-Tasten die Startadresse des Fensters um jeweils eine Instruktion. ← und → verändern die Startadresse jeweils um ein Byte. Die Shift-Tasten können wiederum für das seitenweise Blättern benutzt werden.
- Im *»Source«-Fenster* verändern ↑ und ↓ die Startadresse des Fensters um eine Zeile, die Shift-Tasten verändern die Startadresse um eine Seite.

4.13 Bildschirm-Umschaltung

MonST benutzt seinen eigenen Bildschirmspeicher und seinen eigenen Bildschirmtreiber, um jeden Konflikt mit dem zu debuggenden Programm zu vermeiden. Um ein »Flackern« des Bildschirms beim »Single-Steppen« zu verhindern, wird von der Bildschirm-Anzeige auf die des Programms erst nach 20 msec umgeschaltet. Es ist auch möglich, den Debugger in einer anderen Auflösung wie das zu debuggende Programm zu betreiben; dies geht aber nur mit einem Farbmonitor.

V Anderer Bildschirm

Dieser Befehl schaltet bis zum nächsten Tastendruck auf den Bildschirm des zu debuggenden Programms.

Ctrl-O Andere Auflösung

Dieser Befehl schaltet MonSTs Anzeige zwischen niedriger und mittlerer Auflösung hin und her. Alle Fenster werden neu initialisiert und auf die Standardgrößen eingestellt. Dieser Befehl hat keine Auswirkung auf das zu debuggende Programm.

Dieser Befehl wird bei einem monochromen Monitor ignoriert. Dadurch, daß MonST seinen eigenen Bildschirmspeicher hat, ist es möglich, den Bildschirmspeicher Ihres Programms anzusehen; dies ist bei Grafikprogramm ideal.

Vorsicht: Wenn Ihr Programm die Bildschirm-Auflösung, -Position oder die Farbpalette verändert, ob mit dem XBIOS oder direkt per Hardware, ist es wichtig, daß Sie die Bildschirm-Umschaltung bei den Voreinstellungen, während Sie diesen Code ausführen, abschalten. Wenn Sie dies nicht tun, bemerkt MonST die Veränderungen nicht.

Wenn Sie einen Diskettenzugriff durchführen, schaltet der Bildschirm auf den Bildschirm Ihres Programms um. Dies geschieht, um eine möglicherweise erscheinende GEM-Alarmbox (z.B. bei einem Diskettenfehler) sofort sichtbar werden zu lassen, Sie können deshalb sofort darauf reagieren.

4.14 Programmunterbrechung

4.14.1 Shift-Alt-Help

Ein laufendes Programm kann mit Shift-Alt-Help unterbrochen werden: durch diese Tastenkombination wird eine Trace-Exception ausgelöst. Wenn Sie in MonST zurückgelangen, kann es sein, daß sich der `PC` im Programm befindet oder die Unterbrechung im ROM stattfindet (was bei Programmen der Fall ist, die ausgiebigen Gebrauch vom ROM machen) oder im Line-F- oder dem TRAP-Handler. Wenn Sie sich nicht im Ihrem Programm befinden, raten wir Ihnen, ein Breakpoint an einer entsprechenden Stelle in Ihr Programm zu setzen und mit Ctrl-R das System weiterlaufen zu lassen. Alt-Help ohne Shift löst einen Ausdruck des Bildschirms von Ihrem Programm aus. Wenn Sie versehentlich das Shift vergessen haben, können Sie den Druck mit nochmaligem Alt-Help abbrechen.

Es kann sein, daß diese Tastenkombination ignoriert wird, nochmaliges Drücken der Tasten müßte MonST eingreifen lassen. Diese Kombination hat keine Wirkung, wenn sie innerhalb von MonST gedrückt wird.

Vorsicht: Ein Programm sollte nie mit Ctrl-C abgebrochen werden, nachdem es innerhalb des ROMs unterbrochen wurde. Dies kann sehr leicht zu einem Absturz führen.

4.14.2 Ctrl-S

Eine Anweisung kann durch Ctrl-S abgebrochen werden. Dieser Vorgang ist gleichbedeutend damit, daß der `PC` x Wörter nach vorne gesetzt wird. Dieser Befehl ist dann angebracht, wenn eine Anweisung seltsame Ergebnisse erzeugen könnte.

4.14.3 Reset

Während Sie sich im Debugger befinden, können Sie mit Ctrl-Alt-. (».« im Num-Block) einen Reset durchführen. Es wird keine Warnung generiert.

4.14.4 Breakpoints

Breakpoints erlauben es Ihnen, Ihr Programm an von Ihnen bestimmten Punkten zu unterbrechen. MonST erlaubt maximal acht gleichzeitig gesetzte Breakpoints, von denen jeder einer der fünf Breakpoint-Typen sein kann. Wenn ein Breakpoint erreicht wird, trifft MonST die Entscheidung, ob das Programm angehalten wird oder weiter abläuft. Diese Entscheidung hängt von den Typen des Breakpoints ab.

Einfache Breakpoints

MonST hält das Programm bei Erreichen der Breakpoints an und löscht dann den Breakpoint.

Stop Breakpoints

Diese Art unterbricht das Programm an der Stelle, an der ein bestimmter Befehl an der Adresse des Breakpoints zum n-ten Male ausgeführt wurde. Ein einfacher Breakpoint ist eigentlich ein Stop-Breakpoint mit dem Wert 1.

Zähler-Breakpoints

Diese Art ist nur ein Zähler. Jedesmal, wenn der Befehl an der Adresse des Breakpoints ausgeführt wird, erhöht sich der Zähler um eins, und das Programm läuft weiter.

Permanente Breakpoints

Dieser Typ ist den einfachen Breakpoints ähnlich, bis auf den Unterschied, daß sie nie gelöscht werden. Jedesmal, wenn dieser Breakpoint erreicht wird, übernimmt MonST die Kontrolle.

Bedingte Breakpoints

Diese Art von Breakpoint erlaubt es Ihnen, Ihr Programm nur dann anzuhalten, wenn eine bestimmte Bedingung erfüllt ist. Jeder bedingte Breakpoint hat einen Ausdruck; dieser wird jedesmal, wenn der Breakpoint erreicht wird, ausgewertet. Wenn der Ausdruck wahr (d.h. ungleich Null) ist, wird der Programmablauf unterbrochen.

Alt-B Breakpoint setzen

Dies ist ein Fenster-Befehl; er erlaubt das Setzen oder Löschen von Breakpoints zu jeder Zeit. Die eingegebene Zeile muß je nach Breakpoint-Typ eines der folgenden Formate haben:

- <Adresse>

  setzt einen einfachen Breakpoint.

- <Adresse>,<Ausdruck>

  setzt einen Stop-Breakpoint an der angegebenen Adresse, der das Programm stoppt, nachdem <Ausdruck> eine bestimmte Anzahl oft ausgeführt wurde.

- <Adresse>,=

setzt einen Zähler Breakpoint. Der Anfangswert des Zählers ist Null

- <Adresse>,*

  setzt einen permanenten Breakpoint.

- <Adresse>,?<Ausdruck>

  setzt einen bedingten Breakpoint, der vom <Ausdruck> abhängig ist.

- <Adresse>,-

  löscht jede Art von Breakpoint an der angegebenen <Adresse>. Breakpoints können nicht an ungeraden und unlesbaren Adressen, und im ROM gesetzt werden. ROM-Breakpoints können emuliert werden, indem man den Run-Until-Befehl benutzt.

  Jedesmal, wenn ein Breakpoint erreicht wird, egal ob das Programm angehalten wird oder nicht, wird der Prozessorstatus im History-Buffer, der später beschrieben wird, behalten.

Help Breakpoint- und Segmentanzeige

Dieser Befehl zeigt die Adressen der `TEXT`, `DATA` und `BSS`-Segmente und deren Längen an, zusammen mit sämtlichen gesetzten Breakpoints. Alt-Befehle können hier auch benutzt werden.

Ctrl-B Breakpoint setzen

Dieser Befehl ist hauptsächlich aus Kompatibilitätsgründen zu MonST1 noch vorhanden. Hiermit wird ein einfacher Breakpoint an der Startadresse des aktuellen Fensters gesetzt. Falls schon ein Breakpoint an dieser Adresse vorhanden ist, egal welcher Art, wird er gelöscht.

U Go Until

Dieser Befehl fragt in einer Dialogbox nach einer Adresse, an der dann ein einfacher Breakpoint gesetzt wird. Das Programm läuft danach sofort weiter ab.

Ctrl-K Kill Breakpoints

Alle gesetzten Breakpoints werden gelöscht.

Ctrl-A Breakpoint setzen und ausführen

Mit diesem Befehl wird ein einfacher Breakpoint nach dem Befehl am `PC` gesetzt, das Programm läuft weiter. Dieser Befehl ist hauptsächlich für DBF-Schleifen gedacht, die man nicht immer wieder durcharbeiten will, sondern deren Ergebnis man gleich sehen will.

Ctrl-D BDOS-Breakpoint

Hiermit wird ein Breakpoint auf den nächsten BDOS-Aufruf gesetzt, dessen Funktionsnummer Sie angeben. Wenn Sie einen bestehenden BDOS-Breakpoint löschen wollen, geben Sie eine leere Zeile ein.

BDOS-Breakpoints müssen gelöscht werden, um deaktiviert zu werden.

History

MonST hat einen sog. History-Speicher, in dem der Prozessorstatus nach jeder Exception behalten wird. Die häufigste Ursache eines Eintrags in den History-Speicher ist das »Single-Steppen«. Aber auch bei jedem Breakpoint und bei bestimmten Arten des Run-Befehls wird ein neuer Eintrag in den Speicher gemacht.

Der Speicher enthält Platz für fünf Einträge. Wenn er voll ist, wird jeweils der älteste Eintrag gelöscht, damit ein neuer aufgenommen werden kann.

H History-Speicher-Anzeige

Ein Fenster wird geöffnet, in dem die maximal fünf Einträge im History-Speicher angezeigt werden. Sowohl alle Registerwerte werden angezeigt, als auch der nächste Befehl, der ausgeführt wird.

Wenn auf einen Befehl im History-Speicher ein Breakpoint gesetzt ist, zeigt der Wert in den eckigen Klammern den Wert des Breakpoints zur *Zeit der Anzeige und nicht zum Zeitpunkt des Eintrags* in den Speicher.

4.15 MonST verlassen

Ctrl-C Abbrechen

Dieser Befehl verursacht einen Programmende-Trap für den aktuellen GEMDOS »Task«. Wenn ein Programm mit MonST debuggt wird, wird zuerst das Programm mit Ctrl-C abgebrochen, die Meldung »Programmende« erscheint unten im Mitteilungsfenster; durch nochmaliges Ctrl-C wird MonST verlassen. Sie können aber auch ein anderes Programm laden, ohne MonST verlassen zu müssen.

Wenn MonST vom Editor aus aufgerufen wurde, kehren Sie sofort nach Beendigung des Programmes in den Editor zurück, ohne Ctrl-C für MonST drücken zu müssen.

Vorsicht: Es kann durchaus zum Absturz kommen, wenn Sie frühzeitig ein GEM-Programm abbrechen, ohne daß z.B. die »Virtual Workstation« geschlossen wurde.

4.16 Laden und Abspeichern

Ctrl-L Ausführbares Programm laden

Mit diesem Befehl wird ein Programm zum Debuggen geladen. Sie werden nach dem Programmnamen (wenn nicht anders angegeben, wird .PRG als Extension angenommen) und dann nach der Kommandozeile gefragt, die dem Programm übergeben werden soll. Wenn MonST schon ein Programm geladen hat, muß dieses Programm erst beendet werden, bevor ein neues geladen werden kann.

Die Datei, die geladen werden soll, muß ein von GEMDOS ausführbares Programm sein. Wenn dies nicht der Fall ist, wird ein TOS-Fehler 66 gemeldet. Weitere Versuche, ein Programm zu laden, werden

normalerweise scheitern, da GEMDOS den für das Programm reservierten Speicher nicht zurückgibt. Wenn dies geschieht, müssen Sie MonST verlassen und neu starten; laden Sie die Datei dann als Binärdatei.

B Binärdatei laden

Es wird nach einem Dateinamen und nach einer (nicht notwendig anzugebenden) Ladeadresse gefragt. Diese Datei wird in den Speicher geladen. Wenn keine Ladeadresse angegeben wird, wird die Datei dort geladen, wo GEMDOS genügend Speicher freigibt. M0 enthält die Startadresse, M1 die Adresse des Dateiendes im Speicher.

S Binärdatei sichern

Dieser Befehl fragt nach einem Dateinamen, unter dem die Datei gesichert werden soll, nach einer Startadresse und einer Schlußadresse. Um eine zuvor geladene Binärdatei zu sichern, können Sie als Start- und Schlußadresse M0 und M1 angeben, vorausgesetzt, die Werte haben sich nicht verändert.

A ASCII-Datei laden

Mit diesem Befehl können Sie eine ASCII-Datei laden, normalerweise eine Quelltextdatei, um sie in MonST anzusehen. Fenster 4 wird geöffnet, falls es noch nicht vorhanden ist, und als »Source«-Fenster eingestellt. Der Speicher für diese Datei wird von GEMDOS angefordert. Es ist deshalb ratsam, die ASCII-Datei vor Ihrem Programm zu laden, damit garantiert genügend Speicher dafür vorhanden ist. In niedriger Auflösung wird Fenster 4 nicht geöffnet. Sie können aber die Datei laden, mit Ctrl-O die Auflösung ändern und mit Alt-S und zweimaligem Alt-T den Text dann ansehen.

Wenn eine ASCII-Datei nach einem Programm geladen wird, gehört der Speicher, in dem die Datei sich befindet, dem Programm und *nicht MonST*. Wenn das Programm zu Ende ist, wird der Speicher, in dem sich der Text befindet, von GEMDOS freigegeben. Die auto-residente Version von MonST kann diese Unterscheidung nicht machen, seien Sie also vorsichtig, wenn Sie eine ASCII-Datei in den auto-residenten MonST laden.

4.17 Programme ausführen

Ctrl-R Zurück zum Programm / Laufenlassen

Hiermit wird das Programm mit voller Geschwindigkeit laufengelassen; es ist die übliche Methode dafür, das Programm nach einem Breakpoint weiterlaufen zu lassen.

Ctrl-Y / Ctrl-Z »Single-Step«

Hiermit wird der Maschinensprachebefehl am `PC` ausgeführt. Ein Trap-, Line-A- oder Line-F-Opcode wird als einzelner Befehl angesehen und als solcher ausgeführt. Dies kann durch »Voreinstellungen« verändert werden.

Ctrl-T Befehl interpretieren

Dieser Befehl interpretiert den Opcode am PC. Er ist dem Ctrl-Y / Ctrl-Z ähnlich, verfolgt aber keine JSR, BSR, TRAP, Line-A- oder Line-F-Befehle. Er erspart Ihnen das Durchlaufen solcher Routinen und kann mit Befehlen im ROM oder im RAM benutzt werden.

R Run

Dies ist ein allgemeiner Befehl und kann auf verschiedene Arten ausgeführt werden.

Run G Go

Dieser Befehl ist mit Ctrl-R identisch und läßt das Programm mit voller Geschwindigkeit laufen.

Run S Langsam (Slowly)

Das Programm wird langsam ausgeführt, jeder Befehl wird im History-Speicher behalten.

Run I Instruktionen

Dieser Befehl ist dem »Run s« ähnlich, nimmt aber einen numerischen Parameter: die Anzahl der Instruktionen, die ausgeführt werden, bevor MonST wieder eingreift.

Run U Bis (Until)

Sie werden nach einem Ausdruck gefragt, der nach jedem Opcode ausgewertet wird. Sobald dieser Ausdruck »wahr« ist, wird das Programm gestoppt.

Sie können z.B. den folgenden Ausdruck verwenden, wenn Sie Ihr Programm im ROM mit Shift-Alt-Help angehalten haben und erst wieder Kontrolle übernehmen wollen, wenn Ihr Programm-Code wieder ausgeführt wird: (PC>TEXT)&(PC<FC0000)

Das Programm wird so lange laufen, bis der PC an einer Adresse ist, die sich nicht im ROM (PC<FC0000) und in Ihrem Programmbereich befindet (PC>TEXT). Sie könnten sicherheitshalber (PC>TEXT)&(PC<DATA)&(PC<FC0000) angeben, dies ist aber meist unnötig und braucht für die Auswertung nach jeder Instruktion länger.

Bei allen Run-Befehlen werden Sie nach »Beobachten J/N?« gefragt. Wenn Sie »Ja« wählen, können Sie von MonST aus den Programmablauf mit allen Registerveränderungen beobachten. Dieser Vorgang läuft ziemlich langsam ab und kann mit dem Drücken beider Shift-Tasten abgebrochen werden. Sie können auch Shift-Alt-Help dazu verwenden, den Beobachtungsmodus anzuhalten. Diese Tastenkombination kann aber aus system-spezifischen Gründen nicht immer den Beobachtungsmodus anhalten, die beiden Shift-Tasten sind in diesem Fall zuverlässiger. Wenn Sie »Nein« wählen, wird das Programm mit seinem eigenen Bildschirm laufengelassen. Den Programmablauf können Sie mit Shift-Alt-Help unterbrechen.

Wenn Sie den Ablauf »beobachten«, ist es ratsam, die Bildschirm-Umschaltung in den Voreinstellungen abzuschalten. Sonst wird bei jedem Befehl der Bildschirm umgeschaltet und erzeugt dadurch ein sehr augenfeindliches Flackern, was besonders bei Farbmonitoren unangenehm ist.

Bei allen Arten von »Run« (außer Go) wird nach jedem ausgeführten Befehl ein Eintrag in den History-

Speicher gemacht. Traps werden als einzelne Befehle behandelt. Sie können dies mit Voreinstellungen verändern, allerdings sollten Sie die Warnungen, die bei der Beschreibung von »Traps verfolgen« gegeben werden, beachten.

Wenn ein Programm in einem der oben erwähnten Modi (außer Go) läuft, werden Sie einige Pixel in der linken oberen Ecke Ihres Bildschirms sehen. Dieses kleine »Lämpchen« zeigt, daß MonST noch aktiv arbeitet. Man könnte sonst annehmen, daß die Maschine hängengeblieben ist, weil sich die Ausführungsgeschwindigkeit stark verlangsamt.

4.18 Speicher durchsuchen

G Speicher durchsuchen

Nach diesem Befehl erscheint »Suchen nach B/W/L/T/I?«, wobei die Kürzel Byte, Wort, Langwort, Text und Instruktion bedeuten. Wenn Sie B, W oder L wählen, werden Sie nach den Zahlensequenzen gefragt, nach denen Sie suchen wollen. MonST findet auch Zahlen an ungeraden Adressen.

Wenn Sie T wählen, geben Sie den Text ein, nach dem gesucht werden soll, er wird dann »case-sensitiv« behandelt.

Geben Sie I ein, so können Sie nach einem ganzen oder nur einem Teil eines Maschinensprache-Befehls suchen. Wenn Sie z.B. nach »$14(A« suchen, werden Sie eine Instruktion wie `MOVE.L D2,$14(A0)` finden. Anders als bei MonST1, wird zwischen Groß-und Kleinschreibung unterschieden. Sie sollten auch das Ausgabeformat des Disassemblers beachten: so sollten Sie z.B. für Zahlen Hex-Code verwenden, sich auf `A7` anstatt `SP` beziehen usw. Wenn Sie die Parameter eingegeben haben, wird die Kontrolle an den N-Befehl übergeben.

N Nächstes Vorkommnis suchen

Dieser Befehl wird nach dem G-Befehl benutzt; er sucht nach dem nächsten Vorkommnis der angegebenen Suchparameter. Wenn Sie nach Byte-, Wort- oder Langwort-Daten suchen, werden Sie immer ein Vorkommnis, nämlich im MonST-Speicher, finden. Wenn Sie nach Text suchen, kann es auch sein, daß der Text sich noch im Tastaturpuffer befindet. Mit diesen Suchparametern wird nach 64 KByte durchsuchten Speichers die Esc-Taste abgefragt. Bei der Suche nach Maschinensprachebefehlen wird immer nach 2 Byte auf die Esc-Taste hin geprüft.

Gesucht wird im Bereich von `0` bis zum Ende des Speichers, dann von `$FA0000` bis `$FEFFFF`, dem Modul- und System-ROM-Bereich, und dann wieder von `0`.

Die Suche beginnt kurz nach der Anfangsadresse des aktuellen Fensters (wenn es nicht das Registerfenster ist). Wenn die angegebenen Daten gefunden wurden, werden die Startadressen von manchen Fenstern auf die Adresse der gefundenen Daten gesetzt. Wenn innerhalb von Fenster 2 oder 3 gesucht wurde, werden die Startadressen von 2 und 3 auf die gefundenen Daten gesetzt. Wenn innerhalb der Fenster 4 oder 5 gesucht wurde, werden die Startadressen von 4 und 5 auf die gefundenen Daten gesetzt, außer dann, wenn Fenster 4 als »Source«-Fenster benutzt wird.

Im »Source«-Fenster suchen

Wenn Sie den G-Befehl im »Source«-Fenster verwenden, kann nur nach Text gesucht werden. Wenn er gefunden worden ist, wird die Zeile, die den Text enthält, angezeigt. Wenn der Text nicht gefunden wurde, wird das Fenster nicht verändert. Daß die die Suche ohne Ergebnis verlief, erkennen Sie daran, daß die Meldung »Suche…« nicht mehr im Mitteilungsfenster unten zu sehen ist.

4.19 Verschiedenes

Ctrl-P Voreinstellungen

Mit dieser Dialogbox können Sie verschiedene Optionen MonSTs einstellen. Die ersten drei Fragen verlangen J/N-Antworten. Sie können, wie in jeder Dialogbox, mit Esc abbrechen und mit Return die neuen Einstellungen übernehmen.

Bildschirm-Umschaltung

Normalerweise ist diese Option an; der Bildschirm wird erst 20 msec nach Kontrollübergabe an Ihr Programm auf den Bildschirm Ihres Programms umgeschaltet. Damit wird unnötiges Flackern verhindert. Diese Option sollte ausgeschaltet werden, bevor das zu debuggende Programm die Auflösung, die Bildschirmadresse oder die Farbpalette verändert; danach sollten Sie die Bildschirm-Umschaltung wieder einschalten.

Traps verfolgen

Normalerweise werden Traps, Line-A- und Line-F-Aufrufe als einzelne Befehle angesehen. Wenn »Traps verfolgen« eingeschaltet ist, werden solche Routinen verfolgt, um so auch das ROM untersuchen zu können.

Vorsicht: Diese Option sollte mit größter Vorsicht behandelt werden, da man damit sehr leicht großen Schaden anrichten kann. Manche zeitkritische Routinen, wie Floppy- oder Festplattentreiber, müssen ununterbrochen ablaufen. Eine Trace-Exception in solchem Code kann ohne weiteres zum Mißlingen und auch zum Datenverlust führen. Andererseits macht es Spaß, dem AES dabei zuzusehen, wie es Menüs oder sich öffnende Fenster aufbaut.

Wenn Sie Ihr Programm mit »Traps verfolgen« und »Run« ablaufen lassen, können Sie mit Shift-Alt-Help den Programmablauf unterbrechen und ihn mit normaler Geschwindigkeit mit Ctrl-R fortsetzen. Sowohl das AES als auch das VDI benutzen `Line-A` und `Line-F` Aufrufe, es ist deshalb sehr gut möglich, daß mehrere noch bevorstehende »Stack Frames« das gesetzte Trace-Bit enthalten. Es kann also den Anschein haben, als ob völlig unvorhersehbare Trace-Exceptions geschehen. Nach jedem derartigen Exception sollten Sie mit Ctrl-R fortfahren. Diese unregelmäßigen Exceptions werden nur dann aufhören wenn Sie auf tiefster Ebene, d.h. wieder in Ihrem Programm, sind.

Es gibt noch eine Nebenwirkung: Wenn durch AES-Event-Aufrufe »getraced« wurde, kann es vorkommen, daß auch »Stack Frames« innerhalb von Desk-Accessories mit gesetztem Trace-Bit existieren. Wenn das Programm zu Ende ist bevor das Desk-Accessory die Möglichkeit hatte, diese »Stack Frames« abzuarbeiten, wird ein Trace-Exception nach Verlassen MonSTs geschehen und einen Systemabsturz verursachen. Der Absturz erfolgt nur dann nicht, wenn AMonST2 oder das mitgelieferte

NOTRACE.PRG aktiv sind.

NOTRACE.PRG

Dieses kleine Programm wird auch als Source mitgeliefert und dient dazu, Trace-Exceptions zu ignorieren anstatt mehrere Bomben auszulösen. Das Programm sollte in Ihren AUTO-Ordner kopiert werden, falls Sie nicht schon AMonST verwenden.

Relative Offsets

Diese Einstellung ist normalerweise an und beeinflußt die symbolische Disassemblierung vom Adreßregister indirekt mit dem Offset Adreßmodus, d.h. xxx(An). Wenn die Option an ist, wird jeder derartige Adreßmodus ausgerechnet und dann die Symboltabelle durchsucht; wenn ein Symbol sich an dieser Adresse befindet, wird der Befehl als Symbol(An) disassembliert. Diese Option ist sehr hilfreich, sowohl bei manchen Assembler-Programmstilen als auch bei der Untersuchung von Hochsprachen, die ein Adreßregister als Basisregister benutzen. HiSoft Basic z.B. verwendet A3 als Zeiger auf das »Runtime«-System.

Symbole

Diese Option ermöglicht es, die Behandlung von Symbolen durch MonST zu verändern. Zuerst werden Sie gefragt, wie viele Buchstaben in einem Symbol signifikant sein sollen (minimal 8, maximal 22) und danach, ob die Symbole »Case-sensitiv« verarbeitet werden sollen. Diese Option ist für diejenigen gedacht, die nicht jedes Mal den ganzen Symbolnamen eingeben und nicht auf Groß- und Kleinschreibung achten wollen.

I Intelligentes Kopieren

Dieser Befehl kopiert einen Speicherbereich in einen anderen. Die Adressen sollten in folgender Weise eingegeben werden: <Anfang>,<Einschließliches_Ende>,<Ziel>

Das Kopieren ist deshalb intelligent, da die Ursprungs- und Endbereiche überlappen können.

Vorsicht: MonST überprüft nicht, ob die Adressen gültig sind. Eine Kopie von oder zu einer nichtexistenten Speicherstelle wird MonST wahrscheinlich abstürzen lassen. Das gleiche gilt für den reservierten Systemspeicher-Bereich.

W Speicher füllen

Dieser Befehl füllt einen Speicherbereich mit dem angegebenen Wert. Die Parameter sind: <Anfang>, <Einschließliches_Ende>,<Füllbyte>

Die Warnung über gültigen Speicher gilt auch für diesen Befehl.

L Labels anzeigen

Wenn MonST von GenST mit Debuggen aufgerufen wurde, wird ein Fenster geöffnet und alle Labels, die

sich z.Zt. in MonSTs Symboltabelle befinden, werden mit ihren Adressen angezeigt. Die Symbole erscheinen in der Reihenfolge, in der sie aufgerufen wurden. Dabei ist es unerheblich, ob sie von Diskette oder vom Speicher eingelesen werden.

P Disassemblieren zum Drucker/zur Diskette

Mit diesem Befehl können Sie einen Speicherbereich zum Drucker oder auf Diskette, sowohl mit vorhandenen Symbolen als auch mit »cross reference«-Labels, disassemblieren. Die erste Zeile der Dialogbox sollte mit <Startadresse>,<Endadresse> angegeben werden.

Die nächste Zeile fragt nach einem Speicherbereich, den MonST zur Erstellung der »cross reference«-Liste benutzen kann <Pufferanfang>,<Pufferende>

Wenn Sie dies nicht wollen, geben Sie einfach eine leere Zeile ein. Die nächste Zeile fragt nach den Speicherbereichen, die als DC-Direktiven disassembliert werden sollen. Die Zeile sollte so aussehen: <Datenanfang>,<Datenende>[,<Größe>]

Die optionale Größe ist die der `DCs` und kann B, W oder L sein, wobei L der Standardwert ist. Wenn alle Datenbereiche definiert wurden, geben Sie eine leere Zeile ein.

Zuletzt wird nach einem Dateinamen gefragt. Wenn keiner angegeben wird, geht die Ausgabe zum Drucker.

Wenn eine »cross reference«-Liste angefertigt werden soll, gibt es zuerst während der Erstellung eine kleine Pause. Die Labels, die so generiert wurden, haben das Format `Lxxxxxx`, wobei `xxxxxx` die Hex-Adresse des Labels ist.

Druckerausgabe

Das Zeilenformat enthält zuerst eine 8-stellige Hex-Zahl, bis zu 10 Wörter Hex-Daten, ggf. maximal 12 Zeichen eines Symbols, und darauf den disassemblierten Befehl. Die Druckerausgabe kann mit Esc abgebrochen werden.

Diskettenausgabe

Das Format kann direkt von GenST verarbeitet werden. Es enthält zuerst ggf. ein Symbol, dann den disassemblierten Befehl, dem ein Tab vorausgeht. Ein Tab trennt auch den Opcode vom Operanden. Wenn Sie einen Speicherbereich ohne Symbole disassemblieren, ist es ratsam, die »cross reference«-Option zu benutzen, da andernfalls keine Labels vorhanden sein würden. Ein Diskettenfehler oder Esc brechen die Ausgabe ab.

M Adresse verändern

Dieser Befehl ist aus Kompatibilitätsgründen zu MonST1 beibehalten worden. Er entspricht Alt-A.

O Auswerten

Dieser Befehl entspricht Alt-O und ist aus Kompatibilitätsgründen zu MonST1 beibehalten worden.

D Laufwerk und Pfad verändern

Mit diesem Befehl können Sie das aktuelle Laufwerk und den aktuellen Pfad verändern.

Ctrl-E Exceptions wieder installieren

Mit diesem Befehl werden die von MonST abgefangenen Exceptions auf ihren Urzustand gebracht. Dies kann nützlich sein beim Debuggen von kompilierten Programmen, deren Runtimes selbst Exceptions verwenden.

4.20 Auto-residenter MonST

Die dritte mitgelieferte Version von MonST heißt `AMONST2.PRG`. Wenn Sie dieses Programm im AUTO-Ordner Ihrer Boot-Diskette ablegen, wird es automatisch nach jedem Reset neu installiert. Diese MonST-Version ruht, bis ein Exception geschieht. Sie ist hauptsächlich für Programmierer gedacht, die AUTO-Programme und Desk-Accessories schreiben und debuggen wollen. Wenn ein Fehler wie ein Bus- oder Adreßfehler innerhalb einer dieser Programme auftritt, hängt die Maschine, bevor Sie einen interaktiven MonST aufrufen können. Es ist auch so möglich, einen illegalen Opcode wie `ILLEGAL` in Ihr Programm zu setzen und dadurch den autoresidenten MonST aufzurufen.

Die auto-residente Version kann auch normal vom Desktop aus aufgerufen werden und wird sich genauso initialisieren, als ob sie im AUTO-Ordner wäre, unter der Voraussetzung, daß nicht bereits ein anderer AMonST installiert ist.

Wenn sie aufgerufen ist, ist die auto-residente Version von MonST der interaktiven Version von der Bedienung her sehr ähnlich. Programme und dessen Symbole können aber nicht geladen werden, die »base-page«-Variablen sind unbekannt und auf Null gesetzt. Der andere Unterschied liegt darin, daß, wenn das Programm beendet wird oder AMonST mit Ctrl-C verlassen wird, AMonST trotzdem im Speicher bleibt.

Wenn AMonST installiert ist, kann jedes Programm mit Shift-Alt-Help unterbrochen werden. Der von MonST benötigte Speicher kann nur durch einen Reset zurückgegeben werden. Während der Installation von AMonST können Sie durch Drücken der beiden Shift-Tasten in das Programm gelangen, um z.B. Breakpoints zu setzen. Wenn Sie mit Ctrl-C AMonST verlassen, bleibt der Debugger resident, ein Verlassen mit Ctrl-R bewirkt, daß die Installation von AMonST unterbrochen wird.

Es kann eine interaktive MonST-Version geladen werden, obwohl ein AMonST installiert ist. Die interaktive Version hat Kontrolle bis sie beendet wird; AMonST wird dann wieder aktiv.

Vorsicht: Rufen Sie nie AMonST innerhalb eines anderen Programmes auf. Es kann dadurch vom Betriebssystem ein großer Speicherbereich bis zum nächsten Reset blockiert werden.

4.21 Debug-Strategien

Tipps: Wenn Sie ein Programm mit Shift-Alt-Help oder durch den Run- Until-Befehl unterbrochen haben, ist es gut möglich, daß der `PC` sich im ROM befindet. Die Methode, dahin zurückzugelangen, von wo aus

Ihr Programm das ROM aufgerufen hat, ist folgende: Wählen Sie die »Traps-Verfolgen«-Option in den Voreinstellungen. Dann geben Sie den Ausdruck sp=a7 beim Run-Until-Befehl ein. Sobald der Prozessor sich wieder im Usermodus befindet, also in Ihrem Programm, wird MonST aktiviert.

Wenn Sie sich in einer Unterroutine befinden, die Sie nicht interessiert, können Sie mit dem Until-Befehl (nicht mit Run-Until!) diese Routine überspringen: Geben Sie den Ausdruck `{sp}` als Zieladresse an. Falls die Unterroutine etwas auf dem Stack abgelegt haben sollte, oder einen lokalen Stack-Frame benutzt (oft bei kompilierten Programmen der Fall), kommen Sie mit Run Until {pc}.w=4e75 weiter. Das Programm wird langsam weiterlaufen, bis der `PC` an einem `RTS` Befehl angelangt ist. Diese Methode funktioniert dann nicht, wenn die Unterroutine eine weitere Unterroutine aufruft. In diesem Fall muß eine weitere Bedingung gestellt werden, z.B. ({pc}.w=4e75)&(sp>xxx) wobei xxx eins weniger ist als der momentane Wert.

Wenn Sie `Run Until` benutzen, werden Sie merken, daß es eine ganze Weile dauern kann, bis die Bedingung, die Sie angegeben haben, wahr wird. Sie können diese Zeit so kurz wie möglich halten, indem Sie den Ausdruck im voraus, soweit Sie können, ausrechnen.

(A3>(3A400-\100+M1)) kann auf A3>xxx reduziert werden, indem Sie xxx mit Hilfe von Alt-O ausrechnen.

Die MonST-Kommandozeile

Wenn Sie einen Kommandointerpreter verwenden, können Sie MonST eine Kommandozeile übergeben. Zuerst geben Sie den Programmnamen ein, und, falls erwünscht, eine Kommandozeile, die dem zu debuggenden Programm übergeben werden soll.

4.22 Die Bug-Jagd

Es gibt wahrscheinlich genausoviele Bug-Jagd-Strategien wie es Programmierer gibt, aber die Erfahrung ist doch der beste Lehrer. Wir haben in der Zeit, in der wir den 68000er programmieren, gelernt, wie man die Suche am besten angeht.

Zuerst ist ein sehr guter Weg, um Fehler zu finden, sich den Quelltext anzusehen. Es ist eine schlechte Angewohnheit, gleich zum Debugger zu greifen und sich erst dann den Quelltext anzusehen. Es könnte sein, daß Sie dabei das System oder die Programmierumgebung wechseln und nicht mehr auf das »Low-Level«- Debuggen zurückgreifen können.

Wenn ein Programm offensichtliche Fehler macht, wie z.B. einen Adreßfehler, ist es wesentlich leichter das Problem zu finden, als wenn das Programm nur manchmal, von Ihrer Sicht aus völlig unberechenbar, das falsche Ergebnis liefert.

Viele Fehler entstehen dadurch, daß bestimmte Speicherbereiche, die wichtige Daten enthalten, versehentlich überschrieben werden. Wenn man den Speicherbereich leicht, wie z.B. bei einem Busfehler, erkennen kann, kann man bedingte Breakpoints an verschiedenen Stellen in Ihrem Programm unterbringen, um den Fehler so früh wie möglich abzufangen. Wenn z.B. die globale Variable `Haupt_Zeiger` irgendwo ungerade wird, wird die Bedingung des Breakpoints wahrscheinlich so aussehen: {Haupt_Zeiger}&1

Wenn diese Methode nicht funktioniert, kann man davon ausgehen, daß die Variable an einer sonst unerfindlichen Stelle verändert wird. Die Lösung ist meist die Benutzung von `Run Until` mit der eben beschriebenen Bedingung. Es kann auf diese Weise sehr lange dauern bis das Problem auftaucht (deshalb auch das »Lämpchen« oben links). Es kann auch ohne weiteres der Fall sein, daß der Fehler auch so nicht zu finden ist. Manche Fehler sind sehr speicherspezifisch und werden durch einen Interrupt, während der Stack an einer bestimmten Stelle ist, ausgelöst. Es kommt auch vor, daß ein Fehler nur dann auftaucht, wenn ausgerechnet kein MonST zugegen ist; in einer solchen Situation hilft meist nur noch Glück.

Zähler-Breakpoints sind auch sehr gute Hilfsmittel, um einen Bug schnell zu lokalisieren. Wenn eine Routine nach einer bestimmten Zeit schief geht, Sie aber nicht sehen können warum, setzen Sie einen Zähler-Breakpoint in diese Routine und lassen das Programm ablaufen. Wenn der Fehler auftritt, schauen Sie sich mit Hilfe von Help an, wie oft die Routine durchgeführt wurde. Dann laden Sie das Programm neu und setzen einen Stop-Breakpoint an diese Stelle mit dem Wert des Zähler-Breakpoints (oder eins weniger). Lassen Sie das Programm laufen. Sie können die Routine dann »single-steppen«, falls sie schiefläuft.

Viel Glück!

4.23 Programme im Auto-Ordner

Wenn ein Programm im `AUTO`-Ordner abstürzt, kann es sein, daß der Rechner in einer unendlichen Schleife versucht, neu zu booten. Benutzen Sie in diesem Fall AMonST, indem Sie AMonST vor dem abstürzenden Programm in den `AUTO`-Ordner kopieren. So wird bei einem Exception der Rechner nicht neugestartet, sondern AMonST greift ein.

4.23.1 Desk-Accessories

Wenn ein Desk-Accessory nicht das tut, was es tun soll, sollten Sie AMonST benutzen. Um ein DA im Speicher zu finden, gehen Sie mit Shift-Alt-Help in AMonST. Dann suchen Sie ab Adresse 0 nach dem Namen Ihres Accessory. Es wird in Großbuchstaben und bis auf 8 Zeichen Länge mit Leerschritten ausgefüllt sein. Ignorieren Sie Vorkommnisse in Directory-Puffern (der Name wird darin von `.ACC` gefolgt) und in MonSTs eigenem Speicher, wobei dem Namen ein `ASCII-T` voransteht. Das richtige Vorkommnis wird 12 Byte nach dem Namen ein Langwort haben, das auf die Base-Page des Accessory zeigt; $100 Byte danach fängt es an. Von hier aus sollten Sie Ihre Hauptschleife erkennen und ein Breakpoint an einer geeigneten Stelle setzen. Mit Ctrl-R läuft dann das Accessory weiter bis zum Breakpoint.

Wenn schon bei der Initialisierung eines Accessory ein Fehler auftritt, müssen Sie es ganz am Anfang, bevor der Fehler passiert, anhalten. Der beste Weg ist, den `ILLEGAL`-Befehl zu verwenden und ihn durch AMonST abfangen zu lassen. Manchmal klappt diese Methode auch nicht. Die folgende Vorgehensweise funktioniert bei den aktuellen ST-ROMs um das AES zu stoppen, bevor es das Accessory startet. Es wird darauf hingewiesen, daß diese Methode kompliziert und nicht für den Anfänger gedacht ist.

Zuerst halten Sie beide Shift-Tasten gedrückt, damit Sie beim Bootvorgang in AMonST gelangen. Dann setzen Sie mit Ctrl-D ein BDOS-Breakpoint auf dem GEMDOS `f_open`-Aufruf, `$3D`. Danach benutzen Sie Ctrl-C, um den Bootvorgang weiterlaufen zu lassen. AMonST wird jedes Mal, wenn ein `f_open`-Aufruf erfolgt, aktiviert.

Machen Sie Fenster 3 zum aktuellen Fenster und setzen Sie jedesmal, wenn Sie auf ein Breakpoint kommen, die Startadresse des Fensters auf {sp+2}. Wenn der erscheinende Name nicht der Ihres Accessory ist, übergehen Sie den Aufruf mit Ctrl-Y. Setzen Sie dann noch ein BDOS-Breakpoint auf `$3D` und lassen den Bootvorgang mit Ctrl-R fortlaufen.

Wenn der Name Ihres Accessory doch auftaucht, setzen Sie ein BDOS-Breakpoint auf `$4B` und geben dann Ctrl-R ein. AMonST stoppt den Ablauf kurz bevor das Accessory geladen wird. Diese Reihenfolge - zuerst ein `f_open` und dann ein `p_exec` - mag zwar etwas seltsam erscheinen, ist aber erfolgversprechend. Mit Ctrl-Y über den GEMDOS-Aufruf und dann mit Alt-B mit der Adresse d0+100 wird ein Breakpoint auf den ersten Befehl des Accessory gesetzt.

Mit Ctrl-R lassen Sie das Betriebssystem so lange laufen, bis Ihr Accessory tatsächlich ausgeführt wird. Diese Methode mag zwar kompliziert und zeitaufwendig sein, sie ist oft aber die einzige Möglichkeit, ein Desk-Accessory zu debuggen.

4.23.2 Die Exception-Analyse

Wenn eine unerwartete Exception passiert, ist es meist sehr nützlich zu wissen, wo und wie sie passierte, um dann, wenn möglich, das Programm zum Weiterlaufen zu bringen.

Busfehler

Wenn der `PC` sich in nicht-existentem Speicher befindet, schauen Sie sich den Stack an und versuchen Sie, eine Rückkehradresse zu finden. Diese gibt meist Aufschluß über den Grund des jetzigen `PC`-Wertes. Wenn der `PC` sich innerhalb Ihres Programmes befindet, wird ein Zugriff auf einen nicht-existenten oder geschützten Speicher den Busfehler ausgelöst haben. Es ist sehr unwahrscheinlich, daß Sie nach einem Busfehler Ihr Programm weiterlaufen lassen können.

Adreßfehler

Wenn der `PC` sich nicht in Ihrem Programm befindet, ist die soeben erwähnte Methode anzuwenden. Ein Adreßfehler hat als Ursache meist einen Wort- oder Langwortzugriff auf eine ungerade Adresse. Eine Registerkorrektur hilft oft, daß das Programm wenigstens für eine Weile weiterlaufen kann.

Illegaler Befehl

Wenn der `PC` in einem sehr niedrigen Speicherbereich ist, etwa unter $30, ist meist ein Sprung an die Adresse `0` schuld. Wenn Sie sich mit MonST diesen Bereich ansehen, erkennen Sie einen `BRA.S`, gefolgt von mehreren `ORI`-Befehlen, die in Wirklichkeit Langwort-Zeiger sind, dicht gefolgt von einem illegalen Befehl.

Privilegverletzung

Dies wird durch die Ausführung eines privilegierten Befehls im User-Modus verursacht. Meistens bedeutet dies, daß Ihr Programm amokgelaufen ist.

4.24 Zusammenfassung der MonST-Kommandos

Fenster-Befehle

Alt-A	Startadresse setzen
Alt-B	Breakpoint setzen
Alt-E	Fenster editieren
Alt-F	Fontgröße
Alt-L	Fenster binden
Alt-O	Ausdruck auswerten
Alt-P	Drucker-Dump
Alt-R	Register setzen
Alt-S	Fenster spalten
Alt-T	Fenstertyp verändern
Alt-Z	Fenster zoomen

Bildschirm-Umschaltung

V	Anderer Bildschirm
Ctrl-O	Auflösung verändern

Breakpoints

Help	Breakpoint- und Segmentanzeigen
Alt-B	Breakpoint setzen
Ctrl-B	Breakpoint setzen
Ctrl-K	Alle Breakpoints löschen
Ctrl-A	Breakpoint setzen und ausführen
Ctrl-D	BDOS-Breakpoint
U	Go Until

Laden und Abspeichern

Ctrl-L	Ausführbares Programm laden
B	Binärdatei laden
S	Binärdatei abspeichern
A	ASCII-Datei laden

Programme ausführen

R	Run (verschiedene Arten)

Ctrl-R	Zurück zum Programm / Run
Ctrl-Y	»Single-Step«
Ctrl-T	Instruktion interpretieren (Trace)

Speicher durchsuchen

G	Durchsuche Speicher
N	Nächstes Vorkommnis

Verschiedenes

Ctrl-C	Abbrechen
Ctrl-P	Voreinstellungen
I	Intelligentes Kopieren
W	Speicher füllen
L	Labels anzeigen
P	An Drucker/Diskette disassemblieren
M	Adresse verändern
O	Auswerten
D	Laufwerk und Pfad verändern
Shift-Alt-Help	Programm unterbrechen
H	History-Speicher anzeigen
Ctrl-E	Exceptions wieder installieren

Benutzung der Assembler-Sprache von Personal Pascal

Hier wird geschildert, wie man Prozeduren und Funktionen in Assembler-Sprache schreibt und mit Pascal-Programmen linkt, indem man Devpac ST und OSS Personal Pascal verwendet. Um Personal-Pascal-kompatible Objekt-Dateien zu generieren, müssen Sie die Devpac ST Version 2.02 oder spätere verwenden, da frühere Versionen nicht kompatibel sind. Sie sollten DRI-Ausgabe- Code wählen, wie es im Handbuch (Seite 48) beschrieben ist.

Fügen Sie danach

```
COMMENT PASCAL
```

in Ihr Programm ein. Dadurch wird GenST angewiesen, das spezielle Dateiformat für Personal Pascal zu erzeugen. Die Deklaration der Funktionen und/oder Prozeduren muß mittels der `XDEF`-Anweisung erfolgen, deren Namen müssen in Großbuchstaben erscheinen. Ihre Anweisungen sollten in der `TEXT`-

Section erfolgen. Globale Variablen sollten Sie in der BSS-Section platzieren. Versuchen Sie nicht, die DATA-Section zu verwenden, denn dadurch könnte der Linker die Kontrolle verlieren.

Beim Assemblieren auf Diskette wird eine .O-Datei erzeugt. Diese ist jedoch nicht identisch mit einer .O-Datei im DRI-Format, denn erstere ist nur mit dem Pascal-Linker verwendbar. Sie können dannach diese Datei als eine zusätzliche Link-Datei gebrauchen. (Lesen Sie bitte dazu Seite 5-3 in Ihrem Handbuch)

Die Regeln bezüglich der von Pascal aufgerufenen Assembler Sprache können Sie auf den Seiten 6-142 finden.

Hier ist ein künstlich einfaches Beispiel, welches den Gebrauch von Parametern oder Return-Werten vermeidet. Es definiert ganz einfach eine Prozedur, einen String am Bildschirm unter Verwendung von GEMDOS auszugeben:

```
         opt       l2,c8+    DRI mode, short case sensitive labels

         COMMENT   PASCAL
         xdef      TESTING  declare export (s)
TESTING  move.l    #message,-(a7)
         move.w    #9,-(a7)
         trap      #1
         addq.l    #6,sp
         rts
message  dc.b      'Hello',13,10,0
```

Diese Funktion sollte in einem Pascal-Programm deklariert werden, die mit

```
PROCEDURE TESTING; EXTERNAL;
```

aufgerufen werden soll, wobei die .O-Datei von Devpac ST als zusätzliche Link-Datei beim Linken des Hauptprogramms spezifiziert ist.

Patch für MONST2.PRG

Zweck des Patches ist, alle Symbole relativ zum TEXT-Segment gerechnet verarbeiten zu können und *nicht* relativ zu den TEXT-, DATA- und BSS-Anfängen, wie das von GenST erzeugt wird.

Alle Symbole relativ zum TEXT-Segment erzeugt z.B. Turbo-C.

Gültig für MONST2.PRG, Version 2.01D, Länge 23.600 Bytes:

```
$1B0A   move.l $18(A1,D2)    ; get bias DATA
$1B12   move.l $8(A1,D2)     ; get bias TEXT
$1B1A   move.l $10(A1,D2)    ; get bias BSS
```

Der mittlere move.l ist das was wir wollen und muss an den Stellen der DATA und BSS moves so übernommen werden.