This document contains a scan of the printed manual for the Atari ST program Templemon.

Bedienungsanleitung zu TEMPLEMON (ab Version 1.5)

Lieber Benutzer des TEMPLEMONS,

Vielen Dank für die Unterstützung des Public Domain Gedankens.
Dies ist die offizielle Anleitung für den TEMPLEMON. Falls Ihnen die Anleitung gefällt und Bekannte von Ihnen daran interessiert sind, dann seien Sie bitte so fair und verweisen diese bitte an mich, da die Ausarbeitung von Programm und Anleitung viel Zeit und Arbeit gekostet hat.

Inhalt

Vorwort

Vielen Dank für Ihr Interesse an TEMPLEMON. Ich hoffe, daß diese Anleitung hilfreich für Sie sein wird. Ich habe mich bemüht, mit vielen Beispielen auch Anwendern, die unerfahren sind in der Verwendung von Debuggern zur Fehlersuche in Programmen, TEMPLEMON als hilfreiches Werkzeug zu erklären.

Empfehlenswert ist, daß Sie mit der Programmierung des 68000 ein wenig vertraut sind. Falls das noch nicht zutrifft, Sie aber dafür etwas von der allgemeinen Struktur eines Mikrocomputers verstehen, empfehle ich Ihnen zusätzlich etwas Lektüre, z.B. das Atari ST Intern von der Firma mit der Billig-Software und dem großen Vorrat an roter Farbe und vor allem ein Prozessorbuch zum 68000, z.B. Teil 1 des Bandes "M68000 Familie" vom Tewi-Verlag.

Um Sie gleich zu beruhigen: Es ist nicht notwendig, daß Sie in 68000-Sprache programmieren können, aber es ist nützlich, wenn Sie verstehen, was die einzelnen 68000er Anweisungen bewirken.

Übrigens : Dieser Monitor ist auf einem GEPARD-Computer mit Hilfe des Modula-2 / Assembler Entwicklungssystems programmiert worden und dann auf den Atari übertragen worden.

Die allgemeine Funktion des Mikroprozessors 68000 im Atari ST

Der 68000 im Atari ist von der Funktionsweise mit jedem anderen Mikroprozessor vergleichbar (nur eben nicht ganz mit den neueren Risc-Prozessoren). Verglichen mit einem BASIC Programmiersystem ist der Mikroprozessor dem BASIC Interpreter gleichzusetzen. Dabei ist nur zu beachten, daß der BASIC Interpreter noch durch eine Shell (von dort aus startet man Programme, speichert und lädt sie, usw.; also der Direktmodus beim BASIC) und einen Editor (ist meist im BASIC integriert; mit ihm erstellt und ändert man die Programmtexte) ergänzt werden, die man hierbei nicht mit in den Vergleich einbeziehen darf, sondern sich vorstellen muß, daß schon ein BASIC Programm mit "RUN" gestartet ist, und das der BASIC Interpreter abarbeitet.

Wenn man weiterhin davon ausgeht, daß in dem BASIC Programm jede Anweisung in einer eigenen Zeile steht, ist das wiederum gut mit dem Mikroprozessor vergleichbar, der ja direkt auf das RAM zugreift, das in viele aufeinander folgende Bytes (je 8 Bit) aufgeteilt ist, die, bei Null beginnend, durchnumeriert sind. Im Folgenden wird beim BASIC Interpreter von Zeilennummern, simultan dazu beim Mikroprozessor von Adressen geredet. Adressen werden beim 68000 in Longwords (4 Bytes bzw. 32 Bit) gespeichert und angegeben, wodurch ca. 4 Milliarden verschiedene Zahlen ausgedrückt werden können, folglich bis zu ca. 4000 Megabyte (Mill. Byte) vom Prozessor direkt angesprochen werden könnten. (Leider ist die Hardware des Atari nicht darauf ausgelegt.)

Ein BASIC Programm wird beim Start (mit "RUN") bei der ersten Programmzeile gestartet, der Mikroprozessor holt sich (wenn er auf der RESET-Leitung einen bestimmten Spannungswert erhält, der z. B. beim Einschalten des Computers automatisch erzeugt wird) von Adresse 4 bis 7 ein Longword und interpretiert den Wert dieses Longwords als Adresse, bei dem er beginnen soll, die Anweisungen abzuarbeiten. Es wird natürlich davon ausgegangen, daß schon beim Einschalten des Computers ein Programm und entsprechende Daten für den Mikroprozessor bereitstehen (z. B. in ROMs).

Der BASIC Interpreter merkt sich intern, wo er sich gerade im Programmtext befindet (z. B. die Zeilennummer), damit er auch weiß, wo er nach Ausführung einer Anweisung fortfahren muß. Ebenso muß er auch jede Anweisung (Zeile) adressieren könnnen, z. B. um ein GOTO auszuführen. In BASIC kann man in der Regel nicht auf diese irgendwo gespeicherte Zahl (Zeilennummer) zugreifen. Beim Mikroprozessor ist das anders. Der Mikroprozessor hat selbst auch ein Mini-RAM, in dem er sich seinen augenblicklichen Status merkt. Einige von diesen Speichern kann man mit den zur Verfügung stehenden Programmanweisungen ansprechen. Diese Speicher nennt man Register. So hat der Mikroprozessor Register, in denen Rechnungen, wie Subtraktion oder Multiplikation durchgeführt werden - man muß dazu die zu verrechnenden Werte per Programm erstma aus dem normalen Speicher in diese Register laden (kopieren) - und solche, wie den PC (program counter), der auf die gerade bearbeitete Anweisung zeigt. Um etwas, wie "GOTO 120" ausführen zu lassen, muß man beim Mikroprozessor eine Anweisung programmieren, die einen Adressenwert in das PC-Register speichert.

Der BASIC Interpreter liest nun Zeichen für Zeichen aus der abzuarbeitenden Programmzeile und versucht, daraus eine BASIC Anweisung zu erkennen (bitte nicht nörgeln, wenn Sie es besser wissen, das würde meine Ausführungen nur unnötig verkomplizieren). Es gibt parameterlose Anweisungen ("RETURN", "STOP", "NEXT", usw.) und die Anderen (z. B. "GOTO", "IF"). Einige nehmen Sonderstellungen ein, wie "NEXT" oder "PRINT", weil sie sowohl als auch (sie wissen schon) sein können. Dazu muß der BASIC Interpreter immer schon eine halbe Anweisung im voraus gucken, um zu wissen, was z. B. alles noch zur "PRINT" Anweisung gehört. Außerdem kann der BASIC Interpreter in vielen Fällen Schreibfehler erkennen (SYNTAX ERROR). Übrigens ist "A = 1" ebenfalls als Anweisung anzusehen, während "DATA" Anweisungen für den BASIC Interpreter eher wie "REM" Anweisungen angesehen werden, indem sie einfach von ihm übersprungen werden. Beim Mikroprozessor ist das mit den Anweisungen etwas strenger. Der 68000 holt sich immer ein Word (16 Bit) auf einmal an der aktuellen Adresse ab, um es als Anweisung zu interpretieren. Dieses Word

kann 65536 (= 2 ↑ 16) verschiedene Werte haben, also konnte man darin ebenso viele verschiedene Anweisungen kodieren. So viele verschiedene Anweisungen könnte sich aber wohl keiner merken. Stattdessen gibt es eine kleine Anzahl elementarster Anweisungen mit jeweils vielfältigen Parametern. Die Parameter sind zum Teil mit dem Anweisungscode zusammen in dem oben genannten Word kodiert. So kann der Mikroprozessor schon an diesem Word erkennen, ob und wie viele Bytes noch folgen, die als Parameter zur Anweisung dienen.

Das Schwierige bei Maschinenprogrammen ist nun, daß, falls man einen (meistens mehrere) Fehler darin hat, dieser sehr schwer aufspürbar ist. Entgegen BASIC, bei dem alles noch einigermaßen langsam abläuft und man jederzeit das Programm mit der STOP- (oder BREAK-) Taste abbrechen kann, hat man bei Maschinenprogrammen nicht diese komfortablen Möglichkeiten. Stattdessen muß man in die Programme selbst Fehlerprüfungsroutinen einfügen und vor Allem alles vor jedem Neustart auf Diskette sichern, da es leicht dazu kommen kann, daß man den Computer (das Betriebssystem) zum "Hängen" bringt und dann durch den Griff mit der Hand an die Gehäuserückseite einen manuellen, interaktiven Speicherlöschbefehl geben darf.

Gespeicherte, berechnete Werte (sprich Variablen) sind bei BASIC ebenfalls viel einfacher anzusehen (von der Kommandoebene aus per "PRINT" Anweisung) als in Maschinensprache, wo man erstens die Adresse kennen muß, wo die Variable steht (in BASIC braucht man dazu ja nur den Namen kennen) und zudem auch das Format der Variablen verstehen muß (eine Integerzahl ist im Speicher etwas anders abgelegt als z.B. eine Realzahl; ein String möchte schon gar nicht als Zahl interpretiert werden). Ach ja, damit ist hoffentlich klar, daß Variablen im Speicher überall stehen können und nur das Programm weiß, wie welche Speicherzellen zu interpretieren sind. Also, ob dort Programmanweisungen stehen, die es irgendwann mal anspringen wird, oder ob es Daten in irgendeiner selbstdefinierten Form sind, die natürlich nicht vom Mikroprozessor ausgeführt werden sollen. Damit ist es z.B. auch möglich, ein Programm zu schreiben, das sich selbst aus bestimmten Daten ein eigenes neues Programm erstellt, das dann von ihm zur Ausführung angesprungen wird.

So, wie in BASIC unter Laufzeit Fehler auftauchen können (z.B Division durch Null, kein Speicher mehr für Variablen, Wertüberlauf einer Variablen), erkennt auch der Mikroprozessor einige Fehler. In BASIC wird normalerweise bei Auftreten eines Fehlers das Programm mit einer Fehlermeldung gestoppt, dafür sorgt der BASIC Interpreter. Es gibt aber auch die Möglichkeit, mit einer Anweisung wie "ON ERROR GOTO ..." Fehler abzufangen und selbst per BASIC Programm auszuwerten. Da die meisten BASIC Interpreter nur ein Programm gleichzeitig zulassen, ist es leider dort nicht möglich, immer von vornherein ein bestimmtes BASIC Programm einmalig zu starten, das eine eigene Fehlerauswertung macht (z.B. die Fehlermeldungen in russisch ausgibt) und automatisch bis zum Ausschalten des Rechners aktiv bleibt. Genau das ist aber beim Mikroprozessor der Fall. Das Betriebssytem teilt gleich beim Booten dem Mikroprozessor mit, welche Maschinenspracheroutinen auszuführen sind, falls der Prozessor auf einen Fehler stößt. Nur haben sich da (zu meinem Glück) die Programmierer des TOS nicht sehr viel Mühe gegeben, sondern haben nur so eine alberne Bömbchen-Ausgabe implementiert.

Nun braucht man nur nach dem Booten dafür zu sorgen, daß statt der öden TOS-Routine ein anderes, etwas nützlicheres Programm gestartet wird, falls der Mikroprozessor auf einen Fehler stößt. Gedacht, getan. Aber was rede ich, Sie haben ja schon den TEMPLEMON (ist übrigens englisch auszusprechen).

So, genug von BASIC. Im folgenden werden öfter Kürzel wie "D0" (Dehnull), "PC", "A7" oder "SR" verwendet. Dies sind Bezeichnungen von Registern des 68000 und wenn Sie was von dem damit verbundenen Gefasel verstehen wollen, empfehle ich Ihnen zum allerletzen Mal die oben genannte Lektüre (es sei denn, Sie sind Autodidakt und werden auch so aus meinen Ausführungen schlau), denn sonst müßte ich mich bald nach einem größeren Verlag für diese Anleitung umsehen.

Allgemeine Arbeitsweise von TEMPLEMON

Sie haben hoffentlich längst die Anweisungen auf der Diskette mit dem TEMPLEMON befolgt, und ihn auf ihrer Bootdisk im Autoordner installiert. Wenn nicht, dann sollten Sie schleunigst Folgendes tun:
Kopieren Sie sich ihre Diskette, die Sie sonst einlegen, wenn Sie den Atari einschalten (sollten Sie ein ROM-TOS haben und keine spezielle Bootdisk verwenden, dann nehmen Sie nun halt eine leere Disk) und legen Sie auf ihr einen Ordner namens "AUTO" an, falls der nicht schon existiert. Dann kopieren die Datei "TEMPLMON.PRG" dort hinein. Wenn Sie nun mit dieser Disk den Atari booten, dann wird TEMPLEMON automatisch installiert. (Hardcopies wollten Sie doch sowieso nicht machen, nicht wahr ?!)

Ist TEMPLEMON mal nicht automatisch installiert worden, kann das immer noch nachgeholt werden, indem man ihn einfach durch Anklicken aktiviert.

TEMPLEMON kann dann entweder durch Drücken von ALT und HELP (ehemalige Hardcopy-Funktion) manuell aufgerufen werden oder er meldet sich von selbst, sobald der Mikroprozessor auf einen Fehler stößt (Zugriff auf nicht existierende Adresse, Zugriff auf ungerade Adresse, Division durch Null, versuchte Ausführung einer illegalen Anweisung, Befehlsverweigerung, usw.).

Vorsicht: Statt ALT/HELP keinesfalls die Hardcopy-Funktion vom Desktop-Menue mit der Maus aufrufen !

Wurde TEMPLEMON aufgerufen, sieht man in der Regel irgendwo einen blinkenden Cursor und darüber ungefähr zwei interessante Zeilen, z.B:

```
  !Unterbrechung durch Tastatur
  !R PC=xxxxxx USP=xxxxxx SSP=xxxxxx T0 S0 I3 X0 N1 Z0 V0 C0
```

Die erste Zeile zeigt immer an, aus welchem Grund TEMPLEMON sich meldet, die andere zeigt einige Registerinhalte an, die der Mikroprozessor unmittelbar vor der Aktivierung des Monitors hatte. Die Bedeutungen der Registerkürzel werden im Kapitel 'Funktionen von TEMPLEMON', bei der 'R'-Funktion erläutert. TEMPLEMON rettet sowieso alle Register des Mikroprozessors, bevor er sie für seine eigenen Funktionen mißbraucht. Nun können Sie dabei gehen und die geretteten Registerwerte ansehen und verändern und dann dem Monitor klarmachen, daß er sich nun wieder zu verabschieden hat, um das unterbrochene Programm mit den geretten, ggf. geänderten, Registerinhalten wieder fortfahren zu lassen. Dieser Monitor verhält sich somit so, als wenn er gar nicht vorhanden wäre für die anderen Programme.
Der Monitor sorgt dafür, daß im gesamten Rechner so gut wie nichts (außer dem für sich selbst beanspruchten Platz) verändert, wenn er mal aufgerufen wird. Vollkommen transparent (das nennt man so, glaube ich) kann er sich jedoch nicht verhalten. Z.B. müßten während seiner Aktivitätszeit alle Timer, die Uhr und die Interrupts gesperrt werden. Um diese Dinge zu meistern, ist aber ein hoher Aufwand zu treiben - beste Ergebnisse erzielt man mit spezieller Hardware - der sich normalerweise nicht lohnt.

Ein einfaches Beispiel für solche Probleme ist z.B. die Zeit während des Diskettenzugriffs. Hierbei stellt das zuständige Maschinenprogramm eine Anforderung zum Lesen (oder Schreiben) der Diskette an einen anderen Mikroprozessor. Sobald dieser bereit ist, die Daten zu Übertragen, erwartet er, daß der 68000 die Daten sofort bei Bereitstellung abnimmt. Dies sind 512 Bytes hintereinander, die innerhalb einer kurzen Zeit vom 68000 angenommen werden müssen. Wollte man nun während dieser Zeit, wenn auch nur kurzzeitig (z.B. mit der Trace-Funktion, s.u.) TEMPLEMON oder eine beliebige andere Routine aufrufen (per Interrupt) würde das Timing nicht hinkommen und es käme unweigerlich zum Datenverlust. Glücklicherweise haben die Softwarewillis bei Atari wenigstens in diesem Fall möglichen Fehlern vorgesorgt, indem sie während des Diskzugriffs den 68000 per Software in einen Modus versetzen, in dem er sich durch keinen Interrupt, also auch nicht durch ALT/HELP, stören läßt.

Übrigens: Wenn sich TEMPLEMON in irgendwelchen Programmen andauernd mit 'Division durch Null' meldet, obwohl die Programme früher nie einen Fehler (Bömbchen) gezeigt haben, lassen Sie sich dadurch nicht irritieren. Das spricht nämlich weniger gegen TEMPLEMON als vielmehr gegen das Programm, in dem der Fehler auftritt. Daß ohne TEMPLEMON diese Fehler nicht auftreten, liegt einfach daran, daß das TOS dafür keine Fehlerbehandlung hat, sondern den Prozessor trotz Fehlers ungehindert weiterlaufen läßt.

TEMPLEMON läßt sich im Groben für folgende Dinge einsetzen:

- Watchdog (Wachhund?): Der Monitor bleibt im Hintergrund. Falls ein Fehler auftritt, erfährt man die Ursache.

- Tracer / Programmtester: Die Auswirkung eines Fehlers ist erkannt worden. Nun wird im Tracemodus, ggf. mit einem eigenen Trace-Upro (Upro heißt Unterprogramm), die Ursache eingekreist.

- Langweilige Programme unverzüglich abbrechen. (Einfach ALT/HELP drücken, dann 'Q' eingeben).

- Usual stuff: Speicher durchwühlen, ändern, verschieben, abspeichern, Disassemblings auf Disk oder Drucker erstellen.

Was TEMPLEMON (z.Zt.) leider nicht kann:

- Mit Symbols arbeiten (also statt Adressen schreiben zu müssen, einfach die Labelnamen zu verwenden, die die meisten Assembler/Compiler/Linker optional zum Programmcode ausspucken).

- Ordentlich mit dem SID (Symbolischer Idiotischer Debugger) zusammenarbeiten.

- Direkt-Assemblieren.

- Programme zur Ausführung laden.

- Einen übersichtlichen Bildschirmaufbau bieten.

- Hardcopies zulassen.

- Einem die Arbeit abnehmen, "ausführliche" Anleitungen zu schreiben.

- Goldene Eier legen.

Die Funktionen von TEMPLEMON im Einzelnen

Eingabe der Kommandos

Steht der blinkende Cursor hinter einem Ausrufezeichen, wird damit angezeigt, daß der Monitor auf die Eingabe einer Kommandozeile wartet. Die Eingabe ist immer mit der RETURN-Taste zu bestatigen (im Gegensatz zur Kommandoeingabe beim Trace, wo nur eine einzelne Taste erwartet wird).

Im Folgenden werden Eingaben, an dessen Ende die RETURN-Taste gedruckt werden muß, mit einfachen Anfuhrungszeichen gekennzeichnet, wohingegen einzelne Tasteneingaben mit dem Zeichen "/" (Slash) eingeklammert werden

Groß-/Kleinschreibung ist gleichwertig, nur bei Angabe eines Full- oder Suchstrings (s.u.) ist sie zu beachten.

Werden Zahlenangaben erwartet, müssen sie normalerweise hexadezimal angegeben werden. Jedoch ist es moglich, stattdessen eine Dezimalzahl anzugeben, indem ihr direkt ein '&' vorangestellt ist. Beispiel: Um in Speicherstelle $1234 den Dezimalwert 56 zu schreiben, kann eingegeben werden: ":1234 &56"

Einzelne Adreßangaben konnen verschieden formuliert werden:
- Direkte Zahlenangabe, z.B "42E".
- Verwendung eines Registerinhalts (der geretteten CPU-Register): "PD7", "R A2" oder "RPC" (In Versionen 1.2 und 1.3 wurde "~" statt "R" verwendet).
- Direkte Zahlenangabe mit Konstant-Offset, z.B. "@12".
- Ausdrucksbildung mit '+' und '-', z.B. "RPC-20" oder "RA0+RD2+4". Achtung: durch schlampige Programmierung darf vor und nach dem Operatorzeichen kein Leerzeichen stehen !

Adreßpaare (Anfangs- und Endadresse) können oft abgekürzt werden. Als Beispiel hier das Disassembler-Kommando im normalen Format: "D 50456 50468". Wenn (nach der Syntax) hinter dem Adreßpaar keine weiteren Parameter erwartet werden, sind folgende zwei Abkürzungen moglich: Sofern man als Anfangs- und Endadresse den gleichen Wert verwenden möchte (nur angegebene Adresse soll angesprochen werden), kann die zweite Adreßangabe wegfallen (z.B "D 45338" disassembliert nur eine Zeile). Soll stattdessen die großtmogliche Endadresse verwendet werden ($FFFFFF), kann sie ebenfalls wegfallen und dafur ein Punkt direkt hinter der Anfangsadresse eingegeben werden (z.B "D 29960." disassembliert solange weiter, bis eine Taste gedrückt wird). Möchte man als neue Anfangsadresse die letze verwendete Endadresse (bei der abgebrochen wurde) angeben, kann sie weggelassen werden und stattdessen ein Punkt gesetzt werden (z.B. "D .30000" disassembliert weiter bis $30000, "D." disassembliert weiter bis zum nächsten Stop mit Tastendruck).

Die Endadresse kann auch als Anzahl der Bytes ab Anfangsadresse formuliert werden. Dazu wird ihr ein "I" direkt vorangestellt (z.B. "D 45552I20" disassembliert die nächsten 32 Bytes ab der angegebenen Startadresse).

Bei den Funktionen "D", "I" und "M" (Beschreibung s.u.) kann auch statt der Endadresse eine Zeilenanzahl mit "L" bestimmt werden (Beispiel: "D L8" disassembliert weitere 8 Zeilen).

Alle Ausgaben können sofort mit der SPACE-Taste angehalten (und mit der selben Taste fortgefuhrt) oder mit einer anderen Taste gestoppt werden (ihnen ist ja wohl klar, daß damit weder die Shift-Taste noch die RESET-Taste gemeint sind !).

Schön ist auch, daß der Monitor nicht den von GEM oder dem Anwenderprogramm benutzten Bildschirm überschreibt, sondern einen eigenen Speicherbereich mit eigenen Bildschirmroutinen dafür benutzt (sogar das Cursorblinken wird nicht über die normale Interruptroutinen erzeugt). Der normale Bildschirm kann sichtbar gemacht werden, indem die Taste "F2" gedrückt wird. Es können während dieser Anzeige ebenso wie sonst die Monitorkommandos eingegeben werden. Tritt eine besondere Situation auf, wenn Sie z.B. einen Fehler während der

Eingabe gemacht haben, wird wieder der Monitorbildschirm angezeigt. Sicherlich haben Sie in Ihrem Spieltrieb schon herausgefunden, daß mit "F1" ebenfalls auf die Anzeige des TEMPLEMON-Bildschirms zurückgeschaltet werden kann.

Die Funktionen

Anm.: Wie schon in der TEMPLEMON Kurzanleitung erwähnt, dienen in Klammern gesetzte Zeichen als Platzhalter (Parameter): <a> bedeutet Anfangsadresse, <e> Endadresse minus Eins, usw. Die Endadresse ist also immer die erste ausgeschlossene Adresse !

Außerdem wird empfohlen, die Kurzanleitung zusätzlich auch auswendig zu lernen ("sonst noch ein Wunsch ?" d. Sozzer). Dort ist alles nochmal etwas anders (vielleicht auch verständlicher) formuliert.

- Speicher hexadezimal anzeigen

"M <a> <e>"	zeigt den gewünschten Speicherbereich byteweise an
"M"	zeigt ab letzter Anfangsadresse nochmals an (Endadresse wird als $FFFFFF angenommen !).

- Speicher ändern

":<a> <w1> <w2> ..."	ab Anfangsadresse können bis zu 16 Bytes abgespeichert werden.

- Disassemblieren

"D <a> <e>"	Disassembliert den gewünschten Speicherbereich.
"D"	zeigt ab letzter Anfangsadresse nochmals an (Endadresse wird als $FFFFFF angenommen !)

- Speicher als Character anzeigen

"I <a> <e>"	Zeigt den gewünschten Speicherbereich in ASCII-Form
"I"	zeigt ab letzter Anfangsadresse nochmals an (Endadresse wird als $FFFFFF angenommen !)

- ASCII-Zeichen in Speicher eingeben

"'<a> abc..."	Nach einem Hochkomma, der ungeraden Anfangsadresse und einem Leerzeichen können bis zu 32 Zeichen folgen.

- Adreßoffsetvariable

"O <x>"	Die Variable "O" wird auf den Wert <x> gesetzt.
"O"	Zeigt aktuellen Wert der Variable an

- Speicher verschieben (Copy)

"C <a> <e> <d>"	Kopiert byteweise von <a> bis <e>-1 zum Speicherbereich ab <d>. Die Bereiche dürfen sich selbstverständlich überlappen.

- Speicher vergleichen (Verify)

"V <a> <e> <d>"	Vergleicht byteweise Bereich von <a> bis <e>-1 mit Bereich ab <d>. Ungleiche Werte in den Bereichen werden jeweils mit Adresse und Speicherinhalt angezeigt.

- Speicher mit Konstanten füllen

"F <a> <e> '<Zeichenkette>"
Füllt den Speicherbereich byteweise mit <Zeichenkette>
"F <a> <e> <b1> <b2> ..."
Füllt den Speicherbereich byteweise mit den angegebenen Bytes

Mit einem Fragezeichen angegebene Zeichen/Bytes werden nicht im Speicherbereich ersetzt.

- Speicher nach Konstanten durchsuchen (Hunt)

"H <a> <e> '<Zeichenkette>"
Durchsucht den Speicherbereich byteweise nach <Zeichenkette>. Gefundene Adressen werden angezeigt
"H <a> <e> <b1> <b2> ..."
Durchsucht den Speicherbereich byteweise nach den angegebenen Bytes. Gefundene Adressen werden angezeigt

Das Zeichen '?' wird bei beiden Sucharten als Joker verwandt. Damit kann man erreichen, daß so gekennzeichnete Zeichen in der gesuchten Zeichenfolge jeden Wert repräsentieren dürfen.

- Registeranzeige

"R"
Zeigt die Werte der standardmäßig ausgewählten Register an.
"R+"
Zeigt Inhalte aller Register an.
"R-"
Zeigt letzte Einsprungsmeldung an.

"R <reg1> <reg2> ..."
Zeigt die Werte der angewählten Register an.

"R <reg> = <x>"
Setzt angewähltes Register auf den Wert <x>.
<reg>: D0, D1, ... D7, A0, ... A7, PC, SSP, USP, SR (Statusregister als gesamtes Wort), F (Statusregister als Flags), BEV (Bus Error Vector), AEV (Address Error Vector).
"R F<f> = <x>"
Setzt Bit in SR (Statusregister).
<f>: T, S, I, V, N, Z, X, C.

"R:"
Zeigt Register-Standardauswahl an.
"R: <reg1> <reg2>"
Wählt neue Register-Standardauswahl.
"R: +"
Wählt alle Register.
"R: -"
Wählt kein Register.
"R + <reg>"
Addiert <reg> zur Register-Standardauswahl.
"R: - <reg>"
Subtrahiert <reg> von der Register-Standardauswahl.

Die Register-Standardauswahl wird bei Eingabe von "R" verwandt (s.o.). Außerdem bestimmt sie, welche Register beim Tracing angezeigt werden sollen.

"RS"
(Register Save). Sichert alle Prozessorregister (also nicht AEV und BEV), um sie nach Veränderung hinterher mit "RR" (s.u.) wiederherstellen zu können.

"RR"
(Register Restore). Stellt die Prozessorregister wieder so her, wie sie beim letzten Aufruf der "RS"-Funktion waren.

- Rechnen

"? <s> <o> <d>" Verknüpft die Zahlen <s> und <d> mit dem Operator <o> und zeigt das Ergebnis an.
<o>: +, -, *, /, O (binares Oder), A (binares Und) und E (binares Exklusiv-Oder).
Bei Addition und Subtraktion wird das Carry mit angezeigt, bei Division der Rest. Bei Multiplikation dürfen beide Operanden nur 16 Bit groß sein, bei der Division gilt das gleiche nur für den Divisor.

- Dezimal nach Hexadezimal umwandeln

"& <x>" Zeigt hexadezimalen Wert der Dezimalzahl <x> an.

- Augenblicklich unterbrochenes Programm abbrechen

"Q" Führt einen GEMDOS (0) Aufruf aus. Dies führt dazu, daß das gerade aktiv gewesene Programm terminiert wird. Außerdem werden alle Breakpoints gelöscht (wie "B-" Funktion).
Vorsicht: diese Funktion darf keinesfalls ausgeführt werden, wenn TEMPLEMON direkt von Desktop aus aufgerufen worden ist, da in diesem Falle das Desktop-Programm abgebrochen werden würde, was zu einem Systemabsturz führt, da es "unter" dem Desktop kein Benutzerprogramm mehr gibt ! Geben Sie in diesem Fall stattdessen 'G' ein, um ins Desktop zurückzugelangen. Hilfreiche Anwendung findet diese Funktion, wenn man aus einem anderen Programm heraus in den Monitor gelangt und aufgrund irgendwelcher Fehler erkannt wird, daß es sich nicht mehr lohnt, das Programm weiter fortlaufen zu lassen.

- I/O-Operationen

Wichtige Anmerkung: Die folgenden Funktionen rufen GEMDOS-Funktionen auf, die in der Regel nicht aus Interruptroutinen aufgerufen werden sollen, da es sonst zu Fehlern und Systemabstürzen kommen kann. Wenn Sie z.B. über ALT HELP in den Monitor gelangen, ist dies aber als Interrupt anzusehen. Fehler können dann auftreten, wenn in den Monitor gelangt wurde, während gerade eine der Dateifunktionen abgearbeitet wurde und nun nochmals aufgerufen wird oder wenn durch rekursive (wiederholt verschachtelte) GEMDOS-Aufrufe der Systemstack vollgelaufen ist. Deshalb seien Sie nicht verwundert, wenn die Ausführung einer der folgenden Funktionen mal nicht funktioniert.

"L <dateiname>" Lädt eine Diskdatei. Dabei wird mit Hilfe der Malloc-Funktion des GEMDOS entsprechend viel Speicher reserviert. Wenn nicht genügend Speicher vorhanden ist, wird nicht geladen und eine Fehlermeldung ausgegeben.

"L <dateiname>, <a>"
Lädt eine Diskdatei in den Speicherbereich ab Adr. <a>.

"L <dateiname>, <a> <e>"
Wie vorige Funktion, es wird jedoch höchstens bis zur vor die Adr. <e> geladen, um den Bereich ab <e> zu schützen.

"L- <a>" Gibt Speicher, der durch die vorige Load-Funktion reserviert wurde, wieder frei. <a> ist die Anfangsadresse, die bei der Ladefunktion angezeigt wurde.

"L <dateiname>, @<b>"
"L <dateiname>, @<b>, <a>"
"L <dateiname>, @<b>, <a> <e>"
Wie die vorigen drei Ladefunktionen, es wird jedoch nicht vom Anfang der Datei an geladen, sondern erst ab dem <b>-ten Byte der Datei.

"S <dateiname>, <a> <e>"
Speichert byteweise den Bereich von <a> bis <e>-1 in die Diskdatei mit dem Namen <dateiname>. Es darf noch keine Datei mit dem Namen <dateiname> existieren

"S <dateiname>, @<r>, <a> <e>"
Wie vorige Funktion, jedoch wird eine bereits bestehende Datei ab der <b>-ten Byteposition überschrieben.

"P" Löscht Bildschirm (gibt Form feed aus auf Datei bzw. Drucker).

"P <dateiname>" Öffnet Protokolldatei mit dem Namen <dateiname> (wie auch sonst). Alle Ausgaben gehen dann neben dem Bildschirm auch in die Datei. Die Datei kann sowohl auf Diskette als auch auf ein Periphergerat eröffnet werden Beispiel: "P PRN:" öffnet die Datei auf den Drucker.

"PC" Schließt die Protokolldatei. (Wird ebenso beim "Q"-Kommando ausgeführt.)

- TEMPLEMON-Vektoren

"VI" Vector Init.
Restauriert alle TEMPLEMON-Exception Vektoren (incl. ALT/HELP-Vektor). Ist nützlich, wenn ein anderer Debugger oder Monitor, wie z.B. SID, gestartet wurde, da dadurch mit Sicherheit einige wichtige Vektoren, zumindest zum Tracen, zerstört wurden.

- Breakpoints

Breakpoints sind so etwas wie Haltepunkte Sie geben Adressen an, die, wenn der Mikroprozessor auf sie trifft, um die dortige Anweisung auszuführen, eine Rückkehr in den Monitor veranlassen. Wenn Sie z.B. einen Programmteil untersuchen, der, weil er sehr lang ist, nicht Schritt für Schritt von Hand getraced werden kann, setzen Sie einfach ans Ende der Routine einen Breakpoint und starten sie dann am Beginn. Sobald der Prozessor dann das Ende der Routine erreicht, also die Adresse, auf die Sie den Breakpoint gesetzt hatten, gelangen Sie wieder automatisch in den Monitor. Neben der Breakpointadresse kann noch ein Zähler bestimmt werden, der bestimmt, beim wievielten Male des Erreichens des Breakpoints erst in den Monitor gelangt werden soll.

Nochmal ausführlicher: Breakpoints werden erst berücksichtigt, wenn der Monitor verlassen wird, um mindestens eine Anweisung des normal aktiven Programmes auszuführen. Es werden dabei zwei Fälle unterschieden: Ist beim Verlassen des Monitors das Trace-Flag des geretteten SR gesetzt, wird also nach Ausführung der Anweisung an der aktuellen Adresse wieder zurück in das Monitor-Programm verzweigt, dann wird nichts weiter unternommen, damit nach der Rückkehr in den Monitor der dann gerettete PC mit den Breakpointadressen verglichen werden kann. Ist stattdessen das Trace-Flag nicht gesetzt, müssen vor Verlassen des Monitors an alle Breakpointadressen eine bestimmte Break-Anweisung für den Mikroprozessor (diese Anweisung hat den Wert $4AFC) gespeichert werden. Wie Sie sicher wissen, gibt es bestimmte Speicherbausteine, auch ROM genannt, die sowas nicht mit sich machen lassen. Meine Empfehlung: Auf die Dinger Rücksicht nehmen und erst gar nicht versuchen, Breakpoints auf solche ROM-Adressen zu richten und dann TEMPLMON ohne Trace-Modus zu verlassen.

Etwas mehr zu den Zählern: Jedesmal, wenn der Monitor durch eine Break-Anweisung (gesetzter Breakpoint) im normalen Programm oder durch den aktiven Tracemodus aufgerufen wird, wird der PC (der auf die nächste, im normal aktiven Programm auszuführende Anweisung zeigt) mit allen Breakpointadressen verglichen. Stimmen PC und eine Breakpointadresse überein, wird der entsprechende Zähler um Eins erniedrigt. Wird der Zähler Null, wird eine entsprechende Meldung angezeigt und in die TEMPLEMON Kommandoeingabe verzweigt. Außerdem wird dabei der Zähler auf seinen Initialisierungswert gesetzt. Das hat den praktischen Nutzen, daß wenn man einen Breakpoint gesetzt hat und nach Ausführung des normalen Programmes über den Breakpoint zurück in den Monitor gelangt, nochmals die gleiche Routine starten kann, ohne den Breakpointzähler erneut setzen zu müssen.

Erfolgt beim Verlassen des Monitors die Meldung "Breakpoint konnte nicht gesetzt werden. .", zeigt dies an, daß entweder an der Breakpointadresse gar kein Speicher vorhanden ist oder daß das T-Bit im SR nicht gesetzt ist und deshalb keine Break-Anweisung ($4AFC) an die entsprechende Stelle im Speicher geschrieben werden kann, weil dort kein RAM sondern höchstenfalls ROM vorhanden ist. Abhilfe: Statt der SPACE-Taste, die den Monitor ohne Setzen der angezeigten Breakpoints verläßt, drücken Sie die ESC-Taste (dadurch kommen Sie zurück zur Haupt-Kommandoeingabe) und überlegen sich dann, ob Sie den Monitor nicht im Tracemodus ("T+"-Funktion) verlassen wollen, da dann keine Breakpoints im RAM gesetzt werden brauchen. Wird dann aber während des Tracens irgendwann das Tracebit vom Programm gelöscht (z.B., weil eine TRAP-Anweisung ausgeführt wird und Sie vorher nicht den kontinuierlichen Tracemodus gewählt haben, also nach "T+" u. "G" nicht /A/, sondern z.B. /O/ eingegeben haben), werden alle nicht setzbaren Breakpoints kommentarlos übergangen.

"B"	Zeigt alle belegten Breakpoints an. Die Ziffer nach dem "B" ist die Breakpointnummer, dann folgt die Adresse des Breakpoints (ist Sie Null, dann ist der Breakpoint unbenutzt), dann ein Initialisierungszahlerwert und der augenblickliche Stand des Zählers.
"B<n> <a>"	Setzt Breakpoint mit Nummer <n> (0-7) auf Adresse <a>. Der Zähler und sein Initialisierungswert werden auf Eins gesetzt.
"I<n> <a> <i>"	Setzt zusätzlich Initialisierungswert und Zähler auf den Wert <i>.
"B<n> <a> <i> <z>"	Setzt Initialisierungswert auf den Wert <i> und den aktuellen Zählerwert auf <z>.
"B<n> 0"	Löscht den Breakpoint wieder.
"B-"	Löscht alle Breakpoints. Wurden Breakpoints gesetzt, sollte diese Funktion ausgelöst werden, sobald das Programm, in dem die Breakpoints gesetzt wurden, beendet ist.

- User-Traceroutine (s u , Trace-Funktionen)

"BU"	Zeigt Adresse der User-Traceroutine an. Ist sie Null, wird keine User-Traceroutine beim Tracen aufgerufen
"BU <a>"	Setzt Adresse der User-Traceroutine auf <a>.
"BU 0"	Löscht Aufruf einer User-Traceroutine.

- Verlassen von TEMPLEMON, Aufruf von anderen Programmen

"G"	Verläßt TEMPLEMON und führt normales Programm bei der Anweisung fort, auf die das PC-Register zeigt (Das ist normalerweise die Adresse, an der der Mikroprozessor unterbrochen wurde, um in den Monitor zu springen)
"G <a>"	Wie oben, jedoch wird an der Adresse <a> mit der weiteren Programmausführung fortgefahren.
"GS <a>"	Verläßt TEMPLEMON. Adresse <a> wird als Unterprogramm ausgeführt. Am Ende des Unterprogramms, das mit der Prozessor-Anweisung "RTS" abschließen muß, erfolgt die Rückkehr in den Monitor mit entsprechender Meldung. Vorsicht: Soll hinterher das unterbrochene Programm fortgeführt werden, müssen vorher die Register gerettet werden, die das Unterprogramm überschreiben konnte (zumindest das PC-Register) ! Dies können Sie mit der "RS" -Funktion (s.o.) erledigen. Zur Sicherheit wird deshalb nach Rückkehr von dem Unterprogramm der PC auf Null gesetzt, damit Sie nicht gedankenlos einfach danach das alte unterbrochene Programm mit "G" mit den wahrscheinlich falschen Registerinhalten starten und es zu unerwarteten Fehlern käme.
"G ,<b1>,<b2>..."	Es können (von mir) sogenannte kurzzeitige Breakpoints beim Verlassen des Monitors angegeben werden (bis zu acht), die ebenso wie normale Breakpoints behandelt werden (mit Zählerwert Eins). Sobald wieder in den Monitor zurückgekehrt wird (egal, ob durch einen dieser Breakpoints oder durch eine der sonstigen Möglichkeiten), werden als erstes alle diese kurzzeitigen Breakpoints wieder gelöscht.
"G <a>,<b1> . "	
"GS <a>,<b1>..."	

- Tracemodus bestimmen

"T"	Zeigt den Tracemodus an (aktiv oder nicht).
"T+"	Schaltet den Tracemodus ein und setzt Trace-Flag im geretteten SR-Register auf Eins.
"T-"	Schaltet den Tracemodus ab und setzt Trace-Flag im geretteten SR-Register zurück (auf Null).

Wenn der Tracemodus aktiv ist, werden vor Verlassen des Monitors die standardmäßig gewählten Register (s. "R:"-Funktion) angezeigt, die folgende Anweisung disassembliert und auf einen einzelnen Tastendruck gewartet (s.u.). Ist beim Verlassen des Monitorprogramms das Trace-Flag des geretteten SR (auf Eins) gesetzt, wird nach Ausführung der ersten Anweisung des normalen Programms sofort in das Monitorprogramm zurückgekehrt. Dann wird einiges intern rumgewuselt (Breakpoints überprüft und evtl. Trace-Unterprogramm aufgerufen) und daraufhin entweder sofort (also ohne, daß erneut ein Kommando per Tastatur eingegeben werden muß) wieder zurück zum normalen Programm gesprungen oder, wie oben beschrieben, Register angezeigt, Disassembliert und auf eine Taste gewartet.

Folgende Tasten können gedrückt werden:

/SPACE/ Führt die angezeigte Anweisung aus und kehrt danach, sofern vorher das (hoffentlich) angezeigte Trace-Flag des SR gesetzt war, wieder hier in diese Tastenabfrage mit vorheriger Anzeige von Registern und nächster Anweisung zurück.

/ESC/ Verläßt diese Abfrageroutine und kehrt in die normale TEMPLEMON - Kommandoeingabe zurück. Wurden kurzzeitige Breakpoints gesetzt -beim G-Funktionsaufruf oder durch hiesiges Drücken von /B/ (s.u.)- werden sie wieder gelöscht !

/O/ (Display Off)
Führt die angezeigte Anweisung aus und kehrt danach nicht zur Trace-Anzeige mit Tastenabfrage zurück, sondern arbeitet blind, jedoch im Tracemodus, weiter. Zurück zur Monitoreingabe gelangt man nur, wenn der Prozessor im Programm auf einen Breakpoint oder einen Fehler (Exception) trifft, oder wenn ALT/HELP gedrückt wird. Trap-Routinen und ähnliche Exceptions werden nicht getraced (normalerweise steckt der Fehler ja nicht dort, sondern in Ihrem eigenen Programm) !

/A/ (Trace All)
Wie /O/, jedoch wird jedesmal vor Verlassen der Traceroutine des TEMPLEMON das Trace-Flag erneut gesetzt, was zur Folge hat, daß alle Anweisungen, also auch die TRAP-Routinen getraced werden. Lediglich Interruptroutinen werden ohne Trace durchlaufen (Prozessorbedingt). Zudem wird, wenn die Interrupt-Flags alle gesetzt sind (Interruptmaske auf Sieben), oder, wenn die Systemvariable "flock" gesetzt ist (Diskzugriff findet statt), das Traceflag rückgesetzt. Dann kann natürlich nur noch in den Tracemodus zurückgelangt werden (also TEMPLEMON die Kontrolle über das getracte Programm zurückerlangen), wenn die Routinen vor Setzen der Flags das SR gerettet haben und hinterher wieder zurückladen, womit dann wieder das normalerweise vorher gesetzte Trace-Flag wieder hergestellt wird und nach der nächsten Anweisung wieder in das Monitor-Traceprogramm verzweigt wird. Dies ist glücklicherweise bei den Diskroutinen des TOS der Fall. (Über die Sperrung aller Interrupts sage ich nichts, das können nämlich nur Sie sich selbst verpfuschen, will sagen, das TOS hat sowas gar nicht nötig.)

/R/ (Return from Subroutine)
Soll dazu dienen, bis zum Ende des Unterprogramms, in dem der Prozessor gerade sein Unwesen treibt, ohne Trace-Anzeige und Tastenabfrage vorzudringen. Ist aber etwas gefährlich: Wenn das Trace-Flag gerade nicht gesetzt ist, kann nicht die Traceroutine mit der Erkennung des Endes des Unterprogramms beschäftigt werden, sondern es muß eine direkte Rücksprungadresse in den Monitor auf den Stack geladen werden und gehofft werden, daß in dem Unterprogramm nicht schon irgendwelcher anderer Kram auf den Stack geladen ist, und die Routine mit "RTS" und nicht etwa mit "RTE" oder "RTR" abschließt. Also, am Besten, Sie passen immer auf, daß vor Benutzen dieser Funktion (also /R/) das Trace-Flag gesetzt ist. Wenn es nicht gesetzt ist, können sie das ja mit /T/ bzw. /1/ kurz ändern.

/Shift/ Beschäftigungstherapie für den unentschlossenen Benutzer. Die Eingabe hat Prioritätslevel 1 in der integrierten Ignorier-Event-Queue (d.S.).

/F/ (Fast)
Führt folgende Anweisungen ohne Tracing aus. Entgegen /0/ und /SPACE/ wird nach Rückkehr zur Anzeige jedoch das Trace-Bit wieder eingeschaltet.

/0/ Löscht T-Bit im SR.
Danach wird erneut auf eine dieser Tasten gewartet, ohne daß etwas ausgeführt wurde.

/1/ Setzt T-Bit im SR.
Danach wird erneut auf eine dieser Tasten gewartet, ohne daß etwas ausgeführt wurde.

/D/ Disassembliert nächste Anweisung. Kann mehrmals wiederholt werden. Danach wird erneut auf eine dieser Tasten gewartet, ohne daß etwas ausgeführt wurde.

/S/ (Skip)
Überspringt die als nächstes auszuführende Anweisung.

/B/ Setzt einen kurzzeitigen Breakpoint (s.o., G-Funktion) auf die Anweisung, die der als nächstes Auszuführenden folgt, oder, falls mit /D/ schon voraus disassembliert wurde, auf die letzte disassemblierte Anweisung. Danach wird erneut auf eine Taste gewartet, ohne daß etwas ausgeführt wurde.
Es können mehrere kurzzeitige Breakpoints nacheinander (mit zwischenzeitlichem, sinnvollen Anwenden der /D/-Funktion) gesetzt werden, bis zur Gesamtanzahl von Neun.

Auf diese Funktion bin ich besonders stolz. Damit können nämlich einige Probleme beim täglichen Debuggen sehr vereinfacht werden: Wenn die nächste Anweisung ein Unterprogrammaufruf ist, der nicht schrittweise durchlaufen werden soll, braucht nur /B/ und dann /O/ gedrückt werden. Sollte die Unterroutine gar überhaupt nicht im Tracemodus durchlaufen werden (weil zeitkritisch oder so), wird /B/ und dann /F/ gedrückt.

Das ist aber noch gar nicht das Beste ! Stellen Sie sich vor, Sie gelangen beim schrittweisen Tracen auf eine Schleife. Dann hätten Sie bisher (bis Version 1.3) entweder eine Zeit lang die SPACE-Taste festklemmen können oder /ESC/ drücken, die nächsten Anweisungen disassemblieren, dann einen Breakpoint hinter das Schleifenende setzen dürfen, 'G', /T/ und /SPACE/ eingeben, dann, nach Rückkehr in den Monitor den Breakpoint wieder löschen und mit 'T+' und 'G' fortfahren können. Und jetzt ? Nur noch bei Erkennen einer Schleife ein paar Mal /D/ gedrückt, bis man hinter das Schleifenende disassembliert hat, dann /B/ und noch entweder /O/ oder /F/ gedrückt. Schon ist man hinter der lästigen Schleife.

Die User-Trace Funktion

Dies eine besonders geniale, wenn auch nicht gerade einfach anzuwendende Funktion. (Bei dieser Gelegenheit ein Hoch auf Dirk Zabel, aus dessen Monitor 'Demon' seines Assembler-Pakets 'ASSI' für Commodore-Rechner ich viele Anregungen für TEMPLEMON erhielt !)
Sie haben die Funktion hoffentlich schon (mindestens) einmal benutzt, denn das Beispielprogramm 'TRACE.TOS' im TEMPLMON-Ordner macht von ihr Gebrauch. Der Sinn dabei ist, daß Sie ein eigenes Trace-Programm erstellen können, das bestimmte Vorgänge in ihren zu testenden Programmen überwachen kann. Das User-Trace-Programm wird nach jeder im Tracemodus ausgeführten Anweisung aufgerufen, sofern natürlich das Tracebit im SR gesetzt war und die Anweisung selbst keine andere Exception auslöst. Einfacher: das User-Programm wird immer aufgerufen, wenn auch die schon vorher beschriebenen Breakpoint-Überprüfungen durchgeführt werden. Damit das Programm überhaupt vom Monitor aufgerufen werden kann, muß mit der 'BU'-Funktion oder sonst irgendwie die Adresse des User-Programms nach $3F0 geschrieben werden (das Beispielprogramm 'TRACE.TOS' schreibt die entsprechende Adresse bei Aufruf selbständig nach $3F0, siehe dazu auch die Source des Programms: 'TRACE.C'). Wird nun in einem Programm ein Fehler gesucht, von dem die Auswirkungen bekannt sind, z.B., daß eine Variable am Ende einen unmöglichen Wert hat, bindet man am Besten in die Source des Programms diese User-Traceroutine ein. Es empfiehlt sich, bei C zwei Include-Files zu erstellen, bei Modula-2 ein passendes Definitions/Implementationsmodul (wirds ab Dez. 86 auch von mir geben, wartens Sie's nur ab ("Alklevel: 1.5 Promille" d.S.)), die die Definitionen der Register und die Initialisierung der Routine beinhalten, so daß nur noch die wesentliche Traceroutine jedesmal individuell neu programmiert werden braucht. Hick ! (Licher Pilsner ["2 Promille" d.S.])
Wenn Sie dann schon die Routine in Ihrem Programm installieren, dann können Sie auch noch gleich einen Sprung in den Monitor veranlassen, indem Sie dort z.B. in Assembler die ILLEGAL-Anweisung programmieren (sie hat den Code $4AFC), damit beim Starten des Programms gleich von Ihnen evtl. Breakpoints und vor allem 'T+' eingegeben werden können, damit die User-Traceroutine auch ständig aufgerufen wird. Die User-Routine wird dann immer im Supervisormodus aufgerufen und bekommt in Register A0 die Adresse des vom Monitor zuvor geretteten Registersatzes übergeben. Die aktuellen Werte der Prozessorregister sind undefiniert !!! Bis auf D7, das obere Byte von SR und natürlich SSP dürfen alle Prozessorregister in der Routine verändert werden. Die Routine muß dann mit 'RTS' abschließen und im unteren Byte von D0 einen wohlüberlegten Wert haben: Ist er Null, wird die nächste Anweisung, auf die das gerettete PC-Register zeigt, ausgeführt, ansonsten wird das Tracing gestoppt und TEMPLEMON meldet sich zurück. Natürlich werden wieder alle geretteten Register zurückgeladen, was Ihnen erlaubt, die geretteten (!) Register, z.B. die Interruptmaske im SR, vorher in der User-Traceroutine zu verändern.

Interner Ablauf der Trace-Funktion

Es folgt ein etwas eigenwilliges Diagramm über die Vorgänge beim Eintritt und Verlassen von TEMPLEMON, vor allem im Tracemodus, sowie einige Abläufe im Mikroprozessor bei Ausführung einer Anweisung. Dies soll den Fortgeschrittenen unter Ihnen ermöglichen, z.B. wenn Sie eine User-Traceroutine programmieren wollen, die komplizierten Abläufe zu erkennen um die damit verbundenen Probleme berücksichtigen zu können.

Da es mir zu umständlich ist, mit einem Texteditor Flußdiagramme oder so was zu erstellen, werden die Abläufe ähnlich wie in einem Assemblerprogrammsource mit Labels (unterstrichen) und Upros dargestellt ["Unidentifizierbare PROgrammteile, oder was ?" d.S.].

Es gibt mehrere Arten, vom normalen Programm in den Monitor zu gelangen: Eine Gruppe bilden ALT/HELP und die meisten Exceptions (Bus Error, Division durch Null, usw.): Die zugehörigen Vektoren zeigen alle in das Monitorprogramm, das einfach dann die Fehlerursache anzeigt und dann die normale Befehlseigabe des TEMPLEMON aufruft. Die zweite Möglichkeit ist das einfache Laden einer Rücksprungadresse (zurück in den Monitor) auf den Stack beim Verlassen des Monitors, wie es bei Benutzen der 'GS'-Funktion Anwendung findet. Ind der dritten Gruppe befinden sich der Trace-Exception-Vektor, die 'Illegal instruction'-Exception (z.B. für Breakpoints) und wiederum der Trick mit der Rücksprungadresse auf m ["oje" d.S. ("OJO!" d.T.)] Stack für das Trace-Kommando /R/ (nur wenn T-Bit im SR gelöscht). Die erste Gruppe wird hierbei nicht weiter behandelt, die zweite wird kurz beim 'G'-Funktionsablauf erklärt.

An dieser Stelle seien eben noch einige Regeln und Tücken des 68000-Prozessors erwähnt:

- Erfolgt eine Exception, werden mindestens das SR und der aktuelle PC auf den Supervisorstack gerettet und dann sowohl der Supervisormode aktiviert (so daß ab dann immer der Supervisorstack benutzt wird) als auch das Tracebit im SR gelöscht. Zuletzt wird noch ein neuer PC aus der Exceptionvektortabelle geholt und dann ganz normal mit der Befehlsausführung fortgefahren.

- Führt der Prozessor einen Befehl aus, merkt er sich zuerst, ob das Tracebit im Augenblick gesetzt ist. Dann führt er die Anweisung aus, auf der der PC steht. Erzeugt die Anweisung selbst eine Exeption, wird, wie eben beschrieben, die dafür zuständige Routine aufgerufen. Z.B. wird bei einem Bus-Error in den Monitor verzweigt, bei einem Line-A Befehl wird die Routine des GEM aufgerufen. Die Line-A und Line-F Routinen des GEM holen sich dann ihre Parameter, führen ihre Funktionen aus und am Ende wird dann die Anweisung 'RTE' ausgeführt, welche dafür sorgt, daß vom Supervisorstack wieder PC und SR, die beim Auftreten der Exception gerettet wurden, zurückgeladen werden und mit der normalen Befehlsausführung fortgefahren wird. Wenn im Befehlsausführungszyklus jedoch keine Exception auftrat, wird zuerst der PC auf die folgende Anweisung gesetzt und dann geprüft, ob vor (!) dieser Befehlsausführung das Tracebit gesetzt war. Ist dies der Fall, wird eine Trace-Exception ausgelöst. Was dabei passiert, wird im Folgenden (hoffentlich) erklärt. Beachtenswert ist also, daß sich die Line-A und Line-F Routinen nicht mit den einfachen Mitteln des TEMPLEMON tracen lassen, da sie sich gar nicht vom gesetzten Tracebit irritieren lassen ! Dagegen können normale Exceptions, wie z.B. die TRAPs, getraced werden, weil Sie als ein Befehlszyklus behandelt werden: Ist vor der TRAP-Anweisung das Tracebit gesetzt, wird nach Behandlung der TRAP-Exception mit Laden des neuen PC, Löschen des Tracebit, usw., der Zyklus zu Ende gebracht, also sich erinnert ["der Prozessor sich, oder wer?" d.S] , daß vorher das Tracebit noch gesetzt war, und eine Trace-Exception ausgelöst wurde. Wenn dann in der Traceroutine (im TEMPLEMON) darauf reagiert wird, indem das Tracebit wieder gesetzt wird, kann sich die TRAP-Routine nicht mehr ohne Trace-Kontrolle davonmachen.

- Interrupts werden, solange Sie nicht vom Hauptprogramm durch eine Maske im SR gesperrt sind, hemmungslos zu jeder Zeit ausgeführt. TEMPLEMON ändert auch nicht die Interruptmaske ! (Das ist auch der Grund, weshalb sich die Maus weiterhin bewegen läßt, wenn man sich im Monitor befindet.)

Angefangen wird beim Verlassen des Monitors mittels der 'G'-Funktion:

Eingabe 'G' (evtl. mit Parameter)

Folgt Adreßangabe ?

Ja: Adreßwert holen und merken

Kurz-BP auswerten

Folgt Komma ?

Ja: Adresse holen, als Kurz-BP eintragen und nach <Kurz-BP auswerten>.

War 'GS' eingegeben ?

Ja: War Adresse angegeben ?

Nein: --> Fehlerausgabe

Ja: Rücksprungadresse nach <GS-Rückkehr> auf Stack laden und A7 (also SSP bzw. USP korrigieren)

Steht PC auf einem normalen BP ?

Ja: Zähler um Eins erhöhen.

Jetzt wird die angegebene Startadresse, falls angegeben, dem gesicherten PC zugewiesen.

Ist der Tracemode aktiv ?

Ja: <Trace-Entry $1> aufrufen (Trace-Anzeige wird eingeschaltet).

Verlassen von TEMPLEMON

Soll alles getraced werden (/A/ eingegeben) ?

Ja: Ist flock ($43E) gesetzt
oder Interruptmaske im SR auf 7 und Supervisorbit gesetzt ?

Nein: T-Bit im SR auf 1 setzen.
Ja: T-Bit im SR auf 0 setzen.

Ist T-Bit im SR gesetzt (also erfolgt eine einigermaßen zuverlässige Rückkehr zum Monitor nach der Befehlsausführung) ?

Nein: An alle Breakpointadressen den Code $4AFC (ILLEGAL) schreiben.

Trat Fehler auf ?

Ja. Fehler anzeigen und auf Taste warten.

Wurde ESC eingeben ?

Ja: Abbruch. Zurück zur Haupt-Kommandoeingabe.

Gerettete Register zurückladen (PC und SR auf Stack) und RTE ausführen.

Ausführung einer Anweisung des Hauptprogramms

(Es folgen prozessorinterne Abläufe, immer bis '-->')

Bus-Zugriffsfehler (Gruppe Null der Exc.-Prioritäten) ?

Ja: Befehlszyklus abbrechen,
BUS o. ADDRESS ERROR - Exception auslösen.
--> TEMPLEMON meldet sich, Hauptkommandoeingabe

Illegale Anweisung (Gruppe Eins der Exc.-Prioritäten) ?

Ja: Befehlszyklus abbrechen,
ILLEGAL INSTRUCTION - Exception auslösen.
--> Ist Anweisung = $4AFC ?
Nein: TEMPLEMON meldet sich, Hauptkommandoeingabe
Ja: Ist PC eine der Breakpointadressen ?
Nein: TEMPLEMON meldet sich, Hauptkommandoeingabe
Ja: Ist BP ein kurzzeitiger ?
Ja: Sprung nach <Trace-Entry $2>
Nein: TEMPLEMON meldet sich mit Breakpoint-
Angabe, Hauptkommandoeingabe

TRAP - Anweisung (Gruppe Zwei der Exc.-Prioritäten) ?

Ja: Exceptionbarbeitung durchführen
(PC und SR retten, T-Bit löschen, S-Bit setzen,
PC auf Adr. des Exception-Vektors setzen)
Kein Abbruch (!), sondern weiter.

War Tracebit im SR am Anfang der Befehlsausführung gesetzt ?

Ja: Exceptionbearbeitung für Trace-Exception durchführen
und weiter.

Interrupt aufgetreten ?

Ja: Exceptionbearbeitung für Trace-Exception durchführen,
Interruptroutine eben ausführen und hier weiter.

So, hier beginnt nun ein neuer Befehlszyklus. D.h., daß einfach die Anweisung, auf der der PC jetzt steht, behandelt wird. War z.B das T-Bit vor der Befehlsausführung gesetzt und der Befehl war eine TRAP-Anweisung, steht nun der PC auf dem Anfang der Traceroutine und da das Tracebit ja von der Exceptionbehandlung gelöscht wurde, wird die Traceroutine (die des TEMPLEMON) normal durchlaufen. Die Traceroutine baut vom Stack erstmal den obersten PC und SR ab und legt sie im RAM ab (es sind die Register, die mit den normalen Anweisungen des TEMPLEMON zugänglich sind). Als das SR gerettet wurde, war das Tracebit schon gelöscht (Nur das vorher durch die TRAP-Anweisung gerettete SR hat ein gesetztes Tracebit, sodaß nach dem RTE der TRAP-Routine das Tracebit wieder gesetzt ist). Also würde normalerweise die TRAP-Routine nicht im Tracemodus durchlaufen werden, da ja keine Trace-Exceptions ausgelöst würden. Da aber vorher noch einmal die Traceroutine zum Zuge kommen kann, ist es möglich, das Tracebit im vom Stack geholten SR zu setzen, damit dann auch die TRAP-Routine weiter getraced werden kann. (Diese Möglichkeit macht sich TEMPLEMON bei der /A/-Funktion zunutze, um die ROM-Routinen zu tracen.)

Trace-Exception-Einsprung

Register retten (D0-A6, SSP u. USP direkt speichern, PC und SR von Stack)

Ist Tracemodus aktiv ?

   Nein T-Bit im SR löschen, alle Register zurück und mit RTE
         zurück zum unterbrochenen Programm.

R-Rückkehr

Ist Longword auf Adresse $3F0 (User-Trace Vektor) ungleich Null ?

   Ja: Adresse des Beginns der geretteten Register nach A0 laden.
       An die Adresse, auf die $3F0 zeigt, per JSR springen.

       Ist D0 B Null ?

          Nein: Abbruch des Tracing, Sprung zur Haupt-Kommandoeingabe
                mit Ausgabe 'Stop durch User-Trace'.

Trace-Entry #1

Ist PC auf einer Breakpointadresse ?

   Trace-Entry #2

   Ja: Ist Breakpoint ein Kurzzeitiger ?

       Ja: Trace-Anzeige wieder anschalten

       Nein: Zugehörigen Zähler erniedrigen.

             Ist Zähler Null ?

                Ja: Init-Wert dem Zähler zuweisen und Abbruch mit
                    Angabe des Breakpoints, zur Haupt-Kommandoeingabe.

                Nein: Interne Flags setzen, so daß, falls Tracemodus nicht
                      aktiv, die Original-Anweisung an die BP-Adresse
                      zurückgetragen wird, die Anweisung dann im Trace-
                      mode (gesetztes T-Bit im SR) ausgeführt wird, hin-
                      terher das T-Bit wieder gelöscht und die ILLEGAL-
                      Anweisung wieder an die BP-Adresse geschrieben
                      wird.

Falls /R/ eingegeben war, prüfen, ob A7 größer dem Vergleichswert ist
und zudem der PC auf einer RTS, RTE oder RTR Anweisung steht. Ist das
der Fall, werden die internen Flags so gesetzt, daß nach der nächsten
Befehlsausführung die Trace-Anzeige wieder eingeschaltet wird.

Ist (oder wurde) Trace-Anzeige eingeschaltet ?

   Nein: Rückkehr zu <Verlassen von TEMPLEMON>

Trace-Anzeige-Routine

War gerade eine Anweisung getraced worden (also kein direkter
Aufruf von 'G'-Kommandoeingabe, sondern von Trace-Einsprung o.ä.) ?

   Ja: Kurzzeitige Breakpoints wieder löschen

Anzeige der Register und Disassemblieren der Anweisung auf (PC).

Warten auf eine Taste

SPACE: Sprung nach <Verlassen von TEMPLEMON>

0: Löschen des Tracebits im SR und Sprung nach <Warten auf Taste>.

1: Setzen des Tracebits im SR und Sprung nach <Warten auf Taste>.

ESC: Abbruch des Tracing und Sprung zur Haupt-Kommandoeingabe.

O: Trace-Anzeige ausschalten und Sprung nach <Verlassen von TEMPLEMON>

A: Trace-Anzeige ausschalten, internes Flag für 'alles tracen' setzen und Sprung nach <Verlassen von TEMPLEMON>.

R: Ist T-Bit im SR gesetzt ?
  Ja: Merken des A7 als Vergleichswert, internes Flag setzen zur Prüfung, Trace-Anzeige aus und Sprung nach <Verlassen von TEMPLEMON>.
  Nein: Laden der Adresse von <R-Rückkehr> auf Stack und Sprung nach <Verlassen von TEMPLEMON>.

B: Falls nur die nächste Anweisung disassembliert ["Ratespiel: Welches Wort fehlt hier ? Lösungen bitte an den Autor" d.S.], wird die Adresse der folgenden Anweisung, sonst die als letzte disassemblierte Adresse als kurzzeitiger Breakpoint eingetragen, welcher dann beim nächsten Ankommen bei <Trace-Anzeige-Routine> oder in der Haupt-Kommandoeingabe wieder gelöscht wird. Sprung nach <Trace-Anzeige-Routine>.

S: Setzen des PC auf die folgende Anweisung, Sprung nach <Warten auf Taste>.

F: Trace-Bit im SR löschen, Trace-Anzeige aus und internes Flag setzen, sodaß bei nächstem Einschalten der Trace-Anzeige (z.B. durch Breakpoint) oder bei Rückkehr direkt zur Haupt- Kommandoeingabe das Trace-Bit wieder gesetzt wird. Sprung nach <Verlassen von TEMPLEMON>.

GS-RÜckkehr

Sprung zur Haupt-Kommandoeingabe mit Ausgabe 'Rückkehr von GS'

Tips und Beispiele zur Käferjagd

Sie werden im Folgenden einige Beispiele zum Bedienen von TEMPLEMON finden. Sollte das eine oder andere Eingabebeispiel nicht bei Ihnen funktionieren, kann es daran liegen, daß Sie eine alte Version des Monitors besitzen (also Version 1.2 bis 1.4). In diesem Fall empfehle ich Ihnen, über den Public Domain Service der Zeitschrift ST-Computer sich eine neue Kopie der Diskette mit dem TEMPLEMON zu besorgen. Sollten Sie allerdings schon die Version 1.5 besitzen, muß der Fehler wohl wo anders dran liegen (ähem) ["<- kein Kommentar vom Sezzer" d.S].

Aufspüren von fehlerhaften Buszugriffen in Programmen

Am Häufigsten wird sich beim Ausprobieren von Programmen TEMPLEMON mit der Meldung BUS EPROR oder ADDRESS ERROR melden. Ursache ist dafür meist eine uninitialisierte Pointer-Variable oder (vor allem in C) falsch übergebene o. berechenete Adreßwerte. Die Ursache zu finden, ist dann oft nicht so leicht (zumindest mit TEMPLEMON). Zuerst stellt sich die Frage, an welcher Stelle im Programm der Fehler auftrat. Bei TEMPLEMON, der ja leider (noch) keine Symbols benutzen kann, muß man dabeigehen, die Anweisungen um den PC herum zu disassemblieren und dann mit dem Assemblerlisting des Programms vergleichen, um die Routine zu erkennen. Falls der Fehler reproduzierbar ist und Sie einen anderen Debugger, der Symbols verarbeiten kann, besitzen (natürlich müßten Sie ihn auch bedienen können), dann nehmen Sie am Besten diesen dazu zur Hilfe. Wenn Sie dann die Stelle im Programmsource gefunden haben, mag es sein, daß der Fehler nicht durch eine falsch benutzte Variable hervorgerufen wurde, sondern daß der Fehler in einem Unterprogramm auftrat und der fehlerhafte Wert von der aufrufenden Routine übergeben wurde. Dann heißt es, die aufrufende Routine zu finden. Hierbei wird ihnen allerdings kein anderer Debugger besser helfen können. TEMPLEMON bietet Ihnen hier zwei Möglichkeiten. Die schnellste ist die manuelle Suchmethode:

Wenn die Routine, in der der Fehler auftrat, ein Unterprogramm war, muß auf dem Stack ja noch die Rücksprungadresse zum Aufrufer liegen. Deshalb Disassemblieren Sie die Unterroutine erstmal weiter (mit "D R PC."), um dort irgendwo dann einen Aussprung aus der Routine zu finden. Da Sie hoffentlich das Disassembling ["Headcrash ?" d.S] lesen können, werden Sie dabei erkennen, daß evtl. vor dem Verlassen der Routine mit 'RTS' noch einiges vom Stack abgeräumt wird (lokaler Variablenbereich, usw.). Finden Sie heraus, wie weit der Stack abgebaut wird. Geben Sie dann "M R A7." ein (und drücken Sie zu gegebener Zeit SPACE oder ESC) und Sie erhalten einen Auszug des Stacks, beginnend bei den letzten darauf abgelegten Daten. Durch forsches Umherstreunen mit Ihren Augen ["Hast schon wieder 'nen Pils drin ?" d.S.] über diesem Speicherauszug, sollte es Ihnen nun ein Leichtes sein, die Rücksprungadresse zur aufrufenden Routine zu finden. Merken Sie sich diese Adresse und finden dann wiederum durch disassemblieren der betreffenden Routine den äkwivalenten ["Info für den gebildeten und sprachlich interessierten Leser, falls er diesen verbalen Anschlag auf diese unsere Sprache überlebt hat: gemeint ist das Wort das lateinischen Ursprung hat und dort soviel wie 'gleichwertig' bedeutet" d. S.] Bereich in Ihrem Programmsource.

Hier ein Beispiel:

Das Programm laute folgendermaßen in Assembler (wenn Sie in Hochsprache programmieren, z.B. in C, dann sehen Sie in Ihrer Dokumentation dazu nach, wie Sie ein Assemblerlisting oder Symbolfile optional dazu erhalten. Beim Megamax-C geht sowas auch, erkundigen Sie sich dazu bitte beim Distributor !) :

```
            ; Hauptprogramm:

Start:      MOVE.L   #Message1,-(A7) ; Laden der Adresse eines Strings
                                     ; auf den Stack.
            BSR      Print           ; Unterprogramm aufrufen
            MOVE.L   Message2,-(A7)  ; Laden (fälschlicherweise) der ersten
                                     ; 4 Bytes des Strings aufn Stack
            BSR      Print           ; Unterprogramm aufrufen
            RTS                      ; Beenden der Hauptroutine

Message1:   DC.B   'H','e','l','d','u','n','g','1',13,10,0
Message2:   DC.B   13,10,'H','e','l','d','u','n','g','2',13,10,0

            ; Unterprogramm (erwartet Zeiger auf String, um ihn auszugeben):

Print:      MOVE.L   -8(A7),A0       ; Laden des Pointers auf den String
PrLoop:     MOVE.B   (A0)+,D0        ; Laden des Zeichen, das in der Adres-
                                     : se, auf die A0 zeigt, steht, nach D0,
                                     ; zus. erhohen des Registers A0 um Eins.
            BEQ      PrEnde          ; Wenn D0 = Null, dann Ende
            ANDI.W   #$00FF,D0       ; Löschen des oberen Bytes von D0.W
            MOVE.W   D0,-(A7)        ; Laden von D0 auf den Stack
            MOVE.W   #2,-(A7)        ; Laden des Funktionswertes für Ausgabe
                                     ; eines Zeichens (GEMDOS (2)) den Stack.
            TRAP     #1              ; Gemdos-Routine aufrufen
            ADDQ.L   #4,A7           ; Stack wieder korrigieren
            BRA      PrLoop          ; Und nochmal von vorn.
PrEnde:     RTS                      ; Beenden des Unterprogramms
```

Wenn dieser Programmteil (ab 'Start') durchlaufen wird, meldet sich TEMPLEMON mit der Ausgabe 'BUS ERROR auf Adr. 4D656C64 /.../ Instr.: 1028'. Außerdem sehen Sie noch weitere Register, wie z.B. den PC. Der Wert 656C64 ist die Adresse, auf die der Prozessor zugreifen wollte, der Wert 1028 ist der Kode der Anweisung, bei der der Fehler auftrat und auf welcher der PC steht. Mit der Eingabe "DRPCZ8" bekommen Sie dann die nächsten acht Anweisungen, beginnend bei der, die den Fehler auslöste, disassembliert. Sie werden erkennen, daß der Fehler bei der Anweisung 'MOVE.B (A0)+,D0' auftrat. Wenn Sie "RA8" eingeben, werden Sie dort ebenfalls die Adresse, auf die nicht zugegriffen werden konnte, sehen. Wenn Sie nun auch noch einige Anweisungen vor dem PC disassemblieren (das müssen Sie leider noch von Hand ausrechnen. Es empfiehlt sich, ca. $20 Bytes vorher mit dem Disassemblieren anzufangen, also geben Sie ein 'D <hier Wert für PC-20> RPC') werden Sie herausfinden, daß das Register A0, das ja den falschen Wert enthielt, vorher vom Stack geladen wurde. An dem Disassembling ["$?!" d.S.] ist erkennbar, daß zu dem Zeitpunkt, als der Fehler auftrat, nichts außer der Rücksprungadresse auf dem Stack geladen ist. Also reicht es, "MRA7X4" einzugeben, um den obersten Wert auf dem Stack zu ermitteln. Dieses ist die Adresse, bei der die Programmausführung weitergeht, wenn das Unterprogramm mit RTS abgeschlossen hat. Also nochmals die letzten Zeilen bis zu dieser Adresse disassemblieren und damit die letzte Zuweisung von A0 finden. Hierbei sollte auf die Anweisung 'MOVE.L Message2,-(A7)' gestoßen werden. Wenn Sie an dieser Stelle immer noch nicht den Fehler erkennen, geben Sie doch 'T+' ein und starten dann nochmal mit 'G <Hier Angabe der Adresse von 'MOVE.L Message2..>' und gehen dann den Programmteil schrittweise mit der SPACE-Taste durch und beobachten dabei die Registerveränderungen (insbesondere A0). Möchten Sie gerne das Programm fortführen, nachdem Sie die Fehlerursache erkannt haben, könnten Sie entweder, solange PC und A7 noch stimmen (wenn Sie die Register mit dem vorher vorgeschlagen nochmaligen Ablaufen der Routine im Tracemodus verändert

haben -zumindest PC und A7 werden dabei verändert- hätten Sie sie besser vorher mit 'RS' gerettet und mit 'RR' hinterher wieder hergestellt), A0 mit "R A0 = <richtige Adresse von Meldung2>" korrigieren und dann einfach mit 'G' das Programm fortfahren (der PC steht ja noch auf der 'MOVE.B ...'-Anweisung, so daß sie dann mit dem nun richtigen A0-Wert nochmal ausgeführt wird). Oder Sie schalten den Tracemodus ein (mit 'T+') und überspringen dann mit öfterem Drücken der Taste /S/ die ganze Unterroutine, natürlich nur bis vor die RTS-Anweisung, und lassen dann das Programm durch Drücken von /G/ und /SPACE/ normal weiterlaufen, nur eben, daß dann der zweite String nicht ausgegeben wurde.

Ach ja, da ja nur zwei Aufrufe des Unterprogramms existierten, hätte man auch mit Hilfe der F2-Taste sich die bisherigen Ausgaben ansehen können und dann daraus, daß die erste Meldung schon ausgegeben wurde, schließen können, daß nur der zweite Aufruf an dem Fehler schuld sein konnte.

Die zweite Methode wird mit der User-Traceroutine realisiert. Dabei geht man so vor, daß der kritische Programmteil im Tracemodus durchlaufen wird und dabei eine selbstprogrammierte Routine auf ein Ereignis wartet, das den schon bekannte Fehler verursachen könnte

Bei dem oberen Beispiel könnte z.B. darauf gewartet werden, daß der Wert, der den Bus Error erzeugt, benutzt wird. Das wäre allerdings nicht ganz einfach, da dann bei jeder Anweisung, die getraced wird, das benutzte oder übertragene Datum (von 'Data') erkannt werden müßte, um es mit dem gesuchten Wert vergleichen zu können. Und das ist, wie gesagt, sehr, sehr aufwendig.

Die beiden Beispiele aus dem File 'TRACE.C' im TEMPLMON-Ordner zeigen schon ganz gut die beiden verschiedenen Einsatzmöglichkeiten. Erstens können gerade ausgeführte Anweisungen direkt überwacht werden (als Beispiel das Abwarten eines Trap- bzw. Line-A Aufrufs), zweitens quasi parallele Vorgänge, die nicht direkt von den Anweisungen abgeleitet werden können (z.B. das Beschreiben von Datenbereichen, wie dem Bildschirm).

Gerade die zweite Möglichkeit findet oft Anwendung. Oft werden nämlich Fehler an falsch berechneten Daten erkannt. Dann braucht nur ein User-Traceprogramm die betreffenden Datenfelder dauernd zu überprüfen, bis irgendwelche unerlaubten Werte auftreten (z.B., daß eine Variable ihren Wertebereich verläßt). Es ist auch möglich, automatisch beim ersten User-Trace-Aufruf über die Datenfelder eine Checksumme zu bilden, die dann vielleicht alle zehn ausgeführten Anweisungen die Checksumme überprüft und daran unerlaubte Schreibzugriffe auf die Datenbereiche erkennt.

Ein besonders hilfreiches User-Traceprogramm sei als letztes hier abgedruckt, welches es erlaubt, Aufzeichnungen über die letzten Programmabläufe vor Auftreten eines Fehlers zu führen. Diese Routine könnte bei dem vorigen Beispielprogramm ebenfalls benutzt werden, um herauszufinden, von welcher Programmstelle das Unterprogramm 'Print' als letztes aufgerufen wurde.

Als erstes muß für die im folgenden verwendete Variable 'AddrList' ein Speicherbereich mit der Anzahl von 'ListLength' Bytes angelegt werden. Darin werden dann die Adressen der letzten Anweisungen, die vor einem erneuten Sprung in den Monitor (z.B. wegen dem lang ersehnten Fehler) ausgeführt wurden, abgelegt. Ist der Speicherbereich voll, dann wird wieder am Anfang des Speicherbereichs fortgefahren. Somit zeigt die Variable 'ListIndex' auf hinter die letzte geschriebene Adresse. Wenn nun das User-Traceprogramm gestartet wird und der Fehler auftritt, muß nur der Wert von 'ListIndex' ermittelt werden und dann kann einfach der Bereich davor, bis zum Anfang des Puffers, und dann evtl. noch vom Ende des Puffers bis zum Index, mit der 'M'-Funktion angesehen werden, um die letzten abgearbeiten Anweisungen herauszufinden. Leider habe ich nicht daran gedacht, für die Anzeige dieser Werte Routinen bereitzustellen, auf die das User-Traceprogramm zugreifen könnte. Dafür sollte man wie in dem folgenden Beispiel die Adreßwerte leicht zugänglich machen. Im Beispiel werden die Werte hinter der User-Traceroutine abgelegt (jeweils ein Longword für Anfang des Puffers, Ende des Puffers und dem

Zeiger hinter die letzte abgelegte Adresse) sodaß man nur mit 'BU' die Adresse der Traceroutine ermitteln und dann mit Disassembling und Hex-Dump sich die drei Werte hinter der Routine ansehen braucht. (Vielleicht wird diese Funktion in einer späteren Version fest implementiert sein.)

Im User-Traceprogramm kann außerdem bestimmt werden, wie weit zwei Anweisungen auseinanderliegen müssen, damit ihre Adressen aufgezeichnet werden. Das Programm ist in der abgedruckten Form mit dem Megamax-C Compiler übersetzbar.

```
#include <osbind.h>

/* Die folgenden beiden Werte sind vom Benutzer frei bestimmbar */
#define ListLength 8000 /* Für 2000 Adressen (je 4 Bytes) */
#define Diff 20         /* Mind. 20 Byte Abstand zw. den Adressen */

#define RegPC   68(A0)

char AddrList [ListLength+4];     /* der Puffer für die Adressen */
                                  /* (mit Sicherheitsbereich)    */

static _usrTrc (), ListBegin (), ListIndex (), ListEnde ();

asm {
LastPC: DC.L    0

_usrTrc:
        LEA     LastPC(PC),A1   ; Adresse von LastPC laden
        MOVE.L  RegPC,D1        ; Adresse der aktuellen Anweisung laden
        MOVE.L  D1,D2           ; und zum Rechnen kopieren
        SUB.L   (A1),D1         ; LastPC (letzte Adresse) subtrahieren
        BPL     isPos           ; -> Ergebnis ist positiv
        NEG.L   D1              ; Positiven Wert bilden
isPos:  MOVE.L  D2,(A1)         ; LastPC aktualisieren
        CMPI.L  #Diff,D1        ; Mit Mindestabstand vergleichen
        BCS     ende            ; zu klein, -> nicht eintragen

        LEA     ListIndex(PC),A3
        MOVE.L  (A3),A2
        MOVE.L  D2,(A2)+        ; PC in Puffer ablegen und
                                ; Pufferzeiger um 4 Byte erhöhen
        CMPA.L  ListEnde(PC),A2 ; Zeiger schon am Ende des Puffers ?
        BCS     ende2           ; Nein
        MOVE.L  ListBegin(PC),A2;Zeiger wieder auf Pufferanfang setzen
ende2:  MOVE.L  A2,(A3)         ; und neuen Pufferzeiger abspeichern

ende:   SF      D0              ; kein Stop des Tracing (D0 auf Null)
        RTS

ListBegin:  DC.L 0              ; hier werden die Adressen zu dem
ListEnde:   DC.L 0              ; Puffer, wie oben beschrieben, zu
ListIndex:  DC.L 0              ; finden sein  (Die Werte werden in
                                ; der Routine 'install_trace' gesetzt )
}
```

```
install_trace ()
{
    asm {
        LEA     _usrTrc(PC),A0  ; Adresse der Traceroutine für
        MOVE.L  A0,0x3F0        ; TEMPLEMON initialisieren

        ; Diverse Variablen initialisieren:

        LEA     AddrList(A4),A1 ; Zeiger auf Pufferanfang laden
        LEA     ListBegin(PC),A0
        MOVE.L  A1,(A0)
        LEA     ListIndex(PC),A0
        MOVE.L  A1,(A0)
        ADDA.L  #ListLength,A1  ; Zeiger auf Pufferende berechnen
        LEA     ListEnde(PC),A0
        MOVE.L  A1,(A0)
    }
}

long keep, prgtop;
extern char *_base;

main ()
{
    Supexec (install_trace);

    asm {
        MOVE.L  A7,prgtop(A4)
    }
    keep = ( prgtop + 0x100 ) - (long) _base;
    Ptermres ( keep, 0 );
}
```

Umgehen von unkritischen Fehlern, z.B. 'Address Errors' bei Ramdisks oder 'Division durch Null'-Fehlern

Vor allem bei C-Programmen kommt es manchmal vor, daß sich TEMPLEMON mit der Fehlermeldung 'Division durch Null' zeigt, obwohl ohne den Monitor das Programm bisher immer ohne Bömbchen funktionierte. Das liegt daran, daß normalerweise der zugehörige Exceptionvektor auf eine RTE-Anweisung zeigt, sodaß ein Auftreten dieser Exception praktisch keine Reaktion auslöst. Natürlich ist das eigentlich nicht in Ordnung, wenn mögliche Fehler auf diese Weise ignoriert werden, aber es kann auch sein, daß die Programme trotz Auftreten der Exception (sie wird ausgelöst, wenn eine der Divisionsanweisungen der 68000 durch Null teilen soll) ohne Anzeichen eines Fehlers weiterlaufen. Ist das Programm eines der Ihren, sollten Sie sich trotzdem zur Sicherheit daran setzen, den Fehler vor einer Division durch Überprüfen des Divisors abzufangen und die notwendigen Konsequenzen daraus ziehen (z.B. die Division dann nicht durchzuführen). Tritt der Fehler bei fremden Programmen auf, an denen Sie nicht 'herumdoktoren' können, brauchen Sie nur 'G' im TEMPLEMON eingeben, um das Programm wie immer weiterlaufen zu lassen (Der PC wurde schon automatisch bei der Exceptionbehandlung auf die Anweisung hinter der Divisionsanweisung gesetzt).

Mehr Probleme gibt es da schon bei Bus Fehlern und Address Fehlern. Tritt eine solche Exception auf, steht der PC auf der fehlererzeugen Anweisung. Die Eingabe von 'G' würde sofort wieder zum Fehler führen (können Sie ruhig tun, es geht dabei nichts kaputt). Stattdessen müssen Sie erst die Fehlerursache erkennen, um dann entscheiden zu können, ob Sie die Anweisung einfach überspringen können, oder den Fehler korrigieren müssen um dann die Anweisung fehlerfrei durchzuführen.

Address Fehler treten immer dann auf, wenn auf ein Word oder ein Longword zugegriffen werden soll und die Zugriffsadresse ungerade ist. Häufig tritt dies z.B. bei schlecht programmierten RAM-Disks auf. Hier können Sie dann mit der Eingabe von 'D R PC' die fehlerauslösende Anweisung ansehen und dann das Adreßregister, das die ungerade Adresse enthält, auf einen geraden Wert korrigieren und dann mit 'G' die Routine fortlaufen lassen. Dabei ist dann aber damit zu Rechnen, daß das Programm, das über die Ramdisk die Daten geladen oder gespeichert hat, ein Byte zuviel oder zuwenig geladen bzw. abgespeichert hat, da die manuelle Korrektur des Adreßregisters dem Programm nicht bekannt ist.

Ein Beispiel mit einem Address Error beim Laden eines File von der RAM-Disk:

TEMPLEMON meldet sich z.B. mit 'ADDRESS ERROR auf 00024ED3 /...'.
Nun wird 'D R PC' eingeben und dabei ungefähr folgende Zeile ausgegeben:
```
!,xxxxxx  MOVE    (A1)+,(A2)+
```
Die Namen der Adreßregister mögen varrieren. Nun sehe man sich die beiden bei der Anweisung benutzten Adreßregister an (also A1 und A2): 'R A1 A2' zeigt dann bespielsweise:
```
!R A1=0000AF56 A2=00024ED3
```
In diesem Beispiel enthält nun A2 den falschen Wert. Das Einfachste ist nun, einfach A2 auf 00024ED2 oder auf 00024ED4 zu setzen und dann mit 'G' das Laden fortfahren zu lassen. Dabei geht aber mindestens ein Byte irgendwo verloren (Beim Texteditor ist so etwa ja noch leicht zu korrigieren).

Besser geht man so vor, daß man sich den Wert auf Adresse 24ED2 merkt, dann A2 auf diese Adresse setzt, dann die Routine zum Speicherverschieben im RAM-Disk Programm ablaufen läßt, aber durch vorheriges Setzen eines Breakpoints dafür sorgt, daß man hinterher wieder zurück in den Monitor gelangt. Dann wird wiederum das Register A2 angesehen, das, wenn der Breakpoint nicht zu weit gesetzt wurde, auf das Ende des geladenen Bereichs zeigen sollte (kann deshalb von ausgegangen werden, weil es unklug wäre, eine solche RAMDISK-Routine anders zu programmieren, oder ?). Nun werden durch Eingabe von 'C 24ED2.R A2 24ED3' die Daten an ihre richtige Adresse verschoben und noch das Byte auf Adresse 24ED2

wieder hergestellt (mit ': 24ED2 xx', wobei xx der gemerkte Wert ist).

Bus Fehler können aus vielen Gründen auftreten. Wenn Sie aufmerksam mitgelesen haben, dürfte Ihnen ein Beispiel weiter oben schon bekannt sein. Hier nur kurz eine Aufzählung der möglichen Fehler:

- Es wird auf eine Adresse zugegriffen, an der weder RAM noch ROM noch irgendwelche I/O-Bausteine liegen.

- Es wird versucht, in ROM zu schreiben.

- Ein Programm, das sich im Usermodus befindet, greift auf Speicherbereiche zu, die nur im Supervisormodus angesprochen werden dürfen. Dies sind die ersten paar KByte im RAM (von Adresse Null an) und der gesamte I/O Bereich (ganz oben im Speicher ab $FF8000). Sollte aus diesem Grund ein Fehler auftreten, kann er einfach umgangen werden, indem man folgendes eingibt: 'F FS=1' um das Programm in den Supervisormode zu versetzen, 'T+' um den Tracemode einzuschalten, 'G' und die /SPACE/ Taste, um die fehlererzeugende Anweisung einzeln auszuführen, dann die /ESC/ Taste und 'R FS=0', um das Supervisorflag wieder zurückzusetzen und zuletzt 'T-' und 'G', um das Programm normal weiterlaufen zu lassen.

Ebenso kann die Meldung 'Privilegsverletzung' auftauchen, wenn unerlaubte Anweisungen im Usermode ausgeführt werden sollen. Hierbei kann dann der Fehler auf die gleiche Weise wie eben beim Bus-Error beschrieben, umgangen werden.

Ende

So das wär's fürs Erste. Mir fallen zwar noch eine Menge Tips ein ("wie immer !" d.S.), andererseits ist es nach fast 75000 Anschlägen und einer Bearbeitungszeit von über einem Monat langsam an der Zeit, endlich die versprochenen Anleitungen auszuliefern (Ich glaub sowieso nicht, daß irgendwer diesen ganzen Kram komplett durchliest ["und ohne größeren intelellen Schaden übersteht" d.S.]).

Ich hoffe, daß Sie, auch wenn Sie nicht alles verstanden haben, doch etwas schlauer durch diese Anleitung geworden sind.

Sollten Sie aber trotzdem noch irgendwelche Fragen speziell zu TEMPLEMON haben, bin ich gerne bereit, falls Sie mich telefonisch (bitte nicht schriftlich, ich habe genug vom Schreiben !) erreichen, Ihnen zu helfen. Allerdings gilt dieses Angebot nur für alle, die bei mir die Anleitung erworben haben. Probieren Sie es unter der folgenden Telefonnummer: 0441 / 486231.

Ach, ich bekam vor einigen Wochen einen Brief von gewissen P. Freyer (weiß jemand, ob der für diesen Laden in Düsseldorf arbeitet ?), in dem er meinen Monitor als 'besonders wertlos' bezeichnete und erst gar nicht nach einer Anleitung fragte, sondern von vornherein mir klarmachte, daß mein Programm nicht funktioniere: '[...] läßt sich der Monitor je nach Betriebssystem-Version entweder überhaupt nicht aufrufen, oder das System kehrt nicht ordnungsgemäß ins GEMDOS zurück und hängt sich auf. Da lob ich mir den DEBUGGER des Doofimat ST (Name vom Autor geändert)'. Ich habe mir genannten "Debugger" mal eingehend angesehen; er ist äußerst langsam beim Bildschirmaufbau, belegt wegen GEM-Steuerung zwei GEM-Windows, erzeugt beim Bedienen öfter Bömbchen und ist auch etwas rar in der Funktionsauswahl (für meinen Geschmack). Vorteil: Er arbeitet mit Symbolen. Der Werbe-Gag von wegen "68020 Single-Step-Emulation": Die 68020 hat einen Tracemodus, in dem nur bei Sprunganweisungen (BRA, JMP, JSR, Bcc, usw.) eine Trace-Exception ausgelöst wird. Und diese Funktion dient beim "Doofimat" dazu, nicht alle Anweisungen schrittweise ausführen zu müssen, sondern nur eine Anzeige bei den Sprunganweisungen zu erhalten.
Ich habe bis heute nicht herausgefunden, warum dieser Herr sich die Mühe gemacht hat, mir seine Meinung zu verkünden. Aber damit sie würdige Beachtung findet, habe ich sie hier nochmal erwähnt.

Diesen Text werde ich jetzt erstmal zu einem Sezzer bringen. Man hat mir da so einen empfohlen, zwar etwas frech, auch nicht so sehr als Pruufrieder geeignet, aber sonst ziemlich gutmütig ["und nicht so eingebildet wie ich"]. Und außerdem macht er es ehrenamtlich ! Naja, man wird ja sehen...

Eine erfolgreiche Fehlersuche und noch mehr fehlerfreie Programme
wünscht Ihnen Thomas Tempelmann ["deutsch auszusprechen", d.S.]