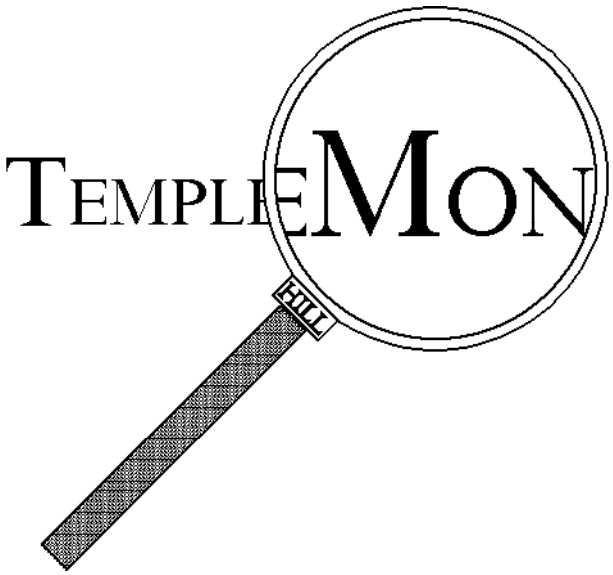

Ein Monitor für Atari-St(e) & Tt

Anleitung zur Version 2.0

Johannes Hill
und
Thomas Tempelmann

10. Dezember 1991

Der Text dieser Anleitung wurde mit LaTeX auf einem Atari Tt Rechner unter TOS formatiert. Eingesetzt wurde eine Shareware TeX-Implementierung von Stefan Lindner. Die Druckvorlage wurde über einen Atari Slm 804 Laserdrucker mit 300 DPI auf DIN A4 ausgedruckt.

Johannes Hill
Alicenstraße 30
6100 Darmstadt

Thomas Tempelmann
Nordendstraße 64
8000 München 40

1. Korrektur der 4. Auflage, 12.02.1992

Haftungsausschluß:

Es wurden alle erdenklichen Maßnahmen getroffen, um die Funktionsfähigkeit des Programmes und die Vollständigkeit und Richtigkeit der Dokumentation zu gewährleisten. Dennoch wird für keinerlei Schäden gehaftet, die sich aus der Benutzung des Programmes oder der Dokumentation ergeben sollten. Für den Inhalt der Dokumentation und die Funktionsfähigkeit des Programmes wird keinerlei Gewährleistung gegeben.

Insbesondere kann keine Gewähr für den Betrieb TempleMons mit dem 68040 Prozessor übernommen werden, da dies mangels geeigneten Gerätes nicht ausgetestet werden konnte.

Fehlermeldungen zu Programm oder Dokumentation werden dankbar entgegengenommen.

Danksagung:

Unser besonderer Dank geht an die Helfer für die tatkräftige Unterstützung beim Erstellen und Testen des Programmes, sowie bei der Aufarbeitung der Programmdokumentation.

Herrn Stefan Wolf danken wir für das Installationsprogramm und Konfigurations - CPX.

Herrn Steffen Steinhäuser danken wir für die tatkräftige Unterstützung bei der Gestaltung der Programmdokumentation.

Herrn Alexander Herzlinger und Herrn Andreas Borchard danken wir für das ausgiebige Testen.

Herrn Karsten Isakovic danken wir für die gute Zusammenarbeit, und dem schönen Programm SysMon, das TempleMon ideal ergänzt.

Inhaltsverzeichnis

Kapitel 1

Vorwort

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 uns, da die Ausarbeitung von Programm und Anleitung viel Zeit und Arbeit gekostet hat. Sollten Ihnen Fehler in der Anleitung auffallen, die uns selbst nach mehrmaligen Korrekturlesen noch immer 'durch die Lappen' gegangen sind, so bitten wir um Nachsicht.

Wir hoffen, daß diese Anleitung hilfreich für Sie sein wird. Wir haben uns bemüht, mit vielen Beispielen auch Anwendern, die unerfahren in der Anwendung von Debuggern sind, 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, empfehlen wir Ihnen zusätzlich etwas Lektüre, z.B. das Atari ST Profibuch von Hans-Dieter Jankowski, Julian Reschke und Dietmar Rabich, erschienen im SYBEX Verlag, und vor allem ein Prozessorbuch zum 68000, z.B. Teil 1 des Bandes "M68000 Familie" vom Tewi-Verlag, oder zum 68030 z.B. "68030 Assembly Language Reference" von Steve Williams, erschienen Addison Wesely Verlag. Falls Sie Interesse an der Original Motorola Literatur haben, so wenden Sie sich an Ihren Buchhändler, da die Original Literatur auch vom Verlag Prentice Hall, allerdings in englischer Sprache, vertrieben wird. Zum Beispiel das "MC68030 Enhanced 32-Bit Microprocessor User's Manual, Third Edition", ISBN 0-13-566423-3, oder das "MC68881/882 Floating-Point Coprocessor User's Manual, Second Edition", ISBN 0-13-567009-8. Sollten Sie gezwungen sein, die Prozessor Befehle von Hand zu kodieren, weil Ihr

Entwicklungssystem die von Ihnen eingesetzten Befehle nicht kodieren kann, werden Sie auch das "M68000 Programmers Reference Manual" brauchen.

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. Allerdings sollten Sie sich schon mit den typischen Fehlermeldungen des Prozessors, den sogenannten Exceptions vertraut machen, und auch wissen, welche Ursachen zu solchen Fehlermeldungen führen. Im Anhang finden Sie eine Liste aller von TempleMon dargestellten Fehlermeldungen. Diese kann Ihnen vielleicht als erstes weiterhelfen.

In dieser Anleitung sind englische Fachausdrücke und deutsche Worte nicht besonders unterschieden, da in der Computerbranche mit den englischen Fachausdrücke genauso selbstverständlich umgegangen wird, wie mit deutschen Fachbegriffen. Oftmals gibt es für einige Fachausdrücke kein entsprechend gutes Pendant in Deutsch.

Übrigens : Erste Versionen dieses Monitors sind auf einem GEPARD-Computer mit Hilfe des Modula-2 / Assembler Entwicklungssystems programmiert und dann auf den Atari übertragen worden. Die neueren Versionen des Monitors wurden direkt auf dem Atari in MEGAMAX MODULA-2 mit dem integrierten Inline Assembler geschrieben.

Kapitel 2

Die Mikroprozessoren der 68000 Familie

Hier erfahren sie nicht, wie die Motorola Prozessoren der 68000 Familie funktionieren, und wie man sie programmiert. Dieser Abschnitt bietet nur einen kleinen Überblick über die einzelnen Register der Prozessoren und deren Bezeichnung in TempleMon. Auch die Adressierungsarten der Prozessoren, sowie deren Darstellung durch TempleMon werden kurz gezeigt.

Die nachstehend aufgeführten Prozessoren werden von TempleMon unterschieden. Ein 68881/882 Koprozessor wird jedoch nur dann erkannt, wenn mindestens ein 68020 Prozessor vorhanden ist. Der Koprozessor ist bei dem 68040 Prozessor schon mit auf dem Chip integriert, und kein extra Baustein mehr. In 68020 und 68030 Systemen wird nur dann ein 68881/882 Koprozessorregistermodell von TempleMon angeboten, wenn tatsächlich im System auch ein solcher Koprozessor eingebaut ist. In einem System mit 68020 Prozessor ignoriert TempleMon die PMMU 68851, falls sie dort vorhanden sein sollte.

2.1 Das Registermodell des 68000 Prozessors

Die Register des 68000:

D0-D7 Datenregister 0 bis 7

A0-A7 Adreßregister 0 bis 7

CCR Condition Code Register

SR Statusregister, das untere Byte entspricht dem Condition Code Register

PC Programmzählerregister

USP User Stackpointer

SSP Supervisor Stackpointer

Das Condition Code Register wird nur bei der disassemblierten Darstellung von TempleMon genutzt. Bei der Registermodifikation oder Registeranzeige wird es von TempleMon nicht ausgegeben oder beachtet, da der Inhalt vollständig im Statusregister enthalten ist.

Alleine im Statusregister (auch im Condition Code Register) besitzen einzelne Bits eine bestimmte Bedeutung. Sie werden als Kennzeichenbits (Flags) bezeichnet. Diese Kennzeichenbits werden in TempleMon in der Motorola üblichen Schreibweise abgekürzt:

C Carry

V Overflow

Z Zero

N Negative

X Extend

I Interrupt Priority Mask

S Supervisor/User State

T Trace Enable
Dieses Bit wird von TempleMon gesetzt, wenn Sie ein Programm tracen.

In TempleMon wird eines dieser Bits entweder durch die Kombination "Registerbezeichnung.Bitbezeichnung" (z.B.: `SR.C=0 SR.Z=0`) oder durch "Registerbezeichnung:Bitbezeichnung" (z.B.: `SR:C=0 Z=0`) dargestellt. Beide Darstellungsvarianten unterscheiden sich dadurch, daß im ersten Fall nur die nächste Bitbezeichnung dem angewählten Register zugeordnet sind, im zweiten Falle alle weiteren Bitbezeichnungen bis zum Ende der Zeile.

2.2 Das Registermodell des 68010 Prozessors

Die Register des 68010:

D0-D7 Datenregister 0 bis 7

A0-A7 Adreßregister 0 bis 7

CCR Condition Code Register

SR Statusregister, das untere Byte entspricht dem Condition Code Register

PC Programmzählerregister

USP User Stackpointer

SSP Supervisor Stackpointer

SFC Sourcefunctioncoderegister

DFC Destinationfunctioncoderegister

VBR Vektorbasisregister

Die Bitbezeichungen des Statusregisters:

C Carry

V Overflow

Z Zero

N Negative

X Extend

I Interrupt Priority Mask

S Supervisor/User State

T Trace Enable

2.3 Die Adressierungsarten der Prozessoren 68000 und 68010

Die Prozessoren 68000 und 68010 bieten insgesamt 12 verschiedene Adressierungsarten an, die von TempleMon in der alten Motorola Schreibweise dargestellt werden, wenn der Disassembler für 68000 oder 68010 Prozessoren disassembliert.

Dn Datenregister direkt

An Adreßregister direkt

(An)	Adreßregister indirekt
(An)+	Adreßregister indirekt mit "postincrement"
-(An)	Adreßregister indirekt mit "predecrement"
d16(An)	Adreßregister indirekt mit 16-Bit Distanzwert
d8(An,Xn)	Adreßregister indirekt mit Index und 8-Bit Distanzwert
d16(PC)	Programmzählerregister indirekt mit 16-Bit Distanzwert
d8(PC,Xn)	Programmzählerregister indirekt mit Index und 8-Bit Distanzwert
(xxxx).W	Absolut kurz
(xxxx).L	Absolut lang
#data	Unmittelbar

Legende:

Dn	eines der Datenregister D0-D7
An	eines der Adreßregister A0-A7
Xn	eines der Datenregister D0-D7 oder eines der Adreßregister A0-A7
PC	das Programmzählerregister
d8	eine 8 Bit große Zahl, die als Distanzwert (oder engl. displacement) bezeichnet wird
d16	eine 16 Bit große Zahl, die als Distanzwert (oder engl. displacement) bezeichnet wird

2.4 Das Registermodell des 68020 Prozessors

Die Register des 68020:

D0-D7	Datenregister 0 bis 7
A0-A7	Adreßregister 0 bis 7
CCR	Condition Code Register
SR	Statusregister, das untere Byte entspricht dem Condition Code Register

PC Programmzählerregister

USP User Stackpointer

MSP Master Stackpointer

ISP Interrupt Stackpointer

SFC Sourcefunctioncoderegister

DFC Destinationfunctioncoderegister

VBR Vektorbasisregister

CACR Cache Control Register

CAAR Cache Address Register

Die Bitbezeichungen des Statusregisters:

C Carry

V Overflow

Z Zero

N Negative

X Extend

I Interrupt Priority Mask

M Master Mode

S Supervisor/User State

T0 Trace On Flow Enable

T1 Trace Enable

Die Bitbezeichnungen des Cache Control Registers:

E Enable Cache

F Freeze Cache

CE Clear Entry

C Clear

2.5 Das Registermodell des 68030 Prozessors

Die Register des 68030:

D0-D7 Datenregister 0 bis 7

A0-A7 Adreßregister 0 bis 7

CCR Condition Code Register

SR Statusregister, das untere Byte entspricht dem Condition Code Register

PC Programmzählerregister

USP User Stackpointer

MSP Master Stackpointer

ISP Interrupt Stackpointer

SFC Sourcefunctioncoderegister

DFC Destinationfunctioncoderegister

VBR Vektorbasisregister

CACR Cache Control Register

CAAR Cache Address Register

CRP CPU Root Pointer Register

SRP Supervisor Root Pointer Register

TT0 Transparent Translation Register 0

TT1 Transparent Translation Register 1

TC Translation Control Register

PSR PMMU Status Register

Die Bitbezeichungen des Statusregisters:

C Carry

V Overflow

Z Zero

N Negative

X Extend

I Interrupt Priority Mask

M Master Mode

S Supervisor/User State

T0 Trace On Flow Enable

T1 Trace Enable

Die Bitbezeichnungen des Cache Control Registers:

EI Enable Instruction Cache

FI Freeze Instruction Cache

CEI Clear Instruction Cache Entry

CI Clear Instruction Cache

IBE Instruction Burst Enable

ED Enable Data Cache

FD Freeze Data Cache

CED Clear Data Cache Entry

CD Clear Data Cache

DBE Data Burst Enable

WA Write Allocate

Die Bitbezeichnungen der Transparent Translation Registers:

LAB Logical Address Base

LAM Logical Address Mask

E Enable

CI Cache Inhibit

R/W Read / Write

RWM Read-Write Mask

FCB Function Code Base

FCM Function Code Mask

Die Bitbezeichnung des Translation Control Registers:

E Enable

SRE Supervisor Root Pointer Enable

FCL Function Code Lookup

PS Page Size

IS Initial Shift

TIA Table Index A

TIB Table Index B

TIC Table Index C

TID Table Index D

Die Bitbezeichnung des PMMU Statusregisters:

B Bus Error

L Limit Violation

S Supervisor Only

W Write Protect

I Invalid

M Modified

T Transparent Access

Number of Levels

Die Bitbezeichnung der Root Pointer Register:

L/U	Lower / Upper
LIMIT	Limit
DT	Descriptor Type
ADDRESS	Table Address

2.6 Das Registermodell des 68040 Prozessors

Die Register des 68040:

D0-D7	Datenregister 0 bis 7
A0-A7	Adreßregister 0 bis 7
CCR	Condition Code Register
SR	Statusregister, das untere Byte entspricht dem Condition Code Register
PC	Programmzählerregister
USP	User Stackpointer
MSP	Master Stackpointer
ISP	Interrupt Stackpointer
SFC	Sourcefunctioncoderegister
DFC	Destinationfunctioncoderegister
VBR	Vektorbasisregister
CACR	Cache Control Register
CRP	CPU Root Pointer Register
SRP	Supervisor Root Pointer Register
ITT0	Instruction Transparent Translation Register 0
ITT1	Instruction Transparent Translation Register 1
DTT0	Data Transparent Translation Register 0

DTT1 Data Transparent Translation Register 1

TC Translation Control Register

PSR PMMU Status Register

FP0-FP7 Floatingpoint Data Register 0 bis 7

FPCR Floatingpoint Control Register

FPSR Floatingpoint Status Register

FPIAR Floatingpoint Instruction Address Register

Die Bitbezeichungen des Statusregisters:

C Carry

V Overflow

Z Zero

N Negative

X Extend

I Interrupt Priority Mask

M Master Mode

S Supervisor/User State

T0 Trace On Flow Enable

T1 Trace Enable

Die Bitbezeichnungen des Cache Control Registers:

DE Enable Data Cache

IE Enable Instruction Cache

Die Bitbezeichnung des Translation Control Registers:

E Enable Translations

P Page Size

Die Bitbezeichnungen der Transparent Translation Register:

LAB Logical Address Base

LAM Logical Address Mask

E Enable

S Supervisor/User Mode

U1 User Page Attribute 1

U2 User Page Attribute 2

CM Cache Mode

W Write Protect

Die Bitbezeichnungen des PMMU Status Registers:

ADDRESS Physical Address

B Bus Error

G Global

U1 User Page Attribute 1

U2 User Page Attribute 2

S Supervisor Protection

CM Cache Mode

M Modified

W Write Protect

T Transparent Translation Register Hit

R Resident

Der 68040 enthält einen Großteil der 68881/882 Befehle und alle ihre Register. Lediglich das "FPIAR" Register kann nur noch gelesen werden. Alle nicht implementierten Fließkomma Befehle stellt Motorola in einer Softwaresimulation zur Verfügung. Leider kennt der Prozessor den Datentyp "Packed Decimal", ".P" nicht mehr, so daß TempleMon nicht mehr die ingeneurwissenschaftliche Darstelllung der Fließkommazahlen bieten kann. Beim Auftreten eines Software emulierten Befehls springt TempleMon automatisch in der XBRA Kette weiter zum Softwareemulator. Die einzelnen Bitbezeichnungen der Register "FPCR und FPSR" entnehmen Sie bitte dem Absatz über die Koprozessoren 68881/68882.

2.7 Die Adressierungsarten der Prozessoren 68020 bis 68040

Die Prozessoren 68020, 68030 und 68040 bieten insgesamt 18 verschiedene Adressierungsarten an, die von TEMPLEMON in der neuen Motorola Schreibweise dargestellt werden, wenn der Disassembler für 68020, 68030 oder 68040 Prozessoren disassembliert. Ein unterdrücktes Basisregister wird durch "ZBR" (Zero Base Register), ein unterdrücktes Programmzählerregister wird durch "ZPC" (Zero Program Counter) dargestellt. Unterdrückte Basisdistanzwerte (Basedisplacement), äußere Distanzwerte (Outer Displacement) und Indexregister (Indexregister) werden bei den entsprechenden Adressierungsarten einfach weggelassen.

Dn	Datenregister direkt
An	Addreßregister direkt
(An)	Addreßregister indirekt
(An)+	Addreßregister indirekt mit "postincrement"
-(An)	Addreßregister indirekt mit "predecrement"
(d16,An)	Addreßregister indirekt mit 16-Bit Distanzwert
(d8,An,Xn)	Addreßregister indirekt mit Index und 8-Bit Distanzwert
(bd,An,Xn)	Addreßregister indirekt mit Index und Basisdistanzwert
([bd,An],Xn,od)	Speicher indirekt mit "postindex"
([bd,An,Xn],od)	Speicher indirekt mit "preindex"
(d16,PC)	Programmzählerregister indirekt mit 16-Bit Distanzwert
(d8,PC,Xn)	Programmzählerregister indirekt mit Index und 8-Bit Distanzwert
(bd,PC,Xn)	Programmzählerregister indirekt mit Index und Basisdistanzwert
([bd,PC],Xn,od)	Programmzählerregister indirekt mit "postindex"

([bd,PC,Xn],od Programmzählerregister indirekt mit "preindex"

(xxxx).W Absolut kurz

(xxxx).L Absolut lang

#data Unmittelbar

Legende:

Dn eines der Datenregister D0-D7

An eines der Adreßregister A0-A7

Xn eines der Datenregister D0-D7 oder eines der Adreßregister A0-A7

PC das Programmzählerregister

d8 eine 8 Bit große Zahl, die als Distanzwert (oder engl. displacement) bezeichnet wird

d16 eine 16 Bit große Zahl, die als Distanzwert (oder engl. displacement) bezeichnet wird

bd eine 8, 16 oder 32 große Zahl, die als Basisdistanzwert (oder engl. base displacement) bezeichnet wird.

od eine 8, 16 oder 32 große Zahl, die als äußerer Distanzwert (oder engl. outer displacement) bezeichnet wird.

Im Gegensatz zum 68000 und 68010 kann bei den Adressierungsarten mit Indexierung eine Indexskalierung mit den Faktoren 2, 4 und 8 vorgenommen werden. Dies wird durch das Multiplikationszeichen mit dem entsprechendem Faktor hinter dem Indexregister kenntlich gemacht.

Die gegenüber dem 68000 hinzugekommenen Adressierungsarten haben die Eigenschaft, daß Teile der Adreßbildung unterdrückt werden können. Die entsprechenden, unterdrückten Register oder Distanzwerte werden bis auf das Basisregister, also dem Adreßregister in diesen Adressierungsarten, nicht ausgegeben. Ist ein unterdrücktes Basisregister ein Adreßregister, so schreibt TEMPLEMON zur Erkennung "ZBR", was soviel wie Zero Base Register heißt, oder bei dem unterdrückten Programmzählerregister "ZPC", was Zero Program Couter heißt.

2.8 Das Registermodell des 68881/882 Fließkommaprozessors

FP0-FP7 Floatingpoint Data Register 0 bis 7

FPCR Floatingpoint Control Register

FPSR Floatingpoint Status Register

FPIAR Floatingpoint Instruction Address Register

Die Bitbezeichnungen des Fließkommakontrollregisters

Das Fließkommakontrollregister teilt sich in die zwei, jeweils 8 Bit großen Teile: Exception Enable und Mode Control. Da beide Teile sich in ihren Bitbezeichnungen unterscheiden, stellt TempleMon diese Teilbezeichnungen nicht vor die Bitbezeichnungen.

- Das Exception Enable Byte:

  BSUN Branch / Set on Unordered
  SNAN Signaling Not A Number
  OPERR Operand Error
  OVFL Overflow
  UNFL Underflow
  DZ Divide by Zero
  INEX2 Inexact Operation
  INEX1 Inexact Decimal Input

- Das Mode Control Byte:

  PREC Rounding Precision
  RND Rounding Mode

Die Bitbezeichnungen des Fließkommastatusregisters

Das Fließkommastatusregister teilt sich in insgesamt vier jeweils 8 Bit große Teile auf, die sich in ihren Bitbezeichnungen teilweise nicht unterscheiden. Daher stellt TempleMon die Teilbezeichnungen des Fließkommastatusregisters vor die Bitbezeichnungen, getrennt durch einen Punkt, und betrachtet sie als Teil der Bitbezeichnungen.

- Das Floating Point Condition Code Byte ("FPCC"):

  FPCC.N Negative
  FPCC.Z Zero
  FPCC.I Infinity
  FPCC.NAN Not A Number or Unordered

- Das Quotient Byte ("QUOT"):

  QUOT.SIGN Sign Of Quotient
  QUOT.QUOT Seven last significant Bits of Quotient

- Das Exception Status Byte ("EXC"):

  EXC.BSUN Branch / Set on Unordered
  EXC.SNAN Signaling Not A Number
  EXC.OPERR Operand Error
  EXC.OVFL Overflow
  EXC.UNFL Underflow
  EXC.DZ Divide by Zero
  EXC.INEX2 Inexact Operation
  EXC.INEX1 Inexact Decimal Input

- Das Accrued Exception Byte ("AEXC"):

  AEXC.IOP Invalid Operation
  AEXC.OVFL Overflow
  AEXC.UNFL Underflow
  AXEC.DZ Divide by Zero
  AEXC.INEX Inexact

Kapitel 3

Allgemeine Arbeitsweise von TEMPLEMON

Sie haben hoffentlich längst TempleMon in den Autoordner Ihrer Bootdiskette oder Ihrer Festplatte kopiert. Beim nächsten Reset, oder nach dem nächsten Einschalten des Rechners, installiert sich TempleMon automatisch. Ist TempleMon mal nicht automatisch installiert worden, kann das immer noch nachgeholt werden, indem man ihn einfach durch Anklicken aktiviert. Die Installation TempleMons kann abgebrochen werden, indem Sie die rechte Shift Taste gedrückt halten.

TempleMon kann dann entweder durch Drücken von CONTROL und HELP 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.). Ein Auftreten eines solchen Fehlers nennt man übrigens eine Exception. Es gibt verschiedene Arten von Exceptions, z.B. den sogenannten Bus Fehler, den Adreß Fehler, die Privilegsverletzung, und andere. Zu jeder Exception gibt es eine eigene Routine, die beim Auftreten einer solchen Exception ausgeführt wird. Diese Routinen können beliebig im Speicher stehen, nur Ihre Adressen werden beim 68000 an festen Speicherstellen vermerkt. Diese nennt man auch Exceptionvektoren. Bei den anderen Prozessoren werden diese Vektoren relativ zum Vektorbasisregister gespeichert, wobei die Distanz der Adresse einer Speicherstelle für einen Vektor gleich der Distanz des entsprechenden Vektors zur Adresse 0 beim 68000 ist. TempleMon geht immer davon aus, daß bei Vorhandensein des Vektorbasisregisters dieses immer auf Null gesetzt ist. Dies ist auch bei allen TOS Versionen der Fall. Seit kurzem gibt es virtuelle Speichersysteme für den Atari TT, die das Vektorbasisregister nutzen. Lassen Sie sich dadurch nicht irritieren. Dies ist völlig in Ordung so. Diese virtuellen Speichersysteme müssen auf jeden Fall immer zu erst im Falle einer Buserrorexception angesprungen werden. Bei allen anderen Exceptions springen diese Speicherverwaltungssysteme sowieso sofort automatisch in die TempleMon Routinen. Zwei dieser virtuellen Speicherverwaltungssysteme seien hier beim Namen genannt: „VRAM“ und „OUTSIDE“. TempleMon muß zum ordentlichen Betrieb mit dieser Art von virtuellen Speichersystemen nicht anders konfiguriert

werden. Bei einem anderen System von virtuellen Speichersystemen, die das Vektorbasisregister nicht verändern, also unverändert auf Null lassen, muß TempleMon umkonfiguriert werden. Das können Sie ein wenig später nachlesen.

Der Sinn und Zweck TempleMons ist, im Falle einer fehlerauslösenden Exception das aktuelle Programm anzuhalten, und eine genaue Analyse des Fehlers mit seiner Ursache zu bieten. Dazu kann man sich Registerinhalte und Speicherauszüge in hexadezimaler oder disassemblierter Form anschauen. Ebenso kann man Registerinhalte oder einzelne Speicherzellen ändern. TempleMon bietet Ihnen dazu einen komfortablen Editor an, der ähnlich einem Programmeditor ist. Man kann mit dem Cursor rauf und runter fahren, dabei Zeichen auf dem Bildschirm einfach überschreiben, und die Änderungen durch Drücken der Taste RETURN einfach übernehmen. Je nach dem, wie Sie den Editor konfiguriert haben, können Sie eine gewisse Anzahl an Zeilen, die durch Scrolling vom oberen Bildschirmrand verschwunden sind, wieder zurückholen und anschauen. Von allen Bildschirmausgaben können Sie sich ein Protokoll in eine Datei oder auf ihren Drucker anfertigen lassen, und so Ausgaben dauerhaft festhalten. Bei dem ganzen Komfort sollten Sie jedoch nicht vergessen, daß TempleMon ein kommandoorientiert arbeitendes Utility ist. In der Regel wird nach dem Bereitschaftszeichen, dem Prompt, dargestellt durch ein Ausrufezeichen, eine Kommandoeingabe von TempleMon erwartet. Die Kommandos bestehen meistens aus nur einem oder zwei Buchstaben und einigen Argumenten, die durch Leerzeichen voneinander getrennt werden.

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
!R SR :T=0 S=0 I=3 X=0 N=1 Z=0 V=0 C=0
```

Die erste Zeile zeigt immer an, aus welchem Grund TempleMon sich meldet, die anderen zeigen 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. Die Anzahl der Register, die TempleMon anzeigt, können Sie mit Hilfe der R-Funktion ändern. Standardmäßig zeigt TempleMon alle 68000 Register an, jedoch nicht die zusätzlichen Register der Prozessoren der 68000 Familie. TempleMon rettet alle Register des Mikroprozessors, bevor er sie für seine eigenen Funktionen mißbraucht. Nun können Sie diese 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 fortfahrenzulassen. Dieser Monitor verhält sich somit so, als ob er für andere Programme gar nicht vorhanden wäre. Er ist fast vollständig transparent. Der Monitor sorgt dafür, daß im gesamten Rechner so gut wie nichts (außer dem für sich selbst beanspruchten Platz) verändert wird, wenn er einmal aufgerufen wurde. Vollkommen transparent 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 vorgebeugt, indem sie während des Diskettenzugriffs, den 68000 per Software in einen Modus versetzen, in dem er sich durch keinen Interrupt, also auch nicht durch CONTROL 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 des Fehlers, ungehindert weiterlaufen läßt. Um jedoch trotzdem TEMPLEMON mit solchen, fehlerhaften Programmen ohne ständige, nervende Fehlermeldung einsetzen zu können, bietet TEMPLEMON Ihnen die Möglichkeit an, die Fehlermeldung einfach zu ignorieren. Doch schließen Sie nicht daraus, daß Programm wäre fehlerfrei, da TEMPLEMON sich nicht mehr meldet. Ein Programm, daß eine 'Division durch Null' Fehlermeldung erzeugt, ist fehlerhaft!

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 CONTROL 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 Symbolen 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.
- Einem die Arbeit abnehmen, "ausführliche" Anleitungen zu schreiben.
- Goldene Eier legen.

Kapitel 4

Die Funktionen von TEMPLEMON im Einzelnen

4.1 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 bestätigen (im Gegensatz zur Kommandoeingabe beim Tracen, wo nur eine einzelne Taste erwartet wird). Die Kommandoeingabe läßt sich mit den üblichen Korrekturtasten bequem editieren. Auch lassen sich die letzten Kommandos, die eingetippt wurden, wieder zurückholen. Die Anzahl der Kommandos, die wieder zurückgeholt werden können, können Sie einstellen. Entnehmen Sie bitte die einzelnen Tastenfunktionen zum Editieren der folgenden Tabelle:

Tastenkombination		Bedeutung
ESC		Eingabezeile löschen
SHIFT	HOME	Eingabezeile ab Cursorposition löschen
CONTROL	HOME	Letzte Zeile am Bildschirm anspringen
HOME		Erste Zeile am Bildschirm anspringen
SHIFT	←	Start der Eingabezeile anspringen

Fortsetzung auf der nächsten Seite

Fortsetzung von vorheriger Seite

Tastenkombination	Bedeutung
SHIFT →	Ende der Eingabezeie anspringen
←	Cursor links bewegen
→	Cursor rechts bewegen
DELETE	Zeichen unter Cursor löschen
BACKSPACE	Zeichen links neben Cursor löschen
CONTROL ←	Ein Wort links springen
CONTROL →	Ein Wort rechts springen
SHIFT CONTROL ←	Wortanfang anspringen
SHIFT CONTROL →	Wortende anspringen
INSERT	Zwischen Einfüge- und Überschreibmodus toggeln. Die Cursordarstellung ändert sich entsprechend: Ausgefüllter Cursor beim Einfügen, Unterstrichcursor beim Überschreiben.
↑	Cursor eine Zeile nach oben bewegen.
↓	Cursor eine Zeile nach unten bewegen.
CONTROL ↑	Eine Seite nach oben blättern.
CONTROL ↓	Eine Seite nach unten blättern.
SHIFT CONTROL ↑	Bildschirmpufferanfang anspringen
SHIFT CONTROL ↓	Bildschirmpufferende anspringen
SHIFT ↑	Letzte eingegebene Kommandozeile zurückholen
SHIFT ↓	Nachfolgende Kommandozeile zurückholen
UNDO	Die durch die Suche der letzten Kommandozeilen oder durch Löschen der Eingabezeile zerstörte Zeile wieder herstellen.

Im Folgenden werden Eingaben, an deren Ende die RETURN Taste gedrückt werden muß, mit einfachen Anführungszeichen gekennzeichnet. Groß-/Kleinschreibung ist gleichwertig, nur bei Angabe eines Füll- oder Suchstrings (s.u.) ist sie zu beachten.

Werden Zahlenangaben erwartet, müssen sie normalerweise hexadezimal angegeben werden. Jedoch ist es möglich, 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 können verschieden formuliert werden:

- Direkte Zahlenangabe, z.B "`42E`".
- Verwendung eines Registerinhalts (der geretteten CPU-Register), z.B. "`RD7`", "`RA2`" oder "`RPC`". Sie müssen also vor dem Registernamen ein "`R`" plazieren. Die folgenden Register sind bei der registerrelativen Eingabeart zugelassen:
  - Alle Datenregister
  - Alle Adreßregister
  - Alle Stapelzeiger
  - Der Programmzähler
  - Alle Rootpointer
  - Das Vektorbasisregister
  - Das Cache Address Register
  - Das Fließkomma Befehlsadreßregister
- mit Konstant-Offset und direkter Zahlenangabe, z.B. "`012`". "`O`" bezeichnet die Adreßoffsetvariable. Man kann sie z.B. mit der Startadresse eines geladenen Programmes belegen, und Adreßangaben relativ zu dieser Startadresse angeben, um Tipparbeit zu sparen.
- Ausdrucksbildung mit '`+`' und '`-`', z.B. "`RPC-20`" oder "`RA0+RD2+4`".
  Achtung:

  Durch schlampige Programmierung darf vor und nach dem Operatorzeichen kein Leerzeichen stehen!
- mit "`P()`". `P` steht für Pointer, d.h. die gesamte nachfolgende Adreßangabe inklusive Rechenanweisungen wird als Zeiger interpretiert. Will man Teile davon ausschließen, so muß richtig geklammert werden! `P` darf mehrfach in einem Ausdruck auftreten. Es wird dann rekursiv geparst.
  Bsp.:

`!D p(84)`	Liefert disassemblierten Hexdump des ersten TRAP1-Handlers.
`!D p(p(84)-4)`	Falls es sich um eine Kette von Handlern handeln sollte, die über die XBRA Struktur verkettet sind, so könnte man den zweiten Handler über diese Konstruktion erreichen.
`!D p(p(p(84)-4)-4)`	So könnte man einen dritten Handler erreichen.

usw.

Dabei läßt sich der Kommandozeileneditor mit dem Eingabenzeilenspeicher ganz praktisch nutzen.

Es ist erlaubt, statt "`p(84)`" nur "`p84`" zu schreiben, um Tipparbeit zu sparen. Jedoch entspricht dies nicht der definierten Syntax. Sollten Sie mehr als nur eine "`p`"-Ebene verwenden, müssen Sie sich unbedingt an korrekte und vollständige Syntax halten!

Adreßpaare (Anfangs- und Endadresse) können oft abgekürzt werden. Als Beispiel hier das Disassemblierkommando im normalen Format:

"`D 50456 50468`".

- Wenn (nach der Syntax) hinter dem Adreßpaar keine weiteren Parameter erwartet werden, sind folgende zwei Abkürzungen möglich: 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 größtmögliche Endadresse verwendet werden (`$FFFFFF` bzw. `$FFFFFFFF` beim 68020 / 68030 / 68040), kann sie ebenfalls wegfallen und dafür 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 "`X`" direkt vorangestellt (z.B. "`D 45552X20`" 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`" oder "`Z`" bestimmt werden (Beispiel: "`D L8`" disassembliert weitere 8 Zeilen).

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

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 TempleMonBildschirms zurückgeschaltet werden kann.

4.2 Die Funktionen

Nun bekommen Sie eine Übersicht ["mehr eine Unübersicht !" der Setzer] über die einzelnen Kommandos sowie deren Syntax präsentiert.

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. Setzer]. Dort ist alles nochmal etwas anders (vielleicht auch verständlicher) ["Nee, zwar anders, aber auch so ein Kauderwelsch" d.S.] formuliert.

4.2.1 Speicherinhalt hexadezimal anzeigen

"M <a> <e>"	zeigt den gewünschten Speicherbereich byteweise an. Statt "M" kann auch "MB" eingegeben werden.
"M"	zeigt ab letzter Anfangsadresse nochmals an (die Endadresse wird als $FFFFFF bzw. $FFFFFFFF beim 68020 / 68030 / 68040 angenommen!). Auch hier kann statt dem "M" ein "MB" eingegeben werden.
"MW <a> <e>"	zeigt den gewünschten Speicherbereich wortweise an.
"MW"	zeigt ab letzter Anfangsadresse nochmals wortweise an.
"ML <a> <e>"	zeigt den gewünschten Speicherbereich langwortweise an.
"ML"	zeigt ab letzter Anfangsadresse nochmals langwortweise an.

4.2.2 Speicherinhalt ändern

":<a> <w1> <w2>..."	ab Anfangsadresse können bis zu 16 Bytes abgespeichert werden.
":w<a> <w1> <w2>..."	ab Anfangsadresse können bis zu 8 Words abgespeichert werden.
":l<a> <w1> <w2>..."	ab Anfangsadresse können bis zu 4 Longwords abgespeichert werden.

4.2.3 Disassemblieren

"D <a> <e>" Disassembliert den gewünschten Speicherbereich.

"D" zeigt ab letzter Anfangsadresse nochmals an (die Endadresse wird als $FFFFFF bzw. $FFFFFFFF beim 68020/68030/68040 angenommen!).

"D:< x **>"** Wählt für den Disassembler die gewünschte CPU aus. Dabei ist statt "x" entweder "0" für 68000, "1" für 68010, "2" für 68020, "3" für 68030 oder "4" für 68040 anzugeben.

4.2.4 Speicherinhalt als Character anzeigen

"I <a> <e>" Zeigt den gewünschten Speicherbereich in ASCII-Form.

"I" zeigt ab letzter Anfangsadresse nochmals an (die Endadresse wird als $FFFFFF bzw. $FFFFFFFF beim 68020 / 68030 / 68040 angenommen!).

4.2.5 ASCII-Zeichen in den Speicher eingeben

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

4.2.6 Adreßoffsetvariable

"O <x>" Die Variable "O" wird auf den Wert <x> gesetzt.

"O" Zeigt aktuellen Wert der Variable an.

4.2.7 Speicher kopieren (Copy)

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

4.2.8 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.

4.2.9 Speicher mit Konstanten füllen (Fill)

"**F** <**a**> <**e**> '<**Zeichenkette**>' "	Füllt den Speicherbereich byteweise mit <Zeichenkette>.
"**F** <**a**> <**e**> <**b1**> <**b2**>..."	Füllt den Speicherbereich byteweise mit den angegebenen Bytes. Die mit einem Fragezeichen angegebenen Zeichen bzw. Bytes werden nicht im Speicherbereich ersetzt.

4.2.10 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.

4.2.11 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 Word), BEV (Bus Error Vector), AEV (Adress Error Vector), sowie gegebenfalls die zusätzlichen Register des Prozessors (siehe Registermodelle).

"R <reg1>.<bit1> = <x> " Setzt ein mit Namen erreichtbares Bit eines Registers auf den Wert <x>. Danach kann in der selben Eingabe ein anderes Bit weiterer anderer Register gesetzt werden. Dazu muß wieder zuerst der Registername, dann das Bit und zuletzt der Wert angegeben werden. Diese Eingabeform ist für die folgenden Register definiert: SR, CACR, TT0, TT1, ITT0, ITT1, DTT0, DTT1, TC, PSR, CRP, URP, SRP, FPCR, FPSR. Bei den Registern FP0 - FP7 wird mit .E die ingeneurwissenschaftliche Darstellung kenntlich gemacht.

"R <reg1>:<bit1> = <x> " Setzt ein mit Namen erreichtbares Bit eines Registers auf den Wert <x>. Danach kann in der selben Eingabe ein anderes Bit des gleichen Registers gesetzt werden. Dazu darf dann nur noch das Bit mit dem zu setzenden Wert angegeben werden. Für die definierten Register der Eingabeform gilt entsprechendes wie bei "R <reg1>.<bit>".

"R:" Zeigt Registerstandardauswahl an.

"R: <reg1> <reg2>" Wählt neue Registerstandardauswahl.

"R:+" Wählt alle Register.

"R:-" Wählt kein Register.

"R:+ <reg>" Addiert <reg> zur Registerstandardauswahl.

"R:- <reg>" Subtrahiert <reg> von der Registerstandardauswahl.

"R: <x>" Wählt alle Register eines bestimmten CPU-Types aus. Dabei ist statt "x" entweder "0" für 68000, "1" für 68010, "2" für 68020, "3" für 68030 oder "4" für 68040 anzugeben. Alle vorherigen Registerauswahleinstellungen werden damit ersetzt.

"R:8" Zusätzlich zu den in der Registerauswahl vorhandenen Registern werden die Register des 68881/882 Koprozessors mit angezeigt. Die Registerstandardauswahl wird bei Eingabe von "R" verwandt (s.o.). Außerdem bestimmt sie, welche Register beim Tracing angezeigt werden sollen. Eine Registerauswahl, die jedoch in Teilen in der Hardware nicht vorhanden ist, wird nur soweit angezeigt, soweit sie in der Hardware vorhanden ist.

"R:B-" oder "R:." Schaltet auf die kurze Darstellungsart aller Register.

"R:B+" oder "R:*" Schaltet auf die Darstellungsart, bei der die Register, die in verschiedene Teile aufgeteilt sind, diese mit deren Bitbezeichnungen ausgegeben werden.

"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.

"RW" (Register Wahl). Schaltet auf die Register der Registerauswahl beim Anzeigen des Exceptiontyps bei Auftreten einer Exception.

"RD" (Register Default). Schaltet auf die Default Registeranzeige beim Anzeigen des Exceptiontyps bei Auftreten einer Exception (PC, Stapelpointer und SR).

4.2.12 Dezimal nach Hexadezimal umwandeln

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

4.2.13 Rechnen

"? <s> <o> <d>" Verknüpft die Zahlen <s> und <d> mit dem Operator <o> und zeigt das Ergebnis an. <o>: +, -, *, /, O (binäres Oder), A (binäres Und) und E (binäres 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.

4.2.14 Augenblicklich unterbrochenes Programm terminieren

"Q" Führt einen GEMDOS PTERM(-1) 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. Wenn Sie in den Monitor gelangten, während gerade eine GEMDOS Funktion ausgeführt wurde, führt das prompte Terminieren leider oft zu weiteren Fehlern, meist ist dann der Rechner neu zu booten – also Vorsicht. 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.

4.2.15 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 CONTROL HELP in den Monitor gelangen, ist dies aber als Interrupt anzusehen. Fehler können dann auftreten, wenn während der Abarbeitung einer Dateifunktion der Monitor aufgerufen wurde, und nun diese Dateifunktion nochmals ausgeführt 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 vorherige 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 vorherige 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 vorherigen drei Ladefunktionen, es wird jedoch nicht vom Anfang der Datei an geladen, sondern erst ab dem <b>-ten Byte der Datei.

Bei allen Ladefunktionen werden automatisch drei interne Variablen TempleMons gesetzt, die die Ladeadresse, die erste unbenutzte Adresse und die Länge des geladenen Speicherbereiches aufnehmen. Dabei steht "S" für Ladeadresse ("S"tartadresse), "N" für die erste unbenutzte Adresse ("N"ächste freie Adresse) und "L" für die Länge des Speicherbereiches ("L"änge). Die Verwendung dieser drei Variablen unterscheidet sich von der, der Offsetvariablen. Zum einen werden sie nur von der Ladeoperation gesetzt, zum anderen müssen bei einer Ausdruckbildung mit diesen Variablen die +/- Operatoren mitgesetzt werden.

"S <dateiname>, <a> <e>"	Speichert byteweise den Bereich von <a> bis <e>-1 in die Datei mit dem Namen <dateiname>. Es darf jedoch keine Datei mit dem Namen <dateiname> existieren.
"S <dateiname>, @<b>, <a> <e>"	Wie vorherige Funktion, jedoch wird eine bereits bestehende Datei ab der <b>-ten Byteposition überschrieben.
"P"	Löscht Bildschirm (gibt Formfeed aus auf Datei bzw. Drucker).
"P <dateiname>"	Öffnet eine Protokolldatei mit dem Namen <dateiname> (wie auch sonst). Alle Ausgaben gehen dann außer auf den Bildschirm auch in die Datei. Die Datei kann sowohl auf Diskette, als auch auf einen Peripheriegerät eröffnet werden. Beispiel: "P PRN:" öffnet die Datei auf den Drucker.
"PC"	Schließt die Protokolldatei. (Wird ebenso beim "Q"-Kommando ausgeführt.)
"PF"	Fügt das Zeichen FormFeed in das Protokollfile ein. Damit kann bei seitenorientierten Druckern wie Laserdruckern die gepufferten Zeichen auf das Blatt ausgedruckt werden. Das Protokollfile wird dabei nicht geschlossen.

4.2.16 TempleMon -Vektoren

"VI" Vector Init. Restauriert alle TempleMon - Exception Vektoren. 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. Sollten gültige XBRA Ketten im Speicher vorliegen, fügt sich TempleMon korrekt aus diesen Ketten aus.

4.2.17 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 unterbrochenen 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 allen 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. Unsere Empfehlung: Auf die Dinger Rücksicht nehmen und erst gar nicht versuchen, Breakpoints auf solche ROM-Adressen zu richten und dann TempleMon 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 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 (weil z.B. 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, nun folgt die Adresse des Breakpoints (ist Sie Null, so ist der Breakpoint unbenutzt), dann ein Initialisierungszählerwert 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.

"B<n> <a> <i>" Setzt zusätzlich den Initialisierungswert und den Zähler auf den Wert <i>.

"B<n> <a> <i> <z>" Setzt den 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.

4.2.18 User-Traceroutine

"BU" Zeigt die Adresse der User-Traceroutine an. Ist sie Null, wird keine User-Traceroutine beim Tracen aufgerufen.

"BU <a>" Setzt die Adresse der User-Traceroutine auf <a>.

"BU 0" Löscht den Aufruf einer User-Traceroutine.

4.2.19 Verlassen von TempleMon, Aufruf von anderen Programmen

"G" Verläßt TempleMon und führt das unterbrochene 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 könnte (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, wobei es zu unerwarteten Fehlern käme.

"G ,<b1>,<b2>..."

"G <a>,<b1>..."

"GS <a>,<b1>..." Es können (von uns) 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.

4.2.20 Tracemodus bestimmen

"T" Zeigt den Tracemodus an (aktiv oder nicht).

"T+" Schaltet den Tracemodus ein und setzt das Trace-Flag im geretteten SR-Register auf Eins.

"T-" Schaltet den Tracemodus ab und setzt das Trace-Flag im geretteten SR-Register zurück (auf Null).

"T0" Wählt für den Tracemodus "Tracing on Flow Control" aus. Diese Option ist erst ab 68020 oder höher möglich.

"T0+" Wählt für den Tracemodus "Tracing on Flow Control" aus. Schaltet den Tracemodus ein und setzt das Trace-Flag im geretteten SR-Register auf Eins. Diese Option ist erst ab 68020 oder höher möglich.

"T1" Wählt den herkömlichen Tracemodus aus. Diese Option ist erst ab 68020 oder höher möglich.

"T1+" Wählt den herkömlichen Tracemodus aus. Schaltet den Tracemodus ein und setzt das Trace-Flag im geretteten SR-Register auf Eins. Diese Option ist erst ab 68020 oder höher möglich.

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 CONTROL 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 das normalerweise vorher gesetzte Trace-Flag wiederhergestellt 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 sagen wir nichts, das können nämlich nur Sie sich selbst verpfuschen. 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. Dies 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, Sie passen am Besten 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-Flag wieder eingeschaltet.

- C (Continued)
  Führt alle folgende Anweisungen mit Tracing und Anzeige aus. Dabei wird solange keine Taste gedrückt wird Befehl für Befehl getraced, als hätte man ständig SPACE gedrückt. Ein Tastendruck führt zum Anhalten.

- 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 die 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. 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.

4.3 Die User-Trace Funktion

Dies ist eine besonders geniale, wenn auch nicht gerade einfach anzuwendende Funktion. Sie haben die Funktion hoffentlich schon (mindestens) einmal benutzt, denn die Beispielprogramme im Anhang machen 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 die Adresse des User-Programms mit der 'BU'-Funktion oder über das Cookie Interface übergeben werden (das Beispielprogramm ruft das Cookie Interface mit der entsprechenden Adresse auf). 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, das Beispielprogramm als Rumpf zu benutzen, das die Definitionen der Register und die Initialisierung der Routine beinhaltet, so daß nur noch die wesentliche Traceroutine jedesmal individuell neu programmiert werden

muß. 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 und 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, A4, das obere Byte von SR und natürlich SSP, bzw. MSP und ISP 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.

Kapitel 5

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, und 1.5 bis 1.20). In diesem Fall empfehlen wir Ihnen, sich über irgend einen Public Domain Service der Zeitschrift ST-Computer eine neue Kopie der Diskette mit dem TempleMon zu besorgen. Sollten Sie allerdings schon die Version 2.0 besitzen, muß der Fehler wohl woanders liegen (ähem) ["←— kein Kommentar vom Setzer" d.S].

5.1 Aufspüren von fehlerhaften Buszugriffen in Programmen

Am Häufigsten wird sich beim Ausprobieren von Programmen TempleMon mit der Meldung BUS ERROR 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 keine Symbole benutzen kann, muß man beginnen, 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 Symbole verarbeiten kann, besitzen (natürlich müßten Sie ihn auch bedienen können), dann nehmen Sie am Besten diesen zur Hilfe. Wenn Sie dann die Stelle im Programmsource gefunden haben, kann 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:

5.1.1 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 sie dann wiederum durch disassemblieren der betreffenden Routine den äquivalenten 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.) :

```
        ; Hauptprogramm:

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

Message1: DC.B  'M','e','l','d','u','n','g','1',13,10,0
Message2: DC.B  13,10,'M','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 Adresse,
                                ; auf die A0 zeigt, steht, nach D0,
                                ; zus. erhoehen des Registers A0 um Eins.
        BEQ     PrEnde          ; Wenn D0 = Null, dann Ende
        ANDI.W  #$00FF,D0       ; Loeschen des oberen Bytes von D0.W
        MOVE.W  D0,-(A7)        ; Laden von D0 auf den Stack
        MOVE.W  #2,-(A7)        ; Laden des Funktionswertes fuer 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 "`RA0`" 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 RPC-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 (inbesondere 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 [0] 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 sich 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.

5.1.2 Die User-Traceroutine

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.

Das Beispiel aus dem File 'TRACE.M' (siehe dazu auch Anhang) 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 jeweils nach 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 vorherigen 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' 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 abgearbeiteten Anweisungen herauszufinden. Leider haben wir nicht daran gedacht, für die Anzeige dieser Werte, die durch das Cookie Jar Interface bereitgestellten Routinen mit diesem User-Traceprogramm zu nutzen. Dafür haben wir, wie in dem folgenden Beispiel, die Adresswerte leicht zugänglich gemacht. Im Beispiel werden die Werte hinter der User-Traceroutine abgelegt (jeweils ein

Longword für Anfang des Puffers, Ende des Puffers und den 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 muß.

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 /* Fuer 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 fuer 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 erhoehen
          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 {
          MOVEA.L $05A0,A3
          TST.L   $05A0
          BEQ     abort
search:   CMPI.L  #'TMon',(A3)
          BEQ     dotmon
          ADDQ.L  #8,A3
          TST.L   (A3)
          BNE     search
abort:    CLR.W   -(SP)
          TRAP    #1
dotmon:   MOVEA.L 4(A3),A3
          MOVEQ   #2,D0
          LEA     _usrTrc(PC),A0
          JSR     (A3)

; 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 );
}
```

5.2 Umgehen von unkritischen 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, so daß 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), oder den TEMPLEMON so zu konfigurieren, daß er die "`Division durch Null`" auch ignoriert.

Mehr Probleme gibt es da schon bei Bus und Adress 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.

Adress Fehler treten beim 68000 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.

Tritt ein Adress Fehler bei einer 68020, 68030 oder 68040 CPU auf, ist eine Korrektur nicht mehr möglich. Dort kann nur dann ein Adress Fehler auftreten, wenn versucht wurde, den nächsten Befehl von einer ungeraden Adresse zu holen. Damit ist also das PC Register nicht mehr in Ordnung. Meistens tritt dieser Fehler beim Überschreiben des Stapels auf. Die eigentliche Fehlerursache i.d.R. ist nur noch mit viel Phantasie und ein bißchen Erfahrung zu finden.

Zurück zum 68000: Ein Beispiel mit einem Adress 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 etwas 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 (ES kann deshalb davon ausgegangen werden, weil es unklug wäre, eine solche RAMDISK-Routine anders zu programmieren, oder ?). Nun werden durch Eingabe von

"`C 24ED2.RA2 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: "`R 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 beim 68000 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.

Kapitel 6

Unterschiede zu den Versionen 1.xx

6.1 Änderungen

Es hat fünf Änderungen im Vergleich zu älteren TempleMon Version gegeben:

- Der Aufruf TempleMons über Tastatur. Bis auf die letzte Version wurde dazu der Hardcopy Vektor verbogen, so daß man mit ALT HELP in den Monitor kam. Dies ist nun anders! Jetzt wird der Interruptvektor der ACIA verbogen, die die Meldungen der Tastatur entgegennimmt. Der Hardcopy Vektor bleibt unangetastet. Da TempleMon nun jede eingetippte Taste auswertet, bot sich die Gelegenheit, weitere Funktionen zu integrieren:
  - Einen Warm- und einen Kaltstart über Tastatur, so wie es das TOS 1.4 schon bietet. Selbstverständlich wird diese Funktion nur bei TOS 1.2 oder noch älter direkt von TempleMon erledigt. Ab TOS 1.4 macht dies das Betriebsystem automatisch, so daß TempleMon dann diese Tastenkombination ignoriert. Der Anwender merkt aber davon nichts. Ein Warmstart wird mit ALT CONTROL DELETE , ein Kaltstart mit zusätzlich gedrückter rechter SHIFT Taste ausgelöst.
  - Im Zusammenhang mit Overscan oder älteren Rechnern trat ab und zu Mal ein seltsamer Effekt auf: das Verschieben des rechten Bildschirmrandes an den linken. Ohne Hilfe eines Programmes ließ dies sich nur noch durch Reset abstellen. Ursache dafür ist ein Wandern des Zeilensynchronsignales, das eigentlich am Ende jeder Zeile erzeugt werden sollte, in die nächste/letzte Zeile hinein. TempleMon bietet nun auch Funktionen, dies zu korrigieren. Dazu bedient sich TempleMon eines kleinen Tricks. Schaltet man nämlich kurzfristig zwischen interner und externer Synchronistation um, wird zwar die Bilddarstellung unterdrückt, die Erzeugung des Zeilen- und des Bildsynchronsignales geht aber noch weiter. Zur Abhilfe bietet TempleMon folgende Manipulationen des Synchronsignales an:

  * Kurzfristiges Umschalten zwischen interner und externer Synchronisation mit ALT CONTROL * auf dem Zehnerblock.
  * Umschalten auf externe Synchronisation mit ALT CONTROL + auf dem Zehnerblock.
  * Umschalten auf interne Synchronisation mit ALT CONTROL − auf dem Zehnerblock.

  Diese Tastenkombinationen sind nur beim Atari ST wirksam.

- Die Anpassung an nicht Standard Bildschirmgrößen. Dies betrifft die AutoSwitch Overscan und HyperScreen User. Bis auf die letzte Version konnte TempleMon nicht mit Overscan betrieben werden. Man mußte immer von Hand den Schalter umlegen. Bei AutoSwitch OverScan können Sie mit den alten TempleMon Version gar nichts mehr anfangen, weil es gar keinen Schalter mehr gibt (wie gemein!). TempleMon versucht nun, bei einer solchen Bildschirmerweiterung wieder auf den originalen ST Schirm zu schalten, solange der TempleMon Schirm zu sehen ist. Dies funktioniert aber nur bei AutoSwitch OverScan. Bei HyperScreen nutzt er einen zweiten Satz an Routinen, die den größeren Bildschirm bedienen, ohne jedoch mehr Zeichen auf dem Bildschirm darzustellen. Die 80x25 Zeichen werden möglichst mittig dargestellt. Es kann aber trotzdem sein, daß die Darstellung gar nicht so sehr mittig ist, weil Sie andere Bildschirmparameter als die Üblichen nutzen. Ein nachträgliches Ändern am TempleMon ist jedoch einfach möglich. Auf dem ST ist die Installationsreihenfolge von TempleMon und dem OverScan Treiber egal, auf dem TT nicht (s.u.).

- Der Fullscreeneditor. Bisher stand nur eine editierbare Kommandozeile zur Verfügung. Nun können Sie mit dem Cursor jeden Punkt auf dem Bildschirm erreichen.

- Die Prozessorenanpassungen an die Prozessoren der 68000 Famile. Die Registermodelle der Prozessoren 68000, 68010, 68020, 68030, 68040, 68881 und 68882 sind nun in TempleMon enthalten. Bisher waren ausschließlich die Register des 68000 zu erreichen.

- Die Programmierschnittstelle im Cookie Jar. Diese Schnittstelle ermöglicht es Ihnen, Teile von TempleMon selbst zu nutzen, oder TempleMon um Eigenschaften durch eigene Programme zu erweitern. Beim Start von TempleMon können eigene Programmroutinen automatisch eingebunden werden. Dazu muß Ihr ausführbares Programm den Namen `TMONHELP.EXE` besitzen. Näheres dazu siehe unter 'Internes für Experten'.

6.2 Die Anpassung an die STE Rechner

Wenn Sie einen STE besitzen, und eine ältere Version von TempleMon ausprobiert haben, werden Sie Sich sicherlich fragen, wozu eine spezielle STE Anpassung, wenn die älteren TempleMon Versionen auch laufen. Dies ist zunächst richtig. Allerdings besitzt der STE ein paar neue Shifter Register, die unter anderem ermöglichen, den Bildschirm in mehreren

Teilen im Speicher verteilt zu halten. TEMPLEMON ist aber auf einen zusammenhängenden Bildschirmspeicher angewiesen. Er löscht beim Aufruf also diese neuen Register, und stellt die alten Registerwerte beim Verlassen wieder her.

6.3 Die Anpassung an die TT Rechner

TEMPLEMON läuft auf dem ATARI TT auch im Fast RAM. Wird ein Farbmonitor am TT betrieben, schaltet TempleMon bei seiner Aktivierung automatisch in die ST Hoch Auflösung, und wieder zurück, wenn das unterbrochene Programm fortgesetzt wird. In TT Hoch bietet TempleMon statt 25 nun 60 Zeilen mit 80 Spalten an, d.h. die Hälfte des Bildschirmes ist ungenutzt. Aber zusammen mit einem anderen, TEMPLEMON ideal ergänzenden Shareware Monitor, dem SYSMON von Karsten Isakovic, ergibt sich ein ganz neues Bild. Da ist dann SYSMON auf der linken Bildschirmhälfte, TEMPLEMON auf der Rechten zu sehen. Dies bietet zwei Vorteile:

- Es wird nur einmal ein Bildspeicher für beide Programme reserviert. Dies bringt die nicht unerhebliche Speicherplatzersparnis von 153600 Bytes.
- Es bietet sich ein vollkommen neuer Überblick über das System an: Zum einen ein vollständiger Überblick über TOS mit seinen Datenstrukturen und allen TOS-Aufrufen, auf der anderen Seite ein vollständiger Überblick über den Prozessor und die Hardware. Inwieweit beide Programme komunizieren und wo Sie den SYSMON beziehen können, erfahren sie in einem eigenen Kapitel.

TEMPLEMON reserviert sich bei der Installation einen Speicherbereich für den Bildschirm mit einem eigenem Aufruf, in dem die Bildschirmdarstellung gespeichert wird. TEMPLEMON selbst sowie der Kommandozeileneingabespeicher und der Pufferspeicher für den Bildschirm werden in einem Speicherblock verwaltet. Je nachdem, wie Sie Ihren TEMPLEMON konfiguriert haben, kann der Speicherplatzbedarf für diese Pufferspeicher erheblich sein. Der TT bietet abhängig von seiner Ausstattung an RAM Speicher zwei verschiedene Arten von RAM Speicher, die von TOS getrennt verwaltet werden, an. Sie können die Speicherart, in den das Programm und die Pufferspeicher geladen werden, durch setzen der "Programmflags" im TEMPLEMON Programmkopf beeinflussen.

TEMPLEMON arbeitet auch auf dem TT mit AutoSwitch OverScan zusammen. Im Gegensatz zum ST muß TEMPLEMON nach dem OverScan Treiber installiert werden, da TEMPLEMON Routinen dieses Treibers zum Schalten der Bildschirmgröße nutzt.

Kapitel 7

Das Zusammenspiel zwischen TEMPLEMON und SYSMON

SYSMON heißt ein Programm, das als System Monitor verstanden werden möchte. Es ist ein Programm von Karsten Isakovic. Das Programm kann gegen eine Entrichtung einer Gebühr direkt beim Autor bezogen werden. Sie erhalten dann auch eine Anleitung. Für weitere Informationen zum Bezug wenden Sie sich bitte an den Autor:

Karsten Isakovic, Wilmersdorfer Straße 82, D-W-1000 Berlin 12.

SYSMON ist die ideale Ergänzung zu TEMPLEMON . Während TEMPLEMON Ihnen einen umfassenden Einblick in die Funktion des Mikroprozessors bietet, und damit der Hardwarekomponenten Ihres Atari Computers, bietet SYSMON Ihnen einen umfassenden Einblick in das Betriebssystem Ihres Computers, dem TOS. Sie können sich dort die augenblickliche Speicherplatzverteilung anschauen, welchem Programm, das mit Namen aufgeführt wird, welcher Speicherplatz zugewiesen wurde, und ob es sich um Programme oder Daten handelt. Sie können auf Wunsch alle Betriebssystemaufrufe, die ein Programm absetzt, auf dem SYSMON Bildschirm mitprotokollieren lassen, und bekommen die Aufrufparameter sowie die Rückgabewerte in Klartextdarstellung angezeigt. Sie können diese "Trace" Option für jede einzelne Betriebssystemfunktion getrennt ein- und ausschalten. Sie können auf, von Ihnen bestimmte Betriebsystemfunktionen Haltepunkte setzen, und dann TEMPLEMON aufrufen lassen, um entweder die Parameter zu modifizieren, oder ab diesem Zeitpunkt zu tracen.

Sind TEMPLEMON und SYSMON gleichzeitig resident im Speicher vorhanden, bietet sich für den Benutzer die Möglichkeit an, mit beiden Programmen zusammen zusätzliche Möglichkeiten zu nutzen. So können Sie SYSMON anweisen, beim Aufruf von bestimmbaren Betriebssystemfunktionen durch ein Programm, den TEMPLEMON zu aktivieren. Dabei können Sie bei neueren SYSMON Versionen entscheiden, ob nun TEMPLEMON vor Eintritt ins TOS aktiviert werden soll, das heißt, Sie könnten die Aufrufparamter ändern, oder bevor in das aufrufende Programm zurückgesprungen wird, das heißt, Sie könnten die vom Betriebssystem zurückgelieferten Rückgabewerte ändern.

- SysMon Version 1.0.4
  Unter Umständen bietet Ihnen eine diese SysMon Versionen auch schon diese Trace-Möglichkeit an. Stellen Sie mit der Taste [T] (T für TempleMon) statt mit der Taste [+] für jede einzelne Betriebsystemfunktion ein, daß diese mit TempleMon getraced werden soll.

- SysMon Version 1.0.5
  Stellen Sie mit der Taste [D] (D für Debugger) statt mit der Taste [+] für jede einzelne Betriebsystemfunktion ein, daß diese mit TempleMon getraced werden soll.

- SysMon ab Version 1.0.7
  Stellen Sie mit der Taste [I] (I für Input, d.h. vor Aufruf der TOS Funktion um die Übergabeparameter zu bekommen) oder mit der Taste [O] (O für Output, d.h. nach Aufruf der TOS Funktion um die Rückgabeparamter zu bekommen) statt mit der Taste [+] für jede einzelne Betriebsystemfunktion ein, daß diese mit TempleMon getraced werden soll.

Diese Möglichkeit des Aufrufes TempleMons durch SysMon ist unabhängig von der Installationsreihenfolge von TempleMon und SysMon.

Besitzen Sie einen SysMon mit Versionsnummer 1.0.8 oder höher, dann sollten Sie immer TempleMon vor SysMon installieren lassen. Für diesen Fall tritt ein weiteres Zusammenspiel zwischen diesen beiden Programmen in Kraft. TempleMon richtet nach seiner Installation einen Eintrag mit der Kennung 'TMon' im Cookie Jar ein. Der Eintrag besteht aus einem Langwort, genauer gesagt der Adresse eines besonderen Funktionshändlers in TempleMon . Über die einzelnen Funktionen und der Parameterübergabe informiert Sie ein eigenes Kapitel. In diesem Funktionshändler sind Funktionen vorhanden, die SysMon nutzt, um das erweiterte Zusammenspiel zwischen diesen beiden Programmen zu ermöglichen. Sollten Sie SysMon vor TempleMon starten lassen, kann SysMon diese Cookie Schnittstelle im Cookie Jar nicht finden, und nicht nutzen, da er das Vorhandensein der Schnittstelle nur beim Installieren überprüft. Daher die Beschränkung auf die Installationsreihenfolge.

7.1 Die Funktionen des erweiterten Zusammenspiels

Es sei gleich darauf hingewiesen, daß dieses Zusammenspiel besonders viel Spaß bereitet, wenn Sie einen Atari TT mit monochromen Großbildschirm besitzen. Diese Gerätekonfiguration ist aber nicht Voraussetzung.

- Es ist die Möglichkeit geben, vom einen in das andere Programm per Tastendruck zu wechseln, als wäre jeweils das eine Programm Teil des anderen Programmes, und würde nur als Unterprogramm aufgerufen. Vorraussetzung für einen solchen Wechsel

ist natürlich, daß entweder SysMon oder TempleMon aktiv ist. Der Wechsel erfolgt jeweils in beiden Programmen mit der Tastenkombination SHIFT links und HELP . Der Aufruf TempleMons durch SysMon ist immer möglich, der Aufruf SysMons durch TempleMon fast immer. Diese Einschränkung ist immer solange gegeben, solange SysMon sich in einem kritischen Zustand befindet. Wird SysMon von TempleMon zum Beispiel durch Tastaturaufruf in diesem kritischen Zustand unterbrochen, dann läßt TempleMon den Aufruf von SysMon nicht zu, das heißt, er ignoriert ganz einfach Ihren Wunsch, in den SysMon zu wechseln, und führt ihn nicht aus. Dieser kritische Zustand SysMons ist während seiner Installation und während des Betriebes hin und wieder gegeben. Sie bemerken es dadurch, daß TempleMon den Wechsel zu SysMon verweigert. Andersherum, also seitens TempleMons, gibt es für SysMon keine solche Einschränkung. Sie können zwischen beiden Programmen ständig hin und herwechseln, ohne eine Schachtelung und damit einen Absturz befürchten zu müssen. Das Protokoll zwischen beiden Programmen ist abgesichert. Haben Sie von TempleMon SysMon aufgerufen, und Sie besitzen die SysMon Version 1.0.8, dann müssen Sie alle Pulldown Menüs SysMons schließen, bevor Sie wieder zu TempleMon zurückkehren können. Diese Einschränkung will der Autor in einer der nächsten SysMon Versionen beseitigen. Haben Sie von SysMon TempleMon aufgerufen, so können Sie sowohl mit SHIFT links HELP oder auch durch Eingeben des TempleMon Kommandos 'G' zu SysMon zurückkehren. Die TempleMon Befehle 'G', 'Q', und 'T' sind, wenn TempleMon über die oben genannte Tastenkombination von SysMon aufgerufen wurde, in Ihrer Funktion eingeschränkt. Alle 'G'o Befehle, ob nun mit Adresse oder als Gosub 'GS' angegeben, werden als einfaches 'G'o behandelt. Der Befehl 'Q' wirkt genauso wie 'G', es wird also kein PTERM(-1) TOS-Aufruf abgesetzt. Alle 'T' Befehle mit allen Kombinationen hintendran werden wie der einfache 'T' Befehl behandelt, d.h. Sie bekommen den Trace Status angezeigt, der zu diesem Zeitpunkt immer "nicht aktiv" ist.

- Besitzen Sie einen Atari TT mit monochromen Großbildschirm, dann werden Sie sicherlich festgestellt haben, daß TempelMon und SysMon zusammen einen Bildschirm teilen. Das bringt zum Einen eine Speicherplatzersparnis von 153600 Bytes, zum Anderen bietet es ein neues Gefühl eines besseren Überblickes. Sie werden auch festgestellt haben, daß solange SysMon und TempleMon sich einen Bildschirm teilen, der TempleMon Zeichensatz invertiert ist. Das hat SysMon aus Gründen der einheitlichen Optik so umgestellt.

Das für das erweiterte Zusammenspiel genutzte Cookie Interface ist so ausgelegt, daß nachträglich einfach Erweiterungen der Definition gemacht werden können. Das SysMon TempleMon Zusammenspiel könnte eines Tages erweitert werden. Um was, das wird hier nicht verraten.

Kapitel 8

Die Cookie Schnittstelle

Hier folgt nun die Beschreibung einer Schnittstelle, die TEMPLEMON anderen Programmen zur Verfügung stellt, um TEMPLEMON zu erweitern, so wie dies SYSMON macht, oder um TEMPLEMON zu konfigurieren und schließlich und endlich die User Tracefunktion, die an anderer Stelle dieses Handbuches mehrfach erwähnt wurde, zu installieren. Ausgangspunkt ist der sogenannte Cookie Jar, eine von von Atari mit Erscheinen von TOS 1.06 dokumentierte Struktur, die zum Datenaustausch zwischen residenten Programmen und normalen Programmen oder Accesories gedacht ist. Das Verfahren ist relativ einfach. Ein Cookie Eintrag besteht aus 2 Langworten. Das erste Langwort enthält das Programmkennzeichen, bei TEMPLEMON ist dies die Zeichenfolge 'TMon'. Das zweite Langwort kann einen beliebigen Wert enthalten, bei TEMPLEMON ist dies die Adresse des Cookie Händlers . Dieser wird per Jump Subroutine angesprungen. Prozessorregister dienen dabei der Parameterübergabe. Zu beachten ist, daß dieser Cookie Händler nur im Supervisor Modus, dem priviligierten Modus des Prozessors angesprungen werden darf. Beachten Sie das nicht, dann verhält sich TEMPLEMON genauso, als hätten Sie eine ihm unbekannte Funktion aufrufen wollen (s.u.). Der Händler selbst schaltet den Stapel nicht um, sondern arbeitet auf dem Aufruferstapel. Dadurch gibt es keine Reentranzprobleme. Das heißt für Sie, Sie dürfen zu jeder Zeit diesen Funktionshändler aufrufen, auch in Interruptroutinen. Zur Parameterübergabe werden nur die Prozessorregister D0, D1 und D2, sowie A0, A1 und A2 genutzt. Die Inhalte aller anderen Prozessorregister werden nicht zerstört. Alle Parameter werden generell als Langwort übergeben. Zeiger auf Funktionen, Prozeduren, Strukturen und Variablen dürfen nur in den Adreßregistern übergeben werden. Ein Zeiger, der sogenannte Nullzeiger, der den Wert Null (0.L) enthält, ist ein ganz besonderer Zeiger. Wird er einer Funktion übergeben, oder liefert TEMPLEMON diesen Zeiger zurück, dann darf diese Funktion, Prozedur, Struktur oder Variable niemals referenziert, also angesprochen oder angesprungen werden. TEMPLEMON selbst handhabt dies auch so. Beispielsweise haben Sie eine User Traceroutine implementiert, und bei TEMPLEMON angemeldet. Nun wollen Sie die User Tracefunktion wieder abschalten. Das können Sie natürlich auch so machen, daß Sie TEMPLEMON zum Beispiel von der Tastatur aus aktivieren, und den entsprechenden Befehl ("BU") eintippen. Wenn Sie es vom Programm aus machen, dann

müssen Sie dazu einen Nullzeiger übergeben. Wie Sie sicherlich schon vermuten, erreichen Sie über das Cookie Interface nicht nur die Funktion zum Setzen der User Traceroutine. Es gibt mehrere Funktionen, die mit einer Funktionsnummer im Prozessorregister D0 ausgewählt werden. Was es da so alles gibt, zeigt Ihnen die weiter unten stehende Übersicht. Bleibt noch als letztes zu klären, was passiert, wenn Sie mit der Funktionsnummer eine Funktion ausgewählt haben, die gar nicht vorhanden ist. Dann liefert TempleMon eine Fehlermeldung zurück, und zwar werden die Prozessorregister D1, D2 A0, A1, A2 mit Null initialisiert, D0 einer Minus Eins. D.h.: haben Sie einen Zeiger auf irgendetwas erfragen wollen und haben einen Nullzeiger zurückbekommen, den Sie nach Definition nicht referenzieren dürfen. So gesehen brauchen Sie nicht einmal einen Test auszuführen, ob die von Ihnen angesprungene Funktion im Cookie Interface vorhanden war. Eine Plausibilitätsprüfung der Returnwerte kann entfallen.

8.1 Die einzelnen Funktionen der Cookie Schnittstelle

8.1.1 Funktion 0 : TempleMon Patchvariablen erfragen

Diese Funktion liefert in A0 einen Zeiger auf die erste Patchvariable. In D0 wird ein reservierter Wert zurückgeliefert. In D1 wird der gefundene Prozessortyp zurückgeliefert. Diese Zahl setzt sich genauso zusammen, wie der zu übergebenen Prozessortyp beim Aufruf des Disassemblers über dieses Interface (s.u.).

8.1.2 Funktion 1 : TempleMon Versionsnummer erfragen

Liefert die Versionsnummer in D0, und das Erstellungsdatum in A0 zurück. Das Erstellungsdatum ist genauso kodiert, wie das TOS Erstellungsdatum im TOS Header. Die Versionsnummer wird im unteren Wort zurückgeliefert. Dabei steht dort im Highbyte die Zahl vor dem Komma, im Lowbyte die Zahl nach dem Komma. Ist das obere Wort in D0 ungleich 0, handelt es sich um eine nicht öffentliche Betaversion von TempleMon.

8.1.3 Funktion 2 : Adresse der User-Traceroutine setzen

Mit dieser Funktion können Sie Ihre User-Traceroutine bei TempleMon anmelden. Dazu übergeben Sie in A0 die Adresse Ihrer User-Traceroutine, oder den Nullzeiger, falls Sie eine schon einmal gesetzte Option abschalten wollen. In A0 bekommen Sie die Adresse der vorher eingestellten User-Traceroutine zurück. Falls Sie als neue User-Traceroutine die Adresse -1.L übergeben, dann wird dieser Wert nicht als neue Adresse für Ihre User-Traceroutine übernommen, aber Sie bekommen nach Aufruf in A0 die Adresse der aktuellen User-Traceroutine

zurück. In Ihrer User-Traceroutine sollten Sie keines der Prozessorregister verändern. Da man üblicherweise aber ohne Nutzen von Prozessorregistern nichts ausrichten kann, sollten Sie daher alle von Ihnen genutzte Prozessorregister sichern, und vor Rückkehr restaurieren. Vor Rückkehr müssen Sie in D0 einen Rückgabewert plazieren, der entscheidet, ob TEMPLEMON weitertraced (D0=0), oder in die Kommandoeingabe verzweigt (D0<>0). In der User-Traceroutine bekommen Sie in A0 einen Zeiger auf den von TEMPLEMON geretteten Registerinhalt des Prozessors. Diesen können Sie in Ihrer Routine ändern. Den Aufbau dieser Struktur der Prozessorregister entnehmen Sie bitte aus den Beispielprogrammen im Anhang.

8.1.4 Funktion 3 : TEMPLEMON initialisieren

Diese Funktion ist nicht für den allgemeinen Gebrauch gedacht, sondern nur für Debuggingzwecke von TEMPLEMON selber. Sie wird aber der Vollständigkeit halber hier dokumentiert. Vorsicht! Die Definition dieser Funktion kann sich jeder Zeit ändern! Sie dient dazu, falls TEMPLEMON selbst von einem zweiten TEMPLEMON debuggt wird, die Exceptionvektoren des debuggenden TEMPLEMON zu initialisieren. In A0 wird der Zeiger auf die lokalen Variablen des zu debuggenden TEMPLEMON übergeben, welchen der debuggende TEMPLEMON in die Adreßoffsetvariable übernimmt. Ist dieser Zeiger jedoch Null, unterbleibt dies. In D0 wird an den zu debuggenden TEMPLEMON eine Null zurückgeliefert, woraufhin dieser auf einen Breakpoint läuft.

Hintergrund ist, alle Exceptionroutinen von TEMPLEMON testen und mit einem zweiten TEMPLEMON tracen zu können. Dies geht selbst für die Traceroutine TEMPLEMONS. Somit kann man TEMPLEMON beim Tracen mit einem TEMPLEMON tracen, was die Fehlersuche in TEMPLEMON sehr erleichtert. In den Patchvariablen kann man dieses Verfahren für jede einzelnen Exceptionroutine getrennt ein- und ausschalten. Dieses Verfahren wird zu Beginn aller Exceptionroutinen von TEMPLEMON ausgeführt, in dem zunächst getestet wird, ob diese Exceptionroutine zu tracen ist (Stichwort Patchvariablen). Ist dies der Fall, dann wird in D1 die korrespondierende Nummer der Exception (sie entspricht der Bitnummer in den Patchvariablen) geladen, in A0 der Zeiger auf die internen lokalen Variablen, und dann die Cookie - Funktion 3 abgesetzt. Der zweite TEMPLEMON speichert gegebenfalls den Zeiger auf die internen Variablen des Ersten, und sucht dann dessen Eintrag im Cookie Jar. Dann initialisert er seine eigenen Exceptionvektoren, ohne dabei zyklische Verkettungen zu erzeugen, löscht D0 und kehrt zurück zum ersten TEMPLEMON, der dann durch die Null in D0 auf einen Breakpoint läuft. Wäre in D0 keine Null gewesen, würde der erste TEMPLEMON in seiner Exceptionroutine ohne Breakpoint weiter fortfahren. Möchten Sie dieses Verfahren für eigene Zwecke nutzen, dann müssen Sie drei Dinge beachten:

- Tragen Sie das Cookie Identifier Ihrer Routine, die von TEMPLEMON aufgerufen werden soll, bei TEMPLEMON in den Patchvariablen ein, oder nutzen Sie standardmäßig 'tmON'.

- Sie dürfen in Ihrer Routine nur die Prozessorregister A0, D0 und D1 zerstören. Alle anderen müssen Sie gegebenfalls sichern.
- Bevor Sie zu TempleMon zurückkehren, plazieren Sie in D0 einen Wert ungleich Null!

8.1.5 Funktion 4 : Parameter des angelegten Bildschirmspeichers erfragen

Diese Funktion erwartet keine Eingangswerte, liefert jedoch in A0 die Adresse des von TempleMon für seine Ausgabe reservierten Bildschirmspeicher, dessen Größe in Bytes in D0 und den Offset der linken oberen Ecke des von TempleMon genutzten Bereiches zur linken oberen Ecke des Speichers in Bytes in D1 zurück.

8.1.6 Funktion 5 : TempleMon Bildschirmoffset neu setzen

Hiermit kann der TempleMon Bildschirm verschoben werden, wenn der Bildschirmspeicher größer als 32KByte ist, andernfalls wird es ignoriert. Eine Verschiebung ist nur in 2 Byte Schritten möglich! Danach wird aber auf jeden Fall der Bildschirm von TempleMon neu aufgebaut, dabei allerdings nicht sichtbar gemacht! Unter Umständen wird dabei der von TempleMon nicht genutzte Bereich des Speichers gelöscht (siehe auch Funktion 6).

8.1.7 Funktion 6 : TempleMon Bildschirmspeichernutzung anmelden

In D1 wird mitgeteilt, ob der von TempleMon nicht genutzte Bereich des Bildschirmes anderweitig genutzt wird, und von TempleMon nicht mehr gelöscht werden darf. Ist D1 ungleich Null, wird diese Funktion aktiviert, ist D1 gleich Null, wird TempleMon den nichtgenutzten Bildschirmspeicher wieder mitinitialisiern.

8.1.8 Funktion 7 : TempleMon Bildschirmausgabe restaurieren lassen

Diese Routine erwartet keine Eingangsparameter. Wird Sie aufgerufen, restauriert TempleMon seinen Bildschirm (unter Beachtung des in Funktion 6 eingestellten Parameters), einmal (!!!) wenn er das nächste mal seinen Bildschirm sichtbar macht. Man kann also nur kurzfristig den Bildschirmspeicher TempleMons für eigene Zwecke mißbrauchen.

8.1.9 Funktion 8 : SysMon Semaphoradresse setzen

Auf Wunsch des SysMon Autors wird diese Funktion hier nicht weiter erläutert.

8.1.10 Funktion 9 : SysMon Einsprungadresse setzen

Auf Wunsch des SysMon Autors wird diese Funktion hier nicht weiter erläutert.

8.1.11 Funktion 10: TempleMon Einsprungpunkt für SysMon

Auf Wunsch des SysMon Autors wird diese Funktion hier nicht weiter erläutert.

8.1.12 Funktion 11: TempleMon Fonts umstellen

Hiermit werden neue Pointer auf den 8 x 16 Font (Adresse in A0 übergeben) und den 8 x 8 Font (Adresse in A1 übergeben) TempleMon übergeben. Falls Nullzeiger übergeben werden, wird der dazugehörige interne TempleMon Fontpointer nicht ersetzt. Nach Aufruf dieser Funktion liefert TempleMon seine vorherigen Fontpointer in den beiden Adreßregistern zurück.

8.1.13 Funktion 12: TempleMon Disassembler aufrufen

Dies ist der Zugang zum TempleMon Disassembler für externe Programme. In A0 wird die Adresse des Codes, der disassembliert werden soll, übergeben. In D1 wird im niederwertigsten Byte der gewünschte Prozessortyp angegeben. Ist das niederwertige Nibble dieses Bytes 0, wird eine 68000 CPU gewünscht, ist es 1 dann 68010, ist es 2 dann 68020, ist es 3 dann 68030 und ist es 4 dann 68040. Ist dieses Nibble größer als 4, wird auch von einer 68000 CPU ausgegangen. Ist das höherwertige Nibble dieses Bytes ungleich Null, dann werden FPU Befehle berücksichtigt, falls die CPU 68020 oder 68030 war. Nach Aufruf dieser Funktion enthält A0 die Adresse des nächsten, folgenden Codes. A1 enthält einen Zeiger auf einen Null terminierten String, der das Disassemblie enthält. A2 enthält einen Zeiger auf den entsprechenden Eintrag des Opcode auf die interne Opcode Tabelle TempleMons falls dieser Opcode gültig war. Ein Eintrag dieser Tabelle besteht aus zwei Langwörtern. Im niederwertigsten Byte des zweiten Langwortes ist die interne Befehlsformatnummer des Befehls kodiert. Ist das niederwertige Wort von D0 gleich -1, handelte es sich nicht um einen gültigen Opcode. Ist dieses nicht der Fall, aber Bit 15 gesetzt, dann war für diesen Opcode eine unzulässige effektive Adresse angegeben. Die Opcodenummer ist in den Bits 0 bis 14 kodiert. Die Zuordnung zwischen TempleMon Opcodenummer und mnemnonischer Darstellung entnehmen Sie bitte aus dem Anhang.

8.1.14 Funktion 13: Adresse der Online Hilfe setzen

Mit dieser Funktion teilen Sie TEMPLEMON die Adresse einer Funktion mit, die immer dann aufgerufen wird, wenn Sie in TEMPLEMON die HELP Taste drücken, ausgenommen jedoch die Kombination SHIFT links und HELP , die für den SYSMON Aufruf reserviert ist. Im Prozessorregistern A0 müssen Sie die Adresse Ihrer Hilfe Funktion plazieren. Ist diese Adresse der Nullzeiger, springt TEMPLEMON diese Funktion nicht an! Haben Sie im Prozessorregister A0 eine -1.L übergeben, wird diese Adresse nicht gesetzt, und Sie bekommen nur den Returnwert. Nach Aufruf erhalten Sie im Prozessorregister A0 die alte Adresse der OnLine Hilfe Funktion zurück (in den Prozessorregister A1 und D1 befinden sich nach Aufruf dieser Funktion reservierte Werte, in D0 ihre übergebene neue Adresse der OnLine Hilfe). Somit können sich mehrere OnLine Hilfe Funktionen einhängen, wenn diese sich gegenseitig aufrufen. Eine OnLine Hilfe Funktion mit der Adresse 0.L, also dem Nullzeiger, darf nicht aufgerufen werden. Statt dessen müssen Sie dann zu TEMPLEMON zurückkehren. Die OnLine Hilfe Funktion wird von TEMPLEMON mit JSR aufgerufen. Mit RTS kehren Sie wieder zu TEMPLEMON zurück. Bei der Rückkehr erwartet TEMPLEMON im Prozessorregister D0 einen Rückgabewert. Ist dieser Rückgabewert 0.L, so springt TEMPLEMON wieder normal in den Eingabemodus, andernfalls interpretiert er den HELP Tastendruck als Druck auf die Taste RETURN , woraufhin TEMPLEMON mit der Auswertung der Eingabezeile beginnt. Die aufgerufene OnLine Hilfe Funktion bekommt vier Parameter mitgeteilt:

- Im Prozessorregister A0 die Adresse des Eingabepuffers. Dort steht die aktuelle Eingabezeile, die 80 Bytes lang und nicht Null-terminiert ist. Verändern Sie die Eingabezeile, so wirkt sich das auf TEMPLEMON aus. Haben Sie die Eingabezeile modifiziert, sollten Sie den Cursor direkt hinter das letzte Zeichen plazieren (s.u.).
- Im Prozessorregister A1 die Adresse der horizontalen Cursorposition. Diese Cursorposition ist eine 16 Bit große Variable, die jedoch nur die Werte 0 bis 79 annehmen darf. Für die korrekten Werte sind Sie selbst verantwortlich!
- Im Prozessorregister D0 den Bitvektor der SHIFT , CONTROL und ALTERNATE Taste. Dieser Bitvektor ist genauso wie die entsprechende BIOS Funktion aufgebaut.
- Im Prozessorregister D1 den Prozessortyp (s.o.).

Die OnLine Hilfe Funktion darf die Datenregister D0-D7 und die Adressregister A0-A6 zerstören. Auf extensiven Stapelbedarf sollten Sie verzichten, da sonst u.U. ein Stapelüberlauf wichtige Datenbereiche durch Überschreiben zerstört. Sollten Sie trotzdem viel Stapelplatz beanspruchen, sollten Sie entweder die TEMPLEMON Stapelgröße erhöhen (s. Patchvariablen), oder auf einem eigenen Stapel umschalten. Auf jeden Fall muß der alte Stapel vor Rückkehr zu TEMPLEMON wieder hergestellt werden.

8.1.15 Funktion 14: Cursorposition setzen und holen

Diese Funktion erlaubt es, die horizontale Cursorposition zu setzen. Dazu übergeben Sie im Prozessoregister D1 die neue horizontale Cursorposition. Ist diese Cursorposition -1.W, dann wird die neue Cursorposition nicht gesetzt. Nach Aufruf dieser Funktion erhalten Sie im unteren Wort von D0 die alte horizontale, im oberen Wort von D0 die vertikale Cursorposition zurück. Im unteren Wort von D1 erhalten Sie die Anzahl an Spalten, im oberen Wort die Anzahl der Zeilen des Bildschirms zurück. Beachten Sie bitte, daß die Cursorposition von Null ab gezählt wird. Die maximale horizontale Cursorposition ist damit Anzahl der Spalten minus eins. Die vertikale Cursorposition kann nur durch Ausgabe von Linefeed beeinflußt werden (s.u.).

8.1.16 Funktion 15: String auf TEMPLEMON Bildschirm ausgeben

Durch Aufruf dieser Funktion können Sie einen String beliebiger Länge an der aktuellen Cursorposition im TEMPLEMON Bildschirm ausgeben. Ist ein Protokoll eröffnet, erscheint dieser String auch auf dem Protokoll. Der String muß Null terminiert sein, und seine Adresse wird im Prozessorregister A0 plaziert. Außer der Menge der Zahlen, der kleinen und großen Buchstaben sind nur noch die Steuerzeichen "Carriage Return" (13) und "Linefeed" (10) zugelassen. Die Ausgabe von "Carriage Return" bewirkt ein Löschen aller Zeichen hinter der aktuellen horizontalen Cursorposition in dieser Zeile. Sie sollten zum Initialisieren der nächsten Zeile daher immer erst "Carriage Return", "Linefeed", und dann wieder "Carriage Return" benutzen. Auf die Kombination erst "Linefeed" ausgeben, und dann den Cursor mittels der Cookie Jar Funktion 14 auf den Zeilenanfang zu setzen, sollten Sie aus Gründen eines ordentlichen Protokolls verzichten.

8.1.17 Funktion 16: Bildschirm restaurieren

Diese Funktion veranlaßt TEMPLEMON, sofort seinen Bildschirm zu restaurieren. Ist D1 ungleich Null, so wird auch der von TEMPLEMON nicht genutzte Bereich initialisiert, sofern nicht eine andere Nutzung bei TEMPLEMON angemeldet ist (s. Funktion 6).

8.1.18 Funktion 17: Adresse der Label und Makroverarbeitung setzen

Mit dieser Funktion teilen Sie TEMPLEMON die Adresse einer Label und Makroverarbeitenden Funktion mit, die immer bei Eingabezeilenabschluß, also beim Drücken der Taste RETURN aufgerufen wird. Für Parameterübergabe und Übernahme, sowie für den Aufruf Ihrer Label und Makro Funktion gelten ansonsten die gleichen Bedingungen, wie für

die OnLine Hilfe Funktion. Beispielsweise könnten Sie mit Hilfe dieser Funktion einen Programmlader realisieren, bei dem TEMPLEMON auch Symbole und die Programmlabels bei der Eingabe zuläßt.

8.1.19 Funktion 18: Keyboardpuffer löschen/anzeigen

Ist D1 gleich Null wird der Keyboardpuffer gelöscht, andernfalls an der aktuellen Zeile ausgegeben.

8.1.20 Funktion 19: Tastendruck abfragen

Diese Funktion liefert in D0 den ASCII Code einer eventuell gerade gedrückten Taste zurück. Dieser ist Null, wenn gerade keine Taste gedrückt ist. Wurde die SPACE Taste gedrückt, wartet dieser Aufruf den Tastendruck der nächsten Taste ab, bevor er zurückkehrt. Ist der zweite Tastendruck ein Space gewesen, wird auch der ASCII Code für Space zurückgeliefert.

Kapitel 9

TempleMon konfigurieren

TempleMon läßt sich an eigene Bedürfnisse anpassen. Dazu besitzt er einen initialisierten Datenbereich mit Konstanten, deren Initialisierungswerte geändert werden können. Diese Werte beeinflußen Speicherplatzbedarf und Funktion von TempleMon. Wir haben diese als Patchvariablen bezeichnet und besonders gekennzeichnet. Laden Sie sich doch einmal das Programmfile TempleMon binär, z.B. mit Hilfe von TempleMon selber. Relativ bald zu Filebeginn kann Sie dort einen String 'TemplemonVar' lesen. Direkt dahinter beginnen die Patchvariablen. Die folgende Übersicht zeigt diese für TempleMon Version 2.0. Die Patchvariablen können sich aber auch ändern! Nutzen Sie daher bitte nur die jeweils zum Programm passende Liste.

Das Installationsprogramm 'TMonInst', mit dem Sie sich hoffentlich Ihre persönlichen Einstellungen erstellt haben, macht nichts anderes, als die Patchvariablen zu modifizieren. Ein Teil der Patchvariablen können zur Laufzeit modifiziert werden, ein andere Teil nicht, bzw. es hätte keine Auswirkungen mehr. Das Konfigurations - CPX bietet die Manipulation der Patchvariablen an, deren Änderung zur Laufzeit eine Änderung im Verhalten TempleMons nach sich zieht. Auf das Installationsprogramm und das Konfigurations - CPX wird nun nicht mehr weiter eingegangen, da sie einfach nach allgemeinen GEM Standard zu bedienen sind. Zudem bietet das Installationprogramm eine Online Hilfe an, in der der jeweilige Menüeintrag bzw. Dialogbutton erklärt ist. Halten Sie doch einfach die CONTROL Taste gedrückt, bevor Sie einen Menüeintrag oder einen Button anwählen. Dann öffnet sich von alleine das zugehörige Hilfefenster.

Alle Patchvariablen TempleMons, die den Speicherbedarf beeinflußen, sollten Sie zur Laufzeit nicht mehr ändern, falls Sie sich ihre eigenen Konfigurationprogramme oder Accessories schreiben sollten. Auch diese, die die Ausdehnungen des Bildschirms festlegen, gehören dazu. Anders ist es mit den Bildschirmfarben, mit den 'virtuell' und 'debugging' Patchvariablen, und denen, die das Verhalten TempleMons bei einem 'MOVE von SR' oder einer 'Division durch Null' festlegen. Diese könnten auch nach einer Installation TempleMons im Speicher geändert werden. An den Speicherplatz der Patchvariablen kommen Sie über das Cookie Jar

Interface. Beachten Sie jedoch, daß Sie dort den Zeiger auf die erste Patchvariable übergeben bekommen, der also direkt hinter den Startstring zeigt. Das Konfigurations CPX, mit dem Sie TEMPLEMON nach seiner Installation umkonfigurieren können, greift auch über das Cookie Jar Interface auf die Patchvariablen zu. Die Patchvariablen sind im Anhang aufgeführt.

Kapitel 10

TempleMon und durch Software ausgetauschtes TOS

Die MMU der 68030 und 68040 Prozessoren bietet viele Möglichkeiten. Eine dieser Möglichkeiten ist, das TOS in das RAM zu legen, und es zu modifizieren, oder gleich eine neuere TOS Version zu laden, ohne gleich die ROMS austauschen zu müssen. Dazu wird dann die MMU so programmiert, daß ein Zugriff auf den Adreßbereich der ROMs, wo das TOS eigentlich lokalisiert ist, in einen anderen Adreßbereich, meistens irgendwo im TT-RAM übersetzt wird. Der Prozessor bekommt dann beim Abarbeiten von Betriebssystemroutinen von der geänderten Adreßlage nichts mit, und glaubt, das TOS befände sich in den ROMs. Nur so ist ein Austausch des TOS z.B. Ersatz durch eine neuere Version ohne Anpassung des Austausch-TOS an die Adreßlage möglich. Es gibt das Public Domian Programm ROMRAM von Alexander Herzlinger, welches in der Lage ist, ein anderes TOS nachzuladen, und es statt des TOS im ROM zu aktivieren. In einem solchen oder ähnlichem System mit nachgeladenem oder modifiziertem TOS kann es zu Problemen kommen, wenn plötzlich die Adreßübersetzung der MMU abgeschaltet wird. Es wird wohl meistens der Fall sein, daß die Adreßlage von Interrupt, Exception und Betriebssystemroutinen zwischen dem ausgetauschten und dem Austausch-TOS variieren. Beim Durchlaufen der Resetroutine werden die Vektoren für diese Routinen initialisiert, und beim Austausch des original TOS wiederrum. Ein Abschalten der Adreßübersetzung nach erfolgreichem Austausch des TOS hat dann zur Folge, daß diese Vektoren plötzlich falsch initialisiert sind. Das wird zur Folge haben, daß der Prozessor vielleicht auf eine illegale Instruktion stößt, und damit TempleMon aktiviert. Leider wird man dann außer ein paar wilden Punkten auf dem TempleMon Bildschirm nichts mehr erkennen können. Das ist allerdings auch vollkommen logisch so, da auch TempleMon auf ein paar Daten und Programmteile des Betriebssystemes angewiesen ist. Man kann aber einer solchen Situation vorbeugen, allerdings ist dazu ein klein wenig Aufwand notwendig. Hier wird nun beschrieben, wie Sie TempleMon davor schützen können, bei einem plötzlichen Wegfall der Adreßübersetzung bei nachgeladendem TOS, abzustürzen. Diese Beschreibung und der sich anschließende Programmvorschlag ist jedoch nur als Anregung zu verstehen, da er einen wesentlichen Aspekt ausläßt, der jedoch auch wichtig sein könnte, nämlich die Sicherung der

ständig wiederkehrenden Interrupts wie Bildrücklauf und Timerinterrupt. Dieser Programmvorschlag basiert auf dem ROMRAM Programm Version 1.1H von Alexander Herzlinger, da es von diesem die physikalisch richtige Adresse des nachgeladenen TOS erfragt.

Die Idee, die dahintersteckt:

- Kopieren des Systemzeichensatzes aus dem ROM und Umsetzen des TEMPLEMON Zeichensatzzeigers von der ROM Adresse auf die Adresse der Kopie im RAM.
- Einfügen von ausführbaren Code in die XBRA Verkettung zwischen TEMPLEMON und TOS an den Stellen, an denen TEMPLEMON ins TOS weiterspringt. Das trifft u.a. auch an den folgenden Stellen zu:
  - "MFP Interruptvektor"
  - "IKBDSYS" Softwarevektor
  - "Privilege Violation" Exceptionvektor
  - "Division by Zero" Exceptionvektor
  - "Buserror" Exceptionvektor bei einfachen virtuellen Speicherverwaltungssystemen.

  Weiterhin wird das Betriebssystem die folgenden zwei Vektoren ständig anspringen:

  - "Timer Interrupt"
  - "VBL Interrupt" (vertikaler Bildrücklauf)

  Das in die XBRA Verkettung eingefügte Stück Code leistet dabei folgendes:

  Wird es angesprungen, schaut es die OS-Version nach, und vergleicht diese mit der Version, die es beim Programmstart gefunden hat. Tritt Übereinstimmung auf, wird ganz normal in der XBRA Kette weitergesprungen, andernfalls wird über einen konstanten Vektor gesprungen. Dieser konstante Vektor wird bei Programminitialisierung nach folgendem Muster berechnet: Alten XBRA-Vektor holen, die drei höchstwertigen Nibbles aus der 32 Bit Adresse ausmaskieren, und die RAM-Startadresse des nachgeladenen TOS, welche aus den Patchvariablen von ROMRAM gelesen wird, draufaddieren. Dies garantiert, daß auch bei einem Wechsel der TOS Version durch Deaktivieren der Adreßumsetzung TempleMon die Exceptionroutinen anspringt, für die TOS Variablen initialisiert wurden, und das bei einer Exception der Einsprungpunkt stimmt.

Hier folgt ein Beispielprogramm, das den TEMPLEMON Zeichensatzzeiger austauscht, und das oben beschriebene Stück Code in die XBRA Kette des MFP Interruptes, IKBDSYS Softwarevektors, "Privilege Violation" Exceptionvektors und des "Division by Zero" Exceptionvektors einfügt. Die restlichen Stellen, wie "Timer" Interrupt oder VBL Interrupt können Sie nach gleichem Verfahren ergänzen.

```
start:          clr.l   -(SP)
                move.w  #$20,-(SP)
                trap    #1
                addq.l  #6,SP
                movea.l $05A0,A0        ; Cookie Jar Pointer
                tst.l   $05A0
                beq     prg_abort
search_PTOS:    cmpi.l  #'PTOS',(A0)    ; ROMRAM Cookie suchen
                beq     set_PTOS
                lea     8(A0),A0
                tst.l   (A0)
                bne     search_PTOS
                bra     prg_abort
set_PTOS:       movea.l 4(A0),A0        ; Adresse ROMRAM Patchvariablen
                move.l  $52(A0),PTOS    ; physikalische TOS Adresse im
                                          RAM aus Patchvariablen
                movea.l $05A0,A0
search_TMon:    cmpi.l  #'TMon',(A0)    ; TempleMon Cookie suchen
                beq     setfnt
                lea     8(A0),A0
                tst.l   (A0)
                bne.s   search_TMon
prg_abort:      clr.w   -(SP)
                trap    #1
setfnt:         movea.l 4(A0),A6        ; Adresse TempleMon Cookie
                                        ; Funktionshandler
                move.l  A6,tmon_routine
                move.w  #34,-(SP)       ; KBDVbase
                trap    #14
                addq.l  #2,SP
                movea.l D0,A0
                movea.l 32(A0),A0       ; IKBD_SYS holen
ikbdloop:       cmpi.l  #'XBRA',-12(A0)
                bne     prg_abort
                cmpi.l  #'TMon',-8(A0)  ; TempleMon IKBD_SYS suchen
                beq     ikbdfound
                movea.l -4(A0),A0
                bra     ikbdloop
ikbdfound:      move.l  A0,tmon_ikbd
                movea.l $0100+6*4,A0    ; MFP Interrupt holen
mfploop:        cmpi.l  #'XBRA',-12(A0)
                bne     prg_abort
                cmpi.l  #'TMon',-8(A0)  ; TempleMon MFP suchen
                beq     mfpfound
                movea.l -4(A0),A0
                bra     mfploop
```

```
mfpfound:       move.l  A0,tmon_mfp
                movea.l $20,A0          ; Privilege Violation Exception holen
privloop:       cmpi.l  #'XBRA',-12(A0)
                bne     prg_abort
                cmpi.l  #'TMon',-8(A0)  ; TempleMon P.V.E. suchen
                beq     privfound
                movea.l -4(A0),A0
                bra     privloop
privfound:      move.l  A0,tmon_priv
                movea.l $14,A0          ; Division by Zero Exception holen
divzloop:       cmpi.l  #'XBRA',-12(A0)
                bne     prg_abort
                cmpi.l  #'TMon',-8(A0)  ; TempleMon D.b.Z. suchen
                beq     divzfound
                movea.l -4(A0),A0
                bra     divzloop
divzfound:      move.l  A0,tmon_divz
                move.w  $E00002,os_version  ; aktuelle OS-Verion sichern
                lea     newmfp(PC),A1   ; neue MFP Routine hinter TempleMon
                movea.l tmon_mfp,A0     ; einhängen
                move.l  -4(A0),-4(A1)
                move.l  A1,-4(A0)
                move.l  -4(A1),D0       ; RAM Adresse MFP Routine berechnen
                andi.l  #$0FFFFF,D0
                add.l   PTOS,D0
                move.l  D0,othermfp
newpriv(PC),A1:
                move.l  -4(A0),-4(A1)
                move.l  A1,-4(A0)
                move.l  -4(A1),D0       ; RAM Adresse Privilege Violation
                andi.l  #$0FFFFF,D0     ; berechnen
                add.l   PTOS,D0
                move.l  D0,otherpriv
                movea.l tmon_divz,A0    ; Division by Zero einhängen
                lea     newdivz(PC),A1
                move.l  -4(A0),-4(A1)
                move.l  A1,-4(A0)
                move.l  -4(A1),D0       ; RAM Adresse Division by Zero
                andi.l  #$0FFFFF,D0     ; berechnen
                add.l   PTOS,D0
                move.l  D0,otherdivz
                DC.W $A000              ; Systemfonts holen
                movea.l A1,A3
                addq.l  #4,A3
                movea.l (A3)+,A2
                movea.l 76(A2),A1       ; 8x8 Fontpointer
```

```
                movea.l (A3)+,A2
                movea.l 76(A2),A0       ; 8x16 Fontpointer
                move.l  mask,D2
                lea     big_font,A3     ; Systemfont 8x16 kopieren
                movea.l A0,A4
                move.w  #1023,D0
copy_big:       move.l  (A4)+,D1
                eor.l   D2,D1           ; damit eventuell Font invertieren
                move.l  D1,(A3)+
                dbra    D0,copy_big
                lea     small_font,A3   ; Systemfont 8x8 kopieren
                movea.l A1,A4
                move.w  #511,D0
copy_small:     move.l  (A4)+,D1
                eor.l   D2,D1           ; damit eventuell Font invertieren
                move.l  D1,(A3)+
                dbra    D0,copy_small
                lea     big_font,A0     ; neuen TempleMon Fontpointer setzen
                lea     small_font,A1
                moveq   #11,D0          ; Funktionsnummer 'Font setzen'
                jsr     (A6)
                move.l  A0,tmon_big     ; alte Fonts
                move.l  A1,tmon_small
redraw:         moveq   #7,D0           ; Redraw TempleMon Screen
                jsr     (A6)
                pea     message(PC)     ; Meldung ausgeben
                move.w  #9,-(SP)
                trap    #1
                addq.l  #6,SP
                clr.w   -(SP)           ; Ptermres
                move.l  #(ende-start+$0100),-(SP)
                move.w  #49,-(SP)
                trap    #1
                addq.l  #8,SP
                pea     worse(PC)       ; Meldung ausgeben
                move.w  #9,-(SP)
                trap    #1
                addq.l  #6,SP
                move.l  tmon_mfp,A0       ; Pterm ging in die Hosen, alles
                move.l  oldmfp(PC),-4(A0) ; wieder zurück!
                move.l  tmon_ikbd,A0
                move.l  oldikbd(PC),-4(A0)
                move.l  tmon_priv,A0
                move.l  oldpriv(PC),-4(A0)
                move.l  tmon_divz,A0
                move.l  olddivz(PC),-4(A0)
```

```
                movea.l tmon_big,A0
                movea.l tmon_small,A1
                moveq   #11,D0
                jsr     (A6)
                bra     prg_abort
                DC.L 'XBRA'
                DC.L 'TMON'
oldmfp:         DC.L 0
newmfp:         move.w  D0,-(SP)
                move.w  os_version,D0
                cmp.w   $E00002,D0
                beq     okmfp
                move.w  (SP)+,D0
                move.l  othermfp,-(SP)
                rts
okmfp:          move.w  (SP)+,D0
                move.l  oldmfp(PC),-(SP)
                rts
                DC.L 'XBRA'
                DC.L 'TMON'
oldikbd:        DC.L 0
newikbd:        move.w  D0,-(SP)
                move.w  os_version,D0
                cmp.w   $E00002,D0
                beq     okikbd
                move.w  (SP)+,D0
                move.l  otherikbd,-(SP)
                rts
okikbd:         move.w  (SP)+,D0
                move.l  oldikbd(PC),-(SP)
                rts
                DC.L 'XBRA'
                DC.L 'TMON'
oldpriv:        DC.L 0
newpriv:        move.w  D0,-(SP)
                move.w  os_version,D0
                cmp.w   $E00002,D0
                beq     okpriv
                move.w  (SP)+,D0
                move.l  otherpriv,-(SP)
                rts
okpriv:         move.w  (SP)+,D0
                move.l  oldpriv(PC),-(SP)
                rts
                DC.L 'XBRA'
                DC.L 'TMON'
```

```
olddivz:        DC.L 0
newdivz:        move.w  D0,-(SP)
                move.w  os_version,D0
                cmp.w   $E00002,D0
                beq     okdivz
                move.w  (SP)+,D0
                move.l  otherdivz,-(SP)
                rts
okdivz:         move.w  (SP)+,D0
                move.l  olddivz(PC),-(SP)
                rts
                DATA
mask:           DC.L 0          ; wenn -1 dann invertierte Bilddarstellung
message:        DC.B 13,10,'TempleMon umgestellt.',13,10,0
worse:          DC.B 13,10,'Ein Fehler ist aufgetreten.',13,10
                DC.B 13,10,'TempleMon Umstellung abgebrochen.',13,10,0
                BSS
othermfp:       DS.L 1
otherikbd:      DS.L 1
otherpriv:      DS.L 1
otherdivz:      DS.L 1
os_version:     DS.W 1
PTOS:           DS.L 1
tmon_routine:   DS.L 1
tmon_ikbd:      DS.L 1
tmon_mfp:       DS.L 1
tmon_priv:      DS.L 1
tmon_divz:      DS.L 1
tmon_big:       DS.L 1
tmon_small:     DS.L 1
big_font:       DS.B 4096
small_font:     DS.B 2048
ende:           END
```

Kapitel 11

Debugging von TempleMon mit TempleMon

Will man TempleMon debuggen, weil man einen Fehler darin entdeckt hat, und wir um ein Protokoll gebeten haben, müssen Sie sich einen zweiten TempleMon zum Debuggen des Ersten generieren. Das ist ganz einfach:

- Laden Sie das TempleMon Programmfile binär, und ändern Sie alle gefundenen 'TMon' XBRA Identifikatoren in 'tmON' um.
- Ändern Sie das 'debug_xbra' in den Patchvariablen von 'tmON' nach 'TMon' (zu den Patchvariablen siehe Anhang).
- Ändern Sie den Tastaturaufruf in den Patchvariablen, z.B. die Patchvariable 'tmon_shiftkeys' von 4 auf 6. Dann wird TempleMon über Tastatur mit [SHIFT-links] [CONTROL] [HELP] aufgerufen.
- Speichern Sie diesen TempleMon unter dem Namen 'TMONDBUG.PRG'.
- Laden Sie erneut das nicht modifizierte TempleMon Programmfile binär.
- Ändern Sie die Patchvariable 'debugging' für die entsprechende Exception, deren Exceptionroutine TempleMons z.B. getraced werden soll, in dem Sie in dem LANGWORT 'debugging' für diese Exception das entsprechende Bit setzen. Die Kodierung können Sie aus den Patchvariablen im Anhang entnehmen.
- Speichern Sie diesen TempleMon unter dem Namen 'TMONTEST.PRG' ab.

Das war es auch schon. Nun müssen Sie nur noch den TMONDBUG vor dem TMONTEST starten, der nicht modifizierte TempleMon sollte dabei nicht im Speicher sein. Nun können Sie mit dem TMONDBUG den TMONTEST debuggen.

Wenn Sie sich nun diese Konfiguration erstellt haben, und Sie möchten eine der Exceptionroutinen TEMPLEMONS tracen, vielleicht sogar die Traceroutine (Sie lesen richtig! Sie können TEMPLEMON beim Tracen tracen.), dann läuft der zu debuggende TEMPLEMON auf einen BREAK. Das ruft den debuggenden TEMPLEMON auf. Wenn Sie nun die Exceptionroutine tracen, werden die ersten beiden Befehle ein 'RTS' und 'MOVE.L (SP)+,D0' sein, die noch zu diesem Mechanismus des gegenseitigen Aufrufes beider TEMPLEMONS gehört. Erst Danach geht die eigentliche Exceptionroutine richtig los.

Kapitel 12

Internes für Experten

Es folgen nun ein paar Betrachtungen über Internes in TempleMon.

12.1 Die Softwareemulation für den 68040

Wenn Sie schon ein bißchen im Anhang geblättert haben, dann haben Sie sicherlich auch die Liste aller Befehle der 68000er Familie gesehen. Wenn Sie sich dabei einmal den Befehlsumfang des 68040 angeschaut haben, wird Ihnen sicherlich an einigen Fließkommabefehlen die Bemerkung "nur mit Softwareemulation" aufgefallen sein. In der Tat existieren die so kenntlich gemachten Befehle im 68040 eigentlich nicht. Trifft die CPU auf einen solchen Befehl, wird sie eine Line-F Exception auslösen. Im Prozessorhandbuch zum 68040 weißt Motorola jedoch ausdrücklich auf die Software Library hin, die diese eigentlich nicht in Hardware vorhandenen Befehle mit Hilfe einer Software Library zur Verfügung stellt. Dazu wird die Line-F Exception benutzt. TempleMon fängt allerdings eine solche Exception ab. Um aber die Softwareemulation nicht zu unterbinden, testet TempleMon den Befehl, der zur Line-F Exception geführt hat. Hat es sich um einen Befehl gehandelt, der durch die Software Library ersetzt wird, springt TempleMon in der XBRA-Kette zum TOS weiter, welches hoffentlich diese Library enthält.

12.2 Virtuelle Speichersysteme

Mit dem Erscheinen des 68030, oder, jedoch nicht so häufig eingesetzt, mit dem 68020 & 68851, ist der sogenannte virtuelle Speicher durch Hardwareunterstützung seitens der PMMU möglich geworden. Virtueller Speicher bedeutet lediglich, daß der durch ein Programm angeforderte Speicher mit seinen Adressen als logische Adressen betrachtet wird, die nicht unbedingt den physikalischen Adressen entsprechen müssen. Diese Adreßumsetzung wird durch

die PMMU automatisch erledigt. Trifft die PMMU während der Adreßumsetzung auf eine nicht existente physikalische Adresse, so wird ein Buserror ausgelöst. In der Buserrorroutine wird dann normalerweise zunächst geprüft, ob der Buserror durch die PMMU ausgelöst wurde, und ob es sich um einen "behebbare" Ursache handelt. TEMPLEMON fängt nun seinerseits den Buserror ab. Somit könnte ein "einfacher" virtueller Speichertreiber nicht mehr auf von der MMU ausgelöste Buserror reagieren, was er aber müßte. Allerdings trifft dies nur zu, wenn das VBR Register gleich Null ist. Wenn wie bei den virtuellen Speichermanager Programmen "OUTSIDE" und "VRAM" das VBR Register benutzt wird, um zu garantieren, daß in jedem Falle einer Exception zuerst die Routinen des Speichermanagers, also auch noch vor denen TEMPLEMONS angesprungen wird, dann ist an TEMPLEMON keine Modifikation weiter notwendig. Ist dies jedoch nicht der Fall, dann muß in der TEMPLEMON Buserrorroutine entschieden werden, ob ein durch die MMU und für den virtuellen Speichermanager bestimmte Exception ausgelöst wurde, der diesen z.B. veranlassen würde, einen auf Platte ausgelagerten Speicherblock wieder ins RAM zu laden. Dazu müssen Sie die Patchvariable "virtuell" entsprechend ändern (siehe auch Anhang). Für die nachfolgenden, von der MMU ausgelösten Buserror ist dies möglich:

- Supervisor Violation.
- Write Violation.
- Limit Violation.
- Invalid Descriptor or Page Fault.
- Bus Error during Table Search.

In der Regel wird für virtuelle Speichersysteme nur "invalid descriptor or page fault" genutzt, um einen wesentlich größeren virtuellen Speicher als physikalisch vorhandenen zu ermöglichen. Sie müssen die entsprechenden Bits in der Patchvariable (siehe Anhang) auf eins setzen, damit TEMPLEMON in einem solchen Falle weiter in der XBRA Kette verzweigt, anstatt eine Fehlermeldung zu präsentieren. Ist diese Patchvariable Null, dann macht TEMPLEMON keinen Test, ob der Buserror von der MMU ausgelöst wurde. Ist diese Patchvariable ungleich Null, macht TEMPLEMON eine vollständige Fehleranalyse, und testet nachdem er den Grund des Fehlers entdeckt hat ab, ob das korrespondierende Bit gesetzt ist, um eventuell in der XBRA Kette weiterzuverzweigen. Ist das Korrespondierende Bit aber Null, so gibt er eine erweiterte Fehlermeldung aus, z.B. `''MMU: Schreibschutz''`. Wenn Sie nun diese erweiterten Fehlermeldungen sehen möchten, ohne jedoch bei einer der von der MMU auslösbaren Fehlern in der XBRA Kette weiterzuspringen, dann setzen Sie bitte eines der nicht benutzten Bits in dieser Patchvariablen auf eins.

12.3 Der Test auf Start und Ende des TOS

Der Prozessorhersteller Motorola hat für die Implementierung von Systemaufrufen die TRAP's vorgesehen, die ja auch teilweise für TOS genutzt werden. Der größte Teil der nutzbaren TRAP Vektoren bleibt jedoch normalerweise ungenutzt. Würde ein ungenutzter TRAP ausgelöst, käme es unweigerlich zu den Bomben. TempleMon fängt deshalb die ungenutzten TRAP Vektoren ab, und zeigt entsprechend eine Fehlermeldung an. Nun kann es aber sein, daß ein bisher ungenutzter TRAP Vektor durch ein residentes Programm genutzt wird, sozusagen als Betriebssystemerweiterung. Dies wäre ja auch der vom Prozessorhersteller vorgesehene Weg. Damit TempleMon einer solchen Erweiterung nicht im Weg steht, prüft er bei der Vektorinitialisierung der üblicherweise ungenutzten TRAP Vektoren, ob diese in das TOS zeigen. Zeigen diese nicht ins TOS, so kann man davon ausgehen, daß eine der oben beschriebenen Erweiterungen installiert ist, und eine Initialisierung für diesen Vektor zu unterbleiben hat. Genauso verfährt auch TempleMon . Dabei ist der Start des Betriebssystems TOS relativ einfach herauszufinden; eine der Systemvariablen zeigt auf den TOS Header, der die Startadresse des TOS enthält. Viel schwieriger ist jedoch der Test auf das Ende des TOS. Dafür gibt es keine legale Systemvariable. Üblicherweise sind jedoch alle Arten von Programmen in drei Teile aufgeteilt; in das Text oder Code Segment, das Daten Segment (die initialisierten Daten) und das BSS ("Block Storage Segment", nicht initialisierte Daten). So ist es auch beim TOS. Es gibt auch eine feste Reihenfolge der Segmente, die eigentlich immer eingehalten wird, zuerst das Text Segment, dann das Daten Segment und zuletzt mit den höchsten Adressen das BSS. Beim TOS befindet sich das Daten Segment direkt hinter dem Text Segment. Das BSS Segment befindet sich im RAM. Die einzelnen Segmente sind schön getrennt voneinander und überlappen sich nicht. Um jetzt das Ende des Betriebssystemcodes herauszufinden, genügt die Kenntnis der Adresse einer Variablen aus dem Daten Segment. Ausführbarer Code kann sich dann nicht mehr oberhalb dieser Adresse befinden. Genau dies macht auch TempleMon . Er nutzt dazu das "GEM Magic" aus dem TOS-Header. Leider hat sich gezeigt, daß bei dem proVME TOS für das proVME Hypercache 030 Beschleunigerboard dieses Magic nicht Teil des Datensegmentes ist, sondern im Textsegment liegt. Um solchen Problemen Abhilfe leisten zu können, kann die automatische Betriebssystemstart und Ende Erkennung mittels einer Patchvariable abgeschaltet werden und die Start und Endadresse in den Patchvariablen vorgegeben werden.

12.4 Die Tastaturvektorsuche

TempleMon unternimmt bei seiner Aktivierung den Versuch, immer einen vollfunktionsfähigen Zustand der Tastatur zu garantieren. Dazu überprüft er ein paar Hardwareregister, die den Interrupt der Tastatur blockieren könnten, und löst eventuell eine solche Blockade mit der entsprechenden Fehlermitteilung. Leider gelingt dies nicht immer, was aber nicht zu ändern ist. Bei diesem Test macht er auch eine Überprüfung auf den MFP Interrupt Vektor, sowie auf den IKBDSYS Softwarevektor. Dabei testet er auf die XBRA Kennung der Routinen, auf

die diese Vektoren deuten. Taucht diese XBRA Kennung in der Liste der bekannten XBRA Identifikatoren auf, so testet er die nächste Ebene ab. Dies macht er, solange bis er entweder auf eine unbekannte, oder auf die eigene XBRA Kennung gestoßen ist. Hat er bei einer unbekannten XBRA Kennung (oder auch einer ungültigen) die Suche abgebrochen, dann initialisiert er, solange er aktiv ist, die Vektoren auf sich in diesem Teil der Kette, den er noch für in Ordnung betrachtet hat. Nach Beendigung von TempleMon stellt er dann wieder den alten Zustand her. Diese Option kann in den Patchvariablen abgeschaltet, sowie zusätzliche XBRA Kennung eingefügt werden. Für den Fall, daß TempleMon den Versuch unternommen hat, einen sicheren Tastaturzustand herzustellen, gibt er eine Meldung aus. Davon kann eine oder mehrere der folgenden Meldungen erscheinen, wobei diese sich auf Hardwareregister beziehen:

- Tastaturflanke restauriert
- Tastaturdatenrichtung restauriert
- Interruptvektorregister restauriert
- Tastaturinterrupt gesperrt
- Interruptregister A schwebend : XX
- Interruptregister B schwebend : XX
- Tastaturinterrupt maskiert
- Interruptregister A in Bearbeitung : XX
- ACSI Bus Interrupt in Bearbeitung
- Tastaturinterrupt in Bearbeitung

Erscheint eine dieser Meldungen, ist irgendetwas bei Ihnen furchtbar schief gelaufen. Es kann durchaus sein, daß trotz des Versuchs von TempleMon, die Tastatur zurückzusetzen, eine Eingabe nicht möglich ist.

Deweiteren können noch die beiden folgenden Meldungen erscheinen, wenn Systemvektoren verbogen sind:

- MFP-Tastaturinterruptvektor restauriert
- IKBDSYS-Tastaturinterruptvektor restauriert

12.5 Der Fall "Division durch Null"

TOS besitzt seit Beginn die Unart, bei einer Exception "Division durch Null" keine Bomben zu zeigen, sondern das fehlerauslösende Programm einfach ohne Fehlerbehandlung fortfahren zu lassen. TEMPLEMON macht Sie natürlich auf diesen Fehler aufmerksam; das ist ja auch seine Aufgabe. Nun haben aber einige Programmautoren aus Ihren Programmen diesen Fehler nicht herausgenommen, und diese Programme haben fehlerhaft Verbreitung gefunden. Daher kann es störend sein, wenn TEMPLEMON ständig diesen Fehler präsentiert. Deshalb können Sie diese Fehlermeldung unterdrücken. Sie müssen dazu nur eine Patchvariable einstellen. Wir möchten aber an dieser Stelle ganz deutlich hinweisen: ein Programm das durch Null teilt, ist fehlerhaft!

12.6 Der Fall "MOVE von SR"

Mit dem Erscheinen des Prozessors 68010 wurde der lesende Zugriff auf das Statusregister im User Modus verboten. Da jedoch der Atari ST mit einem 68000 Prozessor ausgestattet ist, und dort diese Zugriffsart nicht verboten ist, hat sich bei der Umstellung auf die anderen Prozessoren der Familie dieser Befehl als Problemfall herausgestellt. Im TOS des TT's wird beim Auftreten einer Privilegsverletzung, die durch diesen Befehl ausgelöst wurde, dieser Befehl im Supervisormodus nachgeholt. TEMPLEMON testet beim Auftreten einer Privilegsverletzung ebenfalls ab, ob dieser Befehl Ursache war, und verzweigt gegebenfalls in der XBRA Kette weiter, wo er hoffentlich das TOS antrifft. Diese Option läßt sich in den Patchvariablen abschalten.

12.7 Bildschirmschoner

Über ihren Nutzen kann man streiten, Probleme mit TEMPLEMON und einem Bildschirmschoner können aber auftreten, wenn der Bildschirmschoner, z.B. durch Umprogrammieren der Farbpalette zugeschlagen hatte, das Bild also dunkel war, und plötzlich ein Fehler auftritt, der TEMPLEMON veranlaßt, aktiv zu werden. Was dann passiert, ist folgendes: TEMPLEMON sichert den abgedunkelten Bildschirm und macht seinen eigenen sichtbar. Nun drücken Sie eine Taste, worauf der Bildschirmschoner glaubt, er müsse den GEM Bildschirm wieder sichtbar machen, jedoch nun die Palettenfarben des TEMPLEMON Bildschirmes ändert. Wenn Sie nun auf den GEM Bildschirm zurückschalten, oder TEMPLEMON verlassen, wird wieder der abgedunkelte Bildschirm hergestellt. Alles in allem keine befriedigende Sache. TEMPLEMON bietet die normalerweise abgeschaltete Option an, Bildschirmschoner "aufzuwecken". Das bedeutet, er simuliert einen Tastendruck, indem er den IKBDSYS Softwarevektor anspringt, bevor er den TEMPLEMON Bildschirm anzeigt. Nach Rückkehr aus dem IKBDSYS wartet er noch zwei Bildrückläufe ab, damit der Bildschirmschoner die alte Farbpalette restaurieren kann.

12.8 TEMPLEMON erweitern

Beim Start von TEMPLEMON wird automatisch die Datei '`TMONHELP.EXE`' als Programm gestartet. Zunächst wird im aktuellen Verzeichnis gesucht, dann im Auto Ordner des aktuellen Laufwerkes. Dieses Modul wird durch die PEXEC Modi 3 und 4 gestartet. D.h., daß der Speicher dieser Module TEMPLEMON gehört. Trotzdem sollten die Module bei erfolgreicher Installation sich resident beenden. Über den Returncode beim Beenden wird TEMPLEMON der Erfolg der Installation des Moduls gemeldet. Ist der Returncode nicht Null, so geht TEMPLEMON von einem Mißerfolg aus. In diesem Falle gibt TEMPLEMON das Environment und die Basepage des Moduls frei, und initialisiert die TEMPLEMON Zeiger der User-Traceroutine, der Online Hilfe Funktion und der Label und Makroverarbeitung nochmals mit Null. Das Modul seinerseits sollte sich in diesem Falle nicht resident beenden.

Kapitel 13

Nachwort

So das wär's wieder (Endlich. Man kann es schon nicht mehr lesen [d.S.]). Es fallen zwar noch eine Menge Tips ein ["wie immer!" d.S.], andererseits ist es Zeit, die Anleitungen auszuliefern (Wir glauben sowieso nicht, daß irgendwer diesen ganzen Kram komplett durchliest ["und ohne größeren intellelen Schaden übersteht" d.S.]).

Wir hoffen, 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, sind wir gerne bereit, Ihnen zu helfen. Bitte richten Sie Ihre Fragen schriftlich vorzugsweise an Johannes, und vergessen Sie Ihre Telefonnummer nicht, damit Sie gegebenfalls zurückgerufen werden können. Falls Sie eine Frage an Thomas stellen, eventuell im Zusammenhang mit MEGAMAX MODULA-2, dann probieren Sie es unter der folgenden Telefonnummer: 089 / 2730166 [Thomas Tempelmann].

Eine erfolgreiche Fehlersuche und noch mehr fehlerfreie Programme wünschen Ihnen

Thomas Tempelmann ["deutsch auszusprechen", d.S.] und

Johannes Hill.

Anhang A

Die Patchvariablen TempleMons

TempleMon ist teilweise an eigene Bedürfnisse anpaßbar. Dies betrifft die Tastenkombinationen der durch TempleMon angebotenen Funktionen, die Farbpalette des TempleMon Bildschirms, den Speicherbedarf des Bildschirms und des Eingabezeilenpuffers, das Verhalten TempleMons bei verschiedenen Exceptions usw. Sollten Sie im Besitz eines Konfigurationsprogrammes für TempleMon sein, so wird dieses diese Einstellungen bieten und fest in TempleMon verankern. Dies funktioniert relativ einfach. Innerhalb TempleMons sind ein paar besondere Variablen mit konstantem Inhalt definiert, die im Programmfile an einer bestimmten Stelle stehen. Ändert man diese konstanten Werte und speichert diese Änderungen in das Programmfile TempleMons zurück, kann man damit sein Verhalten beeinflußen. Diese Variablen heißen Patchvariablen. Sie stehen relativ nahe am Anfang des Programmfiles. Laden Sie dazu das Programm TEMPLMON.PRG binär mit TempleMon selbst oder einem sonstigen Diskettenmonitor. Im folgenden sind alle Zahlen in der hexadezimalen Darstellung aufgeführt. Wenn Sie von Programmanfang aus sich das File als ASCII Zeichen betrachten, werden Sie innerhalb der ersten 512 Bytes den String 'TemplemonVar' erkennen können. Er markiert den Start dieser Patchvariablen. Am Ende der Patchvariablen werden Sie den String 'TemplemonVarEnd' finden. Er markiert das Ende der Patchvariablen. Die nachstehende Tabelle listet alle Patchvariablen auf. Bitte bedenken Sie, daß diese Liste nur für die Version 2.0 TempleMons gültig ist. Die Spalte "Offset" bezeichnet die Entfernung in Bytes direkt nach dem Startstring 'TemplemonVar'.

Offset	Größe	Name	Bedeutung
$0	LONG	StackSize	Stapelgröße TEMPLEMONS
$4	LONG	BlinkRate	Blinkfrequenz des Cursors
$8	WORD	IRMask	Interruptmaske in TEMPLEMON
$A	LONG	DumpDelay	Verzögerungszeit bei der Bildschirmausgabe ab 68020 und 32KB großen Bildschirm
$E	LONG	DumpDelay2	Verzögerungszeit bei der Bildschirmausgabe ab 68020 und Bildschirmen mit mehr als 32KB Größe
$12	WORD	ExtFlag	Flag ohne weitere Bedeutung mehr. Sollte immer auf 0 sein
Tastencodes allgemein			
$14	BYTE	tmon_shiftkeys	Status der Tastaturumschalttasten SHIFT, ALTERNATE und CONTROL für Tastaturaufruf
$15	BYTE	tmon_scancode	Scancode für Tastaturaufruf
$16	BYTE	reset_kalt_shiftkeys	Status der Tastaturumschalttasten für Kaltstart über Tastatur
$17	BYTE	reset_warm_shiftkeys	Status der Tastaturumschalttasten für Warmstart über Tastatur
$18	BYTE	reset_scancode	Scancode für Reset über Tastatur (Kalt- und Warmstart)
$19	BYTE	constant_1	Füllbyte
HyperScreen Bildschirmkonstanten in ST Hoch			
$1A	WORD	m_VideoBytes	Anzahl Bytes pro Zeile
$1C	LONG	m_VideoMemory	Größe des Bildschirms in Bytes
$20	LONG	tmon_offset_mon	Offset des von TEMPLEMON genutzten Bildschirmes zur linken oberen Ende - in Bytes
HyperScreen Bildschirmkonstanten in ST Mittel			
$24	WORD	c_VideoBytes	Anzahl Bytes pro Zeile
$26	LONG	c_VideoMemory	Größe des Bildschirms in Bytes
$2A	LONG	tmon_offset_col	Offset des von TEMPLEMON genutzten Bildschirmes zur linken oberen Ende - in Bytes

Offset	Größe	Name	Bedeutung
Bildschirmkonstanten für TT Hoch			
$2E	WORD	t_VideoBytes	Anzahl Bytes pro Zeile
$30	LONG	t_VideoMemory	Größe des Bildschirms in Bytes
$34	LONG	tmon_offset_tt	Offset des von TEMPLEMON genutzten Bildschirmes zur linken oberen Ende - in Bytes
Shifterresynchronisation			
$38	BYTE	sync_shiftkeys	Status der Tastaturumschalttasten für Shifterresynchronisation
$39	BYTE	sync_scancode	Scancode für Shifterresynchronisation
$3A	WORD	sync_time_mono	Resynchronisationszeit in 10*Takt für Monochrom Auflösung
$3C	WORD	sync_time_mid50	Resynchronisationszeit für Mid 50 Hertz
$3E	WORD	sync_time_mid60	Resynchronisationszeit für Mid 60 Hertz
$40	WORD	sync_time_low50	Resynchronisationszeit für Low 50 Hertz
$42	WORD	sync_time_mid60	Resynchronisationszeit für Low 60 Hertz
$44	BYTE	video_sync_flag	Video Sync Register mit video_sync_byte füllen (<>0)?
$45	BYTE	video_sync_byte	Inhalt des Video Sync Registers, wenn nicht der ursprüngliche Inhalt gesetzt werden soll
$46	BYTE	video_sync_intern	zum Testen auf interne Synchronisation schalten (0 = externe Sync.)?
$47	BYTE	abort_key_mask	Status der Umschalttasten bei Installationsabbruch
$48	BYTE	edit_lines	Anzahl der zu puffernden Eingabezeilen
$49	BYTE	constan_3	Füllbyte
Farbpaletten			
$4A	WORD	mono_0	Farbregister 0 für Monochrom
$4C	WORD	tt_mono_0	Farbregister 254 für Duochrome
$4E	WORD	tt_mono_1	Farbregister 255 für Duochrome
$50	WORD	color_0	Farbregister 0 für Color
$52	WORD	color_1	Farbregister 1 für Color
$54	WORD	color_2	Farbregister 2 für Color

Offset	Größe	Name	Bedeutung
$56	WORD	color_3	Farbregister 3 für Color
$58	BYTE	edit_modus	Editormodus festlegen: $FF = Insertmodus, $00 = Overwritemodus, $7F = Leerzeichen-einfügemodus
$59	BYTE	edit_minimum	Mindestlänge einer Eingabezeile, bevor sie in den Befehlszeilenpuffer eingefügt wird
$5A	BYTE	sync_on_shiftkeys	Status der Umschalttaste für int. Synchronisation
$5B	BYTE	sync_on	Scancode für interne Synchronisation
$5C	BYTE	sync_off_shiftkeys	Status der Umschalttaste für ext. Synchronisation
$5D	BYTE	sync_off	Scancode für externe Synchronisation
$5E	WORD	more_edit	Zusätzliche Bildschirmzeilen für Fullscreen-neditor
$60	WORD	SRPatch	Abfangen von MOVE von SR? (0=TempleMon)
$62	WORD	DIVZPatch	Abfangen von Division durch Null? (0=TempleMon)
$64	WORD	KeyCheck	Keyboardcheck durchführen? (0=Nein)
$66	LONG	XBRATab1	Es folgen alle bekannten XBRA's für Keyboardcheck
$6A	LONG	XBRATab2	
$6E	LONG	XBRATab3	
$72	LONG	XBRATab4	
$76	LONG	XBRATab5	
$7A	LONG	XBRATab6	
$7E	LONG	XBRATab7	
$82	LONG	XBRATab8	
$86	LONG	XBRATab9	
$8A	LONG	XBRATab10	
$8E	LONG	XBRAEnde	Endemarkierung, immer Null!
$92	WORD	no_Atari_OS	Atari TOS oder Modifikation? (0=Atari)
$94	LONG	OS_Start	Startadresse des TOS, wenn nicht von Atari
$98	LONG	OS_Ende	Endadresse des TOS, wenn nicht von Atari

Offset	Größe	Name	Bedeutung
$9C	WORD	virtuell	Virtuelle Speicherverwaltung, die nicht das VBR Register nutzt (<> 0).
$9E	LONG	debugging	Ermöglicht das Debugging der TEMPLEMON Exceptionverarbeitung mit Hilfe eines zweiten TEMPLEMONS. Dabei muß in diesem Langwort für die zu debuggende Exceptionroutine das entsprechende Bit gesetzt sein.
$102	LONG	debug_xbra	XBRA Kennung des zweiten TEMPLEMONS

Anmerkung zur Patchvariablen 'virtuell':

Die folgenden Bits bestimmen, wann in den Speichermanager gesprungen wird (Bit = 1 setzen):

0 Supervisor Violation

1 Write Violation

2 Limit Violation

3 Invalid Descriptor or Page Vault

4 Bus Error during Table Search

Anmerkung zur Patchvariable 'debugging':

Die folgende Übersicht zeigt, welche Bits gesetzt werden müssen, damit in der entsprechende Exceptionroutine als erstes der zweite TEMPLEMON angesprungen wird.

0 Buserror

1 Adresserror

2 Illegal_Instruction

3 Division_by_Zero

4 CHK

5 TRAPV

6 Privilege_Violation

7 Trace

8 Line_F

9 Coprocessor_Protocol_Error

10 Format_Error

11 Uninitialized_Interrupt

12 Spurios_Interrupt

13 unused_TRAP

14 Unordered_Condition

15 Inexact_Result

16 Divide_by_Zero

17 Underflow

18 Operand_Error

19 Overflow

20 Signaling_NAN

21 Unassigned

22 MMU_Configuration_Error

Anhang B

Disassembler durch Programme aufrufen

Der Aufruf des Disassemblers durch fremde Programme ist einfach möglich. Das Disassemblie wird dann nicht auf dem TempleMon Bildschirm geschrieben, sondern in einen internen TempleMon String plaziert, dessen Adresse dem Aufrufer mitgeteilt wird. Dieser String ist Null-terminiert. Nachfolgend ein kleines Beispielprogramm, das die Funktionsweise einfach demonstriert. Bitte beachten, daß das TempleMon Cookie Interface NUR IM SUPERVISOR-MODUS aufgerufen werden darf!!!

```
start:          clr.l   -(SP)
                move.w  #$20,-(SP)       ; in den Supervisormodus schalten
                trap    #1
                addq.l  #6,SP
                movea.l $05A0,A3
                tst.l   $05A0
                beq     prg_abort
search:         cmpi.l  #'TMon',(A3)     ; das TempleMon Cookie suchen
                beq     dotmon
                addq.l  #8,A3
                tst.l   (A3)
                bne.s   search
prg_abort:      clr.w   -(SP)
                trap    #1
dotmon:         moveq   #12,D0           ; Funktionsnummer 12 für
                                         ; Disassembliere
                lea     testcode(PC),A0  ; Adresse des Befehls, der
                                         ; disassembliert werden soll
                movea.l 4(A3),A3         ; Adresse der TempleMon Routine
                                         ; aus Cookie-Jar holen
                moveq   #$13,D1          ; gewünschten CPU Typ festlegen:
                                         ; unteres Nibble die CPU (hier
```

```
                                ; 3=68030), oberes Nibble FPU
                                ; (hier 1=mit FPU Opcode)
                jsr     (A3)    ; TempleMon anspringen
                DC.W $4AFC      ; jetzt kann man sich das Ergebnis
                                ; anschauen (mit Hilfe TempleMon)
                                ; In A1 befindet sich der Pointer
                                ; auf den String, in D0 die
                                ; Befehlsnummer. Sollte D0 eine
                                ; -1.W (Word!) enthalten, hat es
                                ; sich um einen unbekannten Befehl
                                ; gehandelt. Ist in D0 das 15 Bit
                                ; gesetzt und D0 <> -1, wurde bei
                                ; der Befehlsdekodierung eine
                                ; unzulässige effektive Adresse
                                ; gefunden. Durch Löschen dieses
                                ; Bits kommt man an die
                                ; Befehlsnummer heran. Weiterhin
                                ; haben sich A0 und A2 geändert!
                                ; A0 enthält nun die Adresse des
                                ; nächsten Befehles. Würde man die
                                ; die Startadresse des
                                ; disassemblierten Befehls davon
                                ; abziehen, ergäbe sich
                                ; daraus die Befehlslänge in Bytes.
                                ; A2 enthält einen Pointer auf die
                                ; interne TempleMon Codetabelle.
                bra     prg_abort
testcode:       DC.B $F2,$39,$00,$30,$00,$00
                END
```

Anhang C

Ein Online Hilfe Rumpfprogramm

Das nachstehende Rumpfprogramm demonstriert, wie Sie Ihre eigene Hilfefunktion bei TEMPLEMON integrieren. Über die eigentlich Online Hilfe müssen Sie sich jedoch selbst Gedanken machen und sie implementierten. Dies dürfte aber für den geübten Programmierer kein Problem darstellen. Das Beispiel ist in Pure C geschrieben und sehr einfach gehalten.

```
/*****************************************************************/
/*                                                               */
/*                          TMonHelp.C                           */
/*                                                               */
/*                   Online Hilfe für TempleMon                  */
/*                                                               */
/*  Rumpfroutine zur Implementation einer Online Hilfe Funktion in  */
/*  TempleMon. Geschrieben mit Pure C.                           */
/*  (c) 1991 Johannes Hill                                       */
/*                                                               */
/*  Geschichte                                                   */
/*                                                               */
/* Version   Datum      Beschreibung                            */
/*   0.1    20.10.91    Basisversion.                           */
/*                                                               */
/*****************************************************************/

#include <tos.h>

typedef struct
{
     long name;
     char*     (*handler)(long function,long d1,long d2,void *p1, void *p2);
} TMONCOOKIE;
```

```
TMONCOOKIE *tmoncookie;

char text[]={"\n\r\nBeispielhilfe\n\r\n"};

#define p_cookie *(TMONCOOKIE **) 0x5a0L

static TMONCOOKIE *find_cookie(long name)
{
     TMONCOOKIE     *jar;
     long      ssp;
     ssp=Super(0l);                    /* In den Supervisormodus         */
     jar=p_cookie;                     /* Anfangsadresse des Cookie Jars */
     Super((void *) ssp);              /* Und zurück                     */
     if(jar==0) return 0;              /* Kein Cookie Jar installiert    */
     while(jar->name)                  /* 00=ende                        */
          {
          if(jar->name==name)          /* gefunden !!                    */
               return jar;
          jar++;
          }
     return 0;                         /* nicht gefunden (Pech)      */
}

long Hilfe(long shift_keys, char *buffer, int *cursor_position)
{
/* Hier ist die eigentliche Hilfefunktion zu implementieren. buffer  */
/* zeigt auf den 80 Zeichen langen, nicht Null terminierten String,  */
/* der die eingetippte Befehlszeile enthält.                         */
/* cursor_position enthält die Position des Cursors relativ          */
/* zu *buffer.                                                       */
/* shift_keys enthält den momentanen Status der Shift, Control und   */
/* Alternate Taste.                                                  */
/* Wenn Sie nun *buffer auslesen, können Sie entscheiden, zu         */
/* Befehl der Anwender eine Hilfe geboten haben möchte. Dadurch,     */
/* Sie auch den Status der SHIFT, CONTROL und ALTERNATE Taste mit    */
/* überreicht bekommen, könnten Sie sogar verschiedene Hilfemodi     */
/* anbieten.                                                         */

/* Hier folgt ein kleines Beispiel, das lediglich den Text           */
/* Beispielhilfe auf den TempleMon Bildschirm bringt. Wenn man sich  */
/* den alten Hilfefunktionszeiger bei der Installation gemerkt       */
/* hätte, könnte man statt bunmittelar zu TempleMon wieder           */
/* zurückzukehren, die vorher installierte Hilfefunktion anspringen  /*
/* So könnte man mehrere Hilfeprogramme installieren, die sich nicht /*
/* gegenseitig aushängen.                                            /*
```

```
    int cursor_pos;

    cursor_pos = *cursor_position;     /* alte Cursorposition merken /*
    *cursor_position = 79;             /* Cursor ans Zeilenende      /*
    (tmoncookie->handler)(15,0,0,text,0);    /* Text asugeben        /*
    *cursor_position = cursor_pos;    /* Cursorposition restaurieren /*
    return 0;                         /* Die Befehlszeile nicht auto-
                                        matisch ausführen lassen    /*
}

void main(void)
{
     void *ssp;

     tmoncookie=find_cookie('TMon');
     if(tmoncookie)
          {
          ssp=(void *) Super(0L);
          (tmoncookie->handler)(13,0,0,Hilfe,0);
          Super(ssp);
          }
     Ptermres(_BasPag->p_tlen+_BasPag->p_dlen+_BasPag->p_blen+0x100,0);
}
```

Anhang D

User-Trace in Megamax Modula-2

```
MODULE Trace;

(*****************************************************************
 * Dieses Programm dient als Debug-Hilfe zum Debugger TEMPLEMON.
 * Es wurde mit dem MEGAMAX Modula-2 Entwicklungssystem erstellt.
 * Das Programm (ohne den besonders markierten Programmabschnitte
 * fuer das Beispielprogramm TRACE.MOS) kann als Rumpf fuer alle
 * weiteren User-Trace Routinen dienen.
 *****************************************************************)

(*$ E MOS   Suffix: Kennzeichnung als TOS-Applikation *)

(* Importe *)

FROM ModCtrl IMPORT InstallModule, ReleaseModule, FirstModuleStart;

FROM MOSGlobals IMPORT MemArea;

FROM Terminal IMPORT WriteLn, WriteString, Read;

FROM Calls IMPORT CallSupervisor;

FROM SYSTEM IMPORT ADDRESS, ASSEMBLER;


VAR  Cookie : ADDRESS;

(* spezifische Konstante fuer Zugriffe auf die Prozessorregister *)

CONST   RegD0    = 04;
```

```
RegD1    = 08;
RegD2    = 12;
RegD3    = 16;
RegD4    = 20;
RegD5    = 24;
RegD6    = 28;
RegD7    = 32;
RegA0    = 36;
RegA1    = 40;
RegA2    = 44;
RegA3    = 48;
RegA4    = 52;
RegA5    = 56;
RegA6    = 60;
RegA7    = 64;
RegSR    = 02;
RegPC    = 68;
RegUSP   = 80;
RegSSP   = 84;
RegBEV   = 80;  (* Pseudoregister *)
RegAEV   = 84;  (* Pseudoregister *)
RegSFC   = 88;  (* AB HIER NUR BEI 68010 ODER HÖHER *)
RegDFC   = 92;
RegVBR   = 96;
RegMSP   = 100; (* AB HIER NUR BEI 68020 ODER HÖHER *)
RegISP   = 104;
RegCAAR  = 108; (* NICHT BEI 68040 *)
RegCACR  = 112;
RegTT0   = 116; (* NUR FÜR 68030 *)
RegTT1   = 120; (* NUR FÜR 68030 *)
RegTC    = 124; (* FÜR 68030 & 68040 *)
RegPSR   = 128; (* FÜR 68030 & 68040 *)
RegCRP   = 132; (* 64 BIT BREIT, NUR FÜR 68030 *)
RegSRP   = 140; (* 64 BIT BREIT BEI 68030, 32 BIT BREIT BEI 68040 *)
RegFP0   = 148; (* IN EXTENDED PRECESITION, AB HIER FÜR 68881/882
                                                        & 68040 *)
RegFP1   = 160;
RegFP2   = 172;
RegFP3   = 184;
RegFP4   = 196;
RegFP5   = 208;
RegFP6   = 220;
RegFP7   = 232;
RegFPCR  = 244;
RegFPSR  = 248;
RegFPIAR = 252;
```

```
        RegDTT0  = 256; (* AB HIER NUR BEI 68040 *)
        RegDTT1  = 260;
        RegITT0  = 264;
        RegURP   = 268;


(******************** User - Trace Routine ***************************)

PROCEDURE usrTrc;
  (*$L-*)
  BEGIN
    ASSEMBLER
        ;Hier ist der Einsprungspunkt fuer die User-Trace Routine.
        ;Hier koennen Abfragen in Assembler programmiert werden.
        ;Die augenblicklichen Register A0 und D7 duerfen nicht
        ;veraendert werden; in D0.W wird Nicht-Null oder Null zu-
        ;rueckgegeben, je nachdem, ob das Tracing abgebrochen wer-
        ;densoll oder ob weitergetraced werden soll.
        ;Auf die im Debugger zwischengespeicherten Register der
        ;CPU fuer die naechste auszufuehrende Instruktion kann
        ;mit den obigen Definitionen (Reg..) zugegriffen werden.
        ;Die Register (Reg..) koennen auch veraendert werden.
        ;Dieses Programm wird immer im Supervisormodus aufgerufen.
        ;
        ;Z.B. kann eine Ueberpruefung, ob als naechste Instruktion
        ;ein Line-A Aufruf stattfindet, folgendermassen program-
        ;miert werden :
        ;
        ; CLR.W   D0
        ; MOVE.L  RegPC(A0),A1  ;PC laden
        ; MOVE.W  (A1),D1       ;Instruktion laden
        ; ANDI.W  #0xF000,D1    ;oberstes Nibble der Instr. maskieren
        ; CMPI.W  #0xA000,D1    ;Line-A Instruktion ?
        ; SEQ     D0            ;je nachdem D0.B auf 0 oder -1 setzen
        ;

        ;**** Beispiel-Text f. TRACE.MOS ******

        CLR.W   D0
        MOVE.L  $44E,A1      ;Adr. des Bildschirmbereichs
        TST.W   640(A1)      ;Sind die Bits der ersten 8 Spalten
                             ;in der zweiten Zeile geloescht ?
        SNE     D0           ;Wenn nicht, dann Stop
    END
  END usrTrc;
  (*$L=*)
```

```
PROCEDURE installTrace (Trace: ADDRESS);
  BEGIN
    ASSEMBLER
            MOVEQ    #2,D0
            MOVEA.L  -(A3),A0
            MOVE.L   Cookie,A1
            JSR      (A1)
    END
  END installTrace;

PROCEDURE deinstallTrace (dummy: ADDRESS);
  BEGIN
    ASSEMBLER
            MOVEQ    #2,D0
            SUBA.L   A0,A0
            MOVE.L   Cookie,A1
            JSR      (A1)
    END
  END deinstallTrace;

PROCEDURE init;
  BEGIN
    ASSEMBLER
            MOVEQ     #0,D0
            MOVE.L    D0,Cookie
            MOVEA.L   $5A0,A0
            TST.L     $5A0
            BEQ       abort
search:     CMPI.L    #'TMon',(A0)
            BEQ       dotmon
            ADDQ.L    #8,A0
            TST.L     (A0)
            BNE.S     search
            BRA.S     abort
dotmon:     MOVEA.L   4(A0),D0
            MOVE.L    D0,Cookie
abort:
    END
  END init;

PROCEDURE restore;
  VAR wsp: MemArea;
  BEGIN
    wsp.bottom:= NIL;
```

```
    CallSupervisor (deinstallTrace, NIL, wsp); (* Routine deinstallieren *)
  END restore;


PROCEDURE wait;
  VAR c: CHAR;
  BEGIN
    WriteLn;
    WriteString ('Bitte eine Taste...');
    WriteLn;
    Read (c)
  END wait;

VAR wsp: MemArea;

BEGIN (* main *)
  wsp.bottom:= NIL;
  IF FirstModuleStart () THEN
    (* Programm beenden, aber Speicher dieses Programms nicht freigeben *)
    InstallModule (restore, wsp);
    CallSupervisor (init, NIL, wsp);         (* TempleMon Cookie suchen    *)
    IF Cookie <> LONG(0) THEN
      wsp.bootom;=NIL;
      CallSupervisor (installTrace, ADDRESS(usrTrc), wsp);
                                             (* Trace-Routine installieren *)
      WriteLn;
      WriteString ('TEMPLEMON User-Trace ist installiert.');
      WriteLn;
      WriteString ('Druecke gleich CNTRL/HELP, um den Monitor aufzurufen,');
      WriteLn;
      WriteString ('sofern er bereits einmal gestartet wurde. Gebe dann');
      WriteLn;
      WriteString ('"T+", danach "G" ein. Druecke dann "A". Nun ist die');
      WriteLn;
      WriteString ('Trace-Routine aktiv und der Monitor meldet sich');
      WriteLn;
      WriteString ('zurueck, soblad die Maus linke obere Ecke des');
      WriteLn;
      WriteString ('Bildschirms erreicht.');
      WriteLn;
      wait
    ELSE
      WriteString ('TEMPLEMON nicht gefunden. Installation abgebrochen.');
      WriteLn;
      wait
    END
```

```
  ELSE
    (* Speicher dieses Programms freigeben *)
    CallSupervisor (deinstallTrace, NIL, wsp); (* Routine deinstallieren *)
    ReleaseModule;
    WriteLn;
    WriteString ('TEMPLEMON User-Trace ist nun entfernt.');
    WriteLn;
    wait
  END
END Trace.
```

Anhang E

User-Trace in Assembler

Ein komplettes Programmbeispiel einer User-Trace Funktion, die dann das Tracing abbricht, wenn das Highbyte von A5 nicht 0 ist. Geschrieben in Assembler:

```
; Ein Beispiel einer Usertrace Funktion.
;
; Als Erstes mal die Usertrace Funktion bei TempleMon installieren:

start:          clr.l   -(SP)           ; In den Supervisormodus schalten
                move.w  #$20,-(SP)
                trap    #1
                addq.l  #6,SP
                movea.l $05A0,A3        ; TempleMon Cookie suchen
                tst.l   $05A0
                beq     prg_abort
search:         cmpi.l  #'TMon',(A3)
                beq     dotmon          ; gefunden!
                addq.l  #8,A3
                tst.l   (A3)
                bne.s   search          ; Cookie Jar bis Ende absuchen

prg_abort:      pea     abort           ; nix gefunden! Programm abbrechen
                move.w  #9,-(SP)
                trap    #1
                addq.l  #6,SP
                clr.w   -(SP)
                trap    #1
dotmon:         movea.l 4(A3),A3        ; Adresse Funktionshändlers
                move.l  a3,tmon_routine ; aus Cookie Eintrag holen
                lea     usertracecode(PC),A0
```

```
                moveq   #2,D0
                jsr     (A3)
                ; Diesen Händler, zum Installieren der neuen
                ; UserTrace Funktion aufrufen. Nach Aufruf
                ; steht in A0 die Adresse der alten Routine.
                ; Ist die Adresse der UserTrace Funktion 0,
                ; springt TempleMon sie nicht an!
                ; Mit A0=0 kann man also die UserTrace
                ; Funktion deaktivieren, mit A0=-1.L kann man
                ; die aktuelle Adresse der UserTrace Funktion
                ; erfragen, ohne daß diese verändert wird.

                pea     ok    ; Programm resident installieren.
                move.w  #9,-(SP)
                trap    #1
                addq.l  #6,SP
                move.w  #0,-(SP)
                move.l  #code_ende-start+256,-(SP)
                move.w  #$31,-(SP)
                trap    #1
                addq.l  #8,SP
                pea     fehler
                move.w  #9,-(SP)
                trap    #1
                addq.l  #6,SP
                movea.l tmon_routine(PC),A3
                moveq   #2,D0
                suba.l  A0,A0
                jsr     (A3)
                bra     prg_abort
;
; Hier nun die UserTrace Routine einhängen:
;
usertracecode:
; Die UserTrace Routine wird im Supervisormodus aufgerufen. Alle
; Register bis auf D0 dürfen nicht verändert werden! Die UserTrace
; Routine wird mit einem RTS und einem Returnwert in D0.B verlassen.
; Dabei wird das Tracen von TempleMon abgebrochen, falls D0 nicht Null
; ist. In A0 wird der Pointer auf den Registersatz der Registersicherung
; übergeben, die beim Tracen des augenblicklichen Befehles angelegt
; wurde. Eine Veränderung des Registersatzes wirkt sich nach außen aus!
; Die einzelnen Register sind mit dem folgenden Displacement zu A0 zu
; erreichen:
; SR    = 2
; D0    = 4
; D1    = 8
```

```
; D2    = 12
; D3    = 16
; D4    = 20
; D5    = 24
; D6    = 28
; D7    = 32
; A0    = 36
; A1    = 40
; A2    = 44
; A3    = 48
; A4    = 52
; A5    = 56
; A6    = 60
; A7    = 64
; PC    = 68
; USP   = 72
; SSP   = 76
; BEV   = 80    Pseudoregister
; AEV   = 84    Pseudoregister
; SFC   = 88    AB HIER NUR BEI 68010 ODER HÖHER
; DFC   = 92
; VBR   = 96
; MSP   = 100   AB HIER NUR BEI 68020 ODER HÖHER
; ISP   = 104
; CAAR  = 108
; CACR  = 112
; TT0   = 116   NUR FÜR 68030
; TT1   = 120   NUR FÜR 68030
; TC    = 124   FÜR 68030 & 68040
; PSR   = 128   FÜR 68030 & 68040
; CRP   = 132    64 BIT BREIT, NUR FÜR 68030
; SRP   = 140    64 BIT BREIT BEI 68030, 32 BIT BREIT BEI 68040
; FP0   = 148    IN EXTENDED PRECESITION, AB HIER FÜR 68881/882 & 68040
; FP1   = 160
; FP2   = 172
; FP3   = 184
; FP4   = 196
; FP5   = 208
; FP6   = 220
; FP7   = 232
; FPCR  = 244
; FPSR  = 248
; FPIAR = 252   NICHT BEI 68040
; DTT0  = 256  AB HIER NUR BEI 68040
; DTT1  = 260
; ITT0  = 264
```

```
; URP   = 268
;
; Hier nun ein Beispiel, daß solange traced, bis das Highbyte von A5
; ungleich Null ist:
;
               move.l  56(A0),D0        ; A5 holen
               andi.l  #$FF000000,D0    ; Highbyte ausmaskieren
               bne     error            ; ist es Null? nein, dann Fehler
               moveq   #0,D0            ; weiter tracen
               rts
error:         moveq   #-1,D0           ; nicht mehr weiter tracen
               rts
;
; Es folgt nun der Datenbereich:
;
tmon_routine:  dc.l    0
abort:         dc.b    13,10,'UserTrace Installation abgebrochen!'
               dc.b    13,10
ok:            dc.b    13,10,'UserTrace Funktion bei TempleMon '
               dc.b    'angemeldet!',13,10
fehler:        dc.b    13,10,'Fehler beim residenten Beenden '
               dc.b    'dieses Programmes.'
               dc.b    ' UserTrace Funktion von TempleMon',13,10
               dc.b    'wieder abgemeldet!'
code_ende:
               END
```

Anhang F

Die von TempleMon abgefangenen Exceptions

Die nachfolgende Tabelle zeigt Ihnen die von TempleMon abgefangenen Exceptions, und die jeweilige Meldung TempleMons. Die Exception Trace wird zum Tracen benutzt, daher ist für sie in der Tabelle keine Meldung aufgeführt.

Bitte beachten Sie, daß einige Exceptions nur bei Vorhandensein eines bestimmten Prozessortypes abgefangen, und auf Exceptionroutinen TempleMons geleitet werden. Diese Exceptions sind mit einer Fußnote gekennzeichnet.

Exception	Vektoradresse	Bezeichnung	TempleMon Meldung
$2	$8	Buserror	BUS ERROR
$3	$C	Addresserror	ADDRESS ERROR
$4	$10	Illegal	Illegale Instruktion
$5	$14	Division by zero	Division durch Null
$6	$18	CHK	CHK - Instruktion
$7	$1C	TRAPV, TRAPcc, FTRAPcc	TRAPV, TRAPcc, FTRAPcc - Instruktion
$8	$20	Privilege violation	Privilegs-Verletzung
$9	$24	Trace	-
$B	$2C	Line-F[1]	Line F Code
$D	$34	Coprocessor protocol violation[2]	Coprocessor Protocol Error
$E	$38	Format error[3]	Format Error
$F	$3C	Uninitialized interrupt[4]	Nicht initialisierter Interrupt

Fortsetzung auf der nächsten Seite

Fortsetzung von vorheriger Seite			
Exception	Vektoradresse	Bezeichnung	TempleMon Meldung
$18	$60	Spurios interrupt	Illegaler Interrupt
$19	$64	Interrupt 1	Direktaufruf
$1B	$6C	Interrupt 3	Interrupt 3
$1D	$74	Interrupt 5	Interrupt 5
$1E	$7C	Interrupt 7	Interrupt 7
$20	$80	Trap #0[5]	Trap 0
$23	$8C	Trap #3[5]	Trap 3
$24	$90	Trap #4[5]	Trap 4
$25	$94	Trap #5[5]	Trap 5
$26	$98	Trap #6[5]	Trap 6
$27	$9C	Trap #7[5]	Trap 7
$28	$A0	Trap #8[5]	Trap 8
$29	$A4	Trap #9[5]	Trap 9
$2A	$A8	Trap #10[5]	Trap A
$2B	$AC	Trap #11[5]	Trap B
$2C	$B0	Trap #12[5]	Trap C
$2F	$BC	Trap #15[5]	Trap F
$30	$C0	Branch or set unordered condition[6]	FPU : Unordered Condition
$31	$C4	Inexact result[6]	FPU : Ungenaues Ergebnis
$32	$C8	Divide by zero[6]	FPU : Division durch Null
$33	$CC	Underflow error[6]	FPU : Unterlauf
$34	$D0	Operand error[6]	FPU : Operanden Fehler
$35	$D4	Overflow error[6]	FPU : Überlauf
$36	$D8	Signaling NAN[6]	FPU : Keine reelle Zahl
$37	$DC	Unimplemented data type[7]	FPU : Nicht implementierter Datentyp
$38	$D0	MMU configuration error[8]	MMU Configuration Error
$46	$118	MFP keyboard/MIDI ACIA	Tastaturaufruf
-	-	IKBDSYS keyboard vector	Tastaturaufruf

[1] Nur wenn Vektor ins TOS zeigt und CPU 68020/68030/68040
[2] CPU 68020/68030
[3] CPU 68010/68020/68030/68040
[4] CPU 68020/68030/68040
[5] Nur wenn Vektor ins TOS zeigt
[6] FPU 68881/68882 oder CPU 68040
[7] CPU 68040
[8] CPU 68030/68040

Anhang G

Die Befehle der Motorola 68000 Familie

Die nachfolgende Tabelle ist sortiert nach der internen Nummerierung der Befehle der 68000 Familie durch TempleMon. Danach folgt die mnemnonische Darstellung, die interne TempleMon Nummer des Befehlsformates, und dann die Übersicht über die Verfügbarkeit auf den verschiedenen Prozessoren.

Nr.	Befehl	Format	68000	68010	68020	68030	68040	68881/2
1	MOVEP	17	×	×	×	×	×	-
2	ORI	14	×	×	×	×	×	-
3	ANDI	14	×	×	×	×	×	-
4	SUBI	14	×	×	×	×	×	-
5	ADDI	14	×	×	×	×	×	-
6	MOVES	29	-	×	×	×	×	-
7	EORI	14	×	×	×	×	×	-
8	CMPI	14	×	×	×	×	×	-
9	BTST	13	×	×	×	×	×	-
10	BCHG	13	×	×	×	×	×	-
11	BCLR	13	×	×	×	×	×	-
12	BSET	13	×	×	×	×	×	-
13	CHK2	32	-	-	×	×	×	-
14	CMP2	32	-	-	×	×	×	-
15	CAS	30	-	-	×	×	×	-
16	CAS2	31	-	-	×	×	×	-
17	RTM	15	-	-	×	-	-	-
18	CALLM	29	-	-	×	-	-	-
19	MOVE	2	×	×	×	×	×	-
20	RESET	0	×	×	×	×	×	-

Fortsetzung auf der nächsten Seite

Fortsetzung von vorheriger Seite

Nr.	Befehl	Format	68000	68010	68020	68030	68040	68881/2
21	NOP	0	×	×	×	×	×	-
22	STOP	27	×	×	×	×	×	-
23	RTE	0	×	×	×	×	×	-
24	RTD	0	-	×	×	×	×	-
25	RTS	0	×	×	×	×	×	-
26	TRAPV	0	×	×	×	×	×	-
27	RTR	0	×	×	×	×	×	-
28	ILLEGAL	0	×	×	×	×	×	-
29	MOVEC	36	-	×	×	×	×	-
29	BKPT	41	-	×	×	×	×	-
30	SWAP	1	×	×	×	×	×	-
31	EXT.W	18	×	×	×	×	×	-
32	EXT.L	18	×	×	×	×	×	-
33	EXTB.L	18	-	-	×	×	×	-
34	UNLK	15	×	×	×	×	×	-
35	LINK (W)	2	×	×	×	×	×	-
36	LINK (L)	39	-	-	×	×	×	-
37	MOVE USP	22	×	×	×	×	×	-
38	TRAP	21	×	×	×	×	×	-
39	MULU.L MULS.L	33	-	-	×	×	×	-
40	DIVU.L DIVS.L	33	-	-	×	×	×	-
41	JSR	1	×	×	×	×	×	-
42	JMP	1	×	×	×	×	×	-
43	MOVE von SR	23	×	×	×	×	×	-
44	MOVE zu CCR	25	×	×	×	×	×	-
45	MOVE von CCR	40	-	×	×	×	×	-
46	MOVE zu SR	24	×	×	×	×	×	-
47	NBCD	1	×	×	×	×	×	-
48	PEA	1	×	×	×	×	×	-
49	TAS	1	×	×	×	×	×	-
50	MOVEM REG zu EA	19	×	×	×	×	×	-
51	MOVEM EA zu REG	19	×	×	×	×	×	-
52	NEGX	8	×	×	×	×	×	-
53	CLR	8	×	×	×	×	×	-
54	NEG	8	×	×	×	×	×	-
55	NOT	8	×	×	×	×	×	-
56	TST	8	×	×	×	×	×	-

Fortsetzung auf der nächsten Seite

Fortsetzung von vorheriger Seite

Nr.	Befehl	Format	68000	68010	68020	68030	68040	68881/2
57	CHK	9	×	×	×	×	×	-
58	LEA	26	×	×	×	×	×	-
59	DBcc	6	×	×	×	×	×	-
60	TRAPcc	37	-	-	×	×	×	-
61	Scc	4	×	×	×	×	×	-
62	ADDQ	5	×	×	×	×	×	-
63	SUBQ	5	×	×	×	×	×	-
64	Bcc, BRA, BSR	6	×	×	×	×	×	-
65	MOVEQ	7	×	×	×	×	×	-
66	SBCD	11	×	×	×	×	×	-
67	PACK	34	-	-	×	×	×	-
68	UNPACK	34	-	-	×	×	×	-
69	DIVU	9	×	×	×	×	×	-
70	DIVS	9	×	×	×	×	×	-
71	OR	10	×	×	×	×	×	-
72	SUBX	11	×	×	×	×	×	-
73	SUB	10	×	×	×	×	×	-
74	CMP	10	×	×	×	×	×	-
75	CMPM	28	×	×	×	×	×	-
76	EOR	10	×	×	×	×	×	-
77	ABCD	11	×	×	×	×	×	-
78	EXG	12	×	×	×	×	×	-
79	EXG	12	×	×	×	×	×	-
80	MULU	9	×	×	×	×	×	-
81	MULS	9	×	×	×	×	×	-
82	AND	10	×	×	×	×	×	-
83	ADDX	11	×	×	×	×	×	-
84	ADD	10	×	×	×	×	×	-
85	BFCHG	35	-	-	×	×	×	-
86	BFCLR	35	-	-	×	×	×	-
87	BFINS	35	-	-	×	×	×	-
88	BFFFO	35	-	-	×	×	×	-
89	BFSET	35	-	-	×	×	×	-
90	BFTST	35	-	-	×	×	×	-
91	BFEXTS	35	-	-	×	×	×	-
92	BFEXTU	35	-	-	×	×	×	-
93	ASR, ASL	16	×	×	×	×	×	-

Fortsetzung auf der nächsten Seite

Fortsetzung von vorheriger Seite

Nr.	Befehl	Format	68000	68010	68020	68030	68040	68881/2
94	LSR, LSL	16	×	×	×	×	×	-
95	ROXR, ROXL	16	×	×	×	×	×	-
96	ROR, ROL	16	×	×	×	×	×	-
97	PMOVE TT0, TT1	45	-	-	-	×	-	-
98	PLOAD	44	-	-	-	×	-	-
99	PFLUSH	43	-	-	-	×	-	-
100	PMOVE SRP, CRP ... TC, PSR	45	-	-	-	×	-	-
101	PTEST	42	-	-	-	×	-	-
102	CPUSH	46	-	-	-	-	×	-
103	CINV	46	-	-	-	-	×	-
104	PTEST	48	-	-	-	-	×	-
105	PFLUSH	47	-	-	-	-	×	-
106	MOVE16	49	-	-	-	-	×	-
107	FMOVE zu FPn	50	-	-	-	-	×	×
108	FINT	50	-	-	-	-	×*	×
109	FSINH	50	-	-	-	-	×*	×
110	FINTRZ	50	-	-	-	-	×*	×
111	FSQRT	50	-	-	-	-	×	×
112	FLOGNP1	50	-	-	-	-	×*	×
113	FETOXM1	50	-	-	-	-	×*	×
114	FTANH	50	-	-	-	-	×*	×
115	FATAN	50	-	-	-	-	×*	×
116	FASIN	50	-	-	-	-	×*	×
117	FATANH	50	-	-	-	-	×*	×
118	FSIN	50	-	-	-	-	×*	×
119	FTAN	50	-	-	-	-	×*	×
120	FETOX	50	-	-	-	-	×*	×
121	FTWOTOX	50	-	-	-	-	×*	×
122	FTENTOX	50	-	-	-	-	×*	×
123	FLOGN	50	-	-	-	-	×*	×
124	FLOG10	50	-	-	-	-	×*	×
125	FLOG2	50	-	-	-	-	×*	×
126	FABS	50	-	-	-	-	×	×
127	FCOSH	50	-	-	-	-	×*	×
128	FNEG	50	-	-	-	-	×	×
129	FACOS	50	-	-	-	-	×*	×

Fortsetzung auf der nächsten Seite

Fortsetzung von vorheriger Seite

Nr.	Befehl	Format	68000	68010	68020	68030	68040	68881/2
130	FCOS	50	-	-	-	-	×*	×
131	FGETEXP	50	-	-	-	-	×*	×
132	FGETMAN	50	-	-	-	-	×*	×
133	FDIV	50	-	-	-	-	×	×
134	FMOD	50	-	-	-	-	×*	×
135	FADD	50	-	-	-	-	×	×
136	FMUL	50	-	-	-	-	×	×
137	FSGLDIV	50	-	-	-	-	×*	×
138	FREM	50	-	-	-	-	×*	×
139	FSCALE	50	-	-	-	-	×*	×
140	FSGLMUL	50	-	-	-	-	×*	×
141	FSUB	50	-	-	-	-	×	×
142	FSINCOS	50	-	-	-	-	×*	×
143	FCMP	50	-	-	-	-	×	×
144	FTST	50	-	-	-	-	×	×
145	FSMOVE	50	-	-	-	-	-	×
146	FSSQRT	50	-	-	-	-	-	×
147	FDMOVE	50	-	-	-	-	-	×
148	FDSQRT	50	-	-	-	-	-	×
149	FSABS	50	-	-	-	-	-	×
150	FSNEG	50	-	-	-	-	-	×
151	FDABS	50	-	-	-	-	-	×
152	FDNEG	50	-	-	-	-	-	×
153	FSDIV	50	-	-	-	-	-	×
154	FSADD	50	-	-	-	-	-	×
155	FSMUL	50	-	-	-	-	-	×
156	FDDIV	50	-	-	-	-	-	×
157	FDADD	50	-	-	-	-	-	×
158	FDMUL	50	-	-	-	-	-	×
159	FSSUB	50	-	-	-	-	-	×
160	FDSUB	50	-	-	-	-	-	×
161	FMOVECR	53	-	-	-	-	×	×
162	FMOVE von FPn	55	-	-	-	-	×	×
163	FMOVE FPcr	51	-	-	-	-	×	×
164	FMOVEM FPcr	52	-	-	-	-	×	×
165	FMOVEM FPn	54	-	-	-	-	×	×
166	FScc	58	-	-	-	-	×	×

Fortsetzung auf der nächsten Seite

Fortsetzung von vorheriger Seite								
Nr.	Befehl	Format	68000	68010	68020	68030	68040	68881/2
167	FDBcc	59	-	-	-	-	×	×
168	FTRAPcc	56	-	-	-	-	×	×
169	FNOP	0	-	-	-	-	×	×
170	FBcc	57	-	-	-	-	×	×
171	FSAVE	1	-	-	-	-	×	×
172	FRESTORE	1	-	-	-	-	×	×

*Nur mit einer Softwareemulation

Index

— Symbole —

— A —

— B —

— C —

— D —

— E —

— F —

— G —

— H —

— I —

— K —

— L —

— M —

— N —

— O —

— P —

— Q —

— R —

— S —

— T —

— U —

— V —

— W —

— X —

— Z —