Der NDR-Klein-Computer

Handbuch

zum

680XX-Grundprogramm

© 1990 Ralph Dombrowski

Vers.
6.0, 6.1 und 6.2

Der NDR-Klein-Computer

Handbuch für die 68000-Serie

Gültig für die Prozessoren
68008, 68000 und 68020 (68881)

Zum Grundprogramm
Version
6.0, 6.1 und 6.2

Alle Rechte vorbehalten
Nachdruck - auch auszugsweise - nur mit Genehmigung

© 1990 Ralph Dombrowski

Glückstadt, März 1990

Änderungen vorbehalten

Herstellung und Bearbeitung
Firma Rautenberg, Glückstadt

Kapitel 0 - Inhaltsverzeichnis

Kapitel 1 - Einführung

1.1. Vorwort

Das bewährte Grundprogramm des NDR-Klein-Computers wird ständig überarbeitet und verbessert. Dabei werden selbstverständlich auch immer die neuesten NKC-Baugruppen berücksichtigt. Mit der jetzt vorliegenden Version 6.21 des Grundprogramms ist ein neuer Standard geschaffen worden, der für alle 680XX-CPUs gleich ist.

Die neue Version des Grundprogramms enthält viele Verbesserungen für alle Programmteile und eine Reihe neuer Unterprogramme. Die Arbeit mit dem Grundprogramm wird durch neue Editor- und Assemblerbefehle abermals erleichtert.

Die vorliegende Dokumentation baut auf dem Handbuch für die Version 4.3 des Grundprogramms auf, ist aber vollständig überarbeitet worden. Die Einteilung der Dokumentation in Kapitel und die übersichtliche Beschreibung der Unterprogramme erlauben es, Informationen über das Grundprogramm leichter und schneller zu finden.

Ich hoffe, daß das Grundprogramm mit seinen vielen neuen Möglichkeiten von seinen Anwendern mit Erfolg genutzt wird.

Anregungen für weitere Verbesserungen des Grundprogramms und Hinweise auf Fehler erwarte ich gern, zumal ich trotz gründlicher Arbeit Fehlerfreiheit natürlich nicht garantieren kann. Deshalb kann ich auch nicht für eventuell entstandene Schäden beim Betrieb des Grundprogramms haften.

An dieser Stelle möchte ich mich auch bei den engagierten NKC-User'n bedanken, die mir im Laufe der Zeit viele Tips zur Verbesserung des Grundprogramms gegeben haben. Mein besonderer Dank gilt dabei Klaus Janßen, Hans-Dieter Bulwien, Uwe Koch und Klaus Rumrich. Auch möchte ich Herrn Dietrich Wilke noch meinen besonderen Dank aussprechen, da er das Handbuch gründlich überarbeitet hat und mir auch beim Inhalt noch einige Tips geben konnte. Durch seine Mitarbeit ist das Handbuch auch leichter verständlich geworden, da ich die Erklärungen mehr aus der Sicht eines Programmierers und nicht eines Anwenders geschrieben hatte. Nicht zuletzt möchte ich der Firma Rautenberg in Glückstadt danken, die es mir ermöglichte, das Handbuch in der hier vorliegenden Form zu gestalten.

Glückstadt, März 1990 Ralph Dombrowski

1.2. Zum NDR-Klein-Computer

Der NKC wurde und wird in der Fernsehserie "Mikroelektronik - Microcomputer selbstgebaut und programmiert" aufgebaut, erklärt und in Betrieb genommen. Diese Serie wird vom Norddeutschen Rundfunk, vom Sender Freies Berlin, vom Bayerischen Fernsehen und von Radio Bremen ausgestrahlt. Auch die Regionalsender anderer Bundesländer werden die Serie bald in ihr Programm aufnehmen.

Der NKC wurde Anfang der achtziger Jahre von Rolf-Dieter Klein entwickelt. Zuerst lief er nur mit einer Z80-CPU und ein paar kleinen Baugruppen. Inzwischen ist er ein sehr leistungsfähiges System geworden, für das es verschiedene CPUs und viele Baugruppen gibt. Er ist nicht nur sehr gut geeignet, Computer-Technologie zu erlernen, sondern läßt sich auch zum Programmieren für nahezu alle Anwendungsgebiete einsetzen.

Auf Grund seiner durch seine Modulbauweise bedingten Flexibiltät kann der NKC auch an viele andere Computersysteme angeschlossen werden und diese unterstützen. Wegen seines offenen Systems kann er jederzeit auf den neuesten Stand gebracht werden und deshalb nie veralten.

In der letzen Zeit ist auch das Software-Angebot für den NKC sehr beachtlich gestiegen. Es gibt bereits eine große Auswahl von Betriebssystemen, die für jeden das Richtige bieten. Dazu gehören so bekannte Systeme wie OS/9 oder CP/M68K. Es gibt aber auch Betriebssysteme, die speziell für den NKC geschrieben sind und die auf die Programmfunktionen und Unterprogramme seines Grundprogramms zugreifen. Diese Betriebssyteme sind JADOS und SYSTEMA-DOS. Weil der NKC mit so vielen Betriebssystemen arbeiten kann, steht für ihn inzwischen viel Software zur Verfügung.

Da es für den NKC mehrere CPUs gibt, muß für jede CPU-Gruppe ein eigenes Betriebssystem bzw. Grundprogramm vorhanden sein. Für die 68000-Serie ist es das hier vorliegende 680XX-Grundprogramm.

1.3. Wozu dient das Grundprogramm ?

Das Betriebssystem eines Computers besteht aus einer Anzahl von Programmen, die es dem Benutzer ermöglichen, mit dem Computer zu arbeiten.

Je leistungsfähiger ein Betriebssystem ist, umso mehr Programme stellt es dem Benutzer zur Verfügung, die die Kommunikation zwischen ihm und dem Computer ermöglichen, aber auch beispielsweise die Erstellung von Texten oder Dateien, die Entwicklung von Programmen einschließlich der Möglichkeit, sie zu testen.

Für den NKC gibt es als Betriebssystem das Grundprogramm. Es ist sehr leistungsfähig und enthält die wichtigsten Funktionen zum Betrieb eines Computers. Dazu gehört u. a. ein Programm-Editor, der nicht nur zum Schreiben von Listings, sondern auch für Korrespondenz geeignet ist. Weiter gibt es im Grundprogramm einen Assembler, der sehr schnell die mit dem Editor geschriebenen Programme übersetzt. Außerdem sind beispielsweise Programme zum Austesten der mit dem Editor geschriebenen und dem Assembler übersetzten Programme im Grundprogramm enthalten, weiter solche zum Manipulieren einzelner Bits in Programmen, zum Brennen von Eproms, zur Steuerung von Druckern, Diskettenlaufwerken, Festplatten usw. Alle diese Funktionen des Grundprogramms sind über eine Menüsteuerung verfügbar, ohne daß der Benutzer Programmierkenntnisse haben muß. Die einzelnen Menüfunktionen werden in Kapitel 3 erläutert.

Ein besonderer Vorteil des Grundprogramms sind die in ihm integrierten "Unterprogramme", die komplette, immer wieder beim Erstellen von Programmen benötigte Routinen zur Verfügung stellen. Von diesen Unterprogrammen gibt es bis jetzt 150. Sie erleichtern, um nur einige Beispiele zu nennen, die Ausgabe von Texten auf dem Monitor oder dem Drucker, berechnen die verschiedensten Werte, ermöglichen graphische Darstellungen, steuern Drucker, Diskettenlaufwerke und Festplatten. Die Einzelheiten zu den Unterprogrammen sind, jeweils mit Beispielen für die Programmierung, in Kapitel 4 dargestellt.

Das Grundprogramm ermöglicht es, andere Betriebssysteme von Disketten oder einer Festplatte zu laden und mit dem NKC unter einem anderen Betriebssystem als dem Grundprogramm zu arbeiten. So läßt sich beispielsweise der NKC mit den Betriebssystemen CP/M68k, JADOS oder OS-9 fahren. Dabei sind die Betriebssysteme JADOS und CP/M68K in das Grundprogramm integriert, d. h. man kann von ihnen aus auf viele Grundprogrammfunktionen zugreifen.

Selbstverständlich können auch andere Programme als Betriebssysteme in den Arbeitsspeicher des NKC geladen und von dort zum Laufen gebracht werden, z. B. Buchhaltungsprogramme, Vokabeltrainer, Schachspiele o. ä.

1.4. Literatur und Informationen

Den NKC-Computer kann man selbst aufbauen. Deshalb sind Informationen über ihn besonders wichtig. Sie sind sehr vielfältig, weshalb hier nur einige Hinweise gegeben werden können.

Bücher:

Rolf-Dieter Klein:
"Die Prozessoren 68000 und 68008: Rechnerarchitektur und Sprache im NDR-KLEIN-Computer"
Franzis-Verlag GmbH, München, 1986
- Unentbehrlich für jeden, der den NKC bauen möchte. -

Werner Hilf, Anton Nausch:
"M68000 Familie, Teil 1: Grundlagen und Architektur"
te-wi Verlag GmbH, München, 1984
- Beschreibt ausführlich den Befehlssatz der 680XX-CPUs und ist sehr empfehlenswert als Nachschlagewerk für den Assembler-Programmierer. -

Motorola:
"MC68020 User's Manual" und "MC68881 User's Manual"
- Werksunterlagen, die alle Funktionen der genannten Prozessoren und ihre Programmierung beschreiben. -

Zeitschriften:

"LOOP"
Graf Electronik Systeme GmbH, Kempten
- Erscheint etwa alle zwei Monate und bringt Beiträge über Hard- und Software für den NKC. -

"SCC - Mitteilungen"
Mitteilungen des Selbstbau-Computer-Clubs e. V., Hamburg
- Erscheint 5 mal jährlich mit - oft kritischen - Beiträgen zur Hard- und Software für den NKC. -

Kapitel 2 - Installation des Grundprogramms

2.1. Hardware-Voraussetzungen

Für das Arbeiten mit dem Grundprogramm ist folgende Mindestkonfiguration des NKC erforderlich:

a) eine CPU68K, CPU68000 oder eine CPU68020,

b) eine ROA64 oder ROA256 mit den Grundprogramm-Eproms und dahinter mindestens 8 KB RAM,

c) außerdem natürlich eine Stromversorgung, ein BUS, ein Tastaturanschluß (KEY), eine GDP68K oder besser noch eine GDP64HS, ein Monitor

d) und schließlich eine oder mehrere BANKBOOT oder, wenn das Grundprogramm von einer Diskette geladen wird, ein Diskettenlaufwerk und eine FLO2 oder FLO3.

Beim Einsatz der CPU68000 müssen die Speicherbaugruppen (ROA) doppelt, beim Einsatz der CPU68020 vierfach vorhanden sein.

2.2. Software-Voraussetzungen

Für die Arbeit mit dem Grundprogramm ist keine weitere Software erforderlich. Allerdings sollte ein Diskettenbetriebssystem installiert werden, damit ein größeres Software-Angebot genutzt werden kann. Eine Aufstellung der verschiedenen Betriebssysteme erfolgt weiter unten.

2.3. Erweiterungsmöglichkeiten

Für den NKC gibt es eine Vielzahl von Baugruppen und Programmen, die zusammen mit dem Grundprogramm eingesetzt werden können. Die Wichtigsten werden nachfolgend kurz erläutert.

Hardware:

AD-Baugruppen - Analog-Digital-Wandler.

BANKBOOT - Die Karte ermöglicht die Installation des Grundprogramms auf höheren Adressen als Adresse $0. Beim Einsatz der BANKBOOT-Karte kann ab der Adresse $0 RAM stehen, was Voraussetzung für den Einsatz von CP/M68K ist. Beim Einsatz der CPU68020 ist diese Karte nicht erforderlich.

CAS oder **CAS2** - Baugruppe zum Anschluß eines Kassettenrecorders zur - eher umständlichen - Speicherung von Daten.

CENT2 - Baugruppe für den Anschluß eines Druckers. Für den fortgeschrittenen Anwender/Programmierer unverzichtbar.

COL256 - Farbgrafik-Karte, die vom Grundprogramm mit Hilfe eines in ihm integrierten GRAFIK-Pakets angesteuert wird.

DA-Baugruppen - Digital-Analog-Wandler, einsetzbar z. B. für Robotersteuerungen.

FLO2 oder **FLO3** - Nötig für den Betrieb von einem oder mehreren Diskettenlaufwerken.

GDP64K oder **GDP64HS** - Grafik-Karten zum Anschluß eines Monitors (Bildschirms). Die GDP64HS verfügt über zusätzliche Funktionen, z. B. Hardscroll, und ist deshalb besonders zu empfehlen.

HARDCOPY-MAUS - Karte zum Anschluß einer Maus und zum Auslesen des GDP-Bildes. Nicht nötig bei Einsatz einer GDP64HS und der KEY3.

IOE oder **IOE2** - Baugruppen zum Anschluß von Peripheriegeräten.

KEY, KEY2 oder KEY3 - Karten zum Anschluß einer parallelen Tastatur.

PROMER oder PROMER2 - Karten zum Programmieren von Eproms.

RELAIS - Baugruppe mit 8 Relais.

ROA64 oder ROA256 - Baugruppen für Speicherausbau.

R256DYN - Dynamische Speicherbaugruppe. Preiswert, verlangsamt aber die Arbeit der CPU, deshalb nicht unbedingt zu empfehlen.

SASI - Baugruppe zur Ansteuerung einer Festplatte, wird vom Grundprogramm nicht unterstützt.

SCSI - Modifizierte SASI-Baugruppe, mit der eine SCSI-Festplatte angesprochen werden kann und die vom Grundprogramm unterstützt wird.

SER - Serielle Baugruppe.

SOUND - Karte für Tonausgabe.

SPRACHE - Karte für einfache Sprachausgabe.

UHR - Uhrenbaugruppe, wird nicht mehr vertrieben. Alternativ: SMART-WATCH, Uhrenbaustein in Form eines 8 x 8 K-RAMs, der wie ein solches RAM auf eine ROA gesteckt wird, und auf die eins der genannten RAMs gesetzt werden kann.

Viele der Baugruppen sind auf eine feste Adresse eingestellt. Sie arbeiten nur, wenn sie auf diesen Adressen eingesetzt sind, weil sie nur dann vom Grundprogramm angesprochen werden können. Die Adressen, auf denen das Grundprogramm die verschiedenen Baugruppen anspricht, ergeben sich aus der folgenden Aufstellung:

$	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0	HEX					SOUND	GDP	GDP	PROMER	SER4	COLOR		FLO	AD/DA	AD16x8	SER
1	HEX					SOUND	GDPHS	GDP	PROMER	SER4			FLO	AD/DA	AD16x8	SER
2	HEX							GDP	PROMER	SER4	FARBT		FLO	AD/DA	AD16x8	SER
3	HEX							GDP	PROMER	SER4	FARBT		FLO	AD/DA	AD16x8	SER
4			SCSI					GDP	PROMER	SER4	CLUT		FLO	AD/DA	AD16x8	
5			SCSI					GDP	PROMER	SER4	CLUT		FLO	AD/DA	AD16x8	
6			SCSI					GDP	PROMER	SER4	CLUT		FLO	AD/DA	AD16x8	
7			SCSI					GDP	PROMER	SER4			FLO	AD/DA	AD16x8	
8					CENT		KEY	GDP	HARDM	SER4	ACRTC		BANKBOOT	SPRACH	AD16x8	DA2x8
9					CENT		KEY	GDP	HARDM	SER4	ACRTC		RELAIS	SPRACH	AD16x8	DA2x8
A								GDP	HARDM	SER4			CAS	SPRACH	AD16x8	
B								GDP	HARDM	SER4			CAS	SPRACH	AD16x8	
C								GDP	HARDM	SER4	COL256		SASI	SPRACH	AD16x8	AD1x10
D								GDP	HARDM	SER4	COL256		SASI	SPRACH	AD16x8	AD1x10
E								GDP	HARDM	SER4	COL256		SASI	SPRACH	AD16x8	UHR
F								GDP	HARDM	SER4	COL256		SASI	SPRACH	AD16x8	

Zu den Abkürzungen (soweit nicht vorstehend erläutert):

AD16x8 - 8 Bit Analog-Digital-Wandler mit 16 Kanälen.
AD1x10 - 10 Bit Analog-Digital-Wandler.
ARCTC - Farbgrafik-Baugruppen, nicht vom Grundprogramm unterstützt.
CLUT - Farbtabelle für COL256 und ACRTC.
COLOR - Port für Farberweiterung der GDP
DA2x8 - 8 Bit Digital-Analog-Wandler mit 2 Kanälen.
FARBT - Farbtabelle für Farberweiterung GDP.
HARDM - Hardcopy-Maus-Baugruppe, auch für Maus-Teil auf KEY3.
HEX - Nicht unterstützte hexadezimale Eingabe-Baugruppe.
PROMER - Adressen gelten für den alten PROMER wie auch den neuen PROMER2. Die PROMER-Baugruppe verwendete aber nur die drei unteren Adressen.
SER4 - Ausgabeeinheit für mehrere serielle Schnittstellen, vom Grundprogramm nicht unterstützt.

Software:

Der NKC sollte auch mit wenigstens einem, besser zwei Diskettenlaufwerken und einem Diskettenbetriebssystem ausgestattet sein.

Zu empfehlen ist für den NKC das Diskettenbetriebssystem JADOS, das preiswert und sehr leistungsfähig ist. Es wird auch direkt vom Editor des Grundprogramms unterstützt.(Laden und Speichern von Texten).

Das (teurere) Betriebssytem CP/M68K eröffnet den Zugang zu einem großen Software-Angebot. Wer allerdings wirklich professionell mit dem NKC arbeiten will, sollte sich OS/9 anschaffen, ein Multi-User-Multi-Tasking-System. Dafür ist allerdings eine Harddisk erforderlich.

2.4. Einsatz verschiedener CPUs

Das 68000-Grundprogramm ist für die Arbeit mit jeder der drei 680XX-Karten bestimmt. Das gilt auch für dieses Handbuch. Es enthält deshalb überall dort, wo die CPUs Unterschiede in den für sie einzusetzenden Grundprogramme bedingen, entsprechende Hinweise.

Das Grundprogramm für die 68000-CPU kann auch mit der 68010-CPU betrieben werden, wenn im Quelltext des Grundprogramms an den gekennzeichneten Stellen die dort beschriebenen Änderungen durchgeführt werden. Anschließend muß es neu assembliert werden.

Das 68000-Grundprogramm enthält alle Funktionen des 68008-Grundprogramms, das 68020-Grundprogramm alles, was Inhalt des 68000-Grundprogramms ist. Programme, die auf einem "schwächeren" Prozessor erstellt wurden, laufen bei Berücksichtigung der IO-Adressen der "stärkeren" Prozessoren auch bei deren Einsatz.

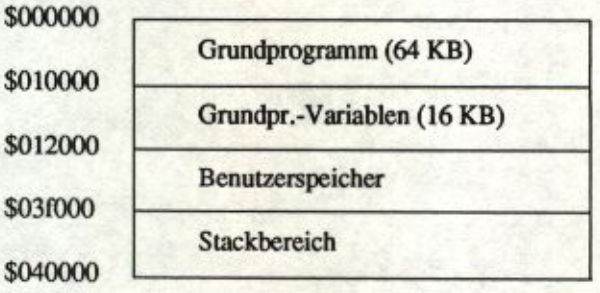
2.5. Speicheraufbau

Es gibt zwei Möglichkeiten für den Betrieb des Grundprogramms. In jedem Fall muß aber direkt hinter dem Grundprogramm mindestens ein Speicherbaustein vorhanden sein. Der weitere Speicheraufbau ist variabel, wobei es vorteilhaft ist, einen möglichst zusammenhängenden Speicherbereiche zu haben, je größer, desto besser.

Betrieb ohne BANKBOOT-Karte:

Wenn keine BANKBOOT-Karte eingesetzt werden soll, muß das Grundprogramm auf Adresse $0 beginnen.

Beispiel:



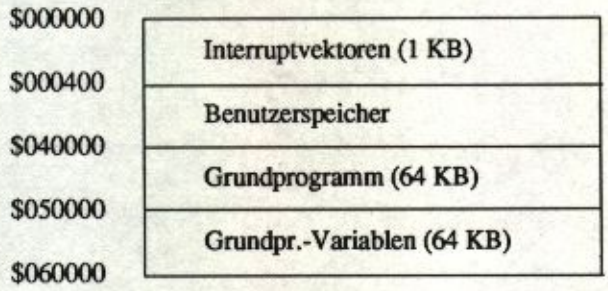
Bis maximal 67 Kbyte.

Je nach Ausbau verschieden.

Betrieb mit BANKBOOT-Karte:

Beim Betrieb des NKC mit der BANKBOOT-Karte oder beim Einsatz der CPU68020 muß das Grundprogramm nicht auf Adresse $0 liegen. Es kann auf jeder beliebigen Adresse eingesetzt werden. Dann muß aber ab der Adresse $0 RAM vorhanden sein, und zwar möglichst viel. Das Grundprogramm sollte in diesem Fall im RAM möglichst weit hinten auf einer beliebigen Adresse liegen. Beim Einsatz der CPU68k wird die Adresse $0E0000 empfohlen. Beim Einsatz der CPU68000 sollte das Grundprogramm auf Adresse $1E0000 liegen, beim Einsatz der CPU68020 auf Adresse $3E0000.

Beispiel:



Bis zu maximal 67 Kbyte.

2.6. Voreinstellungen für das Grundprogramm

Bevor das Grundprogramm gestartet werden kann, müssen auf der KEY-Karte (auch KEY2 und KEY3) verschiedene Voreinstellungen vorgenommen werden. Den DIL-Schaltern auf der KEY-Karte sind nämlich seit der Version 6.0 des Grundprogramms bestimmte Funktionen zugewiesen, die für die Arbeit mit dem Grundprogramm erfüllt sein müssen.

Schalternummer	Stellung Aus (Masse)	Stellung Ein (+ 5 Volt)
1	Altes Menü	Neues Menü
2	Kein BOOT-Menü	BOOT-Menü beim Start
3	Uhrenbaugruppe nein	Uhrenbaugruppe ja
4	Alte GDP64K	Neue GDP64HS
5	HARDDISK nein	SCSI-HARDDISK ja
6	Reserviert	Reserviert
7	Reserviert	Reserviert
8	Reserviert	Reserviert

Soll beim ersten Einschalten des NKC das alte Menü erscheinen, d.h. das mit mehreren Seiten, muß der DIL-Schalter 1 auf der KEY-Karte auf AUS gestellt werden. Andernfalls erscheint beim ersten Einschalten das neue Menü, das alle Funktionen auf einer Seite enthält. Diese Voreinstellung kann jederzeit während der Arbeit mit dem Grundprogramm durch das Grundprogramm selbst geändert werden.

Soll beim ersten Einschalten das BOOT-Menü erscheinen, muß der DIL-Schalter 2 auf der KEY-Karte eingeschaltet werden.

Bei Verwendung einer Uhrenbaugruppe ist der DIL-Schalter 3 einzuschalten. Erkennt das Grundprogramm das Vorhandensein einer SMART-WATCH, spricht es die SMART-WATCH automatisch an. Ist die SMART-WATCH vorhanden, so ist dieser DIL-Schalter eigentlich ohne Funktion, da diese der Uhrenbaugruppe vorgezogen wird.

Beim Einsatz der SMART-WATCH ist darauf zu achten, daß sie auf einer ROA-Karte im RAM-Bereich eingesteckt wird. Dabei darf die SMART-WATCH nicht auf einem Steckplatz direkt hinter den Grundprogramm-Eproms und nicht am Ende des RAM-Bereichs eingesetzt werden. Andernfalls werden die Eintragungen im RAM für die Systemvariablen des Grundprogramms und sein Stack gestört und es kann zum Absturz des Grundprogramms kommen.

Wenn die GDP64HS im NKC eingebaut ist, sollte der DIL-Schalter 4 auf der KEY-Karte eingeschaltet werden, damit deren neue Möglichkeiten wie Hardscroll oder Auslesen des Bildschirms genutzt werden können.

Ab Version 6.2 des Grundprogramms ist auch der Betrieb einer HARDDISK unter dem Grundprogramm möglich. Die Beschreibung der HARDDISK und der dafür verfügbaren Programme erfolgt in Kapitel 4.2. Um die Routinen der HARDDISK aufrufen bzw. die HARDDISK booten zu können, muß der DIL-Schalter 5 auf der KEY-Karte eingeschaltet sein.

2.7. Der erste Start

2.7.1. Grundprogramm auf Eproms

Wenn die DIL-Schalter auf der KEY-Karte eingestellt sind, müssen die Eproms mit dem Grundprogramm eingesetzt werden. Dazu werden bei Verwendung der CPU68K die Eproms in der richtigen Reihenfolge hintereinander auf einer ROA-Karte eingesetzt, die auf die Adresse adressiert werden muß, auf der das Grundpogramm laufen soll. Die Einstellung der Adressen auf der ROA ist dem Handbuch für die ROA zu entnehmen.
Beim Einsatz der CPU68000 und der CPU68020 ist darauf zu achten, daß die Grundprogramm-Eproms jeweils in der richtigen Reihenfolge auf der richtigen BUS-Seite (gerade und ungerade Adressen) eingesetzt werden. Wegen der Einteilung der Adressen wird auf das Handbuch für die genannten CPUs verwiesen. Hinter dem Grundprogramm muß in jedem Fall auf jeder BUS-Seite mindestens ein RAM-Baustein vorhanden sein.

Sind alle Einbaumaßnahmen durchgeführt, sollte noch einmal der richtige Sitz der Eproms auf der ROA-Karte bzw. den ROA-Karten geprüft werden, bevor der NKC eingeschaltet und das Grundprogramm gestartet wird.

Nach dem Einschalten des NKC muß sich nach einer kurzen Anlaufzeit das Grundprogramm mit seinem Copyright-Hinweis melden und danach mit dem Menü, je nach der Voreinstellung des DIL-Schalters auf der KEY-Karte mit dem einseitigen oder der ersten Seite des mehrseitigen Menüs (bzw. dem BOOT-Menü). Ist dies nicht der Fall, sollte die RESET-Taste gedrückt werden. Passiert auch dann nichts, muß der Computer ausgeschaltet werden und die Fehlersuche kann beginnen.

Meldet sich das Grundprogramm mit dem voreingestellten Menü, kann die Arbeit mit dem Grundprogramm beginnen. Nähere Einzelheiten dazu sind in Kapitel 3 beschrieben.

2.7.2. Grundprogramm auf Diskette

Das Grundprogramm der Version 6.21 wird auf JADOS-Disketten geliefert.

Auf der ersten Diskette befindet sich ein File, das das übersetzte Grundprogramm enthält (GRUNDXX.COM). Nur dieses File ist für das Folgende interessant. Alle anderen Files auf den Disketten sind in dem File LIESMICH.TXT beschrieben.

Zunächst muß sichergestellt sein, daß das alte Grundprogramm, das auf Eproms im System vorhanden sein muß, nicht auf Adresse $0 beginnt, da das neue Grundprogramm sonst nicht gestartet werden kann, wenn es von der Diskette ins RAM geladen worden ist. Deshalb ist bei Einsatz der CPU68K das Vorhandensein einer BANKBOOT-Karte erforderlich, beim Einsatz der CPU68000 das Vorhandensein von zwei BANKBOOT-Karten. Wird der NKC mit der CPU68020 betrieben, genügt es, die Eproms mit dem alten Grundprogramm in den hinteren Teil des RAM-Bereiches zu stecken.

Der erste Speicherbereich muß groß genug sein, um das neue Grundprogramm mit seinen 64 KB und die Systemvariablen, die für den ersten Test ungefähr 5 KB benötigen, aufzunehmen.

Nunmehr muß das neue Grundprogramm GRUND08.COM, GRUND00.COM bzw. GRUND20.COM von der Diskette auf eine Adresse im RAM geladen werden, die mindestens 70 Kbyte vor dem Ende des ersten durchgehenden Speicherbereiches liegt. Später, nach den ersten Tests sollte das neue Grundprogramm so plaziert werden, daß hinter ihm noch ca. 50 Kbyte liegen, damit die Symboltabelle für den Assembler genügend freies RAM zur Verfügung hat.

Ist das Grundprogramm geladen, genügt ein Druck auf die RESET-Taste, und das neue Grundprogramm sollte sich melden.

2.8. Fehlermöglichkeiten

Funktioniert der Computer nicht richtig, muß zunächst folgendes überprüft werden:

1) Stecken die Eproms richtig in der Fassung ?
2) Ist Ram hinter den Eproms vorhanden ?
3) Funktioniert der Computer ansonsten einwandfrei (ältere Version) ?
4) Sind die DIL-Schalter richtig eingestellt ?
5) Sind alle Baugruppen korrekt eingebaut ?

Dies sind die wichtigsten Dinge, die zuerst überprüft werden sollten. Natürlich können auch andere Probleme mit verschiedenen Baugruppen auftreten. Die Behebung dieser Schwierigkeiten ist in den Handbüchern der entsprechenden Baugruppen beschrieben.

Kapitel 3 - Grundprogramm-Menüs

In diesem Kapitel werden die einzelnen Menüpunkte und Funktionen des Grundprogramms beschrieben, die ohne Aufruf eines Unterprogramms oder eines zweiten Betriebssystems verfügbar sind.

Die Eingaben erfolgen außer im Editor meistens in einem Eingabefeld, das durch das Unterprogramm READ erzeugt wird. Alle Funktionen, die während dieser Eingaben möglich sind, werden bei der Erläuterung des Unterprogramms READ beschrieben (siehe Kapitel 4.2 unter READ). In das Eingabefeld dürfen alle Ausdrücke eingegeben werden, die beim WERT-Befehl erlaubt sind, außerdem auch Symbol-Zuweisungen wie bei ZUWEIS (siehe Kapitel 4.2 unter WERT und ZUWEIS).

Jeder Menüpunkt des Grundprogramms kann durch Eingabe des Buchstabens oder der Zahl, die dem Menüpunkt zugeordnet ist, auf der Tastatur aufgerufen werden. Dabei ist es gleichgültig, ob die Buchstaben in Groß- oder Kleinschrift eingegeben werden.

Will man von dem aufgerufenen Untermenü zurück in das Menü, von dem aus der Aufruf erfolgte, gibt man ein M auf der Tastatur ein oder auch ein ESC (= ESCAPE). Allerdings sind bei einigen Untermenüs andere Tastatureingaben nötig, um in das Ausgangsmenü zu gelangen. Das ist dann jeweils bei den Untermenüs angegeben. ESC ist aber immer möglich.

Arbeitet man mit dem alten, mehrseitigen Menü, kann man mit der Eingabe eines W (W = Weiter) die nächste Menüseite aufrufen, durch Eingabe eines Z im Menü zurückblättern.

Während des direkten Grundprogramm-Betriebs, also nicht, wenn mit Hilfe des Grundprogramms dritte Programme laufen, kann jederzeit vom Monitorbild mit der Tastenkombination CTRL-@ eine Hardcopy erzeugt werden. Welche Möglichkeiten man dabei hat, ist aus der Beschreibung des Unterprogramms CI ersichtlich (siehe Kapitel 4.2).

3.1. Menüfunktionen im einzelnen

Die Beschreibung der einzelnen Menüfunktionen erfolgt nachstehend an Hand des alten, mehrseitigen Menüs. Für das neue, einseitige Menü gelten die Erläuterungen entsprechend.

Beim Starten des Grundprogramms erscheint zunächst ein kurzer Copyright-Hinweis, danach bei Verwendung der alten Menüform die erste Seite. Das ist dann die Hauptseite, die vier wichtige Menü-Funktionen enthält. Hierher kehrt das Grundprogramm auch nach einem RESET oder nach Eintritt einer Ausnahmebedingung wie z. B. einem INTERRUPT oder einem ADRESSFEHLER zurück.

3.1.1. Ändern

Da die Speicherbereiche beim Grundprogramm nicht vom System verwaltet werden, sondern für jedes Programm (fast) frei verfügbar sind, ist der unbeschränkte Zugriff auf jeden Speicherplatz erlaubt.

Mit diesem Menüpunkt ist es ohne weiteres möglich, im RAM einzelne Bytes, Worte, Langworte oder ganze Speicherbereiche beliebig zu verändern. Dazu wird zuerst die Adresse eingegeben, ab der der Speicherinhalt geändert werden soll. Nach korrekter Adresseingabe werden 5 Bytes in hexadezimaler und dezimaler Darstellung sowie als ASCII-Zeichen (soweit möglich) ausgegeben. Außerdem werden die Werte der Adressen als Assembler-Befehl interpretiert, wenn eine gerade Adresse eingestellt ist. Handelt es sich insoweit um einen Befehl, wird er ausgegeben. Dabei wird der DIS-Assembler des EINZELSCHRITT benutzt.

Außerdem erscheint ein Eingabefeld, in das Werte und Befehle eingegeben werden können. Was nicht als Befehl eingegeben ist, wird als Wert interpretiert und berechnet. Tritt dabei kein Fehler auf, wird der Wert mit der angegebenen Länge (.B, .W oder .L am Ende) im Speicher an der aktuellen Adresse oder mit der Länge abgelegt, die beim WERT-Befehl ermittelt wurde. Dabei hängt die Länge vom Wert ab. Der Adresszähler wird nach der Ablage im Speicher auf den nächsten Speicherplatz hinter dem Wert gesetzt, und die neue Adresse wird mit den neuen Daten angezeigt. Es können übrigens auch Strings beliebiger Länge eingegeben werden, die mit ' eingegrenzt sein müssen.

Zusätzlich gibt es noch einige Befehle, die jeweils nur einen Buchstaben lang sind. Der Befehl CR ist eine Ausnahme. Er wird ausgeführt, wenn nur <RETURN> gedrückt wird und addiert eine 1 auf die aktuelle Adresse, die dann angezeigt wird.

Verfügbare Befehle:

M - Rückkehr zum Menü.

- - Mit der Eingabe eines Minuszeichen wird die Adresse um eins erniedrigt. Die neue Adresse wird angezeigt.

D - Es wird auf die Funktion ANSEHEN umgeschaltet. Von dort wird mit M in das Menü ANSEHEN zurückgeschaltet. Die Adresse kann sich durch Blättern in der Dump-Funktion ändern.

S - Mit dieser Funktion wird ein beliebiger Wert im Speicher gesucht. Dazu muß zuerst eingegeben werden, wie viele Bytes ab der aktuellen Adresse nach dem Wert durchsucht werden sollen. Dann wird der Wert selbst eingegeben, wobei auch Strings beliebiger Länge erlaubt sind, die durch ' eingegrenzt sein müssen. Wird der Ausdruck gefunden, wird die neue Adresse eingestellt. Die Werte dort werden angezeigt, sonst ändert sich nichts.

Beispiele:

S CR 100 CR $AA CR
Es werden die nächsten 100 Bytes ab der aktuellen Adresse nach $AA durchsucht.

S CR 1024 CR 'Hallo' CR
1 KB wird nach dem String "Hallo" durchsucht.

F - Hiermit wird ein Speicherbereich mit beliebigen Werten gefüllt. Dazu wird zuerst die Anzahl der zu füllenden Bytes eingegeben und anschließend der Wert (oder String). Es werden immer genau so viele Bytes überschrieben, wie angegeben wurde. Der Rest (z.B. eines Langwortes) wird beim Überschreiten der Anzahl abgeschnitten. Nach dem Füllen ist die aktuelle Adresse direkt hinter den gefüllten Bereich gesetzt.

Beispiel:

F CR 9 CR $12345678 CR
Der Speicher wird ab der aktuellen Adresse so gefüllt:
$12 $34 $56 $78 $12 $34 $56 $78 $12.

R - Nach Eingabe eines R kann eine neue Adresse aufgerufen werden.

3.1.2. Starten

Es erscheint ein Eingabefenster, in dem die Adresse eines Programms eingegeben werden soll. Dabei können auch Symbole (Namen) verwendet werden. Nur wenn der eingegebene Ausdruck ohne Fehler ausgewertet wurde, wird das Programm auf der angegebenen Adresse gestartet. Andernfalls wird in das Menü zurückgekehrt. Bei der Adressenangabe muß auf Korrektheit geachtet werden, da die Angabe einer Adresse, auf der sich kein Programm befindet, zu einer Exception (Systemabsturz) führen kann. Deshalb sollten nur Symbole verwendet werden. Wird nämlich ein falsches Symbol eingegeben, wird das erkannt. Es erfolgt ein Rücksprung in das Menü.

3.1.3. Ansehen

Um schnell einen Überblick über die Werte in bestimmten Speicherbereichen zu bekommen, benutzt man diesen Menüpunkt. Es wird zuerst eine Adresse eingegeben, ab der ein Ausschnitt aus dem Speicher in hexadezimaler und ASCII-Schreibweise ausgegeben werden soll. Außerdem wird eine Prüfsumme angegeben, die durch Langwort-Addition der Byte-Werte erzeugt wird. Die Ausgabe ist dabei immer auf eine 16-Byte Grenze gesetzt, wodurch die Darstellung übersichtlicher ist. Die aktuelle Adresse ist durch einen Pfeil gekennzeichnet.

Folgende Befehle gibt es, die durch Tastendruck angewählt werden:

M - Der Menüpunkt wird beendet. Normalerweise erscheint die erste Seite des alten bzw. das einseitige Menü. Wenn aber vom ÄNDERN aus das Programm ANSEHEN aufgerufen wurde, wird nach Eingabe eines M in das Programm ÄNDERN zurückgekehrt.

- - Es wird eine halbe Seite zurückgeblättert.

+ - Es wird eine halbe Seite vorgeblättert.

R - Eine neue Adresse kann ausgewählt werden.

S - Werte suchen (wie bei ÄNDERN). Die aktuelle Adresse ist durch den Pfeil markiert. Von dieser Adresse aus wird gesucht.

F - Speicherbereich füllen (wie bei ÄNDERN). Es wird ab der aktuellen Adresse (Pfeil) gefüllt.

1 - Es kann ein Byte geändert werden. Dabei muß die Adresse eingegeben werden, auf der das zu ändernde Byte steht (= bedeutet aktuelle Adresse). Dann erscheint ein Eingabefenster an der Stelle des zu ändernden Bytes. Das Eingabefeld ist leer. Es kann der Wert in hexazimaler Darstellung (ohne $) eingegeben werden oder als ASCII-Zeichen in '.

2 - Wie bei 1, allerdings ist im Eingabefenster der alte Wert vorhanden.

3 - Im Prinzip wie 1, allerdings werden 16 Bytes der Reihe nach abgearbeitet. Die angegebene Adresse wird vorher auf eine 16-Byte Grenze gebracht.

4 - Wie 3, allerdings ist immer im Eingabefenster der alte Wert vorhanden.

3.1.4. Symbole

Auf dem Bildschirm wird die gesamte Symboltabelle ausgegeben. Dabei können die Symbole durch den Assembler erzeugt worden sein oder durch eine Zuweisung mit dem Unterprogramm ZUWEIS. Eine Zuweisung eines Wertes an ein Symbol kann im Grundprogrammenü bei jeder Eingabe in einem Eingabefeld durchgeführt werden. Alle Namen werden alphabetisch ausgegeben, weshalb vorher die Symboltabelle durchlaufen werden muß, was etwas Speicher im Stack-Bereich erfordert, der aber nur bei sehr langen Tabellen knapp werden könnte.

Zuerst wird der Name, dann der Wert des Symbols und zum Schluß eine Kennung ausgegeben. Der Wert des Symbols wird als hexadezimale Zahl angezeigt. Die Kennung wird auch noch als Text ausgegeben, wodurch angezeigt wird, ob der Wert dem Symbol richtig zugewiesen wurde oder ob z.B. ein Syntax-Fehler bei der Definition aufgetreten ist.

Wenn die CO2-Routine, mit der die Ausgabe erfolgt, umgelenkt ist, kann die Symboltabelle auch auf einem Drucker oder über eine serielle Schnittstelle ausgegeben werden. Die Umlenkung ist bei der Routine CO2 beschrieben und erfolgt durch z.B. CO2SER, LST oder USR. Es wird dabei keine Seitenformatierung durchgeführt.

Ist die neue GDP64HS im System vorhanden, wird automatisch auf HARDSCROLL umgeschaltet, wodurch die Ausgabe sehr viel schneller wird. Mit CTRL-Q kann sie angehalten und mit CTRL-S fortgesetzt werden. Die Ausgabegeschwindigkeit kann bei HARDSCROLL-Betrieb mit SPACE (= Leertaste) umgeschaltet werden.

3.2. Seite 1

Auf dieser Seite sind wieder vier Funktionen verfügbar, die allerdings zum Teil noch über Untermenüs verfügen. Diese Seite ist für die Programmentwicklung wichtig, da hier der Editor und der Assembler zu finden sind.

3.2.1. Editor

Der Editor des Grundprogramms wird aufgerufen. Er ist hauptsächlich für die Erstellung von Programmen gedacht, kann aber auch zum Schreiben von Briefen verwendet werden. Der Editor ist immer auf eine aktuelle Adresse eingestellt, die beim ersten Start des Grundprogramms initialisiert wird, aber jederzeit geändert werden kann.

Die zu bearbeitenden oder zu erstellenden Texte können beliebig lang sein. Da der Text im Speicher linear abgelegt und aus Platzgründen nicht als Liste verwaltet wird, sinkt die Verarbeitungsgeschwindigkeit bei längeren Texten.

Wenn ein Text mit dem Editor bearbeitet werden soll, muß er folgende Bedingungen erfüllen: Er muß mit einer binären Null enden. Außerdem muß jede Zeile mit $0D $0A abgeschlossen sein.

Das Kontrollzeichen $09 wird als TAB interpretiert und intern in entsprechende Leerzeichen umgewandelt. Alle anderen Kontrollzeichen werden ignoriert.

Der Editor verfügt über viele Funktionen, die die Arbeit mit ihm sehr erleichtern. Dazu gehören Befehle wie SCROLLEN oder ZEILE EINFÜGEN, die eigentlich für jeden Editor selbstverständlich sind. Zusätzlich gibt es aber auch Befehle, die direkt mit dem Betriebssystem JADOS zusammenarbeiten und natürlich nur benutzt werden können, wenn man mit diesem Betriebssystem arbeitet.

Außerdem gibt es im Editor eine Statuszeile, die verschiedene Informationen kurz und übersichtlich wiedergibt. Dazu gibt es folgende Anzeigen in dieser Reihenfolge:

Start	Hier wird die Textstartadresse angegeben.
Fenster	Das ist die Startadresse im Speicher des aktuellen Bildausschnitts.
Tor	Die Adresse des Textendes, wobei allerdings noch die Länge des Bildausschnitts zugerechnet werden muß. Die wirkliche Adresse des Textendes wird angezeigt, wenn ganz hinter den Text gescrollt wird.
'einf'	Wird eingeblendet, wenn der Einfügemode angeschaltet ist.
'deut'	Der deutsche Zeichensatz ist aktiv.
'amer'	Der amerikanische Zeichensatz ist aktiv.
'Maus'	Die Maus ist eingeschaltet, mit der der Cursor bewegt werden kann.
'T'	Die Autotab-Funktion ist aktiv.
'S'	Es wird seitenweise gescrollt.
'Z'	Es wird zeilenweise gescrollt.

Nun folgt eine Aufstellung aller verfügbarer Befehle.

CTRL-A

Der Cursor wird auf den Anfang des nächsten Wortes links gesetzt. Dabei gilt als Trennung zwischen zwei Worten nur das Leerzeichen. Es wird auch über mehrere Zeilen positioniert.

CTRL-B

Der Cursor wird auf den Anfang der aktuellen Zeile gesetzt. Das geschieht unterschiedlich, je nachdem, ob die Funktion Autotab eingeschaltet ist oder nicht.

1) Autotab ist nicht angeschaltet:

   Der Cursor wird direkt auf den linken Rand gesetzt.

2) Autotab ist angeschaltet:

   Stehen in der Zeile Zeichen, wird der Cursor auf das erste Zeichen der Zeile gesetzt.
   Ist die aktuelle Zeile leer, sind aber Zeilen darüber belegt, wird der Cursor auf die gleiche senkrechte Position gesetzt wie das erste Zeichen der Zeile, die als nächstes weiter oben gefunden wird. Dabei wird nur der Bildschirmbereich abgesucht.
   Ist im Bildschirm oberhalb der aktuellen Zeile kein Zeichen vorhanden, wird der Cursor auf den linken Rand gesetzt.

CTRL-C

Es wird im Editor um 15 Zeilen nach vorne geblättert, d.h. es wird ein Bildschirmausschnitt weiter hinten im Text sichtbar.

CTRL-D

Der Cursor wird um ein Zeichen nach rechts verschoben. Nach dem Erreichen des Zeilenendes wird auf den Anfang der nächsten Zeile gesprungen, notfalls wird der Bildschirm gescrollt.

CTRL-E

Der Cursor wird eine Zeile nach oben bewegt. Befindet er sich bereits in der obersten Zeile, wird der Bildschirm verschoben.

CTRL-F

Es wird an den Anfang des Wortes rechts vom Cursor gesprungen. Steht in der Zeile rechts vom Cursor kein Wort mehr, springt der Cursor in die nächste Zeile. Dabei wird als Trennung zwischen zwei Worten nur ein Leerzeichen anerkannt.

CTRL-G

Ein Zeichen auf der aktuellen Cursor-Position wird gelöscht, der rechts vom gelöschten Zeichen stehende Teil der Zeile wird nach links verschoben.

CTRL-H

Wie CTRL-S.

CTRL-I

Der Cursor wird auf die nächste TAB-Position gesetzt, die im Abstand von jeweils 8 Zeichen steht, immer vom Zeilenanfang an gerechnet.

CTRL-J

Die Hilfe-Seite, d. h. die Seite mit der Erläuterung der Editor-Funktionen, wird aufgerufen. Die Eingabe eines beliebigen Tastaturzeichens führt in den Editor zurück.

CTRL-K

Diese Zeichenkombination aktiviert eine spezielle Abfrage. Mit der nächsten Eingabe wird die gewünschte Funktion ausgewählt.

CTRL-KA

Der Editor wird verlassen und der Assembler aufgerufen. Wird beim Assemblieren ein Fehler entdeckt, erscheint nach einer beliebigen Tastatureingabe wieder der Editor-Text, und zwar mit dem Cursor auf der ersten fehlerhaften Zeile. Wurde korrekt assembliert, endet das Programm, ohne daß der Editor noch einmal aufgerufen wird. Sobald man irgendein Zeichen auf der Tastatur eingibt, gelangt man ins einseitige Menü oder auf die erste Seite des mehrseitigen Menüs. Außerdem wird dann eine Meldung übergeben, daß der Editor mit dem Befehl CTRL-KA verlassen wurde (siehe Kapitel 4.2 unter EDIT). Der Befehl sollte nicht beim Einsatz des JADOS-Betriebssystems eingesetzt werden, da Betriebssysteme eigene Adressen für ihren Assembler verwenden. In jedem Fall muß beim Befehl CTRL-KA darauf geachtet werden, auf welche Adressen der Assembler den erzeugten Maschinencode ablegt, damit er nicht den Editor-Text überschreibt.

CTRL-KB

Mit dieser Tastenkombination wird der Anfang eines Blocks markiert, der dann mit weiteren Befehlen bearbeitet werden soll. Es erscheint an der aktuellen Cursor-Position ein nach rechts zeigender Pfeil. Die betreffende Zeile wird nach rechts verschoben. Enthält die Zeile achtzig Zeichen, geht das letzte Zeichen verloren. Ist auch das Ende des Blocks definiert, können verschiedene Block-Operationen durchgeführt werden. Ob zuerst der Anfang oder das Ende eines Blocks markiert werden, ist gleichgültig.

CTRL-KC

Diese Operation ist nur möglich, wenn ein Block markiert ist. Der Block wird dann an die aktuelle Cursor-Position kopiert. Allerdings darf die Cursor-Position nicht innerhalb des Blocks liegen, dann wird das Kommando ignoriert.

CTRL-KH

Die Marken für einen Block, die mit CTRL-KB und CTRL-KK gesetzt wurden, werden mit diesem Befehl entfernt. Der Befehl wird auch automatisch beim Verlassen des Editors ausgeführt.

CTRL-KI

Hiermit kann das Inhaltsverzeichnis einer beliebigen JADOS-Diskette angezeigt werden. Dazu muß vorher die Angabe des Laufwerks (z. B. 1:) und der Dateien erfolgen, etwa: *.TXT. Die möglichen Eingaben hängen von der jeweiligen JADOS-Version ab und können beim Befehl DIR im JADOS-Handbuch nachgelesen werden.

CTRL-KK

Mit diesem Befehl wird das Ende eines Blocks markiert (siehe auch CTRL-KB).

CTRL-KL

Von einer JADOS-Diskette wird ein beliebiges File an der aktuellen Cursor-Position eingefügt. Dazu muß der Dateiname entsprechend der JADOS-Namenskonvention eingegeben werden. Tritt beim Lesen des angewählten Laufwerks ein Fehler auf oder ist die gesuchte Datei nicht auf der Diskette, erfolgt keine Einfügung. Das gesuchte File muß im übrigen den Bedingungen für einen Editor-Text entsprechen und eine Null-Ende-Kennung enthalten.

CTRL-KQ

Die Bearbeitung des Editor-Textes wird beendet. Der Bildschirmausschnitt wird im Speicher abgelegt und eventuell vorhandene Blockmarken werden gelöscht. Das Ausgangsprogramm übernimmt wieder die Kontrolle. Außerdem wird eine Kennung übergeben, die festhält, daß der Editor mit diesem Befehl verlassen wurde (siehe Kapitel 4.2 unter EDIT).

CTRL-KR

Da im Speicher mehrere Editor-Texte gleichzeitig abgelegt sein können, kann mit diesem Befehl ein zweiter Text bei der aktuellen Cursor-Position eingefügt werden. Dazu wird die Adresse des zweiten Textes angegeben. Diese Adresse darf natürlich nicht im aktuellen Editor-Bereich liegen, sonst wird der aktuelle Text ab der Stelle, an der der Cursor steht, überschrieben. Außerdem sollte der einzufügende Text nicht direkt hinter vom Editor belegten Speicher-Bereich stehen, da sonst durch das Einfügen des zweiten Textes in den aktuellen Text möglicherweise der zweite Text zerstört wird.

CTRL-KS

Dieser Befehl kann nur ausgeführt werden, wenn ein Block mit CTRL-KB und CTRL-KK markiert wurde. Der Block wird dann als eigenes File unter dem eingegebenen Namen (gemäß JADOS Namenskonvention), der vorher eingegeben wurde, auf einer JADOS-Diskette abgespeichert. Es muß bei der Namensvergabe beachtet werden, daß nicht bereits ein anderes File mit dem selben Namen existiert, das nicht gelöscht werden soll.

CTRL-KT

Diese Funktion ist vom Ablauf her identisch mit dem Befehl CTRL-KS. Allerdings wird kein Block, sondern der gesamte Text abgespeichert.

CTRL-KW

Ein markierter Block wird im Speicher abgelegt. Dabei muß vorher die Zieladresse eingegeben werden, ab der der Block abgelegt werden soll. Die Adresse darf nicht im Bereich des Editor-Textes liegen. Das Programm fügt bei der Befehlsausführung am Schluß automatisch eine Ende-Null hinzu, weshalb der ab der angegebenen Adresse eingeschriebene Text als neues, selbstständiges File behandelt wird. Bei der Ablage des Textes sollte man auf jeden Fall einen freien Speicherplatz auswählen.

CTRL-KX

Auch hiermit wird wie mit CTRL-KQ der Editor verlassen. Allerdings wird eine andere Kennung zurückgegeben (siehe Kapitel 4.2 unter EDIT).

CTRL-KY

Der mit CTRL-KB und CTRL-KK markierte Block wird vollständig gelöscht.

CTRL-L

Der letzte Suchvorgang (bzw. Suchen und Ersetzen) wird wiederholt.

CTRL-M

Der Befehl entspricht der Funktion der Taste CR (auch <RETURN> genannt). Bei Ausführung des Befehls wird der Cursor auf die nächste Zeile gesetzt. Ist der Einfügemode aktiv, wird zuerst eine Zeile eingefügt. Danach wird der Befehl CTRL-B ausgeführt.

CTRL-N

Eine Zeile wird eingefügt, wobei der restliche Bildschirminhalt nach unten wandert.

CTRL-P

Der Zeichensatz wird von deutsch auf amerikanisch umgeschaltet oder umgekehrt. Die aktuelle Einstellung wird in der Status-Zeile des Editors angezeigt. Bei deutschem Zeichensatz können die Sonderzeichen ä ö ü Ä Ö Ü ß und bei amerikanischen Zeichensatz | { } \ [ ] ~ eingegeben werden. Die Ausgabe erfolgt gemischt.

CTRL-Q

Auch diese Tastenkombination leitet wie CTRL-K eine Spezialfunktion ein. Die nächste Taste wählt die gewünschte Funktion aus.

CTRL-QA

Der Befehl ermöglicht es, Zeichenfolgen im gesamten Text zu verändern. Zuerst wird eingegeben, welche Zeichenfolge verändert werden soll. Dann muß die neue Zeichenfolge, die an die Stelle der alten treten soll, angegeben werden. Wird die Zeichenfolge im Text gefunden, wird der Cursor dahinter gesetzt. Durch Eingabe eines J (= Ja) auf der Tastatur wird die Änderung veranlaßt. Gibt man auf der Tastatur eine N (= Nein) ein, wird die nächste Textstelle mit der zuerst eingegebenen Zeichenfolge gesucht. Die Betätigung jeder anderen Taste als J oder N beendet die Befehlsausführung.

CTRL-QC

Der sichtbare Editor-Bereich wird nach oben verschoben. Dabei wird der Ausschnitt so eingestellt, daß vom bisher sichtbaren Text nur noch die letzten 15 Zeilen im oberen Teil des Editor-Fensters zu lesen sind. Dieser Befehl ist nützlich, wenn man an das Ende des Textes springen will. Mit diesem Befehl geht das schneller, als wenn man mit dem Befehl CTRL-C den ganzen Editor-Text durchblättern müßte.

CTRL-QF

Hiermit wird ein bestimmter String gesucht. Die Zeichenfolge wird eingegeben, worauf der Cursor bei erfolgreicher Suche direkt hinter den Ausdruck gesetzt wird. Mit CTRL-L kann weiter gesucht werden.

CTRL-QR

Der Cursor wird an den Anfang des gesamten Textes gesetzt. Man braucht also nicht mit dem Befehl CTRL-R durch den ganzen Editor-Text zurückzublättern, um an den Textanfang zu gelangen.

CTRL-R

Der Bildschirmausschnitt wandert um 15 Zeilen zum Textanfang hin.

CTRL-S

Der Cursor wird um ein Zeichen nach links bewegt. Dabei wird bei Erreichen eines Zeilenanfangs an das Ende der vorherigen Zeile gesprungen. Stand der Cursor bereits am Textanfang, wird der Bildschirm verschoben.

CTRL-T

Ab der Cursor-Position wird der Rest der Zeile gelöscht.

CTRL-U

An der aktuellen Cursor-Position wird ein Zeichen eingefügt, die restliche Zeile wird nach rechts geschoben. Das letzte Zeichen geht verloren, wenn die Zeile zu lang wird.

CTRL-V

Der Einfügemodus wird an- bzw. ausgeschaltet. Ist er aktiviert, wird automatisch jedes eingegebene Zeichen an der aktuellen Cursor-Position eingefügt. Außerdem wird beim Befehl CTRL-M eine neue Zeile eingefügt.

CTRL-W

Der gesamte Text wird um eine Zeile nach unten gescrollt, was bedeutet, daß der Bildschirmausschnitt um eine Zeile zum Textanfang wandert.

CTRL-X

Der Cursor wird eine Zeile nach unten gesetzt. Befindet er sich in der letzten Zeile, wird der Bildschirmausschnitt verschoben.

CTRL-Y

Die Zeile, auf der sich der Cursor befindet, wird gelöscht, die restlichen Zeilen wandern nach oben.

CTRL-Z

Der gesamte Text wird um eine Zeile nach oben bewegt, der Bildausschnitt wandert also um eine Zeile zum Textende hin.

ESC A

Da das Grundprogramm nicht den Speicher verwaltet, können beliebig viele Texte an den verschiedensten Stellen im Speicher stehen. Mit dem Befehl wird ein neuer Textstart eingestellt. An der betreffenden Stelle muß aber tatsächlich ein Text vorhanden sein (siehe Kapitel 4.2.4.2. - TEXTSTART). Es folgt dann ein Sprung in den betreffenden Text. Der Ausgangstext wird nicht verändert.

ESC L

Der linke Rand im Editor ist variabel im Bereich 0 bis 63. Normalerweise ist er auf den Wert 0 gesetzt, d.h. der Text beginnt immer ganz links. Soll ein Rand in einem bestimmten Bereich oder im ganzen Text vorhanden sein, kann er mit ESC L eingerichtet werden, und zwar ggf. an verschiedenen Stellen im Text mit wechselnden Randeinstellungen. Der Rand wird durch Einfügen von Leerzeichen realisiert.

ESC M

Der Cursor kann außer mit den Funktionen, die bereits beschrieben wurden, auch mit der Maus der HARDCOPY-MAUS-Baugruppe bzw. KEY3 gesteuert werden. Dazu muß die Maus aber erst aktiviert werden, da es bei nicht vorhandener Baugruppe sonst zu Fehlern kommen kann. Ist die Maus aktiviert, wird der Cursor durch Drücken der linken Maustaste und Bewegung der Maus positioniert. Werden beide Maustasten gedrückt, bewegt sich der Cursor schneller. Der Cursor kann nur im Bildschirm-Bereich mit der Maus gesetzt werden.

ESC N

Im Prinzip wie ESC A, jedoch wird, beginnend auf der einzugebenden Adresse, ein neuer Textstart festgelegt und der Editor dort eröffnet, unabhängig davon, ob auf der betreffenden Adresse bereits ein Text steht oder nicht.

ESC O

Der Inhalt des aktuell sichtbaren Bildschirmausschnitts wird auf dem Drucker ausgegeben. Dabei erfolgt keine irgendwie geartete Druckersteuerung.

ESC P

Wie ESC O, hier aber mit Druckersteuerung. Vor dem Ausdruck wird der Drucker mit den Werten initialisiert, die mit Hilfe der Druckersteuerung eingestellt wurden (siehe Kapitel 3.4.4). Das entspricht der Funktion O der Druckersteuerung. Allerdings wird die Seitenlänge nicht berücksichtigt, wenn die Seitenlänge kürzer als 24 Zeilen eingestellt ist. Nach dem Ausdruck wird ein Seitenvorschub durchgeführt. Auch der Zeichensatz wird umgeschaltet, wenn der NDR-Zeichensatz bei der Druckersteuerung eingestellt ist.

ESC S

Es gibt zwei verschiedene Scrollarten. Ist in der Statuszeile ein S zu sehen, werden immer 15 Zeilen gleichzeitig vor- oder zurückgeblättert. Bei einem Z werden die Zeilen nacheinander bewegt. Mit ESC S schaltet man die Scrollart um.

ESC T

Hiermit wird der Autotab ein- bzw. ausgeschaltet. Ein T in der Statuszeile zeigt an, daß die Funktion aktiviert ist. Der Autotab erleichtert das Schreiben von Programmen, z.B. daß in einer Schleife alle Befehle gleich weit eingerückt werden.

3.2.2. Assembler

Der Assembler ist zusammen mit dem Editor das wichtigste Instrument zum Erstellen von Programmen. Der Assembler übersetzt den Programmtext in eine Folge von Hex-Zahlen, die der Computer beim Programmablauf verarbeiten kann.

Hier soll nicht erklärt werden, wie man Programme schreibt, sondern es wird nur auf die Funktionen des Assemblers eingegangen. Für das Erlernen der Programmiersprache Assembler gibt es genug Literatur.

Der Assembler des Grundprogramms versteht alle Befehle in der Form, wie sie von Motorola, dem Hersteller der 680XX-CPUs, definiert sind. Der Assembler des 68000-Grundprogramms kann auch die Befehle der 68010-CPU übersetzen. Beim 68020-Grundprogramm sind alle Befehle und Adressierungsarten der 68020-CPU und der FPU (68881) verfügbar.

Weiterhin gibt es eine Reihe von Pseudobefehlen, die den Umgang mit dem Assembler vereinfachen. Nur sie sollen hier erklärt werden.

Nachstehend zunächst eine kurze Übersicht über alle Pseudobefehle:

CO2SER	Ausgabe auf serielle Karte lenken.
CRT	Ausgabe auf Bildschirm lenken.
DC.<SIZE> Wert,Wert,...	Konstanten Ausdruck definieren.
DEBUGAN	Debugger für Einzelschritt anschalten.
DEBUGAUS	Debugger für Einzelschritt ausschalten.
DF.<SIZE> Anzahl,Wert	Bereich mit Werten füllen.
DS.<SIZE> Anzahl	Bereich freihalten.
END	Ende des Programmes markieren.
ENDMACRO	Ende eines Macros definieren.
EQU	Wert einem Symbol zuweisen.
LST	Ausgabe auf Drucker lenken.
MACRO Name	Beginn eines Macros definieren.
NIL	Nur Fehlerausgabe auf den Bildschirm.
NEWEQU	Wie EQU, aber ohne Berücksichtigung alter Werte.
NEWMACRO	Wie MACRO, aber mit Überschreiben alter Macros.
OFFSET Adresse	Ablageadresse für Maschinencode verschieben.
ORG Adresse	Übersetzungs- und Ablageadresse definieren.
RS.<SIZE> Anzahl	Bereich unabhängig vom PC reservieren.

RSRESET	RS-Zähler auf Null setzen.
RSSET Wert	RS-Zähler auf gewünschten Wert setzen.
SETFPX, SETFPY, SETFPZ	Variable X, Y oder Z für FP-Zahlen definieren.
SYMBOL:	Sprungmarke oder Variable definieren.
SYMCLR	Symboltabelle löschen.
USR	Ausgabe auf Benutzerschnittstelle lenken.
***** oder **;**	Kennzeichnung für Beginn eines Kommentars.

CO2SER

Normalerweise lenkt das Grundprogramm alle Ausgaben auf den Bildschirm, unabhängig davon, ob das ganze assemblierte Programm ausgegeben wird oder nur Fehlermeldungen. Ist ein Drucker an der seriellen Schnittstelle des NKC angeschlossen und soll das Programm beim Assemblieren über diese Schnittstelle ausgegeben werden, gibt man den Pseudobefehl CO2SER im Programm ein. In diesem Falle wird die Ausgabe des Listings von der Stelle an, an der CO2SER steht, auf den Drucker umgelenkt. Dabei werden nicht nur die Fehler übertragen, sondern alle Befehle (siehe auch CRT, NIL, LST, USR). Vor der ersten Benutzung des Befehls CO2SER muß die serielle Schnittstelle mit einem Hilfsprogramm initialisiert werden, da das Grundprogramm sonst nicht weiß, mit welcher Baud-Rate und sonstigen Einstellungen es die Daten übertragen soll.

CRT

Die Ausgabe wird auf den Bildschirm gelenkt. Es wird dann das gesamte Listing ausgegeben.

DC

Der Assembler kann nicht nur Befehle übersetzen und die übersetzten Werte dann im Speicher ablegen. Er ist auch in der Lage, Variablen bzw. Konstante zu definieren. Mit Hilfe der Anweisung DC wird in dem Speicherplatz, auf den sich der Befehl bezieht, ein zu definierender Wert abgelegt. Das kann Byte-, Wort- und Langwortweise geschehen. Der zu definierende Wert kann auch aus arithmetischen Ausdrücken gemäß den Konventionen des WERT-Befehls bestehen (siehe Kapitel 4.2 unter WERT).

DC.<SIZE> Wert,Wert,...	Allgemeine Definition.
DC.B Wert,Wert,'Textfolge'	Im Speicher werden die Werte an der aktuellen PC-Adresse als Bytes abgelegt. Es können in einer Zeile beliebig viele Werte durch Komma getrennt abgelegt werden. Es können auch Texte als Konstanten abgelegt werden. Sie müssen durch Hochkommas eingekleidet sein.
DC.W Wert,Wert,'AB'	Hier werden die Werte als Wörter (2 Bytes) abgelegt. Die Größenangabe kann entfallen (DC). Bei Texten werden nur die letzen beiden Buchstaben berücksichtigt.
DC.L Wert,Wert,'ABCD'	Die Werte werden als Langworte abgelegt. Texte werden nur mit den letzten vier Buchstaben abgelegt.

DC.W und DC.L müssen auf einer Wortgrenze beginnen, da die 680XX-CPUs nicht auf ungerade Adressen zugreifen. Das ist auch beim Einsatz der CPU68020 zu beachten, damit unter ihr geschriebene Programme für den Einsatz unter anderen 680XX-CPUs kompatibel sind. Der Prozessor 68020 erlaubt Zugriffe auch auf andere Datengrößen, die mit dem Coprozessor zusammenhängen. Unter der CPU68020 können Werte entsprechend der Konvention des Programmes FPUWERT benutzt werden. Dazu muß aber eine FPU im System vorhanden sein.

DC.S Wert,Wert	Eine Floating-Point-Zahl wird in Single-Density (einfacher Genauigkeit/Langwort) abgelegt.
DC.D Wert,Wert	Double-Density (2 Langworte).
DC.X Wert,Wert	Extended (3 Langworte).
DC.P Wert,Wert	Packed-Dezimal (3 Langworte/BCD Format).

DEBUGAN

Der Debugger ist durch das Grundprogramm voreingestellt. Das hat zur Folge, daß beim Assemblieren das Listing automatisch für den Trace-Betrieb (EINZELSCHRITT) dokumentiert werden kann. Bei umfangreicheren Programmen wird dadurch die betreffende Tabelle sehr lang. Deshalb kann es zweckmäßig sein, nur einige Adressen in die Tabelle aufzunehmen. Mit diesem Befehl wird der Anfang eines solchen Bereichs markiert. Folgt kein Befehl DEBUGAUS, wird jeder Befehl bis zum Programmende in die Tabelle eingetragen. In diesem Fall kann dann jeder Befehl beim Einzelschritt mit dem Quelltext zusammen angesehen werden. Beispiele folgen anschließend bei der Erläuterung des Befehls DEBUGAUS.

DEBUGAUS

Dieser Befehl ist das Gegenstück zur DEBUGAN. Er bewirkt, daß die Tabelle nicht weiter gefüllt und Speicher gespart wird. DEBUGAN und DEBUGAUS können beliebig eingesetzt werden. Wenn aber die Tabelle auch wirklich genutzt werden soll, muß DEBUGAN als letztes eingegeben sein.

Beispiel:

```
start:
        DEBUGAN                 * Debugtabelle wird belegt.
        CLR     D1              * Diese Befehle werden in
        CLR     D2              * die Tabelle aufgenommen und
        MOVEQ   #!MOVETO,D7     * werden beim Einzelschritt mit
        TRAP    #1              * dem Rest der Zeile angezeigt.
        DEBUGAUS                * Ab hier nicht weiter.
        MOVEQ   #100,D1         * Diese beiden Befehl werden vom
        MOVEQ   #100,D2         * DIS-Assembler übersetzt.
        DEBUGAN                 * Ab jetzt wieder angeschaltet.
        MOVEQ   #!DRAWTO,D7     * Jetzt ist die Tabelle aktiv,
        TRAP    #1              * da als letzter Befehl DEBUGAN
        RTS                     * steht.
```

DF

Mit diesem Befehl kann ein Speicherbereich gefüllt werden. Dazu wird zuerst die Anzahl der Adressen, die beschrieben werden sollen, und dann der Wert selbst eingegeben. Die Zahl der Adressen muß unbedingt angegeben werden oder eine Konstante sein, da sonst beim zweiten Assemblerdurchlauf eine Adressverschiebung zustande käme. Das Ergebnis wäre unsinnig. Der Befehl erlaubt im übrigen die Verwendung aller Datengrößen wie bei dem Befehl DC.

DF.<size> Anzahl,Wert — Gemäß der Größenangabe werden "Anzahl" Bytes, Worte oder Langworte mit "Wert" gefüllt.

Beispiele:

```
        DF.B    27,0            * Es werden 27 Nullen als Bytes abgelegt.
        DF.W    100,$FFFF       * Es werden 100 $FFFF abgelegt.
        DF.L    3,$12345678     * Es werden 3 $12345678 abgelegt.
```

Beim 68020-Grundprogramm sind auch Floating-Point-Zahlen erlaubt.

Beispiele:

```
        DF.S    3,1.1234        * 3 mal 1.1234 ablegen
        DF.D    100,5*7/3       * 100 mal 15 ablegen
        DF.X    5,exp(x)/sin(5) * 5 mal den Funktionswert ablegen
```

DS

Hiermit kann Speicher freigehalten werden. Es wird nur der PC erhöht, aber der Speicher nicht verändert. Ansonsten ist der Befehl ähnlich wie DC.

DS.<SIZE> Anzahl — Gemäß der Größenangabe werden "Anzahl" Bytes, Worte oder Langworte reserviert.

Beispiele:

```
DS.B    100     * 100 Bytes reservieren.
DS.W    50      * 50 Worte reservieren.
DS.L    25      * 25 Langworte reservieren.
```

Beim 68020-Grundprogramm kann als Größe auch S, D, X und P angegeben werden.

END

Soll der Assembler nicht am Ende des Textes mit der Bearbeitung aufhören, sondern an einer früheren Stelle, gibt man an der Stelle den Befehl END ein, von der an nicht weiter assembliert werden soll.

ENDMACRO

Ein Macro-Bereich wird beendet. MACRO und ENDMACRO dürfen nicht geschachtelt werden. Nach jedem MACRO muß also irgendwann ein ENDMACRO folgen. Geschieht das nicht, wird die Übersetzung nach dem ersten Durchlauf abgebrochen.

EQU

Mit dieser Anweisung kann jedem Symbol ein Wert zugewiesen werden, wie es mit dem Befehl ZUWEIS möglich ist. Dann wird jedesmal, wenn das Symbol verwendet wird, der dem Symbol zugeordnete Wert eingesetzt. Dabei darf das Symbol vor der Zuweisung des gewünschten Wertes nicht bereits mit einem anderen Wert belegt sein, da dies erkannt wird und eine Fehlermeldung ausgegeben wird. Der Assembler akzeptiert nur Integerwerte (ganzzahlige Werte), da die Symboltabelle nur solche verarbeiten kann.

Beispiele:

```
SYMBOL      EQU     5
SYMBOL1     EQU     SYMBOL*27/3
```

KOMMENTARE

Jede Zeile im Quelltext kann Kommentare enthalten. Dazu muß hinter dem Befehl, der erläutert werden soll, wenigstens ein Leerzeichen stehen, dem ein * oder ein ; folgen muß. Beim Assemblieren einer solchen Zeile wird der Kommentar ignoriert.

LST

Wie CO2SER, allerdings wird die Ausgabe auf den parallelen Port (CENT) geschaltet.

MACRO

Jetzt kommt ein sehr mächtiger Befehl. Mit der Anweisung MACRO wird ein Stück Quelltext gekennzeichnet. Diesen Teil kann man dann an beliebigen Stellen im Listing einsetzen und dadurch viel Tipparbeit sparen. Außerdem können Werte übergeben werden, sodaß Variationen möglich sind.

Eingeleitet wird die Definiton durch das Wort MACRO und den Namen, unter dem später auf den definierten Teil zugegriffen werden kann. Dann kommt die Befehlsfolge und anschließend als Endekennung das Wort ENDMACRO. Aufgerufen wird ein Macro durch einen Punkt mit anschließendem Namen (Beispiel: .Name). Macros dürfen sich auch untereinander aufrufen, wobei ein aufgerufener Macro vorher definiert sein muß. Dadurch werden gegenseitige Aufrufe vermieden, die zum Systemabsturz führen würden.

Da Macros auch über Platzhalter verfügen, können den mit Macros bearbeiteten Programmteilen wechselnde Werte übergeben werden. Die Werte werden dann anstelle der Zeichen eingesetzt. Das sind die Zeichen \0, \1, \2 bis \9, sodaß 10 Werte übergeben werden können. Weiterhin gibt es den Platzhalter \\, der nicht übergeben wird und die Nummer des Aufrufs des Macros repräsentiert. Dadurch können Schleifen in Macros aufgebaut werden.

Übergeben werden die Platzhalter beim Aufruf hinter dem Namen. Sie werden durch Komma getrennt und können von beliebiger Gestalt sein. Allerdings muß hinter dem letzten Platzhalter ein Komma stehen, falls noch Kommentare in die betreffende Zeile aufgenommen werden sollen. Wird als erstes Zeichen eines Platzhalters das Zeichen \ übergeben, werden die gesamten Zeilen des Macros, die den entsprechenden Platzhalter enthalten, ignoriert. Dadurch können Registerbelegungen beibehalten werden, falls ein Register nicht verändert werden soll.

Beispiele:

Die Routine WRITE des Grundprogramms wird als Macro definiert, so daß eine WRITE-Routine sehr viel kürzer wird als bei herkömmlicher Programmierung.

```
MACRO WRITE                                 * Anfang und Namen festlegen.
        MOVE.B      #\0,D0                  * Platzhalter 0 gibt die Schriftgröße an.
        MOVE.W      #\1,D1                  * X-Position des Textes = Platzhalter 1.
        MOVE.W      #\2,D2                  * Y-Position des Textes = Platzhalter 2.
        LEA         \3,A0                   * Adresse des Textes = Platzhalter 3.
        MOVEQ       #!WRITE,D7              * Text ausgeben.
        TRAP        #1
ENDMACRO                                    * Ende festlegen.

START:                                      * Es werden drei Texte ausgegeben
        .WRITE      $11,100,100,TEXT0(pc),  * Text 0 ausgeben.
        .WRITE      $21,0,20,TEXT1(pc),     * Text 1 ausgeben.
        .WRITE      \,0,0,TEXT1(pc),        * Text 1 ausgeben, aber d0 nicht laden,
        RTS                                 * d0 bleibt unverändert ($21).

TEXT0:
        dc.b 'Hallo',0
TEXT1:
        dc.b 'Testtext',0
```

Es soll eine Schleife realisiert werden:

```
MACRO WARTE                                 * Name des Makros.
        MOVE.W      #\0,D0                  * Anzahl der Schleifendurchgänge.
LOOP\?:                                     * Sprungmarke festlegen. Für \ wird bei jedem Aufruf eine neue
                                            * Zahl eingesetzt, sodaß es zu keiner Mehrfachdefinition kommt.
        NOP
        DBRA        D0,LOOP\?               * Schleifenende.
ENDMACRO                                    * Ende des Macros.
```

NIL

Die Ausgabe erfolgt auf dem Bildschirm, wobei aber nur etwaige Fehler angezeigt werden.

NEWEQU

Wie EQU, allerdings wird der vorherige Wert des Symbols ignoriert, wenn einer vorhanden war. Dadurch sind Umdefinitionen möglich. Es ist dann aber darauf zu achten, daß das Symbol vor der ersten Benutzung umdefinert wird, was sonst ja nicht nötig ist. Dadurch können lokale Definitionen vorgenommen werden.

Beispiel:

```
                                            * Beim ersten Durchlauf ist SYMBOL hier nicht definiert, beim
                                            * zweiten Durchlauf hat es den Wert 7
SYMBOL      NEWEQU      5

                                            * In diesem Bereich hat das Symbol den Wert 5
SYMBOL      NEWEQU      7                   * Mit EQU käme hier eine Fehlermeldung
                                            * Ab jetzt hat das Symbol den Wert 7
```

NEWMACRO

Wie MACRO, allerdings wird die vorherige MACRO-Definition überschrieben, wodurch wie bei NEWEQU Umdefinitionen möglich sind. Allerdings muß auch hier wie bei MACRO die Definition vor dem ersten Aufruf erfolgen. Durch diese Anweisung können sozusagen lokale Macros definiert werden (siehe MACRO).

OFFSET

Damit wird die Startadresse für die Ablage der Maschinencode-Adresse verschoben. Die Startadresse wird auf die aktuelle Adresse des PC addiert. Auf die sich aus der Addition ergebenden Adresse wird der Code abgelegt. Das ist nötig, wenn man Programme für Adressen schreiben will, für die im Computer kein RAM vorhanden ist und die nicht relokativ sind (z.B. im Eprom).

Beispiel:

```
OFFSET    $100      *Auf jede Ablageadresse wird der Wert $100 addiert.
                    * Die Übersetzung wird nicht beeinflußt.
```

ORG

Das Grundprogramm legt aufgrund einer entsprechenden Voreinstellung, wenn keine ORG-Anweisung erfolgt, den Maschinencode, d. h. das übersetzte Programm, ab einer bestimmten Adresse ab, die relativ kurz hinter dem Grundprogramm und der Symboltabelle liegt. Bei größeren Programmen ist das sehr unzweckmäßig, da die Symboltabelle, je länger sie wird, mit dem Maschinencode in Konflikt gerät. Um das zu vermeiden, gestattet der Assembler die Bestimmung anderer als der voreingestellten Ablageadresse. Das geschieht mit dem ORG-Befehl, der in einem Programm auch mehrfach verwendet werden kann, je nachdem, wohin die übersetzten Programmteile abgelegt werden sollen.

Verwendet man den ORG-Befehl zusammen mit dem OFFSET-Befehl, geschieht zweierlei: Das Programm wird so übersetzt, daß es auf der mit dem ORG-Befehl eingegebenen Adresse ablauffähig ist. Der Maschinencode wird aber verschoben, und zwar auf den Speicherbereich, der mit der angegebenen OFFSET-Adresse beginnt. ORG und OFFSET addieren sich zur Ablageadresse.

Beispiel:

```
ORG       $10000    * Das Programm wird für die Adresse $10000 übersetzt
                    * und auch dort abgelegt,
                    * falls nicht zusätzlich der Befehl OFFSET verwendet wird.
```

RS

Dieser Befehl ist eng verwandt mit der Anweisung DS. Auch mit ihm werden Freiräume für Variablen geschaffen. Hinter dem RS steht die Anzahl der Bytes, Wörter oder Langwörter, die freigehalten werden sollen. Der Befehl ist dafür gedacht, Variablen getrennt vom aktuellen PC-Stand zu definieren. Dazu ist ein eigener Zähler vorhanden. Der Zähler wird immer nur erhöht, wenn der Befehl RS verwendet wird. Er kann aber auch durch RSSET und RSRESET gezwungen werden, seinen Inhalt zu verändern.

Steht der RS-Befehl allein in einer Programmzeile, wird nur der Zähler entsprechend der Datengröße der RS-Anweisung verändert, denn auch hier sind wie bei DS alle Datengrößen erlaubt. Bei dem Befehl RS.B wird also der Zähler um einen Wert erhöht, bei RS.W um zwei Werte usw.

Beispiele:

```
RSRESET               * Zähler auf Null.
RS.B      100         * Zähler ist danach auf 100.
RS.W      50          * Zähler jetzt auf 200.
RS.L      25          * Zähler jetzt auf 300.
```

Definiert man vor einer RS-Anweisung ein Symbol, wird dem Symbol der Wert des Zählers zugewiesen, anschließend wird der Zähler erhöht, wie vorstehend beschrieben. Dadurch können Variablen überall im Programm fortlaufend definiert werden, ohne daß man auf den PC achten muß. Die relative Adressierung zu einem Adressregister ist dadurch sehr viel einfacher.

Beispiel:

```
RSRESET               * Zähler auf Null.
VAR1:     RS.B    1   * Var1 wird der Wert 0 zugewiesen.
VAR2:     RS.B    3   * Var2 wird der Wert 1 zugewiesen.
VAR3:     RS.W    2   * Var3 wird der Wert 4 zugewiesen.
          RS.L    2   * Der Zähler hat jetzt den Wert 8
                      * und jetzt 16.
```

RSRESET

Setzt den Zähler für die RS-Anweisung auf Null.

RSSET

Setzt den Zähler für die RS-Anweisung auf einen beliebigen Wert. Dadurch können viele verschiedene Speicherbereiche unabhängig voneinander definiert werden.

Beispiel:

```
RSSET     100     * Die nächste Variable, die mit RS definiert wird,
                  * bekommt den Wert 100.
```

SETFPX, SETFPY, SETFPZ

(Nur für 68020-Grundprogramm)

Mit diesen drei Befehlen können die internen Variablen X, Y und Z gesetzt werden, die bei jeder Auswertung von Floating-Point-Zahlen benutzbar sind. Die Anweisungen sind also identisch mit der NEWEQU-Anweisung, aber für FP-Zahlen. Siehe auch Kapitel 4.2 (SETFPX, SETFPY, SETFPZ).

Beispiel:

```
SETFPX    1.123456789e-56     * X hat jetzt den Wert 1.123456789 * 10^(-56).
```

SYMBOL:

In jedem Programm können Marken gesetzt werden. Beim Assembler funktioniert das durch Angabe des Symbolnamens mit anschließendem Doppelpunkt. Der Assembler akzeptiert ein Symbol aber nur, wenn es als erstes in einer Programmzeile steht. Leerzeichen dürfen allerdings vor dem Symbol stehen. Beim Assemblieren wird dem Symbol die Adresse des PC zugewiesen, auf die der PC zeigt, wenn der Assembler auf das Symbol trifft (Ausnahme: Es folgt RS). Nun kann das Symbol als Sprungmarke oder Variable eingesetzt werden. Die PC-Adresse wird nämlich nicht verändert, sodaß der nächste Befehl genau auf der Adresse des Symbols beginnt. Bei relokativen (verschiebbaren) Programmen sollte der RS-Befehl bei Variablen in Verbindung mit der Symbol-Anweisung verwendet werden.

SYMCLR

Die Symboltabelle wird wie im Menü gelöscht. Dieser Befehl sollte unbedingt vor der ersten Symboldefinition benutzt werden, da es sonst zu Fehlermeldungen kommt, wenn Symbole benutzt werden, die aber durch SYMCLR wieder gelöscht wurden.

USR

Einige der vorstehend beschriebenen Befehle können die Ausgabe des Listings auf vom Grundprogramm definierte Schnittstellen lenken. Will man aber die Ausgabe beispielsweise auf selbst gebaute Peripheriegeräte oder ins RAM lenken, steht dafür der Befehl USR zur Verfügung. Er lenkt die Routine CO2 um. Allerdings muß gemäß der Beschreibung von CO2 ein JMP-Befehl umgelenkt werden, denn sonst wird trotzdem auf den Bildschirm geschrieben (siehe Kapitel 4.2. unter CO2).

WEITERE BEFEHLE

Zusätzlich zu den im Programm einfügbaren Pseudo-Befehlen für den Assembler gibt es einige Befehle, die den Vorgang des Assemblierens betreffen und während des Assemblierens über die Tastatur eingegeben werden können.

ESC bricht den Assembliervorgang ab. Wird der Befehl im ersten Durchgang des Assemblers gegeben, ist noch kein Wert im RAM abgelegt.

CTRL-S stoppt den Assembliervorgang (wichtig bei Bildschirmausgaben), CTRL-Q setzt ihn fort.

Mit **SPACE** (Leertaste) kann die Ausgabegeschwindigkeit umgeschaltet werden, wenn die neue GDPHS vorhanden ist.

Für fortgeschrittene Programmierer:

Ab Version 6.0 liest das Grundprogramm den Text für den Assembler nicht mehr über die umlenkbare Schnittstelle CI2 ein, sondern holt ihn direkt aus dem RAM. Dadurch arbeitet der Assembler sehr viel schneller. Außerdem ist das für eine schnelle und Speicher sparende Implementation von MACROs nötig.

3.2.3. Bibliothek

Im Speicher des NKC können gleichzeitig mehrere Programme an verschiedenen Stellen abgelegt sein, auf die der Benutzer jederzeit zugreifen kann. Allerdings wäre es schwierig, wenn man sich alle Startadressen der Programme merken müßte. Man kann deshalb Programme in bestimmter Weise kennzeichnen und dadurch dem Grundprogramm ermöglichen, solche Programme leicht im Speicher aufzufinden und, wenn gewünscht, zu starten. Die Programme müssen dann an ihrem Anfang mit einer Kennung versehen sein, die den Namen des Programms enthält, seine Startadresse und einige andere Daten. Die Kennung muß auf einer 1-KB-Grenze liegen. Wie diese Kennung im einzelnen aussehen muß, ist in Kapitel 5.7. erläutert.

Damit man die erörterten Programme einfach aufgerufen kann, gibt es diesen Menüpunkt. Er listet die Programme auf. Dabei wird zu jedem der Programme sein Name, seine Startadresse, seine Länge und die Art seiner Programmierung (verschiebbar = relokativ oder mit festen Adressen) angegeben. Außerdem steht jeweils ein Buchstabe vor dem Programmnamen, mit dem das Programm über die Tastatur aufgerufen wird.

Sind so viele Einträge in dieser Bibliothek vorhanden, daß sie nicht auf eine Bildschirmseite passen, so kann man mit + und - zwischen den Eintragungen auf den Bildschirmseiten hin- und herblättern. Im unteren Teil des Bildschirms wird angegeben, welche Buchstaben bei Benutzung der Bibliotheksfunktion benutzt werden können. Mit M kann man die Bibliothek verlassen, wenn man kein Programm aufrufen möchte.

3.2.4. Optionen

Die OPTIONEN führen zu mehreren Untermenüs, die verschiedene Einstellungen des NKC in bezug auf Ausgabe, Editor-Adresse und Zeichenanzahl sowie die Einstellung des Trace-Betriebs (EINZELSCHRITT) zum Testen von Programmen erlauben.

3.2.4.1. Assembler-Optionen

Hier kann man verschiedene Ausgabearten für den Assembler einstellen. Die Einstellungen gelten dabei aber auch für alle Ausgaben mit der CO2-Routine (siehe Kapitel 4.2. unter CO2). Die Seiten haben bei Assemblerausgabe die Länge, die durch das Druckmenü eingestellt ist, auch wenn die Ausgabe auf dem Bildschirm erfolgt. Dadurch wird ein einheitliches Seitenformat garantiert.

Bei der Ausgabe des Listings gibt es folgende Möglichkeiten:

Nur Fehlerausgabe — Beim Assemblieren werden nur die Seitenüberschriften und die Fehler auf dem Bildschirm ausgegeben. Die restliche Ausgabe wird unterdrückt.

Ausgabe auf CRT — Das gesamte Programm wird mit den dazugehören Adressen, Befehlen und Kommentaren auf dem Bildschirm ausgegeben.

Ausgabe auf LST — Wie "Ausgabe auf CRT", allerdings erfolgt die Ausgabe auf dem Drucker.

Auf LST ohne LF — Die Ausgabe erfolgt auf dem Drucker, wobei aber das Zeichen $0A (Linefeed) unterdrückt wird, da einige Drucker es automatisch einfügen.

3.2.4.2. Textstart

Die Einstellung des Textstartes ermöglicht es, mehrere Texte gleichzeitig im Speicher zu halten und sie nach Belieben zu bearbeiten. Mit diesem Menüpunkt kann dazu zwischen den Texten hin- und hergeschaltet und auch neue Texte eingerichtet werden. Die Textadressen können aber auch während des Editor-Betriebs mit ESC A bzw. ESC N geändert werden.

Beim Textstart gibt es zwei Einstellungsmöglichkeiten. Zum einen kann eine Adresse gewählt werden, auf der sich bereits ein Text befindet. Dieser "Alte Text" wird beim Aufruf nicht verändert und kann anschließend im Editor bearbeitet werden. Beim zweiten Menüpunkt wird auf der einzugebenden Adresse ein Text begonnen, d.h. der Editor ist leer, wenn man ihn aufruft. Da Texte, die schon auf dieser Adresse beginnen, durch die Eingabe einer "Neuen Text"-Adresse zerstört werden, erfolgt vor der Einrichtung des neuen Textstarts eine Sicherheitsabfrage.

3.2.4.3. Löschen Symbole

Mit Hilfe dieser Funktion wird nach einer Sicherheitsabfrage die gesamte Symboltabelle gelöscht, weshalb danach keine Symbole mehr gültig sind. Das kann manchmal notwendig sein, wenn ein neues Programm übersetzt werden soll und kein Platz mehr in der Symboltabelle ist.

3.2.4.4. Zeichen/Zeile & Debug

Der Editor ist auf 80 Zeichen pro Zeile voreingestellt. Manchmal ist es aber erwünscht, daß nur 40 Zeichen pro Zeile dargestellt werden sollen, z. B. weil die Schrift besser lesbar sein soll. Wird auf 40 Zeichen/Zeile umgeschaltet, erfolgen alle Bildschirmausgaben der Routinen CO2, CO und CHAR (Symbole ausgeben, Editor, Assembler) mit der größeren Schrift.

Testet man Programme im Einzelschritt, ist es oft nützlich, bei jedem Schritt zu sehen, welcher Befehl des Listings gerade abgearbeitet wird. So kann man leicht Fehler im Programm feststellen. Die Anzeige der Befehle erfolgt, wenn der Debugger eingeschaltet ist. In diesem Falle legt nämlich der Debugger beim Übersetzen eine Tabelle mit den erforderlichen Daten ab. Ist der Debugger beim Übersetzen eingeschaltet, darf der Editor anschließend nicht aufgerufen werden, da sonst der Debugger wieder abgeschaltet wird. Wird der Debugger nicht mehr benötigt, sollte er abgeschaltet werden, da die Tabelle für den Debugger bei größeren Programmen viel Speicherplatz benötigt (siehe auch ASSEMBLER und EINZELSCHRITT).

3.3. Seite 2

Diese Seite enthält einige Funktionen, die nur selten benutzt werden. Es sind das die Kassettenfunktionen, die nur deshalb beibehalten worden sind, weil manchmal noch aus Kostengründen Kassettenrecorder zum Speichern von Programmen eingesetzt werden oder weil noch alte Programme auf Kassette vorhanden sind. Außerdem befindet sich hier die Funktion zum Booten eines Disketten- oder Festplatten-Betriebssystems.

3.3.1. Speichern CAS

Hier sind zwei Arten von Daten zu unterscheiden: Daten und Texte.

Bei der Speicherung von Texten ist die Startadresse des Textes und sein Name anzugeben. Das Ende des Textes wird automatisch gefunden, da das Textende immer durch eine Null gekennzeichnet ist. Als Adresse des Textes kann auch ein Fragezeichen eingegeben werden, wenn der aktuelle Text gespeichert werden soll.

Bei der Speicherung von Daten bzw. Programmen müssen der Name, die Startadresse und die Endadresse eingegeben werden, weil das Ende nicht erkannt werden kann.

Vor dem Start der Funktion muß der Kassettenrecorder auf Aufnahme gestellt werden. Wenn das Relais auf der CAS-Baugruppe benutzt wird, schaltet sich der Motor automatisch ein.

3.3.2. Laden CAS

Um abgespeicherte Daten zu laden, sind wieder zwei Programme vorhanden. Bei Texten muß die Startadresse angegeben werden, da alle Texte auf beliebige Adressen geladen werden können. Bei Daten sind keine Angaben erforderlich, da sie nur auf die alte Adresse geladen werden können. Relokative Daten und Programme werden nicht unterstützt.

3.3.3. Prüfen CAS

Wie LADEN CAS. Allerdings werden die Daten bzw. Texte nicht in den Speicher geladen, sondern nur mit den dort stehenden alten Werten verglichen. Deshalb dürfen die Werte zwischen SPEICHERN und PRÜFEN nicht verändert werden.

3.3.4. Booten

Mit diesem Menüpunkt kann ausgewählt werden, wie ein Betriebssystem, das sich auf einer Diskette oder einer Festplatte befindet, gestartet werden soll. Dabei können bis zu vier Diskettenlaufwerke oder anstelle eines Diskettenlaufwerks eine SCSI-Festplatte verwendet werden.

Bei der Wahl eines Diskettenlaufwerks wird der erste Sektor der Diskette in das RAM gelesen. Dann wird das Programm dieses Sektors gestartet, wenn der erste Befehl ein NOP ist. Außerdem wird eine Null im Register D0.L übergeben. Dadurch können eigene Programme erkennen, daß sie von einer Diskette gestartet wurden. Die Laufwerksnummer ergibt sich dabei aus Register D4.B, das entsprechend dem Befehl FLOPPY belegt ist. Kann der Sektor nicht gelesen werden oder steht kein NOP am Anfang, geht es zurück in das Grundprogramm.

Bei einem Festplattenstart wird ebenfalls der erste Sektor der Festplatte gelesen und das Programm gestartet, wenn ein NOP am Anfang steht. Hier wird aber eine Eins in D0.L übergeben. Auch hier werden Fehler erkannt.

3.4. Seite 3

Auf dieser Seite befinden sich die Programme zur Bedienung des Promer und des Promer2 sowie die Speicherbereichsanzeige, die eine Übersicht über RAM- und EPROM-Bereiche gibt. Außerdem ist ein Programm vorhanden für die Ausgabe eines Textes auf den Drucker und für die Druckersteuerung.

3.4.1. Eprom programmieren

Für den NDR-Klein-Computer gibt es zwei Baugruppen, mit denen man EPROMS brennen kann. Das Grundprogramm unterstützt beide Baugruppen. Für die PROMER-Baugruppe gibt es sogar zwei Programmieralgorithmen.

Bevor die Eproms programmiert werden, muß eingegeben werden, welche PROMER-Baugruppe angesprochen werden soll. Dazu erscheint ein Auswahlmenü, das folgende Möglichkeiten der Auswahl bietet:

A = Promerl 2716
B = Promerl 2732
C = Promerl 2764

Diese Programmiermodi setzen voraus, daß die PROMER-Baugruppe auf einen Programmierimpuls von 50 ms eingestellt ist. Dazu muß die PROMER-Baugruppe so aufgebaut sein, wie es im Handbuch beschrieben ist. Die vorstehenden Programmiermöglichkeiten stellen den Standard-Programmieralgorithmus für die PROMER-Baugruppe dar.

D = Promers 2716
E = Promers 2732
F = Promers 2764

Ist die PROMER-Baugruppe so abgeändert, daß der Impuls nur ca. 1 ms lang ist, kann mit einem anderen (intelligenten) Algorithmus programmiert werden. Er bringt einen Geschwindigkeitsvorteil, ist aber nicht für alle EPROMS geeignet.

Jetzt kommen die Auswahlmöglichkeiten für die PROMER2-Baugruppe, die ebenfalls mit einem intelligenten Verfahren programmiert wird (außer bei 2716- und 2732-Eproms). Die nachstehend genannten Programmierspannungen werden nur beim Schreiben in die Eproms benötigt. Beim Lesen der Eproms wird die Programmierspannung abgeschaltet und nur die Versorgungsspannung (5 Volt) eingestellt.

Epromtyp	Impulslänge	Programmierspannung	Versorgungsspannung
G = Promer2 2716	50,0 ms	25,00 Volt	5,00 Volt
H = Promer2 2732	50,0 ms	21,00 Volt	5,00 Volt
I = Promer2 2764a	1,0 ms	21,00 Volt	6,25 Volt
J = Promer2 2764b	1,0 ms	12,75 Volt	6,25 Volt
K = Promer2 27128a	1,0 ms	21,00 Volt	6,25 Volt
L = Promer2 27128b	0,5 ms	12,75 Volt	6,25 Volt
M = Promer2 27256	100 us	12,75 Volt	6,25 Volt
N = Promer2 27512	100 us	12,75 Volt	6,25 Volt
O = Promer2 27010	100 us	12,75 Volt	6,25 Volt

Außerdem gibt es die Möglichkeit, das zuletzt eingestellte EPROM auszuwählen. Das geschieht mit dem Punkt P, der auf Promers 2764 voreingestellt ist, dann aber immer das aktuelle Eprom anzeigt. Mit Z kommt man wieder in das Hauptmenü zurück.

Erst jetzt, wenn die PROMER-Baugruppe, der Epromtyp und der Programmiermodus eingestellt ist, darf das Eprom in das Programmier-Tool eingelegt werden. Das gilt natürlich nicht, wenn für das Brennen des nächsten Eproms die Einstellungen beibehalten werden sollen.

Nunmehr müssen die Adressen eingegeben werden. VON bezeichnet die Anfangsadresse im Speicher, von der ab Daten in ein Eprom gebrannt werden sollen, BIS die Adresse im Speicher, bis zu der die Daten im Eprom festgehalten werden sollen. NACH ist die Anfangsadresse im Eprom. Mit ABSTAND wird festgelegt, in welchem Abstand die Bytes aus dem Speicher geholt werden sollen. Dadurch können mit dem NKC unabhängig von der eingesetzten Prozessorkarte Eproms für jeden NKC gebrannt werden, unabhängig wieder davon, welche Prozessorkarte für den anderen NKC verwendet wird. Es sind die Eingaben 1, 2 und 4 möglich. Die Angabe 1 muß erfolgen, wenn das Eprom in einem 68008-System verwendet werden soll, die 2, wenn eine 68000-CPU eingesetzt wird, und die 4 bei Verwendung in einem 68020-System. Die Endadresse, die als zweites einzugeben ist, wird immer mitgebrannt, es sei denn, sie wird übersprungen, weil der Abstand ungleich 1 ist. Dann endet der Programmiervorgang bei der letzten Adresse davor.

Die Eingaben werden vom Grundprogramm auf ihre Richtigkeit überprüft. Sie müssen wiederholt werden, wenn die EPROM-Adressen überschritten werden.

Anschließend wird geprüft, ob der zu programmierende Bereich im EPROM frei, also mit $FF gefüllt ist. Ist das nicht der Fall, erscheint eine Fehlermeldung. Es kann dann aber trotzdem programmiert werden. Der Programmiervorgang kann mit ESC abgebrochen werden kann. Das funktioniert, wenn eine PROMER-Baugruppe vorhanden ist, ansonsten kann sich das Programm eventuell totlaufen. Fehlerhaft gebrannte Bytes werden sofort gemeldet. Außerdem wird in diesem Falle abgefragt, ob die Programmierung abgebrochen werden soll oder nicht. Sind die Daten aus dem eingegebenen Speicherbereich im Eprom eingebrannt, gibt das Grundprogramm eine Bestätigung über die erfolgreiche Programmierung des Eproms aus, nachdem noch einmal alle Werte überprüft wurden. Adressen werden nur bei einem auftretenden Fehler ausgegeben, da so die Programmierzeit sehr viel kürzer wird.

3.4.2. Eprom lesen

Zunächst muß das Eprom und damit auch die Baugruppe wie bei EPROM PROGRAMMIEREN ausgewählt werden. Dabei ist der Lesealgorithmus bei Promerl und Promers gleich. Anschließend wird die Anfangs- und Endadresse im EPROM eingegeben. Dann erfolgt die Eingabe der Zieladresse im RAM und der Abstand, in dem die Bytes abgelegt werden. Werden falsche Adressen eingegeben, so muß der Eingabevorgang wiederholt werden, ansonsten erfolgt sofort das Lesen.

3.4.3. Speicherbereiche

Es werden auf dem Bildschirm alle verfügbaren Speicherbereiche angezeigt. Die durchgehenden Flächen sind dabei RAM-Bereiche. Die straffierten Bereich sind Eproms, die mit Werten belegt sind. Leere ($00) oder volle ($FF) Eprom-Teile werden nicht angezeigt. Der Speicherbereich einer eventuell vorhandenen COL-Karte wird nicht ausgegeben.

In der obersten Zeile wird noch der Arbeitsspeicher für das Grundprogramm aufgeführt, der für Variable und die Symboltabelle verwendet werden kann und immer direkt hinter dem Grundprogramm beginnt.

3.4.4. Druckersteuerung

Oft ist es nötig, längere Texte bzw. Programme auszudrucken, da mit zunehmender Länge die Texte unübersichtlich werden und oft im Editor geblättert werden muß. Damit die Druckausgabe komfortabler durchgeführt werden kann, ist die Druckersteuerung vorhanden. Mit ihr werden verschiedene Einstellungen des Druckers vorgenommen. An sich ist die Druckersteuerung nur dafür gedacht, den aktuellen Editor-Text auszudrucken. Auf sie wird aber auch in der Routine CO beim Ausdrucken des Bildschirmspeichers zurückgegriffen. Die Einstellungen werden dabei durch Tastendruck geändert. Die aktuelle Einstellung wird jeweils angezeigt. Änderungen der Druckersteuerung werden nicht sofort an den Drucker übermittelt. Dafür gibt es einen gesonderten Befehl.

Hier die Befehle für die Druckersteuerung:

A = Seitenlänge

Die Seitenlänge gibt an, wie lang eine Seite sein soll. Dabei wird die Länge der Seite selbst, wie sie im Drucker eingestellt ist, nicht verändert. Es wird lediglich nach Erreichen der angegebenen letzten Zeile ein Seitenvorschub durchgeführt. Der mögliche Zeilen-Bereich geht dabei von 5 bis 127.

B = Linker Rand

Der linke Rand wird eingestellt. Dies geschieht durch einen Druckerbefehl und nicht durch Einfügen von Leerzeichen.

C = Druckart

Dadurch wird eingestellt, ob der gesamte Text normal, schmal oder breit ausgedruckt wird.

D = Schriftart

Es kann zwischen einem Ausdruck in PICA oder ELITE ausgewählt werden.

E = Druckrichtung

Normalerweise wird ein Ausdruck immer bidirektional ausgedruckt, was bedeutet, daß der Druckerkopf in beide Richtungen druckt. Wird aber ein genauerer Ausdruck gewünscht, kann auf unidirektionalen Ausdruck geschaltet werden. Da die Einstellungen nicht nur für die Druckersteuerung gelten, sondern auch für alle nachfolgenden Ausdrucke, kann dieser Befehl für nachfolgende GRAFIK-Ausdrucke verwendet werden, wobei die Spalten bei unidirektionalem Druck genauer untereinander stehen.

F = Langsamer Druck

Der langsamere Ausdruck ist für einen geringeren Geräuschpegel z.B. bei langem Ausdruck im Hintergrund gedacht.

G = Fettdruck

Beim Fettdruck wird eine Zeile zweimal ausgedruckt, wobei beim zweiten Druck der Zeile die Punkte eines jeden Buchstabens etwas versetzt werden.

H = Doppeldruck

Auch hierbei wird jede Zeile zweimal gedruckt, wobei die Punkte der Buchstaben übereinander liegen. Dadurch wird das Schriftbild intensiver.

I = Proportionaldruck

Normalerweise haben beim Ausdruck alle Buchstaben die gleiche Breite. Wird der Proportionaldruck eingeschaltet, ist das z. B. das i schmaler als das m. Dies entspricht mehr der normalen Schrift und wird meistens bei Briefpost verwendet.

J = Kursivdruck

Mit dieser Funktion wird jeder Buchstabe schräg gedruckt.

K = Papierendeerkennung

Normalerweise gibt jeder Drucker eine akustische Meldung aus, wenn das Papier zu Ende ist und hört mit dem Drucken auf. Soll aber die letzte Seite bis zum Ende bedruckt werden, kann die Erkennung unterdrückt werden.

L = Zeichensatz.

Es stehen drei verschiedene Zeichensätze zur Verfügung.

NDR Es wird automatisch zwischen deutschem und amerikanischem Zeichensatz umgeschaltet.

Deutsch Nur der deutsche Zeichensatz wird benutzt. Die Zeichen |, \, {, [, },] und ~ werden als ö, Ö, ä, Ä, ü, Ü und ß ausgegeben.

Amerikanisch Nur der amerikanische Zeichensatz wird benutzt. Die Zeichen ö, Ö, ä, Ä, ü, Ü und ß werden als |, \, {, [, },] und ~ ausgegeben.

N >

Hier können selbstdefinierte Druckerbefehle eingegeben werden. Dabei ist Platz für maximal 19 Bytes. Durch ESC oder ein leeres Eingabefeld wird die Eingabe abgeschlossen. Der Wert $FF ist als Befehl nicht erlaubt, da er intern als Endekennung verwendet wird. Diese Befehle werden beim Befehl P mit übergeben und zwar zuletzt. Dadurch können andere Befehle korrigiert werden. Durch diese Möglichkeit können außerdem zusätzliche Einstellungen am Drucker vorgenommen werden (z. B. Zeilenabstand).

M = Kopien

In der Regel wird der Text nur einmal ausgedruckt. Mit Hilfe dieser Funktion kann jeder Text bis zu achtmal ausgegeben werden. Nach jedem Ausdruck wird ein Seitenvorschub durchgeführt.

O = Werte rücksetzen

Alle Einstellungen werden auf ihren voreingestellten Wert zurückgesetzt.

P = Werte an Drucker

Die eingestellten Werte werden an den Drucker geschickt. Erst dadurch werden die bisherigen Einstellungen des Druckers geändert.

Q = Seitenvorschub

Der Wert $0C (= 12) wird an den Drucker geschickt und löst einen Seitenvorschub aus.

R = Editor ausdrucken

Dieser Befehl bewirkt den Ausdruck des aktuellen Editor-Textes.

S = Druckerbefehle ändern

Zu fast allen der vorstehenden Einstellungen gehören bestimmte Druckerbefehle. Da diese bei den einzelnen Druckern unterschiedlich sind, müssen sie ggf. verändert werden. Voreingestellt sind die Druckerbefehle nach dem Epson-ESC/P-Standard. Bei jedem Befehl können bis zu drei Bytes abgelegt werden. Ein Befehl kann auch keine Zeichen enthalten. Wird bei irgendeiner Eingabe RETURN eingegeben, so wird dadurch die Eingabe beendet. Die Eingabe eines Druckerbefehls wird durch Drücken des entsprechenden Buchstabens im Druckermenü eingeleitet, danach erfolgt die Eingabe. Es kann auch ein Befehl eingegeben werden, der nicht die ihm eigentlich zugewiesene Bedeutung hat. Dadurch kann jede der vorstehend beschriebenen Einstellungen sozusagen mißbraucht werden. So könnte beispielsweise mit dem Befehl "Kursivdruck an" auch eine Schriftart gewählt werden, die speziell für einen Drucker vorgesehen ist (z.B. Sans Serif). Mit W werden die ursprünglich voreingestellten Befehlsfolgen zurückgesetzt. Z macht die erste Seite der Druckersteuerung wieder sichtbar. Auf Disketten, die zum Grundprogramm erhältlich sind, gibt es auch ein Programm, das alle aktiven Druckerbefehle abspeichert, sodaß nach dem Anschalten des NKC nicht alle Befehlsfolgen erneut eingegeben werden müssen.

Z = Zurück

Hiermit wird die Druckersteuerung verlassen.

Zusätzlich zu diesen Befehlen bzw. Voreinstellungen kann der Ausdruck jederzeit mit ESC abgebrochen werden.

3.5. Seite 4

Jetzt kommt die letzte Seite. Sie enthält zwei Punkte zur Anzeige und Manipulation von beliebigen Adressen, wobei hauptsächlich an IO-Adressen gedacht ist. Weiterhin kann der sehr leistungsfähige EINZELSCHRITT gestartet werden. Er kann Programme Befehl für Befehl ausführen. Dadurch kann man Fehler in einem Listing leichter finden. Außerdem kann von hier aus auf die andere Menüform (neues Menü) umgeschaltet werden.

3.5.1. IO lesen

Da auf den IO-Adressen sehr oft veränderliche Daten liegen, die man einmal ansehen möchte, ist diese Funktion vorhanden. ANSEHEN ist nicht dafür geeignet, da die Werte schnell wechseln können. Durch die Anzeige des schnellen Wechsels ist dieser Menüpunkt auch sehr gut zum Testen von IO-Baugruppen geeignet.

Zuerst muß die gewünschte IO-Adresse angegeben werden. Dann wird der Wert von dieser Adresse gelesen. Die Ausgabe erfolgt dabei in binärer, dezimaler und hexadezimaler Form. Durch Drücken der Taste D wird auf einen Modus geschaltet, der den Wert dauernd (alle 20 ms) liest und ausgibt, wodurch fast jede Änderung ermittelt werden kann. Mit S wird der Auslesevorgang wieder angehalten. R ermöglicht die Eingabe einer neuen Adresse. Nach M ist wieder die Menüseite sichtbar.

3.5.2. IO setzen

Es gibt viele verschiedene IO-Baugruppen, die durch Schreiben von bestimmten Werten auf feste Adressen Befehle ausführen (GDP, AD/DA-Wandler, SOUND, usw.). Zur Prüfung der betreffenden Funktionen ist es möglich, auf eine bestimmte IO-Adresse einen Wert zu schreiben. Dazu wird erst die IO-Adresse angegeben, wobei auf die CPU geachtet werden muß, d. h., die IO-Adresse muß mit 1, 2 oder 4 multipliziert werden. Dann wird der Bytewert eingegeben, der auf die Adresse geschrieben werden soll. Dabei ist jede Eingabe gemäß der Konvention des WERT-Befehls erlaubt. Werden größere Werte als Byte-Werte eingegeben, so werden nur die Bits 0 bis 7 der Eingabe berücksichtigt. Mit R wird die Wiederholung des Befehls IO SETZEN eingeleitet. Mit M kehrt man in das Menü zurück.

3.5.3. Einzelschritt

Umfangreiche Programme funktionieren meistens nicht auf Anhieb fehlerfrei. Die groben Fehler sind aber leicht zu entdecken, nur bei den versteckten "Bugs" kann die Suche länger dauern. Diese Fehler verursachen dann ADRESS ERROR oder rufen merkwürdige Erscheinungen hervor. Für die Suche nach den Fehlern ist der EINZELSCHRITT bestimmt. Er ist das beste Mittel zum Ermitteln von Programmierfehlern.

Das Grundprogramm kann jedes Programm im EINZELSCHRITT abarbeiten. Dabei wird normalerweise jeder Befehl einzeln ausgeführt und in der untersten Zeile angezeigt. Absolute Unterprogrammaufrufe (JSR, TRAP) können als Ganzes ausgeführt werden. Die Unterprogramme werden dann nicht im einzelnen mitgeteilt. Außerdem werden alle Befehle, die in der DEBUG-Tabelle gefunden werden, aus dem Editor-Text mit eventuell vorhandenen Kommentaren ausgegeben. Nur wenn die Befehle nicht in dieser Tabelle gefunden werden bzw. keine DEBUG-Tabelle vorhanden ist, wird der DIS-Assembler eingeschaltet, der jeden Befehl übersetzen kann. Der DIS-Assembler versteht alle Befehle für den jeweiligen Prozessortyp und beim Prozessor 68020 auch die Befehle der FPU 68881.
Es gibt allerdings eine kleine Einschränkung beim Betrieb des EINZELSCHRITT: Intern werden im Grundprogramm die Speicher EINBUF und AUSBUF für den EINZELSCHRITT benötigt, wenn die Statuszeilen ebenfalls ausgegeben werden sollen. Da aber auch andere Unterprogramme im Grundprogramm diese Speicher benötigen, kann es zu erheblichen Fehlern kommen, wenn die Statuszeilen eingeschaltet sind und dieses Unterprogramm nicht als Ganzes, d. h. Befehl für Befehl abgearbeitet werden. Diesem Umstand kann nur abgeholfen werden, indem die Statuszeilen ausgeschaltet oder die Unterprogramme als Ganzes ausgeführt werden.

Der EINZELSCHRITT wird gestartet, indem die Adresse des gewünschten Programmes eingegeben wird. Danach wird der erste Befehl des Programmes auf dem Bildschirm ausgegeben. Nach Drücken der Taste RETURN wird dieser Befehl ausgeführt und der nächste Befehl angezeigt. Bei der Ausführung eines jeden Befehls werden am unteren Rand des Bildschirms alle dort dargestellten Informationen aktualisiert. Es werden dabei immer die Inhalte aller Register, über die der eingesetzte Prozessor verfügt, mitgeteilt. Im Zusammenhang mit dem Statusregister werden auch die Flags, die gesetzt sind, mit den sie kennzeichnenden Buchstaben ausgegeben, was die Übersichtlichkeit sehr erhöht.

Folgende Befehle sind vorhanden, mit deren Hilfe der EINZELSCHRITT auf sehr komfortable Weise bedient werden kann. Sie werden im übrigen beim Start des EINZELSCHRITT in einer Kurzübersicht angezeigt. Der Aufruf der Befehle erfolgt durch Eingabe der für sie in der Kurzübersicht mitgeteilten Buchstaben auf der Tastatur.

B = Bis ADRESSE/BEREICH ausführen

Das Programm wird so weit ausgeführt werden, bis der Befehl auf der Adresse ausgeführt wird, die vorher angegeben wurde. Ist nicht bekannt, auf welcher Adresse der betreffende Befehl steht, gibt man einen Bereich an, der in EINZELSCHRITT überprüft werden soll.
Wird ein Bereich angegeben, der nie erreicht wird, so wird das gesamte Programm im EINZELSCHRITT durchlaufen.

Beispiele für mögliche Eingaben:

$10000	Das Programm wird ausgeführt, bis die Adresse $10000 erreicht und der dort stehende Befehl ausgeführt ist.
$10000,100	Das Programm wird ausgeführt, bis ein Befehl im Bereich $10000-100 und $10000+100 erreicht wird.

C = Bildschirm löschen

Alle vier Seiten des Bildschirms der GDP werden gelöscht.

D = TRAP/ISR direkt oder indirekt

Normalerweise wird ein Unterprogramm, das mit JSR oder TRAP aufgerufen wird, immer als Ganzes ausgeführt, ohne daß die unter TRAP oder JSR zusammengefaßten Befehle, insbesondere Unterprogramme des Grundprogramms ausgegeben werden. Will man auch Grundprogrammaufrufe oder Aufrufe für JADOS (TRAP #6) im einzelnen nachverfolgen, kann man mit der D-Funktion des EINZELSCHRITT jedes mit JSR oder TRAP angesprochene Unterprogramm aus dem Grundprogramm Befehl für Befehl ausführen. Der D-Befehl kann an- oder ausgeschaltet werden. Ist auf Einzelausführung geschaltet, wird auf der Statuszeile rechts ein D ausgegeben, beim 68020-Grundprogramm nur auf der ersten Statusseite.

E = Bis zum nächsten RTS/RTE/(RTR beim 68020-Grundprogramm)

Das jeweilige Programm wird ausgeführt, bis ein RTS oder RTE oder beim 68020-Grundprogramm auch ein RTR gefunden wird. Dadurch kann man leicht das Ende des nächsten Unterprogramms finden.

F = FLAGS (CCR) setzen

Hiermit können die FLAGS manipuliert werden. Dadurch kann man Sprünge korrigieren oder erzwingen. Es werden nur die letzten 5 Bits der Eingabe beachtet und direkt in das CCR geladen.

G = Grundprogrammroutinen aufrufen

Es wird das Unterprogramm GRUND aufgerufen entsprechend der Zahleneingabe im Eingabefeld. Dadurch kann man jeden Menüpunkt während des Einzelschritts erreichen. Welche Eingaben zulässig sind, wird beim Unterprogramm GRUND in Kapitel 4.2 beschrieben. Durch den Aufruf wird allerdings bei den meisten Menüpunkten der Bildschirm gelöscht.

I = Info an/aus, beim 68020-Grundprogramm auch umschalten

Beim Grundprogramm 68008/68000:

Die Statuszeilen werden ein- oder ausgeschaltet. Nur wenn sie eingeschaltet sind, funktioniert die Funktion T.

Beim Grundprogramm 68020:

Die Statuszeilen werden umgeschaltet. Es gibt die Anzeige wie beim 68000, die die Daten- und Adressregister sowie einige Controlregister ausgibt. Die zweite Seite zeigt die Register der FPU und einige spezielle 68020-Register an. Als dritte Möglichkeit gibt es noch die Abschaltung der Statuszeilen. Nur wenn eine Statuszeile sichtbar ist, arbeitet die Funktion T.

J = Befehl einzeln/Unterbrechung bei JMP/JSR

(Nur beim 68020-Grundprogramm)

Dieser Befehl ist nur beim 68020-Grundprogramm verfügbar. Normalerweise werden beim Trace-Betrieb alle Befehle einzeln ausgeführt. Der Prozessor 68020 kennt aber auch eine TRACE-Verarbeitung, die die Programmausführung nur stoppt, wenn eine Unterbrechung des Flusses durchgeführt werden soll (Sprünge, TRAPs, Exceptions). Mit dem Befehl kann zwischen den beiden Trace-Verarbeitungen umgeschaltet werden.

L = B/E/N/W wiederholen

Damit können die Befehle B, E, N und W wiederholt werden. Es sind keine weiteren Eingaben nötig. Es wird immer der zuletzt benutzte Befehl aus dieser Gruppe wiederholt.

N = N Befehle ausführen.

Es werden so viele Befehle als Ganzes ausgeführt, wie die einzugebende Zahl angibt.

P = PC neu laden (neue Programmadresse)

Wenn in einem Programm Teile übersprungen werden sollen, so kann hiermit der aktuelle PROGRAMM-COUNTER verändert werden. Vorsicht ist dabei aber geboten, da es zu Stackfehlern kommen kann. Die neue Adresse sollte deshalb auf der gleichen Unterprogrammebene liegen. Soll eine andere Ebene erreicht werden, so muß der Stack korrigiert werden.

R = Register setzen

Alle Daten- und Adressregister sind direkt manipulierbar. Beim Prozessor 68020 können auch die Datenregister der FPU verändert werden. Dazu ist das Register in dem Eingabefeld anzugeben (z.B. D4, A7, FP3). Danach wird in dem Eingabefeld der Inhalt des Registers ausgegeben. Er kann jetzt verändert werden. Wird bei der Registernummer eine 8 eingegeben, werden nacheinander alle Registerinhalte angezeigt und können ebenfalls verändert werden.

S = Leseseite auswählen

Es kann die Leseseite bestimmt werden. Daher kann jede Seite angesehen kann, auch wenn ein Programm auf eine unsichtbare Seite schreibt. Es sind die Werte 0..3 erlaubt, da die GDP-Karte 4 Seiten zur Verfügung stellt.

T = Tabellieren auf Drucker

Wird diese Funktion aktiviert, wird das Wort LIST in den unteren Zeilen ausgegeben (Beim 68020-Grundprogramm nicht auf der zweiten Seite), sofern diese Zeilen angeschaltet sind. Dies bedeutet, daß jeder Befehl, der auch auf dem Bildschirm ausgegeben wird, zusätzlich noch über die CENT-Baugruppe ausgedruckt wird. Dadurch kann ein Programmverlauf kontrolliert werden.

W = Weiter bis Adresse & Maske = Wert

Zunächst muß eine Adresse eingegeben werden. Dann erfolgt die Eingabe einer 8-Bit-Maske und die Eingabe eines Wertes. Das Programm wird so weit ausgeführt, bis unter der Maske auf der Adresse der Wert erscheint.

Beispiel:

Adresse $FFFFFF68 Maske $01 Wert $01
Das Programm wird ausgeführt, bis das Bit 0 auf Adresse $FFFFFF68 zu 1 wird.

Adresse $FFFFFF68 Maske $F1 Wert $01
Das Programm wird ausgeführt, bis das Bit 0 auf Adresse $FFFFFF68 zu 1 wird, und die Bits 4 bis 7 zu 0 werden.

3.5.4. Menüaufbau

Beim Grundprogramm bis einschließlich Version 4.3 waren die einzelnen Menüfunktionen auf mehrere Seiten verteilt, die man durchsuchen bzw. durchlaufen mußte, um eine bestimmte Funktion aufzurufen. Durch den Befehl "Menüaufbau" kann man auf eine einseitige Darstellung umschalten, die übersichtlicher und leichter zu bedienen ist. Das einseitige Menü kann mit dem DIL-Schalter auf der KEY voreingestellt werden (siehe Kapitel 2.6.).

Der Aufruf der einzelnen Menüfunktionen auf der einseitigen Menütafel ist leicht möglich. Vor jeder Funktion befindet sich ein Buchstabe bzw. eine Zahl. Durch Eingabe des Buchstabens oder der Zahl wird die gewünschte Menüfunktion aufgerufen. Der Aufruf der Menüpunkte kann auch mit der Maus erfolgen. Diese muß vorher mit "0" aktiviert werden. Dann kann eine Markierung auf einen Punkt gesetzt werden. Durch Betätigen der linken Maustaste wird diese Funktion aufgerufen.

Als zusätzliche Funktionen werden in den beiden untersten Zeilen noch einige Daten über die Einstellungen im Grundprogramm angezeigt. Die Daten erklären sich selbst. In der rechten unteren Ecke wird das Datum und die Uhrzeit eingeblendet, wenn eine Uhr vorhanden ist. Beim Stellen der Uhr ist noch zu bemerken, daß der Wochentag nicht als Buchstabe, sondern als Zahl eingegeben wird (0 = Montag, 1 = Dienstag usw.).

Wurde das Grundprogramm durch das Unterprogramm GRUND (siehe Kapitel 4) aufgerufen, ist es möglich, mit Z das so aufgerufene Grundprogramm zu beenden und die Kontrolle wieder an das aufrufende Programm zu übergeben. Nur im einseitigen Menü ist ein Rücksprung möglich.

Wem weder die alte noch die neue Menüform gefällt, der kann sich leicht eine eigene Oberfläche programmieren, da jeder Menüpunkt einzeln aufgerufen werden kann. Wie dies geschieht, steht im Kapitel 4 beim Unterprogramm GRUND.

Kapitel 4 - Unterprogramme

Das Grundprogramm enthält fast 150 Programmroutinen (Unterprogramme), die beim Programmieren einbezogen werden können. Um die Verwendung der Unterprogramme in eigenen Anwendungen zu erleichtern, ist der Assembler mit einer Funktion ausgestattet, die ihn erkennen läßt, ob ein solches Unterprogramm benutzt werden soll. Die Namen der Unterprogramme sind zu diesem Zweck in einer internen Tabelle abgelegt.

Die Unterprogramme können auf zweierlei Weise aufgerufen werden.

Aufruf mit JSR @NAME

Die Unterprogramme haben, wie man leicht aus den folgenden Tabellen erkennt, alle einen bestimmten Namen. Mit deren Hilfe kann der Assembler die Adresse des jeweiligen Programms ermitteln. Allerdings laufen Programme, die die Unterprogramme verwenden, möglicherweise nicht auf anderen Computern mit 680xx-Prozessoren. Die Programmierung mit JSR @NAME kommt deshalb eigentlich nur für kleinere Programme in Betracht, die man einmal zwischendurch benötigt. Bei dieser Programmierung muß darauf geachtet werden, daß das Adressregister A5.L der 680xx-Prozessoren mit dem richtigen Wert belegt ist (siehe Kapitel 4.2. unter SETA5).

```
Beispiel:       JSR     @SCHREITE
```

Aufruf über den TRAP-Mechanismus

Dieser Aufruf ist für alle fortgeschrittenen Programmierer Pflicht, die nicht nur für sich selbst Programme schreiben und auch in Zukunft kompatibel mit neuen Grundprogrammen bleiben wollen.

Bei dieser Programmierung wird im Register D7.W die Zahl abgelegt, die der Nummer des Unterprogramms entspricht. Dann wird der Befehl TRAP #1 ausgeführt. Beim Aufruf des Unterprogramms springt das Programm auf eine feste Adresse, von der aus das betreffende Unterprogramm aufgerufen wird. Dabei wird auch gleich das Register A5.L gesetzt. Den genauen Ablauf eines Programmteils, der den TRAP-Befehl verwendet wird, kann man Büchern über die 680xx-Prozessoren nachlesen.

```
Beispiel:       MOVEQ   #!SCHREITE,D7
                TRAP    #1
```

4.1. Tabellen der Unterprogramme

Die folgenden Tabellen sollen es ermöglichen, möglichst schnell ein gewünschtes Unterprogramm zu finden. Außerdem können aus den Tabellen verschiedene Informationen über das jeweilige Unterprogramm entnommen werden. Falls man vergessen hat, welche Ein- oder Ausgaberegister für die jeweilige Funktion benötigt werden oder welche Register beim Aufruf eines Unterprogramms zerstört werden, kann man auch das nachsehen.

Zu beachten ist, daß beim Aufruf eines Unterprogramms über den TRAP-Befehl immer die bisherigen Inhalte der Register D7 und A6 überschrieben werden. Der CPU-Status (USER- oder SUPERVISOR-Modus) bleibt beim Aufruf von Unterprogrammen erhalten. Die Flags bleiben so, wie das Grundprogramm-Unterprogramm sie hinterläßt. Allerdings sollten die Anwenderprogramme immer nur im SUPERVISOR-Modus ablaufen, da intern im Grundprogramm priviligierte Befehle benutzt werden. Der Computer befindet sich beim Einschalten immer im SUPERVISOR-Modus, so daß nichts geändert werden muß.

4.1.1 Liste der Unterprogramme nach TRAP-Nummern - Teil 1

TRAP Nr.	Befehls-name	Kurzbeschreibung	Ab Vers.	Befehlsgruppe
1	SCHREITE	Schildkröte schreitet	3.1	Schildkrötengrafik
2	DREHE	Schildkröte dreht sich	3.1	Schildkrötengrafik
3	HEBE	Schildkrötenspur ausschalten	3.1	Schildkrötengrafik
4	SENKE	Schildkrötenspur einschalten	3.1	Schildkrötengrafik
5	FIGURXY	Figur mit X-Y-Vergrößerung zeichnen	6.0	Figurgrafik
6	WRITELF	Text auf dem Bildschirm ausgeben	6.0	Textausgabe
7	SET	Schildkröte absolut setzen	3.1	Schildkrötengrafik
8	MOVETO	Position GDP setzen	3.1	Grafik
9	DRAWTO	Linie zur neuen X-Y-Position zeichnen	3.1	Grafik
10	WRITE	Text auf dem Bildschirm ausgeben	3.1	Textausgabe
11	READ	Text vom Bildschirm einlesen	3.1	Texteingabe
12	CI	Zeichen über Tastatur oder SER lesen	3.1	Zeicheneingabe
13	CSTS	Zeichen von Tastatur oder SER da?	3.1	Zeicheneingabe
14	RI	Zeichen von CAS lesen	3.1	CAS-Baugruppe
15	PO	Zeichen über CAS schreiben	3.1	CAS-Baugruppe
16	CLR	Alle Bildschirmseiten löschen	3.1	Grafik
17	CLPG	Eine Bildschirmseite löschen	3.1	Grafik
18	WAIT	Warten, bis GDP fertig	3.1	Grafik
19	SCHR16TEL	Schildkröte schreitet	3.1	Schildkrötengrafik
20	CLRSCREEN	Bildschirm löschen und Cursor setzen	3.1	Zeichenausgabe
21	CO	Zeichen auf Bildschirm ausgeben	3.1	Zeichenausgabe
22	LO	Zeichen über Centronics ausgeben	3.1	Zeichenausgabe
23	SIN	Sinus*256 berechnen	3.1	Berechnungen
24	COS	Cosinus*256 berechnen	3.1	Berechnungen
25	SIZE	Zeichengröße einstellen (für CO2 ..)	3.1	Zeichenausgabe
26	CMD	Befehl an GDP	3.1	Grafik
27	NEWPAGE	Schreibseite und Leseseite GDP setzen	3.1	Grafik
28	SYNC	Vertikal-Synchronimpuls GDP abfragen	3.1	Synchronisation
29	WERT	Ganzzahligen Wert berechnen	3.1	Berechnungen
30	ZUWEIS	Symbol in Symboltabelle eintragen	3.1	Symboltabelle
31	CIINIT2	Zeiger auf Textstart setzen	3.1	Zeicheneingabe
32	CI2	Zeichen aus Speicher lesen (umlenkbar)	3.1	Zeicheneingabe
33	CO2	Zeichen- und Steuerzeichenausgabe	3.1	Zeichenausgabe
34	SETFLIP	Umschaltrate für Autoflip setzen	3.1	Grafik
35	DELAY	Warteschleife in 10-tel Sekunden	3.1	Synchronisation
36	FIRSTTIME	Neustart Schildkröte	3.1	Schildkrötengrafik
37	SETPEN	Schreiber in GDP auf SCHREIBEN	3.1	Grafik
38	ERAPEN	Schreiber in GDP auf LÖSCHEN	3.1	Grafik
39	GRAPOFF	Schildkrötengrafik ausschalten	3.1	Schildkrötengrafik
40	CMDPRINT	Befehlsausgabe mehrerer Befehle an GDP	3.1	Grafik
41	PRINT2X	Sedezimale Ausgabe 2 Stellen (1 Byte)	3.1	Wertausgabe
42	PRINT4X	Sedezimale Ausgabe 4 Stellen (2 Bytes)	3.1	Wertausgabe
43	PRINT6X	Sedezimale Ausgabe 6 Stellen (3 Bytes)	3.1	Wertausgabe
44	PRINT8X	Sedezimale Ausgabe 8 Stellen (4 Bytes)	3.1	Wertausgabe
45	PRINT8B	Binäre Ausgabe 8 Bit (1 Byte)	3.1	Wertausgabe
46	PRINT4D	Dezimale Ausgabe ohne Vorzeichen	3.1	Wertausgabe
47	HIDE	Schildkröte ausblenden	3.1	Schildkrötengrafik
48	SHOW	Schildkröte einblenden	3.1	Schildkrötengrafik
49	CRT	CO2 auf Bildschirm lenken	3.1	Zeichenausgabe
50	LST	CO2 auf Drucker lenken	3.1	Zeichenausgabe

TRAP Nr.	Befehls-name	Kurzbeschreibung	Ab Vers.	Befehlsgruppe
51	USR	CO2 auf Benutzerschnittstelle lenken	3.1	Zeichenausgabe
52	NIL	CO2 ins Leere lenken	3.1	Zeichenausgabe
53	SETERR	Fehlercode setzen	3.1	Assembler
54	GETERR	Fehlercode lesen	3.1	Assembler
55	SETPASS	Laufnummer setzen	3.1	Zeichenausgabe
56	EDIT	Editor aufrufen	3.1	Editor
57	FIGUR	Figur mit XY-Vergrößerung zeichnen	3.1	Figurgrafik
58	SETFIG	Figur festsetzen (einfrieren)	3.1	Figurgrafik
59	GETRAM	Ram-Bereich testen	3.1	System-Routinen
60	AUTOFLIP	Automatische Bildseitenumschaltung	3.1	Grafik
61	CURSEIN	Cursor einblenden	3.1	Zeichenausgabe
62	CURSAUS	Cursor ausblenden	3.1	Zeichenausgabe
63	CHAR	Zeichen ausgeben	3.1	Zeichenausgabe
64	PROGZGE	Frei definierbares Zeichen ausgeben	3.1	Textausgabe
65	ASSEMBLE	Assembler aufrufen	3.1	Assembler
66	GETSTX	Textstart-Adresse lesen	3.1	Editor
67	PUTSTX	Textstart-Adresse setzen	3.1	Editor
68	GETORG	Default-ORG-Adresse lesen	3.1	Assembler
69	PUTORG	Default-ORG-Adresse setzen	3.1	Assembler
70	PRINT8D	Dezimale Ausgabe ohne Vorzeichen	4.0	Wertausgabe
71	PRINTV8D	Dezimale Ausgabe mit Vorzeichen	4.0	Wertausgabe
72	MULS32	32-Bit Multiplikation	4.0	Berechnungen
73	DIVS32	32-Bit-Division	4.0	Berechnungen
74	FLINIT	FLOPPY-Variablen initialisieren	4.0	FLO-Baugruppe
75	FLOPPY	FLOPPY-Befehl ausführen	4.0	FLO-Baugruppe
76	GETFLOP	FLOPPY-Format feststellen	4.0	FLO-Baugruppe
77	SETXOR	XOR-Modus GDP setzen	4.0	Grafik
78	GETXOR	XOR-Modus GDP lesen	4.0	Grafik
79	SETCOLOR	Farbcode GDP setzen	4.0	Grafik
80	GETCOLOR	Farbcode GDP lesen	4.0	Grafik
81	CURON	Cursor zeigen bei CO, CO2, CHAR	4.0	Zeichenausgabe
82	CUROFF	Cursor nicht zeigen bei CO, CO2, CHAR	4.0	Zeichenausgabe
83	ADJ360	Wert auf 360-Grad-Bereich beschränken	4.0	Berechnungen
84	PRTSYM	Symboltabelle ausgeben	4.0	Symboltabelle
85	SYMCLR	Symboltabelle löschen	4.0	Symboltabelle
86	GETSYM	Symboltabellenstart lesen	4.0	Symboltabelle
87	GETNEXT	Offset für Symboltabelle lesen	4.0	Symboltabelle
88	PUTNEXT	Offset für Symboltabelle setzen	4.0	Symboltabelle
89	GETBASIS	Basis-Adresse des Grundprogramms lesen	4.0	System-Routinen
90	GETVAR	Adresse des Arbeitsspeichers lesen	4.0	System-Routinen
91	SETA5	RAM-Zeiger A5 wieder gültig setzen	4.0	System-Routinen
92	AUFXY	Schildkröte absolut setzen	4.0	Schildkrötengrafik
93	KORXY	Schildkrötenkoordinaten lesen	4.0	Schildkrötengrafik
94	AUFK	Schildkrötenrichtung setzen	4.0	Schildkrötengrafik
95	GETK	Schildkrötenrichtung lesen	4.0	Schildkrötengrafik
96	RND	Zufallsgenerator	4.0	Berechnungen
97	GETVERS	Versionsnummer Grundprogramm lesen	4.0	System-Routinen
98	GETSN	Seriennummer Grundprogramm lesen	4.0	System-Routinen
99	CRLF	Zeilenvorschub über CO2 ausgeben	4.0	Zeichenausgabe
100	GETLINE	Zeilenadresse Bildwiederholspeicher lesen	4.0	Zeichenausgabe

TRAP Nr.	Befehls-name	Kurzbeschreibung	Ab Vers.	Befehlsgruppe
101	GETCURXY	Cursorposition lesen	4.0	Zeichenausgabe
102	SETCURXY	Cursorposition setzen	4.0	Zeichenausgabe
103	GETXY	Aktuelle GDP-Position lesen	4.0	Grafik
104	SI	Zeichen von serieller Schnittstelle	4.0	SER-Baugruppe
105	SO	Zeichen an serielle Schnittstelle	4.0	SER-Baugruppe
106	SISTS	Zeichen von SER da?	4.0	SER-Baugruppe
107	SOSTS	SER bereit zur Ausgabe?	4.0	SER-Baugruppe
108	SIINIT	SER initialisieren (Baudrate, Parität)	4.0	SER-Baugruppe
109	GETAD8	Wert vom 8-Bit AD-Wandler lesen	4.0	AD/DA-Baugruppen
110	GETAD10	Wert vom 10-Bit AD-Wandler lesen	4.0	AD/DA-Baugruppen
111	SETDA	DA-Wandler setzen	4.0	AD/DA-Baugruppen
112	SPEAK	Sprachausgabe mit 5 Parametern	4.0	Sprache und Sound
113	SPEAK1	Sprachausgabe nur Phonemcode	4.0	Sprache und Sound
114	SOUND	SOUND-Generator setzen	4.0	Sprache und Sound
115	GETUHR	Uhrzeit und Datum lesen	4.0	Echtzeituhren
116	SETUHR	Uhrzeit und Datum setzen	4.0	Echtzeituhren
117	LSTS	Drucker bereit?	4.0	Zeichenausgabe
118	RELAN	Relais auf CAS2 einschalten	4.0	CAS-Baugruppe
119	RELAUS	Relais auf CAS2 ausschalten	4.0	CAS-Baugruppe
120	ASSERR	Fehleranzahl Assembler lesen	4.0	Assembler
121	PRINTFP0	Wert in FP0 in ASCII wandeln	5.0	68020-FPU
122	GETFLOAT	FP-Zahl in ASCII in BCD wandeln	5.0	68020-FPU
123	READAUS	Texteingabe mit vorheriger Ausgabe	6.0	Texteingabe
124	GRUND	Grundprogramm als Unterprog. aufrufen	6.0	System-Routinen
125	HARDCOPY	Ansteuerung der HARDCOPY-Baugruppe	6.0	Grafik
126	GRAFIK	Großes GRAFIK-Paket für GDP und COL	6.0	Grafik
127	GDPVERS	GDP oder GDPHS?	6.0	Grafik
128	SER	CI, LO oder FLOPPY auf SER lenken	6.0	SER-Baugruppe
129	CO2SER	CO2 auf SER lenken	6.0	Zeichenausgabe
130	CLUTINIT	CLUT-Baugruppe auf Standardwerte	6.0	CLUT-Baugruppe
131	CLUT	CLUT-Farbwerte beliebig setzen	6.0	CLUT-Baugruppe
132	RELAIS	Relais auf RELAIS-Baugruppe setzen	6.0	RELAIS-Baugruppe
133	RELAISIN	Port der RELAIS-Baugruppe lesen	6.0	RELAIS-Baugruppe
134	SETDA12	DA-Wandler auf AD/DA-Baugruppe setzen	6.0	AD/DA-Baugruppen
135	GETAD12	AD-WANDLER auf AD/DA-Baugruppe lesen	6.0	AD/DA-Baugruppen
136	DISASS	Einen Befehl disassemblieren	6.1	DIS-Assembler
137	SUCHBIBO	Bibliothekseinträge suchen	6.1	System-Routinen
138	SI2	Zeichen von serieller Schnittstelle	6.1	SER-Baugruppe
139	SYSTEM	System-Informationen lesen	6.1	System-Routinen
140	UHRPRINT	Uhrzeit und Datum in ASCII wandeln	6.2	Echtzeituhren
141	HARDDISK	Befehl an Harddisk geben und ausführen	6.2	SCSI-Harddisk
142	HARDTEST	Prüft, ob eine Harddisk vorhanden ist	6.2	SCSI-Harddisk
143		Reserviert		
144		Reserviert		
145	PRTFP0	Wert in FP0 in ASCII wandeln	6.1	68020-FPU
146	FPUWERT	FP-Zahl berechnen und FP0 laden	6.1	68020-FPU
147	SETFPX	FP-Variable X setzen	6.1	68020-FPU
148	SETFPY	FP-Variable Y setzen	6.1	68020-FPU

4.1.2. Liste der Unterprogramme nach TRAP-Nummern - Teil 2

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
1	SCHREITE	d0.w	keine	d0
2	DREHE	d0.w	keine	d0
3	HEBE	keine	keine	keine
4	SENKE	keine	keine	keine
5	FIGURXY	d0.w-d2.w/a0.l	keine	keine
6	WRITELF	d0.b/d1.w/d2.w/a0.l	keine	keine
7	SET	d1.w-d3.w	d0.w	keine
8	MOVETO	d1.w/d2.w	keine	keine
9	DRAWTO	d1.w/d2.w	keine	keine
10	WRITE	d0.b/d1.w/d2.w/a0.l	keine	keine
11	READ	d0.b/d1.w-d3.w/a0.l	d4.l/d5.l/a0.l/Carry	keine
12	CI	keine	d0.l	keine
13	CSTS	keine	d0.l/Flags(d0.b)	keine
14	RI	keine	d0.l/Carry/Flags(d0.b)	keine
15	PO	d0.b	Carry	keine
16	CLR	keine	keine	d0
17	CLPG	keine	keine	keine
18	WAIT	keine	keine	keine
19	SCHR16TEL	d0.w	keine	d0
20	CLRSCREEN	keine	keine	keine
21	CO	d0.b	keine	d0/a0-a2
22	LO	d0.b	keine	keine
23	SIN	d0.w	d0.w/Flags	keine
24	COS	d0.w	d0.w/Flags	keine
25	SIZE	d0.b	keine	keine
26	CMD	d0.b	keine	keine
27	NEWPAGE	d0.b/d1.b	d0.b	keine
28	SYNC	keine	d0.w/Flags	keine
29	WERT	a0.l	d0.l/d1.l/a0.l/Carry	d2/d3/a1-a3
30	ZUWEIS	a0.l	a0.l/(d0.l/d1.l/a3.l)	d2/d3/a1/a2
31	CIINIT2	keine	keine	keine
32	CI2	keine	d0.l/Carry	keine
33	CO2	d0.b	Carry	keine
34	SETFLIP	d0.b/d1.b	keine	keine
35	DELAY	d0.l	keine	d0
36	FIRSTTIME	keine	keine	keine
37	SETPEN	keine	keine	keine
38	ERAPEN	keine	keine	keine
39	GRAPOFF	keine	keine	keine
40	CMDPRINT	d0.b/d1.w/d2.w/a0.l	a0.l	d0
41	PRINT2X	d0.b/a0.l	a0.l	keine
42	PRINT4X	d0.w/a0.l	a0.l	keine
43	PRINT6X	d0.l/a0.l	a0.l	keine
44	PRINT8X	d0.l/a0.l	a0.l	keine
45	PRINT8B	d0.b/a0.l	a0.l	keine
46	PRINT4D	d0.w/a0.l	a0.l	d0
47	HIDE	keine	d0.b	keine
48	SHOW	keine	d0.b	keine
49	CRT	keine	keine	keine
50	LST	keine	keine	keine

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
51	USR	keine	keine	keine
52	NIL	keine	keine	keine
53	SETERR	d0.w	keine	keine
54	GETERR	keine	d0.w	keine
55	SETPASS	d0.w	keine	keine
56	EDIT	keine	Carry(d0.l)	alle außer a5
57	FIGUR	d0.b/d1.w/d2.w/a0.l	d0.w	keine
58	SETFIG	keine	keine	keine
59	GETRAM	a0.l	d0.l/a0.l/a1.l/Carry	keine
60	AUTOFLIP	keine	keine	d0
61	CURSEIN	keine	keine	keine
62	CURSAUS	keine	keine	keine
63	CHAR	d0.b	keine	d0/a0-a2
64	PROGZGE	a0.l	keine	keine
65	ASSEMBLE	keine	Carry	alle außer a5
66	GETSTX	keine	d0.l	keine
67	PUTSTX	d0.l	a0.l	keine
68	GETORG	keine	d0.l	keine
69	PUTORG	d0.l	keine	keine
70	PRINT8D	d0.l/a0.l	a0.l	d0
71	PRINTV8D	d0.l/a0.l	a0.l	d0
72	MULS32	d0.l/d2.l	d0.l/d1.l	keine
73	DIVS32	d0.l/d2.l	d0.l/d1.l	keine
74	FLINIT	keine	keine	keine
75	FLOPPY	d1.w/d2.b-d4.b/a0.l	d0.w/Carry	keine
76	GETFLOP	d4.b	d0.w/d1.b/d4.w/Carry	keine
77	SETXOR	d0.w	d0.b	keine
78	GETXOR	keine	d0.l	keine
79	SETCOLOR	d0.b	d0.b	keine
80	GETCOLOR	keine	d0.l	keine
81	CURON	keine	keine	keine
82	CUROFF	keine	keine	keine
83	ADJ360	d0.w	d0.w	keine
84	PRTSYM	keine	keine	keine
85	SYMCLR	keine	a0.l	keine
86	GETSYM	keine	d0.l/a0.l	keine
87	GETNEXT	keine	d0.l	keine
88	PUTNEXT	d0.w	keine	keine
89	GETBASIS	keine	d0.l/a0.l	keine
90	GETVAR	keine	d0.l/a0.l	keine
91	SETA5	keine	a5.l	keine
92	AUFXY	d1.w/d2.w	keine	keine
93	KORXY	keine	d1.l/d2.l	keine
94	AUFK	d0.w	keine	keine
95	GETK	keine	d0.w	keine
96	RND	d0.w	d0.l	keine
97	GETVERS	keine	d0.l	keine
98	GETSN	keine	d0.l	keine
99	CRLF	keine	Carry	d0
100	GETLINE	d0.b	a0.l-a2.l	d0

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
101	GETCURXY	keine	d1.l/d2.l	keine
102	SETCURXY	d1.b/d2.b	keine	keine
103	GETXY	keine	d1.l/d2.l	keine
104	SI	keine	d0.l	keine
105	SO	d0.b	keine	keine
106	SISTS	keine	d0.l/Flags	keine
107	SOSTS	keine	d0.l/Flags	keine
108	SIINIT	d0.b/d1.b	keine	keine
109	GETAD8	d0.b	d0.l	keine
110	GETAD10	keine	d0.l	keine
111	SETDA	d1.b/d2.b	keine	keine
112	SPEAK	a0.l	keine	keine
113	SPEAK1	a0.l	keine	keine
114	SOUND	a0.l	keine	keine
115	GETUHR	a0.l	a0.l	keine
116	SETUHR	a0.l	a0.l	keine
117	LSTS	keine	d0.l/Flags(d0.b)	keine
118	RELAN	keine	keine	keine
119	RELAUS	keine	keine	keine
120	ASSERR	keine	d0.l	keine
121	PRINTFP0	d0.w/a0.l/fp0.x	a0.l	keine
122	GETFLOAT	a0.l/a1.l	a0.l/Carry	keine
123	READAUS	d0.b/d1.w-d3.w/a0.l	d4.l/d5.l/a0.l/Carry	keine
124	GRUND	d0.w	keine	keine
125	HARDCOPY	Verschiedene Ein- und Ausgaberegister		d7
126	GRAFIK	Verschiedene Ein- und Ausgaberegister		d7/a6
127	GDPVERS	keine	d0.l	keine
128	SER	d0.b	Carry	keine
129	CO2SER	keine	Carry	keine
130	CLUTINIT	keine	keine	keine
131	CLUT	a0.l	keine	keine
132	RELAIS	d0.b/a0.l	a0.l	keine
133	RELAISIN	a0.l	d0.b/a0.l	keine
134	SETDA12	d0.w/d1.w	keine	keine
135	GETAD12	d1.b	d0.w	keine
136	DISASS	d0.l/a0.l	keine	keine
137	SUCHBIBO	d0.w/(d2.l/d3.l/a0.l/a1.l)	Carry/(a1.l/(d1.l-d7.l))	d0/(d1-d2)
138	SI2	keine	d0.l	keine
139	SYSTEM	keine	d0.l	keine
140	UHRPRINT	d0.w/a0.l	a0.l/Carry	keine
141	HARDDISK	d1.b/d4.b/(d2/d3/a0/a1)	d0.l/Carry	keine
142	HARDTEST	d4.b	d0.l/Carry	keine
143		Reserviert		
144		Reserviert		
145	PRTFP0	d0.w/a0.l/fp0.x	a0.l/Carry	keine
146	FPUWERT	a0.l	d1.l/a0.l/fp0.x/Carry	keine
147	SETFPX	fp0.x	keine	keine
148	SETFPY	fp0.x	keine	keine
149	SETFPZ	fp0.x	keine	keine

4.1.3. Liste der Unterprogramme nach Namen - Teil 1

Befehls-name	TRAP Nr.	Kurzbeschreibung	Ab Vers.	Befehlsgruppe
ADJ360	83	Wert auf 360-Grad-Bereich beschränken	4.0	Berechnungen
ASSEMBLE	65	Assembler aufrufen	3.1	Assembler
ASSERR	120	Fehleranzahl Assembler lesen	4.0	Assembler
AUFK	94	Schildkrötenrichtung setzen	4.0	Schildkrötengrafik
AUFXY	92	Schildkröte absolut setzen	4.0	Schildkrötengrafik
AUTOFLIP	60	Automatische Bildseitenumschaltung	3.1	Grafik
CHAR	63	Zeichen ausgeben	3.1	Zeichenausgabe
CI	12	Zeichen über Tastatur oder SER lesen	3.1	Zeicheneingabe
CI2	32	Zeichen aus Speicher lesen (umlenkbar)	3.1	Zeicheneingabe
CIINIT2	31	Zeiger auf Textstart setzen	3.1	Zeicheneingabe
CLPG	17	Eine Bildschirmseite löschen	3.1	Grafik
CLR	16	Alle Bildschirmseiten löschen	3.1	Grafik
CLRSCREEN	20	Bildschirm löschen und Cursor setzen	3.1	Zeichenausgabe
CLUT	131	CLUT-Farbwerte beliebig setzen	6.0	CLUT-Baugruppe
CLUTINIT	130	CLUT-Baugruppe auf Standardwerte	6.0	CLUT-Baugruppe
CMD	26	Befehl an GDP	3.1	Grafik
CMDPRINT	40	Befehlsausgabe mehrerer Befehle an GDP	3.1	Grafik
CO	21	Zeichen auf Bildschirm ausgeben	3.1	Zeichenausgabe
CO2	33	Zeichen- und Steuerzeichenausgabe	3.1	Zeichenausgabe
CO2SER	129	CO2 auf SER lenken	6.0	Zeichenausgabe
COS	24	Cosinus*256 berechnen	3.1	Berechnungen
CRLF	99	Zeilenvorschub über CO2 ausgeben	4.0	Zeichenausgabe
CRT	49	CO2 auf Bildschirm lenken	3.1	Zeichenausgabe
CSTS	13	Zeichen von Tastatur oder SER da?	3.1	Zeicheneingabe
CUROFF	82	Cursor nicht zeigen bei CO, CO2, CHAR	4.0	Zeichenausgabe
CURON	81	Cursor zeigen bei CO, CO2, CHAR	4.0	Zeichenausgabe
CURSAUS	62	Cursor ausblenden	3.1	Zeichenausgabe
CURSEIN	61	Cursor einblenden	3.1	Zeichenausgabe
DELAY	35	Warteschleife in 10-tel Sekunden	3.1	Synchronisation
DISASS	136	Einen Befehl disassemblieren	6.1	DIS-Assembler
DIVS32	73	32-Bit-Division	4.0	Berechnungen
DRAWTO	9	Linie zur neuen X-Y-Position zeichnen	3.1	Grafik
DREHE	2	Schildkröte dreht sich	3.1	Schildkrötengrafik
EDIT	56	Editor aufrufen	3.1	Editor
ERAPEN	38	Schreiber in GD Pauf LÖSCHEN	3.1	Grafik
FIGUR	57	Figur mit XY-Vergrößerung zeichnen	3.1	Figurgrafik
FIGURXY	5	Figur mit X-Y-Vergrößerung zeichnen	6.0	Figurgrafik
FIRTSTIME	36	Neustart Schildkröte	3.1	Schildkrötengrafik
FLINIT	74	FLOPPY-Variablen initialisieren	4.0	FLO-Baugruppe
FLOPPY	75	FLOPPY-Befehl ausführen	4.0	FLO-Baugruppe
FPUWERT	146	FP-Zahl berechnen und FP0 laden	6.1	68020-FPU
GDPVERS	127	GDP oder GDPHS?	6.0	Grafik
GETAD8	109	Wert vom 8-Bit AD-Wandler lesen	4.0	AD/DA-Baugruppen
GETAD10	110	Wert vom 10-Bit AD-Wandler lesen	4.0	AD/DA-Baugruppen
GETAD12	135	AD-WANDLER auf AD/DA-Baugruppe lesen	6.0	AD/DA-Baugruppen
GETBASIS	89	Basis-Adresse des Grundprogramms lesen	4.0	System-Routinen
GETCOLOR	80	Farbcode GDP lesen	4.0	Grafik
GETCURXY	101	Cursorposition lesen	4.0	Zeichenausgabe
GETERR	54	Fehlercode lesen	3.1	Assembler
GETFLOAT	122	FP-Zahl in ASCII in BCD wandeln	5.0	68020-FPU
GETFLOP	76	FLOPPY-Format feststellen	4.0	FLO-Baugruppe
GETK	95	Schildkrötenrichtung lesen	4.0	Schildkrötengrafik
GETLINE	100	Zeilenadresse Bildwiederholsp. lesen	4.0	Zeichenausgabe

Befehls-name	TRAP Nr.	Kurzbeschreibung	Ab Vers.	Befehlsgruppe
GETNEXT	87	Offset für Symboltabelle lesen	4.0	Symboltabelle
GETORG	68	Default-ORG-Adresse lesen	3.1	Assembler
GETRAM	59	Ram-Bereich testen	3.1	System-Routinen
GETSN	98	Seriennummer Grundprogramm lesen	4.0	System-Routinen
GETSTX	66	Textstart-Adresse lesen	3.1	Editor
GETSYM	86	Symboltabellenstart lesen	4.0	Symboltabelle
GETUHR	115	Uhrzeit und Datum lesen	4.0	Echtzeituhren
GETVAR	90	Adressedes Arbeitsspeicher lesen	4.0	System-Routinen
GETVERS	97	Versionsnummer Grundprogramm lesen	4.0	System-Routinen
GETXOR	78	XOR-Modus GDP lesen	4.0	Grafik
GETXY	103	Aktuelle GDP-Position lesen	4.0	Grafik
GRAFIK	126	Großes GRAFIK-Paket für GDP und COL	6.0	Grafik
GRAPOFF	39	Schildkrötengrafik ausschalten	3.1	Schildkrötengrafik
GRUND	124	Grundprogramm als Unterprog. aufrufen	6.0	System-Routinen
HARDCOPY	125	Ansteuerung derHARDCOPY-Baugruppe	6.0	Grafik
HARDDISK	141	Befehl an Harddisk geben und ausführen	6.2	SCSI-Harddisk
HARDTEST	142	Prüft, ob eine Harddisk vorhanden ist	6.2	SCSI-Harddisk
HEBE	3	Schildkrötenspur ausschalten	3.1	Schildkrötengrafik
HIDE	47	Schildkröte ausblenden	3.1	Schildkrötengrafik
KORXY	93	Schildkrötenkoordinaten lesen	4.0	Schildkrötengrafik
LO	22	Zeichen über Centronics ausgeben	3.1	Zeichenausgabe
LST	50	CO2 auf Drucker lenken	3.1	Zeichenausgabe
LSTS	117	Drucker bereit?	4.0	Zeichenausgabe
MOVETO	8	Position GDP setzen	3.1	Grafik
MULS32	72	32-BitMultiplikation	4.0	Berechnungen
NEWPAGE	27	Schreibseite und Leseseite GDP setzen	3.1	Grafik
NIL	52	CO2 ins Leere lenken	3.1	Zeichenausgabe
PO	15	Zeichen über CAS schreiben	3.1.	CAS-Baugruppe
PRINT2X	41	Sedezimale Ausgabe 2 Stellen (1Byte)	3.1	Wertausgabe
PRINT4D	46	Dezimale Ausgabe ohne Vorzeichen	3.1	Wertausgabe
PRINT4X	42	Sedezimale Ausgabe 4 Stellen (2Bytes)	3.1	Wertausgabe
PRINT6X	43	Sedezimale Ausgabe 6 Stellen (3Bytes)	3.1	Wertausgabe
PRINT8B	45	Binäre Ausgabe 8 Bit (1Byte)	3.1	Wertausgabe
PRINT8D	70	Dezimale Ausgabe ohne Vorzeichen	4.0	Wertausgabe
PRINT8X	44	Sedezimale Ausgabe 8 Stellen (4Bytes)	3.1	Wertausgabe
PRINTFP0	121	Wert in FP0 in ASCII wandeln	5.0	68020-FPU
PRINTV8D	71	Dezimale Ausgabe mit Vorzeichen	4.0	Wertausgabe
PROGZGE	64	Frei definierbares Zeichen ausgeben	3.1	Textausgabe
PRTFP0	145	Wert in FP0 in ASCII wandeln	6.1	68020-FPU
PRTSYM	84	Symboltabelle ausgeben	4.0	Symboltabelle
PUTNEXT	88	Offset für Symboltabelle setzen	4.0	Symboltabelle
PUTORG	69	Default-ORG-Adresse setzen	3.1	Assembler
PUTSTX	67	Textstart-Adresse setzen	3.1	Editor
READ	11	Text vom Bildschirm einlesen	3.1	Texteingabe
READAUS	123	Texteingabe mit vorheriger Ausgabe	6.0	Texteingabe
RELAIS	132	Relais auf RELAIS-Baugruppe setzen	6.0	RELAIS-Baugruppe
RELAISIN	133	Port der RELAIS-Baugruppe lesen	6.0	RELAIS-Baugruppe
RELAN	118	Relais auf CAS2 einschalten	4.0	CAS-Baugruppe
RELAUS	119	Relais auf CAS2 ausschalten	4.0	CAS-Baugruppe
RI	14	Zeichen von CAS lesen	3.1	CAS-Baugruppe
RND	96	Zufallsgenerator	4.0	Berechnungen
PO	15	Zeichen über CAS schreiben	3.1	CAS-Baugruppe

Befehls-name	TRAP Nr.	Kurzbeschreibung	Ab Vers.	Befehlsgruppe
SCHR16TEL	19	Schildkröte schreitet	3.1	Schildkrötengrafik
SCHREITE	1	Schildkröte schreitet	3.1	Schildkrötengrafik
SENKE	4	Schildkrötenspur einschalten	3.1	Schildkrötengrafik
SER	128	CI, LO oder FLOPPY auf SER lenken	6.0	SER-Baugruppe
SET	7	Schildkröte absolut setzen	3.1	Schildkrötengrafik
SETA5	91	RAM-Zeiger A5wieder gültig setzen	4.0	System-Routinen
SETCOLOR	79	Farbcode GDP setzen	4.0	Grafik
SETCURXY	102	Cursorposition setzen	4.0	Zeichenausgabe
SETDA	111	DA-Wandler setzen	4.0	AD/DA-Baugruppen
SETDA12	134	DA-Wandler auf AD/DA-Baugruppe setzen	6.0	AD/DA-Baugruppen
SETERR	53	Fehlercode setzen	3.1	Assembler
SETFIG	58	Figur fest setzen (einfrieren)	3.1	Figurgrafik
SETFLIP	34	Umschaltrate für Autoflip setzen	3.1	Grafik
SETFPX	147	FP-Variable X setzen	6.1	68020-FPU
SETFPY	148	FP-Variable Y setzen	6.1	68020-FPU
SETFPZ	149	FP-Variable Z setzen	6.1	68020-FPU
SETPASS	55	Laufnummer setzen	3.1	Zeichenausgabe
SETPEN	37	Schreiber in GDP auf SCHREIBEN	3.1	Grafik
SETUHR	116	Uhrzeit und Datum setzen	4.0	Echtzeituhren
SETXOR	77	XOR-Modus GDP setzen	4.0	Grafik
SHOW	48	Schildkröte einblenden	3.1	Schildkrötengrafik
SI	104	Zeichen von serieller Schnittstelle	4.0	SER-Baugruppe
SI2	138	Zeichen von serieller Schnittstelle	6.1	SER-Baugruppe
SIINIT	108	SER initialisieren (Baudrate,Parität)	4.0	SER-Baugruppe
SIN	23	Sinus*256 berechnen	3.1	Berechnungen
SISTS	106	Zeichen von SER da?	4.0	SER-Baugruppe
SIZE	25	Zeichengröße einstellen (fürCO2..)	3.1	Zeichenausgabe
SO	105	Zeichen an serielle Schnittstelle	4.0	SER-Baugruppe
SOSTS	107	SER bereit zur Ausgabe?	4.0	SER-Baugruppe
SOUND	114	SOUND-Generator setzen	4.0	SpracheundSound
SPEAK	112	Sprachausgabe mit 5 Parametern	4.0	SpracheundSound
SPEAK1	113	Sprachausgabe nur Phonemcode	4.0	SpracheundSound
SUCHBIBO	137	Bibliothekseinträge suchen	6.1	System-Routinen
SYMCLR	85	Symboltabelle löschen	4.0	Symboltabelle
SYNC	28	Vertikal-Synchronimpuls GDP abfragen	3.1	Synchronisation
SYSTEM	139	System-Informationen lesen	6.1	System-Routinen
UHRPRINT	140	Uhrzeit und Datum in ASCII wandeln	6.2	Echtzeituhren
USR	51	CO2 auf Benutzerschnittstelle lenken	3.1	Zeichenausgabe
WAIT	18	Warten, bis GDP fertig	3.1	Grafik
WERT	29	Ganzzahligen Wert berechnen	3.1	Berechnungen
WRITE	10	Text auf dem Bildschirm ausgeben	3.1	Textausgabe
WRITELF	6	Text auf dem Bildschirm ausgeben	6.0	Textausgabe
ZUWEIS	30	Symbol in Symboltabelle eintragen	3.1	Symboltabelle

4.1.4. Liste der Unterprogramme nach Namen - Teil2

Befehls-name	TRAP Nr.	Eingabe-Register	Ausgabe-Register	Zerstörte Register
ADJ360	83	d0.w	d0.w	keine
ASSEMBLE	65	keine	Carry	alle außer a5
ASSERR	120	keine	d0.l	keine
AUFK	94	d0.w	keine	keine
AUFXY	92	d1.w/d2.w	keine	keine
AUTOFLIP	60	keine	keine	d0
CHAR	63	d0.b	keine	d0/a0-a2
CI	12	keine	d0.l	keine
CI2	32	keine	d0.l/Carry	keine
CIINIT2	31	keine	keine	keine
CLPG	17	keine	keine	keine
CLR	16	keine	keine	d0
CLRSCREEN	20	keine	keine	keine
CLUT	131	a0.l	keine	keine
CLUTINIT	130	a0.l	keine	keine
CMD	26	d0.b	keine	keine
CMDPRINT	40	d0.b/d1.w/d2.w/a0.l	a0.l	d0
CO	21	d0.b	keine	d0/a0-a2
CO2	33	d0.b	Carry	keine
CO2SER	129	keine	Carry	keine
COS	24	d0.w	d0.w/Flags	keine
CRLF	99	keine	Carry	d0
CRT	49	keine	keine	keine
CSTS	13	keine	d0.l/Flags(d0.b)	keine
CUROFF	82	keine	keine	keine
CURON	81	keine	keine	keine
CURSAUS	62	keine	keine	keine
CURSEIN	61	keine	keine	keine
DELAY	35	d0.l	keine	d0
DISASS	136	d0.l/a0.l	keine	keine
DIVS32	73	d0.l/d2.l	d0.l/d1.l	keine
DRAWTO	9	d1.w/d2.w	keine	keine
DREHE	2	d0.w	keine	d0
EDIT	56	keine	Carry/(d0.l)	alle außer a5
ERAPEN	38	keine	keine	keine
FIGUR	57	d0.b/d1.w/d2.w/a0.l	d0.w	keine
FIGURXY	5	d0.b/d1.w/d2.w/a0.l	keine	keine
FIRSTTIME	36	keine	keine	keine
FLINIT	74	keine	keine	keine
FLOPPY	75	d1.w/d2.b-d4.b/a0.l	d0.w/Carry	keine
FPUWERT	146	a0.l	d1.l/a0.l/fp0.x/Carry	keine
GDPVERS	127	keine	d0.l	keine
GETAD8	109	d0.b	d0.l	keine
GETAD10	110	keine	d0.l	keine
GETAD12	135	d1.b	d0.w	keine
GETBASIS	89	keine	d0.l/a0.l	keine
GETCOLOR	80	keine	d0.l	keine
GETCURXY	101	keine	d0.l/d1.l	keine
GETERR	54	keine	d0.w	keine
GETFLOAT	122	a0.l/a1.l	a0.l/Carry	keine
GETFLOP	76	d4.b	d0.w/d1.b/d4.b/Carry	keine
GETK	95	keine	d0.w	keine
GETLINE	100	d0.b	a0.l-a2.l	d0

Befehls-name	TRAP Nr.	Eingabe-Register	Ausgabe-Register	Zerstörte Register
GETNEXT	87	keine	d0.l	keine
GETORG	68	keine	d0.l	keine
GETRAM	59	a0.l	d0.l/a0.l/a1.l/Carry	keine
GETSN	98	keine	d0.l	keine
GETSTX	66	keine	d0.l	keine
GETSYM	86	keine	d0.l/a0.l	keine
GETUHR	115	a0.l	a0.l	keine
GETVAR	90	keine	d0.l/a0.l	keine
GETVERS	97	keine	d0.l	keine
GETXOR	78	keine	d0.l	keine
GETXY	103	keine	d1.l/d2.l	keine
GRAFIK	126	Verschiedene Ein- und Ausgaberegister		d7/a6
GRAPOFF	39	keine	keine	keine
GRUND	124	d0.w	keine	keine
HARDCOPY	125	Verschiedene Ein- und Ausgaberegister		d7
HARDDISK	141	d1.b/d4.b/(d2/d3/a0/a1)	d0.l/Carry	keine
HARDTEST	142	d4.b	d0.l/Carry	keine
HEBE	3	keine	keine	keine
HIDE	47	keine	d0.b	keine
KORXY	93	keine	d1.l/d2.l	keine
LO	22	d0.b	keine	keine
LST	50	keine	keine	keine
LSTS	117	keine	d0.l/Flags(d0.b)	keine
MOVETO	8	d1.w/d2.w	keine	keine
MULS32	72	d0.l/d2.l	d0.l/d1.l	keine
NEWPAGE	27	d0.b/d1.b	d0.b	keine
NIL	52	keine	keine	keine
PRINT2X	41	d0.b/a0.l	a0.l	keine
PRINT4D	46	d0.w/a0.l	a0.l	d0
PRINT4X	42	d0.w/a0.l	a0.l	keine
PRINT6X	43	d0.l/a0.l	a0.l	keine
PRINT8B	45	d0.l/a0.l	a0.l	keine
PRINT8D	70	d0.l/a0.l	a0.l	d0
PRINT8X	44	d0.l/a0.l	a0.l	keine
PRINTFP0	121	d0.w/a0.l/fp0.x	a0.l	keine
PRINTV8D	71	d0.l/a0.l	a0.l	d0
PROGZGE	64	a0.l	keine	keine
PRTFP0	156	d0.w/a0.l/fp0.x	a0.l/Carry	keine
PRTSYM	84	keine	keine	keine
PUTNEXT	88	d0.w	keine	keine
PUTORG	69	d0.l	keine	keine
PUTSTX	67	d0.l	a0.l	keine
READ	11	d0.b/d1.w-d3.w/a0.l	d4.l/d5.l/a0.l/Carry	keine
READAUS	123	d0.b/d1.w-d3.w/a0.l	d4.l/d5.l/a0.l/Carry	keine
RELAIS	132	d0.b/a0.l	a0.l	keine
RELAISIN	133	a0.l	a0.l/a0.l	keine
RELAN	118	keine	keine	keine
RELAUS	119	keine	keine	keine
RI	14	keine	d0.l/Carry/Flags(d0.b)	keine
RND	96	d0.w	d0.l	keine
PO	15	d0.b	Carry	keine

Befehls-name	TRAP Nr.	Eingabe-Register	Ausgabe-Register	Zerstörte Register
SCHR16TEL	19	d0.w	keine	d0
SCHREITE	1	d0.w	keine	d0
SENKE	4	keine	keine	keine
SER	128	d0.b	Carry	keine
SET	7	d1.w-d3.w	d0.w	keine
SETA5	91	keine	a5.l	keine
SETCOLOR	79	d0.b	d0.b	keine
SETCURXY	102	d1.b/d2.b	keine	keine
SETDA	111	d1.b/d2.b	keine	keine
SETDA12	134	d0.w/d1.w	keine	keine
SETERR	53	d0.w	keine	keine
SETFIG	58	keine	keine	keine
SETFLIP	34	d0.b/d1.b	keine	keine
SETFPX	147	fp0.x	keine	keine
SETFPY	148	fp0.x	keine	keine
SETFPZ	149	fp0.x	keine	keine
SETPASS	55	d0.w	keine	keine
SETPEN	37	keine	keine	keine
SETUHR	116	a0.l	a0.l	keine
SETXOR	77	d0.w	d0.b	keine
SHOW	48	keine	d0.b	keine
SI	104	keine	d0.l	keine
SI2	138	keine	d0.l	keine
SIINIT	108	d0.b/d1.b	keine	keine
SIN	23	d0.w	d0.w/Flags	keine
SISTS	106	keine	d0.l/Flags	keine
SIZE	25	d0.b	keine	keine
SO	105	d0.b	keine	keine
SOSTS	107	keine	d0.l/Flags	keine
SOUND	114	a0.l	keine	keine
SPEAK	112	a0.l	keine	keine
SPEAK1	113	a0.l	keine	keine
SUCHBIBO	137	d0.w/(d2.l/d3.l/a0.l/a1.l)	Carry/(a1.l/(d1.l-d7.l))	d0/(d1-d2
SYMCLR	85	keine	a0.l	keine
SYNC	28	keine	d0.w/Flags	keine
SYSTEM	139	keine	d0.l	keine
UHRPRINT	140	d0.w/a0.l	a0.l/Carry	keine
USR	51	keine	keine	keine
WAIT	18	keine	keine	keine
WERT	29	a0.l	d0.l/d1.l/a0.l/Carry	d2/d3/a1-a3
WRITE	10	d0.b/d1.w/d2.w/a0.l	keine	keine
WRITELF	6	d0.b/d1.w/d2.w/a0.l	keine	keine
ZUWEIS	30	a0.l	a0.l/(d0.l/d1.l/a3.l)	d2/d3/a1/a2

4.1.5. Liste der Unterprogramme nach Gruppen und TRAP-Nummer

Gruppe1: AD/DA-Baugruppen

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
109	GETAD8	d0.b	d0.l	keine
110	GETAD10	keine	d0.l	keine
111	SETDA	d1.b/d2.b	keine	keine
134	SETDA12	d0.w/d1.w	keine	keine
135	GETAD12	d1.b	d0.w	keine

Gruppe2: Assembler

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
53	SETERR	d0.w	keine	keine
54	GETERR	keine	d0.w	keine
65	ASSEMBLE	keine	Carry	alle außer a5
68	GETORG	keine	d0.l	keine
69	PUTORG	d0.l	keine	keine
120	ASSERR	keine	d0.l	keine

Gruppe3: Berechnungen

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
23	SIN	d0.w	d0.w/Flags	keine
24	COS	d0.w	d0.w/Flags	keine
29	WERT	a0.l	d0.l/d1.l/a0.l/Carry	d2/d3/a1-a3
72	MULS32	d0.l/d2.l	d0.l/d1.l	keine
73	DIVS32	d0.l/d2.l	d0.l/d1.l	keine
83	ADJ360	d0.w	d0.w	keine
96	RND	d0.w	d0.l	keine

Gruppe4: CAS-Baugruppe

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
14	RI	keine	d0.l/Carry/Flags(d0.b)	keine
15	PO	d0.b	Carry	keine
118	RELAN	keine	keine	keine
119	RELAUS	keine	keine	keine

Gruppe5: CLUT-Baugruppe

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
130	CLUTINIT	keine	keine	keine
131	CLUT	a0.l	keine	keine

Gruppe6: DIS-Assembler

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
136	DISASS	d0.l/a0.l	keine	keine

Gruppe7: Echtzeituhren

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
115	GETUHR	a0.l	a0.l	keine
116	SETUHR	a0.l	a0.l	keine
140	UHRPRINT	d0.w/a0.l	a0.l/Carry	keine

Gruppe8: Editor

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
56	EDIT	keine	Carry(d0.l)	alle außer a5
66	GETSTX	keine	d0.l	keine
67	PUTSTX	d0.l	a0.l	keine

Gruppe9: Figurgrafik

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
5	FIGURXY	d0.w-d2.w/a0.l	keine	keine
57	FIGUR	d0.b/d1.w/d2.w/a0.l	d0.w	keine
58	SETFIG	keine	keine	keine

Gruppe10: FLO-Baugruppe

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
74	FLINIT	keine	keine	keine
75	FLOPPY	d1.w/d2.b-d4.b/a0.l	d0.w/Carry	keine
76	GETFLOP	d4.b	d0.w/d1.b/d4.w/Carry	keine

Gruppe11: Grafik

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
8	MOVETO	d1.w/d2.w	keine	keine
9	DRAWTO	d1.w/d2.w	keine	keine
16	CLR	keine	keine	d0
17	CLPG	keine	keine	keine
18	WAIT	keine	keine	keine
26	CMD	d0.b	keine	keine
27	NEWPAGE	d0.b/d1.b	keine	keine
34	SETFLIP	d0.b/d1.b	keine	keine
37	SETPEN	keine	keine	keine
38	ERAPEN	keine	keine	keine
40	CMDPRINT	d0.b/d1.w/d2.w/a0.l	a0.l	d0
60	AUTOFLIP	keine	keine	d0
77	SETXOR	d0.w	d0.b	keine
78	GETXOR	keine	d0.l	keine
79	SETCOLOR	d0.b	d0.b	keine
80	GETCOLOR	keine	d0.l	keine
103	GETXY	keine	d1.l/d2.l	keine
125	HARDCOPY	Verschiedene Ein- und Ausgaberegister		d7
126	GRAFIK	Verschiedene Ein- und Ausgaberegister		d7/a6
127	GDPVERS	keine	d0.l	keine

Gruppe12: RELAIS-Baugruppe

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
132	RELAIS	d0.b/a0.l	a0.l	keine
133	RELAISIN	a0.l	d0.b/a0.l	keine

Gruppe13: Schildkrötengrafik

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
1	SCHREITE	d0.w	keine	d0
2	DREHE	d0.w	keine	d0
3	HEBE	keine	keine	keine
4	SENKE	keine	keine	keine
7	SET	d1.w-d3.w	d0.w	keine
19	SCHR16TEL	d0.w	keine	d0
36	FIRSTTIME	keine	keine	keine
39	GRAPOFF	keine	keine	keine
47	HIDE	keine	d0.b	keine
48	SHOW	keine	d0.b	keine
92	AUFXY	d1.w/d2.w	keine	keine
93	KORXY	keine	d1.l/d2.l	keine
94	AUFK	d0.w	keine	keine
95	GETK	keine		d0.w keine

Gruppe14: SCSI-Harddisk

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
141	HARDDISK	d1.b/d4.b/(d2/d3/a0/a1)	d0.l/Carry	keine
142	HARDTEST	d4.b	d0.l/Carry	keine

Gruppe15: SER-Baugruppe

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
104	SI	keine	d0.l	keine
105	SO	d0.b	keine	keine
106	SISTS	keine	d0.l/Flags	keine
107	SOSTS	keine	d0.l/Flags	keine
108	SIINIT	d0.b/d1.b	keine	keine
128	SER	d0.b	Carry	keine
138	SI2	keine	d0.l	keine

Gruppe16: Sprache und Sound

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
112	SPEAK	a0.l	keine	keine
113	SPEAK1	a0.l	keine	keine
114	SOUND	a0.l	keine	keine

Gruppe17: Symboltabelle

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
30	ZUWEIS	a0.l	a0.l/(d0.l/d1.l/a3.l)	d2/d3/a1/a2
84	PRTSYM	keine	keine	keine
85	SYMCLR	keine	a0.l	keine
86	GETSYM	keine	d0.l/a0.l	keine
87	GETNEXT	keine	d0.l	keine
88	PUTNEXT	d0.w	keine	keine

Gruppe18: Synchronisation

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
28	SYNC	keine	d0.w/Flags	keine
35	DELAY	d0.l	keine	d0

Gruppe19: System-Routinen

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
59	GETRAM	a0.l	d0.l/a0.l/a1.l/Carry	keine
89	GETBASIS	keine	d0.l/a0.l	keine
90	GETVAR	keine	d0.l/a0.l	keine
91	SETA5	keine	a5.l	keine
97	GETVERS	keine	d0.l	keine
98	GETSN	keine	d0.l	keine
124	GRUND	d0.w	keine	keine
137	SUCHBIBO	d0.w/(d2.l/d3.l/a0.l/a1.l)	Carry(a1.l/(d1.l-d7.l))	d0/(d1-d2)
139	SYSTEM	keine	d0.l	keine

Gruppe20: Textausgabe

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
6	WRITELF	d0.b/d1.w/d2.w/a0.l	keine	keine
10	WRITE	d0.b/d1.w/d2.w/a0.l	keine	keine
64	PROGZGE	a0.l	keine	keine

Gruppe21: Texteingabe

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
11	READ	d0.b/d1.w-d3.w/a0.l	d4.l/d5.l/a0.l/Carry	keine
123	READAUS	d0.b/d1.w-d3.w/a0.l	d4.l/d5.l/a0.l/Carry	keine

Gruppe22: Wertausgabe

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
41	PRINT2X	d0.b/a0.l	a0.l	keine
42	PRINT4X	d0.w/a0.l	a0.l	keine
43	PRINT6X	d0.l/a0.l	a0.l	keine
44	PRINT8X	d0.l/a0.l	a0.l	keine
45	PRINT8B	d0.b/a0.l	a0.l	keine
46	PRINT4D	d0.w/a0.l	a0.l	d0
70	PRINT8D	d0.l/a0.l	a0.l	d0
71	PRINTV8D	d0.l/a0.l	a0.l	d0

Gruppe23: Zeichenausgabe

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
20	CLRSCREEN	keine	keine	keine
21	CO	d0.b	keine	d0/a0-a2
22	LO	d0.b	keine	keine
25	SIZE	d0.b	keine	keine
33	CO2	d0.b	Carry	keine
49	CRT	keine	keine	keine
50	LST	keine	keine	keine
51	USR	keine	keine	keine
52	NIL	keine	keine	keine
55	SETPASS	d0.w	keine	keine
61	CURSEIN	keine	keine	keine
62	CURSAUS	keine	keine	keine
63	CHAR	d0.b	keine	d0/a0-a2
81	CURON	keine	keine	keine
82	CUROFF	keine	keine	keine
99	CRLF	keine	Carry	d0
100	GETLINE	d0.b	a0.l-a2.l	d0
101	GETCURXY	keine	d1.l/d2.l	keine
102	SETCURXY	d1.b/d2.b	keine	keine
117	LSTS	keine	d0.l/Flags(d0.b)	keine
129	CO2SER	keine	Carry	keine

Gruppe24: Zeicheneingabe

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
12	CI	keine	d0.l	keine
13	CSTS	keine	d0.l/Flags(d0.b)	keine
31	CIINIT2	keine	keine	keine
32	CI2	keine	d0.l/Carry	keine

Gruppe25: 68020-FPU

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register	Zerstörte Register
121	PRINTFP0	a0.l/fp0.x	d0.l	keine
122	GETFLOAT	a0.l/a1.l	a0.l/Carry	keine
145	PRTFP0	d0.w/a0.l/fp0.x	a0.l/Carry	keine
146	FPUWERT	a0.l	d1.l/a0.l/fp0.x/Carry	keine
147	SETFPX	fp0.x	keine	keine
148	SETFPY	fp0.x	keine	keine
149	SETFPZ	fp0.x	keine	keine

4.2. Beschreibung der Unterprogramme

Alle Unterprogramme, die über TRAP #1 aufgerufen werden können, sind für den Programmierer ein nützliches Mittel, um Programme einfach schreiben zu können. Sie sind dauerhaft im Eprom vorhanden und dadurch immer verfügbar. Auch der Aufruf über die TRAP-Funktion ist sehr schnell, obwohl der Aufruf über JSR @NAME auch möglich ist. Er sollte aber vermieden werden, da bei einer Änderung des Grundprogramms oder einem Grundprogramm mit anderem Prozessor die Adressen verschoben sind. Durch den Aufruf mit TRAP #1 ist eine Kompatibilität zu anderen Grundprogrammen immer gewährleistet.

Die Beschreibung der Unterprogramme erfolgt nach einem einheitlichen Schema. Jedes Unterprogramm hat einen Kopf, der die wichtigsten Informationen über das betreffende Unterprogramm enthält:

TRAP-Nummer : Unter dieser Nummer kann das Programm mit der TRAP-Funktion aufgerufen werden. Diese Nummer wird auch vom Assembler eingesetzt, wenn als Wert !NAME benutzt wird.

Befehlsname : Hier steht der Name des Programms, den der Assembler erkennt. Er kann z.B. beim Aufruf jsr @NAME verwendet werden.

Befehlsgruppe : Hier steht die Gruppe, zu der der Befehl gehört.

Kurzbeschreibung : Hier wird kurz die Funktion erklärt.

Eingaberegister : Hier sind alle Register mit ihrer Funktion aufgeführt, die beim Aufruf mit den entsprechenden Werten zu laden sind. Wenn einige Bits keine Funktion haben, sollten sie immer auf Null gesetzt sein.

Ausgaberegister : Alle Ausgaberegister sind unter diesem Punkt aufgeführt. Zwei zusätzliche Informationen betreffen das Carry-Flag, das z. B. angeben kann, ob ein Programm richtig ausgeführt wurde oder ob ein Fehler auftrat. Mit Hilfe des Carry-Flags können nach dem Ende eines Unterprogramms Sprünge mit den Befehlen BCC und BCS durchgeführt werden, wodurch sich ggf. das Abfragen von Registern erübrigt. Dabei wird nur das Carry-Flag verändert, die anderen Flags bleiben erhalten. Falls nur ein Register ausgegeben wird und die Angabe FLAGS gemacht wird, bedeutet das, daß alle Flags entsprechend dem Wert in diesem Register gesetzt sind, wodurch das Abfragen des Registers direkt nach Programmende wegfallen kann. Die Länge des Registers wird wie im Inhaltsverzeichnis angegeben. Dabei müssen aber nicht alle Bits benutzt werden. Bei der Ausgabe eines Langwortes beispielsweise kann es sein, daß nur ein Byte wirklich ausgegeben wird. Die anderen Bits sind dann immer auf Null gesetzt. Es erfolgt deshalb eine Ausgabe als Langwort, damit der Wert z. B. gleich mit dem DIVS-Befehl dividiert werden kann. Nähere Angaben zu den Registern erfolgen immer unter der Beschreibung.

Zerstörte Register : Hier sind alle Register aufgeführt, die durch das Unterprogramm zerstört werden können. Diese Register sollten also vor dem Programmaufruf gerettet werden, falls sie noch benötigt werden. Hier erfolgt keine Längenangabe. Es muß damit gerechnet werden, daß das gesamte Langwort zerstört werden kann. Die Angaben über die zerstörten Register gelten erst ab Version 6.2. Bei Grundprogrammen früherer Versionen können unter Umständen auch mehr Register zerstört werden.

Ab Version : Ab der hier stehenden Versionsnummer des Grundprogramms steht das Unterprogramm zur Verfügung.

Änderungen : Wenn Änderungen zu vorherigen Grundprogrammversionen erfolgt sind, die die Kompatibilität beeinträchtigen können, sind sie hier aufgeführt. Sind zur Version 6.1 Änderungen vorhanden, gelten diese auch für die Versionen 6.0 und 4.3.

Siehe auch : Hier stehen alle anderen Befehle der Gruppe, zu der der Befehl gehört.

Nach diesen Informationen folgt die Beschreibung der Funktion mit Beispielen und Hinweisen.

Die Beispiele sind sehr einfach gehalten. Für die Unterprogrammaufrufe wird immer die TRAP-Funktion gewählt. Wenn nur ein Byte in ein Register geladen werden muß, wird öfters der Befehl MOVEQ verwendet. Er wird benutzt, da er schneller ist und zeigen soll, wie man ein Programm optimiert. Es werden auch kurze Sprünge (BCC.S) verwendet oder ähnliche Befehle, um das Programm so schnell wie möglich zu machen, obwohl es in den Beispielsfällen eigentlich nicht nötig ist.

In der Zeitschrift LOOP sind bereits viele kleine Programme erschienen, die die Funktionen des Grundprogramms nutzen. Für das Schreiben umfangreicherer Programme sollte man sich diese Listings genauer ansehen, um daraus zu lernen. Außerdem ist das Listing des Grundprogramms auf Diskette erhältlich, sodaß man alle Einzelheiten zu den einzelnen Unterprogrammen nachlesen kann. Da das Listing auch noch kommentiert ist, kann man die Unterprogramme leicht nachvollziehen. Schließlich gibt es inzwischen zu den 68000-Prozessoren viele Bücher, die die Programmiersprache Assembler leicht verständlich erklären. Die Kenntnis der Programmiersprache Assembler ist im übrigen Vorraussetzung für den Umgang mit den Unterprogrammen.

TRAP-Nummer	: **1**		
Befehlsname	: **SCHREITE**		
Befehlsgruppe	: Schildkrötengrafik.		
Kurzbeschreibung	: Schildkröte schreitet vorwärts oder rückwärts.		
Eingaberegister	: D0.W = Anzahl der Schritte und Richtung		
Ausgaberegister	: Keine.		
Zerstörte Register	: D0		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: DREHE (2)	HEBE (3)	SENKE (4)
	SET (7)	SCHR16TEL (19)	FIRSTTIME (36)
	GRAPOFF (39)	HIDE (47)	SHOW (48)
	AUFXY (92)	KORXY (93)	AUFK (94)
	GETK (95)		

Der Befehl SCHREITE bewegt die Schildkröte auf dem Bildschirm in die Richtung, in die sie blickt bzw. in die entgegengesetzte Richtung. Das Register D0.W gibt dabei die Anzahl der Schritte und die Richtung an. Ein positiver Wert läßt die Schildkröte vorwärts schreiten, während ein negativer Wert sie rückwärts bewegt. Ein Schritt entspricht in X-Richtung einem Bildpunkt und in Y-Richtung einem halben Bildpunkt. Damit wird erreicht, daß sich symmetrische Figuren auf dem Bildschirm ergeben.

Einfache Beispiele:

Die Schildkröte bewegt sich 10 Schritte nach vorne.

```
START:
        MOVEQ     #10,D0          * Zehn Schritte vorwärts gehen.
        MOVEQ     #!SCHREITE,D7
        TRAP      #1
        RTS
```

Die Schildkröte wandert 15 Schritte rückwärts.

```
START:
        MOVEQ     #-15,D0         * Fünfzehn Schritte rückwärts gehen.
        MOVEQ     #!SCHREITE,D7
        TRAP      #1
        RTS
```

Komplexeres Beispiel:

Es wird ein Quadrat mit der Kantenlänge 50 gezeichnet.

```
START:
        MOVEQ     #4-1,D1         * 4 Durchgänge.
SCHLEIFE:
        MOVEQ     #50,D0
        MOVEQ     #!SCHREITE,D7   * 50 Schritte schreiten.
        TRAP      #1
        MOVEQ     #90,D0
        MOVEQ     #!DREHE,D7      * Um 90 Grad drehen.
        TRAP      #1
        DBRA      D1,SCHLEIFE     * Wiederholen.
        RTS
```

Bemerkung:

Wenn die angegebene Zahl der Schritte zu groß ist, verschwindet die Schildkröte aus dem sichtbaren Bildschirmbereich. Der Wert sollte aber 4096 nicht überschreiten, sonst erscheint die Schildkröte wieder auf dem Bildschirm.

Bei allen Befehlen der Schildkrötengrafik werden automatisch zwei Bildseiten verwendet. Diese beiden Bildseiten werden alle 20ms umgeschaltet. Auf der Bildseite 0 wird die Grafik dargestellt und auf der Bildseite 1 die Schildkröte, die im folgenden auch Zeiger genannt wird. Der Zeiger besteht aus einem Dreieck, dessen Spitze in die aktuelle Blickrichtung der Schildkröte zeigt. Der Zeiger kann die Richtung in 45 Grad-Schritten anzeigen. Die Schildkröte kann aber intern auf ein Grad genau ausgerichtet werden. Will man den Umschaltvorgang der Bildseiten nicht haben, so muß man ihn mit dafür bestimmten Befehlen abschalten.

TRAP-Nummer	: **2**		
Befehlsname	: **DREHE**		
Befehlsgruppe	: Schildkrötengrafik.		
Kurzbeschreibung	: Schildkröte dreht sich im oder gegen den Uhrzeigersinn.		
Eingaberegister	: D0.W = Relativer Drehwinkel und Drehrichtung.		
Ausgaberegister	: Keine.		
Zerstörte Register	: D0		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SCHREITE (1)	HEBE (3)	SENKE (4)
	SET (7)	SCHR16TEL (19)	FIRSTTIME (36)
	GRAPOFF (39)	HIDE (47)	SHOW (48)
	AUFXY (92)	KORXY (93)	AUFK (94)
	GETK (95)		

Die Schildkröte wird entsprechend dem Wert in D0.W gedreht. Eine positive Angabe dreht die Schildkröte entgegen dem Uhrzeigersinn (mathematisch positiv), während ein negativer Wert sie im Uhrzeigersinn dreht. Der Winkel, um den sie gedreht werden soll, wird dabei in Grad angegeben.

Einfache Beispiele:

Die Schildkröte dreht sich um 45 Grad entgegen dem Uhrzeigersinn.

```
START:
        MOVEQ     #45,D0          * Um 45 Grad entgegen dem
        MOVEQ     #!DREHE,D7      * Uhrzeigersinn drehen.
        TRAP      #1
        RTS
```

Die Schildkröte dreht sich um 30 Grad im Uhrzeigersinn.

```
START:
        MOVEQ     #-30,D0         * Um 30 Grad im Uhrzeigersinn
        MOVEQ     #!DREHE,D7      * drehen.
        TRAP      #1
        RTS
```

Komplexeres Beispiel:

Es wird ein Blumenmuster gezeichnet.

```
START:
        MOVEQ     #!HIDE,D7       * Schildkröte unsichtbar.
        TRAP      #1
        MOVEQ     #10-1,D5        * 10 Durchgänge.
SCHLEIF0:
        MOVEQ     #10-1,D4        * 10 Durchgänge.
SCHLEIF1:
        MOVEQ     #5-1,D3         * 5 Durchgänge.
SCHLEIF2:
        MOVEQ     #2-1,D2         * 2 Durchgänge.
SCHLEIF3:
        MOVEQ     #45-1,D1        * 45 Durchgänge.
SCHLEIF4:
        MOVEQ     #2,D0
        MOVEQ     #!DREHE,D7      * 2 Grad drehen.
        TRAP      #1
        MOVEQ     #10,D0          * 10 kleine Schritte schreiten.
        MOVEQ     #!SCHR16TEL,D7
        TRAP      #1
        DBRA      D1,SCHLEIF4     * Wiederholen.
```

```
MOVEQ     #90,D0
MOVEQ     #!DREHE,D7        * 90 Grad drehen.
TRAP      #1
DBRA      D2,SCHLEIF3       * Blatt fertig.
MOVEQ     #360/5,D0
MOVEQ     #!DREHE,D7        * Ausgleichsdrehung.
TRAP      #1
DBRA      D3,SCHLEIF2       * Blüte fertig.
MOVEQ     #-80,D0
MOVEQ     #!SCHREITE,D7     * Ausgleich.
TRAP      #1
MOVEQ     #36,D0
MOVEQ     #!DREHE,D7        * 36 Grad drehen.
TRAP      #1
DBRA      D4,SCHLEIF1       * Blume fertig.
MOVEQ     #36,D0
MOVEQ     #!DREHE,D7        * Um 36 Grad drehen.
TRAP      #1
DBRA      D5,SCHLEIF0       * Blumenkranz fertig.
MOVEQ     #!GRAPOFF,D7      * Schildkröte abschalten.
TRAP      #1
RTS
```

Bemerkung:

Ein Winkel, der größer als 359 Grad oder kleiner als 0 Grad ist, wird im Programm DREHE immer automatisch auf den Bereich 0 bis 359 Grad umgerechnet und dann der Zeiger danach ausgerichtet. Wichtig ist noch, daß sich der angegebene Winkel immer auf die aktuelle Stellung der Schildkröte bezieht, nicht dagegen auf den Bildschirmrand.

TRAP-Nummer	: **3**		
Befehlsname	: **HEBE**		
Befehlsgruppe	: Schildkrötengrafik.		
Kurzbeschreibung	: Die Schildkröte hinterläßt ab jetzt keine Spur mehr.		
Eingaberegister	: Keine.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SCHREITE (1)	DREHE (2)	SENKE (4)
	SET (7)	SCHR16TEL (19)	FIRSTTIME (36)
	GRAPOFF (39)	HIDE (47)	SHOW (48)
	AUFXY (92)	KORXY (93)	AUFK (94)
	GETK (95)		

Die Schildkröte hinterläßt ab sofort keine Schreibspur mehr bei ihren Bewegungen. Damit kann man Figuren zeichnen, die keine Verbindung mit einander haben. Der Befehl gilt nur für die Schildkröte, nicht für Grafik allgemein.

Zu dem Befehl gehört auch das Gegenstück SENKE.

Einfaches Beispiel:

Nach Ausführung diese Programmteils hinterläßt die Schildkröte keine Schreibspur mehr.

```
START:
        MOVEQ     #!HEBE,D7          * Keine Schreibspur mehr.
        TRAP      #1
        RTS
```

Komplexeres Beispiel:

Es wird eine gestrichelte Linie gezeichnet.

```
START:
        MOVEQ     #10-1,D1
SCHLEIFE:
        MOVEQ     #5,D0              * 5 Schritte schreiten.
        MOVEQ     #!SCHREITE,D7
        TRAP      #1
        MOVEQ     #!HEBE,D7          * Dann Stift anheben.
        TRAP      #1
        MOVEQ     #5,D0              * Wieder 5 Schritte gehen.
        MOVEQ     #!SCHREITE,D7
        TRAP      #1
        MOVEQ     #!SENKE,D7         * Stift absenken.
        TRAP      #1
        DBRA      D1,SCHLEIFE
        RTS
```

TRAP-Nummer	: **4**
Befehlsname	: **SENKE**
Befehlsgruppe	: Schildkrötengrafik.
Kurzbeschreibung	: Die Schildkröte hinterläßt wieder eine Spur.
Eingaberegister	: Keine.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 3.1
Änderungen zu 4.3	: Nein.
Änderungen zu 6.1	: Nein.
Siehe auch	: SCHREITE (1), DREHE (2), HEBE (3), SET (7), SCHR16TEL (10), FIRSTTIME (36), GRAPOFF (39), HIDE (47), SHOW (48), AUFXY (92), KORXY (93), AUFK (94), GETK (95)

Der Schreibstift wird gesenkt. Von da an wird eine Schreibspur hinterlassen, wenn sich die Schildkröte bewegt. Auch dieser Befehl wirkt nur auf die Schildkrötengrafik.

Die Schildkröte hinterläßt nach dem ersten Aufruf eines Schildkrötenbefehls immer eine Spur, so daß der Befehl normalerweise nur nach dem Befehl HEBE benutzt werden muß.

Einfaches Beispiel:

Nach dem Programm hinterläßt die Schildkröte wieder eine Spur.

```
START:
        MOVEQ     #!SENKE,D7        * Schreibspur wieder vorhanden.
        TRAP      #1
        RTS
```

Komplexeres Beispiel:

Ein perforierter Kreis wird gezeichnet.

```
START:
        MOVEQ     #360/3,D1         * Einen Kreis zeichnen.
SCHLEIFE:
        MOVEQ     #1,D0             * 1 Schritt schreiten.
        MOVEQ     #!SCHREITE,D7
        TRAP      #1
        MOVEQ     #!HEBE,D7         * Dann Stift anheben.
        TRAP      #1
        MOVEQ     #1,D0             * 1 Grad drehen.
        MOVEQ     #!DREHE,D7
        TRAP      #1
        MOVEQ     #4,D0             * 4 Schritte gehen.
        MOVEQ     #!SCHREITE,D7
        TRAP      #1
        MOVEQ     #!SENKE,D7        * Stift absenken.
        TRAP      #1
        MOVEQ     #2,D0             * 2 Grad drehen.
        MOVEQ     #!DREHE,D7
        TRAP      #1
        DBRA      D1,SCHLEIFE
        RTS
```

TRAP-Nummer	: **5**
Befehlsname	: **FIGURXY**
Befehlsgruppe	: Figurgrafik.
Kurzbeschreibung	: Figur mit getrennter X- und Y-Vergrößerung zeichnen.
Eingaberegister	: D0.W = Größe der Figur. D1.W = X-Koordinate. D2.W = Y-Koordinate. A0.L = Adresse der Figur-Daten.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 6.0
Änderungen zu 4.3	: ——
Änderungen zu 6.1	: Nein.
Siehe auch	: FIGUR (57) SETFIG (58)

Dieser Befehl entspricht in seiner Funktion dem Befehl FIGUR. Er hat aber den Vorteil, daß die X- und die Y-Vergrößerung getrennt eingestellt werden können. Dabei wird die Vergrößerung im Register D0.W angegeben. Die Größe errechnet sich nach der Formel:

G = DX * 256 + DY.

DX ist die X-Vergrößerung, die einen Bereich von 0 bis 255 hat. DY ist die Y-Vergrößerung mit dem selben Bereich.

Beispiel:

DX = 5
DY = 9

=> G = 5*256 + 9 = 1289 = $0509

Aufgrund der hexadezimalen Darstellung kann man die Vergrößerung sofort erkennen.

Da die Funktion des Befehls ansonsten mit der des FIGUR-Befehls gleich ist, wird auf die Erläuterungen zu dem Befehl FIGUR (57) verwiesen.

Einfaches Beispiel:

Ein Rechteck wird in X-Richtung nicht vergrößert, sondern nur in Y-Richtung. Dabei wird immer automatisch die alte Figur gelöscht.

```
START:
        MOVE    #$0120,D0       * Keine X-Vergrößerung, nur für DY.
        MOVEQ   #100,D1         * X-Position.
        MOVEQ   #100,D2         * Y-Position.
        LEA     FIGUR(PC),A0    * Adresse der Figur-Daten.
        MOVEQ   #!FIGURXY,D7    * Figur zeichnen.
        TRAP    #1
        RTS
FIGUR:
        DC.B    0,0,0,0,0,2,4,4,4,4,4,6,10   * Rechteck, vgl. FIGUR (57).
```

Komplexeres Beispiel:

Ein Rechteck wird erst in X- und dann in Y-Richtung gedehnt. Dies ist mit dem FIGUR-Befehl nicht möglich. Auch hier wird zuerst immer die alte Figur gelöscht.

```
START:
        MOVE        #$0101,D3       * Größe der Figur.
        MOVEQ       #2,D1           * X-Position.
        MOVEQ       #2,D2           * Y-Position.
        LEA         FIGUR(PC),A0    * Adresse der Figurdaten.
        MOVE        #250-1,D4       * 250 mal.
SCHLEIF0:
        MOVEQ       #!SYNC,D7       * 20 ms warten.
        TRAP        #1
        BEQ.S       SCHLEIF0
        MOVE        D3,D0           * Größe nach D0.
        MOVEQ       #!FIGURXY,D7    * Figur zeichnen.
        TRAP        #1
        ADD         #$0100,D3       * X-Vergrößerung ändern.
        DBRA        D4,SCHLEIF0     * Wiederholen.
        MOVE        #250-1,D4       * 250 mal.
SCHLEIF1:
        MOVEQ       #!SYNC,D7       * 20 ms warten.
        TRAP        #1
        BEQ.S       SCHLEIF1
        MOVE        D3,D0           * Größe nach D0.
        MOVEQ       #!FIGURXY,D7    * Figur zeichnen.
        TRAP        #1
        ADDQ        #1,D3           * Y-Vergrößerung ändern.
        DBRA        D4,SCHLEIF1     * Wiederholen.
        RTS

FIGUR:
        DC.B        0,0,2,4,4,6,10  * Rechteck, vgl. FIGUR (57).
```

TRAP-Nummer	: **6**
Befehlsname	: **WRITELF**
Befehlsgruppe	: Textausgabe.
Kurzbeschreibung	: Text mit wählbarer Größe wird auf dem Screen ausgeben.
Eingaberegister	: D0.B = Schriftgröße. D1.W = X-Koordinate. D2.W = Y-Koordinate. A0.L = Adresse des Textes.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 6.0
Änderungen zu 4.3	: ——
Änderungen zu 6.1	: Nein.
Siehe auch	: WRITE (10) PROGZGE (64)

Die Funktion ist fast identisch mit dem Befehl WRITE, bis auf zwei Ausnahmen:

1.) Der Text wird nicht vorgelöscht, weshalb man mit diesem Befehl nur auf leere Teile des Bildschirms schreiben sollte.

2.) Das ASCII-Zeichen LF (als Code $0a oder 10) bewirkt einen Zeilenvorschub. Die neue Zeile fängt an der gleichen X-Position, aber eine Zeile tiefer an. Dadurch können längere Texte leicht ausgegeben werden, ohne daß man den WRITE-Befehl mehrmals hintereinander aufrufen muß.

Einfaches Beispiel:

Ein Text wird über mehrere Zeilen ausgegeben.

```
START:
        MOVEQ     #$22,D0              * Schriftgröße.
        MOVEQ     #10,D1               * X-Position.
        MOVE      #200,D2              * Y-Position.
        LEA       TEXT(PC),A0          * Adresse des Textes.
        MOVEQ     #!WRITELF,D7         * Befehl ausführen.
        TRAP      #1
        RTS
TEXT:
        DC.B      'Dieser Text wird in',10
        DC.B      'mehreren Zeilen ausgegeben,',10
        DC.B      'um die Wirkung des Befehls',10
        DC.B      'WRITELF zu demonstrieren.',0
```

Für weitere Informationen vgl. die Beschreibung des WRITE-Befehls (10).

TRAP-Nummer	:	7
Befehlsname	:	SET
Befehlsgruppe	:	Schildkrötengrafik.
Kurzbeschreibung	:	Schildkröte auf eine absolute Position setzen.
Eingaberegister	:	D1.W = X-Koordinate. D2.W = Y-Koordinate. D3.W = Absolute Blickrichtung.
Ausgaberegister	:	D0.W = Blickrichtung (360-Grad-Bereich).
Zerstörte Register	:	Keine.
Ab Version	:	3.1
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	SCHREITE (1), DREHE (2), HEBE (3), SENKE (4), SCHR16TEL (19), FIRSTTIME (36), GRAPOFF (39), HIDE (47), SHOW (48), AUFXY (92), KORXY (93), AUFK (94), GETK (95)

Damit kann die Schildkröte auf eine absolute Position gesetzt werden. Sie hinterläßt dabei keine Schreibspur.
Register D1.W erhält die X-Koordinate (Bereich 0 bis 511).
Register D2.W erhält die Y-Koordinate (Bereich 0 bis 511).
Register D3.W erhält die Blickrichtung in Grad,
wobei die Schildkröte bei 0 Grad nach rechts und bei 90 Grad nach oben zeigt. Die Blickrichtung ist hier also absolut anzugeben.
Als Rückgabe erhält man den absoluten Blickwinkel der Schildkröte zurück. Dieser hat einen Bereich von 0 bis 359, siehe SET (7).

Einfaches Beispiel:

Die Schildkröte wird in die Mitte des Bildschirms gesetzt.

```
START:
        MOVE    #256,D1         * X = 256.
        MOVE    #256,D2         * Y = 256.
        MOVEQ   #90,D3          * Blick nach oben.
        MOVEQ   #!SET,D7        * Befehl ausführen.
        TRAP    #1
        RTS
```

Komplexeres Beispiel:

Die Schildkröte wird entlang einer Kreisbahn positioniert. Der Blickwinkel der Schildkröte ist immer nach außen gerichtet. Dann wird noch ein kurzer Strich gezeichnet, so daß ein Strahlenring entsteht.

```
START:
        MOVE    #360-1,D4       * 360 Grad ist ein Kreis.
SCHLEIFE:
        MOVE    D4,D0
        MOVEQ   #!SIN,D7        * Sinus ist X-Koordinate.
        TRAP    #1
        MOVE    D0,D1
        ASR     #1,D1           * Durch 2, damit der Kreis nicht so groß wird.
        ADD     #256,D1         * In Bildschirmmitte.
        MOVE    D4,D0
        MOVEQ   #!COS,D7        * Cosinus ist Y-Koordinate.
        TRAP    #1
        MOVE    D0,D2
        ASR     #1,D2           * Damit der Kreis nicht so groß wird.
        ADD     #256,D2         * In Bildschirmmitte.
        MOVEQ   #90,D3
        SUB     D4,D3           * Blickwinkel immer nach außen.
        MOVEQ   #!SET,D7        * Schildkröte setzen.
        TRAP    #1
        MOVEQ   #10,D0
        MOVEQ   #!SCHREITE,D7   * Linie ziehen.
        TRAP    #1
        MOVEQ   #1,D0
        MOVEQ   #!DELAY,D7
        TRAP    #1              * Verzögerung 1/10 Sekunde.
        DBRA    D4,SCHLEIFE
        RTS
```

TRAP-Nummer	: **8**
Befehlsname	: **MOVETO**
Befehlsgruppe	: Grafik.
Kurzbeschreibung	: Position GDP setzen.
Eingaberegister	: D1.W = X-Koordinate. D2.W = Y-Koordinate.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 3.1
Änderungen zu 4.3	: Nein.
Änderungen zu 6.1	: Nein.
Siehe auch	: DRAWTO (9), CLR (16), CLPG (17), WAIT (18), CMD (26), NEWPAGE (27), SETFLIP (34), SETPEN (37), ERAPEN (38), CMDPRINT (40), AUTOFLIP (60), SETXOR (77), GETXOR (78), SETCOLOR (79), GETCOLOR (80), GETXY (103), HARDCOPY (125), GRAFIK (126), GDPVERS (127)

Damit wird die nächste Schreib- oder Zeichenstelle für Graphik oder Text vorbereitet. Außerdem werden die Register des Graphik-Prozessors eingestellt.

Als Parameter steht in Register D1.W die X-Koordinate. Die Schreibstelle ist auf dem Bildschirm sichtbar, wenn die Koordinate im Bereich 0 bis 511 liegt, sonst wird im unsichtbaren Bereich gezeichnet. Insgesamt können 4096 Bildpunkte auf dem Monitor erreicht werden. Demgemäß entpricht die Koordinate X = 4096 der Koordinate 0 und die Koordinate X = 4097 dem Wert 1.

Im Register D2.W steht die Y-Koordinate. Der hier sichtbare Bereich liegt zwischen den Y-Koordinaten 0 und 255. Es gibt wieder 4096 verschiedene Koordinaten. Negative Werte sind zulässig, aber ebenfalls nicht im sichtbaren Bereich.

Einfaches Beispiel:

Es wird ein Punkt an der Stelle X = 200 und Y = 100 gezeichnet.

```
START:
        MOVE    #200,D1         * X = 200.
        MOVEQ   #100,D2         * Y = 100.
        MOVEQ   #!MOVETO,D7     * Positionieren.
        TRAP    #1
        MOVEQ   #$80,D0         * Befehl für Punkt setzen.
        MOVEQ   #!CMD,D7        * Befehl an GDP.
        TRAP    #1
        RTS
```

Komplexeres Beispiel:

Der Bildschirm wird von oben nach unten mit waagerechten Strichen gefüllt.

```
START:
        MOVEQ   #!SETPEN,D7     * Auf jeden Fall zeichnen.
        TRAP    #1
        MOVE    #256-1,D2       * Bildschirmhöhe ist 256.
SCHLEIFE:
        CLR     D1              * X = 0.
        MOVEQ   #!MOVETO,D7     * Positionieren.
        TRAP    #1
        MOVE    #511,D1         * X = 511.
        MOVEQ   #!DRAWTO,D7     * Linie zeichnen.
        TRAP    #1
        MOVEQ   #1,D0           * 1/10 Sekunde warten.
        MOVEQ   #!DELAY,D7
        TRAP    #1
        DBRA    D2,SCHLEIFE     * Nächste Linie.
        RTS
```

TRAP-Nummer	:	9
Befehlsname	:	**DRAWTO**
Befehlsgruppe	:	Grafik.
Kurzbeschreibung	:	Linie zur neuen Position zeichnen.
Eingaberegister	:	D1.W = X-Koordinate. D2.W = Y-Koordinate.
Ausgaberegister	:	Keine.
Zerstörte Register	:	Keine.
Ab Version	:	3.1
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	MOVETO (8), CLR (16), CLPG (17), WAIT (18), CMD (26), NEWPAGE (27), SETFLIP (34), SETPEN (37), ERAPEN (38), CMDPRINT (40), AUTOFLIP (60), SETXOR (77), GETXOR (78), SETCOLOR (79), GETCOLOR (80), GETXY (103), HARDCOPY (125), GRAFIK (126), GDPVERS (127)

Der Graphik-Prozessor zeichnet eine Linie von der zuletzt eingestellten Position zur neuen Position, die mit dem DRAWTO-Befehl angegeben wird.
Register D1.W enthält die X-Koordinate, Register D2.W erhält die Y-Koordinate. Für die Bereiche gilt das gleiche wie beim MOVETO-Befehl (X von 0 bis 511 und Y von 0 bis 255).

Einfaches Beispiel:

Es wird eine Linie in der Bildschirmdiagonalen gezeichnet.

```
START:
        CLR     D1          * X = 0.
        CLR     D2          * Y = 0.
        MOVEQ   #!MOVETO,D7 * Startpunkt festlegen.
        TRAP    #1
        MOVE    #511,D1     * Bis X = 511
        MOVE    #255,D2     * und bis Y = 255
        MOVEQ   #!DRAWTO,D7 * ergibt Diagonale.
        TRAP    #1
        RTS
```

Komplexeres Beispiel:

Es wird ein großer Kreis gezeichnet, aber nicht mit der Schildkrötengrafik.

```
START:
        MOVE    #256,D1     * X = 256.
        MOVE    #255,D2     * Y = 255.
        MOVEQ   #!MOVETO,D7 * Zuerst Anfangsposition festlegen.
        TRAP    #1
        MOVE    #361-1,D3   * 360 Grad ist ein Kreis.
SCHLEIFE:
        MOVE    D3,D0
        MOVEQ   #!SIN,D7    * Sinus ist X-Koordinate.
        TRAP    #1
        MULS    #250,D0     * Radius ist 250.
        DIVS    #256,D0     * Durch 256 teilen, da SIN(x)*256 berechnet
        MOVE    D0,D1       * wurde.
        ADD     #256,D1     * In Bildschirmmitte.
        MOVE    D3,D0
        MOVEQ   #!COS,D7    * Cosinus ist Y-Koordinate.
        TRAP    #1
        MULS    #250,D0     * Radius.
        DIVS    #256,D0
        MOVE    D0,D2
        ASR     #1,D2       * Für Symmetrie.
        ADD     #128,D2     * In Bildschirmmitte.
        MOVEQ   #!DRAWTO,D7 * Linie ziehen.
        TRAP    #1
        DBRA    D3,SCHLEIFE
        RTS
```

Bemerkung:

Der Wiedereintritt in den Bildbereich wird vom Graphik-Prozessor korrekt ausgeführt, wodurch man Ausschnitte aus großen Zeichnungen durch Verschieben der Anfangskoordinaten erzeugen kann. Wenn man den Mittelpunkt des Kreises verschiebt, kann dies leicht ausprobiert werden, da dann ein Teil des Kreises verschwindet.

Will man eine Linie löschen, kann man mit dem Befehl ERAPEN die Schreibart LÖSCHEN vorwählen. Mit SETPEN wird der Grafik-Prozessor wieder auf SCHREIBEN gesetzt.

TRAP-Nummer	:	**10**
Befehlsname	:	**WRITE**
Befehlsgruppe	:	Textausgabe.
Kurzbeschreibung	:	Text in wählbarer Größe auf Screen ausgeben.
Eingaberegister	:	D0.B = Schriftgröße. D1.W = X-Koordinate. D2.W = Y-Koordinate. A0.L = Textadresse.
Ausgaberegister	:	Keine.
Zerstörte Register	:	Keine.
Ab Version	:	3.1
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	WRITELF (6) PROGZGE (64)

Ein Befehl, mit dem man Texte auf dem Bildschirm ausgeben kann. Dazu wird im
Register D0.B die Textgröße festgelegt,
Register D1.W die X-Koordinate (0 bis 511) und
Register D2.W die Y-Koordinate (0 bis 255).
In das Adreßregister A0.L wird die Startadresse des Textes geladen. Der Text besteht aus ASCII-Zeichen. Das Ende des Textes wird durch eine 0 (Code $00) angegeben. Der Hintergrund wird vorgelöscht, wodurch alte Texte automatisch überschrieben werden.

Textgröße festlegen:

Man kann die Größe eines Schriftzeichens in X- und Y-Richtung getrennt angeben. Dazu muß man beide Werte mit einer kleinen Formel zusammenbauen.

G := DX * 16 + DY.

DX ist dabei die Vergrößerung in X-Richtung (Bereich 1 bis 16), wobei die Vergrößerung 1 einer Breite von 6 Punkten pro Buchstaben entspricht.

Für DY gelten die gleichen Werte, allerdings entspricht die Vergrößerung 1 einer Höhe von 10 Punkten (einschließlich Abstand zur nächsten Zeile).

Wenn DX oder DY den Maximalwert von 16 annehmen sollen, muß man anstelle der Zahl 16 den Wert 0 einsetzen. Das ist ein Sonderfall und vom Hersteller des Graphikprozessors EF 9366 vorgegeben.

Beispiel:

DX = 1
DY = 1

=> G = 1 * 16 + 1.

Also ist g = 17. Man kann den Wert auch sedezimal darstellen, es ergibt sich g = $11. An diesem Wert kann man den Zusammenbau aus den beiden Werten noch besser erkennen.

Beispiel:

DX = 16
DY = 16

=> G = 0.

G nimmt dann den Wert 0 ($00) an (Sonderfall).

Textbuffer:

Der abzulegende Text kann im Assembler einfach mit

```
DC.B    '....'
```

angegeben werden.

Die Ablage des Endes erreicht man mit

```
DC.B    0.
```

Die Ausgabe von mehreren Textzeilen kann man durch eine entsprechende Anzahl von WRITE-Befehlen erreichen, deren Y-Koordinaten sich unterscheiden. Eine andere Möglichkeit, mehrere Textzeilen auszugeben, die mit ihrem ersten Wort jeweils genau untereinander stehen sollen, stellt der Befehl WRITELF (6) zur Verfügung, der für die Ausgabe mehrzeiliger Texte einfacher und in der Ausführung schneller ist.

Einfaches Beispiel:

Der Text "Hallo" wird ausgegeben.

```
START:
        MOVEQ     #$32,D0          * Breitschrift DX = 3, DY = 2.
        MOVEQ     #120,D1          * X = 120.
        MOVEQ     #123,D2          * Y = 123 (Mitte).
        LEA       TEXT(PC),A0      * Adresse des Textes.
        MOVEQ     #!WRITE,D7       * Text ausgeben.
        TRAP      #1
        RTS                        * Ende des Programms.
TEXT:
        DC.B      'Hallo',0        * Text: Hallo.
```

Bemerkung:

Die 0 beim DC.B-Befehl, die hinter dem Komma steht, darf nicht vergessen werden. Sonst weiß das Grundprogramm nicht, wann der Text zu Ende ist. Die merkwürdigsten Ausgaben sind dann die Folge.

Man kann den Bufferinhalt aber auch wie folgt definieren:

```
DC.B    'Hallo'
DC.B    0
```

Hier wird die 0 mit einem getrennten DC.B-Befehl abgelegt.

Wenn hinter dem Text weitere Programmteile folgen, kann beim Assemblieren die Fehlermeldung "Nicht auf Wortgrenze" erscheinen. Dann muß man den Befehl

```
DS      0
```

vor den Befehl schreiben, der dem Befehl DC.B folgt. Damit wird automatisch ein leeres Byte nach dem DC.B-Befehl eingefügt, wenn die Adresse nach dem DC.B-Befehl ungerade ist. Das ist ggf. deshalb nötig, weil 680xx-Befehle immer nur auf geraden Adressen beginnen dürfen.

Die Anweisungen für die Textablage und Endekennungen gelten auch für andere Unterprogramme. Eine ausführliche Beschreibung der Befehle DC und DS steht in Kapitel 3.2.2. im Zusammenhang mit Erläuterungen zum Assembler des Grundprogramms.

TRAP-Nummer	:	**11**
Befehlsname	:	**READ**
Befehlsgruppe	:	Texteingabe.
Kurzbeschreibung	:	Liest einen Text vom Bildschirm ein.
Eingaberegister	:	D0.B = Schriftgröße. D1.W = X-Koordinate. D2.W = Y-Koordinate. D3.W = Maximale Anzahl Zeichen. A0.L = Textbuffer.
Ausgaberegister	:	D4.L = Zahl der eingegebenen Zeichen. D5.L = Letztes eingegebenes Zeichen. A0.L = Adresse der Endekennung. Carry = 1, wenn mit ESC beendet, sonst Null.
Zerstörte Register	:	Keine.
Ab Version	:	3.1
Änderungen zu 4.3	:	Die Eingabe kann nur durch ein CR (CARRIAGE RETURN) oder durch vollständiges Beschreiben des Fensters beendet werden. Jedes Controlzeichen, das nicht zur Cursorsteuerung dient, wird ignoriert. Es sind D4.L und D5.L gültig.
Änderungen zu 6.1	:	Die Eingabe von CR führt nicht mehr dazu, daß Text hinter der Cursorposition abgeschnitten wird. Der hinter dem Cursor stehende Text bleibt unverändert. Um Text hinter dem Cursor zu entfernen, kann CTRL-T verwendet werden. Endeleerzeichen werden bis zur Cursorposition abgeschnitten. Mit ESC kann die Bearbeitung von Texten im Eingabefeld ebenfalls beendet werden.
Siehe auch	:	READAUS (123)

Einlesen eines Textes von der Tastatur. Damit kann man Eingabefelder erzeugen, wie sie auch im Grundprogramm, z.B. zur Eingabe von START, ADRESSE usw. verwendet werden.

Das Register D0.B erhält die Textgröße wie beim WRITE-Befehl.
Das Register D1.W erhält die X-Koordinate,
das Register D2.W die Y-Koordinate und
das Register D3.W die maximale Anzahl der einzulesenden Zeichen. Mit Letzterem wird die Breite des Textfensters bestimmt.
Das Register A0.L schließlich erhält die Adresse des Textbuffers, in dem der eingegebene Text nach der Ausführung des Befehls abgelegt ist.

Nach dem Aufruf steht in Register D4.L die tatsächlich eingegebene Anzahl der Zeichen und in Register D5.L das Zeichen, das zuletzt eingetippt wurde. Damit kann man unterscheiden, ob die Eingabe mit einem CR (Code $0D) beendet wurde oder durch ein anderes Zeichen, weil z.B. mehr Zeichen eingegeben wurden als für Register D3.W maximal zulässig waren. Trifft Letzteres zu, wird das CARRY-Flag zurückgesetzt. Wird die Eingabe mit ESC (Code $1B) beendet, wird in D5.L der Wert $1B zurückgegeben. Das CARRY-Flag ist dann 1. Im Textbuffer steht an letzter Stelle eine 0 (Code 00), die das Ende des eingegebenen Textes markiert. Daher muß für den Text immer ein Byte mehr reserviert werden als durch die Eingabe in D3.W für die Breite des Textfensters festgelegt wird. A0.L zeigt nach dem Aufruf auf die Endekennung.

Die Eingaben im Textfenster können mit vielen Befehlen, die denen des Editors entsprechen, bearbeitet werden. Dadurch ist eine komfortable Eingabe in das Fenster möglich.

Folgende Kommandos sind für die Eingabe verfügbar:

CTRL-S	Der Cursor wandert ein Zeichen zurück.
CTRL-D	Der Cursor wandert ein Zeichen vor.
CTRL-U	An der aktuellen Stelle wird ein Leerzeichen eingefügt. Die Zeichen ab Cursorposition werden nach rechts geschoben. Das letzte Zeichen einer Zeile verschwindet.
CTRL-V	Der automatische Einfügemodus wird an- oder ausgeschaltet. Der Einfügemodus hängt mit dem des Editors zusammen. Wenn also hier umgeschaltet wird, gilt das auch für den Editor.
CTRL-G	Das Zeichen an der Cursorposition wird gelöscht. Alle rechts des Cursors stehenden Zeichen rücken nach links.
DEL	Das Zeichen links des Cursors wird gelöscht. Alle Zeichen ab der Cursorposition rücken nach links.

CTRL-T	Alle Zeichen ab der Cursorposition werden gelöscht.
CTRL-A	Der Cursor wird an den Textanfang gesetzt.
CTRL-F	Der Cursor springt auf das erste Feld hinter dem Text.
CTRL-P	Der Zeichensatz wird umgeschaltet. Die Umstellung bezieht sich auch auf den Editor.
CR	Das Textfenster wird verlassen, die Eingabe beendet. Alle Zeichen werden unverändert im Fenster belassen, unabhängig davon, wo der Cursor steht. Im Register D5.L wird der Wert $0D (13) abgelegt.
ESC	Wie CR, allerdings wird das CARRY-Flag auf 1 gesetzt.

Einfaches Beispiel:

Ein maximal 12 Zeichen langer Text kann in einem Fenster eingegeben werden.

```
START:
        MOVEQ       #$22,D0             * Schriftgröße.
        MOVEQ       #10,D1              * X = 10.
        MOVEQ       #100,D2             * Y = 100.
        MOVEQ       #12,D3              * Maximal 12 Zeichen.
        LEA         BUFFER(PC),A0       * Adresse des Buffers.
        MOVEQ       #!READ,D7           * Text einlesen.
        TRAP        #1
        RTS
BUFFER:
        DS.B        13                  * Reservierter Speicher (+ 1 für Endekennung).
```

Bemerkung:

Wenn X = 0 oder 1 oder Y = 0 oder 1 ist, wird ein Teil der Umrahmung des Fensters abgeschnitten, da der Fensterrahmen nicht bei den eingegebenen Koordinaten anfängt, sondern zwei Punkte weiter links und unten. Das Gleiche geschieht auch, wenn die Werte für X oder Y zu groß werden.

TRAP-Nummer	: **12**
Befehlsname	: **CI**
Befehlsgruppe	: Zeicheneingabe.
Kurzbeschreibung	: Ein Zeichen von der Tastatur einlesen.
Eingaberegister	: Keine.
Ausgaberegister	: D0.L = Zeichen im ASCII-Code.
Zerstörte Register	: Keine.
Ab Version	: 3.1
Änderungen zu 4.3	: Die Routine CI hat im Grundprogrammbetrieb eine Hardcopy-Funktion, die unten beschrieben ist. Diese Funktion kann auch in eigenen Programmen aktiviert werden (siehe HARDCOPY). Außerdem kann sie auf die serielle Karte gelenkt werden, was einen Terminalbetrieb ermöglicht (Siehe SER).
Änderungen zu 6.1	: Nein.
Siehe auch	: CSTS (13) CIINIT2 (31) CI2 (32)

Ein Zeichen wird von der Tastatur eingelesen. Dabei wird solange gewartet, bis das Zeichen wirklich eingegeben wird. Das Zeichen erscheint im ASCII-Code im Register D0.L.

Wenn man sich in der Software-Umgebung des Grundprogramms befindet, d.h. daß kein Programm über STARTEN, BIBLIOTHEK, EINZELSCHRITT oder BOOTEN aufgerufen wurde, ist bei jeder Eingabe über diese Routine das Auslösen der HARDCOPY-Funktion möglich. Dazu muß die Tastenkombination CTRL-@ gedrückt werden. Anschließend müssen weitere Eingaben gemacht werden. Es gibt dazu drei Möglichkeiten :

Tastatureingabe (zuerst immer CTRL-@):

1 Es wird eine Hardcopy in den Speicher ausgeführt und sofort auf den Drucker ausgegeben. Dabei wird eine Auflösung für einen 9-Nadel-Drucker gewählt (Entspricht Funktion 12 bei HARDCOPY).

2 Wie 1, aber für 24-Nadel-Drucker (Entspricht Funktion 13 bei HARDCOPY).

3 Es wird nur eine Hardcopy in den Speicher durchgeführt. Dabei muß genau wie bei 1 und 2 hinter der Symboltabelle ein Speicherbereich von 16 Kbyte zur Verfügung stehen. Die Hardcopy wird bei jeder dieser Funktionen 18 Byte hinter dem Symboltabellenende abgelegt. Dadurch kann die Hardcopy immer wieder bearbeitet werden, wenn die Symboltabelle nicht verändert wurde (Entspricht Funktion 8 bei HARDCOPY).

Die Hardcopy-Funktion ist auch beim Programm HARDCOPY beschrieben. Sie kann nämlich auch in jedem beliebigen Programm aktiviert werden.

Beispiel zur Berechnung der Adresse für die Bildschirmablage:

```
START:
        MOVEQ     #!GETSYM,D7       * Anfang der Symboltabelle.
        TRAP      #1
        MOVEQ     #!GETNEXT,D7      * Länge der Symboltabelle.
        TRAP      #1
        ADDA.L    D0,A0
        ADDA.W    #18,A0            * A0 zeigt jetzt auf die Hardcopy.
        ......                      * Auswertung der Hardcopy.
```

Außerdem kann die Routine CI auf andere Geräte umgelenkt werden. Eine Umlenkung auf die serielle Karte erfolgt mit dem Befehl SER. Eine Umlenkung auf eine Benutzerroutine ist ebenfalls leicht möglich. Dazu muß man in die Speicherzelle $2b (IOSTATB) den Wert 6 schreiben. Dann erfolgt beim Aufruf der CI-Routine ein Sprung auf Adresse $18 (USERCI). Dort muß ein Sprung auf eine eigene Routine stehen. Im Register D0.W wird der Wert 2 an die Benutzerroutine übergeben, um die CI-Routine von der CI2-Routine unterscheiden zu können. Dieses verwendet nämlich bei der Umschaltung die gleiche Adresse, jedoch mit dem Wert 0 in D0.W.

Wenn man die Routine wieder zurücksetzen will, muß man den Wert 0 auf Adresse $2b schreiben. Die Adressen beziehen sich auf den Anfang des Variablenbereichs, der mit dem Unterprogramm GETVAR oder aus dem Register A5.L ermittelt werden kann.

Gleichzeitig mit CI wird auch CSTS umgeschaltet, denn CI und CSTS gehören immer zusammen.

Es ist darauf zu achten, daß der zurückgegebene Wert in D0.L der Konvention der CI-Routine entspricht.

Einfaches Beispiel:

Es wird ein Zeichen von der Tastatur gelesen.

```
START:
        MOVEQ     #!CI,D7          * Zeichen einlesen.
        TRAP      #1
        RTS                        * Ergebnis ist D0.L.
```

Hier wird so lange ein Zeichen gelesen, bis die Taste CR oder auch RETURN gedrückt wird.

```
SCHLEIFE:
        MOVEQ     #!CI,D7          * Zeichen einlesen
        TRAP      #1
        CMP.B     #$D,D0           * und warten bis CR
        BNE.S     SCHLEIFE         * gedrückt wird.
        RTS                        * Dann erst Rückkehr.
```

Kombiniertes Beispiel:

Funktionen der Schildkröte durch Tasten steuern.

```
START:
        MOVEQ     #!CI,D7          * Zeichen einlesen.
        TRAP      #1
        CMP.B     #'f',D0          * War es die Taste Klein-F?
        BNE.S     SPRUNG1          * Nein, dann weiter.
        MOVEQ     #10,D0           * Ja, dann Schreiten.
        MOVEQ     #!SCHREITE,D7    * In der Schildkrötensprache.
        TRAP      #1
        BRA.S     START            * Dann wieder zurück.
SPRUNG1:
        CMP.B     #'d',D0          * War es die Taste Klein-D ?
        BNE.S     SPRUNG2          * Nein, dann weiter.
        MOVEQ     #45,D0           * Ja, dann um 45 Grad drehen.
        MOVEQ     #!DREHE,D7       * Befehl für Drehen.
        TRAP      #1
        BRA.S     START            * Und wieder zurück.
SPRUNG2:
        CMP.B     #'h',D0          * War es die Taste Klein-H ?
        BNE.S     SPRUNG3          * Nein, dann weiter.
        MOVEQ     #!HEBE,D7        * Keine Schildkrötenspur mehr.
        TRAP      #1
        BRA.S     START            * Dann zurück.
SPRUNG3:
        CMP.B     #'s',D0          * Oder war es die Taste Klein-S ?
        BNE.S     SPRUNG4          * Nein, dann weiter.
        MOVEQ     #!SENKE,D7       * Jetzt wieder Schreibspur.
        TRAP      #1
        BRA.S     START            * Und wieder zurück.
SPRUNG4:
        CMP.B     #$0D,D0          * RETURN gedrückt ?
        BNE.S     START            * Nein, dann zurück.
        RTS                        * Sonst Ende.
```

Dies ist eine Mini-Schildkrötensprache. Wenn man das Programm startet und die Taste f drückt, werden 10 Schritte ausgeführt und eine kurze Linie erscheint. Mit der Taste d kann man die Schildkröte in 45 Grad-Schritten drehen. Mit der Taste h kann man verhindern, daß eine Schreibspur hinterlassen wird und mit der Taste s kann man sie wieder einschalten. Damit kann man kleine Zeichnungen erstellen. Das Programm wird mit RETURN verlassen.

TRAP-Nummer	: **13**
Befehlsname	: **CSTS**
Befehlsgruppe	: Zeicheneingabe.
Kurzbeschreibung	: Liegt ein Zeichen von der Tastatur vor ?
Eingaberegister	: Keine.
Ausgaberegister	: D0.L = Flag, ob Zeichen vorhanden ist. Flags(d0.b).
Zerstörte Register	: Keine.
Ab Version	: 3.1
Änderungen zu 4.3	: Die Routine kann auf die serielle Karte gelenkt werden. Dazu dient die Routine SER.
Änderungen zu 6.1	: Nein.
Siehe auch	: CI (12) CIINIT2 (31) CI2 (32)

Eng verwandt mit dem CI-Befehl ist der Befehl CSTS. Damit ist die Abfrage möglich, ob eine Taste gedrückt wurde oder nicht. Im Register D0.L erscheint eine 0, wenn keine Taste gedrückt wurde, andernfalls der Wert SFF. Das Zeichen wird aber nicht von der Tastatur eingelesen, sondern muß mit dem Befehl CI geholt werden. Der Inhalt des Registers D0.L bleibt so lange erhalten, bis das Zeichen mit CI geholt wurde. Die FLAGS sind ebenfalls entsprechend dem Wert des eingelesenen Zeichens (d0.b) gesetzt, weshalb gleich nach dem Aufruf ein Sprung (z B. BEQ oder BNE) durchgeführt werden kann.

Wenn die Routine CI auf die serielle Karte gelenkt wird, geschieht das gleichzeitig auch mit der Routine CSTS.

Ein Umlenken der CSTS-Routine auf eine Benutzerschnittstelle ist ebenfalls möglich. Das Umlenken funktioniert gemeinsam mit der Umlenkung der CI-Routine und kann dort nachgelesen werden. Allerdings wird bei der Umlenkung der CSTS-Routine die Adresse $1E (USERCSTS) angesprungen. Wie beim Befehl CI wird auch hier der Wert 2 in Register D0.L übergeben. Die Adresse $1E bezieht auf den Anfang des Variablenbereichs.

Wird die CSTS-Routine umgelenkt, muß natürlich die für sie geltende Konvention der CSTS-Routine eingehalten werden.

Einfaches Beispiel:

Dieses Programm prüft, ob eine Taste gedrückt wurde.

```
START:
        MOVEQ     #!CSTS,D7        * Status holen.
        TRAP      #1
        RTS                        * D0.L = 0, wenn kein Zeichen da.
```

Kombiniertes Beispiel:

Es wird so lange an einem Kreis gezeichnet, bis eine Taste gedrückt wird.

```
START:
        MOVE.W    #200,D1          * X-Position der Schildkröte.
        MOVE.W    #256,D2          * Y-Position.
        MOVE.W    #-90,d3          * Blickrichtung der Schildkröte.
        MOVEQ     #!SET,D7         * Schildkröte setzen.
        TRAP      #1
SCHLEIFE:
        MOVEQ     #1,D0            * Zeichnen eines Kreises (= 360-Eck).
        MOVEQ     #!SCHREITE,D7    * 1 Punkt weiter.
        TRAP      #1
        MOVEQ     #1,D0
        MOVEQ     #!DREHE,D7       * Ein Grad drehen.
        TRAP      #1
        MOVEQ     #1,D0            * Verzögerung 1/10 Sekunde.
        MOVEQ     #!DELAY,D7
        TRAP      #1
        MOVEQ     #!CSTS,D7        * Fragen, ob Taste gedrückt.
        TRAP      #1
        BEQ.S     SCHLEIFE         * Nein, dann zurück.
        RTS                        * Ja, dann aufhören.
```

TRAP-Nummer	: **14**	
Befehlsname	: **RI**	
Befehlsgruppe	: CAS-Baugruppe	
Kurzbeschreibung	: Es wird ein Zeichen von der CAS-Baugruppe gelesen.	
Eingaberegister	: Keine.	
Ausgaberegister	: D0.L = Enthält das gelesene Zeichen. Flags(d0.b). Carry = Gibt an, ob der Befehl abgebrochen wurde.	
Zerstörte Register	: Keine.	
Ab Version	: 3.1	
Änderungen zu 4.3	: Ja.	
Änderungen zu 6.1	: Die Routine kann mit ESC abgebrochen werden.	
Siehe auch	: PO (15)	RELAN (118) RELAUS (119)

Ein Zeichen wird von der CAS-Schnittstelle gelesen. Dabei wird so lange gewartet, bis ein Zeichen ankommt. Das Zeichen erscheint im ASCII-Code im Register D0.L.

Mit der Taste ESC kann der Lesevorgang abgebrochen werden. Es wird dann das CARRY-Flag gesetzt, ansonsten ist es auf Null.

Einfaches Beispiel:

Ein Zeichen von der CAS-Baugruppe lesen.

```
START:
        MOVEQ     #!CI,D7          * Zeichen einlesen.
        TRAP      #1
        RTS                        * Zeichen in D0.L (D0.B).
```

Kombiniertes Beispiel:

Es werden so lange Zeichen gelesen, bis mit ESC abgebrochen wird.

```
START:
        LEA       BUFFER(PC),A0    * Zieladresse in A0.
SCHLEIFE:
        MOVEQ     #!RI,D7
        TRAP      #1               * Zeichen lesen.
        BCS.S     ENDE             * Abbruch mit ESC.
        MOVE.B    D0,(A0)+         * Zeichen in Speicher ablegen.
        BRA.S     SCHLEIFE         * Wiederholen.
ENDE:
        RTS
BUFFER:
        DS.B      1000             * Z. B. 1000 Zeichen.
```

Das Programm kann zum Beispiel dazu genutzt werden, um ein Band mit Daten in den Speicher einzulesen. Dabei ist aufgrund des verwendeten PE-Verfahrens das erste eingelesene Zeichen ungültig. Bei Daten, die z. B. vom Grundprogramm stammen, wird daher am Anfang der Routine immer ein unbedeutender Datenstrom den einzulesenden Daten vorweggeschickt.

Das obige Programm kann nur mit der Tastenkombination ESC beendet werden, denn hier ist ja die Länge der Daten nicht bekannt.

Mit dem Programm ANSEHEN kann man den Inhalt des Buffers überprüfen.

TRAP-Nummer	: **15**	
Befehlsname	: **PO**	
Befehlsgruppe	: CAS-Baugruppe.	
Kurzbeschreibung	: Gibt ein Zeichen über die CAS-Baugruppe aus.	
Eingaberegister	: D0.B = Zeichen, das ausgegeben werden soll.	
Ausgaberegister	: Carry = Gibt an, ob der Befehl abgebrochen wurde.	
Zerstörte Register	: Keine.	
Ab Version	: 3.1	
Änderungen zu 4.3	: Ja.	
Änderungen zu 6.1	: Der Befehl kann mit ESC abgebrochen werden.	
Siehe auch	: RI (14)	RELAN (118) RELAUS (119)

Ein Zeichen wird über die CAS-Baugruppe ausgeben. Dabei steht das Zeichen zuvor im Register D0.B.

Auch dieser Befehl kann mit ESC abgebrochen werden. Dann ist das CARRY-Flag gesetzt, ansonsten ist es zurückgesetzt.

Einfaches Beispiel:

Das Zeichen A wird über die CAS-Schnittstelle ausgegeben.

```
START:
        MOVEQ     #'A',D0          * Ausgabe des Zeichens A.
        MOVEQ     #!PO,D7
        TRAP      #1               * Ausgabe über CAS.
        RTS
```

Kombiniertes Beispiel:

Es folgen zwei Programme, die das Lesen und Schreiben über die CAS-Baugruppe ermöglichen. Dabei wird immer bis zu einer Ende-Null geschrieben oder gelesen.

Mit SCHREIBE wird der Text Hallo geschrieben. Beim Lesen wird er auch noch auf dem Bildschirm ausgegeben.

```
SCHREIBE:
        MOVEQ     #10-1,D3         * 10 Synchronisationsbytes.
SCHRLP0:
        MOVEQ     #$FF,D0          * Z. B. $FF verwenden.
        MOVEQ     #!PO,D7
        TRAP      #1               * Zeichen ausgeben.
        BCS.S     SCHRENDE         * Abbruch.
        DBRA      D3,SCHRLP0       * Ende der Schleife.
        CLR.B     D0               * Kennung für Start.
        MOVEQ     #!PO,D0
        TRAP      #1               * Zeichen ausgeben.
        BCS.S     SCHRENDE         * Abbruch.
        LEA       AUSBUF(PC),A0    * Adresse des Textes laden.
SCHRLP1:
        MOVE.B    (A0)+,D0         * Datenbyte laden.
        MOVEQ     #!PO,D7
        TRAP      #1               * Zeichen ausgeben.
        BCS.S     SCHRENDE         * Abbruch.
        TST.B     D0               * D0 testen,
        BNE.S     SCHRLP1          * bis das Zeichen NUL erscheint.
SCHRENDE:
        RTS                        * Ausgabe beendet.
AUSBUF:
        DC.B      'HALLO',0        * Text.
```

```
LIES:
        MOVEQ       #!RI,D7             * Zeichen holen.
        TRAP        #1
        BCS.S       LIESENDE            * Abbruch.
        CMP.B       #$FF,D0             * Suchen nach den Sync-Bytes.
        BNE.S       LIES                * Immer weiter.
LIESLP0:
        MOVEQ       #!RI,D7             * Dann muß 0 oder $FF folgen.
        TRAP        #1
        BCS.S       LIESENDE            * Abbruch.
        CMP.B       #$FF,D0             * Bei $FF OK.
        BEQ.S       LIESLP0             * Sonst weiter suchen.
        TST.B       D0                  * Sonst muß es 0 sein .
        BNE.S       LIESLP0             * oder bei Störung zurück.
        LEA         EINBUF(PC),A0       * Ziel für Eingabe.
LIESLP1:
        MOVEQ       #!RI,D7             * Zeichen holen.
        TRAP        #1
        BCS.S       LIESENDE            * Abbruch.
        MOVE.B      D0,(A0)+            * Und ablegen
        BNE.S       LIESLP1             * bis 0 aufgetreten.
        LEA         EINBUF(PC),A0       * Dann den Text auf
        MOVEQ       #$22,D0             * Bildschirm ausgeben.
        MOVEQ       #2,D1               * X = 2.
        MOVE        #128,D2             * Y = 128.
        MOVEQ       #!WRITE,D7
        TRAP        #1
LIESENDE:
        RTS                             * Ende.
EINBUF:
        DS.B        100                 * Hier Platz zum Einlesen.
```

Der Aufzeichungsvorgang mit SCHREIBE geht bei diesem Beispiel sehr schnell, daher meldet sich das Menü auch sofort wieder.

Bei der Wiedergabe muß der Text auf dem Bildschirm erscheinen, dazu wird das Programm LIES gestartet.

In der Praxis muß man dieses einfache Programm noch um eine Prüfroutine erweitern. Denn treten beim Lesen Fehler auf, muß eine Fehlermeldung erfolgen.

Meist wird dazu eine Prüfsumme verwendet. Das heißt, man summiert alle Datenbytes auf, verwendet aber nur z. B. 16 Bits des Ergebnisses und gibt es als zwei Bytes am Schluß mit aus. Beim Einlesen bildet man erneut die Prüfsumme und vergleicht diese mit dem aufgezeichneten Wert. So ist es auch im Grundprogramm realisiert, wenn man damit Texte oder Daten aufzeichnet. Ferner ist es nützlich, die Anzahl der Datenbytes am Anfang mit aufzuzeichnen, um auch beliebige Daten abspeichern zu können, die z. B. auch eine 0 enthalten.

TRAP-Nummer	:	**16**
Befehlsname	:	**CLR**
Befehlsgruppe	:	Grafik.
Kurzbeschreibung	:	Alle vier Seiten der GDP-Karte werden gelöscht.
Eingaberegister	:	Keine.
Ausgaberegister	:	Keine.
Zerstörte Register	:	D0
Ab Version	:	3.1
Änderungen zu 4.3	:	Der Hardscrollwert wird auf Null gesetzt.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	MOVETO (8), DRAWTO (9), CLPG (17), WAIT (18), CMD (26), NEWPAGE (27), SETFLIP (34), SETPEN (37), ERAPEN (38), CMDPRINT (40), AUTOFLIP (60), SETXOR (77), GETXOR (78), SETCOLOR (79), GETCOLOR (80), GETXY (103), HARDCOPY (125), GRAFIK (126), GDPVERS (127)

Alle vier Bildseiten der GDP werden gelöscht. Der Vorgang dauert ca. 20 ms * 4, also ca. 80 ms.

Die aktuelle Lese- und Schreibseite wird auf 0 eingestellt, sodaß Bildschirmausgaben anschließend auf der Seite 0 erfolgen und auch dort sichtbar sind.

Ab Version 6.1 wird auch der HARDSCROLL-Wert auf Null gesetzt, d. h. daß sich der Bildschirm in seiner "normalen" Lage befindet und nicht nach oben oder unten verschoben ist. Das Verschieben kann z. B. durch die Routine CO2 verursacht werden. Bei der alten GDP passiert so etwas natürlich nicht.

Einfaches Beispiel:

Alle vier Bildschirmseiten werden gelöscht.

```
START:
        MOVEQ     #!CLR,d7
        TRAP      #1              * Löschen des Bildschirms.
        RTS
```

Bemerkung:

Falls nur eine Bildschirmseite gelöscht werden soll, und diese die aktuelle Schreib- und Leseseite ist, kann das auch durch Programmierung des Grafik-Prozessors geschehen (siehe Handbuch GDP).

TRAP-Nummer	: **17**
Befehlsname	: **CLPG**
Befehlsgruppe	: Grafik.
Kurzbeschreibung	: Es wird die aktuelle Schreibseite gelöscht.
Eingaberegister	: Keine.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 3.1
Änderungen zu 4.3	: Nein.
Änderungen zu 6.1	: Nein.
Siehe auch	: MOVETO (8), DRAWTO (9), CLR (16), WAIT (18), CMD (26), NEWPAGE (27), SETFLIP (34), SETPEN (37), ERAPEN (38), CMDPRINT (40), AUTOFLIP (60), SETXOR (77), GETXOR (78), SETCOLOR (79), GETCOLOR (80), GETXY (103), HARDCOPY (125), GRAFIK (126), GDPVERS (127)

Damit kann eine unsichtbare Seite gelöscht werden. Der Vorgang dauert aber wesentlich länger als der schnelle Löschbefehl des GDP, denn man kann bei unterschiedlicher Schreib-und Leseseite nicht den normalen Löschbefehl (Code 4 oder 7) verwenden.

Im Grundprogramm wird das Löschen der unsichtbaren Seite durch Überschreiben mit dem Befehl $0A (5 x 8 Block) in der Schriftgröße 16 erreicht.

Den CLPG-Befehl benötigt man, wenn man auf einer unsichtbaren Bildseite ein neues Bild aufbauen will und dabei eine andere Bildseite sieht.

Einfaches Beispiel:

Die aktuelle Schreibseite wird gelöscht.

```
START:
        MOVEQ     #!CLPG,D7
        TRAP      #1              * Aktuelle Schreibseite löschen.
        RTS
```

Komplexeres Beispiel:

Ein Uhr-Zeiger bewegt sich auf dem Bildschirm. Abbruch durch Tastendruck.

```
START:
        MOVEQ     #1,D6           * Schreibseite.
        MOVEQ     #0,D4           * Leseseite.
        MOVEQ     #0,D5           * Zähler für SIN, COS.
SCHLEIFE:
        MOVE      D6,D0           * Schreibseite.
        MOVE      D4,D1           * Leseseite.
        MOVEQ     #!NEWPAGE,D7
        TRAP      #1              * Seite einstellen.
        MOVEQ     #!CLPG,D7
        TRAP      #1              * Unsichtbare Seite löschen.
        MOVE      #256,D1         * X = 256.
        MOVE      #128,D2         * Y = 128.
        MOVEQ     #!MOVETO,D7     * Startpunkt Mitte.
        TRAP      #1
        MOVE      D5,D0
        MOVEQ     #!COS,D7        * X = COS256(phi) + 256.
        TRAP      #1
        ADD       #256,D0
        MOVE      D0,D1           * X-Koordinate.
        MOVE      D5,D0
        MOVEQ     #!SIN,D7        * Y = SIN256(phi) / 2 + 128.
```

```
TRAP    #1
ASR     #1,D2
ADD     #128,D0         * Damit zentrisch.
MOVE    D0,D2           * Y-Koordinate.
MOVEQ   #!DRAWTO,D7     * Ziellinie.
TRAP    #1
EXG     D6,D4           * Lese- und Schreibseite vertauschen.
ADD     #10,D5          * 10 Grad dazu.
MOVEQ   #!CSTS,D7
TRAP    #1
BEQ.S   SCHLEIFE        * Und wiederholen, bis Zeichen eingegeben wird.
RTS
```

Wenn man die Graphik schneller haben will, verwendet man nicht den CLPG-Befehl, sondern löscht das alte Bild dadurch, daß man den GDP auf Löschen umstellt (ERAPEN) und dann das alte Bild erneut schreibt. Dann darf man nicht vergessen, den GDP vorher wieder auf Schreiben (SETPEN) umzustellen.

CLPG benötigt man eigentlich nur dann, wenn man das alte Bild nicht mehr rekonstruieren kann oder wenn das erneute Einschreiben länger dauert als die Ausführung des CLPG-Befehl.

TRAP-Nummer	: **18**		
Befehlsname	: **WAIT**		
Befehlsgruppe	: Grafik.		
Kurzbeschreibung	: Das Programm wartet, bis der GDP-Prozessor bereit ist.		
Eingaberegister	: Keine.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: MOVETO (8)	DRAWTO (9)	CLR (16)
	CLPG (17)	CMD (26)	NEWPAGE (27)
	SETFLIP (34)	SETPEN (37)	ERAPEN (38)
	CMDPRINT (40)	AUTOFLIP (60)	SETXOR (77)
	GETXOR (78)	SETCOLOR (79)	GETCOLOR (80)
	GETXY (103)	HARDCOPY (125)	GRAFIK (126)
	GDPVERS (127)		

Wenn man direkt Daten auf die GDP-Ports ausgeben will, muß man immer warten, bis der GDP den letzten Befehl ausgeführt hat. Dazu benutzt man den WAIT-Befehl.

Einfaches Beispiel:

Warten, bis GDP fertig ist.

```
START:
        MOVEQ       #!WAIT,D7
        TRAP        #1              * Warten, bis GDP fertig.
        RTS
```

Komplexeres Beispiel:

Der Buchstabe A wird in verschiedenen Größen ausgegeben.

```
START:
        MOVE        #!SYSTEM,D7
        TRAP        #1              * System-Informationen holen.
        AND.W       #7,D0           * Nur CPU lassen.
        MOVE        D0,D1           * Merken.
        MULS        #$FF70,D0
        MOVEA.L     D0,A0           * GDP Port 0.
        MULS        #$FF73,D1
        MOVEA.L     D1,A1           * GDP Port 1.
        CLR         D3              * Größe zurücksetzen.
        CLR         D1              * X = 0, Y = 0.
        CLR         D2
        MOVEQ       #!MOVETO,D7
        TRAP        #1              * Position einstellen.
SCHLEIFE:
        MOVEQ       #!WAIT,D7
        TRAP        #1              * Warten, bis GDP fertig.
        ADDQ.B      #1,D3           * Größe verändern.
        MOVE.B      D3,(A1)         * CSIZE-Register.
        MOVE.B      #'A',(A0)       * Buchstabe A ausgeben.
        MOVEQ       #1,D0           * 1/10 Sekunde warten.
        MOVEQ       #!DELAY,D7
        TRAP        #1
        MOVEQ       #!CLR,D7        * Dann erst löschen.
        TRAP        #1
        MOVEQ       #!CSTS,D7
        TRAP        #1
        BEQ.S       SCHLEIFE        * Immer weiter, bis Taste gedrückt.
        RTS
```

Will man die Ausgabe flimmerfrei haben, so muß man mit zwei Bildebenen arbeiten.

TRAP-Nummer	: **19**		
Befehlsname	: **SCHR16TEL**		
Befehlsgruppe	: Schildkrötengrafik.		
Kurzbeschreibung	: Die Schildkröte schreitet einen 1/16 Schritt.		
Eingaberegister	: D0.W = Anzahl der Schritte und Richtung.		
Ausgaberegister	: Keine.		
Zerstörte Register	: D0		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SCHREITE (1)	DREHE (2)	HEBE (3)
	SENKE (4)	SET (7)	FIRSTTIME (36)
	GRAPOFF (39)	HIDE (47)	SHOW (48)
	AUFXY (92)	KORXY (93)	AUFK (94)
	GETK (95)		

Der Befehl ermöglicht es, die Schildkröte jeweils einen 1/16 Bildpunkt zu bewegen. Das ist z. B. nötig, um Kreise mit unterschiedlichen Radien zu zeichnen.

Die Anzahl der Schritte wird in Register D0.W übergeben.

Einfaches Beispiel:

Die Schildkröte schreitet um einen Bildpunkt.

```
START:
        MOVEQ       #16,D0              * Schreitet um einen Bildpunkt.
        MOVEQ       #!SCHR16TEL,D7
        TRAP        #1
        RTS
```

Jetzt soll ein Kreis mit vorgegebenem Radius gezeichnet werden:

Für einen Kreis mit vorgegebenem Radius gibt es eine bekannte Formel, um den Umfang zu bestimmen:

U = 2 * PI * R,

wobei PI = 3.141592... , R = Radius und U = Umfang ist.

Das Kreisprogramm hat die allgemeine Form:

```
START:
        MOVE        #360-1,D3           * 360 mal.
SCHLEIFE:
        MOVE        #ANZAHL,D0          * Muß berechnet werden.
        MOVEQ       #!SCHR16TEL,D7
        TRAP        #1                  * Schildkröte bewegen.
        MOVEQ       #1,D0
        MOVEQ       #!DREHE,D7          * Schildkröte um 1 Grad drehen.
        TRAP        #1
        DBRA        D3,SCHLEIFE
        RTS
```

Es gilt nun ANZAHL zu berechnen.

Der Umfang ergibt sich aus der Formel:

U = 360 * ANZAHL / 16,

wobei als Einheit 1 Bildpunkt gewählt wurde.

Damit läßt sich die Unbekannte ANZAHL ermitteln:

ANZAHL = (2 * PI * R) * 16 / 360.

Man kann die Formel mit einer EQU-Anweisung annähern:

ANZAHL EQU (R*100)/358.

Nun kann mit Werten für R experimentiert werden.

Die Genauigkeit ist bei diesem Verfahren nicht sehr groß, da nur mit ganzzahligen Werten gerechnet wird. Die Ungenauigkeit wird größer, je kleiner der Radius wird. Sollen Kreise mit größerer Genauigkeit gezeichnet werden, sollte man mit der SINUS- und COSINUS-Funktion arbeiten. Es kann natürlich auch die schnellere Kreisfunktion des GRAFIK-Pakets benutzt werden.

TRAP-Nummer	:	20
Befehlsname	:	**CLRSCREEN**
Befehlsgruppe	:	Zeichenausgabe.
Kurzbeschreibung	:	Löscht Bildschirm und bereitet ihn für Textausgabe vor.
Eingaberegister	:	Keine.
Ausgaberegister	:	Keine.
Zerstörte Register	:	Keine.
Ab Version	:	3.1
Änderungen zu 4.3	:	Der Hardscrollwert wird auf Null gesetzt.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	CO (21), LO (22), SIZE (25), CO2 (33), CRT (49), LST (50), USR (51), NIL (52), SETPASS (55), CURSEIN (61), CURSAUS (62), CHAR (63), CURON (81), CUROFF (82), CRLF (99), GETLINE (100), GETCURXY (101), SETCURXY (102), LSTS (117), CO2SER (129)

Die vier (sichtbaren und unsichtbaren) Seiten des Bildschirm werden gelöscht. Gleichzeitig wird die Textausgabe vorbereitet. Deshalb erscheint ein blinkendes Cursorfeld links oben auf dem Bildschirm. Auch der interne Bildschirmspeicher wird gelöscht. Außerdem wird der HARDSCROLL-Wert wieder auf Null gesetzt. Das bedeutet, daß der Bildschirm wieder so liegt, als würde er ohne HARDSCROLL betrieben.

Wenn man die Ausgaberoutinen CO und CO2 verwenden will, muß man die Ausgabe immer mit CLRSCREEN vorbereiten und damit den Cursor einschalten.

Einfaches Beispiel:

Bildschirm löschen und für CO2 oder CO vorbereiten.

```
START:
        MOVEQ       #!CLRSCREEN,D7
        TRAP        #1                  * Bildschirm löschen.
        RTS
```

Komplexeres Beispiel:

Der Bildschirm wird gelöscht und ein Text ausgegeben.

```
START:
        MOVEQ       #!CLRSCREEN,D7      * Bildschirm löschen.
        TRAP        #1
        LEA         TEXT(PC),A0         * Adresse des Textes.
SCHLEIFE:
        MOVE.B      (A0)+,D0            * Zeichen holen.
        BEQ.S       ENDE                * Bei Null ist das Ende erreicht.
        MOVEQ       #!CO2,D7            * Sonst Zeichen ausgeben.
        TRAP        #1
        BRA.S       SCHLEIFE            * Wiederholen.
ENDE:
        RTS
TEXT:
        DC.B        'Ausgabe auf dem',$d,$a
        DC.B        'Bildschirm mit CO2',$d,$a
        DC.B        'und CLRSCREEN !',0
```

Der Cursor blinkt danach direkt hinter dem Text.

TRAP-Nummer	:	**21**
Befehlsname	:	**CO**
Befehlsgruppe	:	Zeichenausgabe.
Kurzbeschreibung	:	Gibt ein Zeichen auf dem Bildschirm aus.
Eingaberegister	:	D0.B = Zeichen, das ausgegeben werden soll.
Ausgaberegister	:	Keine.
Zerstörte Register	:	D0/A0-A2
Ab Version	:	3.1
Änderungen zu 4.3	:	Neue Befehle für Hardscroll. Es erfolgt keine Zeichensatzanpassung bei der Ausgabe.
Änderungen zu 6.1	:	Hardscroll ist auch mit Cursordarstellung möglich. Hardcopy-Funktion gibt den Bildschirminhalt als Text aus.
Siehe auch	:	CLRSCREEN (20), LO (22), SIZE (25), CO2 (33), CRT (49), LST (50), USR (51), NIL (52), SETPASS (55), CURSEIN (61), CURSAUS (62), CHAR (63), CURON (81), CUROFF (82), CRLF (99), GETLINE (100), GETCURXY (101), SETCURXY (102), LSTS (117), CO2SER (129)

Mit Hilfe des Befehls wird ein Zeichen im Textmode ausgegeben. Das Zeichen steht in Register D0.B. Es können auch Steuerzeichen ausgegeben werden.

CO ist eine Ausgaberoutine nur für den Bildschirm. Sie kann nicht umgelenkt werden. Sie wird von CO2 aufgerufen, wenn auf den Bildschirm ausgegeben werden soll.

Bevor man CO verwenden kann, muß CLRSCREEN ausgeführt werden, sonst erscheint kein Cursor auf dem Bildschirm.

Die Zeichengröße ist bei der Ausgabe, wie man sie im OPTIONEN-Menü eingestellt hat (40 oder 80 Zeichen pro Zeile). Es können andere Größen mit dem Befehl SIZE eingestellt werden. Danach ist aber ein CLRSCREEN-Aufruf nötig.

Seit Version 6.0 ist auch der HARDSCROLL mit der neuen GDPHS möglich. Dabei wird beim Scrollen nicht mehr der ganze Bildschirm neu aufgebaut, sondern es wird nur der Inhalt eines Registers auf der GDPHS-Karte verändert und die oberste Zeile wird gelöscht. Der HARDSCROLL-Modus darf ab der Version 6.2 auch mit der Cursordarstellung betrieben werden. Weiterhin sind in diesem Modus alle Befehle erlaubt, die nur in der Cursorzeile zu Veränderungen führen. Befehle wie REST DES BILDSCHIRMS LÖSCHEN oder ZEILE EINFÜGEN sind nicht erlaubt. Diese werden vom Programm zwar durchgeführt. Es kommt aber in den meisten Fällen zu Bildschirmstörungen. Der Programmierer sollte also auf diese Befehle verzichten. Außerdem darf z. B. keine Statuszeile beim HARDSCROLL-Betrieb vorhanden sein, da der ganze Bildschirm verschoben wird. Der HARDSCROLL ist deshalb nur bei Ausgaben mit CO2, CHAR oder CO möglich und kann somit auch nicht im Editor verwendet werden.

Einfaches Beispiel:

Das Zeichen A wird ausgegeben.

```
START:
        MOVEQ     #!CLRSCREEN,D7      * Bildschirm löschen.
        TRAP      #1
        MOVEQ     #'A',D0             * Zeichen A auf dem
        MOVEQ     #!CO,D7             * Bildschirm ausgeben.
        TRAP      #1
        RTS
```

Beispiel eines Terminals:

Die Zeichen, die auf der Tastatur eingegeben werden, werden gleich ausgegeben. Mit der Tastenkombination CTRL-A kann das Programm abgebrochen werden. Es funktionieren auch alle unten beschriebenen Steuerfunktionen.

```
START:
        MOVEQ     #!CLRSCREEN,D7
        TRAP      #1                  * Bildschirm löschen.
SCHLEIFE:
        MOVEQ     #!CI,D7             * Zeichen holen.
        TRAP      #1
        CMP.B     #1,D0               * CTRL-A bricht ab,
        BEQ.S     ENDE                * wenn gedrückt.
```

```
        MOVEQ     #!CO,D7          * Sonst ausgeben
        TRAP      #1
        BRA.S     SCHLEIFE         * und wiederholen.
ENDE:
        RTS                        * Abbruch.
```

Nun zu den Befehlen:

Zeichen

Im Textbereich von $20 bis $7F werden die normalen ASCII-Zeichen ausgegeben.

Zeichen, die einen Code größer als $80 besitzten, sind Sonderzeichen wie Ä etc. Ab Version 4.0 des Grundprogramms werden auch diese direkt ausgegeben, um einen gemischten Betrieb mit den Klammern wie [etc. zu ermöglichen.

Wenn ein Zeichen ganz rechts eingegeben wird, springt der Cursor in die nächste Zeile nach links. Wird ein Zeichen in der letzten Zeile ganz rechts ausgeben, folgt ein Scrollvorgang auf dem Bildschirm. Der Bildschirm wird dabei um eine Zeile nach rechts verschoben.

Einfaches Beispiel:

Es werden alle lesbaren ASCII-Zeichen außer den Sonderzeichen ausgegeben.

```
START:
        MOVEQ     #!CLRSCREEN,D7   * Bildschirm löschen.
        TRAP      #1
        MOVEQ     #$20,D0          * Erstes Zeichen.
SCHLEIFE:
        MOVE      D0,-(A7)         * Zeichen merken,
        MOVEQ     #!CO,D7          * dann ausgeben.
        TRAP      #1
        MOVE      (A7)+,D0         * Zeichen wieder zurück.
        ADDQ.B    #1,D0            * Nächstes Zeichen.
        CMP.B     #$80,D0          * Ende erreicht ?
        BNE.S     SCHLEIFE         * Nein, dann weiter.
        RTS                        * OK.
```

Achtung! Man sieht nur dann alle Zeichen, wenn man mit dem OPTIONEN-Menü auf 80 Zeichen pro Zeile umgeschaltet hat.

CTRL-Befehle

Hier werden Codes im Bereich $0 bis $1F als Befehle interpretiert. Dabei hat der Code $1B eine besonderer Funktion, da mit ihm weitere Befehle, die ESC-Befehle, angesprochen werden.

Neben dem Code steht der Control-Code. Wenn man die Taste CTRL und gleichzeitig die angegebene Taste nach dem Bindestrich drückt, erhält man den gewünschten Code.

Code $08: CTRL-H

Der Cursor wandert ein Zeichen zurück. Man nennt dieses Zeichen auch BACKSPACE (BS), was Rückwärtsschritt bedeutet. Befindet sich der Cursor schon auf der linken Seite des Bildschirmes, wandert er eine Zeile nach oben und kommt ganz rechts zum Vorschein. Stand der Cursor in der obersten Zeile ganz links, kann man ihn nicht mehr bewegen.

Code $09: CTRL-I

Der Cursor wird ein Zeichen nach rechts verschoben. War der Cursor ganz rechts auf der Zeile, erfolgt ein Sprung in die nächste Zeile. Befand sich der Cursor dabei auch auf der letzten Zeile, wird der Bildausschnitt um eine Zeile nach oben verschoben.

Code $0A: CTRL-J

Der Cursor wandert eine Zeile nach unten. Wenn der Cursor auf der Zeile 23 (unterste Zeile) stand, wandert er nicht weiter nach unten, sondern der gesamte Bildschirminhalt wird um eine Zeile nach oben verschoben. Diese Funktion nennt man Scroll. Der Befehl wird auch LINEFEED genannt.

Code $0B: CTRL-K

Der Cursor wird eine Zeile nach oben transportiert. Er verändert dabei seine X-Koordinate nicht. Wenn der Cursor schon in der obersten Zeile stand, bleibt er stehen.

Code $0C: CTRL-L

Der Cursor wird ein Zeichen nach rechts verschoben. War der Cursor ganz rechts auf der Zeile, springt er in die nächste Zeile. Stand er gleichzeitig auf der letzten Zeile, wird der Bildschirminhalt um eine Zeile nach oben verschoben.

Der Code $0C hat die gleiche Wirkung wie der Code $09 und wurde aus Kompatibilitätsgründen berücksichtigt.

Code $0D: CTRL-M

Der Cursor wird auf die linke Seite der Zeile gestellt, in der er sich vor dem Aufruf befunden hat. Der Befehl wird auch CARRIAGE RETURN oder kurz CR genannt.

Code $16: CTRL-V

Der Cursor wird eine Zeile nach unten transportiert. War er auf der untersten Zeile, wird er nicht weiter nach unten gesetzt. Es erfolgt kein Scrollvorgang.

Code $1A: CTRL-Z

Die Wirkung ist die gleiche wie der Aufruf des CLRSCREEN-Kommandos. Der Bildschirm wird gelöscht, der Cursor wandert in die linke obere Ecke.

Code $1E: CTRL-^

Der Cursor wandert in die linke obere Ecke. Der Bildschirm bleibt im übrigen unverändert. Der Befehl wird auch HOME genannt.

Code $1B: ESC-Befehle

Bei den ESC-Befehlen werden mehrere Codes übertragen. Auch sie dienen der Cursor- und Bildschirmsteuerung.

Es muß erst die ESC-Taste gedrückt werden und danach die Zeichensequenz. Das nachfolgende Zeichen muß als Großbuchstabe eingegeben werden.

Code $1B $23 $yy $xx: ESC =..

Damit kann der Cursor absolut positioniert werden. Die neue X-/Y-Position wird in Zeicheneinheiten übergeben. In der X-Richtung gibt es die Werte 0 bis 79 und in der Y-Richtung die Werte 0 bis 23. Für die Ausgabe wird jetzt aber noch zuvor jeweils der Wert 32 addiert. Damit ergeben sich lesbare ASCII-Zeichen für die Koordinaten.

Beispiel:

ESC =!A

positioniert den Cursor in Spalte 33 (X-Position) und in Reihe 1 (Y-Position). X = 0, Y = 0 ist die linke obere Ecke des Bildschirms.

Code $1B $24: ESC $

Der Zeichensatz wird auf Deutsch umgestellt. Seit Version 6.0 wird die Zeichensatzanpassung aber nicht mehr von dieser Routine durchgeführt. Das ermöglicht immer die gemischte Ausgabe mit deutschem und amerikanischem Zeichensatz. Die Auswertung der eingegebenen Zeichen muß von den Anwenderprogrammen selbst durchgeführt werden, da z B. die Routine CI keine Zeichensatzanpassung durchführt. Die Auswirkung des Befehls wird nur dann erkennbar, wenn z. B. nach der Ausführung der Editor aufgerufen wird. Dann ist der Zeichensatz auf Deutsch gestellt.

Code $1B $25: ESC %

Der Zeichensatz wird wieder auf Amerikanisch umgestellt. Hier gibt es die gleichen Änderungen gegenüber Version 4.3 wie beim Befehl ESC $.

Code $1B $45: ESC E

Eine leere Zeile wird eingefügt. Dabei wandert alles von der aktuellen Cursorstelle um eine Zeile nach unten. Die letzte Zeile geht verloren.
Dieser Befehl darf nicht verwendet werden, wenn der HARDSCROLL eingeschaltet ist.

Code $1B $4F: ESC O

Mit dieser Funktion ist ein Ausdruck des momentanen Bildschirminhalts möglich. Dabei werden die Zeichen im Gegensatz zur Funktion ESC-P nur aus dem Speicher direkt an den Drucker gegeben. Es erfolgt keine Druckersteuerung.

Code $1B $50: ESC P

Auch diese Funktion druckt den Bildschirminhalt aus. Allerdings wird zuerst der Drucker mit den Werten initialisiert, die im DRUCKMENÜ eingestellt sind. Das entspricht dem Befehl O des DRUCKMENÜS (siehe auch Kapitel 3.4.3.). Allerdings wird die eingestellte Seitenlänge nicht berücksichtigt, wenn eine Seite weniger als 24 Zeilen hat. Am Ende wird ein Seitenvorschub durchgeführt. Außerdem wird der Zeichensatz bei der Ausgabe umgeschaltet, wenn der NDR-Zeichensatz in der DRUCKER-STEUERUNG des Grundprogramms eingestellt ist.

Code $1B $51: ESC Q

Ein Leerzeichen wird an der aktuellen Cursorstelle eingefügt. Alle Zeichen wandern auf der betreffenden Zeile um eins nach rechts. Wenn ganz rechts ein Zeichen steht, geht es verloren.

Code $1B $52: ESC R

Die Zeile, in der sich der Cursor befindet, wird gelöscht. Alle Zeilen darunter werden um eine Zeile nach oben geschoben.
Der Befehl darf nicht im HARDSCROLL-Modus benutzt werden.

Code $1B $54: ESC T

Alle Zeichen von der Cursorstelle bis zum Ende der Zeile werden gelöscht.

Code $1B $57: ESC W

Das Zeichen an der Cursorstelle wird gelöscht. Der Rest der Zeile wandert um eine Zeichenstelle nach links.

Code $1B $59: ESC Y

Alle Zeichen, beginnend bei der Cursorstelle, werden bis zum Ende der Bildseite gelöscht.
Auch dieser Befehl darf nicht im HARDSCROLL-Modus benutzt werden.

Code $1B '1' - '5': ESC 1 - ESC 5

Der Befehl ist seit Version 6.0 vorhanden. Damit kann der HARDSCROLL-Modus angewählt werden. Er ist aber nur mit der neuen GDPHS verfügbar, was auf der KEY-Karte eingestellt werden muß. Der Befehl ESC 1 setzt den HARDSCROLL auf den langsamsten Wert. ESC 5 wählt die schnellste Ausgabegeschwindigkeit.

Code $1b '0': ESC 0

Dieser Befehl ist das Gegenstück zu ESC 1 - ESC 5. Durch ihn wird der SOFTSCROLL eingestellt. Er baut den Bildschirm vollständig neu auf, wobei der Screen danach wieder in der normalen Stellung steht (Scroll-Wert auf Null). Da der Befehl den Bildschirm neu aufbauen muß, benötigt er ein wenig Zeit. Deshalb sollte nicht dauernd bei der Ausgabe hin- und hergeschaltet werden.

TRAP-Nummer	: 22		
Befehlsname	: **LO**		
Befehlsgruppe	: Zeichenausgabe.		
Zeichenausgabe	: Zeichen über Centronics an Drucker ausgeben.		
Eingaberegister	: D0.B = Zeichen, das ausgegeben werden soll.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Umlenkung auf serielle Schnittstelle möglich.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: CLRSCREEN (20)	CO (21)	SIZE (25)
	CO2 (33)	CRT (49)	LST (50)
	USR (51)	NIL (52)	SETPASS (55)
	CURSEIN (61)	CURSAUS (62)	CHAR (63)
	CURON (81)	CUROFF (82)	CRLF (99)
	GETLINE (100)	GETCURXY (101)	SETCURXY (102)
	LSTS (117)	CO2SER (129)	

Das Zeichen, das in Register D0.B steht, wird über die Druckerschnittstelle ausgegeben. Das geschieht über die CENT-Baugruppe.

Seit Version 6.0 kann die Ausgabe auf die serielle Karte umgelenkt werden. Die Umlenkung ist durch den Befehl SER möglich. Zusammen mit der LO-Routine wird auch LSTS umgelenkt, da die beiden Befehle wie CI und CSTS zusammengehören.

Einfaches Beispiel:

Das Zeichen A wird zum Drucker geschickt. Dabei erfolgt beim Drucker meist keine Ausgabe, da der Drucker immer nur vollständige Zeilen ausgibt. Deshalb muß normalerweise immer noch $0D und $0A hinter den auszudruckenden Zeichen hergeschickt werden.

```
START:
        MOVEQ     #'A',D0          * Zeichen A ausgeben.
        MOVEQ     #!LO,D7
        TRAP      #1
        RTS
```

Komplexeres Beispiel:

Der aktuelle Editortext wird auf dem Drucker ausgegeben. Dabei wird keine Zeichensatzumschaltung vorgenommen. Soll das geschehen, muß mit Hilfe der DRUCKERSTEUERUNG ausgedruckt werden.

```
START:
        MOVEQ     #!CIINIT2,D7     * Vorbereitung für CI2.
        TRAP      #1
SCHLEIFE:
        MOVEQ     #!CI2,D7         * Ein Zeichen aus RAM lesen (Editortext).
        TRAP      #1
        BCS.S     ENDE             * Wenn Null, dann Ende.
        MOVEQ     #!LO,D7          * Auf dem Drucker ausgeben.
        TRAP      #1
        BRA.S     SCHLEIFE         * Wiederholen.
ENDE:
        RTS
```

Bemerkung:

Mit dem Programm ist es möglich, einen Text, der z.B. mit dem Editor eingegeben wurde, auf dem Drucker auszugeben. Wenn man das Programm startet, druckt es sich zunächst selbst aus.

Will man einen anderen Text ausgeben, muß man entweder einen neuen Text mit dem Editor eingeben oder mit der Funktion TEXTSTART (OPTIONEN-Menü) einen neuen Textanfang einstellen.

TRAP-Nummer	: 23
Befehlsname	: SIN
Befehlsgruppe	: Berechnungen.
Kurzbeschreibung	: Berechnet den Sinus * 256 eines Wertes.
Eingaberegister	: D0.W = Wert, für den Sinus berechnet werden soll.
Ausgaberegister	: D0.W = Sinus des Wertes. Flags.
Zerstörte Register	: Keine.
Ab Version	: 3.1
Änderungen zu 4.3	: Nein.
Änderungen zu 6.1	: Nein.
Siehe auch	: COS (24) WERT (29) MULS32 (72) DIVS32 (73) ADJ360 (83) RND (96)

Damit steht die SIN-Funktion zur Verfügung. Der Sinus wird mit Hilfe einer Tabelle ermittelt, was sehr schnell geht. Damit man mit einem Integer-Bereich auskommt, ist der Sinuswert mit 256 multipliziert und die Nachkommastellen sind gerundet. Der Bereich ist somit:

+/- 256.

Der Parameter für die Berechnung wird im Register D0.W als vorzeichenbehaftete Größe in Grad eingegeben.

Anschließend erhält man das Ergebnis im Register D0.W, ebenfalls als vorzeichenbehaftete Größe im Bereich von -256 bis +256.

Intern wird die Routine SIN u. a. dazu verwendet, um die Berechnungen für die Schildkrötengraphik durchzuführen.

Einfaches Beispiel:

Das Programm berechnet den Sinus von 10 Grad.

```
START:
        MOVEQ   #10,D0          * 10 Grad.
        MOVEQ   #!SIN,D7        * Sinus berechnen.
        TRAP    #1
        RTS                     * Ergebnis ist 256 * sin(10).
```

Da der Sinus sehr schnell berechnet wird, ist die Anwendung vorwiegend für Echtzeitaufgaben wie Rotation von Körpern oder Fouriersynthese bestimmt.

Komplexeres Beispiel:

Die Sinuskurve wird als Flächengrafik ausgegeben.

```
START:
        CLR     D1              * X-Koordinate und Winkel.
SCHLEIFE:
        MOVE    #128,D2         * Y = 128.
        MOVEQ   #!MOVETO,D7
        TRAP    #1              * Startposition.
        MOVE    D1,D0           * Dann Y-Wert ermitteln,
        MOVEQ   #!SIN,D7        * dazu Sinus berechnen.
        TRAP    #1
        ASR     #2,D0           * Durch 4 teilen, ergibt Bereich von -64 bis 64.
        ADD     #128,D0         * Basis verschieben.
        MOVE    D0,D2           * Neuer Wert.
        MOVEQ   #!DRAWTO,D7
        TRAP    #1              * Linie ausgeben.
        ADDQ    #1,D1           * X von 0 bis 511.
        CMP     #512,D1         * Wenn 512 erreicht, dann
        BNE.S   SCHLEIFE        * beenden, sonst zurück.
        RTS
```

Hier wird eine Fouriersynthese durchgeführt nach der Formel:

$$y = k * (\sin(x) + \sin(3*x) / 3 + \sin(5*x) / 5).$$

Die Kurve wird dann ausgegeben.

```
START:
        CLR     D1          * X-Koordinate.
        CLR     D3          * Startwinkel.
SCHLEIFE:
        MOVE    #128,D2     * Y = 128.
        MOVEQ   #!MOVETO,D7
        TRAP    #1          * Startposition.
        CLR     D2          * Vorbereiten für Summe.
        MOVEQ   #1,D4       * Term 1
        BSR.S   SINTEIL     * berechnen.
        MOVEQ   #3,D4       * Term 2
        BSR.S   SINTEIL     * berechnen.
        MOVEQ   #5,D4       * Term 3
        BSR.S   SINTEIL     * berechnen.
        MOVEQ   #7,D4       * Term 4
        BSR.S   SINTEIL     * berechnen, dann Stop der Reihe.
        ASR     #2,D2       * Bereich anpassen (durch 4 teilen).
        ADD     #128,D2     * Basislinie.
        MOVEQ   #!DRAWTO,D7
        TRAP    #1          * Linie ausgeben.
        ADDQ    #2,D3       * In 2 Grad-Schritten.
        ADDQ    #1,D1       * X von 0 bis 511.
        CMP     #512,D1     * Wenn 512 erreicht, dann
        BNE.S   SCHLEIFE    * beenden, sonst zurück.
        RTS

SINTEIL:
        MOVE    D3,D0
        MULU    D4,D0       * sin(k * x).
        MOVEQ   #!SIN,D7
        TRAP    #1
        EXT.L   D0          * Wegen DIVS.
        DIVS    D4,D0       * sin(k * x) / k.
        ADD     D0,D2
        RTS
```

TRAP-Nummer	: **24**		
Befehlsname	: **COS**		
Befehlsgruppe	: Berechnungen.		
Kurzbeschreibung	: Berechnet den Cosinus * 256 eines Wertes.		
Eingaberegister	: D0.W = Wert, für den Cosinus berechnet werden soll.		
Ausgaberegister	: D0.W = Cosinus des Wertes.		
	Flags.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SIN (23)	WERT (29)	MULS32 (72)
	DIVS32 (73)	ADJ360 (83)	RND (96)

Für die COS-Routine gilt das Gleiche wie schon bei SIN. Im Register D0.W wird der Wert des Parameters in Grad übergeben. Das Ergebnis steht nach dem Aufruf wieder im Register D0.W. Der Wert ist cos(x) * 256, die Nachkommastellen entfallen. Der Wert ist vorzeichenbehaftet und im Zweierkomplement dargestellt.

Einfaches Beispiel:

Der Cosinus von -10 Grad wird berechnet.

```
START:
        MOVEQ   #-10,D0         * Winkel = -10 Grad.
        MOVEQ   #!COS,D7
        TRAP    #1              * Berechnen von cos(-10) * 256.
        RTS
```

Kombiniertes Beispiel:

Es wird eine Linie vom Mittelpunkt eines Kreises zu den Kreisrandpunkten gezogen. Dabei muß in 1 Grad-Schritten verfahren werden, da die Sinus- und Cosinus-Funktion nur mit Integerzahlen arbeitet. Der Kreis ist deshalb nicht voll ausgefüllt.

```
START:
        MOVE    #360-1,D3       * Phi = Winkelstart = 359.
SCHLEIFE:
        MOVE    #256,D1         * Bildmitte einstellen.
        MOVE    #128,D2         * Man beachte die
        MOVEQ   #!MOVETO,D7
        TRAP    #1              * unterschiedliche Auflösung.
        MOVE    D3,D0           * X = COS(phi) + 256.
        MOVEQ   #!COS,D7
        TRAP    #1
        ADD     #256,D0
        MOVE    D0,D1           * X-Koordinate.
        MOVE    D3,D0           * Y = SIN(phi) / 2 + 128.
        MOVEQ   #!SIN,D7
        TRAP    #1
        ASR     #1,D0           * Auf +/- 128 bringen.
        ADD     #128,D0
        MOVE    D0,D2           * Ergibt Y-Koordinate.
        MOVEQ   #!DRAWTO,D7
        TRAP    #1              * Verbindung zeichnen.
        DBRA    D3,SCHLEIFE     * Nächster Winkel (359 bis 0).
        RTS
```

Wenn man den Sinus genauer als 1 Grad benötigt, muß man die Zwischenpunkte approximieren. Dies wird z. B. im Grafik-Paket bei den Kreisen und Ellipsen so gemacht.

TRAP-Nummer	: **25**		
Befehlsname	: **SIZE**		
Befehlsgruppe	: Zeichenausgabe.		
Kurzbeschreibung	: Stellt Schriftgröße für CO2, CO und CHAR ein.		
Eingaberegister	: D0.B = Schriftgröße, die eingestellt werden soll.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Schaltet den HARDSCROLL aus, wenn die Y-Vergrößerung nicht 1 ist.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: CLRSCREEN (20)	CO (21)	LO (22)
	CO2 (33)	CRT (49)	LST (50)
	USR (51)	NIL (52)	SETPASS (55)
	CURSEIN (61)	CURSAUS (62)	CHAR (63)
	CURON (81)	CUROFF (82)	CRLF (99)
	GETLINE (100)	GETCURXY (101)	SETCURXY (102)
	LSTS (117)	CO2SER (129)	

Diese Routine setzt die Ausgabegröße für CO, CO2 und CHAR. Dabei ist es nicht möglich, unterschiedliche Größen zu mischen. Im Register D0.B wird die Zeichengröße übergeben.
Nachdem man eine neue Größe gesetzt hat, sollte man den Bildschirm löschen. Die Codierung der Größe ist dabei genau so wie beim WRITE-Befehl. Zwei Codes sind dabei besonders interessant:

Mit $11 wird auf 80 Zeichen pro Zeile und 24 Zeilen umgeschaltet, mit $21 auf 40 Zeichen pro Zeilen ebenfalls mit 24 Zeilen. Alle Zeichen, die nach der 40 Spalte ausgegeben werden, bleiben dann unsichtbar.

Die beiden Codes werden auch im OPTIONEN-Menü eingestellt, wenn man die Zeichengröße festsetzt. Andere Zeichengrößen sind prinzipiell möglich, jedoch kann es sein, daß dann auf dem Monitor nur ein kleiner Ausschnitt dargestellt wird.

Ist der HARDSCROLL eingeschaltet und wird dann eine Schriftgröße eingestellt, die eine Y-Vergrößerung ungleich 1 enthält, wird der HARDSCROLL ausgeschaltet.

Einfache Beispiele:

Auf 80 Zeichen pro Zeile umschalten und den Bildschirm löschen.

```
START:
        MOVEQ     #$11,D0          * Umschalten auf 80 Zeichen/Zeile.
        MOVEQ     #!SIZE,D7        * Erst neue Größe einstellen.
        TRAP      #1
        MOVEQ     #!CLRSCREEN,D7   * Dann Bildschirm löschen.
        TRAP      #1
        RTS
```

Auf 40 Zeichen pro Zeile umschalten und den Bildschirm löschen.

```
START:
        MOVEQ     #$21,D0          * Umschalten auf 40 Zeichen/Zeile.
        MOVEQ     #!SIZE,D7
        TRAP      #1               * Erst neue Größe einstellen.
        MOVEQ     #!CLRSCREEN,D7   * Dann Bildschirm löschen.
        TRAP      #1
        RTS
```

Es wird auf 30 Zeichen pro Zeile umgeschaltet, wobei die Zeichenhöhe doppelt so hoch ist wie bei 80 Zeichen.

```
START:                             * Doppelter Zeilenabstand und
        MOVEQ     #$32,D0          * umschalten auf 30 Zeichen/Zeile.
        MOVEQ     #!SIZE,D7        * Erst neue Größe einstellen.
        TRAP      #1
        MOVEQ     #!CLRSCREEN,D7   * Dann Bildschirm löschen.
        TRAP      #1
        RTS
```

Wenn man nach Aufruf dieses Programms wieder in den Editor zurückkommt, erscheint ein großer Cursor auf dem Bildschirm. Auch der Text ist gedehnt. Man sieht aber nur einen Bildausschnitt. Es erscheinen auf dem Bildschirm nur ca. 12 Zeilen. Die restlichen Zeilen sind weiter unten und unsichtbar. Will man sie sehen, kann man mit CTRL-Z den Bildschirm nach oben schieben. Es ist auch möglich, den Cursor in ein unsichtbares Gebiet zu schieben. Soll er wieder erscheinen, gibt man z. B. CTRL-QR ein, und der Cursor steht oben links im Editorfeld.

Diese Darstellungsart empfiehlt sich, wenn man mehreren Personen gleichzeitig Einzelheiten bei der Eingabe zeigen will.

Man kann die Textgröße einfach zurückstellen, in dem man sie im OPTIONEN-Menü wieder auf 80 oder 40 Zeichen pro Zeile einstellt.

TRAP-Nummer	:	**26**
Befehlsname	:	**CMD**
Befehlsgruppe	:	Grafik.
Kurzbeschreibung	:	Befehl direkt an GDP.
Eingaberegister	:	D0.B = Befehl, der ausgeführt werden soll.
Ausgaberegister	:	Keine.
Zerstörte Register	:	Keine.
Ab Version	:	3.1
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	MOVETO (8), DRAWTO (9), CLR (16), CLPG (17), WAIT (18), NEWPAGE (27), SETFLIP (34), SETPEN (37), ERAPEN (38), CMDPRINT (40), AUTOFLIP (60), SETXOR (77), GETXOR (78), SETCOLOR (79), GETCOLOR (80), GETXY (103), HARDCOPY (125), GRAFIK (126), GDPVERS (127)

Damit kann man einen Befehl direkt an den Graphik-Prozessor ausgeben. Der Befehl steht dazu zuvor im Register D0.B. Mit der Ausführung wird so lange gewartet, bis der GDP bereit ist. Erst dann wird der Befehl an den GDP übergeben.

Eine Liste aller Befehle findet man bei der GDP-Beschreibung des EF9366. Hier ein paar besonders wichtige Codes:

$0	Schreiben.
$1	Löschen.
$2	Stift senken.
$3	Stift heben.
$4	Bildschirm löschen.
$5	X- und Y-Register auf 0 setzen.
$6	Wie 4 und 5 zusammen.
$7	Alle Register auf 0 (CSIZE auf $11) und wie 6.
$8	Lichtblitz für Lichtgriffel und abtasten.
$9	Lichtgriffel abtasten ohne Lichtblitz.
$A	5 x 8 Block zeichnen.
$B	4 x 4 Block zeichnen.
$C	Bildschirm hell oder dunkel setzen.
$D	X-Register auf 0 setzen.
$E	Y-Register auf 0 setzen.
$F	DMA-Zugriff (für Hardcopy-Funktion mit GDPHS).

Die Codes $10 bis $1F sind für Vektoren gedacht, die Codes $80 bis $FF für Kurzvektoren. Die Vektoren sind aber über die Befehle DRAWTO oder FIGUR leichter zu bedienen.

Die Codes $20 bis $7F sind die Darstellung der lesbaren ASCII-Zeichen. Damit kann man Texte sehr schnell auf dem Bildschirm ausgeben.

Einfaches Beispiel:

Der gesamte Bildschirm wird weiß eingefärbt.

```
START:
        MOVEQ     #$C,D0          * Bildschirm weiß stellen.
        MOVEQ     #!CMD,D7        * Ausführen.
        TRAP      #1
        RTS
```

Der Bildschirm leuchtet nach Ausführung des Programms hell. Damit kann man inverse Darstellung vorbereiten, wenn man anschließend Zeichen oder Graphik im Löschmode (CMD mit D0 = 1 oder ERAPEN) ausgibt.

Komplexeres Beispiel:

Ein Text wird über den Bildschirm bewegt.

```
START:
        MOVEQ       #!WAIT,D7
        TRAP        #1              * Init Textgröße.
        MOVE        #!SYSTEM,D7
        TRAP        #1              * System-Informationen lesen.
        AND         #7,D0
        MULS        #$FF73,D0       * GDP-Register 3 * CPU.
        MOVEA.L     D0,A0
        MOVE.B      #$22,(A0)       * In GDP-Register 3.
        CLR         D1              * X = 0.
        CLR         D2              * Y = 0.
SCHLEIFE:
        MOVEQ       #!MOVETO,D7     * Position einstellen.
        TRAP        #1
        MOVEQ       #0,D0           * Schreiben.
        BSR.S       SCHREIBE        * Text ausgeben.
SYNC:
        MOVEQ       #!SYNC,D7       * 20 Millisekunden warten,
        TRAP        #1
        BEQ.S       SYNC            * sonst zu schnell.
        MOVEQ       #!MOVETO,D7     * Wieder zurück.
        TRAP        #1
        MOVEQ       #1,D0           * Dann löschen.
        BSR.S       SCHREIBE
        ADDQ        #2,D1           * Diagonal
        ADDQ        #1,D2           * bewegen.
        CMP         #256,D2         * Wenn größer als 256.
        BMI.S       SCHLEIFE        * dann stoppen.
        RTS

SCHREIBE:
        MOVEQ       #!CMD,D7        * Befehl ausführen.
        TRAP        #1
        LEA         TEXT(PC),A0     * Text ausgeben.
SCHRLP:
        MOVE.B      (A0)+,D0        * Zeichen holen.
        BEQ.S       ENDE            * Null, dann Ende
        MOVEQ       #!CMD,D7        * und ausgeben.
        TRAP        #1
        BRA.S       SCHRLP          * Wiederholen.
ENDE:
        RTS
TEXT:
        DC.B        'Bewegter Text',0
```

Mit dem Befehl SYNC wird erreicht, daß der Text ohne Flimmern über den Bildschirm wandert. Zur Probe kann man die drei Zeilen mit dem SYNC-Befehl weglassen.

Soll die Schrift schneller bewegt werden, addiert man einfach einen größeren Wert auf D1 und D2. Man kann den Text auch auf zwei Bildschirmseiten darstellen und dabei ohne SYNC arbeiten. Dann wird die Schrift sehr schnell bewegt, ohne daß Bildstörungen auftreten.

TRAP-Nummer	: **27**			
Befehlsname	: **NEWPAGE**			
Befehlsgruppe	: Grafik.			
Kurzbeschreibung	: Schreibseite und Leseseite neu einstellen.			
Eingaberegister	: D0.B = Schreibseite (0 - 3). D1.B = Leseseite (0-3).			
Ausgaberegister	: D0.B = Wert, der direkt an GDP-Port $60 gegeben wird.			
Zerstörte Register	: Keine.			
Ab Version	: 3.1			
Änderungen zu 4.3	: Nein.			
Änderungen zu 6.1	: Nein.			
Siehe auch	: MOVETO (8)	DRAWTO (9)	CLR (16)	
	CLPG (17)	WAIT (18)	CMD (26)	
	SETFLIP (34)	SETPEN (37)	ERAPEN (38)	
	CMDPRINT (40)	AUTOFLIP (60)	SETXOR (77)	
	GETXOR (78)	SETCOLOR (79)	GETCOLOR (80)	
	GETXY (103)	HARDCOPY (125)	GRAFIK (126)	
	GDPVERS (127)			

Die GDP-Baugruppe besitzt vier Bildseiten mit 512 x 256 Punkten. Für die Auswahl der Bildseite gibt es den Befehl NEWPAGE. Im Register D0.B steht die Seitennummer der Schreibseite und im Register D1.B die Seitennummer der Leseseite. Alle Graphik-Befehle werden auf der Schreibseite ausgeführt. Das Bild auf dem Bildschirm stammt von der Leseseite. Dadurch ist es möglich, auf einer unsichtbaren Seite Bilder zu erstellen und sich gleichzeitig eine andere Seite anzusehen. Mit dem Befehl NEWPAGE kann man also flimmerfreie Graphiken zu erzeugen.

Als Wert für die Schreib- und Leseseite kann man eine Zahl von 0 bis 3 wählen.

Wenn man die Schildkrötengraphik verwendet, wird die Schildkröte selbst immer auf der Seite 1 gezeichnet und die Graphik normalerweise auf der Seite 0.

Einfaches Beispiel:

Schreib- und Leseseite werden auf Null gesetzt.

```
START:
        CLR     D0              * Schreibseite = 0.
        CLR     D1              * Leseseite = 0.
        MOVEQ   #!NEWPAGE,D7    * Seite anwählen.
        TRAP    #1
        RTS
```

Komplexere Beispiele:

Es wird ein Kreis dargestellt, der immer größer wird.

```
START:
        MOVEQ   #!FIRSTTIME,D7  * Schildkröte initialisieren.
        TRAP    #1
        MOVEQ   #!HIDE,D7
        TRAP    #1              * Dann Flip ausschalten.
        MOVEQ   #1,D4           * Schreibseite = 1.
        MOVEQ   #0,D5           * Leseseite = 0.
        MOVEQ   #1,D3           * Info für Bild.
SCHLEIFE:
        MOVE    D4,D0           * Schreibseite.
        MOVE    D5,D1           * Leseseite.
        MOVEQ   #!NEWPAGE,D7    * Seite anwählen.
        TRAP    #1
        MOVEQ   #!CLPG,D7       * Seite unsichtbar löschen.
        TRAP    #1
        BSR.S   BILD            * Bild aufbauen.
        EXG     D4,D5           * Schreibseite und Leseseite austauschen.
        CMP     #400,D3         * Maximaler Wert.
        BCS.S   SCHLEIFE        * Wieder zurück.
        RTS
BILD:
        MOVEQ   #36-1,D6        * Kreis zeichnen,
```

```
BILDLP:
        MOVE        D3,D0           * der größer wird.
        MOVEQ       #!SCHR16TEL,D7
        TRAP        #1
        MOVEQ       #10,D0          * 10 Grad drehen.
        MOVEQ       #!DREHE,D7
        TRAP        #1
        DBRA        D6,BILDLP       * Bis beendet.
        ADDQ        #5,D3           * Neuer Wert.
        RTS
```

Mit den ersten beiden Befehlen FIRSTTIME und HIDE wird erreicht, daß die Schildkröte initialisiert wird, dann wird der Zeiger mit dem Befehl HIDE abgestellt. Erst danach kann man die zweite Bildseite mitverwenden.
Eine andere Möglichkeit ist es auch, das alte Bild zu löschen, indem die Graphik mit Hilfe des Befehls ERAPEN neu gezeichnet wird.

Es wird ein Kreis vergrößert und außerdem im Kreis auf dem Bildschirm bewegt.

```
START:
        MOVEQ       #!FIRSTTIME,D7  * Schildkröte anschalten.
        TRAP        #1
        MOVEQ       #!HIDE,D7       * Schildkröte nicht zeigen.
        TRAP        #1
        MOVEQ       #1,D4           * Schreibseite = 1.
        MOVEQ       #0,D5           * Leseseite = 0.
        MOVEQ       #1,D3           * Info für Bild.
        MOVE        D4,D0           * Schreibseite und
        MOVE        D5,D1           * Leseseite
        MOVEQ       #!NEWPAGE,D7    * neu einstellen.
        TRAP        #1
        MOVEQ       #!CLPG,D7       * Unsichtbar löschen.
        TRAP        #1
SCHLEIFE:
        MOVE        D4,D0
        MOVE        D5,D1
        MOVEQ       #!NEWPAGE,D7    * Seite einstellen.
        TRAP        #1
        MOVEQ       #!ERAPEN,D7
        TRAP        #1
        BSR.S       BILD            * Altes Bild löschen.
        MOVEQ       #12,D0          * Dann drehen.
        MOVEQ       #!DREHE,D7
        TRAP        #1
        ADDQ        #3,D3           * Für neues Bild.
        MOVEQ       #!SETPEN,D7
        TRAP        #1
        BSR.S       BILD            * Neues Bild zeichnen.
        MOVEQ       #-6,D0
        MOVEQ       #!DREHE,D7      * Korrektur für nächstes Bild.
        TRAP        #1
        SUBQ        #6,D3
        EXG         D4,D5           * Schreibseite und Leseseite austauschen.
        CMP         #400,D3         * Maximaler Wert.
        BMI.S       SCHLEIFE        * Wiederholen.
        MOVEQ       #!GRAPOFF,D7    * Verhindert Auftauchen der Schildkröte.
        TRAP        #1
        RTS                         * Ende.
BILD:
        MOVEQ       #36-1,D6        * Kreis zeichnen,
BILDLP:
        MOVE        D3,D0           * der größer wird.
        MOVEQ       #!SCHR16TEL,D7
        TRAP        #1
        MOVEQ       #10,D0          * 10 Grad drehen.
        MOVEQ       #!DREHE,D7
        TRAP        #1
        DBRA        D6,BILDLP       * Bis beendet.
        ADDQ        #3,D3           * Neuer Wert.
        RTS
```

Mit dem NEWPAGE-Befehl kann man auf diese Weise einfach bewegte Graphik erstellen. Ein weiterer Befehl, der SYNC-Befehl, kann dazu verwendet werden, den Bildwechsel mit der Graphikerstellung zu synchronisieren. Man vermeidet so störende Bildschnitte.

TRAP-Nummer	: 28
Befehlsname	: SYNC
Befehlsgruppe	: Synchronisation.
Kurzbeschreibung	: Fragt Vertikal-Synchronimpuls der GDP ab.
Eingaberegister	: Keine.
Ausgaberegister	: D0.W = Flag, ob SYNC aufgetreten oder nicht. Flags.
Zerstörte Register	: Keine.
Ab Version	: 3.1
Änderungen zu 4.3	: Nein.
Änderungen zu 6.1	: Nein.
Siehe auch	: DELAY (35)

Mit dem Befehl kann man den Vertikal-Synchronisationsimpuls des GDP abzufragen. Nach jedem Durchgang wird $FFFF im Register D0 übergeben, sonst der Wert 0. Damit kann man Abläufe mit dem Bildwechsel synchronisieren. Der SYNC-Befehl liefert aber nur einmal den Wert $FFFF, auch wenn man den V-Sync-Impuls sehr schnell hintereinander abfragt und sich noch in der vertikalen Austastlücke befindet. Diese Konstruktion des SYNC-Befehls ist erforderlich, um Zeitsynchronisationen zu ermöglichen. Bei einer Dauerabfrage liefert das Signal alle 20 Millisekunden den Wert $FFFF. Damit kann man quarzgenaue Messungen durchführen; denn auf der GDP-Baugruppe befindet sich ein Quarz, der auch für diesen Takt verantwortlich ist.

Einfaches Beispiel:

Das Programm wartet auf den SYNC-Impuls

```
START:
        MOVEQ   #!SYNC,D7       * Vertikal-Synchronisationsimpuls abfragen.
        TRAP    #1
        BEQ.S   START           * Wiederholen, bis SYNC da ist.
        RTS
```

Komplexere Beispiele:

Eine Linie wird über den Bildschirm bewegt.

```
START:
        CLR     D1              * X = 0.
SCHLEIFE:
        MOVEQ   #!SETPEN,D7     * Auf Schreiben.
        TRAP    #1
        CLR     D2              * Y = 0.
        MOVEQ   #!MOVETO,D7     * Anfang der Linie.
        TRAP    #1
        MOVE    #255,D2         * Y = 255.
        MOVEQ   #!DRAWTO,D7     * Ende der Linie.
        TRAP    #1
WARTE:
        MOVEQ   #!SYNC,D7       * Auf V-Sync warten.
        TRAP    #1
        BEQ.S   WARTE
        MOVEQ   #!ERAPEN,D7     * Auf Löschen.
        TRAP    #1
        CLR     D2              * Y = 0.
        MOVEQ   #!DRAWTO,D7     * Linie löschen.
        TRAP    #1
        ADDQ    #1,D1           * Nächste X-Position.
        CMP     #512,D1
        BNE.S   SCHLEIFE        * Wenn rechter Rand erreicht, dann Ende.
        RTS
```

Wenn man den SYNC-Befehl wegläßt, sieht man den gleichen Vorgang, jedoch ist die Linie unterbrochen und wandert viel schneller über den Bildschirm. Die Unterbrechungen erklären sich daraus, daß man den Lösch- und Schreibvorgang überlagert sieht.

Es wird eine kleine Zeitmessung in Minuten und Sekunden durchgeführt. In diesem Beispiel könnte auch der Befehl DELAY verwendet werden.

```
START:
        MOVEQ       #0,D4               * Zeitzähler.
SCHLEIFE:
        MOVEQ       #50-1,D3            * 50 * 20 Millisekunden.
WARTE:
        MOVEQ       #!SYNC,D7           * Messen.
        TRAP        #1
        BEQ.S       WARTE               * Bis 20 Millisekunden.
        DBRA        D3,WARTE            * Bis 1 Sekunde.
        ADDQ.L      #1,D4               * Sekundenzähler erhöhen.
        LEA         BUFFER(PC),A0       * Ausgabe durchführen.
        MOVE.L      D4,D0
        DIVS        #60,D0              * Minuten.
        MOVE.L      D0,D1               * Merken.
        MOVEQ       #!PRINT4D,D7        * Dezimal ausgeben.
        TRAP        #1
        MOVE.B      #':',(A0)+          * Trennen.
        MOVE.L      D1,D0               * Dann Sekunden.
        SWAP        D0                  * Modulo-Rest bei DIVS.
        MOVEQ       #!PRINT4D,D7        * Auch dezimal.
        TRAP        #1
        MOVE.B      #' ',(A0)+          * Zum Löschen der ersten und
        MOVE.B      #' ',(A0)+          * und zweiten Stelle.
        CLR.B       (A0)                * Endekennung neu.
        LEA         BUFFER(PC),A0       * Und auf Bildschirm.
        MOVEQ       #$22,D0             * Schriftgröße.
        MOVEQ       #2,D1               * X = 2.
        MOVE        #128,D2             * Y = 128.
        MOVEQ       #!WRITE,D7
        TRAP        #1                  * Ausgabe.
        MOVEQ       #!CSTS,D7           * Taste abfragen.
        TRAP        #1
        BEQ.S       SCHLEIFE            * Immer wieder, bis Taste gedrückt.
        RTS
BUFFER:
        DS.B        10                  * Textbuffer.
```

TRAP-Nummer	: 29		
Befehlsname	: WERT		
Befehlsgruppe	: Berechnungen.		
Kurzbeschreibung	: Ganzzahligen Wert aus arithmetischem Ausdruck berechnen.		
Eingaberegister	: A0.L = Adresse des arithmetischen Ausdrucks.		
Ausgaberegister	: D0.L = Errechneter Wert.		
	D1.L = Fehlerkennung.		
	A0.L = Zeigt hinter das letzte Zeichen oder Fehler.		
	Carry = Zeigt an, ob ein Fehler auftrat.		
Zerstörte Register	: D2/D3/A1-A3		
Ab Version	: 3.1		
Änderungen zu 4.3	: D1.L ist gültig.		
Änderungen zu 6.1	: Änderung bei MODULO-Operation.		
Siehe auch	: SIN (23)	COS (24)	MULS32 (72)
	DIVS32 (73)	ADJ360 (83)	RND (96)

Jetzt kommt ein sehr komplexer Befehl. Mit ihm ist es möglich, Ausdrücke zu berechnen, z. B. Symbole oder mathematische Angaben. Das Grundprogramm nutzt den Befehl u. a., um im Assembler beispielsweise mathematische Ausdrücke zu berechnen. Weiter dient der Befehl der Eingabe von Startadressen usw., also immer, wenn es gilt, einen Wert aus einer Formel oder einer Eingabe zu ermitteln. Es können auch Symbole verwendet werden, die etwa der Assembler definiert hat.

A0.L muß beim Aufruf auf einen Text zeigen, der einen arithmetischen Ausdruck enthält. Der Text muß mit einer Null oder einem Leerzeichen enden.

Nach dem Aufruf:

D0.L Ergebnis der Berechnung. Vorzeichenbehaftete 32 Bit-Zahl.
D1.L Fehlercode.
A0.L Zeigt hinter den Ausdruck bzw. hinter den Fehler, wenn einer aufgetreten ist.

Für den Fehlercode in D1.L gilt: Für das Carry-Flag gilt:

0	Syntax-Fehler aufgetreten.	1	Gesetzt.
1	Bytewert geladen.	0	Zurückgesetzt.
2	Wortwert geladen.	0	Zurückgesetzt.
3	Langwortwert geladen.	0	Zurückgesetzt.
4	Reserviert.		
5	Symbolaufruf ohne Symboldefiniton.	0	Zurückgesetzt.

Als Operationen sind zulässig:

+	Addition,
-	Subtraktion,
*	Multiplikation,
/	Division,
\	Modulo-Division - das Vorzeichen des Ergebnisses entspricht dem Vorzeichen des Divisors -,
()	Klammerrechnung,
!	Oder-Verknüpfung,
&	Und-Verknüpfung.

Weiter Bedeuten:

*	Programmstand des Assemblers,
?	Aktuelle Textanfangsadresse,
'...'	Text - wird als ASCII-Wert(e) geladen -,
~	Nicht-Operation,
-	Vorzeichen einer Zahl,
$nnnnnnnn	sedezimale Zahl,
dddddddd	dezimale Zahl,
%bbbbbbb	binäre Zahl,
@symbol	Adresse eines Symbols,
!symbol	Index eines Symbols - für TRAP-Befehl -.

Alle Operationen werden mit einer Genauigkeit von 32 Bit durchgeführt.

Die nachfolgenden Zusätze können hinter einem Ausdruck (ganz hinten) stehen.

.B/.S	Bytegröße markieren.
.W	Wortgröße markieren.
.L	Langwortgröße markieren.

Ohne Angabe wird die Größe durch Testen des Wertes in D0.L ermittelt. Eine negative Zahl ist immer ein Langwort. Wird eine Größe angegeben, wird nur der Wert in D1.L entsprechend eingestellt, auch wenn das Ergebnis nicht der Eingabe entspricht.

Einfaches Beispiel:

Es wird der Ausdruck 23 * 3 / 4 berechnet.

```
START:
        LEA     TEXT(PC),A0     * Adresse des Textes laden.
        MOVEQ   #!WERT,D7       * Wert berechnen.
        TRAP    #1
        RTS                     * Ergebnis in D0.L.
TEXT:
        DC.B    '23*3/4',0
```

Das Beispiel liefert den Wert sedezimal $11 oder dezimal 17. Man kann das Ergebnis überprüfen, wenn man das Programm im Einzelschritt nachvollzieht.

Komplexeres Beispiel:

Es kann ein Ausdruck eingegeben werden, der berechnet und ausgegeben wird. RETURN beendet das Programm.

```
START:
        LEA     BUFFER(PC),A0   * Text eingeben.
        MOVEQ   #$22,D0         * Größe.
        MOVEQ   #2,D1           * X = 2.
        MOVE    #128,D2         * Y = 128.
        MOVEQ   #30,D3          * Anzahl der Zeichen.
        MOVEQ   #!READ,D7
        TRAP    #1              * Zeichen lesen.
        BCS.S   ENDE            * Abbruch mit ESC
        TST.B   D4              * Ende, wenn nur RETURN gedrückt wird.
        BEQ.S   ENDE
        LEA     BUFFER(PC),A0   * Dann auswerten.
        MOVEQ   #!WERT,D7
        TRAP    #1              * Wert jetzt in D0.L.
        BCS.S   START           * Bei Fehler.
        CMP     #5,D1
        BEQ.S   START           * Wenn undefiniert.
        LEA     BUFFER(PC),A0   * Wieder Adresse laden.
        MOVE.B  #'$',(A0)+      * Kennung, daß sedezimal.
        MOVEQ   #!PRINT8X,D7
        TRAP    #1
        LEA     BUFFER(PC),A0   * Für Ausgabe.
        MOVEQ   #$22,D0         * Textgröße.
        MOVEQ   #2,D1           * X = 2.
        MOVE    #200,D2         * Y = 200.
        MOVEQ   #!WRITE,D7      * Auf den Bildschirm.
        TRAP    #1
        BRA.S   START           * Immer weiter.
ENDE:
        RTS
BUFFER:
        DS.B    31              * Textbuffer.
```

Man versuche dann einmal folgende Eingaben:

```
?
*
3*4
10/2
10\3
!SCHREITE
@SCHREITE
?+1
3*(4+5)
5!12
'A'
'AB'
%1010.
```

Die Ausgabe ist jetzt sedezimal. Möchte man eine dezimale Ausgabe haben, muß man die Befehle PRINT4D oder PRINTV8D verwenden. Dabei muß man aber den Buffer mit Leerzeichen auffüllen, sonst bleibt eventuell der Rest der vorhergehenden Ausgabe stehen, falls die neue Ausgabe weniger Stellen hat als die alte.

Achtung! Zwischen den einzelnen Eingaben für die Berechnung dürfen keine Leerzeichen stehen. Andernfalls kann z. B. der Assembler nicht unterscheiden, ob das Zeichen * einen Kommentar kennzeichnet oder für eine Multiplikation eingesetzt werden soll.

Beispiel:

```
MOVE        #3,$54          * 3 ist der Ladewert.
```

Wenn Leerzeichen erlaubt wären, ergäbe sich

```
$54 * 3
```

als Adresse für den Transport.

TRAP-Nummer	: **30**		
Befehlsname	: **ZUWEIS**		
Befehlsgruppe	: Symboltabelle.		
Kurzbeschreibung	: Symbol in Symboltabelle eintragen.		
Eingaben	: A0.L = Zeigt auf die Symboldefinition.		
Ausgaben	: A0.L = Zeigt hinter die Definition.		
	Carry = Symbol eingetragen / nicht eingetragen.		
	Wenn Eintragung erfolgreich:		
	D0.L = Wert des Symbols wie bei WERT.		
	D1.L = Definition des Symbols wie bei WERT.		
	A3.L = Adresse, an der Name eingetragen wurde.		
Zerstörte Register	: D2/D3/A1/A2 bei Fehler auch D0/D1/A3		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: PRTSYM (84)	SYMCLR (85)	GETSYM (86)
	GETNEXT (87)	PUTNEXT (88)	

Der Befehl ermöglicht es, einem Symbol einen Wert zuzuordnen. Das geschieht in Form einer Zuweisung:

SYMBOL := wert.

Auf der rechten Seite des Doppelpunkts steht ein beliebiger Ausdruck, wie er bei dem Befehl WERT möglich ist. Auf der linken Seite des Doppelpunkts steht ein Name, den man selbst gewählt hat. Diesem Namen wird dann der Wert zugeordnet.

Dabei muß der Name folgender Konvention entsprechen:

- Das erste Zeichen muß ein Buchstabe oder das Zeichen _ sein.
- Als weitere Zeichen sind Buchstaben, Zahlen und _ erlaubt.
- Die deutschen Sonderzeichen sind erlaubt.
- Im Namen dürfen sonst keine anderen Zeichen vorkommen.

Folgendes wird vom Grundprogramm durchgeführt:

- Kleinbuchstaben werden in Großbuchstaben gewandelt.
- Mehr als 8 Zeichen werden nicht berücksichtigt.

Eine Zuweisung muß immer ausgeführt werden. Geschieht das nicht, wird ein CARRY = 1 zurückgemeldet und die Zuweisung nicht ausgeführt.

A0.L muß auf die Definition zeigen. Wird die Zuweisung korrekt ausgeführt, steht der Wert des Symbols in D0.L und das Attribut in D1.L (wie D1.L bei dem Befehl WERT). Wird das Symbol nicht korrekt in die Symboltabelle eingetragen, werden unter Umständen die Inhalte der Register D0 und D1 zerstört. Bei korrekter Zuweisung zeigt A3.L zusätzlich auf die Adresse, in der der Symboleintrag steht. Wie ein solcher Eintrag aussieht, ist in Kapitel 5.4. näher beschrieben.

Einfaches Beispiel:

Dem Symbol NEU wird der Wert 75 zugeordnet.

```
START:
        LEA         TEXT(PC),A0          * Textadresse laden.
        MOVEQ       #!ZUWEIS,D7
        TRAP        #1                   * Symbol in Symboltabelle eintragen.
        RTS
TEXT:
        DC.B        'neu:=75',0
```

Nach Ausführung der Anweisung findet man in der Symboltabelle den Namen "NEU". Er besitzt den Wert 75 oder sedezimal $4B.

Komplexeres Beispiel:

Es werden Werte berechnet und Symbole in die Symboltabelle eingetragen. Erfolgt keine Wertzuweisung, wird der Wert berechnet und ausgegeben.

```
START:
        LEA         BUFFER(PC),A0       * Text eingeben.
        MOVEQ       #$22,D0             * Größe.
        MOVEQ       #2,D1               * X = 2.
        MOVE        #128,D2             * Y = 128.
        MOVEQ       #30,D3              * Maximale Anzahl der Zeichen.
        MOVEQ       #!READ,D7
        TRAP        #1                  * Ausdruck lesen.
        TST         D4
        BEQ.S       ENDE                * Ende, wenn nur RETURN.
        LEA         BUFFER(PC),A0       * Dann auswerten
        MOVEQ       #!ZUWEIS,D7         * und danach den Wert ausgeben.
        TRAP        #1
        BCC.S       WEITER              * OK, war Zuweisung.
        MOVEQ       #!WERT,D7           * Sonst mit WERT probieren.
        TRAP        #1
        BCS.S       START               * Fehler bei WERT.
        CMP         #5,D1
        BEQ.S       START               * Undefiniertes Symbol.
WEITER:
        LEA         BUFFER(PC),A0       * Wieder Adresse laden.
        MOVEQ       #!PRINT8X,D7
        TRAP        #1                  * Sedizimale Ausgabe.
        LEA         BUFFER(PC),A0       * Für Ausgabe.
        MOVEQ       #$22,D0             * Textgröße.
        MOVEQ       #2,D1               * X = 2.
        MOVE        #200,D2             * Y = 200.
        MOVEQ       #!WRITE,D7
        TRAP        #1                  * Ausgabe.
        BRA.S       START               * Wiederholen.
ENDE:
        RTS
BUFFER:
        DS.B        31                  * Buffer für Text.
```

Man probiere nacheinander folgende Eingaben:

```
NEU:=5
5+NEU
TEXT:=?
TEXT+1
NEU:=NEU+1
NEU.
```

Achtung! Zwischen den Operationen dürfen keine Leerzeichen stehen.
Auf der Anzeige erscheint jeweils der Wert des berechneten Ausdrucks.

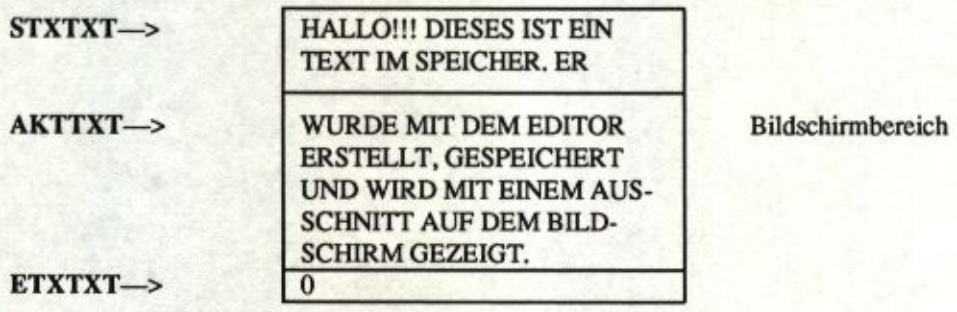
TRAP-Nummer : **31**
Befehlsname : **CIINIT2**
Befehlsgruppe : Zeicheneingabe.
Kurzbeschreibung : Zeiger auf Textstart setzen für CI2.

Eingaberegister : Keine.
Ausgaberegister : Keine.
Zerstörte Register : Keine.
Ab Version : 3.1
Änderungen zu 4.3 : Nein.
Änderungen zu 6.1 : Nein.
Siehe auch : CI (12) CSTS (13) CI2 (32)

Der aktuelle Textzeiger wird auf den Textanfang gesetzt. Der Befehl wird zusammen mit dem Befehl CI2 verwendet, um Texte aus dem Speicher zu erreichen. Es gibt drei wichtige Textzeiger, die auch für den Editor verwendet werden:

STXTXT Adresse des Textanfangs,
AKTTXT Adresse der Startzeile des Bildschirms,
ETXTXT Adresse des Textendes.

Die Speicheraufteilung eines Textes sieht so aus:



Die 0 am Textende hat die Codierung 00 und ist nicht mit dem ASCII-Zeichen 0 mit der Codierung $30 zu verwechseln.

AKTTXT zeigt hier auf das W von WURDE.
STXTXT zeigt auf den ersten Buchstaben im Text.

Mit CIINIT2 wird AKTTXT auf den Wert von STXTXT gesetzt. Die Variablen STXTXT ($36(A5)) und AKTTXT ($3A(A5)) zeigen dann auf den Textanfang.

Der Textanfang kann durch das OPTIONEN-Menü (TEXTSTART) oder im Editor mit Hilfe der Befehle ESC A bzw. ESC N geändert werden.

Komplexeres Beispiel:

Der aktuelle Editortext wird auf dem Bildschirm ausgegeben.

```
START:
        MOVEQ       #!CIINIT2,D7        * CI2 vorbereiten.
        TRAP        #1
        MOVEQ       #!CLRSCREEN,D7      * Bildschirm für CO2 vorbereiten.
        TRAP        #1
SCHLEIFE:
        MOVEQ       #!CI2,D7            * Ein Zeichen holen.
        TRAP        #1
        BCS.S       ENDE                * Null ist Endekennung.
        MOVEQ       #!CO2,D7            * Zeichen ausgeben.
        TRAP        #1
        BRA.S       SCHLEIFE            * Wiederholen.
ENDE:
        RTS
```

Mit dem Programm wird der Textinhalt auf dem Bildschirm ausgegeben. Wenn man das Programm aufruft, ohne den TEXTSTART zu verändern, gibt es das Programm selbst aus.

TRAP-Nummer	: **32**		
Befehlsname	: **CI2**		
Befehlsgruppe	: Zeicheneingabe.		
Kurzbeschreibung	: Es wird ein Zeichen aus dem Speicher gelesen.		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.L = Eingelesenes Zeichen. Carry = Flag, ob Ende erreicht.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: CI (12)	CSTS (13)	CIINIT2 (31)

Mit dem Unterprogramm ist es möglich, Texte aus dem Speicher oder mit Hilfe einer eigenen Routine zu lesen.

Normalerweise wird vom Speicher gelesen. Dabei wird der Zeiger AKTTXT verwendet. Mit dem CIINIT2-Befehl kann dieser auf STXTXT, den Textstart, zurückgesetzt werden. Das Ergebnis des Aufrufs ist im Register D0.L (.B) abgelegt. Wenn das Textende erreicht ist, also das Zeichen 0 (Code NUL) gefunden wurde, wird zusätzlich das CARRY-Flag gesetzt. Es genügt aber auch, einfach auf 0 abzufragen. Ferner wird beim Textende die Variable ERRFLAG - $34(a5) - auf den Wert 2 gesetzt. Damit wird erreicht, daß eine vorher möglicherweise unterdrückte Ausgabe mit Hilfe der CO2-Routine wieder eingeschaltet wird.

Es gibt noch eine zweite Möglichkeit, Texte einzulesen. Wenn man in die Variable $2B relativ zum Variablenanfang (IOSTATB) den Wert 5 schreibt, werden die Eingaben über einen Sprungvektor gelesen. Wenn man den Sprungvektor, der auf Adresse $18(A5) steht, durch einen Sprung auf eine eigene Routine umsteuert, kann man Eingaben auch durch eine eigene Routine erzeugen.

Nach einen CIINIT2-Aufruf wird beim nächsten CI2-Aufruf im Register D0.L der Wert 1 an die USERCI-Routine übergeben, sonst der Wert 0. Das ist nötig, um Mehrpassvorgänge auch mit USERCI zu steuern, z.B. CP/M68K, um die Quelle von der Diskette mehrfach zu lesen.

Der Sprungvektor USERCI ($18) ist nach dem RESET auf die Routine CI geschaltet.

Einfaches Beispiel:

Der Editortext wird auf dem Drucker ausgegeben.

```
START:
        MOVEQ       #!CIINIT2,D7        * Textzeiger auf Textanfang.
        TRAP        #1
SCHLEIFE:
        MOVEQ       #!CI2,D7            * Zeichen lesen.
        TRAP        #1
        BCS.S       ENDE                * Ende.
        MOVEQ       #!LO,D7             * Zeichen auf dem Drucker ausgeben.
        TRAP        #1
        BRA.S       SCHLEIFE            * Wiederholen.
ENDE:
        RTS
```

TRAP-Nummer	:	33
Befehlsname	:	**CO2**
Befehlsgruppe	:	Zeichenausgabe.
Kurzbeschreibung	:	Ein Zeichen wird ausgegeben.
Eingaberegister	:	D0.B = Auszugebendes Zeichen.
Ausgaberegister	:	Carry = Gibt an, ob Zeichen ausgegeben wurde oder ob Spezialfunktion durchgeführt wird.
Zerstörte Register	:	Keine.
Ab Version	:	3.1
Änderungen zu 4.3	:	Die Funktionen CTRL-S und CTRL-Q gibt es nicht mehr. Eine Umlenkung auf die serielle Karte ist möglich.
Änderungen zu 6.1	:	Das Carry-Flag hat jetzt festgelegte Funktionen.
Siehe auch	:	CLRSCREEN (20), CO (21), LO (22), SIZE (25), CRT (49), LST (50), USR (51), NIL (52), SETPASS (55), CURSEIN (61), CURSAUS (62), CHAR (63), CURON (81), CUROFF (82), CRLF (99), GETLINE (100), GETCURXY (101), SETCURXY (102), LSTS (117), CO2SER (129)

Zunächst hat der Befehl CO2 die gleiche Funktion wie der CO-Befehl, nämlich die Ausgabe eines Zeichens, das im Register D0.B steht, auf den Bildschirm. Die Ausgabe auf dem Bildschirm ist bei der Erläuterung des CO-Befehls genau beschrieben.

Der Befehl kann aber noch mehr: Er kann die Ausgabe auf einen Drucker (LST), eine eigene Routine (USR), ins Leere (NIL) oder auf die serielle Karte (CO2SER) lenken.

Die Ausgabe erfolgt nur, wenn die Variable PASSFLAG den Wert 2 enthält. Der Wert ist vom Grundprogramm voreingestellt. Die Abfrage des PASSFLAG ist speziell für den Assembler gedacht, wird aber auch von anderen Übersetzern genutzt, die mit dem Grundprogramm zusammenarbeiten.

Die Umsteuerung geschieht über die Speicherzelle $2A (IOSTAT) relativ zum Variablenanfang.

Folgende Codes sind für die Umsteuerung möglich:

1 Unterdrückt Ausgabe.
Es wird ERRFLAG geprüft. Nur wenn dessen Inhalt ungleich 0 ist, wird ausgegeben.

2 Ausgabe auf den Bildschirm.
Danach wird die Schreibseite auf 0 gestellt.

3 Ausgabe auf den Drucker.

4 Ausgabe auf den Drucker,
aber das Zeichen LF (Code $0A) wird unterdrückt. Bei manchen Druckern ist das erforderlich.

5 Ausgabe auf USERCO.
Dabei steht ein Sprungvektor auf der Adresse $24 relativ zum Variablenanfang. Nach einem RESET ist die Ausgabe immer auf die Routine CO geschaltet.

6 Die Ausgabe wird auf die serielle Schnittstelle umgelenkt. Sie muß allerdings vorher mit dem Befehl SIINIT auf die gewünschten Werte eingestellt werden.

Ab Version 6.2 ist das Carry-Flag auf Null gesetzt, wenn ein Zeichen ausgegeben wurde (Über LST, NIL usw.). Es ist auf Eins gesetzt, wenn eine der unten beschriebenen Extrafunktionen (CTRL-A) durchgeführt wurde. Ist CO2 auf USERCO geschaltet, muß die USER-Routine dafür sorgen, daß Carry den richtigen Wert enthält.

Der Befehl CO2 hat im übrigen eine weitere Funktion: Gibt man beim Aufruf der CO2-Routine im Register D0.B den Code $01 (CTRL-A) ein, werden alle folgenden Zeichen bis zum Line-Feed (Code $0A) im Buffer AUSBUF zwischengespeichert. Danach erfolgt die Ausführung.

Dazu gibt es folgende Befehle (ADR und WERT können beliebige Ausdrücke sein, siehe WERT, z. B. selbst definierte Symbole):

E ADR WERT0 WERT1 WERT2 WERT3 WERT4 WERT5 WERT6 WERT7

Ausführen eines Programms ADR mit den Parametern WERT0 bis WERT7. Die Parameter können auch weggelassen werden. Dabei werden die Parameter von 0 bis 7 in die Register D0.L bis D7.L gespeichert, bevor das Programm ADR aufgerufen wird. Damit kann man in Hochsprachen Maschinenunterprogramme aufrufen, auch wenn das ursprünglich im Sprachkonzept nicht vorgesehen war.

Da das Aufspeichern und Aufrufen des Maschinenprogramms einige Zeit in Anspruch nimmt, sollten möglichst nur größere zeitaufwendige Unterprogramme aufgerufen werden.

Einfaches PASCAL-Beispiel:

Das Unterprogramm SCHREITE wird aufgerufen, wobei im Register D0.L der Wert 50 übergeben wird.

```
BEGIN
      WRITELN(chr(1),'@SCHREITE 50');
END.
```

G ADR

Ein Byte wird von der Adresse ADR geholt. Bei der nächsten CI-Eingabe wird der Wert dort eingelesen. Als Trennung wird bei der CI-Eingabe als letztes Zeichen ein Leerzeichen mit ausgegeben. Der Wert wird als dezimale ASCII-Ausgabe zurückgegeben.

Mit diesem Befehl ist es möglich, unabhängig von der Programmiersprache Inhalte von Speicherzellen abzufragen. Eine CO2-Ausgabe, die während der Eingabe erfolgt, wird so lange unterdrückt, bis die Eingabe beendet ist, um unerwünschte Echo-Effekte auf dem Bildschirm zu vermeiden.

Einfaches PASCAL-Beispiel:

In die Variable I (als Integer deklariert) wird der Inhalt der Speicherzelle $8000 gelesen.

```
BEGIN
      WRITELN(CHR(1),'G $8000');
      READ(I);
END.
```

P ADR WERT

Auf die Speicherzelle ADR wird der Wert WERT geschrieben. Dabei wird aber nur ein Byte adressiert. Damit kann man von Hochsprachen aus Werte im Speicher modifizieren. Natürlich kann man auch sehr bequem auf IO-Adressen zugreifen.

Einfaches PASCAL-Beispiel:

Auf die Adresse $FFFFFF70 wird der Wert $0C geschrieben und der Bildschirm weiß eingefärbt.

```
BEGIN
      WRITELN(CHR(1),'P $FFFFFF70 12');
END.
```

Der Text für diese Spezialfunktionen darf nicht länger als 131 Zeichen sein, da der Textbuffer nicht größer ist.

Im übrigen gelten alle Befehle, die schon beim CO-Befehl gezeigt wurden. CO2 ist also ein Unterprogramm, das es ganz schön in sich hat. Hier nun ein einfaches Beispiel:

Einfaches Beispiel:

Das Zeichen A wird ausgegeben.

```
START:
        MOVEQ     #!CLRSCREEN,D7      * Init des Cursor und Bildschirm löschen.
        TRAP      #1
        MOVEQ     #'A',D0             * Buchstabe A ausgeben.
        MOVEQ     #!CO2,D7
        TRAP      #1
        RTS
```

Die Ausgabe des Buchstabens A kann man ganz leicht auf den Drucker umlenken, wenn man im OPTIONEN-Menü die Ausgabe auf den Drucker umstellt.

Komplexeres Beispiel:

Es werden zwei Schildkrötenbefehle mit CO2 ausgeführt. Dadurch wird ein gleichschenkliges Dreieck gezeichnet.

```
START:
        MOVEQ     #3-1,D3             * 3 Durchläufe.
SCHLEIF0:
        LEA       TEXT(PC),A0         * Ausgabetextadresse.
SCHLEIF1:
        MOVE.B    (A0)+,D0            * Zeichen holen.
        BEQ.S     WEITER              * Null ist Ende.
        MOVEQ     #!CO2,D7            * Ausgabe.
        TRAP      #1
        BRA.S     SCHLEIF1            * Wiederholen.
WEITER:
        DBRA      D3,SCHLEIF0         * Äußere Schleife.
        RTS
TEXT:
        DC.B      1,'E @SCHREITE 100',$A
        DC.B      1,'E @DREHE 120',$A,0
```

TRAP-Nummer	: 34
Befehlsname	: **SETFLIP**
Befehlsgruppe	: Grafik.
Kurzbeschreibung	: Die automatische Bildseitenumschaltung wird eingestellt.
Eingaberegister	: D0.B = Wert für Zwei-Seiten-Umschaltung. D1.B = Wert für Vier-Seiten-Umschaltung.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 3.1
Änderungen zu 4.3	: Nein.
Änderungen zu 6.1	: Nein.
Siehe auch	: MOVETO (8), DRAWTO (9), CLR (16), CLPG (17), WAIT (18), CMD (26), NEWPAGE (27), SETPEN (37), ERAPEN (38), CMDPRINT (40), AUTOFLIP (60), SETXOR (77), GETXOR (78), SETCOLOR (79), GETCOLOR (80), GETXY (103), HARDCOPY (125), GRAFIK (126), GDPVERS (127)

Die GDP-Baugruppe besitzt vier Bildseiten. Mit SETFLIP kann man eine automatische Umschaltrate beeinflussen, die z.B. mit AUTOFLIP ausgeführt wird. Die Umschaltung erfolgt auch dann, wenn man z. B. nach Ausführung eines Programms die Anweisung M=Menü erhält oder Befehle der Schildkrötensprache ausführt.

Es gibt zwei Arten der automatischen Umschaltung: Den Wechsel zwischen zwei Bildseiten oder den zwischen vier Bildseiten.

Dazu übergibt man beim Aufruf zwei Parameter. Im Register D0.B steht die Wechselrate für die Zwei-Seiten-Umschaltung und im Register D1.B die Wechselrate für die Vier-Seiten-Umschaltung. Der Wert 0 unterdrückt die Umschaltung, jeder Wert <> 0 führt zur Umschaltung alle 20*Wert Millisekunden.

Einfaches Beispiel:

Der Cursor blinkt nur noch im Sekundentakt.

```
START:
        MOVEQ     #!CLRSCREEN,D7     * Bildschirm löschen und Cursor darstellen.
        TRAP      #1
        MOVEQ     #50,D0             * Blinkrate für Zwei-Seiten-Umschaltung auf
        CLR       D1                 * 1 Sekunde einstellen.
        MOVEQ     #!SETFLIP,D7       * Umschaltrate setzen.
        TRAP      #1
        RTS
```

Komplexeres Beispiel:

Eine Buchstabenreihe wandert über den Bildschirm. Die Bewegung erklärt sich daraus, daß das A auf alle Bildseiten geschrieben wird. Dabei sind aber die Buchstaben jeweils versetzt. Durch das Umschalten der Seiten kommt der Bewegungseffekt zustande.

```
START:
        MOVE      #!SYSTEM,D7        * System-Informationen holen.
        TRAP      #1
        AND       #7,D0              * Nur CPU-Einstellung lassen.
        MULS      #$FF73,D0          * Register 3 GDP.
        MOVEA.L   D0,A0              * Adresse merken.
        MOVEQ     #!WAIT,D7          * Warten, bis GDP fertig ist.
        TRAP      #1
        MOVE.B    #$33,(A0)          * Größe einstellen.
        CLR       D1
        MOVE      #128,D2
        MOVEQ     #!MOVETO,D7        * Anfangsposition.
        TRAP      #1
        CLR       D3                 * Schreibseite.
        MOVEQ     #40-1,D4           * 40 Buchstaben.
```

```
SCHLEIFE:
        MOVE    D3,D0
        AND     #3,D0           * Schreibseite (0 - 3).
        CLR     D1              * Leseseite ist immer Seite 0.
        MOVEQ   #!NEWPAGE,D7    * Seite einstellen.
        TRAP    #1
        MOVEQ   #'A',D0         * Zeichen A
        MOVEQ   #!CMD,D7        * ausgeben.
        TRAP    #1
        ADDQ    #1,D3           * Schreibseite um 1 weiterschalten.
        DBRA    D4,SCHLEIFE
        CLR     D0              * Keine Zwei-Seiten-Umschaltung.
        MOVEQ   #5,D1           * Vier-Seiten-Umschaltung alle 1/10 Sekunde.
        MOVEQ   #!SETFLIP,D7    * Seitenumschaltung an.
        TRAP    #1
        MOVEQ   #!CI,D7         * Warten, bis Taste gedrückt.
        TRAP    #1
        RTS
```

Die automatische Bildschirmumschaltung mit vier Seiten ermöglicht bewegte Graphiken, die sehr komplex sein können, wenn man die Bewegung in vier Teilphasen zerlegen kann. Beispiel: Drehen von Zahnrädern (auch 3D), Abläufe mit sich bewegenden Pfeilen etc. Im allgemeinen benötigt man aber nur zwei Seiten.

TRAP-Nummer : **35**
Befehlsname : **DELAY**
Befehlsgruppe : Synchronisation.
Kurzbeschreibung : Warteschleife in 1/10 Sekunden.

Eingaberegister : D0.L = Zeit, die gewartet werden soll.
Ausgaberegister : Keine.
Zerstörte Register : D0
Ab Version : 3.1
Änderungen zu 4.3 : Nein.
Änderungen zu 6.1 : Nein.
Siehe auch : SYNC (28)

Die Routine DELAY gestattet quarzgenaue Verzögerungen im Raster von 100 Millisekunden. Dabei wird während der Verzögerungszeit das automatische Seitenumschalten aufrechterhalten, sodaß sich keine Störungen bei der Seitenumschaltung durch SETFLIP ergeben. Das ist insbesondere bei Bildern mit der Schildkrötengraphik wichtig. Im Register D0.L wird die Anzahl der 1/10 Sekunden angegeben. Wenn die Zahl gleich 0 ist, erfolgt keine Ausführung des Befehls.

Einfaches Beispiel:

Das Programm wartet eine Sekunde.

```
START:
        MOVEQ       #10,D0              * Eine Sekunde
        MOVEQ       #!DELAY,D7          * warten.
        TRAP        #1
        RTS
```

Komplexeres Beispiel:

Es wird ein Quadrat und ein Kreis gezeichnet. Dann wird die Darstellung jede Sekunde durch Umschalten der Seiten gewechselt.

```
START:
        MOVE        #360-1,D3           * Kreis zeichnen.
SCHLEIF0:
        MOVEQ       #1,D0
        MOVEQ       #!SCHREITE,D7       * 1 Schritt gehen.
        TRAP        #1
        MOVEQ       #1,D0
        MOVEQ       #!DREHE,D7          * 1 Grad drehen.
        TRAP        #1
        DBRA        D3,SCHLEIF0         * 360 mal.
        MOVEQ       #10,D0
        MOVEQ       #!DELAY,D7          * 1 Sekunde warten.
        TRAP        #1
        MOVEQ       #!CLR,D7            * Bildschirm löschen.
        TRAP        #1
        MOVEQ       #4-1,D3             * 4 Durchgänge.
SCHLEIF1:
        MOVEQ       #100,D0             * 100 Schritte gehen.
        MOVEQ       #!SCHREITE,D7
        TRAP        #1
        MOVEQ       #90,D0              * 90 Grad drehen.
        MOVEQ       #!DREHE,D7
        TRAP        #1
        DBRA        D3,SCHLEIF1         * Quadrat.
        MOVEQ       #10,D0
        MOVEQ       #!DELAY,D7          * 1 Sekunde warten.
        TRAP        #1
        MOVEQ       #!CLR,D7            * Bildschirm löschen.
        TRAP        #1
        MOVEQ       #!CSTS,D7           * Wiederholen, bis Taste gedrückt.
        TRAP        #1
        BEQ.S       START
        RTS
```

TRAP-Nummer : **36**
Befehlsname : **FIRSTTIME**
Befehlsgruppe : Schildkrötengrafik.

Eingaberegister : Keine.
Ausgaberegister : Keine.
Zerstörte Register : Keine.
Ab Version : 3.1
Änderungen zu 4.3 : Nein.
Änderungen zu 6.1 : Nein.
Siehe auch : SCHREITE (1), DREHE (2), HEBE (3), SENKE (4), SET (7), SCHR16TEL (19), GRAPOFF (39), HIDE (47), SHOW (48), AUFXY (92), KORXY (93), AUFK (94), GETK (95)

Der Befehl installiert die Schildkrötengrafik und setzt die Schildkröte in die Bildmitte. Gleichzeitig aktiviert er die Bildseitenautomatik für die Umschaltung. Ferner stellt er die Schreib- und Leseseite auf 0. FIRSTTIME wird auch automatisch beim ersten Schildkrötenbefehl mit aufgerufen. Es ist also nur in bestimmten Sonderfällen nötig, diesen Befehl zu verwenden. Ein Fall ist z. B., daß man mit mehreren Bildseiten arbeiten und dabei die Schildkröte nicht auf dem Bildschirm haben möchte.

Einfaches Beispiel:

Schildkrötengrafik anschalten, aber Schildkröte nicht zeigen und Seitenumschaltung ausschalten.

```
START:
        MOVEQ   #!FIRSTTIME,D7      * Schildkröte anschalten.
        TRAP    #1
        MOVEQ   #!HIDE,D7           * Schildkröte nicht darstellen.
        TRAP    #1
        RTS
```

Die Schildkröte wird in diesem Beispiel nach dem Start nicht gesetzt, weil die Variable AUTOFLIP durch den HIDE-Befehl auf 0 gesetzt und damit die automatische Bildseitenumschaltung ausgeschaltet wird.

Wenn man AUTOFLIP später verwenden will, um alle Bildseiten nacheinander darzustellen, muß man mit dem Befehl GRAPOFF die Schildkrötengraphik ausschalten.

Komplexeres Beispiel:

Es wird auf allen vier Seiten ein Dreieck ausgegeben, das aber jeweils gedreht ist. Danach werden die Seiten umgeschaltet, wodurch ein Dreheffekt auftritt.

```
START:
        MOVEQ   #!FIRSTTIME,D7      * Schildkröte anschalten.
        TRAP    #1
        MOVEQ   #!HIDE,D7           * Schildkröte nicht darstellen.
        TRAP    #1
        MOVEQ   #4-1,D4             * Auf alle vier Seiten schreiben.
SCHLEIF0:
        MOVE    D4,D0               * Schreibseite.
        CLR     D1                  * Leseseite ist jetzt immer 0.
        MOVEQ   #!NEWPAGE,D7        * Seiten einstellen.
        TRAP    #1
        MOVEQ   #6-1,D5             * 6 mal drehen.
SCHLEIF1:
        MOVEQ   #3-1,D6             * Dreieck.
```

```
SCHLEIF2:
        MOVEQ     #100,D0
        MOVEQ     #!SCHREITE,D7     * 100 Schritte gehen.
        TRAP      #1
        MOVEQ     #120,D0           * 120 Grad drehen.
        MOVEQ     #!DREHE,D7
        TRAP      #1
        DBRA      D6,SCHLEIF2       * Wiederholung.
        MOVEQ     #60,D0
        MOVEQ     #!DREHE,D7        * Um 60 Grad drehen.
        TRAP      #1
        DBRA      D5,SCHLEIF1       * Wiederholen.
        MOVEQ     #45,D0
        MOVEQ     #!DREHE,D7        * Um 45 Grad drehen, damit Figuren versetzt
        TRAP      #1                * sind.
        DBRA      D4,SCHLEIF0
        MOVEQ     #!GRAPOFF,D7      * Schildkrötengrafik ausschalten.
        TRAP      #1
        CLR       D0
        MOVEQ     #5,D1             * Vier-Seiten-Umschaltung
        MOVEQ     #!SETFLIP,D7      * alle 1/10 Sekunde.
        TRAP      #1
        MOVEQ     #!CI,D7           * Auf Tastatureingabe warten.
        TRAP      #1
        RTS
```

TRAP-Nummer	: **37**		
Befehlsname	: **SETPEN**		
Befehlsgruppe	: Grafik.		
Kurzbeschreibung	: Schreibstift der GDP wird auf Schreiben gesetzt.		
Eingaberegister	: Keine.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: MOVETO (8)	DRAWTO (9)	CLR (16)
	CLPG (17)	WAIT (18)	CMD (26)
	NEWPAGE (27)	SETFLIP (34)	ERAPEN (38)
	CMDPRINT (40)	AUTOFLIP (60)	SETXOR (77)
	GETXOR (78)	SETCOLOR (79)	GETCOLOR (80)
	GETXY (103)	HARDCOPY (125)	GRAFIK (126)
	GDPVERS (127)		

Damit wird der Graphikprozessor in den Schreibmode mit gesenktem Schreibstift versetzt. Alle nachfolgenden Grafikbefehle erfolgen also mit weißer Linie auf ggf. dunklem Hintergrund.

Das Gegenstück zu diesem Befehl ist ERAPEN.

Einfaches Beispiel:

Das Programm ruft nur SETPEN auf.

```
START:
        MOVEQ       #!SETPEN,D7
        TRAP        #1
        RTS
```

Mit dem Befehl kann man auch vorausgegangene Befehle, die einen Löschvorgang bewirken, zurücksetzen.

Ein komplexeres Beispiel folgt bei der Darstellung des nächsten Befehls.

TRAP-Nummer	: **38**
Befehlsname	: **ERAPEN**
Befehlsgruppe	: Grafik.
Kurzbeschreibung	: Stellt den Schreibstift der GDP auf Löschen ein.
Eingaberegister	: Keine.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 3.1
Änderungen zu 4.3	: Nein.
Änderungen zu 6.1	: Nein.
Siehe auch	: MOVETO (8), DRAWTO (9), CLR (16), CLPG (17), WAIT (18), CMD (26), NEWPAGE (27), SETFLIP (34), SETPEN (37), CMDPRINT (40), AUTOFLIP (60), SETXOR (77), GETXOR (78), SETCOLOR (79), GETCOLOR (80), GETXY (103), HARDCOPY (125), GRAFIK (126), GDPVERS (127)

Schreibstift auf Löschen umschalten. Dabei wird der Stift auch gesenkt. Nun kann man dunkle Linien auf hellem Grund schreiben oder vorhandene Grafikteile löschen.

Das Gegenstück zu diesem Befehl ist SETPEN.

Einfaches Beispiel:

Schreibstift auf LÖSCHEN schalten.

```
START:
        MOVEQ       #!ERAPEN,D7
        TRAP        #1
        RTS
```

Komplexeres Beispiel:

Der Buchstabe A wandert über den Bildschirm.

```
START:
        MOVE        #!SYSTEM,D7     * System-Informationen holen.
        TRAP        #1
        AND         #7,D0           * Nur CPU-Information lassen.
        MULS        #$FF73,D0
        MOVEA.L     D0,A0           * Size-Register GDP.
        CLR         D1
        MOVE        #128,D2         * Anfangsposition.
        MOVEQ       #!WAIT,D7
        TRAP        #1
        MOVE.B      #$33,(A0)       * Größe.
SCHLEIFE:
        MOVEQ       #!MOVETO,D7     * Positionieren.
        TRAP        #1
        MOVEQ       #'A',D0
        MOVEQ       #!CMD,D7        * A direkt ausgeben.
        TRAP        #1
SYNC:
        MOVEQ       #!SYNC,D7       * 20 ms warten.
        TRAP        #1
        BEQ.S       SYNC
        MOVEQ       #!MOVETO,D7     * Neu positionieren.
        TRAP        #1
        MOVEQ       #!ERAPEN,D7     * Auf Löschen.
        TRAP        #1
        MOVEQ       #'A',D0
        MOVEQ       #!CMD,D7        * A ausgeben.
        TRAP        #1
        MOVEQ       #!SETPEN,D7     * Auf Schreiben.
        TRAP        #1
        ADDQ        #1,D1           * Nächste X-Position.
        CMP         #512,D1
        BNE.S       SCHLEIFE        * Bis rechter Rand erreicht.
        RTS
```

TRAP-Nummer	: 39		
Befehlsname	: **GRAPOFF**		
Befehlsgruppe	: Schildkrötenbefehl.		
Kurzbeschreibung	: Schildkrötengrafik wird ausgeschaltet.		
Eingaberegister	: Keine.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SCHREITE (1)	DREHE (2)	HEBE (3)
	SENKE (4)	SET (7)	SCHR16TEL (19)
	FIRSTTIME (36)	HIDE (47)	SHOW (48)
	AUFXY (92)	KORXY (93)	AUFK (94)
	GETK (95)		

Mit dem Befehl wird die Schildkrötengrafik ausgeschaltet, d. h. die Schildkröte wird von da an nicht mehr auf die Bildseite 1 gezeichnet. Sonst ändern sich keine Einstellungen. Wenn man danach einen Schildkrötenbefehl wie z. B. SCHREITE gibt, beginnt die Schildkröte wieder in der Mitte des Bildschirms zu zeichnen.

Intern wird durch den Befehl die Variable FIRST mit dem Wert 1 belegt.

Einfaches Beispiel:

Die Schildkrötengrafik wird ausgeschaltet.

```
START:
        MOVEQ     #!GRAPOFF,D7      * Schildkröte ausschalten.
        TRAP      #1
        RTS
```

Komplexeres Beispiel:

Es wird eine Linie gezogen. Danach wird die Schildkrötengrafik ausgeschaltet. Beim nächsten Befehl beginnt sie wieder in der Mitte zu zeichnen.

```
START:
        MOVEQ     #100,D0           * Linie zeichnen.
        MOVEQ     #!SCHREITE,D7
        TRAP      #1
        MOVEQ     #!GRAPOFF,D7      * Ausschalten.
        TRAP      #1
        MOVEQ     #10,D0            * 1 Sekunde warten.
        MOVEQ     #!DELAY,D7
        TRAP      #1
        MOVEQ     #90,D0
        MOVE      #!DREHE,D7        * 90 Grad drehen.
        TRAP      #1                * Dadurch Schildkröte wieder an.
        MOVEQ     #100,D0
        MOVEQ     #!SCHREITE,D7     * Linie zeichnen.
        TRAP      #1
        RTS
```

TRAP-Nummer	:	40
Befehlsname	:	**CMDPRINT**
Befehlsgruppe	:	Grafik.
Kurzbeschreibung	:	Befehlsfolge direkt an den GDP geben.
Eingaberegister	:	D0.B = Schriftgröße. D1.W = X-Position. D2.W = Y-Position. A0.L = Adresse der Ausgabedaten.
Ausgaberegister	:	A0.L = Zeigt hinter die Endekennung.
Zerstörte Register	:	D0
Ab Version	:	3.1
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	MOVETO (8), DRAWTO (9), CLR (16), CLPG (17), WAIT (18), CMD (26), NEWPAGE (27), SETFLIP (34), SETPEN (37), ERAPEN (38), AUTOFLIP (60), SETXOR (77), GETXOR (78), SETCOLOR (79), GETCOLOR (80), GETXY (103), HARDCOPY (125), GRAFIK (126), GDPVERS (127)

Mit CMDPRINT kann man eine Befehlssequenz direkt an den GDP ausgeben. Dazu muß man im Register D0.B die Schriftgröße einstellen (wie Register 3 des GDP-Prozessors EF9366). Wenn keine Schriftzeichen ausgegeben werden, wird der Wert ignoriert. In D1.W steht die X-Koordinate und in D2.W die Y-Koordinate. Das Register A0.L enthält die Adresse, in der Daten stehen. Eine binäre 0 (Code 00) beendet den Text. Man kann mit dem Befehl CMDPRINT nicht den Befehl AUF SCHREIBEN SCHALTEN mit dem Code 00 an den GDP ausgeben. Das muß man ggf. mit den Befehlen SETPEN oder mit CMD tun. Der Befehl CMDPRINT eignet sich sowohl zur Textausgabe als auch zur Ausgabe von Grafikbefehlen, den sogenannten Kurzvektoren, mit denen man sehr schnell kleine Figuren zeichnen kann.

Einfaches Beispiel:

Es wird der Text HALLO ausgegeben.

```
START:
        LEA         DATEN(PC),A0        * Adresse der Daten.
        MOVEQ       #$22,D0             * Textgröße.
        MOVE        #200,D1             * X-Position.
        MOVEQ       #120,D2             * Y-Position.
        MOVEQ       #!CMDPRINT,D7       * Ausgabe direkt an GDP.
        TRAP        #1
        RTS
DATEN:
        DC.B        'HALLO!',0          * Quelle.
```

Da die Anwendung der Kurzvektoren so interessant ist, hier der prinzipielle Aufbau eines Kurzvektors:

```
B7   B6   B5   B4   B3   B2   B1   B0
1    —DX—      —DY—      —RICHTUNG—
```

Am 1. Bit (B7) erkennt der GDP, daß es sich um einen Kurzvektor handelt. Mit DX wird die Anzahl der Punkte in X-Richtung beschrieben, mit DY die Anzahl der Punkte in Y-Richtung. RICHTUNG legt die Richtung des Vektors fest. %00 bedeutet dabei in X- bzw. Y-Richtung keinen Schritt ausführen, %01 bedeutet einen Schritt, %10 sind zwei Schritte und %11 drei Schritte. Mit dem Code $80 kann man z. B. einen einzelnen Punkt setzen.

Richtungscode:

```
            010
       011      001
110                      000
       111      101
            100
```

Für weitere Auskünfte über die Kurzvektoren oder andere Befehle des GDP sollte man entweder das Handbuch der GDPHS bzw. Datenblätter des EF9366 zu Hilfe nehmen.

Einfaches Beispiel:

Es wird ein kleines Achteck ausgegeben.

```
START:
        LEA       DATEN(PC),A0        * Adresse der Daten.
        MOVEQ     #$22,D0             * Größe (Wird eingestellt aber dann ignoriert).
        MOVE      #250,D1             * X-Position.
        MOVEQ     #120,D2             * Y-Position.
        MOVEQ     #!CMDPRINT,D7       * Befehle ausführen.
        TRAP      #1
        RTS
DATEN:
        DC.B      %11111000           * Nach rechts mit 3 Schritten.
        DC.B      %11111001           * Schräg rechts nach oben.
        DC.B      %11111010           * Nach oben.
        DC.B      %11111011           * Schräg links nach oben.
        DC.B      %11111110           * Nach links.
        DC.B      %11111111           * Schräg links nach unten.
        DC.B      %11111100           * Nach unten.
        DC.B      %11111101           * Schräg rechts nach unten.
        DC.B      0                   * Endekennung.
```

TRAP-Nummer	: **41**		
Befehlsname	: **PRINT2X**		
Befehlsgruppe	: Wertausgabe.		
Kurzbeschreibung	: Sedezimale Ausgabe mit 2 Stellen.		
Eingaberegister	: D0.B = Auszugebender Wert. A0.L = Zieladresse für Ausgabe.		
Ausgaberegister	: A0.L = Zeigt auf Endekennung.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: PRINT4X (42)	PRINT6X (43)	PRINT8X (44)
	PRINT8B (45)	PRINT4D (46)	PRINT8D (70)
	PRINTV8D (71)		

Sedezimale Ausgabe eines Wertes. Es werden zwei sedezimale Ziffern ausgegeben. Der Wert steht zuvor im Register D0.B. Im Register A0.L ist die Zieladresse für die sedezimalen Ziffern angegeben. Das Ergebnis wird im ASCII-Code abgelegt. A0.L zeigt anschließend auf den nächsten freien Platz. Dort ist eine 0 (Code 00) abgelegt, sodaß man den Text mit Befehlen wie WRITE direkt ausgeben kann.

Einfaches Beispiel:

Es wird der Wert $41 im Zielbuffer abgelegt.

```
START:
        LEA     BUFFER(PC),A0       * A0.L ist Adresse Zielbuffer.
        MOVEQ   #$41,D0             * Wert, in ASCII-Code: A.
        MOVEQ   #!PRINT2X,D7        * Ausgabe.
        TRAP    #1
        RTS
BUFFER:
        DS.B    2                   * Zielbuffer
```

Komplexeres Beispiel:

Es wird von $00 bis $FF gezählt, die Zahlen werden ausgegeben.

```
START:
        CLR     D4                  * D4 = Zähler.
SCHLEIFE:
        LEA     BUFFER(PC),A0       * Adresse für Ablage.
        MOVE.B  D4,D0
        MOVEQ   #!PRINT2X,D7        * Zahl ausgeben.
        TRAP    #1
        LEA     BUFFER(PC),A0       * Adresse wieder laden.
        MOVEQ   #$22,D0             * Schriftgröße.
        MOVE.W  #255,D1             * X-Position.
        MOVE    #128,D2             * Y-Position.
        MOVEQ   #!WRITE,D7          * Ausgabe auf den Bildschirm.
        TRAP    #1
        MOVEQ   #1,D0               * 1/10 Sekunde warten.
        MOVEQ   #!DELAY,D7
        TRAP    #1
        ADDQ    #1,D4               * Nächste Zahl.
        CMP     #256,D4
        BNE.S   SCHLEIFE            * Wiederholen.
        RTS
BUFFER:
        DS.B    4                   * Eigentlich nur 3 Bytes nötig.
```

Wenn man den Befehl DELAY wegläßt, erfolgt die Ausgabe so schnell, daß man nichts mehr erkennen kann.

TRAP-Nummer	: **42**		
Befehlsname	: **PRINT4X**		
Befehlsgruppe	: Wertausgabe.		
Kurzbeschreibung	: Sedezimale Ausgabe mit 4 Stellen.		
Eingaberegister	: D0.W = Auszugebender Wert. A0.L = Zieladresse für Ausgabe.		
Ausgaberegister	: A0.L = Zeigt auf Endekennung.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: PRINT2X (41)	PRINT6X (43)	PRINT8X (44)
	PRINT8B (45)	PRINT4D (46)	PRINT8D (70)
	PRINTV8D (71)		

Sedezimale Ausgabe eines Wertes. Es werden vier sedezimale Ziffern ausgegeben. Der Wert steht zuvor im Register D0.W. Im Register A0.L ist die Zieladresse für die sedezimalen Ziffern angegeben. Das Ergebnis wird im ASCII-Code abgelegt. A0.L zeigt anschließend auf den nächsten freien Platz. Dort ist eine 0 (Code 00) abgelegt, sodaß man den Text mit Befehlen wie WRITE direkt ausgeben kann.

Einfaches Beispiel:

Im Zielbuffer wird der Wert $12AB abgelegt.

```
START:
        LEA     BUFFER(PC),A0       * A0.L ist Adresse Zielbuffer.
        MOVE.W  #$12AB,D0           * Wert.
        MOVEQ   #!PRINT4X,D7        * Ausgabe.
        TRAP    #1
        RTS
BUFFER:
        DS.B    6                   * Zielbuffer.
```

TRAP-Nummer	: **43**		
Befehlsname	: **PRINT6X**		
Befehlsgruppe	: Wertausgabe.		
Kurzbeschreibung	: Sedezimale Ausgabe 6 Stellen.		
Eingaberegister	: D0.L = Auszugebender Wert (aber nur 3 Bytes). A0.L = Zieladresse für Ausgabe.		
Ausgaberegister	: A0.L = Zeigt auf Endekennung.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: PRINT2X (41)	PRINT4X (42)	PRINT8X (44)
	PRINT8B (45)	PRINT4D (46)	PRINT8D (70)
	PRINTV8D (71)		

Sedezimale Ausgabe eines Wertes. Es werden sechs sedezimale Ziffern ausgegeben. Der Wert steht zuvor im Register D0.L. Im Register A0.L ist die Zieladresse für die sedezimalen Ziffern angegeben. Das Ergebnis wird im ASCII-Code abgelegt. A0.L zeigt anschließend auf den nächsten freien Platz. Dort ist eine 0 (Code 00) abgelegt, sodaß man den Text mit Befehlen wie WRITE direkt ausgeben kann.

Einfaches Beispiel:

Im Zielbuffer wird der Wert $12ABCD abgelegt.

```
START:
        LEA     BUFFER(PC),A0       * A0.L ist Adresse Zielbuffer.
        MOVE.L  #$12ABCD,D0         * Wert.
        MOVEQ   #!PRINT6X,D7        * Ausgabe.
        TRAP    #1
        RTS
BUFFER:
        DS.B    8                   * Zielbuffer.
```

TRAP-Nummer	: **44**		
Befehlsname	: **PRINT8X**		
Befehlsgruppe	: Wertausgabe.		
Kurzbeschreibung	: Sedezimale Ausgabe 8 Stellen.		
Eingaberegister	: D0.L = Auszugebender Wert. A0.L = Zieladresse für Ausgabe.		
Ausgaberegister	: A0.L = Zeigt auf Endekennung.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: PRINT2X (41)	PRINT4X (42)	PRINT6X (43)
	PRINT8B (45)	PRINT4D (46)	PRINT8D (70)
	PRINTV8D (71)		

Sedezimale Ausgabe eines Wertes. Es werden acht sedezimale Ziffern ausgegeben. Der Wert steht zuvor im Register D0.L. Im Register A0.L ist die Zieladresse für die sedezimalen Ziffern angegeben. Das Ergebnis wird im ASCII-Code abgelegt. A0.L zeigt anschließend auf den nächsten freien Platz. Dort ist eine 0 (Code 00) abgelegt, sodaß man den Text mit Befehlen wie WRITE direkt ausgeben kann.

Einfaches Beispiel:

Im Zielbuffer wird der Wert $1234ABCD abgelegt.

```
START:
        LEA     BUFFER(PC),A0       * A0.L ist Adresse Zielbuffer.
        MOVE.L  #$1234ABCD,D0       * Wert.
        MOVEQ   #!PRINT8X,D7        * Ausgabe.
        TRAP    #1
        RTS
BUFFER:
        DS.B    10                  * Zielbuffer.
```

TRAP-Nummer	: **45**		
Befehlsname	: **PRINT8B**		
Befehlsgruppe	: Wertausgabe.		
Kurzbeschreibung	: Binäre Ausgabe 8 Bit.		
Eingaberegister	: D0.B = Auszugebender Wert. A0.L = Zieladresse für Ausgabe.		
Ausgaberegister	: A0.L = Zeigt auf Endekennung.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: PRINT2X (41)	PRINT4X (42)	PRINT6X (43)
	PRINT8X (44)	PRINT4D (46)	PRINT8D (70)
	PRINTV8D (71)		

Binäre Ausgabe eines Wertes. Es werden acht Dualstellen ausgegeben. Der Wert steht zuvor im Register D0.B. Im Register A0.L ist die Zieladresse für die dualen Ziffern angegeben. Das Ergebnis wird im ASCII-Code abgelegt. A0.L zeigt anschließend auf den nächsten freien Platz. Dort ist eine 0 (Code 00) abgelegt, sodaß man den Text mit Befehlen wie WRITE direkt ausgeben kann.

Einfaches Beispiel:

Im Zielbuffer wird der Wert %00011010 abgelegt.

```
START:
        LEA     BUFFER(PC),A0       * A0.L ist Adresse Zielbuffer.
        MOVEQ   #$1A,D0             * Wert.
        MOVEQ   #!PRINT8B,D7        * Ausgabe.
        TRAP    #1
        RTS
BUFFER:
        DS.B    10                  * Zielbuffer.
```

Komplexeres Beispiel:

Es werden die Zahlen von 0 bis 255 dual ausgegeben.

```
START:
        CLR     D4                  * D4 = Zähler.
SCHLEIFE:
        LEA     BUFFER(PC),A0       * Adresse für Ablage.
        MOVE.B  D4,D0
        MOVEQ   #!PRINT8B,D7        * Zahl ausgeben.
        TRAP    #1
        LEA     BUFFER(PC),A0       * Wieder Adresse laden.
        MOVEQ   #$22,D0             * Schriftgröße.
        MOVE.W  #200,D1             * X-Position.
        MOVE    #128,D2             * Y-Position.
        MOVEQ   #!WRITE,D7          * Ausgabe auf den Bildschirm.
        TRAP    #1
        MOVEQ   #2,D0               * 200 ms warten.
        MOVEQ   #!DELAY,D7
        TRAP    #1
        ADDQ    #1,D4               * Nächste Zahl.
        CMP     #256,D4
        BNE.S   SCHLEIFE            * Wiederholen.
        RTS
BUFFER:
        DS.B    10                  * Eigentlich nur 9 Bytes nötig.
```

Das Programm zählt von 0 bis 255 und gibt das Ergebnis alle 1/5 Sekunde in dualer Schreibweise auf dem Bildschirm aus. Wenn man den Befehl DELAY wegläßt, erfolgt die Ausgabe so schnell, daß man nichts mehr erkennen kann.

TRAP-Nummer	: **46**		
Befehlsname	: **PRINT4D**		
Befehlsgruppe	: Wertausgabe.		
Kurzbeschreibung	: Dezimale Ausgabe ein Wort.		
Eingaberegister	: D0.W = Auszugebender Wert. A0.L = Zieladresse für Ausgabe.		
Ausgaberegister	: A0.L = Zeigt auf Endekennung.		
Zerstörte Register	: D0		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: PRINT2X (41)	PRINT4X (42)	PRINT6X (43)
	PRINT8X (44)	PRINT8B (45)	PRINT8D (70)
	PRINTV8D (71)		

Dezimale Ausgabe eines Wertes. Der Wert steht zuvor im Register D0.W. Im Register A0.L ist die Zieladresse für die dezimalen Ziffern angegeben. Das Ergebnis wird im ASCII-Code abgelegt. A0.L zeigt anschließend auf den nächsten freien Platz. Dort ist eine 0 (Code 00) abgelegt, sodaß man den Text mit Befehlen wie WRITE direkt ausgeben kann. Die Ausgabe erfolgt ohne Vorzeichen. Wenn man Zweierkomplement-Zahlen verwendet, muß man bei einer negativen Zahl das Vorzeichen selbst ausgeben.

Bei der dezimalen Ausgabe werden immer nur so viele Stellen ausgegeben, wie sie durch die Zahl belegt sind. Daher muß man, wenn man Zahlen übereinanderschreibt, den Zielbuffer mit Leerzeichen auffüllen und die Endekennung eventuell neu setzen.

Einfaches Beispiel:

Im Zielbuffer wird der Text 4666 abgelegt.

```
START:
        LEA         BUFFER(PC),A0       * A0.L: Adresse Zielbuffer.
        MOVE.W      #$123A,D0           * Wert.
        MOVEQ       #!PRINT4d,D7        * Ausgabe.
        TRAP        #1
        RTS
BUFFER:
        DS.B        6                   * Zielbuffer.
```

Komplexeres Beispiel:

```
START:
        MOVE        #250,D4             * D4 = Zähler.
SCHLEIFE:
        LEA         BUFFER(PC),A0       * Adresse laden.
        MOVE.L      #'    ',(A0)        * Vorlöschen.
        MOVE        D4,D0               * Wert laden.
        MOVEQ       #!PRINT4D,D7        * Ausgabe in Buffer.
        TRAP        #1
        MOVE.B      #' ',(A0)           * Endenull löschen.
        LEA         BUFFER(PC),A0
        CLR.B       4(A0)               * Ende neu setzen.
        MOVEQ       #$22,D0             * Schriftgröße.
        MOVE.W      #250,D1             * X-Position.
        MOVE        #128,D2             * Y-Position.
        MOVEQ       #!WRITE,D7          * Auf den Bildschirm.
        TRAP        #1
        MOVEQ       #1,D0               * 1/10 Sek warten.
        MOVEQ       #!DELAY,D7
        TRAP        #1
        DBRA        D4,SCHLEIFE         * Wiederholen.
        RTS
BUFFER:
        DS.B        6                   * Ablageadresse.
```

Das Programm zählt von 250 bis 0 und gibt das Ergebnis alle 1/5 Sekunde in dezimaler Schreibweise auf dem Bildschirm aus. Wenn man den Befehl DELAY wegläßt, erfolgt die Ausgabe sehr viel schneller.

TRAP-Nummer	: 47
Befehlsname	: **HIDE**
Befehlsgruppe	: Schildkrötengrafik.
Kurzbeschreibung	: Schildkröte ausblenden.
Eingaberegister	: Keine.
Ausgaberegister	: D0.B = Enthält Wert des GDP-Ports $60.
Zerstörte Register	: Keine.
Ab Version	: 3.1
Änderungen zu 4.3	: Nein.
Änderungen zu 6.1	: Nein.
Siehe auch	: SCHREITE (1), DREHE (2), HEBE (3), SENKE (4), SET (7), SCHR16TEL (19), FIRSTTIME (36), GRAPOFF (39), SHOW (48), AUFXY (92), KORXY (93), AUFK (94), GETK (95)

Wenn man den Befehl eingibt, wird die automatische Seitenumschaltung abgestellt und die Schildkröte nicht mehr ins Bild eingeblendet.

Als Ausgabe erhält man in D0.B den Wert, der vom Programm HIDE auf Port $60 der GDP geschrieben wird. Dieser Wert enthält den Code für die Lese- und Schreibseite sowie den XOR-Mode.

Einfaches Beispiel:

Es wird eine kurze Linie gezeichnet und dann die Schildkrötengrafik abgestellt.

```
START:
        MOVEQ     #100,D0
        MOVEQ     #!SCHREITE,D7      * 100 Schritte schreiten.
        TRAP      #1
        MOVEQ     #!HIDE,D7          * Schildkröte ausblenden.
        TRAP      #1
        RTS
```

Will man vermeiden, daß die Schildkröte noch auf der Bildseite 1 gezeichnet wird, weil man diese Seite auch verwenden will, empfiehlt sich folgende Vorgehensweise:

```
START:
        MOVEQ     #!FIRSTTIME,D7     * Schildkröte an.
        TRAP      #1
        MOVEQ     #!HIDE,D7          * Nicht zeigen.
        TRAP      #1
        MOVEQ     #100,D0
        MOVEQ     #!SCHREITE,D7      * 100 Schritte gehen.
        TRAP      #1
        MOVEQ     #!GRAPOFF,D7       * Jetzt ganz ausschalten.
        TRAP      #1
        RTS
```

Wenn man das Programm startet, erscheint eine stehende Linie auf dem Bildschirm. Wenn man F=Flip betätigt, werden beide Bildseiten umgeschaltet, ohne daß die Schildkröte erscheint.

TRAP-Nummer : **48**
Befehlsname : **SHOW**
Befehlsgruppe : Schildkrötengrafik.
Kurzbeschreibung : Schildkröte einblenden.

Eingaberegister : Keine.
Ausgaberegister : D0.B = Enthält Wert des GDP-Ports $60.
Zerstörte Register : Keine.
Ab Version : 3.1
Änderungen zu 4.3 : Nein.
Änderungen zu 6.1 : Nein.
Siehe auch : SCHREITE (1), DREHE (2), HEBE (3), SENKE (4), SET (7), SCHR16TEL (19), FIRSTTIME (36), GRAPOFF (39), HIDE (47), AUFXY (92), KORXY (93), AUFK (94), GETK (95)

Eine durch HIDE ausgeschaltete Bildschirmseitenumschaltung wird wieder rückgängig gemacht. Die Schildkröte erscheint wieder.

Als Ausgabe erhält man in D0.B den Wert, der vom Programm HIDE auf Port $60 der GDP geschrieben wird. Dieser Wert enthält den Code für die Schreib- und Leseseite sowie für den XOR-Mode.

Einfaches Beispiel:

Nachdem eine Linie gezogen ist, passiert nichts mehr, da HIDE und SHOW sich gegenseitig aufheben.

```
START:
        MOVEQ     #100,D0
        MOVEQ     #!SCHREITE,D7     * 100 Schritte gehen.
        TRAP      #1
        MOVEQ     #!HIDE,D7         * Abschalten.
        TRAP      #1
        MOVEQ     #!SHOW,D7         * Wieder anschalten.
        TRAP      #1
        RTS
```

Komplexeres Beispiel:

Alle Sekunde wechselt die Darstellung zwischen sichtbarer und unsichtbarer Schildkröte.

Das Programm kann durch Tastendruck abgebrochen werden.

```
START:
        MOVEQ     #100,D0
        MOVEQ     #!SCHREITE,D7     * 100 Schritte schreiten.
        TRAP      #1
SCHLEIFE:
        MOVEQ     #!HIDE,D7         * Schildkröte aus.
        TRAP      #1
        MOVEQ     #10,D0            * 1 Sekunde warten.
        MOVEQ     #!DELAY,D7
        TRAP      #1
        MOVEQ     #!SHOW,D7         * Schildkröte an.
        TRAP      #1
        MOVEQ     #10,D0            * 1 Sekunde warten.
        MOVEQ     #!DELAY,D7
        TRAP      #1
        MOVEQ     #!CSTS,D7
        TRAP      #1
        BEQ.S     SCHLEIFE          * Immer wieder, bis Taste gedrückt.
        RTS
```

TRAP-Nummer	: **49**		
Befehlsname	: **CRT**		
Befehlsgruppe	: Zeichenausgabe.		
Kurzbeschreibung	: CO2 auf Bildschirm lenken.		
Eingaberegister	: Keine.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: CLRSCREEN (20)	CO (21)	LO (22)
	SIZE (25)	CO2 (33)	LST (50)
	USR (51)	NIL (52)	SETPASS (55)
	CURSEIN(61)	CURSAUS (62)	CHAR (63)
	CURON (81)	CUROFF (82)	CRLF (99)
	GETLINE (100)	GETCURXY (101)	SETCURXY (102)
	LSTS (117)	CO2SER (129)	

Die CO2-Ausgabe ist auf verschiedene Ausgabegeräte umstellbar. Mit dem Aufruf von CRT wird die Ausgabe auf den Bildschirm gestellt. Der Befehl ist insbesondere für höhere Programmiersprachen wichtig, um die Ausgaben für die verschiedenen Peripheriegeräte zu steuern. Mit CRT wird aber meist die Ausgabe so zurückgesetzt, daß sie wieder auf dem Bildschirm erfolgt.

Einfaches Beispiel:

Ausgabe von CO2 wieder auf den Bildschirm setzen.

```
START:
        MOVEQ       #!CRT,D7        * Ausgabe auf den Bildschirm.
        TRAP        #1
        RTS
```

Einfaches Pascal-Beispiel:

```
BEGIN
        .....
        WRITELN(CHR(1),'E @CRT'); { Ausgabe auf Bildschirm }
        .....
END.
```

TRAP-Nummer	: **50**		
Befehlsname	: **LST**		
Befehlsgruppe	: Zeichenausgabe.		
Kurzbeschreibung	: CO2 auf Drucker lenken.		
Eingaberegister	: Keine.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: CLRSCREEN (20)	CO (21)	LO (22)
	SIZE (25)	CO2 (33)	CRT (49)
	USR (51)	NIL (52)	SETPASS (55)
	CURSEIN (61)	CURSAUS (62)	CHAR (63)
	CURON (81)	CUROFF (82)	CRLF (99)
	GETLINE (100)	GETCURXY (101)	SETCURXY (102)
	LSTS (117)	CO2SER (129)	

Die Ausgabe über den CO2-Befehl wird auf den Drucker umgesteuert. Damit kann man z. B. von PASCAL aus den Drucker über die normale WRITELN-Anweisung bedienen.

Einfaches Beispiel:

Die Ausgabe von CO2 wird auf den Drucker gelenkt.

```
START:
        MOVEQ       #1LST,D7
        TRAP        #1              * Umschalten auf den Drucker.
        RTS
```

Die Funktion ist genau wie im OPTIONEN-Menü, wenn man die Ausgabe des Assemblers auf den Drucker umstellt.

Einfaches Pascal-Beispiel:

Es wird auf dem Drucker und auf dem Bildschirm ausgegeben.

```
BEGIN
        .....
        WRITELN(CHR(1),'E @LST');
        WRITELN('Kommt auf dem Drucker')
        WRITELN(CHR(1),'E @CRT');
        WRITELN('Jetzt wieder auf dem Bildschirm');
        .....
END.
```

TRAP-Nummer	:	**51**
Befehlsname	:	**USR**
Befehlsgruppe	:	Zeichenausgabe.
Kurzbeschreibung	:	Schaltet CO2 auf Benutzervektor um.
Eingaberegister	:	Keine.
Ausgaberegister	:	Keine.
Zerstörte Register	:	Keine.
Ab Version	:	3.1
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	CLRSCREEN (20), CO (21), LO (22), SIZE (25), CO2 (33), CRT (49), LST (50), NIL (52), SETPASS (55), CURSEIN (61), CURSAUS (62), CHAR (63), CURON (81), CUROFF (82), CRLF (99), GETLINE (100), GETCURXY (101), SETCURXY (102), LSTS (117), CO2SER (129)

Die Ausgabe des CO2-Befehls wird auf die Benutzerschnittstelle umgelenkt.

Danach wird bei jedem Aufruf von CO2 auf den Benutzervektor USERCO (Adresse $24 relativ zum Variablenanfang) gesprungen, wo man einen Sprung auf eine eigene Routine einbauen kann (siehe CO2).

Einfaches Beispiel:

CO2 auf Benutzerschnittstelle umschalten.

```
START:
        MOVEQ       #!USR,D7
        TRAP        #1
        RTS
```

Einfaches Pascal-Beispiel:

Über USERCO wird ein kleiner Text ausgegeben.

```
BEGIN
        .....
        WRITELN(CHR(1),'E @USR');
        WRITELN('Kommt über die Benutzerroutine')
        .....
END.
```

Wenn man das Programm ausführt, erscheint die Ausgabe normalerweise auf dem Bildschirm. Um sie umzulenken, muß man zusätzlich den Benutzervektor im RAM durch ein kleines Assemblerprogramm umsteuern (siehe auch CO2- Beschreibung).

TRAP-Nummer	: **52**		
Befehlsname	: **NIL**		
Befehlsgruppe	: Zeichenausgabe.		
Kurzbeschreibung	: Schaltet CO2 auf Ausgabe ins Leere um.		
Eingaberegister	: Keine.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: CLRSCREEN (20)	CO (21)	LO (22)
	SIZE (25)	CO2 (33)	CRT (49)
	LST (50)	USR (51)	SETPASS (55)
	CURSEIN (61)	CURSAUS (62)	CHAR (63)
	CURON (81)	CUROFF (82)	CRLF (99)
	GETLINE (100)	GETCURXY (101)	SETCURXY (102)
	LSTS (117)	CO2SER (129)	

Alle Ausgaben mit dem Befehl CO2 werden unterdrückt, es sein denn, die Variable ERRFLAG wurde umbesetzt (siehe CO2-Beschreibung). Die Funktion ist vorwiegend für Übersetzer gedacht, bei denen man das Listing abschalten möchte, um die Übersetzung zu beschleunigen.

Einfaches Beispiel:

Die Ausgabe über CO2 wird unterdrückt.

```
START:
        MOVEQ       #!NIL,D7
        TRAP        #1
        RTS
```

Die Ausführung des Befehls entspricht der Eingabe im OPTIONEN-Menü 1 = NUR FEHLERAUSGABE.

TRAP-Nummer	: **53**		
Befehlsname	: **SETERR**		
Befehlsgruppe	: Assembler.		
Kurzbeschreibung	: Setzt den Fehlercode.		
Eingaberegister	: D0.W = Zu setzender Fehlercode.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: GETERR (54)	ASSEMBLE (65)	GETORG (68)
	SETORG (69)	ASSERR (120)	

Es wird die Variable ERRFLAG gesetzt. Im Register D0.W steht der Wert. Wenn ERRFLAG einen Wert ungleich 0 annimmt, wird eine z. B. mit NIL unterdrückte CO2-Ausgabe wieder freigegeben. Das ist besonders für Übersetzer gedacht, die im Fehlerfall Meldungen an die Console geben müssen.

Einfaches Beispiel:

CO2 durch NIL ausschalten, aber durch SETERR wieder anschalten.

```
START:
        MOVEQ   #!CLRSCREEN,D7      * Bildschirm löschen und für CO2 vorbereiten.
        TRAP    #1
        MOVEQ   #!NIL,D7            * Ausgabe ausschalten.
        TRAP    #1
        MOVEQ   #'A',D0
        MOVEQ   #!CI,D7             * Buchstabe ausgeben, erscheint nicht.
        TRAP    #1
        MOVEQ   #1,D0
        MOVEQ   #!SETERR,D7         * Fehler setzen, dadurch Ausgabe wieder an.
        TRAP    #1
        MOVEQ   #'B',D0
        MOVEQ   #!CO2,D7            * A ausgeben.
        TRAP    #1
        RTS
```

Der Bildschirm ist zunächst leer. Die erste Bildschirmausgabe wird unterdrückt. Das B erscheint dann aber auf dem Bildschirm, da die Ausgabe durch SETERR wieder eingeschaltet wird.

Wenn das Register D0.W nach einem Aufruf von SETERR einen Wert <> 0 hat und man jetzt SETERR mit einem Wert von 0 in D0.W aufruft, wird die Ausgabe wieder unterdrückt.

TRAP-Nummer	: **54**		
Befehlsname	: **GETERR**		
Befehlsgruppe	: Assembler.		
Kurzbeschreibung	: Liest den Fehlercode.		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.W = Fehlercode.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SETERR (53)	ASSEMBLE (65)	GETORG (68)
	SETORG (69)	ASSERR (120)	

Der Inhalt der Variablen ERRFLAG wird in das Register D0.W gelesen. Damit kann man sich über den Fehlerstatus informieren. ERRFLAG wird zum Beispiel automatisch auf den Wert 2 gesetzt, wenn man CI2 solange aufruft, bis über das Ende des Textes gelesen wird. Wichtig ist dabei, daß eine danach folgende Ausgabe mit CO2 auch dann erfolgt, wenn sie mit NIL unterdrückt war, und daß aller Text nach der Fehlerbedingung auf dem Bildschirm ausgegeben wird.

Einfaches Beispiel:

Der Fehlercode wird gelesen.

```
START:
        MOVEQ     #!GETERR,D7        * Fehlercode nach D0.W.
        TRAP      #1
        RTS
```

TRAP-Nummer	: **55**
Befehlsname	: **SETPASS**
Befehlsgruppe	: Zeichenausgabe.
Kurzbeschreibung	: Setzt die Laufnummer und schaltet ggf. die Ausgabe ein.
Eingaberegister	: D0.W = Wert der Laufnummer.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 3.1
Änderungen zu 4.3	: Nein.
Änderungen zu 6.1	: Nein.
Siehe auch	: CLRSCREEN (20), CO (21), LO (22), SIZE (25), CO2 (33), CRT (49), LST (50), USR (51), NIL (52), CURSEIN (61), CURSAUS (62), CHAR (63), CURON (81), CUROFF (82), CRLF (99), GETLINE (100), GETCURXY (101), SETCURXY (102), LSTS (117), CO2SER (129)

Auch ein Befehl, der für Übersetzer interessant ist. Damit wird die Variable PASSFLAG gesetzt. Der Wert steht zuvor in Register D0.W. Dabei ist wichtig, daß eine Ausgabe über CO2 nur dann erfolgt, wenn der Wert in PASSFLAG = 2 ist. Sonst wird die Ausgabe unterdrückt. Will man beim ersten Durchlauf einer Übersetzung noch keine Ausgabe des Listings auf dem Bildschirm haben, muß man PASSFLAG = 1 setzen. Erst beim zweiten Durchlauf wird z. B. beim Assembler eine Ausgabe erzeugt, wenn man PASSFLAG durch den Befehl SETPASS auf 2 setzt. Der Wert 2 wird übrigens vom Grundprogramm automatisch voreingestellt, um die Ausgabe freizuschalten.

Einfaches Beispiel:

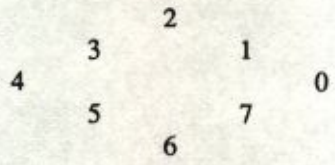
Der Buchstabe A wird nicht ausgegeben. Der Buchstabe B wird ausgegeben.

```
START:
        MOVEQ   #!CLRSCREEN,D7      * Bildschirm löschen für CO2.
        TRAP    #1
        MOVEQ   #1,D0               * Laufnummer 1.
        MOVEQ   #!SETPASS,D7
        TRAP    #1
        MOVEQ   #'A',D0
        MOVEQ   #!CO2,D7            * Buchstabe ausgeben, wird aber unterdrückt.
        TRAP    #1
        MOVEQ   #2,D0               * Laufnummer 2.
        MOVEQ   #!SETPASS,D7
        TRAP    #1
        MOVEQ   #'B',D0             * Buchstabe wird ausgegeben.
        MOVEQ   #!CO2,D7
        TRAP    #1
        RTS
```

TRAP-Nummer	: **57**
Befehlsname	: **FIGUR**
Befehlsgruppe	: Figurgrafik.
Kurzbeschreibung	: Figur mit XY-Vergrößerung zeichnen.
Eingaberegister	: D0.B = Größe der Figur. D1.W = X-Position. D2.W = Y-Position. A0.L = Adresse der Figurdaten.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 3.1
Änderungen zu 4.3	: Es wird nicht mehr der Kurzvektor-Befehl verwendet.
Änderungen zu 6.1	: Nein.
Siehe auch	: FIGURXY (5) SETFIG (58)

Damit kann man bewegliche Figuren definieren, die man über den Bildschirm bewegen kann. Im Register D0.B wird dazu die Größe der Figur angeben. Ist der Wert = 1, erhält man die kleinste Figur. Mit höheren Werten kann man größere Figuren ausgeben. Im Register D1.W steht die X-Koordinate und im Register D2.W die Y-Koordinate. Das Register A0.L enthält die Adresse der Figurendefinition.

Um eine Figur zeichnen zu können, muß sie zunächst definiert werden. Dazu wird die Figur in einzelne Vektoren zerlegt und aus ihnen zusammengesetzt. Die Codierung ist wie folgt:



Eine 0 bedeutet beispielsweise einen Bildpunkt nach rechts, eine 1 einen Bildpunkt nach links oben in einem Winkel von 45 Grad.

Ferner gibt es:

8 = Schreibstift heben,
9 = Schreibstift senken,
10 = Ende der Tabelle.

Wenn man bei einem erneuten Aufruf die Größe oder die X-Koordinate oder die Y-Koordinate oder die Adresse der Figur ändert, wird zunächst die alte Figur gelöscht und dann die Neue gezeichnet. Dadurch ergibt sich der Bewegungseffekt. Wenn man als neue Größe den Wert 0 im Register D0.W angibt, wird nur die alte Figur gelöscht, aber keine Neue gezeichnet. Bei diesem Befehl kann nur eine gemeinsame Größe für X- und Y-Vergrößerung angegeben werden. Möchte man verschiedene Größen haben, muß man den FIGURXY-Befehl verwenden, der ansonsten die gleiche Funktion hat wie der FIGUR-Befehl.

Einfaches Beispiel:

Es wird ein deformiertes Achteck gezeichnet.

```
START:
        LEA     FIGUR(PC),A0        * Adresse der Figur.
        MOVEQ   #20,D0              * Vergrößerung.
        MOVE.W  #220,D1             * X-Position.
        MOVE.W  #150,D2             * Y-Position.
        MOVEQ   #!FIGUR,D7          * Figur zeichnen.
        TRAP    #1
        RTS
FIGUR:
        DC.B    6,5,0,7,2,2,1,4,3,6,10
```

Komplexere Beispiele:

Das deformierte Achteck bewegt sich von links nach rechts über den Bildschirm.

```
START:
        MOVEQ       #-20,D1             * X-Anfangsposition.
        MOVEQ       #100,D2             * Y-Position.
        LEA         FIGUR(PC),A0        * Adresse der Daten für die Figur.
SCHLEIFE:
        MOVEQ       #3,D0               * Vergrößerung.
        MOVEQ       #!FIGUR,D7          * Figur zeichnen.
        TRAP        #1
SYNC:
        MOVEQ       #!SYNC,D7           * 20 ms warten.
        TRAP        #1
        BEQ.S       SYNC
        ADDQ        #3,D1               * Neue X-Position.
        CMP         #550,D1             * Endposition.
        BNE.S       SCHLEIFE
        RTS
FIGUR:
        DC.B        6,5,0,7,2,2,1,4,3,6,10
```

Man kann die Bewegung auch dadurch erreichen, daß man ständig die Adresse der sich bewegenden Figuren verändert.

Ein Pack-Man, der den Mund immer auf und zu macht, bewegt sich über den Bildschirm.

```
START:
        MOVEQ       #-40,D1             * X-Position.
        MOVE        #128,D2             * Y-Position.
        LEA         FIGUR1(PC),A0       * Adresse Figur 1.
        LEA         FIGUR2(PC),A1       * Adresse Figur 2.
        MOVEQ       #290/8,D4           * Anzahl der Durchläufe.
SCHLEIF0:
        MOVEQ       #8-1,D3             * Anzahl der Durchläufe.
SCHLEIF1:
        MOVEQ       #!SYNC,D7           * Ohne SYNC wäre es zu schnell.
        TRAP        #1
        BEQ.S       SCHLEIF1
        MOVEQ       #4,D0               * Größe der Figur.
        MOVEQ       #!FIGUR,D7          * Neue Figur ausgeben und alte löschen.
        TRAP        #1
        ADDQ        #2,D1               * Neue X-Position.
        DBRA        D3,SCHLEIF1         * Innere Schleife.
        EXG.L       A0,A1               * Figurenadresse tauschen, dadurch Bewegung.
        DBRA        D4,SCHLEIF0         * Äußere Schleife.
        RTS
FIGUR1:
        DC.B        0,0,0,0,1,2,2,4,4,0,0,2,2
        DC.B        3,4,4,8,6,9,6,4,2,0,8,2,9
        DC.B        4,4,5,6,6,6,6,7,10
FIGUR2:
        DC.B        0,0,0,0,1,2,4,4,2,2,0,0,2
        DC.B        3,4,4,8,6,9,6,4,2,0,8,2,9
        DC.B        4,4,5,6,6,6,6,7,10
```

TRAP-Nummer	: **58**	
Befehlsname	: **SETFIG**	
Befehlsgruppe	: Figurgrafik.	
Kurzbeschreibung	: Figur festsetzen (einfrieren).	
Eingaberegister	: Keine.	
Ausgaberegister	: Keine.	
Zerstörte Register	: Keine.	
Ab Version	: 3.1	
Änderungen zu 4.3	: Nein.	
Änderungen zu 6.1	: Nein.	
Siehe auch	: FIGURXY (5)	FIGUR (57)

Damit kann man die zuletzt mit FIGUR oder FIGURXY gezeichnete Figur auf dem Bildschirm fixieren. Wenn man anschließend FIGUR oder FIGURXY mit neuen Parametern aufruft, bleibt die alte Figur auf dem Bildschirm.

Komplexeres Beispiel:

Es werden nach einander zwei Figuren ausgegeben.

```
START:
        LEA     FIGUR(PC),A0        * Adresse der Figur.
        MOVEQ   #4,D0               * Größe.
        MOVEQ   #10,D1              * X-Position.
        MOVEQ   #100,D2             * Y-Position.
        MOVEQ   #!FIGUR,D7          * Figur ausgeben.
        TRAP    #1
        MOVEQ   #!SETFIG,D7         * Figur festsetzen.
        TRAP    #1
        MOVE    #180,D2             * Neue Y-Position.
        MOVEQ   #!FIGUR,D7          * Figur ausgeben, ohne die alte zu löschen.
        TRAP    #1
        RTS
FIGUR:
        DC.B    0,0,0,0,1,2,2,4,4,0,0,2,2
        DC.B    3,4,4,8,6,9,6,4,2,0,8,2,9
        DC.B    4,4,5,6,6,6,6,7,10
```

TRAP-Nummer	:	59
Befehlsname	:	**GETRAM**
Befehlsgruppe	:	System-Routinen.
Kurzbeschreibung	:	Sucht Anfang und Ende von RAM-Bereichen.
Eingaberegister	:	A0.L = Adresse für Begin der Suche.
Ausgaberegister	:	D0.L = Suchstartadresse auf 1 Kbyte-Grenze. A0.L = Erste gefundene Nicht-Ram-Zelle. A1.L = Erste gefundene RAM-Zelle. Carry = Anzeige, ob Suche erfolgreich war.
Zerstörte Register	:	Keine.
Ab Version	:	3.1
Änderungen zu 4.3	:	RAM-Bereiche sind eindeutig und werden nicht mehrmals erkannt. Der Such-Bereich ist an die CPU angepasst.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	GETBASIS (89), GETVAR (94), SETA5 (91), GETVERS (97), GETSN (98), GRUND (124), SUCHBIBO (137), SYSTEM (139)

Mit diesem Befehl kann man einen Speicherbereich ermitteln, in dem sich RAM befindet. Die Suche nach RAM beginnt bei der Adresse, die im Register A0.L steht. Dabei steht nach dem Aufruf im Register A0.L die Adresse der nach dem Bereich folgenden Nicht-Ram-Zelle und im Register A1.L die Adresse der ersten gefunden RAM-Zelle. Ein Carry=1 wird geliefert, falls kein RAM-Bereich gefunden wurde. Ansonsten ist das Flag zurückgesetzt.

Bevor die Suche beginnt, wird die angegebene Adresse auf eine 1K-Seitengrenze gebracht. Die Suche erfolgt bis zum maximalen Bereich der CPU (68008 = 1 Mbyte - 1 Kbyte / 68000 = 2 Mbyte - 1 Kbyte / 68020 = 4 Mbyte). Der Inhalt des geprüften Bereichs wird nicht zerstört.

Wenn man alle RAM-Bereiche feststellen will, muß man GETRAM so lange aufrufen, bis ein Carry erscheint. Man merke sich dazwischen jeweils die Speicherbereiche. A0.L steht nach jedem Aufruf so, daß es vom nächsten GETRAM wieder direkt verwendet werden kann.

Die Routine ist seit der Version 6.0 schneller geworden. Deshalb wird jetzt beim 68008 der Speicherbereich der COL256 nicht mehr gefunden. Außerdem ist die Routine nicht für einen RAM-Check gedacht. Sie diente ursprünglich nur dazu, den RAM-Bereich für den Stack zu finden.

Einfaches Beispiel:

Es wird der erste RAM-Bereich gesucht.

```
START:
        SUBA.L      A0,A0               * Von Adresse 0 an suchen.
        MOVEQ       #!GETRAM,D7         * A0, A1 sind Ergebnis.
        TRAP        #1
        RTS
```

Mit der Einzelschrittfunktion kann man das vorherige Ergebnis überprüfen. Dazu startet man das Programm im Einzelschritt (vorher im OPTIONEN-Menü eventuell DEBUG anschalten). Nach dem Aufruf des Befehls GETRAM erscheinen in den Registern A0.L und A1.L die Werte.

Komplexeres Beispiel:

Es werden alle RAM-Bereiche über die CO2-Routine ausgegeben.

```
START:
        MOVEQ       #!CLRSCREEN,D7      * Bildschirm für CO2 vorbereiten.
        TRAP        #1
        SUBA.L      A0,A0               * Von Adresse 0 an suchen.
```

```
SCHLEIFE:
        MOVEQ       #!GETRAM,D7     * RAM-Bereich suchen.
        TRAP        #1
        BCS.S       ENDE            * Keiner mehr gefunden.
        MOVEA.L     A0,A2           * A0 merken.
        LEA         BUFFER(PC),A0
        MOVE.L      A1,D0           * Anfangsadresse RAM nach A0.L.
        MOVEQ       #!PRINT8X,D7    * Anfangsadresse ausgeben.
        TRAP        #1
        MOVE.B      #' ',(A0)+
        MOVE.B      #'-',(A0)+      * Trennzeichen.
        MOVE.B      #' ',(A0)+
        MOVE.L      A2,D0           * Adresse erste Nicht-RAM-Zelle.
        SUBQ.L      #1,D0           * Jetzt letzte RAM-Adresse.
        MOVEQ       #!PRINT8X,D7    * Ausgabe.
        TRAP        #1
        LEA         BUFFER(PC),A0
CO2SCHLEIF:
        MOVE.B      (A0)+,D0
        BEQ.S       SPRUNG0
        MOVEQ       #!CO2,D7        * Bis zur Endenull ausgeben.
        TRAP        #1
        BRA.S       CO2SCHLEIF
SPRUNG0:
        MOVEQ       #!CRLF,D7       * Zeilenvorschub.
        TRAP        #1
        MOVEA.L     A2,A0           * A0 zurück.
        BRA.S       SCHLEIFE        * Weiter suchen.
ENDE:
        RTS
BUFFER:
        DS.B        20              * Buffer für Zeichenablage.
```

TRAP-Nummer	:	**60**
Befehlsname	:	**AUTOFLIP**
Befehlsgruppe	:	Grafik.
Kurzbeschreibung	:	Automatische Bildseitenumschaltung.
Eingaberegister	:	Keine.
Ausgaberegister	:	Keine.
Zerstörte Register	:	D0
Ab Version	:	3.1
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	MOVETO (8), DRAWTO (9), CLR (16), CLPG (17), WAIT (18), CMD (26), NEWPAGE (27), SETFLIP (34), SETPEN (37), ERAPEN (38), CMDPRINT (40), SETXOR (77), GETXOR (78), SETCOLOR (79), GETCOLOR (80), GETXY (103), HARDCOPY (125), GRAFIK (126), GDPVERS (127)

Das Unterprogramm steuert die automatische Bildseitenumschaltung. Dabei ist die Umschaltrate von den Variablen FLIP und FLIP1 abhängig. Diese Variablen kann man mit dem Befehl SETFLIP belegen. Sie werden aber auch automatisch durch die Schildkrötenbefehle besetzt. Alle 20 ms werden die Variablen abgefragt. FLIP steuert dabei die Umschaltung zwischen zwei Bildseiten. Ist FLIP = 1, wird alle 20 ms gewechselt. Dabei wird entweder zwischen Seite 0 und 1 oder zwischen Seite 2 und 3 umgeschaltet, je nach dem, welche Leseseite zuvor aktiv war. Die Schreibseite wird nicht beeinflußt. Wenn FLIP1 = 1 ist und FLIP = 0, wird alle 20 ms zwischen den vier Bildseiten umgeschaltet. Dabei wird von 0 auf 1, von 2 auf 3 und von 3 auf 0 geschaltet. Ein Wert größer 1 bewirkt eine Multiplikation mit jeweils 20 ms. Im Sekundentakt umschalten bedeutet FLIP = 50 oder FLIP1 = 50.

AUTOFLIP wird sonst z. B. in CI und bei Anwendung der Schildkrötenbefehle automatisch aufgerufen. Man muß AUTOFLIP nur dann verwenden, wenn man z. B. in einer großen Schleife Berechnungen durchführt oder lange Zeit nicht auf CI oder die Schildkrötenbefehle zurückgreift, aber dennoch eine Seitenumschaltung erreichen möchte.

Einfaches Beispiel:

Die Seitenumschaltung wird bei der Benutzung von AUTOFLIP aufrecht erhalten.

```
START:
        MOVEQ     #100,D0
        MOVEQ     #!SCHREITE,D7      * Linie mit Schildkröte ziehen.
        TRAP      #1
SCHLEIFE:
        MOVEQ     #!AUTOFLIP,D7      * Automatische Seitenumschaltung.
        TRAP      #1
        MOVEQ     #!CSTS,D7          * Tastatur abfragen.
        TRAP      #1
        BEQ.S     SCHLEIFE           * Ende bei Druck beliebiger Taste.
        RTS
```

Die Schildkröte bleibt weiterhin eingeblendet. Wenn man AUTOFLIP nicht verwendet, ist nur eine Seite sichtbar, jedoch ist nicht einmal definiert, welche. Mit NEWPAGE könnte man zumindest das noch definieren.

TRAP-Nummer	: **61**		
Befehlsname	: **CURSEIN**		
Befehlsgruppe	: Zeichenausgabe.		
Kurzbeschreibung	: Cursor einblenden.		
Eingaberegister	: Keine.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Ja.		
Änderungen zu 6.1	: Cursordarstellung ist auch bei HARDSCROLL möglich.		
Siehe auch	: CLRSCREEN (20)	CO (21)	LO (22)
	SIZE (25)	CO2 (33)	CRT (49)
	LST (50)	USR (51)	NIL (52)
	SETPASS (55)	CURSAUS (62)	CHAR (63)
	CURON (81)	CUROFF (82)	CRLF (99)
	GETLINE (100)	GETCURXY (101)	SETCURXY (102)
	LSTS (117)	CO2SER (129)	

Einschalten des Cursors. Normalerweise geschieht das automatisch beim Aufruf der Routine CI. Durchläuft aber das Programm eine große Schleife und will man den Cursor auf dem Bildschirm sehen, muß man die Routine verwenden. Bevor ein Aufruf von CO, CO2 oder CI erfolgt, muß man den Cursor wieder mit CURSAUS ausschalten. Ferner muß man mit AUTOFLIP die Seitenumschaltung aufrechterhalten (in DELAY z. B. wird automatisch die Routine AUTOFLIP aufgerufen). Es ist jetzt auch möglich, bei normaler Bildschirmdarstellung mit dem Cursor den HARDSCROLL zu verwenden. Dazu muß nichts weiter eingestellt werden (siehe auch CO2).

Einfaches Beispiel:

Der Cursor blinkt 5 Sekunden lang und wird dann ausgeschaltet. Abbruch des Programms durch Tastendruck.

```
START:
        MOVEQ   #!CLRSCREEN,D7      * Bildschirm löschen und Cursor aktivieren.
        TRAP    #1
        MOVEQ   #!CURSEIN,D7        * Cursor darstellen.
        TRAP    #1
        MOVEQ   #50,D0              * 5 Sekunden warten.
        MOVEQ   #!DELAY,D7
        TRAP    #1
        MOVEQ   #!CURSAUS,D7        * Cursor ausschalten.
        TRAP    #1
SCHLEIFE:
        MOVEQ   #!CSTS,D7           * Auf Tastendruck warten.
        TRAP    #1
        BEQ.S   SCHLEIFE
        RTS
```

Wenn man die Schleife wegläßt, wird der Cursor wieder sichtbar, denn bei der Rückkehr in das Grundprogramm wird der Cursor immer eingeschaltet.

TRAP-Nummer	: **62**		
Befehlsname	: **CURSAUS**		
Befehlsgruppe	: Zeichenausgabe.		
Kurzbeschreibung	: Cursor ausblenden.		
Eingaberegister	: Keine.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Ja.		
Änderungen zu 6.1	: Cursor kann auch bei HARDSCROLL ausgeschaltet werden.		
Siehe auch	: CLRSCREEN (20)	CO (21)	LO (22)
	SIZE (25)	CO2 (33)	CRT (49)
	LST (50)	USR (51)	NIL (52)
	SETPASS (55)	CURSEIN (61)	CHAR (63)
	CURON (81)	CUROFF (82)	CRLF (99)
	GETLINE (100)	GETCURXY (101)	SETCURXY (102)
	LSTS (117)	CO2SER (129)	

Der Cursor, der mit CURSEIN eingeschaltet war, wird wieder abgestellt.

Normalerweise wird z. B. bei CSTS kein Cursor auf den Bildschirm gebracht. Es gibt aber Fälle, in denen man in einer Schleife durch CSTS abfragt, ob ein Zeichen angekommen ist, und es dann erst mit CI einliest, was z. B. das 68000 FORTH von Laboratory Microsystems für CP/M68K tut.

Der Befehl kann genau wie CURSEIN mit HARDSCROLL betrieben werden.

Komplexeres Beispiel:

Es erfolgt eine manuelle Ein- und Ausschaltung des Cursors. In der Warteschleife könnte jetzt zum Beispiel im Hintergrund ein Druckprogramm laufen. Der Abbruch des Programms erfolgt durch Eingabe eines RETURN.

```
START:
        MOVEQ       #!CLRSCREEN,D7      * Bildschirm löschen.
        TRAP        #1
SCHLEIFE:
        MOVEQ       #!CURSEIN,D7        * Cursor einschalten.
        TRAP        #1
WARTE:
        MOVEQ       #!AUTOFLIP,D7       * Automatische Seitenumschaltung.
        TRAP        #1
        MOVEQ       #!CSTS,D7           * Zeichen da?
        TRAP        #1
        BEQ.S       WARTE               * Nein, dann warten.
        MOVEQ       #!CURSAUS,D7        * Cursor ausschalten.
        TRAP        #1
        MOVEQ       #!CI,D7             * Zeichen holen.
        TRAP        #1
        CMP.B       #$D,D0              * RETURN bricht ab.
        BEQ.S       ENDE
        MOVEQ       #!CO2,D7            * Zeichen ausgeben.
        TRAP        #1
        BRA.S       SCHLEIFE            * Wiederholen.
ENDE:
        RTS
```

Wichtig ist, daß man den Cursor ausschaltet, bevor man die CO2-Ausgabe aktiviert, da sonst z.B. bei Scroll das alte Zeichen falsch auf dem Bildschirm zurückgeschrieben wird.

TRAP-Nummer	: **63**		
Befehlsname	: **CHAR**		
Befehlsgruppe	: Zeichenausgabe.		
Kurzbeschreibung	: Gibt ein Zeichen aus (keine Steuerzeichen).		
Eingaberegister	: D0.B = Auszugebendes Zeichen.		
Ausgaberegister	: Keine.		
Zerstörte Register	: D0/A0-A2		
Ab Version	: 3.1		
Änderungen zu 4.3	: Ausgabe mit HARDSCROLL ist möglich.		
Änderungen zu 6.1	: Es kann immer mit HARDSCROLL gearbeitet werden.		
Siehe auch	: CLRSCREEN (20)	CO (21)	LO (22)
	SIZE (25)	CO2 (33)	CRT (49)
	LST (50)	USR (51)	NIL (52)
	SETPASS (55)	CURSEIN (61)	CURSAUS (62)
	CURON (81)	CUROFF (82)	CRLF (99)
	GETLINE (100)	GETCURXY (101)	SETCURXY (102)
	LSTS (117)	CO2SER (129)	

Ausgabe eines Zeichens, das im Register D0.B steht, auf den Bildschirm. Dabei wird wie bei CO verfahren, jedoch sind Steuerfunktionen nicht verfügbar. Damit können alle darstellbaren Zeichen von 0 bis $FF verwendet werden. Da es sich um eine interne Funktion des Grundprogramms handelt, ist bei der Anwendung Vorsicht geboten. Man sollte möglichst immer CO oder CO2 verwenden.

Auch CHAR kann wie CO und CO2 jetzt immer mit HARDSCROLL verwendet werden.

Einfaches Beispiel:

Es wird genau wie mit CO der Buchstabe A ausgegeben.

```
START:
        MOVEQ   #!CLRSCREEN,D7      * Bildschirm löschen.
        TRAP    #1
        MOVEQ   #'A',D0
        MOVEQ   #!CHAR,D7           * Buchstaben ausgeben.
        TRAP    #1
        RTS
```

TRAP-Nummer : **64**
Befehlsname : **PROGZGE**
Befehlsgruppe : Textausgabe.
Kurzbeschreibung : Gibt ein frei definierbares Zeichen aus.

Eingaberegister : A0.L = Adresse der Daten des Zeichens.
Ausgaberegister : Keine.
Zerstörte Register : Keine.
Ab Version : 3.1
Änderungen zu 4.3 : Nein.
Änderungen zu 6.1 : Nein.
Siehe auch : WRITELF (6) WRITE (10)

Mit dem Befehl ist es möglich, einen frei programmierbaren Zeichengenerator zu verwenden. Man definiert sich dazu in einer 8 * 5 Matrix die Zeichen und kann sie mit diesem Befehl in der aktuellen Schriftgröße auf den Bildschirm ausgeben.

Das Register A0.L zeigt auf den Datenblock für das Zeichen. Die Ausgabe erfolgt an der aktuellen Koordinatenstelle, die man mit MOVETO einstellen muß. Als Größe wird die aktuelle Schriftgröße verwendet, die gerade im Register 3 ($FF73) des GDP steht. Wenn man die Position des Zeichens und die Größe gleich mit angeben will, muß man die entsprechende Funktion des Befehls GRAFIK verwenden.

Nach dem Aufruf befindet sich die aktuelle Schreibposition des GDP genau an der richtigen Stelle für das nächste Zeichen, sodaß mehrere PROGZGE-Aufrufe nach einander erfolgen können.

Im Datenblock wird das Zeichen von links nach rechts abgelegt. Dabei entspricht Bit 7 der untersten Zeile und Bit 0 der obersten Zeile eines Zeichens. Wenn ein Bit auf 1 gesetzt ist, erscheint es als heller Punkt auf dem Bildschirm. 8 * 5 Matrix bedeutet, daß das Zeichen in ein Rechteck eingetragen wird mit einer Höhe von 8 und einer Breite von 5 Bildpunkten. Im nachstehenden Beispiel sieht die Matrix für das Ohm-Zeichen so aus:

Breite des Zeichen = Byte	1	2	3	4	5
Höhe des Zeichen = Bit					
7	0	1	1	1	0
6	1	0	0	0	1
5	1	0	0	0	1
4	1	0	0	0	1
3	1	0	0	0	1
2	1	1	0	1	1
1	0	1	0	1	0
0	1	1	0	1	1

Setzt man für die Einsen jeweils ein * und an die Stelle der Nullen ein Leerzeichen, wird das Ohmzeichen deutlich:

Breite des Zeichen = Byte	1	2	3	4	5
Höhe des Zeichen = Bit					
7		*	*	*	
6	*				*
5	*				*
4	*				*
3	*				*
2	*	*		*	*
1		*		*	
0	*	*		*	*

Wie das Ohm-Zeichen für den PROGZGE-Befehl zu definieren ist, ergibt sich aus der Eingabe ZEICHEN im folgenden Beispiel.

Komplexeres Beispiel:

Das OHM-Zeichen wird an der Stelle 100,125 in der Größe $EE ausgegeben.

```
START:
        MOVE.W      #225,D1          * X-Position.
        MOVEQ       #100,D2          * Y-Position.
        MOVEQ       #!MOVETO,D7      * Koordinaten setzen.
        TRAP        #1
        MOVE        #!SYSTEM,D7      * System-Informationen holen.
        TRAP        #1
        AND         #7,D0            * Nur CPU-INFORMATION lassen.
        MULS        #$FF73,D0        * GDP-Register 3 (SIZE).
        MOVEA.L     D0,A0
        MOVE.B      #$EE,(A0)        * Größe einstellen.
        LEA         ZEICHEN(PC),A0   * Adresse der Daten.
        MOVEQ       #!PROGZGE,D7     * Zeichen ausgeben.
        TRAP        #1
        RTS
ZEICHEN:
        DC.B        %10111110        * Ohm-Zeichen.
        DC.B        %11100001
        DC.B        %00000001
        DC.B        %11100001
        DC.B        %10111110
```

Der Bildaufbau erfolgt hier relativ langsam. Deshalb sollte man einer vektoriellen Darstellung den Vorzug geben, wann immer das möglich ist.

Die Routine wird im Grundprogramm für die Darstellung der deutschen Sonderzeichen verwendet, da ihre vektorielle Darstellung angesichts der vielen denkbaren Schriftgrößen zu aufwendig wäre.

TRAP-Nummer	:	**65**
Befehlsname	:	**ASSEMBLE**
Befehlsgruppe	:	Assembler.
Kurzbeschreibung	:	Der Assembler wird aufgerufen.
Eingaberegister	:	Keine.
Ausgaberegister	:	Carry = Gesetzt, wenn Assembler abgebrochen wurde.
Zerstörte Register	:	Alle außer A5.
Ab Version	:	3.1
Änderungen zu 4.3	:	Der HARDSCROLL wird - wenn möglich - eingeschaltet. Die Zeichen werden nicht mehr über CI2, sondern direkt aus dem RAM gelesen. Eine Macrotabelle wird direkt hinter dem Editortext angelegt. Beim ersten Durchlauf wird nichts im RAM abgelegt.
Änderungen zu 6.1	:	Abbruch jetzt mit ESC.
Siehe auch	:	SETERR (53) GETERR (54) GETORG (68) PUTORG (69) ASSERR (120)

Damit kann man den Assembler als Unterprogramm aufrufen. Der Befehl ist insbesondere für Übersetzer (Compiler) interessant. Man kann jedoch auch andere Effekte damit erreichen. So ist es möglich, Assemblercode als Ergebnis eines Programms zu erzeugen, automatisch zu übersetzen und das Programm zu starten.

Die Übersetzung beginnt beim aktuellen Textstart (STXTXT), der ggf. mit dem Befehl PUTSTX verändert werden kann.

Der Programmcode wird an der Standard-ORG-Adresse abgelegt, sofern diese nicht mit dem Befehl ORG verändert wurde.

Nach einem Abbruch des Assemblerlaufs mit ESC ist das CARRY-Flag gesetzt. Außerdem wird der Fehlerzähler (ERRCNT) um eins erhöht. Eine ausführliche Beschreibung des Assemblers bietet Kapitel 3.2.2.

Komplexeres Beispiel:

Beim Start des Programms wird zunächst der Assembler aufgerufen, dann das als Text angegebene Programm übersetzt und anschließend gestartet.

```
START:
        MOVEQ     #!GETSTX,D7          * Aktuelle Textadresse.
        TRAP      #1
        MOVE.L    D0,-(A7)             * Merken.
        LEA       TEXT(PC),A0
        MOVE.L    A0,D0
        MOVEQ     #!PUTSTX,D7          * Neue Textadresse.
        TRAP      #1
        MOVEQ     #!ASSEMBLE,D7        * Übersetzen.
        TRAP      #1
        MOVEQ     #50,D0               * 5 Sekunden warten.
        MOVEQ     #!DELAY,D7
        TRAP      #1
        MOVEQ     #!CLR,D7             * Bildschirm löschen.
        TRAP      #1
        BSR       NEU                  * Programm ausführen.
        MOVE.L    (A7)+,D0
        MOVEQ     #!PUTSTX,D7          * Alte Textadresse.
        TRAP      #1
        RTS
TEXT:
        DC.B      'ORG NEU',$D,$A
        DC.B      ' MOVE #100,D0',$D,$A
        DC.B      ' JSR @SCHREITE',$D,$A
        DC.B      ' RTS',$D,$A,0
        DS        0
        DS.W      200                  * Platz für Macrotabelle.
NEU:
        DS.W      100                  * Hier kommt Code hin.
```

TRAP-Nummer	: **66**
Befehlsname	: **GETSTX**
Befehlsgruppe	: Editor.
Kurzbeschreibung	: Aktuelle Textadresse wird gelesen.
Eingaberegister	: Keine.
Ausgaberegister	: D0.L = Textadresse.
Zerstörte Register	: Keine.
Ab Version	: 3.1
Änderungen zu 4.3	: Nein.
Änderungen zu 6.1	: Nein.
Siehe auch	: EDIT(56) PUTSTX (67)

Damit kann man die aktuelle Textadresse des Editors ermitteln. Diese Adresse ist in der Variablen STXTXT gespeichert und wird nach dem Aufruf im Register D0.L übergeben.

Einfaches Beispiel:

Die aktuelle Textstartadresse wird um 5 KByte nach hinten verlegt.

```
START:
        MOVEQ     #!GETSTX,D7      * Alte Textadresse nach D0.L.
        TRAP      #1
        ADD.L     #1024*5,D0       * 5 KByte nach hinten verschieben.
        MOVEQ     #!PUTSTX,D7      * Neue Adresse setzen.
        TRAP      #1
        RTS
```

Komplexeres Beispiel:

Die aktuelle Textadresse des Editors wird kurzfristig auf die Adresse von NEU gelegt. Dort kann ein Text eingegeben werden. Beim Verlassen des Editors wird wieder die alte Adresse eingestellt.

```
START:
        MOVEQ     #!GETSTX,D7      * Alte Textadresse holen.
        TRAP      #1
        MOVE.L    D0,-(A7)         * Merken.
        MOVE.L    #NEU,D0          * Neue Textadresse einstellen.
        MOVEQ     #!PUTSTX,D7
        TRAP      #1
        MOVEQ     #!EDIT,D7        * Editor aufrufen.
        TRAP      #1
        MOVE.L    (A7)+,D0         * Alte Adresse holen und
        MOVEQ     #!PUTSTX,D7      * wieder einstellen.
        TRAP      #1
        RTS
        DC.B      1024             * Platz für Macro-Tabelle
NEU:                               * Leerer Editor-Bildschirm ab hier.
```

TRAP-Nummer	: **67**	
Befehlsname	: **PUTSTX**	
Befehlsgruppe	: Editor.	
Kurzbeschreibung	: Es wird eine neue Textadresse (alter Text) gesetzt.	
Eingaberegister	: D0.L = Adresse des Textes.	
Ausgaberegister	: A0.L = Zeigt direkt hinter den Text.	
Zerstörte Register	: Keine.	
Ab Version	: 3.1	
Änderungen zu 4.3	: Nein.	
Änderungen zu 6.1	: Nein.	
Siehe auch	: EDIT (56)	GETSTX (66)

Damit kann man den Textanfang umdefinieren, und zwar genau so wie im OPTIONEN-Menü. Im Register D0.L steht die neue Adresse. Das Textende wird automatisch gesetzt. Dabei gibt eine 0 (Code 00) im Speicher an, wo sich das Textende befindet. Das Programm sucht nach der 0. Daher Achtung: Wenn man sie vergißt, kann die Suche unter Umständen lange dauern. Als Ausgabe gibt das Programm in A0.L einen Zeiger direkt auf das nächste Zeichen hinter dem Text zurück.

Einfaches Beispiel:

Die Textstartadresse wird 64 KByte hinter das Grundprogramm gelegt.

```
START:
        LEA     $10000(A5),A0       * Neue Textadresse.
        MOVE.L  A0,D0               * Nach D0.L.
        CLR.B   (A0)                * Endemarkierung.
        MOVEQ   #!PUTSTX,D7         * Adresse einstellen.
        TRAP    #1
        RTS
```

Wenn man das Programm startet, passiert zunächst nichts Aufregendes. Erst wenn man anschließend in den Editor zurückgeht, fehlt der Text. In der Status-Zeile sieht man aber, daß der Textanfang auf einer neuen Adresse liegt. Wenn man den alten Programmtext wieder sehen will, kann man dies z. B. mit dem OPTIONEN-Menü durch Eingabe des TEXTSTART (ALT), d. h. mit der alten Adresse des Textes tun.

TRAP-Nummer	: **68**		
Befehlsname	: **GETORG**		
Befehlsgruppe	: Assembler.		
Kurzbeschreibung	: Es wird die eingestellte Übersetzungsadresse gelesen.		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.L = Übersetzungsadresse.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SETERR (53)	GETERR (54)	ASSEMBLE (65)
	PUTORG (69)	ASSERR (120)	

Damit kann man den Defaultwert des Programmzählers für die Codeablage durch den Assembler abfragen. Im Register D0.L steht das Ergebnis. Der Wert wird nur durch die PUTORG-Anweisung, aber nicht durch ORG-Befehl im Assembler verändert.

Komplexeres Beispiel:

Die voreingestellte Übersetzungsadresse wird ausgegeben.

```
START:
        MOVEQ     #!GETORG,D7       * Übersetzungsadresse Assembler holen.
        TRAP      #1
        LEA       BUFFER(PC),A0     * Ziel für Ausgabe.
        MOVEQ     #!PRINT8X,D7      * In hexadezimale Darstellung wandeln.
        TRAP      #1
        LEA       BUFFER(PC),A0     * Adresse der Zeichen.
        MOVEQ     #$22,D0           * Schriftgröße.
        MOVEQ     #2,D1             * X-Position.
        MOVE      #128,D2           * Y-Position.
        MOVEQ     #!WRITE,D7        * Ausgabe auf Bildschirm.
        TRAP      #1
        RTS
BUFFER:
        DS.B      10                * Freiraum für Ablage.
```

Bemerkung:

Normalerweise erscheint der Wert $11C00 auf dem Bildschirm. Dorthin wird nämlich der Assemblercode gelegt, wenn man keine ORG-Anweisung gibt. Bei einem Grundprogramm, das nicht auf der Adresse 0 liegt, wird der Basiswert natürlich addiert.

TRAP-Nummer	: **69**		
Befehlsname	: **PUTORG**		
Befehlsgruppe	: Assembler		
Kurzbeschreibung	: Es wird die Übersetzungsadresse eingestellt.		
Eingaberegister	: D0.L = Zieladresse für Übersetzung		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 3.1		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SETERR(53)	GETERR(54)	ASSEMBLE(65)
	SETORG(68)	ASSERR(120)	

Damit kann man den Defaultwert für die Assembler-Codeablage ändern. Im Register D0.L steht dann die neue Adresse. Das ist z. B. für Übersetzungsprogramme interessant, die automatisch die Adressen für den Code bestimmen wollen.

Einfaches Beispiel:

Die Übersetzungsadresse wird auf die Adresse ENDE gesetzt.

```
START:
        LEA     ENDE(PC),A0
        MOVE.L  A0,D0           * Neue Adresse
        MOVEQ   #!PUTORG,D7     * für Assembler einstellen.
        TRAP    #1
        RTS
        DS.W    200             * Freiraum für Macro-Tabelle.
ENDE:
```

Wenn man das Programm startet, passiert noch nichts weiter. Wenn man aber dann den Assembler erneut startet, legt er den neuen Programmcode nicht auf der normalerweise voreingestellten Adresse ab, sondern direkt hinter den letzten Programmcode. Wenn man dann das Programm nochmals startet, wird bei der nächsten Übersetzung der Code wieder dahinter abgelegt.

TRAP-Nummer	: **70**		
Befehlsname	: **PRINT8D**		
Befehlsgruppe	: Wertausgabe.		
Kurzbeschreibung	: Wert ohne Vorzeichen in dezimale Darstellung wandeln.		
Eingaberegister	: D0.L = Zu wandelnder Wert. A0.L = Zieladresse für Zeichen.		
Ausgaberegister	: A0.L = Zeigt auf die Endekennung.		
Zerstörte Register	: D0		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: PRINT2X (41)	PRINT4X (42)	PRINT6X (43)
	PRINT8X (44)	PRINT8B (45)	PRINT4D (46)
	PRINTV8D (71)		

Damit kann eine 32-Bit-Größe, die im Register D0.L steht, dezimal ausgegeben werden. Die Ausgabe erfolgt ohne Vorzeichen. Im Register A0.L steht die Adresse des Buffers, in den die Zahl in Form von ASCII-Zeichen abgelegt wird. A0.L zeigt anschließend auf das nächste freie Byte. Dort wird auch eine 0 (Code 00) als Endekennung abgelegt.

Komplexeres Beispiel:

Die Zahl 4294967295 wird auf dem Bildschirm ausgegeben.

```
START:
        LEA     BUFFER(PC),A0       * Adresse Zielbuffer.
        MOVEQ   #$FF,D0             * $FFFFFFFF nach D0.
        MOVEQ   #!PRINT8D,D7        * Wert wandeln und ablegen.
        TRAP    #1
        LEA     BUFFER(PC),A0       * Adresse der Daten.
        MOVEQ   #$22,D0             * Schriftgröße.
        MOVE.W  #200,D1             * X-Position.
        MOVEQ   #120,D2             * Y-Position.
        MOVEQ   #!WRITE,D7          * Ausgabe.
        TRAP    #1
        RTS
BUFFER:
        DS.B    12
```

TRAP-Nummer	: **71**		
Befehlsname	: **PRINTV8D**		
Befehlsgruppe	: Wertausgabe.		
Kurzbeschreibung	: Wert mit Vorzeichen in dezimale Darstellung wandeln.		
Eingaberegister	: D0.L = Zu wandelnder Wert. A0.L = Ablageadresse für Zeichen.		
Ausgaberegister	: A0.L = Zeigt auf die Endekennung.		
Zerstörte Register	: D0		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: PRINT2X (41)	PRINT4X (42)	PRINT6X (43)
	PRINT8X (44)	PRINT8B (45)	PRINT4D (46)
	PRINT8D (70)		

Damit kann man eine 32-Bit-Größe, die im Register D0.L steht, dezimal ausgeben. Die Ausgabe erfolgt mit Vorzeichen. In Register A0.L steht die Adresse des Buffers, in dem die Zahl in Form von ASCII-Zeichen abgelegt wird. A0.L zeigt danach auf das nächste freie Byte. Dort wird auch eine 0 (Code 00) als Endekennung abgelegt.

Einfaches Beispiel:

Die Zahl -2147483648 wird auf dem Bildschirm ausgegeben.

```
START:
        LEA     BUFFER(PC),A0       * Zieladresse für Zeichen.
        MOVE.L  #$80000000,D0       * Wert nach D0.
        MOVEQ   #!PRINTV8D,D7       * Wert in Zeichen wandeln.
        TRAP    #1
        LEA     BUFFER(PC),A0       * Adresse der Zeichen.
        MOVEQ   #$22,D0             * Schriftgröße.
        MOVE.W  #180,D1             * X-Position.
        MOVEQ   #120,D2             * Y-Position.
        MOVEQ   #!WRITE,D7          * Ausgabe auf Bildschirm.
        TRAP    #1
        RTS
BUFFER:
        DS.B    12                  * Platz für ASCII-Zeichen.
```

Komplexeres Beispiel:

Man kann Rechenaufgaben, sogar mit Klammerrechnung, eingeben. Das Ergebnis wird nach der Eingabe von RETURN auf dem Bildschirm angezeigt. Dabei erfolgt die Rechnung nur mit ganzen Zahlen (siehe WERT-Befehl). Das Programm wird durch Eingabe von RETURN abgebrochen.

```
START:
        LEA     BUFFER(PC),A0       * Zieladresse für Text.
        MOVEQ   #$22,D0             * Schriftgröße.
        MOVEQ   #75,D1              * X-Position.
        MOVE.W  #130,D2             * Y-Position.
        MOVEQ   #30,D3              * Maximale Anzahl der Zeichen.
        MOVEQ   #!READ,D7           * Zeichen einlesen.
        TRAP    #1
        TST     D4
        BEQ.S   ENDE                * Ende, wenn RETURN eingegeben wird.
        LEA     BUFFER(PC),A0
        MOVEQ   #!WERT,D7           * Wert berechnen.
        TRAP    #1
        TST     D1
        BEQ.S   START               * Syntax-Fehler.
        CMP     #5,D1
        BEQ.S   START               * Undefiniertes Symbol.
        LEA     BUFFER(PC),A0
        MOVEQ   #!PRINTV8D,D7       * Ausgabe mit Vorzeichen.
        TRAP    #1
        MOVEQ   #10-1,D0
```

```
SCHLEIFE:
        MOVE.B      #' ',(A0)+      * Damit alte Zahl ganz überschrieben wird.
        DBRA        D0,SCHLEIFE
        CLR.B       (A0)            * Endekennung neu setzen.
        LEA         BUFFER(PC),A0
        MOVEQ       #$22,D0         * Schriftgröße.
        MOVEQ       #75,D1          * X-Position.
        MOVEQ       #100,D2         * Y-Position.
        MOVEQ       #!WRITE,D7      * Ausgabe auf Schirm.
        TRAP        #1
        BRA.S       START           * Wiederholen.
ENDE:
        RTS
BUFFER:
        DS.B 32                     * Platz für Zeichenablage.
```

TRAP-Nummer	: **72**		
Befehlsname	: **MULS32**		
Befehlsgruppe	: Berechnungen.		
Kurzbeschreibung	: Multiplikation von zwei 32-Bit-Zahlen.		
Eingaberegister	: D0.L = Multiplikand.		
	D2.L = Multiplikator.		
Ausgaberegister	: D0.L = Ergebnis untere 32 Bit.		
	D1.L = Ergebnis obere 32 Bit.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Ja.		
Änderungen zu 6.1	: D1.L ist jetzt vorzeichenbehaftet.		
Siehe auch	: SIN (23)	COS (24)	WERT (29)
	DIVS32 (73)	ADJ360 (83)	RND (96)

Der 68000/8-Prozessor kann multiplizieren, allerdings nur mit 16 Bit-Größen. Weil man für bestimmte Rechnungen aber auch 32 Bit-Multiplikationen benötigt, gibt es diese Routine. Der Multiplikand wird im Register D0.L übergeben, der Multiplikator im Register D2.L. Das Ergebnis steht nach dem Aufruf im Register D0.L und D1.L als Überlaufstelle. Dabei wird die Arithmetik vorzeichenbehaftet ausgeführt, die Zahlendarstellung erfolgt also im Zweierkomplement. Das Vorzeichen der Überlaufstelle wird auch angeglichen.

Das Grundprogramm für den 68020-Prozessor verfügt aus Gründen der Kompatibilität ebenfalls über den Befehl. Es wird aber nur eine interne Multiplikation ausgeführt, die wesentlich schneller ist.

Operation: D0.L * D2.L -> D1.L, D0.L

Einfaches Beispiel:

Im Register D0.L erscheint nach dem Aufruf die Zahl $7FFFFFFE. Man kann das Ergebnis im Einzelschritt überprüfen. Register D1.L wird auf Null gesetzt.

```
START:
        MOVE.L    #$3FFFFFFF,D0      * Erster Operand.
        MOVEQ     #2,D2              * Zweiter Operand.
        MOVEQ     #!MULS32,D7        * Ausrechnen.
        TRAP      #1
        RTS                          * Ergebnis in D0.L und D1.L.
```

Da die Multiplikation MULS32 langsamer erfolgt als mit den eingebauten 68000/8-Multiplikationen (MULU, MULS), sollte man bei zeitkritischen Operationen versuchen, mit den 16-Bit-Operationen auszukommen.

Beim 68020 erfolgt die Multiplikation natürlich mit höherer Geschwindigkeit.

TRAP-Nummer	: **73**		
Befehlsname	: **DIVS32**		
Befehlsgruppe	: Berechnungen.		
Kurzbeschreibung	: Ausführung einer 32-Bit-Division.		
Eingaberegister	: D0.L = Divisor. D2.L = Dividend.		
Ausgaberegister	: D0.L = Ergebnis der Division. D1.L = Rest der Division.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Ja.		
Änderungen zu 6.1	: Das Vorzeichen des Restes entspricht dem Vorzeichen des Divisors.		
Siehe auch	: SIN (23)	COS (24)	WERT (29)
	MULS32 (72)	ADJ360 (83)	RND (96)

Die Prozessoren 68008 und 68000 haben eine eingebaute Division, die aber nur mit 16 Bit arbeitet. Um auch 32 Bit-Divisionen durchführen zu können, gibt es diesen Befehl.

Im Register D0.L wird der Divisor abgelegt. Im Register D2.L steht der Dividend. Das Ergebnis der Division steht anschließend im Register D0.L und der Rest im Register D1.L. Die Division wird vorzeichenbehaftet durchgeführt.

Das Vorzeichen des Restes (D1.L) entspricht dem Vorzeichen des Divisors (D0.L vorher).

Der Prozessor 68020 führt diese Operation intern durch.

Operation: D0.L / D2.L -> D0.L REST D1.L

Einfaches Beispiel:

Wenn man das Programm im Einzelschritt durchläuft, steht nach dem Aufruf im Register D0.L der Wert $3FFFFFFF und im Register D1.L der Wert 1.

```
START:
        MOVE.L    #$7FFFFFFF,D0     * Positive maximale Zahl.
        MOVEQ     #2,D2             * Durch 2 teilen.
        MOVEQ     #!DIVS32,D7       * Division ausführen.
        TRAP      #1
        RTS                         * Ergebnis in D0.L / Rest in D1.L.
```

Da die Division mit DIVS32 langsamer erfolgt als mit der eingebauten 68000/8-Division (DIVU, DIVS), sollte man bei zeitkritischen Operationen versuchen, mit den 16-Bit-Operationen auszukommen.

Beim 68020 erfolgt die Division natürlich mit höherer Geschwindigkeit.

TRAP-Nummer	: **74**
Befehlsname	: **FLINIT**
Befehlsgruppe	: FLO-Baugruppe.
Kurzbeschreibung	: Initialisierung der Floppy-Variablen.
Eingaberegister	: Keine.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 4.0
Änderungen zu 4.3	: Ja.
Änderungen zu 6.1	: Es wird die Steprate 0 gesetzt (siehe FLOPPY). Außerdem wird bei der Steprate das SSO-Bit gesetzt.
Siehe auch	: FLOPPY (75) GETFLOP (76)

Diesen Befehl braucht man beim Anschluß einer Floppy. Dazu wird die FLO2- oder die FLO3-Baugruppe benötigt.

Der Befehl setzt die Steprate des Laufwerks auf den schnellsten Wert und initialisiert eine Spurtabelle, die besagt, daß kein Laufwerk gültig ist. Damit wird beim ersten Zugriff auf das Laufwerk sichergestellt, daß die Routinen keinen Fehler melden.

Die Routine FLINIT wird auch beim Start des Grundprogramms automatisch aufgerufen, sodaß man sie normalerweise nicht benötigt, wenn man mit der Floppy arbeiten will. Sie funktioniert aber auch richtig, wenn keine FLO-Baugruppe vorhanden ist.

Einfaches Beispiel:

Interne Spurtabelle initialisieren und Steprate auf Null setzen.

```
START:
        MOVEQ     #!FLINIT,D7        * Voreinstellungen durchführen.
        TRAP      #1
        RTS
```

Achtung! Der Befehl FLINIT ist nicht zur Formatierung der Diskette verwendbar. Zur Formatierung benötigt man ein Programm, das nicht im Grundprogramm enthalten ist, aber z. B. im JADOS.

TRAP-Nummer	: **75**
Befehlsname	: **FLOPPY**
Befehlsgruppe	: FLO-Baugruppe.
Kurzbeschreibung	: Floppy-Befehl ausführen.
Eingaberegister	: D1.W = Befehlsauswahl. D2.B = Sektor, wenn benötigt. D3.B = Spur oder Steprate, wenn benötigt. D4.B = Laufwerk und Dichtecode, wenn benötigt. A0.L = Adresse der Daten, wenn benötigt.
Ausgaberegister	: D0.L = Fehlercode 0 oder -1 oder Statusregister. Carry = Gesetzt, wenn Fehler aufgetreten ist.
Zerstörte Register	: Keine.
Ab Version	: 4.0
Änderungen zu 4.3	: Alle Befehle des Controlers sind verfügbar. Die Routine kann auf die serielle Karte umgelenkt werden.
Änderungen zu 6.1	: Der letzte aufgetretene Fehler kann genau ermittelt werden. D0.L ist als Fehlerausgabe gültig.
Siehe auch	: FLINIT (74) GETFLOP (76)

Diese Routine ist für den Betrieb mit der FLO2-Baugruppe bestimmt. Damit kann man alle Operationen durchführen, die der FLOPPY-Controler zur Verfügung stellt. Außerdem ist eine Funktion vorhanden, die den letzten aufgetretenen Fehler zurückgibt.

Bei jedem Fehler wird im Register D0.L der Wert $FFFFFFFF (-1) zurückgegeben. Gleichzeitig wird das CARRY-Flag gesetzt.

Die FLOPPY kann auf die serielle Karte gelenkt werden. Dabei sind aber nur die Funktionen SEKTOR LESEN und SEKTOR SCHREIBEN möglich.

Registerbelegung:

D2.B **Sektor:**

1...n (n = 16, 18, 26 je nach Laufwerk).
Hier wird der Sektor angegeben, auf den zugegriffen werden soll. Dabei hängt die Anzahl der Sektoren pro Spur von der Formatierung ab.

D3.B **Spur (Bei D1 ≠ 0):**

0...k (k = 34, 39, 76, 79 je nach Laufwerk).
Die Anzahl der Spuren hängt von der Art des Laufwerks ab.

Steprate (Bei D1 ≠ 0):

Bei dem Befehl D1.W = 0 (Steprate setzen) hat dieses Register eine besondere Funktion. Hier wird angegeben, welchen Wert die Steprate haben soll, d. h. wie schnell der Laufwerkskopf positioniert werden soll.

0 - 3 Stepraten bei Mini-Laufwerken.

4 - 7 Stepraten bei Maxi-Laufwerken für Mini-Laufwerke. Der kleinere Wert steht für die schnellere Steprate. Manche 5 1/4- und 3 1/2-Zoll Laufwerke vertragen auch schnellere Stepraten. Dann können Einstellungen von 4 - 7 für die Laufwerke eingestellt werden. Ansonsten sollte ein Wert von 0 bis 3 benutzt werden. Bei einer Steprate von 4 bis 7 wird die Spur nicht mehr durch Prüflesen identifiziert. Die schnelle Steprate ist bei Schreibvorgängen abgeschaltet.
Soll auch auf die Seite 1 eines Laufwerks zugegriffen werden, muß bei der Einstellung der Steprate das Bit 7 gesetzt werden. Nur wenn dieses Bit gesetzt ist, kann der Controler ordnungsgemäß auch die Seite 1 zugreifen.

D4.B **Laufwerks und Dichtecode:**

Dieses Register entspricht genau einem Port auf der FLO-Baugruppe. Dabei haben die Bits folgende Bedeutung:

Bits 0 - 3 Laufwerkscodierung.

Normalerweise steht jedes Bit für ein Laufwerk, nämlich Bit 0 für Laufwerk 1, Bit 1 für Laufwerk 2 usw. Es ist aber auch eine andere Codierung denkbar (Nur mit Hardwareänderung).

Bit 4 Dichte (1 = einfache Dichte).

Bit 5 Laufwerk (1 = Minilaufwerk).

Bit 6 Motorsteuerung (0 = Motor ein).

Bit 7 Seitenauswahl (1 = Zugriff erfolgt auf Seite 1).

Dieses Bit wählt die Seite bei einem doppelseitigem Diskettenlaufwerk aus. Soll auch wirklich ein Zugriff auf Seite 1 erfolgen, so muß zusätzlich das Bit 7 der Steprate gesetzt sein.

A0.L Adresse für Ziel oder Quelle:

Das Register wird benötigt, wenn Daten zum Diskettenlaufwerk oder vom Diskettenlaufwerk in den Speicher transportiert werden müssen. Dabei zeigt A0.L immer auf das Ziel bzw. die Quelle. Benötigt bei D1.W = 1, 2, 3, 4, 5.

D1.W Befehlscode:

In diesem Register wird festgelegt, welcher Befehl ausgeführt werden soll.

0 = Steprate in D3.B setzen

Wenn dieser Befehl aufgerufen wird, muß im Register D3.B die Steprate stehen (0 - 7) und eventuell das SSO-Bit (Bit 7). Das SSO-Bit erlaubt oder verbietet den Zugriff auf Seite 1 einer Diskette. Wird das Bit nicht gesetzt, so erfolgt nur ein Zugriff auf Seite 0. Bei dem Befehl werden keine weiteren Register benötigt.

1 = Sektor lesen

Je nach Formatierung werden 128, 256, 512 oder 1024 Bytes gelesen. Dabei muß Register D2.B, D3.B sowie Register D4.B richtig belegt sein. Im Register A0.L wird die Zieladresse übergeben. Ist die FLOPPY-Routine auf die serielle Karte gelenkt, werden immer 1024 Bytes gelesen, gleichgültig, welche Werte in D4.B eingestellt sind. Außerdem werden vor dem Lesen die Werte der Register D1.B, D2.B, D3.B und das Register D4.B (Bits 4 - 6 gelöscht) an die serielle Karte übertragen. Die Werte werden direkt mit der Routine SO übertragen.

2 = Sektor schreiben

Je nach Formatierung werden 128, 256, 512 oder 1024 Bytes geschrieben. Dabei muß Register D2.B, D3.B sowie Register D4.B richtig belegt sein. Im Register A0.L wird die Quelladresse übergeben.

Ist die FLOPPY-Routine auf die serielle Karte gelenkt, gilt das Gleiche wie bei SEKTOR LESEN mit dem Unterschied, daß die 1024 Bytes geschrieben und nicht gelesen werden.

3 = Track lesen

Dieser Befehl funktioniert wie SEKTOR LESEN. Allerdings wird der gesamte Track mit allen Synchronisationsbytes gelesen. Die Sektornummer (D2.B) muß nicht übergeben werden. Siehe auch FLO-Handbuch für nähere Informationen über Track-Daten.

4 = Track schreiben

Dieser Befehl funktioniert wie TRACK LESEN. Allerdings wird der gesamte Track mit allen Synchronisationsbytes geschrieben. Die Sektornummer (D2.B) muß nicht übergeben werden.

Der Befehl wird zum Formatieren einer Diskette benötigt. Siehe FLO-Handbuch für nähere Informationen über Track-Daten.

5 = Startkopf lesen

Dieser Befehl funktioniert wie der Befehl SEKTOR LESEN, allerdings wird nicht der Inhalt des Sektors gelesen, sondern nur der Kopf des Sektors, in dem wichtige Informationen über Länge, Dichte usw. stehen.

6 = Restore

Der Befehl bewirkt, daß der Kopf des Laufwerks auf die Spur Null gefahren wird. Register D4.B wird benötigt.

7 = Spur suchen

Mit diesem Befehl kann der Kopf auf eine bestimmte Spur gefahren werden. Dazu muß im Register D3.B die Spurnummer übergeben werden. Außerdem muß D4.B richtig gesetzt sein, da ein Prüflesen erfolgt.

8 = Schreibkopf eine Spur nach innen

Der Kopf des Laufwerks wird um eine Spur nach innen, d.h. zur Spur 0 hin gefahren. Weitere Register werden nicht benötigt.

9 = Schreibkopf eine Spur nach außen

Der Kopf des Laufwerks wird um eine Spur nach außen hin gefahren. Weitere Register werden nicht benötigt.

10 = Letzten Fehler lesen

Wurde bei einem Zugriff auf das Laufwerk vom Controler ein Fehler entdeckt, wird zuerst der Wert $FFFFFFFF im Register D0.L zurückgegeben. Außerdem ist das CARRY-Flag gesetzt. Zur genaueren Analyse des Fehlers kann danach dieser Befehl aufgerufen werden. Er gibt im Register D0.L das Statusregister des Controlers beim Auftreten des Fehlers zurück (Wert aber nur in D0.B). Dabei hat das Bit 0 eine besondere Bedeutung. Wenn es auf Null gesetzt ist, war der Befehl, bei dem der Fehler aufgetreten ist, einer aus der Gruppe 1, sonst war es ein Befehl aus der Gruppe 2 oder 3.

Wurde auf die serielle Karte zugegriffen, sollte dieser Befehl nicht aufgerufen werden. Die Werte stammen dann nämlich noch vom letzten Zugriff auf die FLOPPY, da sie in dem Falle nicht verändert werden. Die Bedeutung der einzelnen Bits kann dem FLO-Handbuch entnommen werden. Die FLAGS sind entsprechend dem Wert in D0.B gültig.

Nach dem Aufruf enthält das Register D0.L einen Fehlercode:

D0.L = 0 bedeutet	Kein Fehler. Gleichzeitig ist das Carry-Flag zurückgesetzt.
D0.L = -1 bedeutet	Fehler aufgetreten. Gleichzeitig ist das Carry-Flag gesetzt. Ein Fehler tritt auch auf, wenn ein Schreibschutz auf der Diskette vorhanden ist und ein Schreibzugriff stattfindet.

Wird eine falsche Nummer in D1.W übergeben (nicht implementierter Befehl), wird in D0.L auch ein $FFFFFFFF übergeben, und das CARRY-Flag ist gesetzt.

Bevor der FLOPPY-Befehl benutzt werden kann, muß die Floppy-Diskette formatiert werden, falls sie nicht schon vom Werk aus formatiert ist. Dazu wird ein Formatierprogramm verwendet, das nicht im Grundprogramm enthalten ist.

Einfaches Beispiel:

Es wird der Sektor 2 auf Spur 0 gelesen und im Buffer abgelegt.

```
START:
        LEA     BUFFER(PC),A0     * Zieladresse festlegen.
        MOVEQ   #1,D1             * Sektor lesen.
        MOVEQ   #2,D2             * Sektor 2.
        MOVEQ   #0,D3             * Spur 0.
        MOVEQ   #$21,D4           * 5 1/4"-Laufwerk, doppelte Dichte, Laufwerk 1.
        MOVEQ   #!FLOPPY,D7       * Befehl ausführen.
        TRAP    #1
        RTS
BUFFER:
        DS.B    128
```

Der Dichtecode muß dem jeweiligen Laufwerk angepaßt werden. Der Buffer ist bei einfacher Dichte 128 Bytes groß, bei doppelter Dichte 256 oder 1024, je nach Formatierung. Das Programm FLOPPY stellt die Größe automatisch anhand der Formatierung auf der Diskette fest und übertragt die richtige Anzahl von Bytes.

Abschließend noch eine Zusammenstellung gebräuchlicher Dichtecodes für Laufwerk 1. Am Laufwerk muß die Laufwerksnummer eingestellt werden.

8 Zoll, einfache Dichte	$11
8 Zoll, doppelte Dichte	$01
5 1/4 Zoll, einfache Dichte	$31
5 1/4 Zoll, doppelte Dichte	$21

Für 3 Zoll- und 3 1/2-Zoll-Laufwerke gibt es jeweils aus dem oberen Beispiel einen gültigen Code.

Will man die Rückseite eines Doppelkopflaufwerks ansprechen, addiert man einfach den Wert $80 auf den Laufwerks-Dichtecode. Damit der Zugriff richtig funktioniert, muß bei der Stepratenеinstellung das Bit 7 gesetzt sein.

TRAP-Nummer	: **76**
Befehlsname	: **GETFLOP**
Befehlsgruppe	: FLO-Baugruppe.
Kurzbeschreibung	: FLOPPY-Format feststellen.
Eingaberegister	: D4.B = Nummer des Laufwerks.
Ausgaberegister	: D0.W = Fehlercode 0 oder $FFFF. D1.B = Statusregister der FLO-Baugruppe. D4.W = Laufwerkscode. Carry = Gesetzt bei nicht richtig formatierter Diskette.
Zerstörte Register	: Keine.
Ab Version	: 4.0
Änderungen zu 4.3	: Nein.
Änderungen zu 6.1	: Der Befehl wird nicht mehr abgebrochen, wenn keine Diskette im Laufwerk ist.
Siehe auch	: FLINIT (74) FLOPPY (75)

Der Befehl ermöglicht es, den Dichtecode einer formatierten Diskette automatisch zu ermitteln. Dazu muß im Register D4.B die Laufwerksnummer angegeben werden. Nach dem Aufruf steht im Register D4.B der Laufwerkscode und der Dichtecode, wie er vom FLOPPY-Befehl verlangt wird. Die Bestimmung erfolgt immer auf Spur 0, weshalb in manchen Fällen, bei denen die inneren Spuren in einer anderen Dichte beschrieben sind, trotzdem eine manuelle Bestimmung nötig ist. Jedoch ist der Befehl in erster Linie für den automatischen Start eines Programms auf der Spur 0 bestimmt. Es wird das Carry-Flag gesetzt, wenn ein Fehler aufgetreten ist. Im Register D0.W erscheint dann der Wert $FFFF, sonst der Wert 0. Der aufgetretene Fehler kann aus den Bits in Register D1.B ermittelt werden, die denen des Statusregisters der FLO-Baugruppe entsprechen.

Einfaches Beispiel:

Mit diesem Programm ist es möglich, den ersten Sektor einer Diskette zu lesen, unabhängig davon, ob die Diskette in einem Mini- oder Maxilaufwerk steckt oder mit einfacher oder doppelter Dichte beschrieben werden kann.

```
START:
        MOVEQ     #1,D4              * Laufwerk 1.
        MOVEQ     #!GETFLOP,D7       * Format bestimmen.
        TRAP      #1
        LEA       BUFFER(PC),A0      * Adresse der Daten.
        MOVEQ     #1,D1              * Sektor lesen.
        MOVEQ     #1,D2              * Sektor 1.
        MOVEQ     #0,D3              * Spur 0.
        MOVEQ     #!FLOPPY,D7        * Befehl ausführen.
        TRAP      #1
        RTS
BUFFER:
        DS.B      1024               * Sicherheitshalber groß.
```

Da GETFLOP unter Umständen etwas Zeit (weniger als 1 Sekunde) braucht, um die Dichte zu ermitteln, sollte man GETFLOP nicht vor jedem FLOPPY-Befehl verwenden, sondern nur einmal am Anfang eines Programms.

Beispiele für Codierung der Laufwerksnummer:

Laufwerk 1 Seite 0	$01
Laufwerk 2 Seite 1	$82
Laufwerk 3 Seite 1	$84
Laufwerk 4 Seite 0	$08

TRAP-Nummer	: 77
Befehlsname	: **SETXOR**
Befehlsgruppe	: Grafik.
Kurzbeschreibung	: Setzt den XOR-Modus (an oder aus).
Eingaberegister	: D0.B = Zu setzender Modus ($0 - $F).
Ausgaberegister	: D0.B = Wert des GDP-Port $60.
Zerstörte Register	: Keine.
Ab Version	: 4.0
Änderungen zu 4.3	: Nein.
Änderungen zu 6.1	: Nein.
Siehe auch	: MOVETO (8), DRAWTO (9), CLR (16), CLPG (17), WAIT (18), CMD (26), NEWPAGE (27), SETFLIP (34), SETPEN (37), ERAPEN (38), CMDPRINT (40), AUTOFLIP (60), GETXOR (78), SETCOLOR (79), GETCOLOR (80), GETXY (103), HARDCOPY (125), GRAFIK (126), GDPVERS (127)

SETXOR ist ein Befehl, der für eine GDP-Erweiterung gedacht ist (GDPHS). Damit ist es möglich, die zweite Hälfte des Seitenregisters auf der GDP-Karte zu setzen. Dazu enthält das Register D0.B einen Wert zwischen $0 und $F. Beim Aufruf wird auch die aktuelle Lese- und Schreibseite gesetzt. Der Wert $0 steht für die normale Einstellung. Als Ausgabe wird in D0.B der Wert geliefert, der auf Port $60 der GDP-Karte geschrieben wird.

Die neue GDPHS hat schon eine XOR-Erweiterung eingebaut. Dabei wird allerdings nur das Bit 0 benutzt. Im Handbuch der GDPHS ist die Schaltung des XOR-Modus beschrieben, weshalb hier auf eine nähere Erläuterung verzichtet wird.

Einfaches Beispiel:

Der XOR-Modus wird eingeschaltet.

```
START:
        MOVEQ     #1,D0          * XOR-Modus anschalten.
        MOVEQ     #!SETXOR,D7    * Befehl ausführen.
        TRAP      #1
        RTS
```

Beim XOR-Modus werden bei Schreibvorgängen alle Punkte komplementiert. Wenn z. B. ein Punkt gesetzt war, wird er gelöscht, wenn er nicht gesetzt war, wird er eingeschrieben. Damit kann man auch einfach bewegte Figuren erzeugen, ohne mit dem AUTOFLIP arbeiten zu müssen. Der Vorteil ist, daß man ein flimmerfreies Bild erhält. Man muß aber bei der Programmierung gut aufpassen, da ein doppelt geschriebener Punkt wieder verschwindet, was z. B. für Teile von Linien gilt, die mit DRAWTO oder FIGUR geschrieben worden sind.

Komplexeres Beispiel:

Der Buchstabe A erscheint auf dem Bildschirm. An der Stelle, an der die Linie den Buchstaben durchquert, ist sie als dunkle Linie auf hellem Grund sichtbar.

```
START:
        CLR       D1             * Linie von 0,0
        CLR       D2
        MOVEQ     #!MOVETO,D7
        TRAP      #1
        MOVE.W    #512,D1        * nach 512,255 zeichnen.
        MOVE.W    #255,D2
        MOVEQ     #!DRAWTO,D7
        TRAP      #1
        MOVEQ     #1,D0
        MOVEQ     #!SETXOR,D7    * XOR-Modus anschalten.
        TRAP      #1
        MOVE.W    #210,D1
        MOVE.W    #60,D2
        MOVEQ     #!MOVETO,D7    * Auf Position 210,60.
        TRAP      #1
```

```
MOVE      #!SYSTEM,D7     * System-Informationen lesen.
TRAP      #1
AND       #7,D0           * Nur CPU-Information lassen.
MULS      #$FF73,D0
MOVEA.L   D0,A0           * GDP-Port $73.
MOVEQ     #!WAIT,D7       * Warten, bis GDP fertig.
TRAP      #1
CLR.B     (A0)            * Schriftgröße auf maximalen Wert.
MOVEQ     #'A',D0
MOVEQ     #!CMD,D7        * A ausgeben.
TRAP      #1
MOVEQ     #0,D0
MOVEQ     #!SETXOR,D7     * XOR-Modus zurücksetzen.
TRAP      #1
RTS
```

TRAP-Nummer	:	**78**
Befehlsname	:	**GETXOR**
Befehlsgruppe	:	Grafik.
Kurzbeschreibung	:	Liest den mit SETXOR eingestellten XOR-Modus zurück.
Eingaberegister	:	Keine.
Ausgaberegister	:	D0.L = XOR-Modus ($0 - $F).
Zerstörte Register	:	Keine.
Ab Version	:	4.0
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	MOVETO (8), DRAWTO (9), CLR (16), CLPG (17), WAIT (18), CMD (26), NEWPAGE (27), SETFLIP (34), SETPEN (37), ERAPEN (38), CMDPRINT (40), AUTOFLIP (60), SETXOR (77), SETCOLOR (79), GETCOLOR (80), GETXY (103), HARDCOPY (125), GRAFIK (126), GDPVERS (127)

Damit kann man den Wert des eingestellten XOR-Modes wieder auslesen. Der Wert erscheint im Register D0.L und hat einen Bereich von $0 bis $F. Siehe SETXOR.

Einfaches Beispiel:

XOR-Modus abfragen.

```
START:
        MOVEQ       #!GETXOR,D7         * Wert lesen.
        TRAP        #1
        RTS
```

GETXOR und SETXOR arbeiten auch, ohne daß die Schaltungsänderung in der GDP-Karte eingebaut ist, jedoch erfolgt keine Reaktion auf dem Bildschirm. Nur die Werte werden behalten. Da ein Wertebereich von $0 bis $F vorliegt, kann man den Portausgang am 74LS273 auch für andere Zwecke verwenden. GETXOR und SETXOR beziehen sich nur auf die unteren vier Bits dieses Ports.

TRAP-Nummer	:	**79**
Befehlsname	:	**SETCOLOR**
Befehlsgruppe	:	Grafik.
Kurzbeschreibung	:	Setzt die Farbe der GDP-Karte.
Eingaberegister	:	D0.B = Zu setzende Farbe.
Ausgaberegister	:	D0.B = Wert des Ports $60 der GDP-Karte.
Zerstörte Register	:	Keine.
Ab Version	:	4.0
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	MOVETO (8), DRAWTO (9), CLR (16), CLPG (17), WAIT (18), CMD (26), NEWPAGE (27), SETFLIP (34), SETPEN (37), ERAPEN (38), CMDPRINT (40), AUTOFLIP (60), SETXOR (77), GETXOR (78), GETCOLOR (80), GETXY (103), HARDCOPY (125), GRAFIK (126), GDPVERS (127)

Ebenfalls für eine Erweiterung ist dieser Befehl gedacht. Ein Wert zwischen $0 und $FF wird im Register D0.B übergeben. Dieser Wert wird auf den Port $FFFFFFA0*CPU ausgegeben. Die Ausgabe erfolgt erneut bei jedem internen AKTPAGE-Aufruf. Dabei ist der Wert $0 für Farbe weiß bestimmt. Er wird vom Grundprogramm nach dem RESET eingestellt. Wenn man eine Farberweiterung bauen will, empfiehlt sich folgende Konvention, um kompatibel zu bleiben:

Bitnummern des Ports:

7	6	5	4	3	2	1	0
TON	WRT	TON	WRT	TON	WRT	TON	WRT
Hintergrund		—Blau—		—Grün—		—Rot—	

Wenn das Bit WRT auf 0 liegt, ist die jeweilige Farbe aktiviert. Wenn das Bit TON auf 0 liegt, wird die Farbe geschrieben, wenn es auf 1 liegt, wird sie gelöscht. Dabei muß aber SETPEN eingeschaltet sein. In der Realisation wird TON über ein Oder-Glied zum Freischalten des DIN-Eingangs bei den RAMs verwendet und WRT zum Freischalten des DW-(R/-W)-Eingangs der RAMs. Damit ist es möglich, transparente oder deckende Farben zu schreiben. Die Hintergrundfarbe kann z. B. weiß sein, um die anderen Farbebenen zu überdecken, oder eine Graustufe. Oder es können alle Farbebenen über eine Farbtabelle geführt werden, um beliebige Zuordnungen zu ermöglichen.

Einfaches Beispiel:

```
START:
        MOVEQ     #$FC,D0          * Farbe Rot wählen.
        MOVEQ     #!SETCOLOR,D7    * Farbe setzen.
        TRAP      #1
        RTS
```

Bemerkung:

Dieser Befehl ist bisher nie benutzt worden und wird vermutlich erst dann in einer etwas anderen Form eine Rolle spielen, wenn die GDP gegen eine Farbgrafik-Baugruppe ausgetauscht wird.

TRAP-Nummer	:	80
Befehlsname	:	**GETCOLOR**
Befehlsgruppe	:	Grafik.
Kurzbeschreibung	:	Fragt den Farbcode der GDP ab.
Eingaberegister	:	Keine.
Ausgaberegister	:	D0.L = Gesetzter Farbcode.
Zerstörte Register	:	Keine.
Ab Version	:	4.0
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	MOVETO (8), DRAWTO (9), CLR (16), CLPG (17), WAIT (18), CMD (26), NEWPAGE (27), SETFLIP (34), SETPEN (37), ERAPEN (38), CMDPRINT (40), AUTOFLIP (60), SETXOR (77), GETXOR (78), SETCOLOR (79), GETXY (103), HARDCOPY (125), GRAFIK (126), GDPVERS (127)

Damit kann der aktuelle Inhalt des Farbregisters ausgelesen werden. Er erscheint im Register D0.L mit dem Wertebereich 0 bis $FF. Siehe auch SETCOLOR.

Einfaches Beispiel:

```
START:
        MOVEQ     #!GETCOLOR,D7        * Farbcode holen.
        TRAP      #1
        RTS
```

Auch ohne Farberweiterung wird hier der zuletzt gesetzte Farbwert abgeliefert.

Bemerkung:

Dieser Befehl ist bisher nie benutzt worden und wird vermutlich erst dann in einer etwas anderen Form eine Rolle spielen, wenn die GDP gegen eine Farbgrafik-Baugruppe ausgetauscht wird.

TRAP-Nummer	:	**81**
Befehlsname	:	**CURON**
Befehlsgruppe	:	Zeichenausgabe.
Kurzbeschreibung	:	Schaltet die Cursordarstellung ein.
Eingaberegister	:	Keine.
Ausgaberegister	:	Keine.
Zerstörte Register	:	Keine.
Ab Version	:	4.0
Änderungen zu 4.3	:	Ja.
Änderungen zu 6.1	:	CURON darf auch im HARDSCROLL-Modus benutzt werden.
Siehe auch	:	CLRSCREEN (20), CO (21), LO (22), SIZE (25), CO2 (33), CRT (49), LST (50), USR (51), NIL (52), SETPASS (55), CURSEIN (61), CURSAUS (62), CHAR (63), CUROFF (82), CRLF (99), GETLINE (100), GETCURXY (101), SETCURXY (102), LSTS (117), CO2SER (129)

Damit wird die interne Variable CURON auf 1 gesetzt. Danach erscheint der Cursor bei der Ausgabe durch CO, CO2 und CHAR. Die Variable CURON wird auch auf 1 gesetzt, wenn man CLRSCREEN aufruft.

Die Routine darf nicht mit CURSEIN verwechselt werden, da mit CURSEIN der Cursor wirklich ausgegeben wird, während mit CURON nur bewirkt wird, daß er z. B. bei CI gezeigt wird.

Einfaches Beispiel:

Die automatische Bildseitenumschaltung wird aktiviert, der Cursor wird automatisch dargestellt.

```
START:
        MOVEQ   #10,D0          * Flip einschalten.
        CLR     D1
        MOVEQ   #!SETFLIP,D7
        TRAP    #1
        MOVEQ   #!CURON,D7      * Cursor einschalten.
        TRAP    #1
        RTS
```

Auf dem Bildschirm erscheint ein blinkender Cursor. Allerdings ist die Position nicht definiert. In bestimmten Fällen kann er auch unsichtbar bleiben, denn CLRSCREEN wurde in unserem Beispiel nicht aufgerufen. Jedoch verwendet der Assembler auch CLRSCREEN, weshalb die meisten Variablen richtig gesetzt sind.

CURON wird meist dazu verwendet, um einen ausgeschalteten Cursor wieder einzuschalten.

TRAP-Nummer	:	**82**
Befehlsname	:	**CUROFF**
Befehlsgruppe	:	Zeichenausgabe.
Kurzbeschreibung	:	Die Cursordarstellung wird ausgeschaltet.
Eingaberegister	:	Keine.
Ausgaberegister	:	Keine.
Zerstörte Register	:	Keine.
Ab Version	:	4.0
Änderungen zu 4.3	:	Ja.
Änderungen zu 6.1	:	CUROFF darf auch im HARDSCROLL-Modus betrieben werden.
Siehe auch	:	CLRSCREEN (20), CO (21), LO (22), SIZE (25), CO2 (33), CRT (49), LST (50), USR (51), NIL (52), SETPASS (55), CURSEIN (61), CURSAUS (62), CHAR (63), CURON (81), CRLF (99), GETLINE (100), GETCURXY (101), SETCURXY (102), LSTS (117), CO2SER (129)

Mit dem Befehl kann der Cursor ausgeschaltet werden, etwa wenn man bei einer Eingabe keinen Cursor auf dem Bildschirm sehen will. Dazu wird intern die Variable CURON auf 0 gesetzt (siehe auch CURON).

Einfaches Beispiel:

Der Buchstabe A wird ausgegeben, der Cursor wird nach der Ausgabe ausgeschaltet.

```
START:
        MOVEQ   #!CLRSCREEN,D7      * Bildschirm für CO2 vorbereiten.
        TRAP    #1
        MOVEQ   #'A',D0
        MOVEQ   #!CO2,D7            * A ausgeben.
        TRAP    #1
        MOVEQ   #!CUROFF,D7         * Cursor ausschalten.
        TRAP    #1
        RTS
```

Wenn man CUROFF wegläßt, blinkt der Cursor rechts neben dem Zeichen. Man kann die Wirkung von CUROFF auch durch Aufruf von CURON aufheben.

TRAP-Nummer	: **83**		
Befehlsname	: **ADJ360**		
Befehlsgruppe	: Berechnungen.		
Kurzbeschreibung	: Begrenzt einen Wert auf einen 360-Grad-Bereich.		
Eingaberegister	: D0.W = Anzugleichender Wert.		
Ausgaberegister	: D0.W = Neuer Wert im Bereich 0 bis 359.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SIN (23)	COS (24)	WERT (29)
	MULS32 (72)	DIVS32 (73)	RND (96)

Mit diesem Befehl wird ein Wert, der im Register D0.W steht, in den Bereich 0 bis 359 gebracht. Die Ausgabe erfolgt über das Register D0.W. Es können auch negative Werte in der Zweierkomplementdarstellung verarbeitet werden. Dabei ist an die Anwendung von Winkelfunktionen gedacht.

Dieser Befehl wird intern verwendet, um die Werte für SIN und COS aufzubereiten.

Einfaches Beispiel:

Im Register D0.W steht nach dem Aufruf der Wert 1.

```
START:
        MOVE    #361,D0          * Wert: 361.
        MOVEQ   #!ADJ360,D7
        TRAP    #1               * Als Ergebnis D0.W = 1.
        RTS
```

Die Umrechnung erfolgt durch fortgesetzte Subtraktion (bzw. Addition bei negativen Werten), weshalb man nicht allzu große Zahlen als Eingabe verwenden sollte.

TRAP-Nummer	: **84**		
Befehlsname	: **PRTSYM**		
Befehlsgruppe	: Symboltabelle.		
Kurzbeschreibung	: Gibt die Symboltabelle über CO2 aus.		
Eingaberegister	: Keine.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: HARDSCROLL wird - wenn möglich - angeschaltet. Die Ausgabe kann abgebrochen werden.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: ZUWEIS (30)	SYMCLR (85)	GETSYM (86)
	GETNEXT (87)	PUTNEXT (88)	

Ausgabe der aktuellen Symboltabelle. Die Symboltabelle wird über die CO2-Schnittstelle ausgegeben. Vorher wird der Bildschirm gelöscht.

Eine Umlenkung ist natürlich auch möglich, da CO2 vorher durch die entsprechenden Befehle umgelenkt werden kann.

Die Ausgabe der Symboltabelle ist in Kapitel 3.1.4 beschrieben.

Einfaches Beispiel:

Die Symboltabelle wird ausgegeben.

```
START:
        MOVEQ     #!PRTSYM,D7        * Symboltabelle ausgeben.
        TRAP      #1
        RTS
```

Nach der Ausgabe wartet PRTSYM auf die Eingabe von M, erst dann wird im Programm fortgefahren. Daher erscheint der Text F=Flip und M=Menü zweimal. Die Symboltabelle wird auf jeden Fall ausgegeben, da intern die Variable PASSFLAG auf 2 gesetzt wird. Eine Umlenkung auf beliebige Ausgabegeräte (NIL, CRT, USR) ist aber möglich.

TRAP-Nummer	: **85**		
Befehlsname	: **SYMCLR**		
Befehlsgruppe	: Symboltabelle.		
Kurzbeschreibung	: Die Symboltabelle wird gelöscht.		
Eingaberegister	: Keine.		
Ausgaberegister	: A0.L = Zeigt auf den Symboltabellenanfang.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Ja.		
Änderungen zu 6.1	: Ausgaberegister A0.l ist jetzt definiert.		
Siehe auch	: ZUWEIS (30)	PRTSYM (84)	GETSYM (86)
	GETNEXT (87)	PUTNEXT (88)	

Damit kann man die Symboltabelle löschen. Der Befehl ist z. B. für automatische Übersetzer wichtig, um Platz für neue Symbole zu schaffen.

A0.L wird als Zeiger auf den Symboltabellenanfang zurückgeliefert. Dadurch kann die Symboltabelle auch für eigene Zwecke direkt nach dem Löschen verwendet werden.

Die genaue Struktur der Symboltabelle ist in Kapitel 5.4 erläutert.

Einfaches Beispiel:

Die Symboltabelle wird gelöscht.

```
START:
        MOVEQ     #!SYMCLR,D7        * Gesamte Symboltabelle löschen.
        TRAP      #1
        RTS
```

Wenn man anschließend im Hauptmenü 4 = Symbole aufruft, erscheinen keine Symbole mehr auf dem Bildschirm.

TRAP-Nummer	: **86**		
Befehlsname	: **GETSYM**		
Befehlsgruppe	: Symboltabelle.		
Kurzbeschreibung	: Liefert die Anfangsadresse der Symboltabelle.		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.L = Adresse des Symboltabellenanfangs. A0.L = Adresse des Symboltabellenanfangs.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: ZUWEIS (30)	PRTSYM (84)	SYMCLR (85)
	GETNEXT (87)	PUTNEXT (88)	

Im Register A0.L und D0.L wird die Adresse des Symboltabellenstarts zurückgegeben. Wenn man keine Symbole verwendet, kann man z. B. in eigenen Programmen dorthin seine Variablen legen. Danach sollte man aber sicherheitshalber mit SYMCLR die Symboltabelle löschen, um keine Fehlinformationen zu hinterlassen.

Eine andere Anwendung ist die Manipulation der Symboltabelle, z. B. Werte verändern etc., doch dabei sollte man sehr vorsichtig sein, um nicht die Struktur zu verändern.

Die genaue Struktur der Symboltabelle ist in Kapitel 5.4 erläutert.

Einfaches Beispiel:

Der Symboltabellenstart wird ermittelt und in Register A4.L abgelegt. Jetzt können z.B. eigene Variablen indirekt zu A4.L adressiert werden.

```
START:
        MOVEQ     #!GETSYM,D7       * Symboltabellenanfang holen.
        TRAP      #1
        MOVEA.L   A0,A4             * Anfang nach A4.L.
        RTS
```

TRAP-Nummer	: 87		
Befehlsname	: GETNEXT		
Befehlsgruppe	: Symboltabelle.		
Kurzbeschreibung	: Stellt die aktuelle Länge der Symboltabelle fest.		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.L = Länge der Symboltabelle.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: ZUWEIS (30)	PRTSYM (84)	SYMCLR(85)
	GETSYM (86)	PUTNEXT (88)	

Damit wird die aktuelle Länge der Symboltabelle ins Register D0.L geladen. Die Länge ist aber auf den Bereich 0..$FFFF begrenzt. Dieser Befehl kann z. B. zusammen mit dem GETSYM-Befehl dazu verwendet werden, um einen freien Platz für eigene lokale Variablen zu bestimmen, ohne die Symboltabelle zu gefährden.

Die genaue Struktur der Symboltabelle ist in Kapitel 5.4 erläutert.

Einfaches Beispiel:

Die Endadresse der Symboltabelle wird ermittelt. Dort wird auch der nächste Eintrag z. B. durch ZUWEIS abgelegt.

```
START:
        MOVEQ   #!GETSYM,D7     * Symboltabellenanfang holen.
        TRAP    #1
        MOVEQ   #!GETNEXT,D7    * Länge holen.
        TRAP    #1
        ADDA.L  D0,A0           * Endadresse berechnen.
        RTS
```

TRAP-Nummer	: **88**		
Befehlsname	: **PUTNEXT**		
Befehlsgruppe	: Symboltabelle.		
Kurzbeschreibung	: Setzt die relative Adresse für den nächsten Eintrag.		
Eingaberegister	: D0.W = Wert relativ zum Symboltabellenanfang.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: ZUWEIS (30)	PRTSYM (84)	SYMCLR (85)
	GETSYM (86)	GETNEXT (87)	

Damit kann man die interne Variable NEXTSYM belegen. Dieser Befehl ist für die Anwendung von Übersetzern gedacht und sollte mit großer Sorgfalt verwendet werden. Im Register D0.W steht der neue Wert, der als Offset zum Symboltabellenanfang verwendet wird. Wenn man diesen Wert verändert, wird beim Erzeugen neuer Symbole von da an angefangen, neue Symbole abzulegen.

Die genaue Struktur der Symboltabelle ist in Kapitel 5.4 erläutert.

Einfaches Beispiel:

Die Adresse für den nächsten Eintrag wird um 10 Bytes nach hinten gesetzt.

```
START:
        MOVEQ   #!GETNEXT,D7    * Alten Wert nach D0.L holen.
        TRAP    #1
        ADD.L   #10,D0          * 10 Bytes freihalten.
        MOVEQ   #!PUTNEXT,D7    * Neu setzen aber nur D0.W.
        TRAP    #1
        RTS
```

TRAP-Nummer	: **89**		
Befehlsname	: **GETBASIS**		
Befehlsgruppe	: System-Routinen.		
Kurzbeschreibung	: Es wird die Basis-Adresse des Grundprogramms ermittelt.		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.L = Basisadresse.		
	A0.L = Basisadresse.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: GETRAM (59)	GETVAR (94)	SETA5 (91)
	GETVERS (97)	GETSN (98)	GRUND (124)
	SUCHBIBO (137)	SYSTEM (139)	

Da man das Grundprogramm ab Version 4.0 verschieben kann, gibt es diesen Befehl, mit dem man die aktuelle Anfangsadresse des Grundprogramms ermitteln kann. Das Ergebnis erscheint in Register A0.L und D0.L. Der Wert ist = 0, wenn die ROMs auf Adresse 0 anfangen.

Einfaches Beispiel:

Anfangsadresse der Eproms feststellen.

```
START:
        MOVEQ       #!GETBASIS,D7
        TRAP        #1
        RTS
```

Im Register A0 und D0.L wird die Anfangsadresse des Grundprogramms übergeben. Diese ist nicht mit der Startadresse des Programms zu verwechseln. Denn auf der Anfangsadresse befindet sich der erste Eintrag des ROM-Satzes.

TRAP-Nummer	: **90**		
Befehlsname	: **GETVAR**		
Befehlsgruppe	: System-Routinen.		
Kurzbeschreibung	: Ermittelt die erste Adresse für System-Variablen.		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.L = Variablenadresse. A0.L = Variablenadresse.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: GETRAM (59)	GETBASIS (93)	SETA5 (91)
	GETVERS (97)	GETSN (98)	GRUND (124)
	SUCHBIBO (137)	SYSTEM (139)	

Damit kann man die vom Grundprogramm verwendete RAM-Startadresse ermitteln. Dabei wird der Wert im Register A0.L und D0.L übergeben. Der Wert ist normalerweise $10000. Nur wenn das Grundprogramm auf einer anderen Adresse als Null sitzt, weicht der Wert davon ab. Dies ist z.B. für die CP/M68K- Umgebung interessant.

Normalerweise hat das Register A5.L genau diesen Wert. Nur wenn A5.L z. B. von einem Betriebssystem verändert wird, muß er über GETVAR ermittelt werden.

Einfaches Beispiel:

Erste Variablenadresse holen.

```
START:
        MOVEQ       #!GETVAR,D7         * Adresse holen.
        TRAP        #1
        RTS
```

Trap-Nummer	: **91**		
Befehlsname	: **SETA5**		
Befehlsgruppe	: System-Routinen.		
Kurzbeschreibung	: Setzt das Register A5 mit der Variablenadresse.		
Eingaberegister	: Keine.		
Ausgaberegister	: A5.L = Anfang des Variablenbereichs.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: GETRAM (59)	GETBASIS (93)	GETVAR (90)
	GETVERS (97)	GETSN (98)	GRUND (124)
	SUCHBIBO (137)	SYSTEM (139)	

Ab Version 4.0 des Grundprogramms ist alles verschiebbar. Im Register A5.L wird normalerweise die Startadresse des vom Grundprogramm verwendeten RAM-Bereichs abgelegt. Man darf daher das Register A5.L nicht ohne weiteres für eigene Zwecke verwenden. Wenn man mit dem TRAP-Befehl arbeitet, wird das Register A5.L automatisch zurückgestellt. Verwendet man aber den Befehl JSR, verläßt sich das Grundprogramm darauf, daß der Inhalt des Registers A5.L gültig ist. Hat man den Inhalt des Registers A5.L verändert, kann man ihn mit dem Aufruf SETA5 wieder auf den vom Grundprogramm voreingestellten Wert zurücksetzen.

Einfaches Beispiel:

A5.L wieder restaurieren.

```
START:
        MOVEQ     #!SETA5,D7        * Register A5 belegen.
        TRAP      #1
        RTS
```

TRAP-Nummer	: **92**		
Befehlsname	: **AUFXY**		
Befehlsgruppe	: Schildkrötengrafik.		
Kurzbeschreibung	: Schildkröte absolut setzen.		
Eingaberegister	: D1.W = X-Position. D2.W = Y-Position.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SCHREITE (1)	DREHE (2)	HEBE (3)
	SENKE (4)	SET (7)	SCHR16TEL (19)
	FIRSTTIME (36)	GRAPOFF (39)	HIDE (47)
	SHOW (48)	KORXY (93)	AUFK (94)
	GETK (95)		

Damit kann man die Schildkröte auf eine absolute Koordinate positionieren. Dazu wird eine Linie zur neuen Koordinate gezogen, wenn SENKE aktiv ist. Im Register D1.W steht die X-Koordinate im Bereich von 0...511 und im Register D2.W die Y-Koordinate ebenfalls im Bereich von 0...511 (nicht wie bei MOVETO). Die Schildkröte wird am Endpunkt eingeblendet. Sie verändert ihre Richtung nicht.

Einfaches Beispiel:

Auf dem Bildschirm erscheint eine Linie, die nach links unten geht. Die Schildkröte zeigt aber nach oben.

```
START:
        MOVEQ     #10,D1          * X = 10.
        MOVEQ     #100,D2         * Y = 100.
        MOVEQ     #!AUFXY,D7      * Schildkröte absolut setzen.
        TRAP      #1
        RTS
```

TRAP-Nummer	: 93		
Befehlsname	: **KORXY**		
Befehlsgruppe	: Schildkrötengrafik.		
Kurzbeschreibung	: Schildkrötenkoordinaten lesen.		
Eingaberegister	: Keine.		
Ausgaberegister	: D1.L = X-Position. D2.L = Y-Position.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SCHREITE (1)	DREHE (2)	HEBE (3)
	SENKE (4)	SET (7)	SCHR16TEL (19)
	FIRSTTIME (36)	GRAPOFF (39)	HIDE (47)
	SHOW (48)	AUFXY (92)	AUFK (94)
	GETK (95)		

Mit Hilfe des Befehls kann man den aktuellen Standpunkt der Schildkröte ermitteln. Im Register D1.L wird die X-Koordinate geliefert, im Register D2.L die Y-Koordinate. Dabei wird beide Male der Wert 0...511 verwendet, wenn die Schildkröte sich im sichtbaren Bereich befindet. Sonst ist der Wert entsprechend kleiner oder größer.

Einfaches Beispiel:

Es wird eine Linie vom aktuellen Standpunkt (Mitte des Bildschirms) um 100 Punkte in positive X- und Y-Richtung gezogen.

```
START:
        MOVEQ   #!FIRSTTIME,D7      * Schildkröte anschalten.
        TRAP    #1
        MOVEQ   #!KORXY,D7          * Startpunkt holen.
        TRAP    #1
        ADD     #100,D1             * X + 100.
        ADD     #100,D2             * Y + 100.
        MOVEQ   #!AUFXY,D7          * Linie ziehen.
        TRAP    #1
        RTS
```

Der Aufruf von FIRSTTIME ist wichtig, denn sonst sind die Koordinaten nicht definiert. Es genügt natürlich auch ein Aufruf eines anderen Schildkrötenbefehls, z. B. SCHREITE oder DREHE.

TRAP-Nummer	:	94
Befehlsname	:	AUFK
Befehlsgruppe	:	Schildkrötengrafik.
Kurzbeschreibung	:	Schildkrötenrichtung setzen.
Eingaberegister	:	D0.W = Absoluter Blickwinkel.
Ausgaberegister	:	Keine.
Zerstörte Register	:	Keine.
Ab Version	:	4.0
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	SCHREITE (1), DREHE (2), HEBE (3), SENKE (4), SET (7), SCHR16TEL (19), FIRSTTIME (36), GRAPOFF (39), HIDE (47), SHOW (48), AUFXY (92), KORXY (93), GETK (95)

Die Schildkröte wird auf die angegebene Richtung im Register D0.W gestellt. Dabei wird der absolute Winkel in Grad angegeben. Bei 0 Grad zeigt die Schildkröte nach rechts, bei 90 Grad zeigt sie nach oben.

Einfaches Beispiel:

Die Schildkröte wird um 90 Grad gedreht, blickt dann aber nach rechts.

```
START:
        MOVEQ   #90,D0
        MOVEQ   #!DREHE,D7      * Nur zum Test drehen.
        TRAP    #1
        MOVE    #180,D0
        MOVEQ   #!AUFK,D7       * Nun absoluten Winkel einstellen.
        TRAP    #1
        RTS
```

Komplexeres Beispiel:

Nach dem Start ergibt sich eine recht komplexe Figur. Durch Variation der Winkel und Schrittzahlen kann man sehr unterschiedliche Gebilde erzeugen.

```
START:
        CLR     D5              * Winkelzähler.
        MOVE    #240-1,D6       * Schleifenzähler.
SCHLEIFE:
        MOVE    D5,D0           * Winkel.
        MULS    #10,D0
        MOVEQ   #!AUFK,D7       * In 10 Grad-Schritten.
        TRAP    #1
        MOVEQ   #20,D0          * Schreite 20.
        MOVEQ   #!SCHREITE,D7
        TRAP    #1
        MOVE    D5,D0           * Winkel.
        MULS    #-15,D0         * In -15 Grad-Schritten.
        MOVEQ   #!AUFK,D7
        TRAP    #1
        MOVEQ   #20,D0          * Schreite 20.
        MOVEQ   #!SCHREITE,D7
        TRAP    #1
        ADDQ    #1,D5           * Nächster Winkel.
        DBRA    D6,SCHLEIFE     * Schleifenende.
        RTS
```

TRAP-Nummer	:	**95**
Befehlsname	:	**GETK**
Befehlsgruppe	:	Schildkrötengrafik.
Kurzbeschreibung	:	Liest den absoluten Blickwinkel der Schildkröte.
Eingaberegister	:	Keine.
Ausgaberegister	:	D0.W = Absoluter Blickrichtung in Grad.
Zerstörte Register	:	Keine.
Ab Version	:	4.0
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	SCHREITE (1), DREHE (2), HEBE (3), SENKE (4), SET (7), SCHR16TEL (19), FIRSTTIME (36), GRAPOFF (39), HIDE (47), SHOW (48), AUFXY (92), KORXY (93), AUFK (94)

Damit läßt sich die aktuelle Blickrichtung der Schildkröte ermitteln. Im Register D0.W wird der Wert in Grad übergeben. Der Bereich ist dabei 0...359 Grad.

Einfaches Beispiel:

Die Schildkröte wird zweimal gedreht. Danach wird der Blickwinkel ermittelt.

```
START:
        MOVEQ   #10,D0
        MOVEQ   #!DREHE,D7      * Um 10 Grad drehen.
        TRAP    #1
        MOVEQ   #15,D0
        MOVEQ   #!DREHE,D7      * Um 15 Grad drehen.
        TRAP    #1
        MOVEQ   #!GETK,D7       * Ergebnis in D0.W = 115.
        TRAP    #1
        RTS
```

Im Register D0.W steht der dezimale Wert 115 = $73; denn der Winkel der Endposition setzt sich aus der Grundposition (90 Grad, Schildkröte zeigt nach oben) und den beiden Werten des DREHE-Befehls (10 Grad und 15 Grad) zusammen.

TRAP-Nummer	: **96**		
Befehlsname	: **RND**		
Befehlsgruppe	: Berechnungen.		
Kurzbeschreibung	: Gibt einen Zufallswert zurück.		
Eingaberegister	: D0.W = Bereich des Zufallwertes (Vorzeichenlos).		
Ausgaberegister	: D0.L = Zufallswert.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SIN (23)	COS (24)	WERT (29)
	MULS32 (72)	DIVS32 (73)	ADJ360 (83)

Ein Zufallszahlengenerator. Im Register D0.W übergibt man den Bereich. Es wird als Ergebnis eine Zahl zwischen 0 und n - 1 ebenfalls im Register D0.L zurückgeliefert. Wenn z. B. im Register D0.W der Wert 6 übergeben wurde, erhält man eine Zufallszahl zwischen 0 und 5 (einschließlich). Beim Wert 0 in D0.W wird als Ausgabe ein Bereich von 0 bis $FFFF ausgegeben. Dies ist der maximale Bereich.

Komplexeres Beispiel:

Nach Start des Programms erhält man ein ungleichmäßiges Punktraster auf dem Bildschirm. Nach und nach werden viele Lücken des Rasters gefüllt.

Das Programm kann durch einen Tastendruck abgebrochen werden.

```
START:
        MOVE    #512,D0
        MOVEQ   #!RND,D7        * Zufallszahl im Bereich 0 .. 511.
        TRAP    #1
        MOVE    D0,D1           * X-Position.
        MOVE    #256,D0
        MOVEQ   #!RND,D7        * Zufallszahl im Bereich 0 .. 255.
        TRAP    #1
        MOVE    D0,D2           * Y-Position.
        MOVEQ   #!MOVETO,D7     * Positionieren.
        TRAP    #1
        MOVEQ   #$80,D0         * Befehl für Punkt zeichnen.
        MOVEQ   #!CMD,D7        * Befehl ausführen.
        TRAP    #1
        MOVEQ   #!CSTS,D7       * Auf Zeichen von der Tastatur warten.
        TRAP    #1
        BEQ.S   START           * Wiederholen, bis Taste gedrückt.
        RTS
```

TRAP-Nummer	: **97**		
Befehlsname	: **GETVERS**		
Befehlsgruppe	: System-Routinen.		
Kurzbeschreibung	: Liefert die Versionsnummer des Grundprogramms zurück.		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.L = Versionsnummer.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: GETRAM (59)	GETBASIS (93)	GETVAR (90)
	SETA5 (91)	GETSN (98)	GRUND (124)
	SUCHBIBO (137)	SYSTEM (139)	

Damit wird die in Adresse $40C stehende Versionsnummer des Grundprogramms im Register D0.L übergeben. Version 6.21 liefert z. B. den Wert $621.

Dieser Befehl ist wichtig, wenn man Programme entwickelt, die von Besonderheiten des Systems Gebrauch machen. Damit kann man den Änderungszustand feststellen.

Einfaches Beispiel:

Versionsnummer holen.

```
START:
        MOVEQ     #!GETVERS,D7        * Versionsnummer nach D0.L.
        TRAP      #1
        RTS
```

TRAP-Nummer	: **98**		
Befehlsname	: **GETSN**		
Befehlsgruppe	: System-Routinen.		
Kurzbeschreibung	: Liefert die Seriennummer des Grundprogramms.		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.L = Seriennummer.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: GETRAM (59)	GETBASIS (93)	GETVAR (90)
	SETA5 (91)	GETVERS (97)	GRUND (124)
	SUCHBIBO (137)	SYSTEM (139)	

Damit wird der Wert in Adresse $410 ausgelesen und im Register D0.L abgelegt. Die Seriennummer ist für kommerzielle Anwendungen gedacht, bei der Software geschützt werden soll. Für den normalen Betrieb ist dieser Aufruf ohne Bedeutung.

Einfaches Beispiel:

Seriennummer holen.

```
START:
        MOVEQ     #!GETSN,D7          * Seriennummer nach D0.L.
        TRAP      #1
        RTS
```

Bemerkung:

Die Seriennummer wird bisher nicht benutzt.

TRAP-Nummer	:	99
Befehlsname	:	**CRLF**
Befehlsgruppe	:	Zeichenausgabe.
Kurzbeschreibung	:	Gibt einen Zeilenvorschub (CR LF) über CO2 aus.
Eingaberegister	:	Keine.
Ausgaberegister	:	Carry = Außer bei USERCO immer auf 1 gesetzt.
Zerstörte Register	:	D0
Ab Version	:	4.0
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	CLRSCREEN (20), CO (21), LO (22), SIZE (25), CO2 (33), CRT (49), LST (50), USR (51), NIL (52), SETPASS (55), CURSEIN (61), CURSAUS (62), CHAR (63), CURON (81), CUROFF (82), GETLINE (100), GETCURXY (101), SETCURXY (102), LSTS (117), CO2SER (129)

Mit dem Befehl kann man über die Routine CO2 ein Zeichen CR (Code $D) und ein Zeichen LF (Code $A) ausgegeben. Dadurch wird der Cursor an den nächsten Zeilenanfang gestellt. Wenn der Cursor auf der untersten Zeile stand, wird der Bildschirminhalt um eine Zeile nach oben verschoben (Scroll). Das CARRY-Flag ist dabei immer gesetzt, es sei denn, die CO2-Routine ist auf USERCO (siehe CO2) geschaltet. Dann muß sich USERCO um die Rückgabe des CARRY-Flags kümmern.

Komplexeres Beispiel:

Der Buchstabe B wird direkt unter den Buchstaben A geschrieben.

```
START:
        MOVEQ   #!CLRSCREEN,D7      * Bildschirm für CO2 vorbereiten.
        TRAP    #1
        MOVEQ   #'A',D0             * Buchstaben A ausgeben.
        MOVEQ   #!CO2,D7
        TRAP    #1
        MOVEQ   #10,D0              * Verzögerung.
        MOVEQ   #!DELAY,D7
        TRAP    #1
        MOVEQ   #!CRLF,D7           * Zeilenvorschub.
        TRAP    #1
        MOVEQ   #'B',D0             * Buchstaben B ausgeben.
        MOVEQ   #!CO2,D7
        TRAP    #1
        RTS
```

TRAP-Nummer	:	**100**
Befehlsname	:	**GETLINE**
Befehlsgruppe	:	Zeichenausgabe.
Kurzbeschreibung	:	Adressen der Zeile des Bildwiederholspeichers lesen.
Eingaberegister	:	D0.W = Zeilennumer (0...23).
Ausgaberegister	:	A0.L = Anfangsadresse des Zeile. A1.L = Anzahl der Zeichen in dieser Zeile. A2.L = Adresse der Cursorspalte in dieser Zeile.
Zerstörte Register	:	D0
Ab Version	:	4.0
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	CLRSCREEN (20), CO (21), LO (22), SIZE (25), CO2 (33), CRT (49), LST (50), USR (51), NIL (52), SETPASS (55), CURSEIN (61), CURSAUS (62), CHAR (63), CURON (81), CUROFF (82), CRLF (99), GETCURXY (101), SETCURXY (102), LSTS (117), CO2SER (129)

Damit ist es möglich, auf den Text in einer Zeile des Bildschirmspeichers zuzugreifen. Dazu wird in D0.B die Zeilennummer von 0 bis 23 gewählt. 0 ist dabei die oberste Zeile. Die aktuelle Cursorposition in X-Richtung bestimmt dann noch eine Zeichenstelle.

Im Register A0.L steht die Adresse des Zeilenanfangs. Im Register A1.L steht die Adresse, die auf eine Speicherzelle zeigt, die die aktuelle Zeilenlänge enthält (Byte-Wert). Im Register A2.L steht die Adresse des Zeichens, das auf der Spalte des Cursors sitzt.

Mit diesem Befehl kann man z. B. einfach eine Bildschirm orientierte Eingabe für Programmiersprachen, Formulare oder Steuerungen realisieren.

Komplexeres Beispiel:

Nach Start des Programms erscheint der Cursor links oben. Nun kann man beliebig auf dem Bildschirm Texte schreiben. Am Schluß gibt man CTRL-D ein (erst die Taste CTRL drücken, dann dazu die Taste D). Der Bildschirminhalt wird auf den Drucker ausgegeben. Mit CTRL-C wird das Programm abgebrochen.

Achtung ! Es ist zwar möglich auf dem ganzen Bildschirm Texte einzugeben und zu verändern, allerdings werden zur Cursorsteuerung nicht die Befehle des Editors verwendet, sondern die Kommandos, die beim Befehl CO beschrieben sind.

```
START:
        MOVEQ     #!CLRSCREEN,D7      * Bildschirm für CO2 vorbereiten.
        TRAP      #1
SCHLEIFE:
        MOVEQ     #!CI,D7             * Zeichen einlesen.
        TRAP      #1
        CMP.B     #3,D0               * CTRL-C bedeutet Ende.
        BEQ.S     ENDE
        CMP.B     #4,D0               * CTRL-D bedeutet Hardcopy.
        BEQ.S     HARDCOPY
        MOVEQ     #!CO2,D7
        TRAP      #1
        BRA.S     SCHLEIFE
HARDCOPY:
        CLR       D4                  * Zeilenzähler.
HARDLP0:
        MOVE      D4,D0               * Zeile holen.
        MOVEQ     #!GETLINE,D7
        TRAP      #1
        MOVE.B    (A1),D3             * Anzahl der Zeichen pro Zeile.
```

```
HARDLP1:
        MOVE.B      (A0)+,D0        * Zeichen holen.
        MOVEQ       #!LO,D7         * Zeichen auf Drucker ausgeben.
        TRAP        #1
        SUBQ.B      #1,D3
        BNE.S       HARDLP1         * Bis Zeile ausgegeben wurde.
        MOVEQ       #$D,D0
        MOVEQ       #!LO,D7         * CR ausgeben.
        TRAP        #1
        MOVEQ       #$A,D0
        MOVEQ       #!LO,D7         * LF ausgeben.
        TRAP        #1
        ADDQ        #1,D4
        CMP         #24,D4          * Bis zur letzten Zeile.
        BNE.S       HARDLP0         * Weitere Eingaben.
ENDE:
        RTS
```

Bemerkung:

Bei diesem Programm wird die Druckerausgabe für die HARDCOPY des Bildschirms selber ausgeführt. CO2 ist aber auch in der Lage, den Bildschirm selber auszudrucken (siehe CO2).

TRAP-Nummer	: **101**		
Befehlsname	: **GETCURXY**		
Befehlsgruppe	: Zeichenausgabe.		
Kurzbeschreibung	: Liest die aktuelle Cursorposition zurück.		
Eingaberegister	: Keine.		
Ausgaberegister	: D1.L = X-Position des Cursors.		
	D2.L = Y-Position des Cursors.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: CLRSCREEN (20)	CO (21)	LO (22)
	SIZE (25)	CO2 (33)	CRT (49)
	LST (50)	USR (51)	NIL (52)
	SETPASS (55)	CURSEIN (61)	CURSAUS (62)
	CHAR (63)	CURON (81)	CUROFF (82)
	CRLF (99)	GETLINE (100)	SETCURXY (102)
	LSTS (117)	CO2SER (129)	

Diese Routine ist im Zusammenhang mit CO2, CO und CHAR zu benutzen.
Im Register D1.L steht als Ergebnis der Wert der X-Position (0...79) des Cursors und im Register D2.L der Wert der Y-Position (0...23). Die Koordinate 0,0 ist dabei links oben auf dem Bildschirm.

Komplexeres Beispiel:

Mit diesem Programm kann man auf dem Bildschirm Zeichen eingeben. Drückt man die Tasten CTRL-D, erscheint die aktuelle Zeile, auf der man sich gerade befindet, auf der untersten (25-ten) Zeile blinkend. Zu dem Blinken kommt es, weil die Zeile nur in eine Bildseite eingeschrieben wurde. Das Programm kann mit CTRL-C beendet werden.

```
START:
        MOVEQ     #!CLRSCREEN,D7      * Bildschirm für CO2 vorbereiten.
        TRAP      #1
SCHLEIFE:
        MOVEQ     #!CI,D7             * Zeichen lesen.
        TRAP      #1
        CMP.B     #3,D0
        BEQ.S     ENDE                * CTRL-C = Ende.
        CMP.B     #4,D0
        BEQ.S     COPY                * CTRL-D = Zeile kopieren.
        MOVEQ     #!CO2,D7            * Sonst Zeichen ausgeben.
        TRAP      #1
        BRA.S     SCHLEIFE            * Wiederholen.
COPY:
        MOVEQ     #!GETCURXY,D7       * Cursorposition holen.
        TRAP      #1
        MOVE      D2,D0               * Zeilennummer.
        MOVEQ     #!GETLINE,D7        * Zeileninformationen lesen.
        TRAP      #1
        CLR.B     79(A0)              * Endekennung für WRITE.
        MOVEQ     #$11,D0             * Schriftgröße.
        CLR       D1                  * X = 0.
        CLR       D2                  * Y = 0.
        MOVEQ     #!WRITE,D7          * Zeile ausgeben.
        TRAP      #1
        BRA.S     SCHLEIFE            * Von vorne.
ENDE:
        RTS
```

TRAP-Nummer	: **102**		
Befehlsname	: **SETCURXY**		
Befehlsgruppe	: Zeichenausgabe.		
Kurzbeschreibung	: Setzt die Cursorposition.		
Eingaberegister	: D1.B = X-Position des Cursors. D2.B = Y-Position des Cursors.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: CLRSCREEN (20)	CO (21)	LO (22)
	SIZE (25)	CO2 (33)	CRT (49)
	LST (50)	USR (51)	NIL (52)
	SETPASS (55)	CURSEIN (61)	CURSAUS (62)
	CHAR (63)	CURON (81)	CUROFF (82)
	CRLF (99)	GETLINE (100)	GETCURXY (101)
	LSTS (117)	CO2SER (129)	

Auch diese Routine funktioniert im Zusammenhang mit CO2, CO und CHAR.

Mit ihr kann man den Cursor auf eine beliebige Position des Bildschirms setzen. Dazu wird im Register D1.B die X-Position (0...79) übergeben und im Register D2.B die Y-Position (0...23). Dabei ist die Koordinate 0,0 links oben.

Einfaches Beispiel:

Auf dem Bildschirm erscheint der Buchstabe A auf der vierten Zeichen-Zeile von oben und auf der zwanzigsten Position von links, da die Koordinaten von 0 an gezählt werden.

```
START:
        MOVEQ     #!CLRSCREEN,D7      * Bildschirm für CO2 vorbereiten.
        TRAP      #1
        MOVEQ     #19,D1              * X = 19.
        MOVEQ     #3,D2               * Y = 3.
        MOVEQ     #!SETCURXY,D7       * Cursor setzen.
        TRAP      #1
        MOVEQ     #'A',D0
        MOVEQ     #!CO2,D7            * Zeichen A ausgeben.
        TRAP      #1
        RTS
```

TRAP-Nummer	: **103**		
Befehlsname	: **GETXY**		
Befehlsgruppe	: Grafik.		
Kurzbeschreibung	: Liest die aktuelle GDP-Position.		
Eingaberegister	: Keine.		
Ausgaberegister	: D1.L = X-Position.		
	D2.L = Y-Position.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: MOVETO (8)	DRAWTO (9)	CLR (16)
	CLPG (17)	WAIT (18)	CMD (26)
	NEWPAGE (27)	SETFLIP (34)	SETPEN (37)
	ERAPEN (38)	CMDPRINT (40)	AUTOFLIP (60)
	SETXOR (77)	GETXOR (78)	SETCOLOR (79)
	GETCOLOR (80)	HARDCOPY (125)	GRAFIK (126)
	GDPVERS (127)		

Nach Aufruf der Routine steht im Register D1.L die X-Koordinate und im Register D2.L die Y-Koordinate der aktuellen Zeichenposition des GDPs. Damit wird direkt der Registerinhalt des Graphikprozessors ausgelesen und das Vorzeichen angeglichen.

Einfaches Beispiel:

Nach dem Start erscheinen 6 große Buchstaben A schräg übereinanderstehend.

```
START:
        MOVEQ   #!SYSTEM,D7     * System-Informationen lesen.
        TRAP    #1
        AND     #7,D0           * Nur CPU-Informationen lassen.
        MULS    #$FF73,D0
        MOVEA.L D0,A0           * GDP Port 3.
        MOVEQ   #!WAIT,D7       * Warten, bis GDP fertig.
        TRAP    #1
        MOVE.B  #$55,(A0)       * Schriftgröße einstellen.
        CLR     D1              * X = 0.
        CLR     D2              * Y = 0.
        MOVEQ   #6-1,D3         * Anzahl der Zeichen.
SCHLEIFE:
        MOVEQ   #!MOVETO,D7     * Positionieren.
        TRAP    #1
        MOVEQ   #'A',D0
        MOVEQ   #!CMD,D7        * A ausgeben.
        TRAP    #1
        MOVEQ   #!GETXY,D7      * Neue Position auslesen.
        TRAP    #1
        ADD     #5*8,D2         * Eine Zeile hoch.
        DBRA    D3,SCHLEIFE     * Nächstes Zeichen.
        RTS
```

TRAP-Nummer	: **104**		
Befehlsname	: **SI**		
Befehlsgruppe	: SER-Baugruppe.		
Kurzbeschreibung	: Zeichen von serieller Schnittstelle lesen.		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.L = Gelesenes Zeichen.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Die Handshakeleitung wird nicht mehr bedient. Dafür ist jetzt SI2 vorhanden. Kompatibilität mit Version 4.3 und früher ist jetzt wieder gewährleistet.		
Siehe auch	: SO (105)	SISTS (106)	SOSTS (107)
	SIINIT (108)	SER (128)	SI2 (138)

Eingabe eines Zeichens über die serielle Schnittstelle. Es wird ein Zeichen von der seriellen Schnittstelle SER in das Register D0.B gelesen. Das Zeichen kann einen Wert zwischen 0 und $FF haben. Die oberen Bits des Registers D0.L sind auf 0 gesetzt. Es wird so lange gewartet, bis ein Zeichen angekommen ist.

Achtung ! Dieser Befehl sollte nicht verwendet werden, wenn keine SER vorhanden ist, da sich das Programm sonst in einer Endlosschleife verlaufen könnte. Ob eine SER vorhanden ist kann mit dem Befehl SYSTEM abgefragt werden.

Wie serielle Baugruppen miteinander verbunden werden und wie man sie einsetzen kann, ist im Handbuch der SER-Baugruppe beschrieben, weshalb hier auf die Darstellung der Anschlußbelegungen verzichtet wird.

Komplexeres Beispiel:

Wenn man das Programm startet, werden alle ankommenden Zeichen auf dem Bildschirm ausgegeben. Dabei ist eine Baudrate von 9600 Baud eingestellt. Wenn man andere Baudraten haben will, kann man sie mit dem SIINIT-Befehl einstellen (siehe SIINIT für Codierung). Das Programm wird durch einen Tastendruck abgebrochen.

```
START:
        MOVEQ     #!CLRSCREEN,D7      * Bildschirm für CO2 vorbereiten.
        TRAP      #1
        MOVEQ     #$1E,D0             * 9600 Baud, 1 Stop-Bit, 8 Bit.
        MOVEQ     #$0B,D1             * Keine Parität.
        MOVEQ     #!SIINIT,D7         * Werte einstellen.
        TRAP      #1
SCHLEIFE:
        MOVEQ     #!SI,D7             * Zeichen von serieller Karte lesen.
        TRAP      #1
        MOVEQ     #!CO2,D7            * Zeichen ausgeben.
        TRAP      #1
        MOVEQ     #!CSTS,D7           * Abbruch durch Taste?
        TRAP      #1
        BEQ.S     SCHLEIFE            * Nein, dann wiederholen.
        RTS
```

Achtung! Der Cursor blinkt nicht bei diesem Programm. Dazu müßte man die Routinen CURON, CUROFF und AUTOFLIP verwenden.

TRAP-Nummer	: **105**		
Befehlsname	: **SO**		
Befehlsgruppe	: SER-Baugruppe.		
Kurzbeschreibung	: Sendet ein Zeichen über die serielle Schnittstelle.		
Eingaberegister	: D0.B = Auszugebendes Zeichen.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SI (104)	SISTS (106)	SOSTS (107)
	SIINIT (108)	SER (128)	SI2 (138)

Ausgabe eines Zeichens über die serielle Schnittstelle. Das Zeichen steht zuvor im Register D0.B und wird mit Hilfe der SER-Baugruppe ausgegeben. Die Ausgabe erfolgt aber nur, wenn der Eingang DSR auf +12V liegt, sonst wird die Ausgabe gesperrt und das Programm wartet, bis die Bedingung erfüllt ist. Der Eingang DSR ist an die Stiftleiste geführt. Die +12V findet man normalerweise am Ausgang RTS, dessen Pegel man mit SIINIT programmieren kann.

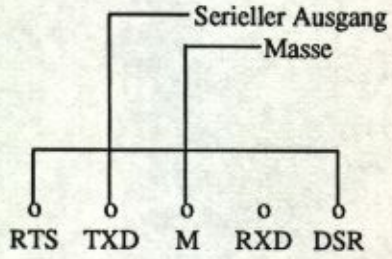
Achtung ! Dieser Befehl sollte nicht verwendet werden, wenn keine SER vorhanden ist, da sich das Programm sonst in einer Endlosschleife verlaufen könnte. Ob eine SER vorhanden ist kann mit dem Befehl SYSTEM abgefragt werden.

Komplexeres Beispiel:

Die serielle Baugruppe wird initialisiert. Es wird dauernd der Wert $5A ausgegeben, bis eine Taste gedrückt wird.

```
START:
        MOVEQ     #$1E,D0       * 9600 Baud, 1 Stop-Bit, 8 Bit.
        MOVEQ     #$0B,D1       * Keine Parität.
        MOVEQ     #!SIINIT,D7   * Wert einstellen.
        TRAP      #1
SCHLEIFE:
        MOVEQ     #!CSTS,D7
        TRAP      #1
        BNE.S     ENDE          * Abbruch durch Tastaturbetätigung
        MOVEQ     #!SOSTS,D7
        TRAP      #1
        BEQ.S     SCHLEIFE      * Wiederholen bis SER bereit zur Ausgabe
        MOVEQ     #$5A,D0       * Testwert.
        MOVEQ     #!SO,D7       * Zeichen ausgeben.
        TRAP      #1
        BRA.S     SCHLEIFE      * Wiederholen
ENDE:
        RTS
```

Wenn der Eingang DSR nicht beschaltet wird, wird die Ausgabe gesperrt. Daher hier die Belegung:



Der Ausgang RTS liefert die +12V, die zur Freigabe des DSR-Eingangs nötig sind.

TRAP-Nummer	: **106**		
Befehlsname	: **SISTS**		
Befehlsgruppe	: SER-Baugruppe.		
Kurzbeschreibung	: Zeichen von serieller Baugruppe vorhanden?		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.L = Kennung, ob Wert vorhanden ist. Flags.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SI (104)	SO (105)	SOSTS (107)
	SIINIT (108)	SER (128)	SI2 (138)

Damit kann man prüfen, ob ein Zeichen an der seriellen Schnittstelle angekommen ist. Die Routine liefert den Wert $FFFFFFFF im Register D0.L, wenn ein Zeichen bereit ist, sonst den Wert 0.

Achtung! Der Status ist kurzzeitig nach einer Ausgabe über SO ungültig. Außerdem kann diese Routine nicht zusammen mit SI2 verwendet werden.

Komplexeres Beispiel:

Mit diesem Programm arbeitet der Computer wie ein Terminal. Zeichen, die auf der Tastatur eingegeben werden, werden über die serielle Schnittstelle ausgegeben. Zeichen, die von der seriellen Schnittstelle kommen, werden auf dem Bildschirm ausgegeben.

```
START:
        MOVEQ     #!CLRSCREEN,D7     * Bildschirm für CO2 vorbereiten.
        TRAP      #1
        MOVEQ     #$1E,D0            * 9600 Baud, 1 Stop-Bit, 8 Bit.
        MOVEQ     #$0B,D1            * Keine Parität.
        MOVEQ     #!SIINIT,D7        * Werte einstellen.
        TRAP      #1
TERMINAL:
        MOVEQ     #!CURSEIN,D7       * Cursor darstellen.
        TRAP      #1
        MOVEQ     #!SISTS,D7         * Zeichen von serieller Karte da?
        TRAP      #1
        BEQ.S     TERMINA0           * Nein, dann weiter.
        MOVEQ     #!CURSAUS,D7       * Cursor ausschalten.
        TRAP      #1
        MOVEQ     #!SI,D7            * Zeichen von SER holen.
        TRAP      #1
        MOVEQ     #!CO2,D7           * Zeichen ausgeben.
        TRAP      #1
        MOVEQ     #!CURSEIN,D7       * Cursor wieder einschalten.
        TRAP      #1
TERMINA0:
        MOVEQ     #!CSTS,D7          * Zeichen von Tastatur da?
        TRAP      #1
        BEQ.S     TERMINA1           * Nein, dann weiter.
        MOVEQ     #!CI,D7            * Zeichen holen.
        TRAP      #1
        MOVEQ     #!SO,D7            * Zeichen über SER ausgeben.
        TRAP      #1
TERMINA1:
        MOVEQ     #!AUTOFLIP,D7      * Cursor soll blinken.
        TRAP      #1
        BRA.S     TERMINAL           * Immer weiter.
```

Bemerkung:

Wenn man die Leitungen RxD mit TxD verbindet und RTS mit DSR, kann man auf dem Bildschirm schreiben; denn die Zeichen, die man eintippt, erscheinen auf dem Bildschirm. Das ist auch ein guter Test dafür, ob die serielle Schnittstelle korrekt arbeitet.

TRAP-Nummer	: **107**		
Befehlsname	: **SOSTS**		
Befehlsgruppe	: SER-Baugruppe.		
Kurzbeschreibung	: Serielle Baugruppe bereit für Ausgabe?		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.L = Kennung, ob bereit. Flags.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SI (104)	SO (105)	SISTS (106)
	SIINIT (108)	SER (128)	SI2 (138)

Der Befehl ermöglicht die Feststellung, ob ein Zeichen über die serielle Schnittstelle ausgegeben worden ist und somit ein neues Zeichen ausgegeben werden kann.

Im Register D0.L erscheint der Wert $FFFFFFFF, wenn der Ausgabebuffer des seriellen Bausteins auf der SER-Karte leer ist. Dann ist die serielle Schnittstelle bereit, ein neues Zeichen auszugeben. Sonst erscheint der Wert 0 im Register D0.L.

Dieser Befehl ist wichtig, wenn man eine serielle Ausgabe im Parallelbetrieb zu anderen Operationen ausführen will, ohne den Computer warten zu lassen, bis das vorhergehende Zeichen ausgegeben wurde.

Komplexeres Beispiel:

Es wird mit der Schildkrötengrafik ein Kreis gezeichnet. Im Hintergrund wird über die serielle Schnittstelle immer $5A ausgegeben.

```
START:
        MOVEQ     #$18,D0          * 1200 Baud, 1 Stop-Bit, 8 Bit.
        MOVEQ     #$0B,D1          * Keine Parität.
        MOVEQ     #!SIINIT,D7      * Wert einstellen.
        TRAP      #1
SCHLEIFE:
        MOVEQ     #1,D0
        MOVEQ     #!SCHREITE,D7    * 1 Punkt schreiten.
        TRAP      #1
        MOVEQ     #1,D0
        MOVEQ     #!DREHE,D7       * 1 Grad drehen.
        TRAP      #1
        MOVEQ     #!SOSTS,D7       * Bereit zur Ausgabe?
        TRAP      #1
        BEQ.S     SCHLEIFE         * Nein, dann weiter.
        MOVEQ     #$5A,D0
        MOVEQ     #!SO,D7          * Zeichen ausgeben.
        TRAP      #1
        MOVEQ     #!CSTS,D7        * Zeichen von Tastatur da?
        TRAP      #1
        BEQ.S     SCHLEIFE         * Nein, dann wiederholen.
        RTS
```

Bemerkung:

Wenn man die Befehle SOSTS und BEQ.S SCHLEIFE wegläßt, wird der Kreis nur sehr langsam durchlaufen; denn dann wartet der Computer jedesmal auf die Beendigung der Ausgabe.
Diesen Effekt kann man aber z. B. bei 9600 Baud nicht mehr so deutlich sehen. Sobald nämlich der Computer zum Durchlauf der Schleife eine ähnlich große Zeit wie für die Übertragung eines Zeichens benötigt, arbeitet SO wieder verlustfrei. Denn in SO wird zunächst abgefragt, ob das vorhergehende Zeichen abgesendet wurde, und dann das nächste Zeichen ausgegeben. Danach wartet die Routine aber nicht, bis das Zeichen auch übertragen wurde.

TRAP-Nummer	: **108**		
Befehlsname	: **SIINIT**		
Befehlsgruppe	: SER-Baugruppe.		
Kurzbeschreibung	: Initialisierung der SER-Baugruppe.		
Eingaberegister	: D0.B = Wert für Control-Register. D1.B = Wert für Command-Register.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SI (104)	SO (105)	SISTS (106)
	SOSTS (107)	SER (128)	SI2 (138)

Damit wird der Baustein 6551 auf der seriellen Schnittstelle SER programmiert. Man kann so Baudrate, Stopbits, Parität etc. einstellen. Dazu werden zwei Parameter übergeben. Im Register D0.B steht der Wert für das Control-Register und im Register D1.B der Wert für das Command-Register.

Control-Register (D0.B):

Bits: Funktion:

0 - 3 Ausgabegeschwindigkeit einstellen:

0000	16x externer Takt
0001	50 Baud
0010	75 Baud
0011	109.92 Baud
0100	134.58 Baud
0101	150 Baud
0110	300 Baud
0111	600 Baud
1000	1200 Baud
1001	1800 Baud
1010	2400 Baud
1011	3600 Baud
1100	4800 Baud
1101	7200 Baud
1110	9600 Baud
1111	19200 Baud

4 0 = externer Takt
1 = interner Takt

5 - 6 Anzahl der Datenbits:

00	8 Datenbits
01	7 Datenbits
10	6 Datenbits
11	5 Datenbits

7 Anzahl der Stop-Bits:

0	1 Stop-Bit
1	2 Stop-Bits oder 1 1/2 Stop-Bits bei 5 Bits ohne Parität

Command-Register (D1.B):

Bits: Funktion:

0 0 = DTR auf 1-Signal
1 = Rxd und Txd freigeben

1 0 = Interrupt freigeben
1 = Interrupt sperren

2 - 3 Interrupt und RTS-Leitung:

00 = Interrupt gesperrt	-RTS 1-Signal
01 = Interrupt frei	-RTS 0-Signal
10 = Interrupt gesperrt	-RTS 0-Signal
11 = Interrupt gesperrt	-RTS 0-Signal und BREAK auf TxD-Leitung

4 0 = Normal
1 = Echo

5 - 7 Parität:

XX0	Keine Parität	
001	Ungerade Parität	(odd)
011	Gerade Parität	(even)
101	Gesetzte Parität	(mark)
111	Ungesetzte Parität	(space)

Mit den Werten aus diesen beiden Tabellen kann der SER-Baustein programmiert werden. Dazu werden aus den binären Mustern die gewünschten Bytes zusammengesetzt. Hier noch einige gebräuchliche Kombinationen:

D0.B	D1.B	Bemerkung:
$1E	$0B	9600 Baud, 8 Bit, keine Parität, 1 Stop
$1C	$0B	4800 Baud, 8 Bit, keine Parität, 1 Stop
$1A	$0B	2400 Baud, 8 Bit, keine Parität, 1 Stop
$18	$0B	1200 Baud, 8 Bit, keine Parität, 1 Stop
$98	$0B	1200 Baud, 8 Bit, keine Parität, 2 Stop

Wann immer möglich, sollte man diese Kombinationen bevorzugen, um Kompatibilität unter den NDR-Klein-Computer-Benutzern zu erreichen.

Einfaches Beispiel:

SER initialisieren.

```
START:
        MOVEQ     #$1E,D0        * 9600 Baud, 1 Stop-Bit, 8 Bit.
        MOVEQ     #$0B,D1        * Keine Parität.
        MOVEQ     #!SIINIT,D7    * Werte an serielles Interface.
        TRAP      #1
        RTS
```

TRAP-Nummer	: **109**		
Befehlsname	: **GETAD8**		
Befehlsgruppe	: AD/DA-Baugruppen.		
Kurzbeschreibung	: Wert vom 8-Bit AD-Wandler lesen.		
Eingaberegister	: D0.B = Kanalnummer.		
Ausgaberegister	: D0.L = Gewandelter Wert.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: GETAD10 (110)	SETDA (111)	SETDA12 (134)
	GETAD12 (135)		

Im Register D0.B wird die Kanalnummer im Bereich von 0 bis $F angegeben. Damit wird ein Kanal des AD-Umsetzers auf der Baugruppe AD 8 x 16 angesprochen und die Wandlung gestartet. Nach der Wandlung wird das Ergebnis im Register D0.L als 8-Bit-Größe abgeliefert. Die Wandlung dauert ca. 100 Mikrosekunden.

Komplexeres Beispiel:

Auf dem Bildschirm wird für jeden Kanal eine Linie dargestellt. Es erscheinen 16 Linien mit der jeweiligen Amplitude des Meßwertes.

```
START:
        CLR     D5              * Schreibseite.
        MOVEQ   #1,D6           * Leseseite.
SCHLEIF0:
        CLR     D3              * Start bei X = 0.
        MOVEQ   #16-1,D4        * Y-Achse und Kanalnummer.
SCHLEIF1:
        MOVE    D5,D0           * Schreibseite.
        MOVE    D6,D1           * Leseseite.
        MOVEQ   #!NEWPAGE,D7    * Seiten einstellen.
        TRAP    #1
        MOVE    D3,D1           * X-Achse.
        MOVEQ   #!ERAPEN,D7     * Löschmodus.
        TRAP    #1
        MOVE    #255,D2
        MOVEQ   #!MOVETO,D7     * Positionieren.
        TRAP    #1
        CLR     D2
        MOVEQ   #!DRAWTO,D7     * Alte Linie löschen.
        TRAP    #1
        MOVE    D4,D0           * Kanalnummer 15...0.
        MOVEQ   #!GETAD8,D7     * Wert lesen.
        TRAP    #1
        MOVE    D0,D2
        MOVEQ   #!SETPEN,D7     * Schreibmodus.
        TRAP    #1
        MOVEQ   #!DRAWTO,D7     * Neue Linie (Amplitude).
        TRAP    #1
        ADD     #20,D3          * Neue X-Koordinate.
        DBRA    D4,SCHLEIF1     * Nächster Kanal.
        EXG.L   D5,D6           * Schreibseite und Leseseite tauschen.
        MOVEQ   #!CSTS,D7
        TRAP    #1
        BEQ.S   SCHLEIF0        * Wiederholen, bis Taste gedrückt.
        RTS
```

TRAP-Nummer	:	**110**
Befehlsname	:	**GETAD10**
Befehlsgruppe	:	AD/DA-Baugruppen.
Kurzbeschreibung	:	Wert vom 10-Bit AD-Wandler lesen.
Eingaberegister	:	Keine.
Ausgaberegister	:	D0.L = Gewandelter Wert.
Zerstörte Register	:	Keine.
Ab Version	:	4.0
Änderungen zu 4.3	:	Nein.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	GETAD8 (109) SETDA (111) SETDA12 (134) GETAD12 (135)

Damit wird ein 10-Bit-Wert in das Register D0.L gelesen. Dazu wird der AD-Umsetzer der Baugruppe AD10 x 1 gestartet.

Da die Baugruppe nur einen Kanal besitzt, entfällt die Verwendung einer Kanalnummer. Der Baustein wandelt in ca. 20 Mikrosekunden einen Wert. Bei sehr schnellen Vorgängen empfiehlt sich die Verwendung eines Sample- und Hold-Elementes, um sich schnell ändernde Vorgänge während der Meßdauer konstant zu halten.

Komplexeres Beispiel:

Bis zum Tastendruck wird alle 1/10 Sekunde der aktuelle Wert des AD-Wandlers geholt und ausgegeben.

```
START:
        MOVEQ     #!GETAD10,D7      * Wert lesen.
        TRAP      #1
        LEA       BUFFER(PC),A0     * Zieladresse.
        MOVEQ     #!PRINT4D,D7      * Wert dezimal ausgeben.
        TRAP      #1
        MOVEQ     #5-1,D0
SCHLEIFE:
        MOVE.B    #' ',(A0)+        * Rest mit Leerzeichen auffüllen.
        DBRA      D0,SCHLEIFE
        CLR.B     (A0)              * Endekennung neu setzen.
        LEA       BUFFER(PC),A0     * Textadresse.
        MOVEQ     #$33,D0           * Schriftgröße.
        MOVEQ     #10,D1            * X-Koordinate.
        MOVEQ     #120,D2           * Y-Koordinate.
        MOVEQ     #!WRITE,D7        * Zahl ausgeben.
        TRAP      #1
        MOVEQ     #1,D0
        MOVEQ     #!DELAY,D7        * 1/10 Sekunde warten.
        TRAP      #1
        MOVEQ     #!CSTS,D7
        TRAP      #1
        BEQ.S     START             * Wiederholen, bis Taste gedrückt.
        RTS
BUFFER:
        DS.B      12                * Ablage für Zahl.
```

TRAP-Nummer	: 111		
Befehlsname	: **SETDA**		
Befehlsgruppe	: AD/DA-Baugruppen.		
Kurzbeschreibung	: DA-Wandler setzen.		
Eingaberegister	: D1.B = Wert für Kanal 0. D2.B = Wert für Kanal 1.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: GETAD8 (109) GETAD12 (135)	GETAD10 (110)	SETDA12 (134)

Mit diesem Befehl kann man die Baugruppe DA8 x 2 ansprechen, die zwei Digital/Analog-Umsetzer enthält. Dazu wird im Register D1.B der Wert für den Kanal 0 angegeben und im Register D2.B der Wert für den Kanal 1. Jeder der Umsetzer kann eine 8 Bit-Größe verarbeiten. Die Wandlung erfolgt in ca. 800 ns (Nanosekunden).

Komplexeres Beispiel:

Wenn man Kanal 0 auf die X-Ablenkung eines Skops legt und Kanal 1 auf die Y-Ablenkung, erhält man ein Quadrat auf dem Bildschirm. Dieses Quadrat entsteht durch die sägezahnförmigen Schwingungen am Ausgang der beiden Kanäle.

```
START:
        CLR     D1              * Wert für Kanal 0.
        CLR     D2              * Wert für Kanal 1.
        LEA     TABELLE(PC),A0  * Additions-Tabelle.
        CLR     D3              * Startoffset der Tabelle.
SCHLEIF0:
        MOVE    #255-1,D4       * Schleifenzähler.
SCHLEIF1:
        MOVEQ   #!SETDA,D7      * DA-Wandler setzen.
        TRAP    #1
        ADD     0(A0,D3.W),D1   * Neuer Wert Kanal 0.
        ADD     2(A0,D3.W),D2   * Neuer Wert Kanal 1.
        DBRA    D4,SCHLEIF1     * Wiederholen.
        ADDQ    #4,D3           * Neuer Index.
        AND     #$F,D3          * Nur 0, 4, 8, 12.
        MOVEQ   #!CSTS,D7
        TRAP    #1
        BEQ.S   SCHLEIF0        * Wiederholen, bis Taste gedrückt.
        RTS
TABELLE:                        * Daten für Quadrat.
        DC.W    1,0,0,1,-1,0,0,-1
```

Nach Start des Programms erscheint der Buchstabe R auf dem Skop-Bild, wenn man Kanal 0 mit der X-Ablenkung und Kanal 1 mit der Y-Ablenkung verbindet. Diese Art der Zeichenerzeugung nennt man Vektorscan.

```
START:
        CLR     D1              * Wert für Kanal 0.
        CLR     D2              * Wert für Kanal 1.
        LEA     TABELLE(PC),A0  * Additions-Tabelle.
        CLR     D3              * Startoffset der Tabelle.
SCHLEIF0:
        MOVEQ   #63-1,D4        * Schleifenzähler.
SCHLEIF1:
        MOVEQ   #!SETDA,D7      * DA-Wandler setzen.
        TRAP    #1
        ADD     0(A0,D3.W),D1   * Neuer Wert Kanal 0.
        ADD     2(A0,D3.W),D2   * Neuer Wert Kanal 1.
        DBRA    D4,SCHLEIF1     * Wiederholen.
        ADDQ    #4,D3           * Neuer Index.
        CMP     #TABENDE-TABELLE,D3
        BCS.S   SCHLEIF0        * Bis Tabellenende wiederholen.
        MOVEQ   #!CSTS,D7
        TRAP    #1
        BEQ.S   START           * Wiederholen, bis Taste gedrückt.
        RTS
TABELLE:
        DC.W    0,2,0,2,2,0,2,0,0,-2,-2,0,-2,0,2,-1,2,-1
TABENDE:
```

Daneben gibt es aber auch ein Rasterscan-Verfahren, so wie es sich bei der Erzeugung des Bildes durch den GDP (Graphik Display Prozessor) ergibt. Ein Beispiel für das Rasterscan-Verfahren auf dem Skop ist hier als Anregung gezeigt.
Wenn man das Programm startet, erscheint ein Text auf dem Skop (Kanal 0 an X und Kanal 1 an Y).

```
START:
        LEA     TABELLE(PC),A0      * Adresse Bildraster.
        MOVE    #130,D3             * X-Position außerhalb.
        MOVEQ   #127,D2             * Y-Position.
        MOVEQ   #16-1,D4            * 16 Durchgänge.
SCHLEIF1:
        CLR     D1                  * X-Position.
        MOVE    (A0)+,D5            * Werte holen.
        MOVEQ   #16-1,D6            * 16 Punkte.
SCHLEIF2:
        BTST    D6,D5               * Null, dann dunkel.
        BEQ.S   SPRUNG0
        MOVEQ   #!SETDA,D7          * Punkt darstellen.
        TRAP    #1
        BRA.S   SPRUNG1             * Überspringen.
SPRUNG0:
        EXG     D1,D3
        MOVEQ   #!SETDA,D7          * Punkt nicht darstellen.
        TRAP    #1
        EXG     D3,D1
SPRUNG1:
        ADDQ    #8,D1               * Neue X-Position.
        DBRA    D6,SCHLEIF2         * Wiederholen.
        SUBQ    #8,D2               * Neue Y-Position.
        DBRA    D4,SCHLEIF1         * Wiederholen.
        MOVEQ   #!CSTS,D7
        TRAP    #1
        BEQ.S   START               * Wiederholen, bis Taste gedrückt.
        RTS
TABELLE:
        DC.W    $F3C9               * Bildwiederholspeicher.
        DC.W    $892A               * Text als Punktraster.
        DC.W    $892C
        DC.W    $F128
        DC.W    $A12C
        DC.W    $912A
        DC.W    $8BC9
        DC.W    0,0,0,0
        DC.W    $FFFF
        DC.W    $8001,$8001,$8001
        DC.W    $FFFF
```

TRAP-Nummer	: **112**
Befehlsname	: **SPEAK**
Befehlsgruppe	: Sprache und Sound.
Kurzbeschreibung	: Sprachausgabe mit 5 Parametern.
Eingaberegister	: A0.L = Adresse des Parameterblocks.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 4.0
Änderungen zu 4.3	: Nein.
Änderungen zu 6.1	: Nein.
Siehe auch	: SPEAK1 (113) SOUND (114)

Mit diesem Befehl kann die Baugruppe SPRACHE bedient werden, die einen Sprachsynthese-IC mit der Bezeichnung SSI 263 verwendet. Bei dem IC handelt es sich um einen sogenannten Phonem-Synthesizer.

Bei diesem Befehl können alle 5 Parameter des Bausteins eingestellt werden. Das Register A0.L erhält dazu die Adresse des Parameterblocks. Das erste Wort des Parameterblocks enthält die Anzahl der Phoneme. Dann folgen pro Phonem jeweils fünf Bytes mit den Daten.

Einfaches Beispiel:

Nach Start des Programms gibt die Sprachkarte das Wort HALLO aus. Durch Ändern der Parameter können unterschiedliche Klangfarbe und Betonung erreicht werden (siehe Beschreibung SSI 263).

```
START:
        LEA     TABELLE(PC),A0      * Adresse der Daten.
        MOVEQ   #!SPEAK,D7          * Sprache ausgeben.
        TRAP    #1
        RTS

TABELLE:
        DC.W    (TABENDE-TABELLE-2)/5
        DC.B    $00,$58,$A8,$5C,$EA
        DC.B    $00,$58,$A8,$5C,$EA
        DC.B    $0A,$68,$D8,$50,$EA
        DC.B    $2C,$50,$A8,$5C,$EA
        DC.B    $0F,$40,$A8,$52,$EA
        DC.B    $0F,$28,$A8,$5A,$EA
        DC.B    $E2,$20,$A8,$5C,$EA
        DC.B    $22,$20,$A8,$5C,$EA
        DC.B    $11,$40,$A8,$5C,$EA
        DC.B    $12,$40,$A8,$5C,$EA
        DC.B    $D6,$30,$A8,$5C,$EA
        DC.B    $00,$38,$A8,$5C,$EA
TABENDE:
```

TRAP-Nummer	: **113**	
Befehlsname	: **SPEAK1**	
Befehlsgruppe	: Sprache und Sound.	
Kurzbeschreibung	: Sprachausgabe nur Phonemcode.	
Eingaberegister	: A0.L = Adresse des Parameterblocks.	
Ausgaberegister	: Keine.	
Zerstörte Register	: Keine.	
Ab Version	: 4.0	
Änderungen zu 4.3	: Nein.	
Änderungen zu 6.1	: Nein.	
Siehe auch	: SPEAK (112)	SOUND (114)

Mit diesem Befehl kann die Baugruppe SPRACHE bedient werden, die einen Sprachsynthese-IC mit der Bezeichnung SSI 263 verwendet. Bei dem IC handelt es sich um einen sogenannten Phonem-Synthesizer.

Bei dem Befehl kann nur 1 Parameter, das Phonem, angegeben werden. Dadurch ergibt sich eine gröbere Aussprache, jedoch benötigt man nur 1 Byte pro Phonem. Die restlichen Parameter werden vom Programm auf Standard-Werte eingestellt.

Das Register A0.L erhält beim Aufruf die Adresse des Parameterblocks. Das erste Wort des Parameterblocks enthält die Anzahl der Phoneme. Dann folgt pro Phonem jeweils ein Byte.

Einfaches Beispiel:

Nach Start des Programm "spricht" der Rechner das Wort SIEBEN.

```
START:
        LEA     TABELLE(PC),A0      * Startadresse der Werte.
        MOVEQ   #!SPEAK1,D7         * Sprache ausgeben.
        TRAP    #1
        RTS

TABELLE:
        DC.W    (TABENDE-TABELLE-2)
        DC.B    $00,$00,$2F,$06,$06,$24,$02,$38,$00,$00
TABENDE:
```

TRAP-Nummer	: **114**
Befehlsname	: **SOUND**
Befehlsgruppe	: Sprache und Sound.
Kurzbeschreibung	: SOUND-Generator setzen.
Eingaberegister	: A0.L = Adresse des Paramterblocks.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 4.0
Änderungen zu 4.3	: Nein.
Änderungen zu 6.1	: Nein.
Siehe auch	: SPEAK (112) SPEAK1 (113)

Damit kann man die Baugruppe SOUND bedienen. Dazu wird im Register A0.L die Adresse des Parameterblocks übergeben. Dort stehen die 16 Werte, die an das IC AY-3-8912 übertragen werden.

Die Werte werden nacheinander an die 16 Register des IC geschickt.

Einfaches Beispiel:

Nach Start des Programms wird ein rythmisches Klanggebilde über den Lautsprechers ausgegeben.

```
START:
        LEA     TABELLE(PC),A0      * Startadresse der Werte.
        MOVEQ   #!SOUND,D7          * Sound ausgeben.
        TRAP    #1
        RTS
TABELLE:
        DC.B    $DE,$01,$DD,$01,$BE,$00
        DC.B    $00,$F8,$10,$10,$10,$00,$0A,$08,$00,$00
```

TRAP-Nummer : **115**
Befehlsname : **GETUHR**
Befehlsgruppe : Echtzeituhren.
Kurzbeschreibung : Uhrzeit und Datum lesen.

Eingaberegister : A0.L = Adresse des Zielbuffers.
Ausgaberegister : A0.L = Zeigt auf das nächste Byte hinter dem Buffer.
Zerstörte Register : Keine.
Ab Version : 4.0
Änderungen zu 4.3 : Es werden 8 Bytes anstatt 7 ausgegeben, da auch 100-tel Sekunden verfügbar sind.
Änderungen zu 6.1 : Nein.
Siehe auch : SETUHR (116) UHRPRINT (140)

Mit dieser Funktion können die Uhrenbaugruppe mit dem IC E050-16 sowie die SMART-WATCH angesprochen werden. Dabei wird die Uhrzeit sowie das Datum gelesen.

Es werden, je nachdem welche Uhr vorhanden ist, alle verfügbaren Daten gelesen. Dabei können bei der Uhrenbaugruppe keine 1/10 und 1/100 Sekunden gelesen werden. Dieses Byte wird auf Null gesetzt.

Im Register A0.L wird die Adresse eines Buffers übergeben. Nach dem Aufruf stehen im Buffer die Daten in einem BCD-Format, sodaß man die Werte direkt ausgeben kann.

Das Format des Buffers lautet (mit Ausgabebeispiel):

0	1	2	3	4	5	6	7
Stunden	Minuten	Datum	Monat	Jahr	Tag	Sekunde	100-tel Sekunden
$18	$15	$23	$09	$84	$01	$00	$00

Einfaches Beispiel:

Auf dem Bildschirm wird die Zeit in Stunden, Minuten und Sekunden ausgegeben. Man kann das Programm natürlich auch ergänzen, um Datum, Jahr und Wochentag auszugeben (siehe UHRPRINT).

```
START:
        LEA     UHRBUF(PC),A0       * Ablageadresse für Uhrdaten.
        MOVEQ   #!GETUHR,D7         * Uhrzeit und Datum holen.
        TRAP    #1
        LEA     AUSBUF(PC),A0       * Ablageadresse für Ausgabe.
        LEA     UHRBUF(PC),A1
        MOVE.B  (A1),D0             * Stunden.
        MOVEQ   #!PRINT2X,D7        * Wegen BCD-Darstellung.
        TRAP    #1
        MOVE.B  #' ',(A0)+          * Leerstelle.
        MOVE.B  1(A1),D0            * Minuten.
        MOVEQ   #!PRINT2X,D7        * Ausgabe.
        TRAP    #1
        MOVE.B  #' ',(A0)+          * Leerstelle.
        MOVE.B  6(A1),D0            * Sekunden.
        MOVEQ   #!PRINT2X,D7        * Ausgabe.
        TRAP    #1
        LEA     AUSBUF(PC),A0       * Adresse Ausgabetext.
        MOVEQ   #$33,D0             * Schriftgröße.
        MOVE.W  #170,D1             * X-Position.
        MOVEQ   #120,D2             * Y-Position.
        MOVEQ   #!WRITE,D7          * Text ausgeben.
        TRAP    #1
        MOVEQ   #5,D0
        MOVEQ   #!DELAY,D7          * Eine halbe Sekunde warten.
        TRAP    #1
        MOVEQ   #!CSTS,D7
        TRAP    #1
        BEQ.S   START               * Wiederholen, bis Taste gedrückt.
        RTS
UHRBUF:
        DS.B    8
AUSBUF:
        DS.B    10
```

TRAP-Nummer	: **116**
Befehlsname	: **SETUHR**
Befehlsgruppe	: Echtzeituhren.
Kurzbeschreibung	: Uhrzeit und Datum setzen.
Eingaberegister	: A0.L = Adresse der Uhrdaten.
Ausgaberegister	: A0.L = Zeigt auf das nächste Byte hinter dem Buffer.
Zerstörte Register	: Keine.
Ab Version	: 4.0
Änderungen zu 4.3	: Es werden 8 Bytes verlangt, da auch 100-tel Sekunden gesetzt werden können.
Änderungen zu 6.1	: Nein.
Siehe auch	: GETUHR (115) UHRPRINT (140)

Der Befehl ermöglicht es, entweder die Uhr auf der Uhrenbaugruppe oder die SMART-WATCH anzusprechen.

Mit dem Befehl wird die Uhrzeit und das Datum eingestellt. Die Uhrenbaugruppe ignoriert die 1/10 und 1/100 Sekunden.

Im Register A0.L wird die Adresse eines Buffers übergeben. Die Daten werden in dem Buffer übergeben.

Das Format des Buffers lautet:

0	1	2	3	4	5	6	7
Stunden	Minuten	Datum	Monat	Jahr	Tag	Sekunde	100-tel Sekunden
$18	$15	$23	$09	$84	$01	$00	$00

Einfaches Beispiel:

Nach dem Aufruf wird der Uhrenbaustein gestellt. Die Zeile bei BUFFER muß natürlich mit den aktuellen Daten gefüllt werden. Anschließend kann man das Ergebnis mit dem Programm des GETUHR-Befehls testen.

```
START:
        LEA     UHRZEIT(PC),A0          * Adresse der Uhrzeit.
        MOVEQ   #!SETUHR,D7
        TRAP    #1
        RTS
UHRZEIT:                                * Uhrzeit und Datum.
        DC.B    $18,$15,$23,$09,$84,$01,$00,$00
```

TRAP-Nummer	:	**117**
Befehlsname	:	**LSTS**
Befehlsgruppe	:	Zeichenausgabe.
Kurzbeschreibung	:	Drucker bereit?
Eingaberegister	:	Keine.
Ausgaberegister	:	D0.L = Kennung, ob Drucker bereit ist. Flags(d0.b).
Zerstörte Register	:	Keine.
Ab Version	:	4.0
Änderungen zu 4.3	:	Eine Umlenkung auf die serielle Karte kann mit dem Befehl SER eingestellt werden.
Änderungen zu 6.1	:	Nein.
Siehe auch	:	CLRSCREEN (20), CO (21), LO (22), SIZE (25), CO2 (33), CRT (49), LST (50), USR (51), NIL (52), SETPASS (55), CURSEIN (61), CURSAUS (62), CHAR (63), CURON (81), CUROFF (82), CRLF (99), GETLINE (100), GETCURXY (101), SETCURXY (102), CO2SER (129)

Mit dem Befehl kann man prüfen, ob der Drucker das letzte Zeichen schon ausgegeben hat und bereit für den Empfang eines neuen Zeichens ist. Der Befehl ermöglicht es auch, den Drucker im Hintergrundbetrieb laufen zu lassen und gleichzeitig mit einem anderen Programm zu arbeiten. Wenn der Drucker bereit ist, wird im Register D0.L der Wert $FF übergeben, sonst der Wert 0.

Ist die Routine auf die serielle Karte gelenkt, wird entweder der Wert $FFFFFFFF oder 0 zurückgegeben.

Komplexeres Beispiel:

Während der Kreis gezeichnet wird, gibt es eine Druckausgabe. Wenn man den Drucker anhält, wird das Kreisprogramm dennoch nicht unterbrochen. Da kein LF (Code: $A) ausgeben wird, erfolgt die Ausgabe immer auf der gleichen Zeile, um Papier zu sparen.

```
START:
        LEA     BUFFER(PC),A0       * Adresse des Textes.
SCHLEIFE:
        MOVEQ   #!CSTS,D7
        TRAP    #1
        BNE.S   ENDE                * Abbruch auf Tastendruck.
        MOVEQ   #1,D0
        MOVEQ   #!SCHREITE,D7       * 1 Punkt schreiten.
        TRAP    #1
        MOVEQ   #1,D0
        MOVEQ   #!DREHE,D7          * 1 Grad drehen.
        TRAP    #1
        MOVEQ   #!LSTS,D7           * Drucker bereit?
        TRAP    #1
        BEQ.S   SCHLEIFE            * Nein!
        MOVE.B  (A0)+,D0            * Zeichen holen.
        BEQ.S   START               * Letzes Zeichen, dann von vorne.
        MOVEQ   #!LO,D7             * Zeichen auf Drucker ausgeben.
        TRAP    #1
        BRA.S   SCHLEIFE            * Wiederholen.
ENDE:
        RTS
BUFFER:                             * Ausgabetexte
        DC.B    'Der Hintergrundbetrieb des Druckers'
        DC.B    ' druckt diesen Text dauernd aus',$D,0
```

TRAP-Nummer	: **118**		
Befehlsname	: **RELAN**		
Befehlsgruppe	: CAS-Baugruppe.		
Kurzbeschreibung	: Schaltet Relais auf der CAS-Baugruppe an.		
Eingaberegister	: Keine.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: RI (14)	PO (15)	RELAUS (119)

Damit kann man das Relais (falls vorhanden) auf der CAS2-Baugruppe einschalten. Das Relais kann den Kassettenmotor bedienen. Es wird auch automatisch vom Grundprogramm eingeschaltet, wenn man über die Menüs Daten aufzeichnet oder lädt.

Einfaches Beispiel:

Das Relais wird angeschaltet.

```
START:
        MOVEQ     #!RELAN,D7        * Relais anschalten.
        TRAP      #1
        RTS
```

TRAP-Nummer	: **119**		
Befehlsname	: **RELAUS**		
Befehlsgruppe	: CAS-Baugruppe.		
Kurzbeschreibung	: Schaltet Relais auf der CAS-Baugruppe aus.		
Eingaberegister	: Keine.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: RI (14)	PO (15)	RELAN (118)

Es wird das Relais (falls vorhanden) auf der CAS2-Baugruppe ausgeschaltet. Dieses Relais kann den Kassettenmotor bedienen.

Einfaches Beispiel:

Relais ausschalten.

```
START:
        MOVEQ   #!RELAUS,D7     * Relais ausschalten.
        TRAP    #1
        RTS
```

TRAP-Nummer	: **120**		
Befehlsname	: **ASSERR**		
Befehlsgruppe	: Assembler.		
Kurzbeschreibung	: Fehleranzahl beim Assemblerdurchlauf lesen.		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.L = Fehleranzahl.		
Zerstörte Register	: Keine.		
Ab Version	: 4.0		
Änderungen zu 4.3	: Nein.		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SETERR (53)	GETERR (54)	ASSEMBLE (65)
	GETORG (68)	PUTORG (69)	

Die Anzahl der Fehler, die bei einer Übersetzung durch den Assembler auftreten, werden im Register D0.L übergeben. Die Maximalzahl ist dabei $FFFF. 0 bedeutet, es ist kein Fehler aufgetreten. Dieser Befehl wird von höheren Programmiersprachen verwendet, um festzustellen, ob Fehler bei einem Assemblerlauf aufgetreten sind, und dann das Programm nicht zu starten.

Komplexeres Beispiel:

```
START:
        MOVEQ     #!GETSTX,D7       * Alter Textstart.
        TRAP      #1
        MOVE.L    D0,-(A7)          * Merken.
        MOVE.L    #BUFFER,D0
        MOVEQ     #!PUTSTX,D7       * Neuen Textstart einstellen.
        TRAP      #1
SCHLEIFE:
        MOVEQ     #!EDIT,D7         * Editor aufrufen.
        TRAP      #1
        MOVEQ     #!ASSEMBLE,D7     * Assembler aufrufen.
        TRAP      #1
        BCS.S     SCHLEIFE          * Abbruch beim Assemblieren.
        MOVEQ     #!ASSERR,D7
        TRAP      #1
        BNE.S     SCHLEIFE          * Fehleranzahl ist nicht Null.
        MOVE.L    (A7)+,D0
        MOVEQ     #!PUTSTX,D7       * Alten Textstart wieder einstellen.
        TRAP      #1
        RTS
        DS.B      1000              * Platz für Macro-Tabelle.
BUFFER:
        DC.B      'ORG ?+$2000',0   * Adresse muß eingestellt werden.
```

Bemerkung:

Nach dem Start des Programms meldet sich der Editor. Man kann jetzt ein Assemblerprogramm eingeben. Betätigt man anschließend die Tasten CTRL-K und X, wird automatisch der Assembler gestartet. Ist ein Fehler im Assemblerprogramm aufgetreten, wird automatisch der Texteditor wieder aufgerufen, damit man den Fehler beseitigen kann. Das Menü meldet sich erst dann wieder, wenn man einen korrekten Assemblertext eingegeben hat. Wichtig ist die ORG-Adresse am Anfang. Man darf sie nicht vergessen, sonst überschreibt das neue Programm das alte, und es gibt Fehler. Dieses Programm ist im Grundprogramm realisiert. Dort kann man mit Hilfe des Befehls CTRL-KA den Assembler vom Editor aus aufrufen.

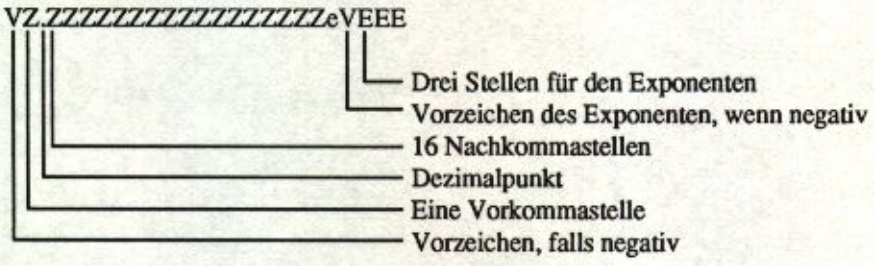
TRAP-Nummer	: **121**		
Befehlsname	: **PRINTFP0**		
Befehlsgruppe	: 68020-FPU.		
Kurzbeschreibung	: Wert in FP0 in ASCII wandeln.		
Eingaberegister	: D0.W = Anzahl der Stellen und Rundungsart. A0.L = Zieladresse für Zahl. FP0.X = Zu wandelnder Wert.		
Ausgaberegister	: A0.L = Zeigt auf Endekennung.		
Zerstörte Register	: Keine.		
Ab Version	: 5.0 (Nur 68020 Grundprogramm)		
Änderungen zu 4.3	: ——		
Änderungen zu 6.1	: Nein.		
Siehe auch	: GETFLOAT (122)	PRTFP0 (145)	FPUWERT (146)
	SETFPX (147)	SETFPY (148)	SETFPZ (149)

Dieser Befehl dient der Unterstützung der FPU (68881) im 68020-Grundprogramm. Beim Aufruf im 68008- oder 68000-Grundprogramm wird nur ein Rücksprung ausgeführt.

Mit diesem Programm können Zahlen, die im Register FP0.X der FPU stehen, in eine ASCII-Darstellung gebracht werden. Dazu muß das Register FP0.X den Wert enthalten. Das Register A0.L muß auf einen Buffer zeigen, der genügend Stellen hat, damit alle Zeichen abgelegt werden können. Nach dem Aufruf zeigt A0.L dann auf das Ende des Textes, das durch eine Null gekennzeichnet wird. Das Register D0.W hat noch eine spezielle Funktion. Da der Wert aus dem Register der FPU mit dem Befehl FMOVE.P FP0,<EA>{D0} gelesen wird, muß in D0.W ein Rundungswert angegeben werden. Dieser Wert ist im Befehl für die FPU festgelegt und kann folgende Werte haben:

-64 bis 0	Gibt an, wie viele Stellen auf der rechten Seite des Dezimalpunktes ausgegeben werden sollen.
+1 bis +17	Gibt die Anzahl der Stellen in der Mantisse an.
+18 bis +63	Wird wie +17 behandelt, aber zusätzlich wird das OPERR-Bit im Register FPSR der FPU gesetzt.

Alle Ziffern, die durch die Rundung abgeschnitten werden, werden automatisch auf Null gesetzt und mit ausgegeben. Dadurch hat die Ausgabe immer das gleiche Format.



Es wird im Gegensatz zum Befehl PRTFP0 nicht geprüft, ob im Register FP0.X wirklich eine gültige Zahl vorhanden ist.

Einfaches Beispiel:

Eine FP-Zahl wird ins Register FP0 geladen und in ASCII-Darstellung im Buffer abgelegt. Das Ergebnis kann man mit ANSEHEN überprüfen.

```
START:
        LEA       BUFFER(PC),A0        * Ziel für ASCII-Zeichen.
        FMOVE.X   #-123.456789E-123,FP0 * Auszugebende Zahl.
        MOVEQ     #17,D0               * Alle Stellen ausgeben, nicht runden.
        MOVEQ     #!PRINTFP0,D7        * Programm aufrufen.
        TRAP      #1
        RTS
BUFFER:                                * Ziel für ASCII-Zeichen.
        DS.B      26
```

Komplexeres Beispiel:

Es werden 100 FP-Zahlen über CO2 ausgegeben.

```
START:
        MOVEQ       #!CLRSCREEN,D7      * Bildschirm für CO2 vorbereiten.
        TRAP        #1
        MOVEQ       #$1B,D0
        MOVEQ       #!CO2,D7
        TRAP        #1
        MOVEQ       #'2',D0             * HARDSCROLL anschalten, falls GDPHS vorhanden.
        MOVEQ       #!CO2,D7
        TRAP        #1
        FMOVE.X     #1.12345678,FP0     * Erste Zahl festlegen.
        MOVEQ       #100-1,D3           * 100 Durchläufe.
SCHLEIF0:
        LEA         BUFFER(PC),A0       * Ziel für Zahl.
        MOVEQ       #17,D0              * Alle Stellen.
        MOVEQ       #!PRINTFP0,D7       * Zahl in ASCII-Darstellung bringen.
        TRAP        #1
        FADD.W      D3,FP0              * Addieren.
        FDIV.W      #10,FP0             * Irgend eine neue Zahl.
        LEA         BUFFER(PC),A0       * Dort steht Zahl.
SCHLEIF1:
        MOVE.B      (A0)+,D0            * Bis Endekennung erscheint.
        BEQ.S       SPRUNG0
        MOVEQ       #!CO2,D7            * Über CO2 ausgeben.
        TRAP        #1
        BRA.S       SCHLEIF1
SPRUNG0:
        MOVEQ       #!CRLF,D7           * Zeilenvorschub.
        TRAP        #1
        DBRA        D3,SCHLEIF0         * Wiederholen.
        RTS
BUFFER:                                 * Ziel für ASCII-Zeichen.
        DS.B        24
```

TRAP-Nummer	: **122**		
Befehlsname	: **GETFLOAT**		
Befehlsgruppe	: 68020-FPU.		
Kurzbeschreibung	: FP-Zahl in ASCII für FPU vorbereiten.		
Eingaberegister	: A0.L = Zahl in ASCII-Darstellung. A1.L = Ziel für FP-Zahl in Packed BCD-Darstellung.		
Ausgaberegister	: A0.L = Zeigt hinter die ASCII-Zahl. Carry = Fehleranzeige.		
Zerstörte Register	: Keine.		
Ab Version	: 5.0 (Nur bei 68020-Grundprogramm)		
Änderungen zu 4.3	: ——		
Änderungen zu 6.1	: Nein.		
Siehe auch	: PRTFP0 (121)	PRTFP0 (145)	FPUWERT (146)
	SETFPX (147)	SETFPY (148)	SETFPZ (149)

Dieser Befehl dient der Unterstützung der FPU (68881) im 68020-Grundprogramm. Beim Aufruf im 68008- oder 68000-Grundprogramm wird nur ein Rücksprung ausgeführt.

Mit diesem Programm wird eine Floating-Point-Zahl in ASCII-Zeichendarstellung in eine Zahl gewandelt, die direkt in ein Register der FPU geladen werden kann (PBCD-Darstellung). Dabei zeigt A0.L auf die zu wandelnde Zahl, A1.L zeigt auf einen Freiraum von 3 Langworten für die gewandelte Zahl. Nach dem Programmaufruf zeigt A0.L direkt hinter die Zahl, also auf die Endenull oder ein Leerzeichen. Wenn ein Fehler aufgetreten ist, zeigt A0.L direkt auf den Fehler. Das CARRY-Flag ist gesetzt, wenn ein Fehler aufgetreten ist.

Für eine richtige Bearbeitung muß die ASCII-FP-Zahl mindestens eine Dezimalzahl enthalten. Es können ein Vorzeichen, ein Komma, beliebig viele Nachkommastellen und ein Exponent mit Vorzeichen vorhanden sein.

Beispiele für richtige Zahlen :

```
1
-1.1234567
+763463.342124523e-234
-2342142.234242E+12
21342134.124124e12
```

Die Anzahl der Ziffern wird intern auf 17 Stellen begrenzt, da die FPU nicht mehr verarbeitet. Außerdem wird der Exponent nach 3 Stellen abgeschnitten, sodaß bei größeren Zahlen der Aufruf dieses Programms sinnlos ist, da das Ergebnis völlig falsch wäre.

Nähere Informationen über die Darstellung einer PBCD-Zahl in FP-Darstellung kann Büchern über die FPU entnommen werden.

Einfaches Beispiel:

Eine FP-Zahl in ASCII-Darstellung wird in das Register FP0.X der FPU geschrieben. Das Ergebnis kann im Einzelschritt überprüft werden.

```
START:
        LEA     ZAHL(PC),A0         * Dort steht die Zahl.
        LEA     ZIEL(PC),A1         * Hier soll sie hin.
        MOVEQ   #!GETFLOAT,D7       * Wandlung durchführen.
        TRAP    #1
        FMOVE.P (A1),FP0            * Zahl ins Register FP0.
        RTS
ZAHL:                               * Zu wandelnde Zahl.
        DC.B    '-1234.56789E-123',0
        DS      0                   * Auf Wortgrenze (nicht unbedingt nötig).
ZIEL:                               * Dort steht nachher FP-Zahl.
        DS.P    1
```

Komplexeres Beispiel:

Es ist die Eingabe einer Zahl möglich, die dann ausgegeben wird, wenn die Wandlung erfolgreich war. Wird bei der Eingabe nur ein RETURN gedrückt, endet das Programm.

```
START:
        LEA         BUFFER(PC),A0       * Ziel für Text.
        MOVEQ       #$21,D0             * Schriftgröße.
        MOVEQ       #10,D1              * X-Position.
        MOVEQ       #100,D2             * Y-Position.
        MOVEQ       #40,D3              * Maximale Anzahl von Zeichen.
        MOVEQ       #!READ,D7           * Zeichen einlesen.
        TRAP        #1
        TST         D4
        BEQ.S       ENDE                * Ende, wenn RETURN gedrückt wird.
        LEA         BUFFER(PC),A0       * Dort steht Zahl.
        LEA         ZIEL(PC),A1         * Dort soll sie hin.
        MOVEQ       #!GETFLOAT,D7       * Wandlung.
        TRAP        #1
        BCS.S       START               * Fehler.
        FMOVE.P     (A1),FP0            * In Register FP0 der FPU.
        LEA         BUFFER(PC),A0       * Ziel für Text.
        MOVEQ       #18,D0              * Alle Ziffern ausgeben.
        MOVE        #!PRTFP0,D7         * Wandlung in ASCII.
        TRAP        #1
        MOVEQ       #20-1,D0
SCHLEIFE:
        MOVE.B      #' ',(A0)+          * Rest auffüllen, damit alles
        DBRA        D0,SCHLEIFE         * überschrieben wird.
        CLR.B       (A0)                * Neue Endekennung.
        LEA         BUFFER(PC),A0       * Dort steht Zahl.
        MOVEQ       #$21,D0             * Schriftgröße.
        MOVEQ       #10,D1              * X-Position.
        MOVE        #140,D2             * Y-Position.
        MOVEQ       #!WRITE,D7          * Ausgabe.
        TRAP        #1
        BRA.S       START               * Von vorn.
ENDE:
        RTS
BUFFER:
        DS.B        42                  * Ziel für Zahlen in ASCII-Darstellung.
ZIEL:
        DS.P        1                   * Ziel für Zahl in FP-Darstellung.
```

TRAP-Nummer	:	123
Befehlsname	:	**READAUS**
Befehlsgruppe	:	Texteingabe.
Kurzbeschreibung	:	Texteingabe mit vorheriger Ausgabe.
Eingaberegister	:	D0.B = Schriftgröße. D1.W = X-Koordinate. D2.W = Y-Koordinate. D3.W = Zusätzliche Anzahl Zeichen zum Ausgabetext - 1. A0.L = Auszugebender Text mit Endenull.
Ausgaberegister	:	D4.L = Wirkliche Anzahl Zeichen. D5.L = Letztes eingegebenes Zeichen. A0.L = Adresse der Endekennung. Carry = 1, wenn mit <ESC> beendet.
Zerstörte Register	:	Keine.
Ab Version	:	6.0
Änderungen zu 4.3	:	——
Änderungen zu 6.1	:	Wenn CR gedrückt wird, wird nicht mehr der Text hinter der Cursorposition abgeschnitten. Der Eingabetext bleibt unverändert. Zum Abschneiden des Textes hinter dem Cursor kann der Befehl CTRL-T verwendet werden. Endeleerzeichen werden bis zur Cursorposition abgeschnitten. Die Eingabe kann wie mit CR auch mit ESC beendet werden.
Siehe auch	:	READ (11)

Dieser Befehl arbeitet ähnlich dem READ-Befehl, deshalb werden hier nur die Unterschiede dargestellt.

Beim READ-Befehl wird nur eine Eingabe in einem Fenster erwartet. Bei diesem Befehl kann vorher in dem Textfenster ein Text als Defaultwert ausgegeben werden. Dazu wird auf der Adresse, auf die A0.L zeigt, ein Text abgelegt, der mit einer Null enden muß. Wird dort nur eine Null abgelegt, ist der Befehl READAUS mit dem Befehl READ fast identisch. Ein weiterer Unterschied betrifft das Register D3.W. Dort steht beim Befehl READ die maximale Anzahl der Zeichen, die im Textfenster Platz haben. Beim Befehl READAUS hingegen wird in D3.W die Anzahl der Zeichen angegeben, die zusätzlich zum ausgegebenen Text Platz im Textfenster haben sollen. Außerdem wird dann noch ein Zeichen hinzugefügt, damit auch bei der Angabe 0 in D3.W noch Platz für den Cursor ist, der immer hinter den Text gesetzt wird.

Beispiel:

D3.W = 3 => Freie Zeichen hinter dem Text = 4

Einfaches Beispiel:

In einem Textfenster wird ein Text ausgegeben. Hinter diesem Text steht der Cursor. Mit ihm kann der Text wie bei READ bearbeitet werden. Wird RETURN gedrückt, wird der alte Text ausgegeben. Hinter dem Text sind dann wieder 11 Zeichen Platz. Der Cursor steht danach wieder hinter dem Text. Ist das Textfenster leer und wird RETURN oder ESC gedrückt, endet das Programm.

```
START:
        MOVEQ     #$21,D0          * Schriftgröße.
        MOVEQ     #10,D1           * X-Position.
        MOVEQ     #100,D2          * Y-Position.
        MOVEQ     #10,D3           * Anzahl der zusätzlichen Zeichen.
        LEA       BUFFER(PC),A0    * Default-Text.
        MOVEQ     #!READAUS,D7     * Programm aufrufen.
        TRAP      #1
        BCS.S     ENDE             * Ende durch ESC
        TST       D4
        BNE.S     START            * Weiter, wenn Buffer nicht leer ist.
ENDE:
        RTS
BUFFER:                            * Default-Text.
        DC.B      'Dies ist ein Test-Text',0
        DS.B      80               * Freiraum.
```

TRAP-Nummer	: **124**
Befehlsname	: **GRUND**
Befehlsgruppe	: System-Routinen.
Kurzbeschreibung	: Grundprogramm oder Teile davon als Unterprogramm aufrufen.
Eingaberegister	: D0.W = Auswahl des Aufrufs.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 6.0
Änderungen zu 4.3	: ——
Änderungen zu 6.1	: Nein.
Siehe auch	: GETRAM (59), GETBASIS (93), GETVAR (90), SETA5 (91), GETVERS (97), GETSN (98), SUCHBIBO (137), SYSTEM (139)

Mit diesem Befehl können das Grundprogramm bzw. Teile davon als Unterprogramm aufgerufen werden. Dabei wird in D0.W eine Auswahl getroffen. So kann man auch von Betriebssystemen aus ein eigenes Grundprogramm-Menü erzeugen oder z. B. den Epromer aufrufen, ohne daß in das Grundprogramm gesprungen werden muß.

Ist D0.W positiv, wird das Grundprogramm als Ganzes aufgerufen, d. h. es erscheint das Grundprogrammenü. Bei einer geraden positiven Zahl (0, 2, 4, ..) wird die alte Menüform benutzt. Bei einer ungeraden positiven Zahl (1, 3, 5, ..) erscheint die neue Menüform auf dem Bildschirm. Aus Kompatibilitätsgründen sollten jedoch nur die Zahlen 0 oder 1 verwendet werden. Bei der neuen Menüform ist beim Aufruf GRUND die Funktion Z verfügbar, die eine Rückkehr in das aufrufende Programm ermöglicht. Also muß man zur Rückkehr auch von der alten Menüform auf die neue Form umschalten.

Wenn D0.W negativ ist, werden die einzelnen Menüpunkte aufgerufen. Dabei sind die Programme in der gleichen Reihenfolge angeordnet wie beim einseitigen Menü. Wenn eine Bildschirmausgabe erfolgt, wird vorher der Bildschirm gelöscht. Einige dieser Programme wie EDITOR oder ASSEMBLER können auch direkt aufgerufen werden. Dazu stehen eigene Unterprogramme zur Verfügung. Manchmal liefern diese Programme Werte in Registern oder FLAGS zurück. Diese Ausgabe erfolgt beim Aufruf durch GRUND nicht. So kann z. B. beim Aufruf des Editors nicht abgefragt werden, ob er mit CTRL-KX oder CTRL-KQ beendet wurde. Beim Aufruf mit EDIT ist das möglich. GRUND ist eigentlich nur dazu gedacht, daß sich jeder Anwender eine eigene Oberfläche schaffen kann bzw. daß das Grundprogramm auch von einem Betriebssystem aus aufgerufen werden kann.

Wert in D0.W:	Funktion:
-1	Ändern.
-2	Starten.
-3	Ansehen.
-4	Symbole ausgeben.
-5	Editor.
-6	Assembler.
-7	Bibliothek.
-8	Text auf CAS speichern.
-9	Daten auf CAS speichern.
-10	Text von CAS laden.
-11	Daten von CAS laden.
-12	Text von CAS prüfen.
-13	Daten von CAS prüfen.
-14	Booten.
-15	Eprom programmieren.
-16	Eprom lesen.
-17	Speicherbereiche ausgeben.
-18	Druckmenü.
-19	IO lesen.
-20	IO setzen.
-21	Einzelschritt aufrufen.
-22	Menüform ändern.
-23	Alten Textstart festlegen.
-24	Neuen Textstart festlegen.
-25	Symboltabelle löschen.
-26	CO auf 40 Zeichen einstellen.
-27	CO auf 80 Zeichen einstellen.
-28	Debug einschalten.
-29	Debug ausschalten.
-30	Nur Fehlerausgabe.
-31	Ausgabe auf Bildschirm.
-32	Ausgabe auf Drucker.
-33	Ausgabe auf Drucker ohne LINEFEED.

Einfache Beispiele:

Grundprogramm mit alter Menüform aufrufen.

```
START:
        MOVEQ       #0,D0           * Alte Menüform.
        MOVEQ       #!GRUND,D7
        TRAP        #1
        RTS
```

Grundprogramm mit neuer Menüform aufrufen.

```
START:
        MOVEQ       #1,D0           * Neue Menüform.
        MOVEQ       #!GRUND,D7
        TRAP        #1
        RTS
```

ANSEHEN aufrufen.

```
START:
        MOVEQ       #-3,D0          * Ansehen.
        MOVEQ       #!GRUND,D7
        TRAP        #1
        RTS
```

DRUCKMENÜ aufrufen.

```
START:
        MOVEQ       #-18,D0         * Druckmenü.
        MOVEQ       #!GRUND,D7
        TRAP        #1
        RTS
```

TRAP-Nummer	: **125**
Befehlsname	: **HARDCOPY**
Befehlsgruppe	: Grafik.
Kurzbeschreibung	: Programmpaket zur Ansteuerung der HARDCOPY-Baugruppe.
Eingaberegister	: D0.W = Auswahl des Programms und Spezialfunktionen. Zusätzlich noch verschiedene Register, die vom Unterprogramm abhängen.
Ausgaberegister	: Ausgaberegister sind vom Unterprogramm abhängig.
Zerstörte Register	: Immer D7 Sonstige Register vom Unterprogramm abhängig.
Ab Version	: 6.0
Änderungen zu 4.3	: ——
Änderungen zu 6.1	: Es kann jedes beliebige Eingabegerät (Maus, Trackball, Joystick) angesprochen werden. D0 bei Mausabfragen (Tasten) ist als Langwort gültig. Die Druckerausgabe ist durch wahlweise Vergrößerung wesentlich verbessert worden.
Siehe auch	: MOVETO (8), DRAWTO (9), CLR (16), CLPG (17), WAIT (18), CMD (26), NEWPAGE (27), SETFLIP (34), SETPEN (37), ERAPEN (38), CMDPRINT (40), AUTOFLIP (60), SETXOR (77), GETXOR (78), SETCOLOR (79), GETCOLOR (80), GETXY (103), GRAFIK (126), GDPVERS (127)

Dieses ist kein einzelnes Unterprogramm, sondern eine Programmsammlung. Es stehen 16 Unterprogramme zur Verfügung, die zum Betrieb mit der HARDCOPY-MAUS-Baugruppe benötigt werden. Die Maus kann auch über die serielle Karte betrieben werden oder mit der KEY3.

Da die Unterprogramme verschiedene Ein- und Ausgaberegister verlangen, sind sie aufgeteilt, wobei die Angaben wieder im Kopf aufgeführt sind. Die Auswahl des Programms erfolgt durch das Register D0.W. Dabei werden nur die untersten 4 Bit für das Programm berücksichtigt. Die anderen Bits haben teilweise noch verschiedene Funktionen. Im folgenden ist die Unterprogrammnummer identisch mit der Zahl in D0.W.

*
**

Unterprogrammnummer	: **0**
Programmfunktion	: **Mausposition relativ abfragen.**
Eingaberegister	: Keine.
Ausgaberegister	: D0.L = Bits der Maustasten. D1.W = Bewegung in X-Richtung seit der letzten Abfrage. D2.W = Bewegung in Y-Richtung seit der letzten Abfrage.

Diese Funktion liefert in D1.W die Bewegung der Maus in X-Richtung und in D2.W die Bewegung der Maus in Y-Richtung zurück. Außerdem wird in D0.L der Wert des Ports ausgegeben, an dem die Maustasten angeschlossen sind. Die Lage der Bits hängt von der Maus, der Anschlußbelegung und der Anzahl der Maustasten ab und sollte deshalb im Handbuch der Baugruppe bzw. der Maus nachgelesen werden. Normalerweise ist das Bit 7 die linke Taste und Bit 6 die rechte Taste.

Zuerst springt diese Routine auf die Adresse $B5E im Variablenbereich. Im Normalfall wird von dort aus auf die Treiberroutine im Grundprogramm gesprungen. Wird aber der Sprung (JMP) auf eine andere Adresse durchgeführt, so kann jede Maus angesprochen werden. Es wird in den Registern D1.W und D2.W eine 0 übergeben. In die Register soll dann die Bewegung der Maus seit der letzten Abfrage geschrieben werden. In D0.L müssen noch die Bits der Maustasten übergeben werden, wobei ein gesetztes Bit besagt, daß die Taste nicht gedrückt ist. Nur Register D7.L darf zerstört werden.

Einfaches Beispiel:

Liest die Mausbewegung seit der letzen Abfrage und die aktuellen Maustasten. Das Programm endet erst, wenn die linke Maustaste gedrückt wird, wenn sie mit Bit 7 verbunden ist.

```
START:
        MOVEQ     #0,D0          * Kennung für Programm.
        MOVEQ     #!HARDCOPY,D7
        TRAP      #1
        TST.B     D0             * Wiederholen, bis linke Maustaste gedrückt.
        BMI.S     START
        RTS
```

*
**

Unterprogrammnummer : **1**
Programmfunktion : **Mausposition absolut abfragen.**
Eingaberegister : D1.W = Letzte X-Position der Maus.
D2.W = Letzte Y-Position der Maus.
Ausgaberegister : D0.L = Bits der Maustasten.
D1.W = Neue X-Position der Maus.
D2.W = Neue Y-Position der Maus.

Auch dieses Unterprogramm ist zur Bedienung der Maus gedacht. Allerdings müssen in D1.W und in D2.W die letzten Positionen der Maus übergeben werden. Auf diese beiden Register wird die Bewegung der Maus seit der letzten Abfrage addiert. Außerdem wird in D0.L der Wert der Maustasten übergeben.

Mit dieser Funktion kann man, wenn D1.W und D2.W nicht zerstört werden, immer die aktuelle Mausposition erfahren, sofern das Programm dauernd aufgerufen wird.

Auch diese Funktion kann durch eine Änderung des JMP auf eine beliebige Adresse umgelenkt werden (Siehe Funktion 0). Der Programmaufruf ist dabei der selbe mit dem Unterschied, daß D1.W und D2.W mit der letzten Mausposition belegt sind.

Komplexeres Beispiel:

Es wird die Mausbewegung seit der letzten Abfrage auf D1.W und D2.W addiert und in D0.L der Wert der Maustasten übergeben. Die aktuelle Mausposition wird auf dem Bildschirm ausgegeben. Das Programm endet mit Betätigung der linken Maustaste.

```
START:
        MOVEQ       #100,D1             * X-Anfang = 100.
        MOVEQ       #100,D2             * Y-Anfang = 100.
SCHLEIF0:
        MOVEQ       #1,D0               * Kennung für Programm.
        MOVEQ       #!HARDCOPY,D7
        TRAP        #1
        TST.B       D0                  * Wurde linke Taste gedrückt?
        BPL.S       ENDE                * Ja, dann Ende.
        MOVEM.L     D1/D2,-(A7)         * D1 und D2 nicht zerstören.
        LEA         BUFFER(PC),A0       * Adresse für Ablage.
        MOVE        D1,D0
        MOVE        #!PRINT4D,D7        * Wert in D1 in ASCII wandeln.
        TRAP        #1
        MOVE.B      #',',(A0)+          * Trennzeichen.
        MOVE        D2,D0
        MOVEQ       #!PRINT4D,D7        * Wert in D2 in ASCII wandeln.
        TRAP        #1
        MOVEQ       #10-1,D0
SCHLEIF1:
        MOVE.B      #' ',(A0)+          * Mit Leerzeichen auffüllen.
        DBRA        D0,SCHLEIF1
        CLR.B       (A0)                * Neue Endekennung.
        MOVEQ       #$11,D0             * Schriftgröße.
        MOVEQ       #0,D1               * X-Position.
        MOVEQ       #0,D2               * Y-Position.
        LEA         BUFFER(PC),A0       * Adresse des Textes.
        MOVEQ       #!WRITE,D7          * Ausgabe des Textes.
        TRAP        #1
        MOVEM.L     (A7)+,D1/D2         * Register zurück.
        BRA.S       SCHLEIF0            * Wiederholen.
ENDE:
        RTS
BUFFER:
        DS.B        20
```

*
**

Unterprogrammnummer : **2**
Programmfunktion : **Maus zurücksetzen.**
Eingaberegister : Keine.
Ausgaberegister : Keine.

Durch dieses Programm werden die internen Mausregister für die X- und die Y-Bewegung auf Null gesetzt. Wenn die Maus jetzt bewegt wird, zählen die Register der Maus wieder von Null an. Der Befehl sollte am Anfang eines Programms benutzt werden, damit die Register der Maus definiert sind.

Auch dieses Programm kann auf ein Anwenderprogramm gelenkt werden. Dazu muß nur der JMP auf der Adresse $B58 relativ zum Variablenanfang umgebogen werden. Das Ziel des Sprungs kann jedes beliebige Programm sein, das eine Maus simuliert. Auch Eingabegeräte wie Joystick oder Trackball sind denkbar. Ohne Umlenkung des JMP wird immer zur HARDCOPY-Maus im Grundprogramm gesprungen.

Einfaches Beispiel:

Es werden die Mausregister gelöscht.

```
START:
        MOVEQ       #2,D0               * Kennung für Unterprogramm.
        MOVEQ       #!HARDCOPY,D7
        TRAP        #1
        RTS
```

*
**

Unterprogrammnummer : **3**
Programmfunktion : **Fadenkreuz setzen.**
Eingaberegister : D1.W = Gewünschte X-Position.
D2.W = Gewünschte Y-Position.
Ausgaberegister : Keine.

Mit diesem Befehl kann das Fadenkreuz gesetzt werden. Dazu wird in Register D1.W die gewünschte X-Position und in Register D2.W die gewünschte Y-Position übergeben. Allerdings sollte darauf geachtet werden, daß sich X im Bereich 0 bis 511 und Y im Bereich 0 bis 255 befindet, da der Rand nicht überwacht wird und sich sonst Bildstörungen ergeben können.

Komplexeres Beispiel:

Das Fadenkreuz wandert von rechts nach links über den Bildschirm und wird dann abgeschaltet.

```
START:
        MOVEQ       #127,D2             * Position des Querbalken.
        MOVE        #511,D1             * X-Position.
SCHLEIFE:
        MOVEQ       #3,D0               * Kennung für Programm.
        MOVEQ       #!HARDCOPY,D7
        TRAP        #1
SYNC:
        MOVEQ       #!SYNC,D7           * 20 ms warten.
        TRAP        #1
        BEQ.S       SYNC
        DBRA        D1,SCHLEIFE         * Nächste X-Position.
        MOVEQ       #4,D0               * Fadenkreuz abschalten.
        MOVEQ       #!HARDCOPY,D7
        TRAP        #1
        RTS
```

*
**

Unterprogrammnummer : 4
Programmfunktion : **Fadenkreuz ausschalten.**
Eingaberegister : Keine.
Ausgaberegister : Keine.

Das Fadenkreuz wird ausgeschaltet und ist nicht mehr auf dem Bildschirm zu sehen.

Komplexeres Beispiel:

Das Fadenkreuz wird eingeblendet und dann wieder abgeschaltet.

```
START:
        MOVE    #255,D1         * X = 255.
        MOVEQ   #127,D2         * Y = 127.
        MOVEQ   #3,D0
        MOVEQ   #!HARDCOPY,D7   * Fadenkreuz setzen.
        TRAP    #1
        MOVEQ   #30,D0
        MOVEQ   #!DELAY,D7      * 3 Sekunden warten.
        TRAP    #1
        MOVEQ   #4,D0
        MOVEQ   #!HARDCOPY,D7   * Fadenkreuz ausschalten.
        TRAP    #1
        RTS
```

*
**

Unterprogrammnummer : **5**
Programmfunktion : **Fadenkreuz mit Maus steuern.**
Eingaberegister : D1.W = Alte X-Position der Maus.
D2.W = Alte Y-Position der Maus.
Ausgaberegister : D0.L = Werte der Maustasten.
D1.W = Neue X-Position der Maus und des Kreuzes.
D2.W = Neue Y-Position der Maus und des Kreuzes.

Falls das Fadenkreuz mit der Maus über den Bildschirm bewegt werden soll, bietet sich dieses Unterprogramm an. Dabei müssen in D1.W und D2.W die alten Koordinaten des Fadenkreuzes (und so auch der Maus) übergeben werden. Das Programm addiert die Mausbewegung seit der letzten Abfrage und setzt das Fadenkreuz auf diese neue Position. Dabei wird allerdings keine Randüberwachung durchgeführt. Die neue Position wird in D1.W und D2.W zurückgegeben. In D0.L sind die Werte der Maustasten vorhanden.

Einfaches Beispiel:

Das Fadenkreuz kann mit der Maus bewegt werden. Das Programm endet, wenn die linke Maustaste gedrückt wird.

```
START:
        MOVEQ   #100,D1         * X-Anfang.
        MOVEQ   #100,D2         * Y-Anfang.
SCHLEIFE:
        MOVEQ   #5,D0           * Kennung für Programm.
        MOVEQ   #!HARDCOPY,D7
        TRAP    #1
        TST.B   D0
        BMI.S   SCHLEIFE        * Weiter, falls nicht linke Maustaste gedrückt.
        RTS
```

*
**

Unterprogrammnummer : **6**
Programmfunktion : **Wert vom AD-Port der Baugruppe lesen.**
Eingaberegister : Keine.
Ausgaberegister : D0.L = Wert des Ports.

Falls der Port der HARDCOPY-MAUS-Baugruppe genutzt werden soll, kann hiermit der Wert des Ports gelesen werden. Da der Port nur 8 Bit groß ist, steht in dem unteren Byte von D0.L der Wert, während die anderen Bits auf Null gesetzt sind.

Einfaches Beispiel:

Es wird der Wert des Ports gelesen.

```
START:
        MOVEQ       #6,D0               * Kennung für Programm.
        MOVEQ       #!HARDCOPY,D7
        TRAP        #1
        RTS                             * D0 enthält jetzt den Wert des Ports.
```

*
**

Unterprogrammnummer : **7**
Programmfunktion : **Copy bei CI an- oder ausschalten.**
Eingaberegister : D0.B = Bit 7 gibt an, ob an- oder ausgeschaltet wird.
Ausgaberegister : Keine.

Wie in Kapitel 3 sowie bei der Routine CI bereits erwähnt wurde, kann im Grundprogramm immer mit CTRL-@ eine Hardcopy des Bildschirminhalts auf den Drucker ausgegeben werden. Wenn aber ein Programm durch STARTEN, BIBLIOTHEK, BOOTEN oder EINZELSCHRITT aufgerufen wird, wird diese Funktion abgeschaltet, damit es nicht zu Schwierigkeiten kommt, weil das Programm möglicherweise die Funktion CTRL-@ für andere Zwecke nutzen könnte.

Mit dieser Funktion kann der Befehl CTRL-@ der Routine CI auch außerhalb des Grundprogramms angeschaltet bzw. wieder ausgeschaltet werden. Dabei wird zusätzlich zur Kennung des Programms in D0.B angegeben, ob die Funktion aktiviert oder deaktiviert werden soll. Ist das Bit 7 auf 1 gesetzt, ist der Befehl CTRL-@ verfügbar. Ansonsten ist er abgeschaltet.

Einfache Beispiele:

Hardcopy bei CI einschalten.

```
START:
        MOVEQ       #$87,D0             * Bit 7 = 1 ==> Hardcopy an.
        MOVEQ       #!HARDCOPY,D7
        TRAP        #1
        RTS
```

Hardcopy bei CI ausschalten.

```
START:
        MOVEQ       #7,D0               * Bit 7 = 0 ==> Hardcopy aus.
        MOVEQ       #!HARDCOPY,D7
        TRAP        #1
        RTS
```

*
**

Unterprogrammnummer : **8**
Programmfunktion : **Bildschirmdaten im Speicher abgelegen.**
Eingaberegister : D0.W = Bits 4 - 7 haben Spezialfunktionen.
A0.L = Gerade Zieladresse für Ablage im Speicher.
Ausgaberegister : Keine.

Dieses Unterprogramm ist die Grundlage für die Funktionen 10, 11, 12 und 13. Es wird immer am Anfang dieser Befehle aufgerufen. Es wird dann der Bildschirmspeicher der Karten GDP oder GDPHS mit Hilfe der Hardcopy-Funktion ausgelesen und im Speicher abgelegt. Dabei ist das Bild im Speicher genau so organisiert wie auf dem Bildschirm, d. h. der Punkt links oben (0,255) ist das erste Bit, der zweite Punkt (1,255) das zweite Bit usw. Der letzte Punkt (511,0) ist das letzte Bit im Speicher. Deshalb wird eine Länge von 16 Kbyte für ein Bild benötigt. Ein gesetztes Bit bedeutet, daß der Punkt auf dem Bildschirm sichtbar ist.

An das Programm müssen in den oberen 4 Bits von D0.B noch ein paar Angaben übergeben werden.

Bit 7 0 = Das Bild wird im Speicher abgelegt, wobei die alten Werte des Speichers überschrieben werden.
1 = Die Bildschirmdaten werden mit den Daten im Speicher mit einem ODER (nicht XOR, sondern OR) verknüpft. Dadurch können Bilder übereinandergelegt werden.

Bit 6 0 = Das Bild wird unverändert übernommen.
1 = Das Bild wird invertiert im Bildspeicher abgelegt.

Bit 5 0 = Es findet keine Veränderung statt.
1 = Das Bild wird spiegelverkehrt in Y-Richtung im Bildspeicher abgelegt.

Bit 4 0 = Es findet keine Veränderung statt.
1 = Das Bild wird spiegelverkehrt in X-Richtung im Bildspeicher abgelegt.

Außerdem muß natürlich noch angegeben werden, wo die Bilddaten abgelegt werden sollen. Diese Angabe muß A0.L enthalten, wobei A0.L auf eine gerade Adresse zeigen muß, da die Daten wortweise übertragen werden und es sonst zu einem ADRESS-ERROR kommt (Beim 68020 natürlich nicht).

In dieser Funktion ist noch eine Besonderheit enthalten. Wenn sie aufgerufen wird, ohne daß eine Extrafunktion gewünscht ist (Bits 4-7 auf Null), wird die gleiche Funktion des GRAFIK-Pakets aufgerufen, die wesentlich schneller arbeitet. Es ist dann aber kein Fadenkreuz zu sehen; die Funktion wird trotzdem ausgeführt. Dazu muß allerdings die GDPHS vorhanden sein.

Einfaches Beispiel:

Das momentane Bild wird im Speicher abgelegt.

```
START:
        MOVEQ       #8,D0               * Kennung für Programm.
        LEA         BUFFER(PC),A0       * Ziel für Bilddaten.
        MOVEQ       #!HARDCOPY,D7
        TRAP        #1
        RTS
BUFFER:                                 * Speicher für Bilddaten.
        DS.B        1024*16
```

Komplexeres Beispiel:

Ein ausgefülltes Quadrat wird gezeichnet und abgespeichert. Danach wird das Quadrat noch einmal spiegelverkehrt abgespeichert. Der Bildschirm wird gelöscht und das Bild aus beiden Quadraten invers auf dem Bildschirm dargestellt.

```
START:
        MOVEQ       #4,D0               * Quadrat ausgefüllt.
        MOVEQ       #100,D1             * X-Position.
        MOVEQ       #100,D2             * Y-Position.
        MOVEQ       #100,D3             * Kantenlänge des Quadrats.
        MOVEQ       #!GRAFIK,D7
        TRAP        #1
        LEA         BUFFER(PC),A0       * Ziel für Bild.
        MOVEQ       #8,D0               * Ohne Extrafunktion.
        MOVEQ       #!HARDCOPY,D7       * Abspeichern.
        TRAP        #1
        MOVEQ       #%10101000,D0       * Jetzt dazuspeichern und spiegelverkehrt
        MOVEQ       #!HARDCOPY,D7       * im Bildspeicher ablegen.
        TRAP        #1
        MOVEQ       #20,D0
        MOVEQ       #!DELAY,D7
        TRAP        #1
        MOVEQ       #!CLR,D7            * Bildschirm löschen.
        TRAP        #1
        MOVEQ       #%01001001,D0       * Invers auf den Bildschirm bringen.
        MOVEQ       #!HARDCOPY,D7
        TRAP        #1
        RTS
BUFFER:                                 * Speicher für Bilddaten.
        DS.B        1024*16
```

*
**

Unterprogrammnummer : **9**
Programmfunktion : **Ein Bild aus dem Speicher auf den Bildschirm bringen.**
Eingaberegister : D0.W = Bits 4 - 7 haben Spezialfunktionen.
A0.L = Gerade Adresse der Bildschirmdaten.
Ausgaberegister : Keine.

Dieses ist der umgekehrte Vorgang zu Funktion 8. Die Bilddaten, die im Speicher stehen, werden wieder auf den Bildschirm gebracht. Dabei muß A0.L wieder auf den Anfang des Speicherbereichs zeigen, der die Daten enthält. D0.B hat bis auf eine Ausnahme die gleiche Funktion wie bei Funktion 8. Die Bits 4, 5 und 6 haben die gleichen Funktionen, nur Bit 7 weicht etwas ab. Ist dieses Bit nicht gesetzt, werden die Daten normal auf den Bildschirm gebracht (so, wie der GDP eingestellt ist). Ist das Bit auf 1 gesetzt, wird der XOR-Modus eingeschaltet. Die Bildschirmfarbe wechselt dann dort, wo ein Punkt gesetzt wird (siehe Handbuch GDPHS).

Einfaches Beispiel:

Der Bildschirmspeicher wird mit einem Muster gefüllt.

```
START:
        LEA         BUFFER(PC),A0       * Adresse der Daten.
        MOVEQ       #9,D0               * Kennung für Programm.
        MOVEQ       #!HARDCOPY,D7
        TRAP        #1
        RTS
BUFFER:
        DF.B        1024*16,$55         * Hier liegt ein Muster.
```

Komplexeres Beispiel:

Ein Kreis wird gezeichnet. Das Bild wird abgespeichert. Dann wird es gespiegelt abgespeichert und mit dem XOR-Modus geladen. Anschließend wird es noch einmal gespiegelt geladen.

```
START:
        MOVEQ       #8,D0               * Kreis wird ausgefüllt.
        MOVEQ       #100,D1             * X-Position.
        MOVEQ       #100,D2             * Y-Position.
        MOVEQ       #100,D3             * Radius des Kreises.
        MOVEQ       #!GRAFIK,D7
        TRAP        #1
        MOVEQ       #20,D0              * 2 Sekunden warten.
        MOVEQ       #!DELAY,D7
        TRAP        #1
        MOVEQ       #8,D0               * Ohne Extrafunktion.
        MOVEQ       #!HARDCOPY,D7       * Abspeichern.
        TRAP        #1
        MOVEQ       #%10101000,D0       * Jetzt dazuspeichern und spiegelverkehrt.
        MOVEQ       #!HARDCOPY,D7
        TRAP        #1
        MOVEQ       #%10001001,D0       * Mit XOR auf den Bildschirm bringen.
        MOVEQ       #!HARDCOPY,D7
        TRAP        #1
        MOVEQ       #%10111001,D0       * Noch einmal mit XOR auf den Bildschirm.
        MOVEQ       #!HARDCOPY,D7
        TRAP        #1
        RTS
BUFFER:                                 * Speicher für Bilddaten.
        DS.B        1024*16
```

*
**

Unterprogrammnummer : **10**
Programmfunktion : **Hardcopy auf 9-Nadel-Drucker.**
Eingaberegister : D0.W = Bits 4 - 7 haben Spezialfunktionen.
A0.L = Gerade Adresse für Hilfsspeicher.
A1.L = Adresse der Druckerbefehle.
Ausgaberegister : Keine.

Für die Ausgabe des Bildschirminhaltes auf dem Drucker gibt es dieses Programm.

Zuerst wird wie bei Funktion 8 eine HARDCOPY in den Speicher durchgeführt. Deshalb müssen die Register D0.B und A0.L auch genau so wie dort beschrieben mit Werten geladen werden. Wenn danach das Bild im Speicher vorhanden ist, werden die Daten aufbereitet an den Drucker geschickt. Dazu wird am Anfang jeder Zeile eine Folge von Befehlen an den Drucker gegeben. Diese Befehle müssen z. B. den Zeilenabstand einstellen und den Befehl für die Einzelnadelansteuerung des Druckers beinhalten. A1.L muß auf diese Tabelle der Druckerbefehle zeigen. Hinter dem letzten Druckerbefehl muß ein Byte folgen, bei dem Bit 7 gesetzt ist. Das funktioniert, da kein Druckerbefehl Bit 7 benutzt. Die Bits 0 bis 6 haben Spezialfunktionen. Die Bits 0 bis 2 bestimmen die Vergrößerung in Y-Richtung und die Bits 4 bis 6 die Vergrößerung in X-Richtung. Das Endebyte sieht also folgendermaßen aus: %1xxx0yyy, wobei jeweils %111 für xxx und %111 für yyy bedeutet, daß keine Vergrößerung durchgeführt wird. %000 bedeutet, daß jeder Punkt zweimal gedruckt wird. Bei %001 erfolgt der Druck eines jeden Punkts dreimal, bei %010 viermal usw.

Wenn diese Befehle am Anfang einer Zeile an den Drucker geschickt worden sind, werden die Bildpunkte übertragen. Dabei werden immer 8 Punkte in einem Byte gleichzeitig übertragen. Diese Bildpunkte stehen untereinander. Es werden also 512 Bytes * Vergrößerung in einer Zeile übertragen. Diese Anzahl ist wichtig, da der Grafik-Befehl für die Drucker immer die Anzahl der übertragenen Bytes enthalten muß. Am Ende einer Zeile wird automatisch ein CR LF (Code $0D und $0A) übertragen.

Beispiel für eine Tabelle der Druckerbefehle (wie im Grundprogramm):

```
TABELLE:
        DC.B    $1B,'A',8       * Zeilenabstand einstellen.
        DC.B    $1B,'L',0,2     * Grafik Befehl (0,2 ist Anzahl der Punkte).
        DC.B    $FF             * Endekennung ist unbedingt notwendig.
                                * Keine Vergrößerung.
```

Einfaches Beispiel:

```
START:
        MOVEQ   #10,D0          * Kennung für Programm.
        LEA     BUFFER(PC),A0   * Freiraum für Bild.
        LEA     TABELLE(PC),A1  * Druckerbefehle.
        MOVEQ   #!HARDCOPY,D7
        TRAP    #1
        RTS
BUFFER:
        DS.B    1024*16
```

Beispiel für Tabelle mit Vergrößerung:

```
TABELLE:
        DC.B    $1B,'<'         * Unidirektionaler Druck für eine Zeile.
        DC.B    $1B,'A',8       * Zeilenabstand.
        DC.B    $1B,'L',0,2     * Grafik Befehl (1024 Spalten).
        DC.B    %10010001       * Endekennung (Vergrößerung 1 für X und Y).
```

*
**

Unterprogrammnummer : **11**
Programmfunktion : **Hardcopy auf 24-Nadel-Drucker.**
Eingaberegister : D0.W = Bits 4 - 7 haben Spezialfunktionen.
A0.L = Gerade Adresse für Hilfsspeicher.
A1.L = Adresse der Druckerbefehle.
Ausgaberegister : Keine.

Auch dieser Befehl ist für die Druckerausgabe gedacht. Dabei müssen aber andere Grafik-Befehle für den Drucker verwendet werden, da immer 24 Bit (3 Byte) in einer Zeile ausgegeben werden. Dadurch können auch höhere Auflösungen für 24-Nadel-Drucker verwendet werden.

Da der Befehl ansonsten identisch mit Funktion 10 ist, soll hier nur als Beispiel eine Tabelle der Druckerbefehle gegeben werden.

```
TABELLE:
        DC.B    $1B,'A',8       * Zeilenabstand.
        DC.B    $1B,'*',40,0,2  * Befehl für 24-Nadel-Grafik.
        DC.B    $FF             * Endekennung (keine Vergrößerung).
```

Beispiel für Tabelle mit Vergrößerung (wie im Grundprogramm):

```
TABELLE:
        DC.B    $1B,'<'         * Unidirektionaler Druck für eine Zeile.
        DC.B    $1B,'A',8       * Zeilenabstand.
        DC.B    $1B,'*',40,0,10 * Grafik Befehl (1024 Spalten) für 24 Nadeln.
        DC.B    %10110011       * Endekennung (Vergrößerung 4 für X und Y).
```

*
**

Unterprogrammnummer : **12**
Programmfunktion : **Standardhardcopy auf 9-Nadel-Drucker.**
Eingaberegister : D0.W = Bits 4 - 7 haben Spezialfunktionen.
A0.L = Gerade Adresse für Hilfsspeicher.
Ausgaberegister : Keine.
Zerstörte Register : A1

Dieser Befehl entspricht dem Programm 10. Es werden aber keine Druckerbefehle ausgegeben, da im Grundprogramm vorhandene verwendet werden. Die zweite aufgeführte Tabelle für den 9-Nadel-Betrieb ist identisch mit der im Grundprogramm. Die Befehle sind für Epson-kompatible Drucker geeignet.

*
**

Unterprogrammnummer : **13**
Programmfunktion : **Standardhardcopy auf 24-Nadel-Drucker.**
Eingaberegister : D0.W = Bits 4 - 7 haben Spezialfunktionen.
A0.L = Gerade Adresse für Hilfsspeicher.
Ausgaberegister : Keine.
Zerstörte Register : A1

Dieser Befehl entspricht dem Programm 11, wobei allerdings keine Druckerbefehle angegeben werden, da im Grundprogramm vorhandene verwendet werden. Die zweite oben aufgeführte Tabelle für den 24-Nadel-Betrieb ist identisch mit der im Grundprogramm. Die Befehle sind für Epson-kompatible Drucker geeignet.

*
**

Unterprogrammnummer : **14**
Programmfunktion : **Speicherinhalt auf 9-Nadel-Drucker.**
Eingaberegister : A0.L = Gerade Adresse für Bilddaten.
A1.L = Adresse der Druckerbefehle.
Ausgaberegister : Keine.

Dieser Befehl entspricht der Funktion 10, mit dem Unterschied, daß die Daten nicht vom Bildschirm geholt werden, sondern schon im Speicher vorliegen müssen. Deshalb haben die Bits 4 - 7 auch keine Spezialfunktionen. Die Funktion des Befehls kann bei Funktion 10 nachgelesen werden.

Einfaches Beispiel:

Daten aus dem Speicher ausdrucken.

```
START:
        LEA     BUFFER(PC),A0   * Adresse der Daten im Speicher.
        LEA     DRUCK(PC),A1    * Adresse der Druckbefehle.
        MOVEQ   #14,D0          * Kennung für Programm.
        MOVEQ   #!HARDCOPY,D7
        TRAP    #1
        RTS
BUFFER:
        DS.B    1024*16         * Bildschirmdaten.
DRUCK:
        DC.B    ....            * Druckerbefehle.
        DC.B    $FF             * Endekennung der Tabelle.
```

*
**

Unterprogrammnummer : **15**
Programmfunktion : **Speicherinhalt auf 24-Nadel-Drucker.**
Eingaberegister : A0.L = Gerade Adresse für Bilddaten.
A1.L = Adresse der Druckerbefehle.
Ausgaberegister : Keine.

Dieser Befehl entspricht der Funktion 11, mit dem Unterschied, daß die Daten nicht vom Bildschirm geholt werden, sondern schon im Speicher vorliegen müssen. Deshalb haben die Bits 4 - 7 auch keine Spezialfunktionen. Die Funktion des Befehls kann bei Funktion 11 nachgelesen werden.

Einfaches Beispiel:

Daten aus dem Speicher ausdrucken.

```
START:
        LEA     BUFFER(PC),A0       * Adresse der Daten im Speicher.
        LEA     DRUCK(PC),A1        * Adresse der Druckbefehle.
        MOVEQ   #15,D0              * Kennung für Programm.
        MOVEQ   #!HARDCOPY,D7
        TRAP    #1
        RTS
BUFFER:
        DS.B    1024*16             * Bildschirmdaten.
DRUCK:
        DC.B    ....                * Druckerbefehle.
        DC.B    $FF                 * Endekennung der Tabelle.
```

TRAP-Nummer	: 126
Befehlsname	: **GRAFIK**
Befehlsgruppe	: Grafik.
Kurzbeschreibung	: Großes Grafik-Paket für GDP und COL.
Eingaberegister	: D0.W = Auswahl des Programms. Zusätzlich noch verschiedene Register, die vom jeweiligen Unterprogramm abhängen.
Ausgaberegister	: Ausgaberegister sind vom Unterprogramm abhängig.
Zerstörte Register	: D7/A6
Ab Version	: 6.0
Änderungen zu 4.3	: ——
Änderungen zu 6.1	: Viele Funktionen geben jetzt ein CARRY zurück. Neue Funktionen sind vorhanden.
Siehe auch	: MOVETO (8), DRAWTO (9), CLR (16), CLPG (17), WAIT (18), CMD (26), NEWPAGE (27), SETFLIP (34), SETPEN (37), ERAPEN (38), CMDPRINT (40), AUTOFLIP (60), SETXOR (77), GETXOR (78), SETCOLOR (79), GETCOLOR (80), GETXY (103), HARDCOPY (125), GDPVERS (127)

Auch dieser Aufruf beinhaltet wie HARDCOPY eine Reihe von Funktionen. Es sind alles Programme, die für Grafik- und Textausgabe bestimmt sind. Die Besonderheit bei diesen Programmen ist, daß die Ausgabe für die GDP-Karte und die COL256-Karte geschrieben sind. Durch einen der unten beschriebenen Befehle kann ausgewählt werden, ob die Ausgabe über die GDP- oder die COL256-Karte erfolgen soll. Normalerweise wird die Ausgabe auf die GDP-Karte gelenkt.

Wichtig ist für die Ausgabe auf der COL-Karte, daß der Ausgang benutzt wird, der eine Auflösung von 512*256 Punkten ermöglicht, da die Routinen dafür ausgelegt sind. Es sind deshalb "nur" 16 Farben darstellbar. Allerdings ist die Auflösung genau so wie bei der GDP-Karte, weshalb die Funktionen alle nahezu kompatibel sind.

Auch hier muß in D0.W entschieden werden, welches Unterprogramm aufgerufen wird. Es gibt 32 Unterprogramme in diesem Programmpaket. Die Register D7 und A6 werden auf jeden Fall zerstört.

Obwohl eigentlich in Y-Richtung nur 256 Punkte vorhanden sind, wird bei jedem Befehl eine Bildschirmgröße von 512*512 Punkten angenommen, wodurch ein rechteckiges Bildschirmformat simuliert wird. Intern wird die Y-Koordinate durch 2 geteilt. Bei allen Befehlen wird eine Randüberwachung durchgeführt, wobei die Werte aber im Bereich vom -2048 bis +2047 bleiben sollten.

Es ist darauf zu achten, daß einige Funktionen nicht mit der GDP-Karte, sondern nur mit der GDPHS-Karte funktionieren. Das ist aber jeweils gesondert erwähnt.

Außerdem sollten keine anderen Grafik-Routinen des Grundprogramms verwendet werden, während mit dem GRAFIK-Pakte gearbeitet wird, da diese eventuell eingestellte Werte verändern können (z. B. NEWPAGE, SETPEN, ERAPEN, ...).

Alle Zeichen-Funktionen, bei denen nichts weiter bemerkt ist, werden in der aktuellen Schreibfarbe und mit der aktuellen Verknüpfung durchgeführt. Dabei bedeutet aktuelle Schreibfarbe bei der GDP, daß die Werte genommen werden, die in den Registern des GDP und im Register für den XOR-Modus stehen. Es erfolgt also z. B. auch eine Änderung mit den Befehlen SETPEN, ERAPEN oder SETXOR. Bei der COL hingegen werden die Werte benutzt, die als letztes im GRAFIK-Paket eingestellt wurden, da sie jedesmal verwendet werden.

*
**

Unterprogrammnummer	: **0**
Programmfunktion	: **Einen Punkt setzen.**
Eingaberegister	: D1.W = X-Koordinate. D2.W = Y-Koordinate.
Ausgaberegister	: Keine.

An der durch D1.W und D2.W bezeichneten Position wird ein Punkt in der aktuellen Schreibfarbe gesetzt. Wenn der Punkt sichtbar sein soll, müssen D1.W und D2.W im Bereich von 0 bis 511 liegen.

Einfaches Beispiel:

Es wird ein Punkt an der Stelle 100,100 gesetzt.

```
START:
        MOVEQ     #100,D1          * X = 100.
        MOVEQ     #100,D2          * Y = 100.
        MOVEQ     #0,D0            * Kennung für Programm.
        MOVEQ     #!GRAFIK,D7
        TRAP      #1
        RTS
```

*
**

Unterprogrammnummer : **1**
Programmfunktion : **Linie zeichnen.**
Eingaberegister : D1.W = Anfangspunkt X.
D2.W = Anfangspunkt Y.
D3.W = Endpunkt X.
D4.W = Endpunkt Y.
Ausgaberegister : Keine.

Hiermit kann eine beliebige Linie gezeichnet werden. Dazu werden in D1.W und D2.W die Koordinaten des Anfangspunktes und in D3.W und D4.W die Koordinaten des Endpunktes angegeben.

Einfaches Beispiel:

Es wird eine Bildschirmdiagonale gezeichnet.

```
START:
        MOVEQ     #0,D1            * X-Anfang.
        MOVEQ     #0,D2            * Y-Anfang.
        MOVE      #511,D3          * X-Ende.
        MOVE      #511,D4          * Y-Ende.
        MOVEQ     #1,D0            * Kennung für Programm.
        MOVEQ     #!GRAFIK,D7
        TRAP      #1
        RTS
```

*
**

Unterprogrammnummer : **2**
Programmfunktion : **Linienfolge zeichnen.**
Eingaberegister : A0.L = Adresse der Punktetabelle.
Ausgaberegister : A0.L = Zeigt auf erstes Byte hinter der Tabelle.

Mit diesem Programm können sehr schnell nacheinander Linien gezeichnet werden. Dabei ist der Endpunkt der ersten Linie der Anfangspunkt der nächsten Linie usw. Die Koordinaten dafür werden in einer Tabelle übergeben. A0.L muß auf diese Tabelle zeigen. Nach dem Aufruf zeigt A0.L genau hinter die Tabelle, wodurch leicht mehrere Tabellen hintereinander aufgerufen werden können.

Die Tabelle muß als ersten Wert die Anzahl der Punkte als Wort enthalten. Dann folgen die einzelnen Punkte jeweils mit der Angabe der X- und der Y-Koordinate. Auch die Koordinaten haben Wortlänge.

Beispiel:

```
TABELLE:
        DC.W      3                * Anzahl der Punkte.
        DC.W      0,0              * Erster Punkt.
        DC.W      200,200          * Zweiter Punkt.
        DC.W      400,100          * Dritter Punkt.
```

Einfaches Beispiel:

Es werden zwei Linien gezogen, wenn die Tabelle aus dem Beispiel genommen wird.

```
START:
        LEA         TABELLE(PC),A0      * Adresse der Daten.
        MOVEQ       #2,D0               * Kennung des Programms.
        MOVEQ       #!GRAFIK,D7
        TRAP        #1
        RTS
```

*
**

Unterprogrammnummer : **3**
Programmfunktion : **Quadrat zeichnen.**
Eingaberegister : D1.W = X-Koordinate linke untere Ecke.
D2.W = Y-Koordinate linke untere Ecke.
D3.W = Positive Kantenlänge.
Ausgaberegister : Keine.

Dieses Programm zeichnet ein Quadrat. Die Position wird durch die Angabe der linken unteren Ecke bestimmt. D1.W muß die X-Koordinate und D2.W die Y-Koordinate enthalten. In D3.W muß die Kantenlänge angegeben werden. Dieser Wert sollte positiv sein, da es sonst zu Fehlern kommen kann.

Einfaches Beispiel:

Ein Quadrat wird mit der linken unteren Ecke auf die Position 50,50 gesetzt.

```
START:
        MOVEQ       #50,D1              * X-Position.
        MOVEQ       #50,D2              * Y-Position.
        MOVEQ       #120,D3             * Kantenlänge.
        MOVEQ       #3,D0               * Kennung für Programm.
        MOVEQ       #!GRAFIK,D7
        TRAP        #1
        RTS
```

*
**

Unterprogrammnummer : **4**
Programmfunktion : **Ausgefülltes Quadrat zeichnen.**
Eingaberegister : D1.W = X-Koordinate linke untere Ecke.
D2.W = Y-Koordinate linke untere Ecke.
D3.W = Positive Kantenlänge.
Ausgaberegister : Keine.

Das Programm zeichnet ein ausgefülltes Quadrat. Die Position wird durch die Angabe der linken unteren Ecke bestimmt. D1.W muß die X-Koordinate und D2.W die Y-Koordinate enthalten. In D3.W wird die Kantenlänge angegeben. Dieser Wert sollte positiv sein, da es sonst zu Fehlern kommen kann.

Einfaches Beispiel:

Ein ausgefülltes Quadrat wird mit der linken unteren Ecke auf die Position 50,50 gesetzt. Die Kantenlänge beträgt 120 Punkte.

```
START:
        MOVEQ       #50,D1              * X-Position.
        MOVEQ       #50,D2              * Y-Position.
        MOVEQ       #120,D3             * Kantenlänge.
        MOVEQ       #4,D0               * Kennung für Programm.
        MOVEQ       #!GRAFIK,D7
        TRAP        #1
        RTS
```

*
**

Unterprogrammnummer : **5**
Programmfunktion : **Rechteck zeichnen.**
Eingaberegister : D1.W = X-Koordinate linke untere Ecke.
D2.W = Y-Koordinate linke untere Ecke.
D3.W = Positive Kantenlänge in X-Richtung.
D4.W = Positive Kantenlänge in Y-Richtung.
Ausgaberegister : Keine.

Das Programm ähnelt sehr der Funktion 4. Allerdings wird kein Quadrat, sondern ein Rechteck gezeichnet. Dazu müssen wieder in D1.W und D2.W die Koordinaten für die linke untere Ecke angegeben werden. In D3.W und D4.W müssen die Angaben für die Kantenlängen vorhanden sein. D3.W gibt an, wie lang das Rechteck in X-Richtung sein soll. Die Kantenlängen müssen positiv sein.

Einfaches Beispiel:

Es wird an die Stelle 50,100 ein Rechteck mit den Kantenlängen 100,300 gesetzt.

```
START:
        MOVEQ     #50,D1        * X-Position.
        MOVEQ     #100,D2       * Y-Position.
        MOVEQ     #100,D3       * X-Kantenlänge.
        MOVE      #300,D4       * Y-Kantenlänge.
        MOVEQ     #5,D0         * Kennung für Programm.
        MOVEQ     #!GRAFIK,D7
        TRAP      #1
        RTS
```

*
**

Unterprogrammnummer : **6**
Programmfunktion : **Ausgefülltes Rechteck zeichnen.**
Eingaberegister : D1.W = X-Koordinate linke untere Ecke.
D2.W = Y-Koordinate linke untere Ecke.
D3.W = Positive Kantenlänge in X-Richtung.
D4.W = Positive Kantenlänge in Y-Richtung.
Ausgaberegister : Keine.

Vom Aufruf her ist dieses Programm identisch mit der Funktion 5. Allerdings wird kein umrandetes Rechteck gezeichnet, sondern ein ausgefülltes. Auch hier ist wieder darauf zu achten, daß D3.W und D4.W positiv sind.

Einfaches Beispiel:

Ein Rechteck füllt die untere Bildschirmhälfte aus.

```
START:
        MOVEQ     #0,D1         * X-Koordinate.
        MOVEQ     #0,D2         * Y-Koordinate.
        MOVE      #511,D3       * Kantenlänge X.
        MOVE      #255,D4       * Kantenlänge Y.
        MOVEQ     #6,D0         * Kennung für Programm.
        MOVEQ     #!GRAFIK,D7
        TRAP      #1
        RTS
```

*
**

Unterprogrammnummer : **7**
Programmfunktion : **Kreis zeichnen.**
Eingaberegister : D1.W = X-Koordinate Mittelpunkt.
D2.W = Y-Koordinate Mittelpunkt.
D3.W = Positiver Radius.
Ausgaberegister : Keine.

Für das schnelle Zeichnen von Kreisen gibt es dieses Programm. Dazu müssen in D1.W und D2.W die Koordinaten des Mittelpunktes übergeben werden. In D3.W ist zusätzlich die positive Angabe des Radius erforderlich.

Einfaches Beispiel:

Ein Kreis mit dem Radius 100 wird in der Mitte des Bildschirms gezeichnet.

```
START:
        MOVE    #255,D1         * X-Koordinate.
        MOVE    #255,D2         * Y-Koordinate.
        MOVEQ   #100,D3         * Radius.
        MOVEQ   #7,D0           * Kennung des Programms.
        MOVEQ   #!GRAFIK,D7
        TRAP    #1
        RTS
```

*
**

Unterprogrammnummer : **8**
Programmfunktion : **Ausgefüllten Kreis zeichnen.**
Eingaberegister : D1.W = X-Koordinate Mittelpunkt.
D2.W = Y-Koordinate Mittelpunkt.
D3.W = Positiver Radius.
Ausgaberegister : Keine.

Vom Aufruf her ist dieses Programm identisch mit Funktion 7, allerdings wird der Kreis nicht nur gezeichnet, sondern ausgefüllt. D3.W sollte auch hier unbedingt positiv sein.

Einfaches Beispiel:

Eine Kreisfläche wird im unteren linken Bildschirmbereich gezeichnet.

```
START:
        MOVEQ   #127,D1         * X-Koordinate.
        MOVEQ   #127,D2         * Y-Koordinate.
        MOVEQ   #63,D3          * Radius.
        MOVEQ   #8,D0           * Kennung des Programms.
        MOVEQ   #!GRAFIK,D7
        TRAP    #1
        RTS
```

*
**

Unterprogrammnummer : **9**
Programmfunktion : **Ellipse zeichnen.**
Eingaberegister : D1.W = X-Koordinate Mittelpunkt.
D2.W = Y-Koordinate Mittelpunkt.
D3.W = Positiver Radius in X-Richtung.
D4.W = Positiver Radius in Y-Richtung.
Ausgaberegister : Keine.

Hiermit wird eine beliebige Ellipse gezeichnet, wobei allerdings die beiden Hauptachsen parallel zu den Koordinatenachsen liegen. Als Eingabe muß wie beim Kreis der Mittelpunkt in den Registern D1.W und D2.W übergeben werden. Der Radius in Richtung der X-Achse muß in D3.W und der Radius in Richtung der Y-Achse in D4.W übergeben werden. D3.W und D4.W sollten positiv sein.

Einfaches Beispiel:

Eine Ellipse wird in der Bildschirmmitte gezeichnet.

```
START:
        MOVE    #255,D1         * X-Mittelpunkt.
        MOVE    #255,D2         * Y-Mittelpunkt.
        MOVE    #50,D3          * Radius in X-Richtung.
        MOVE    #200,D4         * Radius in Y-Richtung.
        MOVEQ   #9,D0           * Kennung.
        MOVEQ   #!GRAFIK,D7     * Aufruf.
        TRAP    #1
        RTS
```

*
**

Unterprogrammnummer : **10**
Programmfunktion : **Ausgefüllte Ellipse zeichnen.**
Eingaberegister : D1.W = X-Koordinate Mittelpunkt.
D2.W = Y-Koordinate Mittelpunkt.
D3.W = Positiver Radius in X-Richtung.
D4.W = Positiver Radius in Y-Richtung.
Ausgaberegister : Keine.

Bis auf die Tatsache, daß hier die Ellipse ausgefüllt wird, ist dieser Befehl identisch mit der Funktion 11. Deshalb sind auch keine weiteren Erklärungen nötig.

Komplexeres Beispiel:

Eine Kreisscheibe dreht sich in der Bildschirmmitte.

```
START:
        MOVEQ       #0,D5           * Leseseite am Anfang.
        MOVEQ       #0,D6           * Anfangswinkel.
SCHLEIFE:
        MOVE        D5,D1           * Leseseite.
        EORI        #1,D5
        MOVE        D5,D2           * Schreibseite.
        MOVEQ       #13,D0          * Kennung.
        MOVEQ       #!GRAFIK,D7
        TRAP        #1
        MOVEQ       #0,D1           * Schreibfarbe.
        MOVEQ       #0,D2           * Verknüpfung.
        MOVEQ       #15,D0          * Funktion.
        MOVEQ       #!GRAFIK,D7
        TRAP        #1
        MOVE        #255,D1         * Mittelpunkt X.
        MOVE        #255,D2         * Mittelpunkt Y.
        MOVEQ       #64,D3          * Radius X.
        MOVEQ       #64,D4          * Radius Y.
        MOVEQ       #10,D0          * Kennung.
        MOVEQ       #!GRAFIK,D7     * Alte Ellipse löschen.
        TRAP        #1
        MOVEQ       #5,D1           * Schreibfarbe.
        MOVEQ       #0,D2           * Verknüpfung.
        MOVEQ       #15,D0          * Kennung.
        MOVEQ       #!GRAFIK,D7
        TRAP        #1
        MOVE        D6,D0           * Winkel.
        MOVEQ       #!COS,D7
        TRAP        #1
        ASR         #2,D0           * Maximal 64.
        BPL.S       SPRUNG0         * Positiv, dann OK.
        NEG         D0              * Sonst Vorzeichen umdrehen.
SPRUNG0:
        MOVE        D0,D3           * Nach D3 als Radius X.
        MOVE        #255,D1         * Mittelpunkt X.
        MOVE        #255,D2         * Mittelpunkt Y.
        MOVEQ       #64,D4          * Radius Y.
        MOVEQ       #10,D0          * Kennung.
        MOVEQ       #!GRAFIK,D7     * Ellipse zeichnen.
        TRAP        #1
        ADDQ        #2,D6           * Neuer Winkel.
        MOVEQ       #!CSTS,D7
        TRAP        #1
        BEQ.S       SCHLEIFE        * Wiederholen, bis Taste gedrückt.
        RTS
```

*
**

Unterprogrammnummer : **11**
Programmfunktion : **Fläche füllen.**
Eingaberegister : D1.W = X-Koordinate des Anfangspunktes.
D2.W = Y-Koordinate des Anfangspunktes.
Ausgaberegister : Carry = Gesetzt, wenn Fehler aufgetreten ist.

Für das Füllen geschlossener Flächen gibt es diesen Befehl. Er funktioniert mit der COL256 und der neuen GDPHS, da Punkte ausgelesen werden müssen.

Es darf nicht mit der Farbe schwarz und gleichzeitig aktiviertem XOR-Modus gearbeitet werden.

Übergeben wird der erste Punkt. Die Koordinaten müssen in D1.W und D2.W übergeben werden. Wird ein Punkt gewählt, der außerhalb des Bildschirmbereichs liegt oder der die gleiche Farbe hat wie die aktuelle Malfarbe, wird der Befehl nicht ausgeführt und das CARRY-Flag gesetzt. Der Befehl wird außerdem nur bei der GDPHS ausgeführt. Bei der GDP wird das CARRY-Flag gesetzt.

Komplexeres Beispiel:

Nach dem Löschen des Bildschirms werden viele kleine Kreise gezeichnet. Dann wird vom Mittelpunkt aus die verfügbare Fläche gefüllt. Das kann ein kleiner Kreisausschnitt sein oder eine große Fläche zwischen den Kreisen. Durch Tastendruck kann das Programm abgebrochen werden.

```
START:
        MOVEQ     #0,D1          * Farbe 0.
        MOVEQ     #0,D2          * Auf Setzen.
        MOVEQ     #15,D0         * Kennung.
        MOVEQ     #!GRAFIK,D7
        TRAP      #1
        MOVEQ     #12,D0
        MOVEQ     #!GRAFIK,D7    * Aktuelle Schreibseite löschen.
        TRAP      #1
        MOVEQ     #7,D1          * Farbe 7.
        MOVEQ     #0,D2          * Auf Setzen.
        MOVEQ     #15,D0
        MOVEQ     #!GRAFIK,D7
        TRAP      #1
        MOVEQ     #100-1,D6      * 100 Durchgänge.
SCHLEIFE:
        MOVE      #512,D0
        MOVEQ     #!RND,D7       * Zufallszahl im Bereich 0 .. 511.
        TRAP      #1
        MOVE      D0,D1          * Nach D1.
        MOVE      #512,D0
        MOVEQ     #!RND,D7       * Zufallszahl im Bereich 0 .. 511.
        TRAP      #1
        MOVE      D0,D2          * Nach D2.
        MOVEQ     #20,D3         * Radius.
        MOVEQ     #7,D0
        MOVEQ     #!GRAFIK,D7    * Kreis zeichnen.
        TRAP      #1
        DBRA      D6,SCHLEIFE    * Wiederholen.
        MOVEQ     #3,D1          * Farbe 3.
        MOVEQ     #0,D2          * Auf Setzen.
        MOVEQ     #15,D0         * Kennung.
        MOVEQ     #!GRAFIK,D7    * Werte einstellen.
        TRAP      #1
        MOVE      #255,D1        * Mittelpunkt X.
        MOVE      D1,D2          * Mittelpunkt Y.
        MOVEQ     #11,D0         * Kennung.
        MOVEQ     #!GRAFIK,D7    * Fläche füllen.
        TRAP      #1
        MOVEQ     #20,D0
        MOVEQ     #!DELAY,D7     * 2 Sekunden warten.
        TRAP      #1
        MOVEQ     #!CSTS,D7      * Taste gedrückt?
        TRAP      #1
        BEQ.S     START          * Nein, dann wiederholen.
        RTS
```

*
**

Unterprogrammnummer : **12**
Programmfunktion : **Seite löschen.**
Eingaberegister : Keine.
Ausgaberegister : Keine.

Es wird die aktuelle Schreibseite in der aktuellen Farbe gelöscht. Dazu wird bei der GDP kurzfristig der Bildschirm ausgeschaltet, die Schreibseite und die Leseseite werden auf den selben Wert gesetzt. Dadurch kann der schnelle interne Löschbefehl der GDP benutzt werden.

Einfaches Beispiel:

Die Seite 0 wird als Schreibseite und Leseseite eingestellt und dann in der aktuellen Farbe gelöscht.

```
START:
        MOVEQ     #0,D1          * Leseseite = 0.
        MOVEQ     #0,D2          * Schreibseite = 0.
        MOVEQ     #13,D0
        MOVEQ     #!GRAFIK,D7    * Seite einstellen.
        TRAP      #1
        MOVEQ     #12,D0
        MOVEQ     #!GRAFIK,D7    * Seite in aktueller Farbe löschen.
        TRAP      #1
        RTS
```

*
**

Unterprogrammnummer : **13**
Programmfunktion : **Seite einstellen**
Eingaberegister : D1.B = Leseseite
D2.B = Schreibseite
Ausgaberegister : Keine.

Ähnlich dem Befehl NEWPAGE arbeitet diese Funktion. Allerdings wird hier natürlich auch die Seite der COL mit eingestellt, falls die RAM-Erweiterung vorhanden ist. In D1.B muß die Leseseite (0-3) und in D2.B die Schreibseite im gleichen Bereich eingestellt werden. Die hier gewählten Werte werden auch bei jedem Aufruf der Funktion 30 eingestellt.

Einfaches Beispiel:

Seite 0 wird als Schreibseite und Seite 3 wird als Leseseite eingestellt.

```
START:
        MOVEQ     #3,D1          * Leseseite = 3.
        MOVEQ     #0,D2          * Schreibseite = 0.
        MOVEQ     #13,D0         * Kennung.
        MOVEQ     #!GRAFIK,D7    * Werte einstellen.
        TRAP      #1
        RTS
```

*
**

Unterprogrammnummer : **14**
Programmfunktion : **Seite abfragen.**
Eingaberegister : Keine.
Ausgaberegister : D1.L = Leseseite
D2.L = Schreibseite

Diese Funktion liefert die Werte zurück, die mit der Funktion 13 eingestellt wurden bzw. den Wert 0 in D1.L und D2.L falls vorher noch keine neue Seite eingestellt wurde, denn dieser Wert ist voreingestellt.

Einfaches Beispiel:

Lese- und Schreibseite abfragen.

```
START:
        MOVEQ       #14,D0
        MOVEQ       #!GRAFIK,D7
        TRAP        #1                  * Ergebnis in D1.L und D2.L.
        RTS
```

*
**

Unterprogrammnummer : **15**
Programmfunktion : **Farbe und Verknüpfung setzen.**
Eingaberegister : D1.B = Farbe.
D2.B = Verknüpfung.
Ausgaberegister : D1.B = Für GDP und COL verschieden.

Um alle Farben der COL ansprechen zu können, reichen nicht wie bei der GDP die Befehle wie ERAPEN und SETPEN aus, sondern es können 16 verschiedene Werte eingestellt werden. Der Wert der Farbe wird in D1.B übergeben. Der Bereich geht bei der COL von 0 bis 15. Bei der GDP ist ein gerader Wert die Farbe schwarz (0, 2, 4, ...) und ein ungerader Wert die Farbe weiß (1, 3, 5, ...). Dadurch kann auch dieser Aufruf für GDP und COL gleich sein.

In D2.B muß die Verknüpfung eingestellt werden. Diese Funktion entspricht dem Befehl SETXOR. Eine 0 gibt an, daß der Punkt gesetzt werden soll. Eine 1 verknüpft die Bits des neuen Punktes mit den des alten Punktes mit dem Befehl EOR.

Bei der GDP wird in D1.B der Wert der Adresse $FFFFFF71 zurückgeliefert. Die Bedeutung der Bits dieses Registers kann dem Handbuch des GDP entnommen werden. Bei der COL hingegen wird die Bitkombination der Farbe geliefert, die in der nachfolgenden Tabelle steht. Diese Kombination ist z. B. wichtig, wenn ein Punkt ausgelesen wird, der auch diese Kombination zurückliefert. Dadurch ist ein Vergleich möglich.

Tabelle der Farben bei der COL:

Nummer (D1.B)	Bitkombination (Ergebnis D1.B)	Farbe
0	%00000000	Schwarz
1	%01010101	Weiß
2	%01000101	Gelb
3	%01000100	Grün
4	%01000001	Rot
5	%01010000	Blau
6	%01010001	Violett
7	%01010100	Zyan
8	%01000000	Dunkelgrau
9	%00010101	Hellgrau
10	%00000101	Braun
11	%00000100	Dunkelgrün
12	%00000001	Dunkelrot
13	%00010000	Dunkelblau
14	%00010100	Zyan dunkel
15	%00010001	Violett dunkel

Diese Farbwerte können natürlich durch die CLUT verfälscht werden.

Einfaches Beispiel:

Bei der GDP wird der Bildschirm weiß eingefärbt, bei der COL Dunkelblau.

```
START:
        MOVEQ       #13,D1              * Farbwert.
        MOVEQ       #0,D2               * XOR-Modus.
        MOVEQ       #15,D0              * Kennung.
        MOVEQ       #!GRAFIK,D7         * Werte einstellen.
        TRAP        #1
        MOVEQ       #12,D0              * Seite löschen.
        MOVEQ       #!GRAFIK,D7
        TRAP        #1
        RTS                             * D1.B ist Ergebnis.
```

*
**

Unterprogrammnummer : **16**
Programmfunktion : **Farbe und Verknüpfung abfragen.**
Eingaberegister : Keine.
Ausgaberegister : D1.L = Farbe.
D2.L = Verknüpfung.

Die Funktion fragt die mit Befehl 15 eingestellten Werte ab. Genau diese werden zurückgeliefert. In D1.L steht dabei die Farbe (COL : 0...15 / GDP : 0...1) und in D2.L der XOR-Modus (0/1). Voreingestellt ist der Wert 1 in D1.L und der Wert 0 in D2.L.

Einfaches Beispiel:

Die Werte des Befehls 15 werden abgefragt.

```
START:
        MOVEQ     #16,D0          * Kennung.
        MOVEQ     #!GRAFIK,D7     * Werte holen.
        TRAP      #1              * Ergebnis in D1.L und D2.L
        RTS
```

*
**

Unterprogrammnummer : **17**
Programmfunktion : **Scrollen.**
Eingaberegister : D1.W = Scrollwert.
Ausgaberegister : Carry = Gesetzt, wenn Fehler aufgetreten ist.

Der Befehl funktioniert mit der COL und mit der neuen GDPHS. Wenn die alte GDP-Karte vorhanden ist, wird das CARRY-Flag gesetzt und das Programm beendet.

Es wird mit dem Wert, der in D1.W übergeben wird, der Bildschirm nach oben oder unten verschoben. Bei der GDPHS erscheinen dabei die Zeilen, die auf der einen Seite herausgeschoben wurden, auf der anderen Seite. Der Bildschirm ist also als eine Rolle zu betrachten. Auch bei der COL mit 64 Kbyte ist es so. Mit 256 Kbyte RAM allerdings bilden alle 4 Seiten zusammen eine Rolle. Das bedeutet, daß eine herausgeschobene Zeile durch eine Zeile einer anderen Seite ersetzt wird. Der Wert liegt also bei der GDPHS zwischen 0 und 255 und bei der COL mit 256 Kbyte Ram zwischen 0 und 1023.

Bei der GDPHS kann hardwaremäßig bedingt nur in Zweier-Schritten, bei der COL sogar nur in Vierer-Schritten gescrollt werden. Der Wert in D1.W wird deshalb intern gerundet. Hier wird im Gegensatz zu allen anderen Routinen der Wert der Zeilen angegeben, die geschoben werden sollen. Es wird also nicht auf eine Auflösung von 512 * 512 umgerechnet.

Ist der Bildschirm verschoben, erscheinen natürlich auch alle späteren Ausgaben versetzt, sodaß die Koordinaten, wenn sie normal erscheinen sollen, vor dem Aufruf umgerechnet werden müssen. Die Routine CO macht das, wenn der Hardscroll angeschaltet ist.

Einfaches Beispiel:

Der Bildschirm wird bei der GDPHS und bei der COL mit 64 Kbyte Ram viermal gedreht und bei der COL mit 256 Kbyte RAM einmal ganz herum gedreht.

```
START:
        MOVE      #255,D1         * Mittelpunkt X.
        MOVE      D1,D2           * Mittelpunkt Y.
        MOVEQ     #100,D3         * Radius.
        MOVEQ     #7,D0           * Kennung.
        MOVEQ     #!GRAFIK,D7     * Kreis zeichnen, damit Verschiebung zu sehen
        TRAP      #1              * ist.
        MOVEQ     #0,D1           * Anfangswert.
```

```
SCHLEIFE:
        MOVEQ     #!SYNC,D7        * Warten, damit nicht so schnell verschoben
        TRAP      #1               * wird.
        BEQ.S     SCHLEIFE
        MOVEQ     #17,D0           * Scrollwert einstellen.
        MOVEQ     #!GRAFIK,D7
        TRAP      #1
        ADDQ      #2,D1            * Wert ändern,
        CMP       #256*4,D1        * bis 4 mal durchgeschoben wurde.
        BNE.S     SCHLEIFE
        RTS
```

*
**

Unterprogrammnummer : **18**
Programmfunktion : **Scrollwert abfragen.**
Eingaberegister : Keine.
Ausgaberegister : D1.L = Scrollwert.

Hiermit wird der Wert, der mit dem Befehl 17 eingestellt wurde, abgefragt. Dabei wird er defaultmäßig auf Null gesetzt. Der Wert wird so zurückgegeben, wie er eingestellt wurde. Es erfolgt keine Anpassung des Vorzeichens. Der Wert wird in D1.L zurückgeliefert und ist bei der alten GDP immer Null, da er nicht verändert werden kann.

Einfaches Beispiel:

Scrollwert zurücklesen.

```
START:
        MOVEQ     #18,D0           * Kennung.
        MOVEQ     #!GRAFIK,D7      * Ergebnis in D1.L
        TRAP      #1
        RTS
```

*
**

Unterprogrammnummer : **19**
Programmfunktion : **Einen Punkt auslesen.**
Eingaberegister : D1.W = X-Koordinate.
D2.W = Y-Koordinate.
Ausgaberegister : D3.B = Farbe bzw. Farbcode bei COL.
Carry = Gesetzt, wenn Fehler auftrat.

Für das Auslesen eines Punktes gibt es dieses Programm. Dabei müssen in D1.W und D2.W die Koordinaten übergeben werden. Sind die Werte außerhalb des sichtbaren Bereichs oder ist nur die alte GDP vorhanden, die keine Punkte auslesen kann, wird das CARRY-Flag gesetzt und das Programm beendet. Ansonsten ist das CARRY-Flag zurückgesetzt und in D3.B wird die Farbe zurückgeliefert.

Bei der GDP wird die Farbe in D3.B zurückgeliefert (0 = Schwarz/1 = Weiß).

Bei der COL hingegen wird der Farbwert zurückgegeben, der der Tabelle der Funktion 15 entspricht. Dieser Wert muß dann gegebenenfalls in die Nummer der Farbe umgerechnet werden.

Mit diesem Programm kann man leicht eine Bildschirmkopie von Teilen des sichtbaren Bereichs erstellen oder Bereiche verschieben.

Einfaches Beispiel:

Es wird der Punkt mit den Koordinaten 100,100 ausgelesen.

```
START:
        MOVEQ     #100,D1          * X-Koordinate.
        MOVEQ     #100,D2          * Y-Koordinate.
        MOVEQ     #19,D0           * Kennung.
        MOVEQ     #!GRAFIK,D7
        TRAP      #1               * Ergebnis in D3.B
        RTS
```

*
**

Unterprogrammnummer : **20**
Programmfunktion : **Hardcopy erstellen.**
Eingaberegister : A0.L = Gerade Zieladresse.
Ausgaberegister : Carry = Gesetzt, wenn Fehler aufgetreten ist.

Das Programm funktioniert genau so wie die HARDCOPY-Funktion beim Programm HARDCOPY, wenn die GDPHS für die Ausgabe angesprochen wird. Allerdings wird sie nicht mit der HARDCOPY-MAUS-Baugruppe, sondern mit der GDPHS durchgeführt, wodurch sie sehr viel schneller ist. Außerdem gibt es keine Extra-Funktionen. Auch hier muß A0.L auf eine gerade Zieladresse zeigen. Das CARRY-Flag wird gesetzt, wenn keine GDPHS vorhanden ist.

Bei der COL wird ein 64 Kbyte großer Ablagespeicher benötigt. Die Werte werden direkt aus dem Speicher der COL zur Zieladresse kopiert. Ein Byte enthält dann zwei Punkte, wobei auch hier zuerst der Punkt 0,255, dann der Punkt 1,255 usw. abgelegt wird. Die Bits 7, 5, 3, 1 gehören zum ersten Punkt im Byte und die Bits 6, 4, 2, 0 gehören zum zweiten Punkt.

Bei der GDP und bei der COL wird nur ein Bildschirmformat von 512*256 abgelegt, worauf geachtet werden muß, falls einzelne Punkte wieder auf den Bildschirm gebracht werden sollen.

Einfaches Beispiel:

Es wird eine Bildschirmkopie erstellt.

```
START:
        MOVEQ     #20,D0              * Kennung.
        LEA       ZIEL(PC),A0         * Zieladresse.
        MOVEQ     #!GRAFIK,D7         * Programm aufrufen.
        TRAP      #1
        RTS
ZIEL:
        DS.B      64*1024             * 64 Kbyte reservieren, da maximaler Bereich für Bild der COL.
```

*
**

Unterprogrammnummer : **21**
Programmfunktion : **Bild laden.**
Eingaberegister : A0.L = Gerade Quelladresse.
Ausgaberegister : Keine.

Dieses ist die umgekehrte Funktion zu Befehl 20. Hiermit wird ein Bild aus dem Speicher auf den Bildschirm gebracht. Beide Funktionen zusammen ermöglichen es, sehr leicht Bilder z. B. auf Diskette abzuspeichern und wieder zu laden.
Das Format der Daten kann dem Befehl 20 entnommen werden.

Auch hier muß wieder in A0.L die Adresse im Speicher angegeben werden, ab der die Bildschirmdaten liegen. A0.L muß unbedingt auf eine gerade Adresse zeigen.

Beim Laden des Bildes wird der Bildschirm nicht gelöscht. Außerdem ist natürlich der aktuelle XOR-Modus (an/aus) wirksam.

Komplexeres Beispiel:

Das aktuelle Bild wird gespeichert. Dann wird mit einer versetzten Adresse neu geladen. Dadurch verschiebt sich das Bild, und in den untersten Zeilen stehen zufällige Farbwerte.

```
START:
        LEA       ZIEL(PC),A0         * Ziel für Bilddaten.
        MOVEQ     #20,D0              * Kennung.
        MOVEQ     #!GRAFIK,D7         * Bild in Speicher kopieren.
        TRAP      #1
        LEA       ZIEL+256*4(PC),A0   * Ladeadresse.
        MOVEQ     #21,D0              * Kennung.
        MOVEQ     #!GRAFIK,D7         * Bild laden.
        TRAP      #1
        RTS
ZIEL:
        DS.B      1024*64             * Maximaler Speicherbedarf für COL-Bild.
```

*
**

Unterprogrammnummer : **22**
Programmfunktion : **Bild wandeln.**
Eingaberegister : A0.L = Gerade Adresse der Bilddaten.
A1.L = Bei COL -> Adresse der Farbtabelle.
Ausgaberegister : Keine.

Das Programm ist in der Lage, ein Bild von der COL in ein Bild für die GDP oder umgekehrt zu wandeln. Dabei muß zwischen den beiden Funktionen unterschieden werden.

Bei beiden Funktionen ist identisch, daß A0.L auf die Adresse der Bilddaten zeigen muß. A0.L muß auch hier eine gerade Adresse sein.

a) Wandlung von GDP in COL:

Wenn die Ausgabe auf die COL gelenkt ist, ist diese Funktion aktiviert. Es muß ein Bild, das von der GDP stammt, im Speicher vorliegen. Dann wird aus den 16 Kbyte, die im Speicher liegen, ein 64 Kbyte Bild, das direkt auf die COL-Karte gebracht werden kann. Dabei wird ein schwarzer Punkt auf der GDP auch zu einem schwarzen Punkt auf der COL. Ein weißer Punkt der GDP wird zu einem Punkt in der aktuellen Schreibfarbe der COL.

Komplexeres Beispiel:

Auf der GDP wird auf schwarzem Bildschirm ein weißer Kreis gemalt. Dieses Bild wird in den Speicher geladen und dann in blau auf der COL dargestellt. Der COL-Bildschirm wird auch vorher gelöscht.

```
START:
        MOVEQ   #30,D0          * Ausgabekarte wählen.
        MOVEQ   #0,D1           * Ausgabe auf GDP.
        MOVEQ   #!GRAFIK,D7
        TRAP    #1
        MOVEQ   #0,D1           * Farbe schwarz.
        MOVEQ   #0,D2           * Kein XOR-Modus.
        MOVEQ   #15,D0          * Kennung.
        MOVEQ   #!GRAFIK,D7     * Werte einstellen.
        TRAP    #1
        MOVEQ   #12,D0          * Bildschirm löschen.
        MOVEQ   #!GRAFIK,D7
        TRAP    #1
        MOVEQ   #1,D1           * Farbe weiß.
        MOVEQ   #0,D2           * Kein XOR-Modus.
        MOVEQ   #15,D0          * Werte einstellen.
        MOVEQ   #!GRAFIK,D7
        TRAP    #1
        MOVEQ   #8,D0           * Ausgefüllten Kreis zeichnen.
        MOVEQ   #100,D1         * X-Mitte.
        MOVEQ   #100,D2         * Y-Mitte.
        MOVEQ   #100,D3         * Radius.
        MOVEQ   #!GRAFIK,D7     * Funktion durchführen.
        TRAP    #1
        LEA     ZIEL(PC),A0     * Ziel für Bild.
        MOVEQ   #20,D0          * Kennung.
        MOVEQ   #!GRAFIK,D7     * Hardcopy erstellen.
        TRAP    #1
        MOVEQ   #30,D0          * Grafik-Karte einstellen.
        MOVEQ   #1,D1           * Auf COL umschalten.
        MOVEQ   #!GRAFIK,D7     * Einstellen.
        TRAP    #1
        MOVEQ   #15,D0          * Farbe einstellen.
        MOVEQ   #0,D1           * Farbe schwarz.
        MOVEQ   #0,D2           * Kein XOR-Modus.
        MOVEQ   #!GRAFIK,D7
        TRAP    #1
        MOVEQ   #12,D0
        MOVEQ   #!GRAFIK,D7     * Bildschirm löschen.
        TRAP    #1
        MOVEQ   #15,D0          * Farbe einstellen.
        MOVEQ   #5,D1           * Farbe blau.
        MOVEQ   #0,D2           * Kein XOR-Modus.
        MOVEQ   #!GRAFIK,D7     * Einstellen.
        TRAP    #1
```

```
        MOVEQ     #22,D0          * Bild wandeln.
        MOVEQ     #!GRAFIK,D7     * Durchführen.
        TRAP      #1
        MOVEQ     #21,D0          * Bild laden.
        MOVEQ     #!GRAFIK,D7     * Durchführen.
        TRAP      #1
        RTS                       * Kreis ist jetzt in blau auf der COL zu sehen.
ZIEL:
        DS.B      1024*64         * 64 Kbyte RAM für COL-Bild nötig.
```

b) Wandlung von COL in GDP:

Wenn die Ausgabe auf die GDP gestellt ist, wird ein 64-Kbyte Bild von der COL in ein 16-Kbyte Bild für die GDP gewandelt. Dabei ist es allerdings nicht ganz so einfach wie bei der Wandlung von der GDP für die COL. Da die COL 16 Farben hat und die GDP nur 2, muß ausgewählt werden, welche Farben in weiß und welche in schwarz gewandelt werden sollen. Dabei muß A1.L auf eine Tabelle zeigen, die diese Informationen enthält. Diese Tabelle muß als ersten Eintrag die Anzahl der Farben enthalten, die in weiß gewandelt werden sollen. Dann folgt die Auflistung der Farben. Alle anderen Farben werden in schwarz gewandelt.

Beispiel für die Tabelle:

```
TABELLE:
        DC.B      3               * 3 Farben
        DC.B      1               * Weiß
        DC.B      2               * Gelb
        DC.B      5               * Blau
```

Komplexeres Beispiel:

Ein blauer Kreis wird mit einem zyan-farbenen Quadrat auf der COL überlagert. Dieses Bild wird auf die GDP kopiert, wobei nur die Farben Blau und Zyan auf der GDP weiß dargestellt werden.

```
START:
        MOVEQ     #30,D0          * Ausgabekarte wählen.
        MOVEQ     #1,D1           * Ausgabe auf COL.
        MOVEQ     #!GRAFIK,D7
        TRAP      #1
        MOVEQ     #0,D1           * Farbe schwarz.
        MOVEQ     #0,D2           * Kein XOR-Modus.
        MOVEQ     #15,D0          * Kennung.
        MOVEQ     #!GRAFIK,D7     * Werte einstellen.
        TRAP      #1
        MOVEQ     #12,D0          * Bildschirm löschen.
        MOVEQ     #!GRAFIK,D7
        TRAP      #1
        MOVEQ     #5,D1           * Farbe blau.
        MOVEQ     #0,D2           * Kein XOR-Modus.
        MOVEQ     #15,D0          * Werte einstellen.
        MOVEQ     #!GRAFIK,D7
        TRAP      #1
        MOVEQ     #8,D0           * Ausgefüllten Kreis zeichnen.
        MOVEQ     #100,D1         * X-Mitte.
        MOVEQ     #100,D2         * Y-Mitte.
        MOVEQ     #100,D3         * Radius.
        MOVEQ     #!GRAFIK,D7     * Funktion durchführen.
        TRAP      #1
        MOVEQ     #15,D0          * Farbe einstellen.
        MOVEQ     #7,D1           * Farbe zyan.
        MOVEQ     #1,D2           * XOR-Modus an.
        MOVEQ     #!GRAFIK,D7     * Einstellen.
        TRAP      #1
        MOVEQ     #4,D0           * Ausgefülltes Quadrat zeichnen.
        MOVE      #130,D1         * X-Mitte.
        MOVE      #130,D2         * Y-Mitte.
        MOVEQ     #100,D3         * Radius.
        MOVEQ     #!GRAFIK,D7     * Funktion durchführen.
        TRAP      #1
        LEA       ZIEL(PC),A0     * Ziel für Bild.
        MOVEQ     #20,D0          * Kennung.
        MOVEQ     #!GRAFIK,D7     * Hardcopy erstellen.
        TRAP      #1
        MOVEQ     #30,D0          * Grafik-Karte einstellen.
        MOVEQ     #0,D1           * Auf GDP umschalten.
        MOVEQ     #!GRAFIK,D7     * Einstellen.
        TRAP      #1
```

```
        MOVEQ     #15,D0          * Farbe einstellen.
        MOVEQ     #0,D1           * Farbe schwarz.
        MOVEQ     #0,D2           * Kein XOR-Modus.
        MOVEQ     #!GRAFIK,D7
        TRAP      #1
        MOVEQ     #12,D0
        MOVEQ     #!GRAFIK,D7     * Bildschirm löschen.
        TRAP      #1
        MOVEQ     #15,D0          * Farbe einstellen.
        MOVEQ     #1,D1           * Farbe weiß.
        MOVEQ     #0,D2           * Kein XOR-Modus.
        MOVEQ     #!GRAFIK,D7     * Einstellen.
        TRAP      #1
        MOVEQ     #22,D0          * Bild wandeln.
        LEA       FARBTAB(PC),A1  * Tabelle der Farben.
        MOVEQ     #!GRAFIK,D7     * Durchführen.
        TRAP      #1
        MOVEQ     #21,D0          * Bild laden.
        MOVEQ     #!GRAFIK,D7     * Durchführen.
        TRAP      #1
        RTS
FARBTAB:
        DC.B      2               * Zwei Farben.
        DC.B      5,7             * Blau und Zyan.
        DS        0
ZIEL:
        DS.B      1024*64         * 64 Kbyte RAM für COL-Bild nötig.
```

*
**

Unterprogrammnummer : **23**
Programmfunktion : **Text ohne Vorlöschen ausgeben.**
Eingaberegister : D1.W = X-Koordinate.
D2.W = Y-Koordinate.
D3.B = Textgröße.
A0.L = Textadresse.
Ausgaberegister : Keine.

Dieses Programm gibt einen beliebigen Text auf der GDP oder der COL aus. Es entspricht genau der Funktion WRITELF. Der Unterschied besteht nur darin, daß in D3.B die Textgröße angegeben wird und nicht in D0.B. Außerdem wird die Schrift in der aktuellen Farbe ausgegeben.

Komplexeres Beispiel:

Der Bildschirm wird gelöscht. Ein Text wird entweder auf der GDP oder der COL ausgegeben.

```
START:
        MOVEQ     #0,D1           * Farbe 0.
        MOVEQ     #0,D2           * Auf Setzen.
        MOVEQ     #15,D0          * Kennung.
        MOVEQ     #!GRAFIK,D7
        TRAP      #1
        MOVEQ     #12,D0
        MOVEQ     #!GRAFIK,D7     * Aktuelle Schreibseite löschen.
        TRAP      #1
        MOVEQ     #9,D1           * Farbe 9.
        MOVEQ     #0,D2           * Auf Setzen.
        MOVEQ     #15,D0          * Kennung.
        MOVEQ     #!GRAFIK,D7
        TRAP      #1
        LEA       TEXT(PC),A0     * Textadresse.
        MOVEQ     #10,D1          * X-Koordinate.
        MOVE      #400,D2         * Y-Koordinate.
        MOVEQ     #$33,D3         * Textgröße.
        MOVEQ     #23,D0          * Kennung.
        MOVEQ     #!GRAFIK,D7     * Text ausgeben.
        TRAP      #1
        RTS
TEXT:                             * Ausgabetext.
        DC.B      'Dies ist ein Test-Text.',10
        DC.B      'Er zeigt die Textausgabe-',10
        DC.B      'Funktion des GRAFIK-Pakets.',10
        DC.B      0
```

*
**

Unterprogrammnummer : **24**
Programmfunktion : **Programmierbarer Zeichengenerator.**
Eingaberegister : D1.W = X-Position für Zeichen.
D2.W = Y-Position für Zeichen.
D3.B = Größe.
A0.L = Adresse der Zeichendaten.
Ausgaberegister : D1.W = X-Position für nächstes Zeichen.
D2.W = Y-Position für nächstes Zeichen.

Der Befehl entspricht in seiner Funktion in etwa dem Programm PROGZGE. Auch hier kann ein frei definierbares Zeichen ausgegeben werden. In A0.L steht die Adresse des Zeichens, das in der gleichen Form wie bei PROGZGE vorliegen muß. Außerdem muß bei diesem Befehl in D1.W und D2.W die Zeichenposition und in D3.B die Größe angegeben werden.

Zurückgeliefert wird in D1.W die neue X-Position und in D2.W die neue Y-Position, die aber immer gleich ist. Dadurch können mehrere Aufrufe dieses Programms hintereinander durchgeführt werden, was die Textausgabe erleichtert. Es muß dann nur die Adresse des auszugebenden Zeichens geändert werden.

Komplexeres Beispiel:

Es wird der Text AHA ausgegeben.
Wenn längere Texte ausgegeben werden sollen, wird zweckmäßigerweise eine Schleife verwendet, in der aus einer Tabelle alle Adressen geholt werden, sodaß nicht jeder Aufruf einzeln erfolgen muß, wie es hier geschieht.

```
START:
        MOVEQ   #25,D0              * Adressen der Zeichentabellen holen.
        MOVEQ   #!GRAFIK,D7
        TRAP    #1
        LEA     ('A'-' ')*5(A0),A0  * Adresse des A
        LEA     ('H'-' ')*5(A0),A1  * Adresse des H
        MOVE    #120,D1             * X-Anfangskoordinate.
        MOVE    #250,D2             * Y-Koordinate.
        MOVEQ   #$55,D3             * Schriftgröße.
        MOVE    #24,D0              * Kennung.
        MOVEQ   #!GRAFIK,D7         * A ausgeben.
        TRAP    #1
        EXG.L   A0,A1               * Adresse des H nach A0.L
        MOVEQ   #!GRAFIK,D7         * H ausgeben.
        TRAP    #1
        EXG.L   A1,A0               * Adressen zurück.
        MOVEQ   #!GRAFIK,D7         * A ausgeben.
        TRAP    #1
        RTS
```

*
**

Unterprogrammnummer : **25**
Programmfunktion : **Adressen der Zeichentabellen lesen.**
Eingaberegister : Keine.
Ausgaberegister : A0.L = Adresse der Tabelle mit allen ASCII-Zeichen.
A1.L = Adresse der Tabelle der Sonderzeichen.
Ab Version : 6.2

Damit man nicht für jedes Programm eine Zeichentabelle anlegen muß, gibt es dieses Programm. In A0.L wird die Adresse der Tabelle zurückgeliefert, die alle Standard-ASCII-Zeichen enthält. In der Tabelle sind alle Zeichen 5 Bytes lang. Es sind 92 Zeichen, beginnend mit dem Zeichen <SPACE> (Code $20), in der Tabelle abgelegt. Das letzte Zeichen ist das Zeichen mit dem Code $7F. Die Zeichen sind in der Form vorhanden, daß sie mit der Routine PROGZGE sowie mit der Funktion 24 direkt verwendet werden können.

A1.L liefert einen Zeiger auf eine Tabelle mit dem gleichen Aufbau, die aber nur 7 Einträge hat. Dies sind die deutschen Sonderzeichen, die in der Reihenfolge ä ö ü Ä Ö Ü ß abgelegt sind.

Das Ergebnis ist bei GDP und COL gleich.

Beispiel zur Berechnung der Adresse des Buchstaben A:

```
START:
        MOVEQ       #25,D0              * Kennung.
        MOVEQ       #!GRAFIK,D7         * Adressen holen.
        TRAP        #1
        LEA         ('A'-' ')*5(A0),A0  * Adresse des A in A0.L
        ....
        RTS
```

*
**

Unterprogrammnummer : **29**
Programmfunktion : **Adresse der COL-Karte einstellen.**
Eingaberegister : D1.L = Adresse der COL-Karte.
Ausgaberegister : Keine.

Da die COL-Karte auf allen Adressen betrieben werden kann, die als unteres Wort $C000 (bei 68000 und 68020 entsprechend multipliziert) haben und verschiedene Programme jeweils andere Adressen benutzen, kann mit diesem Befehl die Adresse der COL-Karte geändert werden. Die neue Adresse muß in D1.L übergeben werden. Intern wird das untere Wort auf $C000 gesetzt. Die Adresse darf bei den 68000- und 68020-Grundprogrammen nicht angepasst werden, sondern muß der Wert sein, der auch auf der COL-Karte eingestellt ist. Die Anpassung erfolgt intern. Voreingestellt ist der Wert $EC000. Ist die GRAFIK-Funktion auf die GDP-Karte geschaltet, wird der Befehl ignoriert.

Einfaches Beispiel:

Auf alte COL-Standard-Adresse ($DC000) einstellen.

```
START:
        MOVE.L      #$DC000,D1          * Neue Adresse.
        MOVEQ       #29,D0              * Kennung.
        MOVEQ       #!GRAFIK,D7         * Wert einstellen.
        TRAP        #1
        RTS
```

*
**

Unterprogrammnummer : **30**
Programmfunktion : **GRAFIK-Karte auswählen.**
Eingaberegister : D1.B = Ausgabe-Baugruppe.
Ausgaberegister : Keine.

Für die Umschaltung zwischen der GDP- und COL-Karte ist diese Funktion bestimmt. In D1.B wird ausgewählt, welche Karte angesprochen werden soll. Eine 0 schaltet die GDP-Karte an, eine 1 die COL-Karte. Dabei werden für die angesprochene Karte alle Parameter wie Seiten, Farbe und Scrollwerte sicherheitshalber neu eingestellt.

Einfaches Beispiel:

Die COL wird als Ausgabekarte eingestellt.

```
START:
        MOVEQ       #1,D1               * COL als Ausgabekarte.
        MOVEQ       #30,D0              * Kennung.
        MOVEQ       #!GRAFIK,D7
        TRAP        #1
        RTS
```

*
**

Unterprogrammnummer : **31**
Programmfunktion : **GRAFIK-Karte abfragen.**
Eingaberegister : Keine.
Ausgaberegister : D1.L = Aktuelle Ausgabe-Baugruppe.

Zur Feststellung, welche Karte die Grafik ausgibt, ist dieser Befehl bestimmt. In D1.L wird der Wert zurückgegeben, der mit Funktion 30 eingestellt wurde (0 = GDP-Karte, 1 = COL-Karte). Die GDP-Karte ist voreingestellt.

Einfaches Beispiel:

Es wird die Ausgabekarte abfragen.

```
START:
        MOVEQ       #31,D0              * Kennung.
        MOVEQ       #!GRAFIK,D7
        TRAP        #1
        RTS                             * D0.L ist Ergebnis Ausgabekarte.
```

TRAP-Nummer	:	**127**
Befehlsname	:	**GDPVERS**
Befehlsgruppe	:	Grafik.
Kurzbeschreibung	:	Information, ob GDP oder GDPHS vorhanden ist.
Eingaberegister	:	Keine.
Ausgaberegister	:	D0.L = Kennung, ob GDP- oder GDPHS-Karte. Flags.
Zerstörte Register	:	Keine.
Ab Version	:	6.0
Änderungen zu 4.3	:	——
Änderungen zu 6.1	:	Nein.
Siehe auch	:	MOVETO (8), DRAWTO (9), CLR (16), CLPG (17), WAIT (18), CMD (26), NEWPAGE (27), SETFLIP (34), SETPEN (37), ERAPEN (38), CMDPRINT (40), AUTOFLIP (60), SETXOR (77), GETXOR (78), SETCOLOR (79), GETCOLOR (80), GETXY (103), HARDCOPY (125), GRAFIK (126)

Mit Hilfe dieser Routine kann man erfahren, welche GDP-Karte aktiv ist. Gefragt wird nach den Karten GDP und GDPHS. Es wird die Information zurückgegeben, die mit den DIL-Schaltern auf der KEY eingestellt sind. Eine 0 in Register D0.L bedeutet, daß die GDP-Karte arbeitet. Eine 1 steht für die GDPHS-Karte.

Einfaches Beispiel:

GDP-Version ermitteln.

```
START:
        MOVEQ       #!GDPVERS,D7        * Programm aufrufen.
        TRAP        #1
        RTS
```

Komplexeres Beispiel:

Es wird auf dem Bildschirm ausgegeben, welche GDP vorhanden ist.

```
START:
        LEA         TEXT1(PC),A0        * Erster Text.
        MOVEQ       #!GDPVERS,D7        * Programm aufrufen.
        TRAP        #1
        BEQ.S       SPRUNG0             * Wenn 0, dann OK.
        LEA         TEXT2(PC),A0        * Sonst anderer Text.
SPRUNG0:
        MOVEQ       #$21,D0             * Schriftgröße.
        MOVEQ       #10,D1              * X-Position.
        MOVEQ       #100,D2             * Y-Position.
        MOVEQ       #!WRITE,D7          * Text ausgeben.
        TRAP        #1
        RTS
TEXT1:
        DC.B        'GDP ohne Extras',0
TEXT2:
        DC.B        'GDP mit Hardscroll und Auslesen',0
```

TRAP-Nummer	: **128**		
Befehlsname	: **SER**		
Befehlsgruppe	: SER-Baugruppe.		
Kurzbeschreibung	: CI, LO oder FLOPPY auf SER lenken.		
Eingaberegister	: D0.B = Wert für Umlenkung der einzelnen Programme. Carry = Gesetzt, wenn keine SER vorhanden.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 6.0		
Änderungen zu 4.3	: ——		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SI (104)	SO (105)	SISTS (106)
	SOSTS (107)	SIINIT (108)	SI2 (138)

Mit diesem Programm können verschiedene andere Unterprogramme auf die serielle Karte gelenkt werden. Dazu muß in D0.B festgelegt werden, welche Routinen umgelenkt werden sollen. Ein gesetztes Bit lenkt die Routine um, während ein nicht gesetztes Bit die Routinen auf ihre normale Funktion einstellt. Ist die serielle Karte nicht vorhanden, so wird keine Umlenkung vorgenommen und das CARRY-Flag ist als Kennung gesetzt. Ansonsten ist es zurückgesetzt.

Mit dieser Funktion kann der Computer zum Terminal werden, wenn man CI und CSTS umleitet und gleichzeitig CO2 mit CO2SER umlenkt. Es ist aber darauf zu achten, daß die serielle Karte vorher mit dem Befehl SIINIT initialisiert wird.

Eine Umlenkung von CI oder CSTS auf die USERCI bzw. USERCSTS hat Vorrang vor der Umlenkung auf die serielle Karte.

Wenn mehrere Programme gleichzeitig auf die serielle Karte zugreifen, muß darauf geachtet werden, daß die Daten nicht durcheinander geraten.

Bit	Umlenkung
0	CI auf SI und CSTS auf SISTS
1	LO auf SO und LSTS auf SOSTS
2	FLOPPY wird umgelenkt (siehe Befehl FLOPPY)
3	Reserviert

Einfache Beispiele:

CI und CSTS werden jetzt umgelenkt.

```
START:
        MOVEQ       #%001,D0
        MOVE        #!SER,D7
        TRAP        #1
        RTS
```

CI, CSTS und FLOPPY werden umgelenkt.

```
START:
        MOVEQ       #%101,D0
        MOVE        #!SER,D7
        TRAP        #1
        RTS
```

TRAP-Nummer	: 129		
Befehlsname	: **CO2SER**		
Befehlsgruppe	: Zeichenausgabe.		
Kurzbeschreibung	: CO2 auf serielle Karte lenken.		
Eingaberegister	: Keine.		
Ausgaberegister	: Carry = 1, wenn keine SER vorhanden.		
Zerstörte Register	: Keine.		
Ab Version	: 6.0		
Änderungen zu 4.3	: ——		
Änderungen zu 6.1	: Nein.		
Siehe auch	: CLRSCREEN (20)	CO (21)	LO (22)
	SIZE (25)	CO2 (33)	CRT (49)
	LST (50)	USR (51)	NIL (52)
	SETPASS (55)	CURSEIN (61)	CURSAUS (62)
	CHAR (63)	CURON (81)	CUROFF (82)
	CRLF (99)	GETLINE (100)	GETCURXY (101)
	SETCURXY (102)	LSTS (117)	

Dieser Befehl schaltet genau wie LST, CRT, NIL oder USR die Routine CO2 um. Danach werden alle Zeichen, die über CO2 ausgegeben werden, mit SO über die serielle Schnittstelle ausgegeben. Vorher muß aber das Programm SIINIT aufgerufen werden, damit die serielle Schnittstelle initialisiert ist. Wird das Programm aufgerufen und es ist keine serielle Karte vorhanden, so erfolgt keine Umlenkung und das CARRY-Flag ist gesetzt. Ansonsten ist es zurückgesetzt.

Es ist darauf zu achten, daß mehrere Programme gleichzeitig auf die serielle Karte gelenkt werden können. Dadurch könnten die Daten durcheinander geraten. Deshalb ist Vorsicht angebracht.

Einfaches Beispiel:

Die serielle Schnittstelle wird initialisiert und CO2 wird umgelenkt.

```
START:
        MOVEQ     #$1E,D0
        MOVEQ     #$0B,D1
        MOVEQ     #SIINIT,D7        * 9600 Baud, 8 Bit, keine Parität, 1 Stop-Bit.
        TRAP      #1
        MOVE      #!CO2SER,D7
        TRAP      #1
        RTS
```

TRAP-Nummer	: **130**
Befehlsname	: **CLUTINIT**
Befehlsgruppe	: CLUT-Baugruppe.
Kurzbeschreibung	: CLUT-Baugruppe auf Standardwerte setzen.
Eingaberegister	: Keine.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 6.0
Änderungen zu 4.3	: ——
Änderungen zu 6.1	: Nein.
Siehe auch	: CLUT (130)

Der Befehl ist für die CLUT im Zusammenhang mit der COL gedacht. Er kann dazu benutzt werden, die CLUT auf die Farbwerte einzustellen, die die COL auch ohne CLUT liefern würde.

Der Aufruf dieses Programms ist nur nötig, wenn die Farbwerte der CLUT verstellt worden sind, da das Grundprogramm die Werte beim ersten Start automatisch einstellt.

Einfaches Beispiel:

CLUT auf Standardwerte setzen.

```
START:
        MOVE        #!CLUTINIT,D7
        TRAP        #1
        RTS
```

TRAP-Nummer	: 131
Befehlsname	: CLUT
Befehlsgruppe	: CLUT-Baugruppe.
Kurzbeschreibung	: CLUT-Farbwerte beliebig setzen.
Eingaberegister	: A0.L = Adresse der Farbwerte-Tabelle.
Ausgaberegister	: A0.L = Zeigt direkt hinter die Tabelle.
Zerstörte Register	: Keine.
Ab Version	: 6.0
Änderungen zu 4.3	: ——
Änderungen zu 6.1	: Nein.
Siehe auch	: CLUTINIT (130)

Mit diesem Programm kann man die 256 Farbregister der CLUT-Baugruppe auf beliebige Farbwerte einstellen. Dazu wird in A0.L die Adresse einer Tabelle übergeben, in der die Einstellungen stehen. Zurückgeliefert wird in A0.L ein Zeiger auf das nächste Byte hinter der Tabelle. Daher können mehrere Tabellen direkt hintereinander stehen und das Programm in einer Schleife mehrfach aufgerufen werden.

Die Tabelle muß dabei folgendermaßen aussehen:

Das erste Byte enthält den Wert, der direkt in das Maskenregister der CLUT geschrieben wird. Damit wird festgelegt, welche Bits der Farben gefiltert werden. Als nächstes folgt das Byte, das das Anfangsfarbregister festlegt. In dieses Register werden die ersten Farbwerte geschrieben. Dann folgen jeweils drei Bytes für eine Farbe. Die drei Bytes stehen für die Farbwerte Rot, Grün und Blau. Diese Bytes haben einen Bereich von 0 bis 63. Das Ende der Tabelle wird mit $FF markiert. Nach drei Bytes wird die Registernummer der Farbe automatisch von der CLUT hochgezählt.

Beispiel einer Tabelle:

```
TABELLE:
        DC.B    $FF         * Maske (alle Bits durchlassen).
        DC.B    0           * Mit Farbe 0 beginnen.
        DC.B    0,0,0       * Rot = 0, grün = 0, blau = 0 ==> Farbe schwarz.
        DC.B    63,63,63    * Rot = 63, grün = 63, blau = 63 ==> weiß.
        DC.B    $FF         * Endemarkierung.
```

Einfaches Beispiel:

```
START:
        LEA     TABELLE(PC),A0      * Adresse der Tabelle.
        MOVE    #!CLUT,D7           * Aufrufen.
        TRAP    #1
        RTS
```

TRAP-Nummer	: 132
Befehlsname	: **RELAIS**
Befehlsgruppe	: RELAIS-Baugruppe.
Kurzbeschreibung	: Schaltet Relais an und aus.
Eingaberegister	: D0.B = Auswahl der zu setzenden Relais. A0.L = Adresse der RELAIS-Baugruppe (ohne CPU).
Ausgaberegister	: A0.L = Wirkliche Adresse der Baugruppe.
Zerstörte Register	: Keine.
Ab Version	: 6.0
Änderungen zu 4.3	: ——
Änderungen zu 6.1	: Nein.
Siehe auch	: RELAISIN (133)

Hiermit können die Relais der RELAIS-Baugruppe gesetzt bzw. zurückgesetzt werden. In A0.L muß die Adresse der Baugruppe angegeben werden. Dabei darf die Adresse nicht mit dem CPU-Wert multipliziert werden, da das Programm das selber macht. Eine Karte mit einer bestimmten Adresse hat also beim 68008 die gleiche Adresse wie beim 68000 oder 68020. In D0.B muß außerdem angegeben werden, welche Relais gesetzt und welche zurückgesetzt werden sollen. Dabei setzt das Bit Null das Relais Null, das Bit 1 setzt das Relais 1 usw. Ein gesetztes Bit schaltet das entsprechende Relais an.

Nach dem Aufruf steht in A0.L die wirkliche Adresse der Baugruppe, wie sie auch von der CPU angesprochen werden kann.

Einfaches Beispiel:

Die Relais 0, 2, 4 und 6 werden angeschaltet und die Relais 1, 3, 5 und 7 ausgeschaltet. Die Baugruppe soll dabei auf der Adresse $FFFFFF50 liegen.

```
START:
        MOVEQ   #%01010101,D0   * Auswahl der Relais.
        LEA     $FF50.W,A0      * Adresse der Baugruppe.
        MOVE    #!RELAIS,D7     * Relais setzen und zurücksetzen.
        TRAP    #1
        RTS
```

TRAP-Nummer	: 133
Befehlsname	: **RELAISIN**
Befehlsgruppe	: RELAIS-Baugruppe.
Kurzbeschreibung	: Port der RELAIS-Baugruppe lesen.
Eingaberegister	: A0.L = Adresse der RELAIS-Baugruppe (ohne CPU).
Ausgaberegister	: D0.B = Wert des Ports. A0.L = Wirkliche Adresse der Baugruppe.
Zerstörte Register	: Keine.
Ab Version	: 6.0
Änderungen zu 4.3	: ——
Änderungen zu 6.1	: Nein.
Siehe auch	: RELAIS (132)

Auf jeder RELAIS-Baugruppe gibt es einen Rücklese-Port, der mit dieser Funktion erreicht werden kann. Dazu muß - wie beim Befehl RELAIS - in A0.L die Basis-Adresse der RELAIS-Baugruppe übergeben werden. Auch hier darf nicht mit dem CPU-Wert multipliziert werden. Als Ergebnis wird dann in D0.B der Wert des Portes zurückgegeben. Außerdem liefert A0.L die wirkliche Adresse der Baugruppe zurück.

Einfaches Beispiel:

Der Port einer RELAIS-Baugruppe auf der Adresse $FFFFFF50 wird ausgelesen.

```
START:
        LEA     $FF50.W,A0          * Adresse der Baugruppe.
        MOVE    #!RELAISIN,D7       * Port auslesen.
        TRAP    #1
        RTS
```

TRAP-Nummer	: **134**		
Befehlsname	: **SETDA12**		
Befehlsgruppe	: AD/DA-Baugruppen.		
Kurzbeschreibung	: DA-Wandler auf AD/DA-Baugruppe setzen.		
Eingaberegister	: D0.W = Wert für Kanal 0. D1.W = Wert für Kanal 1.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 6.0		
Änderungen zu 4.3	: ——		
Änderungen zu 6.1	: Nein.		
Siehe auch	: GETAD8 (109) GETAD12 (135)	GETAD10 (110)	SETDA (111)

Die AD/DA-Baugruppe enthält zwei DA-Wandler und 16 AD-Wandler. Dieses Unterprogramm bedient den DA-Teil. Es stehen dort 2 12-Bit-Wandler zur Verfügung. In D1.W muß der Wert für den Kanal 0 und in D1.W der Wert für den Kanal 1 übergeben werden. Der DA-Wandler wertet nur 12 Bit aus.

Für die richtige Ausgangsspannung ist nicht nur der Wert in den Registern wichtig, sondern auch die Belegung der Jumper auf der Karte, da mit diesen der Spannungsbereich festgelegt wird. Näheres dazu steht im Handbuch.

Einfaches Beispiel:

Der Kanal 0 liefert eine Spannung von 0 Volt (bzw. -5 Volt bei bipolarem Betrieb) und der Kanal 1 eine Spannung von 10 Volt (bzw. 5 Volt).

```
START:
        MOVEQ   #0,D0           * Null Volt.
        MOVE    #4095,D1        * Maximale Spannung.
        MOVE    #!SETDA12,D7    * Ausgangsspannung einstellen.
        TRAP    #1
        RTS
```

TRAP-Nummer	:	135
Befehlsname	:	**GETAD12**
Befehlsgruppe	:	AD/DA-Baugruppen.
Kurzbeschreibung	:	AD-Wandler der AD/DA-Baugruppe lesen.
Eingaberegister	:	D1.B = Kanalnummer, Warte-Modus und Darstellung.
Ausgaberegister	:	D0.W = Gelesener Wert.
Zerstörte Register	:	Keine.
Ab Version	:	6.0
Änderungen zu 4.3	:	——
Änderungen zu 6.1	:	Nein.
Siehe auch	:	GETAD8 (109) GETAD10 (110) SETDA (111) SETDA12 (134)

Das Programm bedient den AD-Teil der AD/DA-Baugruppe. Dort stehen 16 Kanäle zur Verfügung, die einzeln angesteuert werden können. In D1.B muß die Kanalnummer angegeben werden. Die Nummer steht in den Bits 0 bis 3. In Bit 4 muß der Warte-Modus festgelegt werden. Eine 0 bedeutet, daß die WAIT-Leitung so lange aktiviert ist, bis der Wert gewandelt wurde. Das ist die normale Betriebsart. Wenn die CPU keine WAIT-Leitung besitzt, kann mit einer 1 in diesem Bit festgelegt werden, daß das Programm durch Polling (ständiges Abfragen) feststellt, wann der Wert zur Verfügung steht. In Bit 5 wird außerdem noch festgelegt, in welcher Zahlendarstellung der Wert zurückgeliefert werden soll. Eine 0 bedeutet, daß die Zahl mit einem Offset zur Anfangsspannung angegeben wird. Eine 1 legt die Zweier-Komplement-Darstellung fest. Diese beiden Funktionen (Bit 3 + 4) sind im Handbuch beschrieben.

Einfaches Beispiel:

Der Wert des Kanals 0 wird als vorzeichenbehaftete Größe gelesen. Dabei wird die WAIT-Leitung aktiviert.

```
START:
        MOVEQ   #%10000,D1      * WAIT-Leitung und Zweier-Komplement-Darstellung
        MOVE    #!GETAD12,D7    * Wert des Kanals 0 lesen.
        TRAP    #1              * Ergebnis steht in D1.W
        RTS
```

TRAP-Nummer	:	**136**
Befehlsname	:	**SUCHBIBO**
Befehlsgruppe	:	System-Routinen.
Kurzbeschreibung	:	Einen oder alle Bibliothekseinträge suchen.
Eingaberegister	:	D0.W = Suchart D2.W + D3.W = Name (wenn D0 = 0 oder 1). A0.L = Anfangssuchadresse (Wenn D0 = 1). A1.L = Gerade Adresse für Ablage (wenn D0 = 2).
Ausgaberegister	:	Wenn D0 = 0 oder 1: D1.L = Adresse des Eintrags. D2.L = Name erster Teil. D3.L = Name zweiter Teil. D4.L = Startadresse des Programms. D5.L = Länge des Programms. D6.B = Relokativ-Byte. D7.B = CPU-Wert. Wenn D0 = 2: A1.L = Zeigt hinter die Tabelle.
Zerstörte Register	:	D0-D2/A0 (abhängig von Suchart).
Ab Version	:	6.1
Änderungen zu 4.3	:	——
Änderungen zu 6.1	:	Die Tabelle (d0=2) hat eine Langwort-Null als Endekennung.
Siehe auch	:	GETRAM (59) GETBASIS (89) GETVAR (94) SETA5 (91) GETVERS (97) GETSN (98) GRUND (124) SYSTEM (139)

Das Programm enthält drei Varianten, die mit dem Register D0.W ausgewählt werden können. Gemeinsam ist allen drei Möglichkeiten, daß sie einen oder mehrere Einträge der Bibliothek suchen und gegebenenfalls auflisten. Es werden nur die Einträge gefunden, die den richtigen CPU-Wert haben.

Nähere Auskünfte für den Aufbau eines Bibliothekeintrags liefert Kapitel 5.7.

D0.W = 0

Es wird ein bestimmter Eintrag der Bibliothek gesucht. Dabei wird bei der Adresse $400 mit der Suche begonnen. Der Name des zu suchenden Eintrags wird in den Registern D2.L und D3.L übergeben. D2.L muß den ersten Teil des Namens enthalten und D3.L den zweiten Teil. Der Name muß genau so vorliegen, wie der Name des zu suchenden Eintrags, allerdings muß nicht auf Groß-Klein-Schreibung geachtet werden. Wird kein Eintrag gefunden, wird das CARRY-Flag gesetzt, ansonsten ist es zurückgesetzt. Außerdem werden in den aufgelisteten Registern alle Daten des Eintrags zurückgegeben.

D1.L = Absolute Adresse des Eintrags.
D2.L = Erster Teil des Namen; der Name wird original so zurückgegeben, wie er im RAM steht.
D3.L = Zweiter Teil des Namen; wie er im Eintrag gefunden wird.
D4.L = Startadresse des Programms; bei relokativen Programmen bereits berechnet.
D5.L = Länge des Programms in Bytes.
D6.B = Relokativ Byte (0 = absolut / 1 = relokativ).
D7.B = CPU-Wert (0, dann für alle CPUs geeignet, sonst Wert der System-CPU).

D0.W = 1

Die Funktion ist fast die gleiche wie bei D0.W = 0, allerdings muß in A0.L die Anfangssuchadresse übergeben werden. Dadurch kann vermieden werden, daß bei schnellen Anwendungen der ganze Speicher durchsucht werden muß.

D0.W = 2

Hierbei wird eine Tabelle mit allen Bibliothekseinträgen angelegt und von der Adresse $400 an der ganze Speicher durchsucht. Die Tabelle wird ab der Adresse abgelegt, die in A1.L übergeben wird. A1.L muß unbedingt einen geraden Wert enthalten. Die Tabelle wird mit einer 0.L als Endekennung versehen. Diese Kennung wird auch gesetzt, wenn kein Eintrag gefunden wird. Dann wird aber zusätzlich noch das CARRY-Flag gesetzt. Die einzelnen Einträge sind genau in der Form abgelegt, die oben schon bei den Registern gewählt wurde. Ein Eintrag ist also 22 Bytes lang. Da alle Einträge gesucht werden, braucht kein Name angegeben zu werden.

TRAP-Nummer	: **137**
Befehlsname	: **DISASS**
Befehlsgruppe	: DIS-Assembler.
Kurzbeschreibung	: Disassembliert einen Befehl.
Eingaberegister	: D0.L = Adresse des Befehls. A0.L = Gerade Zieladresse für Text.
Ausgaberegister	: Keine.
Zerstörte Register	: Keine.
Ab Version	: 6.1
Änderungen zu 4.3	: ——
Änderungen zu 6.1	: Nein.
Siehe auch	: ——

Im Einzelschritt wird auf diesen DIS-Assembler zurückgegriffen. Er ist in der Lage, einen Befehl, der auf einer angegebenen Adresse liegt, zu übersetzten. Eine Übersetzung von mehreren hintereinander liegenden Befehlen ist nicht möglich, da keine Länge des Befehls übergeben wird und so nicht festgestellt werden kann, wo der nächste Befehl beginnt. Das wurde nicht eingebaut, da es für den Einzelschritt nicht wichtig ist. Dieses Programm ist nämlich eigentlich nur ein "Abfallprodukt". In D0.L muß die Adresse des gewünschten Befehls übergeben werden. Sollte D0.L bei der Übergabe nicht auf eine gerade Adresse zeigen, so wird intern Bit 0 gelöscht, da Befehle ja nur auf geraden Adressen beginnen können. Der Befehl wird in ASCII-Zeichen ab der Adresse abgelegt, die in A0.L übergeben wird. A0.L muß unbedingt auf eine gerade Adresse zeigen. Wenn die fragliche Adresse keinen Befehl enthält, so wird nur der Inhalt der Adresse übergeben. Für den Zielbuffer sollten beim Einsatz der Prozessoren 68008 und 68000 etwa 60 Zeichen und bei dem Prozessor 68020 ca. 100 Zeichen reserviert werden. Der Ausgabetext wird mit einer binären Null abgeschlossen.

Einfaches Beispiel:

Der erste Befehl des Programms wird ausgegeben.

```
START:
        LEA     START(PC),A0        * Adresse dieses Befehls holen.
        MOVE.L  A0,D0               * Nach D0.L
        LEA     ZIEL(PC),A0         * Dort Text ablegen.
        MOVE    #!DISASS,D7         * DIS-Assembler aufrufen.
        TRAP    #1
        MOVEQ   #$21,D0             * Schriftgröße.
        MOVEQ   #10,D1              * X-Position.
        MOVEQ   #120,D2             * Y-Position.
        MOVEQ   #!WRITE,D7          * Befehl ausgeben.
        TRAP    #1
        RTS
ZIEL:
        DS.B    30                  * Ziel für Befehl.
```

TRAP-Nummer	: **138**		
Befehlsname	: **SI2**		
Befehlsgruppe	: SER-Baugruppe.		
Kurzbeschreibung	: Zeichen von serieller Schnittstelle.		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.L = Gelesenes Zeichen.		
Zerstörte Register	: Keine.		
Ab Version	: 6.1		
Änderungen zu 4.3	: ——		
Änderungen zu 6.1	: Nein.		
Siehe auch	: SI (104)	SO (105)	SISTS (106)
	SOSTS (107)	SIINIT (108)	SER (128)

Der Befehl ist fast identisch mit dem Befehl SI. Allerdings bedient der SI-Befehl die Handshake-Leitung RTS nicht. Dadurch können eventuell Daten verloren gehen, wenn nicht schnell genug abgefragt wird. Die Routine SI2 hingegen bedient diese Leitung, hat aber auch einen Nachteil. Die Routine SISTS kann nicht verwendet werden, da die Datenzufuhr außerhalb der Routine SI2 gesperrt ist und so auch eine Abfrage, ob Daten vorhanden sind, nicht sinnvoll ist. Es muß also bei jedem einzelnen Fall entschieden werden, welche Routine verwendet werden soll.
In D0.L wird das gelesene Zeichen zurückgeliefert, wobei natürlich nur die untersten 8 Bit interessant sind. Es wird immer so lange gewartet, bis wirklich ein Byte angekommen ist.

Einfaches Beispiel:

Ein Zeichen wird gelesen.

```
START:
        MOVE    #!SI2,D7
        TRAP    #1              * Ergebnis in D0.L
        RTS
```

TRAP-Nummer	: **139**		
Befehlsname	: **SYSTEM**		
Befehlsgruppe	: System-Routinen.		
Kurzbeschreibung	: System-Informationen lesen.		
Eingaberegister	: Keine.		
Ausgaberegister	: D0.L = Informationen bitweise kodiert.		
Zerstörte Register	: Keine.		
Ab Version	: 6.1		
Änderungen zu 4.3	: ——		
Änderungen zu 6.1	: Es sind mehr Informationen vorhanden.		
Siehe auch	: GETRAM (59)	GETBASIS (89)	GETVAR (94)
	SETA5 (91)	GETVERS (97)	GETSN (98)
	GRUND (124)	SUCHBIBO (137)	

Nach dem Aufruf dieses Programms enthält D0.L einige Informationen, die für bestimmte Anwendungen interessant sein können. Dabei sind die Informationen bitweise kodiert. Hier die Funktion der einzelnen Bits:

Bits:	Funktion:
0	1 = CPU 68008 im System vorhanden.
1	1 = CPU 68000 oder 68010.
2	1 = CPU 68020.
3	1 = Neue GDPHS vorhanden.
4	0 = GDP wird über das GRAFIK-Programm angesprochen. 1 = COL wird über das GRAFIK-Programm angesprochen.
5 - 6	0 = Keine Uhr vorhanden. 1 = Uhrenbaugruppe vorhanden. 2 = Smartwatch vorhanden. 3 = Reserviert (Uhr auf KEY3)
7	1 = Programm läuft im Einzelschritt.
8	1 = Hardscroll bei CO ist eingeschaltet.
9	1 = Einseitiges Menü ist aktuelle Menüform.
10	1 = Hardcopyfunktion bei CI auch außerhalb des Grundprogramms verfügbar.
11 - 14	Gibt an, welche Programme auf serielle Karte gelenkt sind. Bit 11 : CI und CSTS. Bit 12 : LO und LSTS. Bit 13 : FLOPPY. Bit 14 : Reserviert.
15	1 = Harddisk vorhanden.
16	Reserviert (KEY3)
17	1 = Serielle Schnittstelle vorhanden.

Einfaches Beispiel:

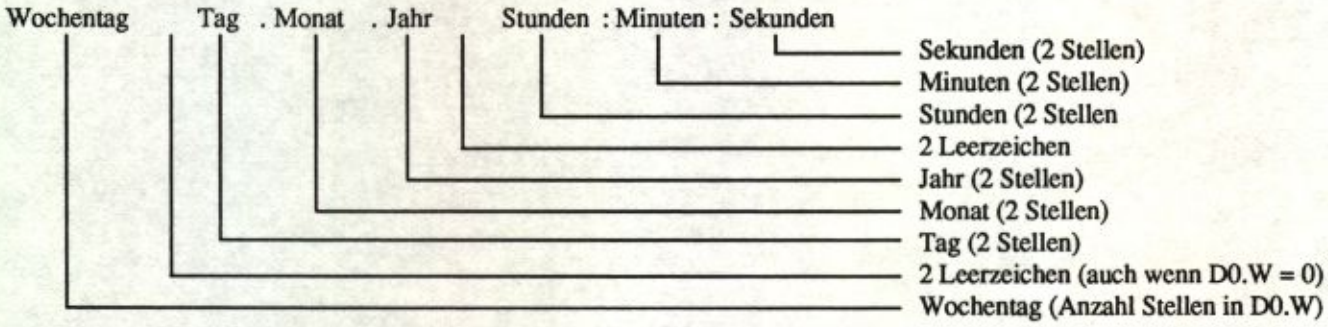
Es wird festgestellt, welche CPU vorhanden ist. Dann können leicht IO-Adressen umgerechnet werden.

```
START:
        MOVE    #!SYSTEM,D7
        TRAP    #1
        AND     #7,D0           * Nur Bits 0 bis 2 lassen.
        RTS
```

TRAP-Nummer	: **140**
Befehlsname	: **UHRPRINT**
Befehlsgruppe	: Echtzeituhren.
Kurzbeschreibung	: Daten für Uhrzeit in ASCII wandeln.
Eingaberegister	: D0.W = Anzahl der Zeichen für Wochentag. A0.L = Adresse für Ablage.
Ausgaberegister	: A0.L = Zeigt auf Endekennung. Carry = Gesetzt, wenn keine Uhr vorhanden ist.
Zerstörte Register	: Keine.
Ab Version	: 6.2
Änderungen zu 4.3	: ——
Änderungen zu 6.1	: ——
Siehe auch	: GETUHR (115) SETUHR (116)

Das Programm wandelt Daten für das Datum und die Uhrzeit in eine lesbare Form. Es wird auch der Wochentag und das Datum ausgegeben. In D0.W wird angegeben, wieviele Zeichen der Wochentag maximal haben darf. Ist D0.W = 0, wird kein Wochentag ausgegeben. Bei einer Zahl in D0.W wird so lange ausgegeben, bis die maximale Anzahl übertragen wurde oder bis der Wochentag übertragen wurde. Hinter dem Wochentag wird das Datum und die Uhrzeit ausgegeben. Die Ablage findet ab der Adresse statt, die in A0.L übergeben wurde. Das Ende wird mit einen binären Null gekennzeichnet. Auf diese Null zeigt A0.L nach dem Aufruf. Das CARRY-Flag wird gesetzt, falls keine Uhr vorhanden ist, da vor der Wandlung die Uhrzeit natürlich gelesen wird.

Ausgabeformat:



Einfaches Beispiel:

Die Uhrzeit wird mit voll ausgeschriebenem Wochentag ausgegeben.

```
START:
        MOVEQ       #-1,D0              * Maximale Anzahl.
        LEA         ZIEL(PC),A0         * Ziel für Uhrzeit.
        MOVE        #!UHRPRINT,D7       * Uhrzeit lesen und ablegen.
        TRAP        #1
        BCS.S       ENDE                * Ende, wenn keine Uhr vorhanden ist.
        LEA         ZIEL(PC),A0         * Dort steht Uhrzeit.
        MOVEQ       #$11,D0             * Schriftgröße.
        MOVEQ       #0,D1               * X-Position.
        MOVE.W      #242,D2             * Y-Position.
        MOVEQ       #!WRITE,D7          * Text ausgeben.
        TRAP        #1
ENDE:
        RTS
ZIEL:
        DS.B        30
```

TRAP-Nummer : 141
Befehlsname : **HARDDISK**
Befehlsgruppe : SCSI-Harddisk.
Kurzbeschreibung : Befehl an Harddisk übergegeben und ausführen.

Eingaberegister : D1.W = Auswahl des Befehls (0 - 31).
D2.L = Je nach Befehl verschieden.
D3.L = Je nach Befehl verschieden.
D4.B = Auswahl der Harddisk.
A0.L = Adresse der Daten, wenn verlangt.
A1.L = Adresse des Befehls bei Funktion 29.
Ausgaberegister : D0.L = Fehlercode.
Carry = 1, wenn kein Laufwerk vorhanden ist.
Zerstörte Register : Keine.
Ab Version : 6.2
Änderungen zu 4.3 : ——
Änderungen zu 6.1 : ——
Siehe auch : HARDTEST (142)

ACHTUNG !!!

Eine SCSI-Harddisk sollte nur der fortgeschrittene Programmierer ansprechen. Er muß dann unbedingt auch eine Beschreibung des Befehlssatzes der HARDDISK sowie der zurückgelieferten Fehlercodes besitzen. Alle HARDDISK-Routinen des Grundprogramms funktionieren nur mit SEAGATE-kompatiblen Laufwerken, da die Befehle auf diese abgestimmt sind.

Mit dem Befehl kann man mit jeder SCSI-Harddisk arbeiten. Es wird jeder beliebige Befehl ausgeführt. Die Adresse der SCSI-Schnittstelle steht im Kapitel 2. Zum Betrieb muß auch der Schalter für die HARDDISK auf der KEY-Karte gesetzt werden.

Da viele verschiedene Befehle ausgeführt werden können, werden diese extra aufgeführt. Gemeinsam ist allen Befehlen, daß in D1.W der Befehl ausgewählt wird. Welche Funktionen mit welcher Nummer belegt sind, folgt unten. D2 und D3 haben je nach Befehl verschiedene Funktionen. In D4.B wird die HARDDISK ausgewählt, die angesprochen werden soll, da die SCSI-Schnittstelle 8 Laufwerke verwalten kann. Ein gesetztes Bit 0 entspricht dem Laufwerk 0, ein gesetztes Bit 1 dem Laufwerk 1 usw. Die Kodierung ist also genau so wie bei einem Diskettenlaufwerk.

Nach dem Aufruf steht in D0.L ein Fehlercode. Ist keine HARDDISK vorhanden, erhält D0.L den Wert -1, und das CARRY-Flag ist gesetzt. Ansonsten wird in D0.L der Fehlercode der HARDDISK zurückgeliefert, der sehr viele verschiedene Werte annehmen kann und je nach Befehl verschieden ist. In Beschreibungen über SCSI-Schnittstellen ist meistens auch eine Tabelle der Returncodes aufgeführt. In D0.L wird übrigens eine -1 zurückgegeben, wenn D1.W beim Aufruf einen falschen Wert enthalten hat (FALSCHER BEFEHL).

Es folgt die detaillierte Beschreibung der einzelnen Befehle. Register werden nicht zerstört. Außer in D0.L werden keine Werte zurückgegeben. Es ist darauf zu achten, daß nicht alle Festplatten alle Befehle ausführen können. Weiter hilft auch hier eine ausführliche Beschreibung. Der Basisbefehlssatz wird von allen Laufwerken akzeptiert. Er ist mit (SB) gekennzeichnet.

*
**

Nummer des Befehls : **0**
Befehlsname : **Rezero Unit (SB).**
Kurzbeschreibung : Harddisk zurücksetzen (auf Track 0).
Eingaberegister : Keine.

Der Schreib-Lese-Kopf wird auf Track 0 gesetzt.

Einfaches Beispiel:

Der Kopf des Laufwerks Null wird auf die Spur Null gefahren.

```
START:
        MOVEQ       #0,D1               * Befehlskennung.
        MOVEQ       #1,D4               * Laufwerk 0.
        MOVE        #!HARDDISK,D7       * Befehl ausführen.
        TRAP        #1
        ...                             * Fehlererkennung.
        RTS
```

*
**

Nummer des Befehls : **1**
Befehlsname : **Read (SB).**
Kurzbeschreibung : Bis zu 256 Sektoren lesen.
Eingaberegister : D2.L = Anfangssektor (nur 21 Bits genutzt).
D3.B = Anzahl der Sektoren.
A0.L = Zielladresse.

Der Befehl liest bis zu 256 hintereinander liegende Sektoren in den Speicher. In D2.L wird der Anfangssektor angegeben. Der kleinste Sektor ist Sektor 0. Der höchste Sektor hängt von der Sektorlänge und der Größe der Festplatte ab und kann mit dem Befehl READ CAPACITY ermittelt werden. In D3.B muß die Anzahl der Sektoren angegeben werden. Eine Null steht für 256 Sektoren. Wenn von einem DOS Sektoren geladen werden sollen, die hintereinander liegen, sollten sie immer in einem Stück eingelesen werden, da dies sehr viel schneller geht als das mehrmalige Einlesen eines Sektors. In A0.L muß die Zieladresse im RAM angegeben werden. Die Anzahl der übertragenen Bytes pro Sektor hängt von der Formatierung ab. Die Blocklänge kann mit dem Befehl MODE SENSE ermittelt werden.

Einfaches Beispiel:

Es werden 256 Sektoren ab dem Sektor 5 der HARDDISK 1 gelesen.

```
START:
        MOVEQ       #1,D1           * Befehlskennung.
        MOVEQ       #5,D2           * Ab Sektor 5.
        MOVEQ       #0,D3           * 256 Sektoren.
        MOVEQ       #2,D4           * Laufwerk 1, da Bit 1 gesetzt.
        LEA         ZIEL(PC),A0     * Zieladresse.
        MOVE        #!HARDDISK,D7
        TRAP        #1
        ....                        * Fehlerbehandlung.
        RTS
ZIEL:                               * Maximaler Platzbedarf.
        DS.B        1024*256
```

*
**

Nummer des Befehls : **2**
Befehlsname : **Write (SB).**
Kurzbeschreibung : Bis zu 256 Sektoren schreiben.
Eingaberegister : D2.L = Anfangssektor (nur 21 Bits genutzt).
D3.B = Anzahl der Sektoren.
A0.L = Quelladresse.

Das Programm funktioniert genau so wie die Funktion 1 READ. Allerdings werden die Sektoren geschrieben und nicht gelesen.

Komplexeres Beispiel:

Es wird der Sektor 1234 der Festplatte 1 gelesen und auf den Sektor 0 des Laufwerks 0 geschrieben.

```
START:
        MOVEQ       #1,D1           * Sektor lesen.
        MOVE.L      #1234,D2        * Sektor 1234 lesen.
        MOVEQ       #1,D3           * 1 Sektor.
        MOVEQ       #2,D4           * Laufwerk 1, da Bit 1 gesetzt.
        LEA         ZIEL(PC),A0     * Zieladresse.
        MOVE        #!HARDDISK,D7
        TRAP        #1
        ....                        * Fehlerbehandlung.
        MOVEQ       #2,D1           * Sektor schreiben.
        MOVEQ       #0,D2           * Sektor 0.
        MOVEQ       #1,D3           * 1Sektor.
        MOVEQ       #1,D4           * Laufwerk 0.
        MOVE        #!HARDDISK,D7
        TRAP        #1
        ....                        * Fehlerbehandlung.
        RTS
ZIEL:                               * Maximaler Platzbedarf.
        DS.B        1024
```

*
**

Nummer des Befehls : **3**
Befehlsname : **Read long.**
Kurzbeschreibung : Sektoren und Prüfbytes lesen.
Eingaberegister : D2.L = Anfangssektor (Nur 21 Bits genutzt).
A0.L = Zieladresse.

Auf dem Medium eines Laufwerks befinden sich nicht nur die Datenbytes, sondern auch Prüfbytes, die normalerweise vom Benutzer nicht benötigt werden. Sollen sie doch gelesen werden, muß der Befehl benutzt werden. Es wird ein Sektor mit den dazugehörigen ECC-Bytes gelesen. In D2.L muß dazu die Nummer des Sektors übergeben werden und in A0.L die Zieladresse für die Daten. Die Anzahl der übertragenen Bytes hängt von der Größe des Sektors ab.

Einfaches Beispiel:

Der Sektor 4321 wird mit Prüfbytes gelesen.

```
START:
        MOVEQ       #3,D1           * Befehlskennung.
        MOVEQ       #1,D4           * Laufwerk 0.
        MOVE.L      #4321,D2        * Sektor 4321.
        LEA         ZIEL(PC),A0     * Ziel für Daten.
        MOVE        #!HARDDISK,D7   * Befehl ausführen.
        TRAP        #1
        ....                        * Fehlerbehandlung.
        RTS
ZIEL:                               * Ziel für Daten.
        DS.B        2000
```

*
**

Nummer des Befehls : **4**
Befehlsname : **Write Long.**
Kurzbeschreibung : Sektoren und Prüfbytes schreiben.
Eingaberegister : D2.L = Anfangssektor (nur 21 Bits genutzt).
A0.L = Quelladresse.

Der Befehl entspricht in seinem Aufruf der Funktion 3. Allerdings wird der Sektor mit den ECC-Bytes auf die Festplatte geschrieben und nicht gelesen.

*
**

Nummer des Befehls : **5**
Befehlsname : **Mode Select (SB).**
Kurzbeschreibung : Laufwerksparameter ändern.
Eingaberegister : D2.B = Anzahl der zu übertragenden Bytes.
A0.L = Adresse der Daten.

Der Befehl veranlaßt die HARDDISK, bestimmte Einstellungen zu ändern. Wie das genau gemacht wird, steht im Handbuch zur HARDDISK. In D2.B muß die Länge der Parameter-Liste übergeben werden. In A0.L wird die Anfangsadresse der Liste angegeben.

Einfaches Beispiel:

Das Laufwerk 0 wird so eingestellt, daß die maximal mögliche Anzahl von Sektoren benutzt werden kann. Das ist normalerweise die Voreinstellung.

```
START:
        MOVEQ     #5,D1              * Befehlskennung.
        MOVEQ     #1,D4              * Laufwerk 0.
        MOVEQ     #ENDE-QUELLE,D2    * Länge der Daten in Bytes.
        LEA       QUELLE(PC),A0      * Adresse der Daten.
        MOVE      #!HARDDISK,D7      * Befehl ausführen.
        TRAP      #1
        ....                         * Fehlerbehandlung.
        RTS
QUELLE:
        DC.B      0                  * Reserviert.
        DC.B      0                  * Feste Harddisk.
        DC.B      0                  * Reserviert.
        DC.B      8                  * 8 Bytes folgen für Block-Deskriptor.
        DC.B      0                  * Feste Harddisk (Density Code).
        DC.B      0,0,0              * Alle Sektoren verfügbar.
        DC.B      0                  * Reserviert.
        DC.B      0,0,0              * Voreingestellte Sektorenlänge benutzen.
ENDE:
```

*
**

Nummer des Befehls : **6**
Befehlsname : **Mode Sense (SB).**
Kurzbeschreibung : Laufwerksparameter lesen.
Eingaberegister : D2.B = Page-Control-Field und Page-Code.
D3.B = Maximale Anzahl der zu übertragenden Bytes.
A0.L = Zieladresse der Daten.

Das Programm ist das Gegenstück zum Befehl 5. Er ist dafür gedacht, Einstellungen der HARDDISK zurückzulesen, um so z. B. festzustellen, wie groß ein Sektor oder wie der INTERLEAVE-Faktor eingestellt ist. In D2.B muß dazu angegeben werden, welche Informationen übertragen werden sollen. Nähere Angaben stehen im Handbuch. Bit 6 und 7 entsprechen dem Page-Control-Field und die Bits 0 bis 5 dem Page-Code. In D3.B kann angegeben werden, wie viele Bytes maximal übergeben werden sollen. Sind entsprechend viele Bytes übergeben worden, werden keine weiteren Daten gesendet, auch wenn noch nicht alle Informationen übertragen worden sind. In A0.L schließlich muß noch die Zieladresse im RAM angegeben werden.

Einfaches Beispiel:

Es werden die Anzahl der Zylinder, die Anzahl der Köpfe und einige andere Informationen gelesen.

```
START:
        MOVEQ     #1,D1              * Befehlskennung.
        MOVE.B    #%10001000,D2      * PC und Page 4.
        MOVEQ     #32,D3             * Maximale Anzahl von Bytes.
        LEA       ZIEL(PC),A0        * Ziel für Daten.
        MOVE      #!HARDDISK,D7      * Befehl ausführen.
        TRAP      #1
        ....                         * Fehlerbehandlung.
        RTS
ZIEL:                                * Hier werden die Daten abgelegt.
        DS.B      1                  * Anzahl der Bytes ohne dieses Byte.
        DS.B      1                  * Typ des Mediums (hier: 0).
        DS.B      1                  * Reserviert.
        DS.B      1                  * Länge des Block-Deskriptors.
        DS.B      1                  * Dichte Code (hier: 0).
        DS.B      3                  * Anzahl der verfügbaren Sektoren.
        DS.B      1                  * Reserviert.
        DS.B      3                  * Länge eines Sektors.
        DS.B      1                  * Page-Code (hier: 4).
        DS.B      1                  * Länge der folgenden Bytes (einschließlich).
        DS.B      3                  * Anzahl der Zylinder.
        DS.B      1                  * Anzahl der Köpfe.
        DS.B      14                 * Reserviert.
```

*
**

Nummer des Befehls : **7**
Befehlsname : **Seek (SB).**
Kurzbeschreibung : Einen Sektor suchen.
Eingaberegister : D2.L = Nummer des Sektors (nur 21 Bits genutzt).

Hiermit kann man einen bestimmten Sektor suchen. Die Nummer des Sektors muß in D2.L angegeben werden. Es werden keine Daten des Sektors gelesen.

Einfaches Beispiel:

Der Sektor 0 wird gesucht und somit der Schreib-Lese-Kopf auf Spur 0 gefahren.

```
START:
        MOVEQ       #7,D1               * Befehlskennung.
        MOVEQ       #1,D4               * Laufwerk 0.
        MOVEQ       #0,D2               * Sektor 0 suchen.
        MOVE        #!HARDDISK,D7       * Befehl ausführen.
        TRAP        #1
        ....                            * Fehlerbehandlung.
        RTS
```

*
**

Nummer des Befehls : **8**
Befehlsname : **Test Unit Ready (SB).**
Kurzbeschreibung : Testen, ob Laufwerk bereit ist.
Eingaberegister : Keine.

Es kann vorkommen, daß die HARDDISK nicht bereit ist, einen Befehl auszuführen. Das passiert z. B. wenn das Laufwerk gerade erst eingeschaltet worden ist. Für die Abfrage, ob das Laufwerk Daten übertragen kann, ist dieses Programm bestimmt.

Einfaches Beispiel:

Es wird so lange gewartet, bis das Laufwerk für die Befehlsausführung bereit ist.

```
START:
        MOVEQ       #8,D1               * Befehlskennung.
        MOVEQ       #1,D4               * Laufwerk 0.
        MOVE        #!HARDDISK,D7       * Befehl ausführen.
        TRAP        #1
        BCS.S       ENDE                * Keine Festplatte vorhanden.
        CMP.B       #4,D0               * Laufwerk fertig?
        BEQ.S       START               * Nein, dann wiederholen.
        ....                            * Fehlerbehandlung.
ENDE:
        RTS
```

*
**

Nummer des Befehls : **9**
Befehlsname : **Stop (SB).**
Kurzbeschreibung : Laufwerkskopf parken.
Eingaberegister : Keine.

Der Laufwerkskopf wird in einen Bereich gefahren, der keine Daten enthält. Dann erst sollte das Laufwerk ausgeschaltet werden. Der Kopf setzt nämlich nach dem Ausschalten auf der Platte auf und kann so im Laufe der Zeit Daten zerstören. Wird er aber geparkt, passiert nichts. Einige Laufwerke haben eine automatische Parkeinrichtung, sodaß dann dieser Befehl nicht notwendig ist. Ein DOS sollte ihn auf jeden Fall zur Verfügung stellen.

Einfaches Beispiel:

Kopf des Laufwerks 0 wird geparkt.

```
START:
        MOVEQ       #9,D1               * Befehlskennung.
        MOVEQ       #1,D4               * Laufwerk 0.
        MOVE        #!HARDDISK,D7       * Befehl ausführen.
        TRAP        #1
        ....                            * Fehlerbehandlung.
        RTS
```

*
**

Nummer des Befehls : **10**
Befehlsname : **Start (SB).**
Kurzbeschreibung : Laufwerkskopf aus Parkposition herausbringen.
Eingaberegister : Keine.

Hiermit wird der Laufwerkskopf wieder aus der Parkposition in den Datenbereich gefahren. Das ist das Gegenstück zu Funktion 9.

Einfaches Beispiel:

Der Kopf des Laufwerks 0 wird in Betriebsbereitschaft versetzt. Der Laufwerkskopf wandert auf Spur 0.

```
START:
        MOVEQ       #10,D1              * Befehlskennung.
        MOVEQ       #1,D4               * Laufwerk 0.
        MOVE        #!HARDDISK,D7       * Befehl ausführen.
        TRAP        #1
        ....                            * Fehlerbehandlung.
        RTS
```

*
**

Nummer des Befehls : **11**
Befehlsname : **Extended Read.**
Kurzbeschreibung : Sektoren lesen.
Eingaberegister : D2.L = Nummer des ersten Sektors.
D3.W = Anzahl der Sektoren.
A0.L = Zieladresse der Daten.

Da die großen Festplatten manchmal zu viele Sektoren haben, um sie mit dem "normalen" READ-Befehl erreichen zu können, braucht man diese Funktion. Mit ihr können wirklich alle Sektoren gelesen werden. Außerdem ist es möglich, mehr als 256 Sektoren auf einmal zu lesen. Wie beim Befehl READ muß in D2.L die Nummer des ersten Sektors übergeben werden. In D3.W wird die Anzahl der hintereinander liegenden und zu lesenden Sektoren übergeben. Bei einer Null wird kein Sektor übertragen. A0.L gibt die Zieladresse für die Daten an.

Einfaches Beispiel:

Es werden 500 Sektoren ab dem Sektor 12345 gelesen.

```
START:
        MOVEQ       #11,D1              * Befehlskennung.
        MOVE.L      #12345,r2           * Anfangssektor.
        MOVE.W      #500,D3             * Anzahl.
        MOVEQ       #1,D4               * Laufwerk 0.
        LEA         ZIEL(PC),A0         * Ziel für Daten.
        MOVE        #!HARDDISK,D7       * Befehl ausführen.
        TRAP        #1
        ....                            * Fehlerbehandlung.
        RTS
ZIEL:
        DS.B        1024*500            * Maximaler Speicherbedarf.
```

*
**

Nummer des Befehls : **12**
Befehlsname : **Extended Write.**
Kurzbeschreibung : Sektoren schreiben.
Eingaberegister : D2.L = Nummer des ersten Sektors.
D3.W = Anzahl der Sektoren.
A0.L = Quelladresse der Daten.

Als Gegenstück zur Funktion 11 ist dieses Programm vorhanden. Der Unterschied besteht nur darin, daß die Sektoren geschrieben und nicht gelesen werden. In D2.L wird wieder der Anfangssektor, in D3.W die Anzahl der Sektoren und in A0.L die Quelladresse übergeben.

Einfaches Beispiel:

Es werden 250 Sektoren ab dem Sektor 76 geschrieben.

```
START:
        MOVEQ       #12,D1              * Befehlskennung.
        MOVE.L      #76,D2              * Anfangssektor.
        MOVE.W      #250,D3             * Anzahl.
        MOVEQ       #1,D4               * Laufwerk 0.
        LEA         QUELLE(PC),A0       * Ziel für Daten.
        MOVE        #!HARDDISK,D7       * Befehl ausführen.
        TRAP        #1
        ....                            * Fehlerbehandlung.
        RTS
QUELLE:                                 * Irgendwelche Daten.
        DS.B        1024*250            * Maximaler Speicherbedarf.
```

*
**

Nummer des Befehls : **13**
Befehlsname : **Read Buffer.**
Kurzbeschreibung : Aus internem Buffer lesen.
Eingaberegister : D2.W = Maximale Anzahl der zu übertragenden Bytes.
A0.L = Zieladresse der Daten.

Jede HARDDISK hat intern einen gewissen Speicher, um Daten, die geschrieben werden sollen oder Daten, die gerade gelesen wurden, so lange zwischenzuspeichern, bis sie abgerufen bzw. geschrieben werden. Bei einigen Festplatten kann dieser Buffer ausgelesen oder beschrieben werden. Das ist normalerweise nicht notwendig, sondern nur für Testzwecke gedacht. Die Daten im Buffer hängen von dem vorherigen Befehl ab und haben keinen DEFAULT-Wert.

Hiermit kann man den internen Buffer auslesen, wobei in D2.W die maximale Anzahl der zu lesenden Bytes angegeben wird. Es werden dabei natürlich nie mehr Bytes gelesen als wie der Buffer Speicher hat. In A0.L muß die Zieladresse für die Daten aus dem Buffer übergeben werden. Der übertragene HEADER enthält übrigens die Angabe, wieviel Bytes wirklich vorliegen.

*
**

Nummer des Befehls : **14**
Befehlsname : **Write Buffer.**
Kurzbeschreibung : In internen Buffer schreiben.
Eingaberegister : D2.W = Maximale Anzahl der zu übertragenden Bytes.
A0.L = Quelladresse der Daten.

Der Befehl schreibt die angegebene Anzahl von Bytes in den Buffer des Laufwerks. Dabei gilt das gleiche wie bei Funktion 13. Wenn mehr Daten übertragen werden sollen als in den Buffer passen, wird die Datenübertragung beendet, wenn der Buffer voll ist.

*
**

Nummer des Befehls : **15**
Befehlsname : **Reserve (SB).**
Kurzbeschreibung : Laufwerk reservieren.
Eingaberegister : D2.W = 3rd PTY/3rd PTY Device ID/Extend/Reservation ID.
D3.W = Länge der Daten.
A0.L = Adresse der zu übertragenden Daten.

Der Befehl wird nur in wenigen Fällen benötigt. Für den NDR-Computer kommt er normalerweise nicht in Betracht. Damit kann ein Rechner einen Teil der Festplatte für sich reservieren, sodaß kein anderer auf diesen Teil zugreifen kann.

Eventuell kann es erforderlich sein, daß ein Rechner bei gleichzeitig ablaufenden Programmen Reservierungen vornehmen kann. Das ist für Multi-Tasking-Systeme von Bedeutung.

In D2.W wird in Bit 12 die 3rd PTY-Kennung übergeben, in den Bits 9 bis 11 die 3rd PTY Identifikation und in den Bits 0 bis 7 die Reservation Identifikation. Die genaue Erklärung der einzelnen Bits liefert ein Handbuch für SCSI-Schnittstellen. In D3.W muß die Länge der zu übertragenden Daten angegeben werden, die die Informationen über reservierte Bereiche enthalten. A0.L gibt die Quelladresse der Daten an.

*
**

Nummer des Befehls : **16**
Befehlsname : **Release (SB).**
Kurzbeschreibung : Laufwerk freigeben.
Eingaberegister : D2.W = 3rd PTY/3rd PTY Device ID/Extend/Reservation ID

Hiermit wird ein mit Funktion 15 (RESERVE) reserviertes Laufwerk wieder freigegeben. In D2.W wird derselbe Wert übergeben wie bei Funktion 15. Die reservierten Bereiche sind wieder für alle Programme oder Rechner verfügbar.

*
**

Nummer des Befehls : **17**
Befehlsname : **Write + Verify.**
Kurzbeschreibung : Sektoren schreiben und prüfen.
Eingaberegister : D2.L = Anfangssektor.
D3.W = Anzahl der Sektoren.
A0.L = Quelladresse der Daten.

Wenn ein Festplattenlaufwerk einen oder mehrere Sektoren auf das Medium schreibt, wird wie bei einem Diskettenlaufwerk nicht geprüft, ob die Daten richtig geschrieben sind, da normalerweise keine Fehler auftreten. Sollen die aufgezeichneten Daten aber gleich nach dem Schreiben auf Richtigkeit überprüft werden, ist ein Lesevorgang notwendig, der viel Zeit in Anspruch nimmt. Manche Laufwerke können dieses Prüfen zusammen mit dem Schreiben in einem Durchgang erledigen, was zwar immer noch viel Zeit benötigt, aber wesentlich schneller ist als beim Diskettenlaufwerk. Dazu muß in D2.L der Anfangssektor und in D3.W die Anzahl der Sektoren übergeben werden. Die Sektoren werden wie beim WRITE-Befehl hintereinander auf die Platte geschrieben. Die Daten werden von A0.L an dem Speicher entnommen.

Der Befehl ist nur bei Programmen notwendig, die eine sehr große Datensicherheit haben müssen, da bei jedem READ-Befehl vom Laufwerk automatisch eine Fehlerkorrektur durchgeführt wird.

Komplexeres Beispiel:

Es werden die Sektoren 0 bis 99 einschließlich gelesen und dann wieder geschrieben, wobei aber gleichzeitig geprüft wird, ob die Daten auch richtig auf der Platte abgelegt wurden.

```
START:
        MOVEQ     #11,D1          * Befehlskennung.
        MOVEQ     #0,D2           * Anfangssektor.
        MOVE.W    #100,D3         * Anzahl.
        MOVEQ     #1,D4           * Laufwerk 0.
        LEA       ZIEL(PC),A0     * Ziel für Daten.
        MOVE      #!HARDDISK,D7   * Befehl ausführen.
        TRAP      #1
```

```
        ....                            * Fehlerbehandlung.
        MOVEQ       #17,D1              * Befehlskennung.
        MOVE        #!HARDDISK,D7
        TRAP        #1
        ....                            * Fehlerbehandlung.
        RTS
ZIEL:
        DS.B        1024*100            * Maximaler Speicherbedarf.
```

*
**

Nummer des Befehls : **18**
Befehlsname : **Verify.**
Kurzbeschreibung : Sektoren prüfen.
Eingaberegister : D2.L = Anfangssektor.
D3.W = Anzahl der Sektoren.

Hiermit können bereits geschriebene Sektoren überprüft werden. Dazu wird intern im Laufwerk die gewünschte Anzahl von Sektoren gelesen und überprüft. Das sollte von Zeit zu Zeit durchgeführt werden, um zu testen, ob noch alle Daten ordnungsgemäß auf der Festplatte vorhanden sind, bevor größere Schäden entstehen.

Es wird mit dem Sektor, der in D2.L angegeben ist, mit dem Prüfvorgang angefangen und die in D3.W gewünschte Anzahl von Sektoren geprüft. Eine 0 in D3.W gibt an, daß kein Sektor geprüft werden soll.

Einfaches Beispiel:

Die Sektoren 1234 bis 1300 werden auf Laufwerk 0 überprüft.

```
START:
        MOVE.L      #1234,D2            * Anfangssektor.
        MOVE.W      #1300-1234+1,D3     * Anzahl der Sektoren.
        MOVEQ       #18,D1              * Befehlskennung.
        MOVEQ       #1,D4               * Laufwerk 0.
        MOVE        #!HARDDISK,D7       * Befehl ausführen.
        TRAP        #1
        ....                            * Fehlerbehandlung.
        RTS
```

*
**

Nummer des Befehls : **19**
Befehlsname : **Send Diagnostic (SB).**
Kurzbeschreibung : Laufwerk soll Selbsttest durchführen.
Eingaberegister : Keine.

Das angesprochene Laufwerk führt einen internen Selbsttest durch. Dabei wird aber keine Mitteilung zurückgegeben, ob der Test ohne Fehler verlaufen ist. Es gibt aber verschiedene andere Befehle, um Fehlermitteilungen zu erhalten.

Einfaches Beispiel:

Laufwerk 1 führt einen Selbsttest durch.

```
START:
        MOVEQ       #19,D1              * Befehlskennung.
        MOVEQ       #2,D4               * Laufwerk 1, da Bit 1 gesetzt.
        MOVE        #!HARDDISK,D7       * Befehl ausführen.
        TRAP        #1
        ....                            * Fehlerbehandlung.
        RTS
```

*
**

Nummer des Befehls : **20**
Befehlsname : **Extended Seek.**
Kurzbeschreibung : Sektor suchen.
Eingaberegister : D2.L = Sektornummer.

Der Schreibkopf wird auf den in D2.L angegebenen Sektor gefahren.

Dies ist nur eine Erweiterung des Befehls SEEK, der bei großen Festplatten mit sehr vielen Sektoren benötigt wird.

Komplexeres Beispiel:

Der letzte Sektor der Festplatte wird gesucht und der Kopf dort positioniert.

```
START:
        MOVEQ       #0,D2           * Keine Sektorenangabe.
        MOVEQ       #0,D3           * Nummer des letzten Sektors holen.
        MOVEQ       #22,D1          * Kennung für READ CAPACITY.
        MOVEQ       #1,D4           * Laufwerk 0.
        LEA         ZIEL(PC),A0     * Ziel für Daten.
        MOVE        #!HARDDISK,D7   * Befehle ausführen.
        TRAP        #1
        ....                        * Fehlerbehandlung.
        MOVE.L      (A0),D2         * Höchster Sektor.
        MOVEQ       #22,D1          * Befehlskennung.
        MOVE        #!HARDDISK,D7   * Befehle ausführen.
        TRAP        #1
        ....                        * Fehlerbehandlung.
        RTS
ZIEL:
        DS.B        8               * Datenablage für READ CAPACITY.
```

*
**

Nummer des Befehls : **21**
Befehlsname : **Read Usage Counter (SB).**
Kurzbeschreibung : Fehler- und Statistik-Zähler lesen.
Eingaberegister : A0.L = Zieladresse für Daten.

Mit dem Befehl kann man verschiedene interne Zähler eines Laufwerkes lesen. Es werden Statistiken über die Anzahl der aufgetretenen Fehler, die Anzahl gelesener Sektoren usw. übergeben. Die Daten sind 9 Bytes lang und werden ab der Adresse abgelegt, die in A0.L angegeben wird. Wenn die Zähler ausgelesen sind, werden sie zurückgesetzt. Bei überlaufenden Zählern erfolgt eine Fehlermeldung beim jeweiligen Befehl (siehe Beschreibung SCSI-Schnittstelle).

Einfaches Beispiel:

Die internen Fehler-Zähler werden gelesen.

```
START:
        MOVEQ       #21,D1          * Befehlskennung.
        MOVEQ       #1,D4           * Laufwerk 0.
        LEA         ZIEL(PC),A0     * Ziel für Daten.
        MOVE        #!HARDDISK,D7   * Befehl ausführen.
        TRAP        #1
        ....                        * Fehlerbehandlung.
        RTS
ZIEL:                               * Ziel für Datenablage.
        DS.B        3               * Bisher belesene Blöcke.
        DS.B        3               * Bisher durchgeführte Suchvorgänge.
        DS.B        1               * Unkorrigierte Lesefehler.
        DS.B        1               * Korrigierte Fehler.
        DS.B        1               * Suchfehler.
```

*
**

Nummer des Befehls : **22**
Befehlsname : **Read Capacity (SB).**
Kurzbeschreibung : Größe des Laufwerks lesen.
Eingaberegister : D2.L = Sektornummer.
D3.B = PMI-Bit.
A0.L = Zieladresse für Daten.

Mit diesem Befehl kann man die Größe einer Festplatte feststellen. Das ist für jedes Betriebssystem notwendig, damit es weiß, wieviele Sektoren zur Verfügung stehen. In D3.B wird ausgewählt, ob die Sektoranzahl des gesamten Laufwerks ausgegeben werden soll oder der letzte Sektor eines bestimmten Tracks. Enthält D3.B den Wert 0, muß D2.L auf Null gesetzt sein. Es wird die Nummer des letzten Sektors des Laufwerks innerhalb der Daten zurückgeliefert. Enthält D3.B den Wert 1, wird bei den Daten der letzte Sektor des Tracks zurückgeliefert, in dem sich der Sektor befindet, der in D2.L angegeben wurde. Die Daten sind 8 Bytes lang und werden ab der in A0.L angegebenen Adresse abgelegt.

Einfaches Beispiel:

Die Größe der Festplatte und die Sektorenlänge werden ermittelt.

```
START:
        MOVEQ       #0,D2           * Keine Sektorenangabe.
        MOVEQ       #0,D3           * Nummer des letzten Sektors holen.
        MOVEQ       #22,D1          * Kennung für READ CAPACITY.
        MOVEQ       #1,D4           * Laufwerk 0.
        LEA         ZIEL(PC),A0     * Ziel für Daten.
        MOVE        #!HARDDISK,D7   * Befehle ausführen.
        TRAP        #1
        ....                        * Fehlerbehandlung.
        RTS
ZIEL:
        DS.L        1               * Ziel für größte Sektornummer.
        DS.L        1               * Ziel für Blocklänge.
```

Einfaches Beispiel:

Die Anzahl der Sektoren pro Spur und die Sektorenlänge werden ermittelt.

```
START:
        MOVEQ       #0,D2           * Erste Spur, deshalb Sektor 0.
        MOVEQ       #1,D3           * Nummer des letzten Sektors der Spur holen.
        MOVEQ       #22,D1          * Kennung für READ CAPACITY.
        MOVEQ       #1,D4           * Laufwerk 0.
        LEA         ZIEL(PC),A0     * Ziel für Daten.
        MOVE        #!HARDDISK,D7   * Befehle ausführen.
        TRAP        #1
        ....                        * Fehlerbehandlung.
        RTS
ZIEL:
        DS.L        1               * Ziel für letzten Sektor der ersten Spur.
        DS.L        1               * Ziel für Blocklänge.
```

*
**

Nummer des Befehls : **23**
Befehlsname : **Receive Diagnostic Results.**
Kurzbeschreibung : Ergebnis eines Selbsttest lesen.
Eingaberegister : A0.L = Zieladresse für Daten.

Hiermit kann man das Ergebnis des letzten Selbsttest auslesen. Es werden die gelieferten Daten ab der Adresse in A0.L abgelegt.

Einfaches Beispiel:

Das Ergebnis eines Selbsttests wird zurückgeliefert.

```
START:
        LEA     ZIEL(PC),A0     * Ziel für Daten.
        MOVEQ   #23,D1          * Befehlskennung.
        MOVEQ   #1,D4           * Laufwerk 0.
        MOVE    #!HARDDISK,D7   * Befehle ausführen.
        TRAP    #1
        ....                    * Fehlerbehandlung.
        RTS
ZIEL:                           * Ziel für Daten.
        DS.B    1               * ADVAL-Bit (Bit 7) / Error-Class / Error-Code.
        DS.B    3               * Logical Block Adresse (LBA).
```

*
**

Nummer des Befehls : **24**
Befehlsname : **Inquiry (SB).**
Kurzbeschreibung : Daten des Laufwerks lesen (Name / Versionsnummer ...).
Eingaberegister : A0.L = Zieladresse für Daten.

Manchmal muß ein Programm wissen, mit welchem Laufwerk es arbeitet, damit es eventuelle Spezial-Befehle oder bestimmte Ausnahmen berücksichtigen kann. Dafür ist der INQUIRY-Befehl gedacht. Er liefert den Namen der Herstellerfirma, den Namen der Festplatte und einige andere Daten zurück. Diese werden ab der in A0.L angegebenen Adresse abgelegt.

Einfaches Beispiel:

Die Daten des Laufwerks 0 werden gelesen.

```
START:
        LEA     ZIEL(PC),A0     * Ziel für Daten.
        MOVEQ   #24,D1          * Befehlskennung.
        MOVEQ   #1,D4           * Laufwerk 0.
        MOVE    #!HARDDISK,D7   * Befehl ausführen.
        TRAP    #1
        ....                    * Fehlerbehandlung.
        RTS
ZIEL:                           * Ziel für Daten.
        DS.B    1               * Device Type Code (0).
        DS.B    1               * Removable Medium Bit / Device Type Qualifier.
        DS.B    1               * Revisions Level.
        DS.B    1               * Response Data Format (1).
        DS.B    1               * Länge der weiteren Daten ($3D).
        DS.B    3               * Reserviert.
        DS.B    8               * Name in ASCII-Zeichen.
        DS.B    16              * Seriennummer in ASCII-Zeichen.
        DS.B    1               * Hardware Revisions Level.
        DS.B    1               * Firmware Revisions Level.
        DS.B    1               * ROM Revisions Level.
        DS.B    1               * Reserviert.
```

*
**

Nummer des Befehls : **25**
Befehlsname : **Read Defect Data (SB).**
Kurzbeschreibung : Liste der defekten Sektoren lesen.
Eingaberegister : D2.B = P-Bit / G-Bit.
D3.W = Maximale Anzahl von Bytes.
A0.L = Zieladresse für Daten.

Für jedes Laufwerk gibt es eine interne Liste aller defekter Sektoren. Die Liste enthält die fehlerhaften Sektoren, die bereits bei der Herstellung des Laufwerks entstanden sind, sowie derjenigen Sektoren, die später bei der Benutzung des Laufwerks fehlerhaft geworden sind. D2.B gibt an, welche Sektoren man haben möchte. Bit 1 ist das P-Bit und Bit 0 das G-Bit. Die Bedeutung der beiden Bits steht in den Beschreibungen der SCSI-Harddisks. Sie geben an, ob nur die vom Werk entdeckten defekten Sektoren oder alle defekten Sektoren in der Liste abgelegt werden sollen. In D3.W wird die Anzahl der maximal zurückgeschickten Daten zurückgegeben. Ist die Liste länger, wird die Übertragung abgebrochen. Die Zieladresse für die Liste der defekten Sektoren wird in A0.L angegeben.

*
**

Nummer des Befehls : **26**
Befehlsname : **Reassign Blocks (SB).**
Kurzbeschreibung : Neue Liste defekter Blöcke anlegen.
Eingaberegister : A0.L = Quelladresse der Daten.

Da im Laufe der Zeit Sektoren fehlerhaft werden können, ist es sinnvoll, fehlerhafte Sektoren in einer Liste aufzunehmen, die auf der Festplatte abgespeichert wird. Die Liste wird von A0.L an zur Festplatte gesandt. Das Aussehen der Liste steht in einer SCSI-Beschreibung.

*
**

Nummer des Befehls : **27**
Befehlsname : **Request Sense (SB).**
Kurzbeschreibung : Informationen über letzten Fehler lesen.
Eingaberegister : A0.L = Zieladresse der Daten.

Es werden Informationen über den letzten aufgetretenen Fehler zurückgegeben. Dabei wird nur das EXTENDED-SENSE-Format unterstützt, da maximal 27 Bytes ab der in A0.L angegebenen Adresse abgelegt werden.

Einfaches Beispiel:

Der letzte aufgetretene Fehler wird spezifiziert.

```
START:
        LEA     ZIEL(PC),A0         * Ziel für Daten.
        MOVEQ   #27,D1              * Befehlskennung.
        MOVEQ   #1,D4               * Laufwerk 0.
        MOVE    #!HARDDISK,D7       * Befehl ausführen.
        TRAP    #1
        ....                        * Fehlerbehandlung.
        RTS
ZIEL:
        DS.B    27                  * Maximal 27 Bytes.
```

*
**

Nummer des Befehls : **28**
Befehlsname : **Format Unit (SB).**
Kurzbeschreibung : Laufwerk formatieren.
Eingaberegister : D2.B = FMT-Data/ CMP-List / Defect-List-Format.
D3.W = Interleave.
A0.L = Quelladresse der Daten, wenn verlangt.

Eine Festplatte kann genau wie jede Diskette neu formatiert werden. Dafür ist dieser Befehl gedacht. Man muß dabei nicht wie bei einer Diskette jedes Byte einzeln übertragen, sondern nur bestimmte Daten. Das Formatieren erfolgt dann ganz selbständig. In D2.B müssen Angaben über die Verwaltung defekter Blöcke gemacht werden. Bit 4 ist das FMT-Bit, Bit 3 das CMP-LST-Bit. Die Bits 2 - 0 geben das Format der Liste der Defekte an. Die Bedeutung dieser Bits kann weiterführenden SCSI-Beschreibungen entnommen werden. In D3.W wird der INTERLEAVE-Faktor angegeben und in A0.L die Quelladresse der Liste der Defekte, falls eine übertragen werden soll.

Einfaches Beispiel:

Die Festplatte wird im jetzigen Format neu formatiert, d.h. die Sektorenlänge oder die Kennung defekter Blöcke wird nicht verändert. A0.L muß nicht belegt sein, da keine neuen defekten Blöcke angegeben werden.

```
START:
        MOVEQ     #0,D2          * Liste der defekten Blöcke nicht ändern.
        MOVEQ     #1,D3          * Interleave = 1.
        MOVEQ     #28,D1         * Befehlskennung.
        MOVEQ     #1,D4          * Laufwerk 0.
        MOVE      #!HARDDISK,D7  * Befehl ausführen.
        TRAP      #1
        ....      * Fehlerbehandlung.
        RTS
```

*
**

Nummer des Befehls : **29**
Befehlsname : ———.
Kurzbeschreibung : Eigene Befehle für direkte Benutzung.
Eingaberegister : A0.L = Wenn vorhanden, Quell- oder Zieladresse der Daten.
A1.L = Adresse des Befehls.

Da Laufwerke möglicherweise zusätzliche Befehle ausführen können oder bei einigen Befehlen Abweichungen aufweisen, kann hiermit jeder Befehl ausgeführt werden. In A1.L wird dazu die Adresse übergeben, auf der die Befehlsfolge steht. Werden auch Daten übertragen, muß in A0.L die Quell- oder Zieladresse angegeben werden.

Einfaches Beispiel:

Der Befehl TEST UNIT READY wird nachgebildet. A0.L ist nicht belegt, da keine Daten verlangt werden.

```
START:
        LEA       COMMAND(PC),A1  * Daten für Befehl.
        MOVEQ     #29,D1          * Befehlskennung.
        MOVEQ     #1,D4           * Laufwerk 0.
        MOVE      #!HARDDISK,D7   * Befehl ausführen.
        TRAP      #1
        ....                      * Fehlerbehandlung.
        RTS
COMMAND:
        DC.B      0               * Operation Code.
        DC.B      0               * LUN / Reserviert.
        DC.B      0,0,0           * Reserviert.
        DC.B      0               * Reserviert / Flag / Link.
```

TRAP-Nummer	: **142**
Befehlsname	: **HARDTEST**
Befehlsgruppe	: SCSI-Harddisk.
Kurzbeschreibung	: Testet, ob ein Laufwerk vorhanden ist.
Eingaberegister	: D4.B = Auswahl des Laufwerks.
Ausgaberegister	: D0.L = Fehlercode. Carry = 1, wenn kein Laufwerk vorhanden ist.
Zerstörte Register	: Keine.
Ab Version	: 6.2
Änderungen zu 4.3	: ——
Änderungen zu 6.1	: ——
Siehe auch	: HARDDISK (141)

ACHTUNG !!!

Die direkte Ansprache einer SCSI-Harddisk sollte sich nur der fortgeschrittene Programmierer zutrauen. Er muß dann unbedingt auch eine Beschreibung des Befehlssatzes der Harddisk sowie der zurückgelieferten Fehlercodes besitzen. Alle HARDDISK-Routinen des Grundprogramms funktionieren nur mit SEAGATE-kompatiblen Laufwerken, da die Befehle auf diese abgestimmt sind.

Mit Hilfe dieses Programms kann ein Anwenderprogramm dahin testen, ob eine bestimmte SCSI-Harddisk vorhanden ist, da mit den DIL-Schaltern auf der KEY nur festgelegt wird, daß mindestens eine verfügbar ist. Wird das angesprochene Laufwerk gefunden, wird außerdem noch der Befehl TEST UNIT READY durchgeführt. Der zurückgelieferte Wert in D0.L entspricht deshalb auch der Rückmeldung dieses Befehls. Der Befehl ist beim Programm HARDDISK beschrieben.

Im Register D4.B muß die Laufwerksnummer übergeben werden. Die Nummer ist wie bei der FLOPPY-Routine kodiert, d. h. daß bei Laufwerk 0 Bit 0 gesetzt ist, bei Laufwerk 1 Bit 1 usw.

Wenn eine Harddisk gefunden wird, wird in D0.L der Fehlercode, den die Harddisk liefert, zurückgegeben, außerdem ist das CARRY-Flag zurückgesetzt. Ansonsten ist das CARRY-Flag gesetzt und D0.L enthält den Wert -1.

Einfaches Beispiel:

Das Programm stellt fest, ob die Harddisk 0 vorhanden ist.

```
START:
        MOVEQ       #1,D4               * Harddisk 0.
        MOVE        #!HARDTEST,D7       * Testen.
        TRAP        #1
        BCS.S       ENDE                * Nicht vorhanden (antwortet nicht).
        ....                            * Hier weitere Befehle (Fehlerauswertung usw.).
ENDE:
        RTS                             * Ende.
```

TRAP-Nummer	: **145**		
Befehlsname	: **PRTFP0**		
Befehlsgruppe	: 68020-FPU.		
Kurzbeschreibung	: Wert in FP0 in ASCII wandeln.		
Eingaberegister	: D0.W = Stellenanzahl und Rundungswert. A0.L = Ablageadresse für Zahl. FP0.X = Zu wandelnde Zahl.		
Ausgaberegister	: A0.L = Zeigt auf Endekennung. Carry = 1, wenn keine Zahl.		
Zerstörte Register	: Keine.		
Ab Version	: 6.1 (Nur beim 68020-Grundprogramm)		
Änderungen zu 4.3	: ——		
Änderungen zu 6.1	: Nein.		
Siehe auch	: PRINTFP0 (121)	GETFLOAT (122)	FPUWERT (146)
	SETFPX (147)	SETFPY (148)	SETFPZ (149)

Der Befehl ist fast identisch mit der Funktion 121 (PRINTFP0). Er wurde nur eingeführt, weil er etwas mehr kann und die Funktionen von PRINTFP0 aus Kompatibilitätsgründen nicht verändert werden sollten.

Der Aufruf ist identisch und soll deshalb hier nicht noch einmal erklärt werden. Die Ausgabe aber ist etwas komfortabler. Alle Ende-Nullen werden nämlich nicht mit ausgegeben, sodaß die Zahl so kurz wie möglich wird. Außerdem wird erkannt, wenn es keine Zahl ist, die ausgegeben werden soll. In jedem besseren Handbuch zur FPU 68881 ist beschrieben, daß es einige Formate der Zahlendarstellung gibt, die einen Überlauf darstellen oder den Wert "KEINE ZAHL". Das erkennt die Routine und gibt dann entsprechend der Kennung den Text "KEINE ZAHL" oder "ÜBERLAUF" aus. Das kann man sehr leicht bei der Registeranzeige der FPU-Register beim Einzelschritt sehen, bevor das erste Mal auf ein Register zugegriffen wird.

Die beiden folgenden Beispiele sind die gleichen wie bei PRINTFP0, allerdings kann man bei einem Vergleich die verschiedenen Ausgaben erkennen.

Komplexeres Beispiel:

Es werden 100 FP-Zahlen über CO2 ausgegeben.

```
START:
        MOVEQ     #!CLRSCREEN,D7      * Bildschirm für CO2 vorbereiten.
        TRAP      #1
        MOVEQ     #$1B,D0
        MOVEQ     #!CO2,D7
        TRAP      #1
        MOVEQ     #'2',D0             * HARDSCROLL an, falls GDPHS vorhanden.
        MOVEQ     #!CO2,D7
        TRAP      #1
        FMOVE.X   #1.12345678,FP0     * Erste Zahl festlegen.
        MOVEQ     #100-1,D3           * 100 Durchläufe.
SCHLEIF0:
        LEA       BUFFER(PC),A0       * Ziel für Zahl.
        MOVEQ     #17,D0              * Alle Stellen.
        MOVE      #!PRTFP0,D7         * Zahl in ASCII-Darstellung bringen.
        TRAP      #1
        FADD.W    D3,FP0              * Addieren.
        FDIV.W    #10,FP0             * Irgend eine neue Zahl.
        LEA       BUFFER(PC),A0       * Dort steht Zahl.
SCHLEIF1:
        MOVE.B    (A0)+,D0            * Bis Endekennung erscheint.
        BEQ.S     SPRUNG0
        MOVEQ     #!CO2,D7            * Über CO2 ausgeben.
        TRAP      #1
        BRA.S     SCHLEIF1
SPRUNG0:
        MOVEQ     #!CRLF,D7           * Zeilenvorschub.
        TRAP      #1
        DBRA      D3,SCHLEIF0         * Wiederholen.
        RTS
BUFFER:                               * Ziel für ASCII-Zeichen.
        DS.B      24
```

TRAP-Nummer	: **146**		
Befehlsname	: **FPUWERT**		
Befehlsgruppe	: 68020-FPU.		
Kurzbeschreibung	: FP-Zahl berechnen und nach FP0.		
Eingaberegister	: A0.L = Adresse des arithmetischen Ausdrucks.		
Ausgaberegister	: D1.W = Fehlerkennung.		
	A0.L = Zeigt auf Ende oder Fehler.		
	FP0.X = Berechneter Wert.		
	Carry = 1, wenn Fehler aufgetreten ist.		
Zerstörte Register	: Keine.		
Ab Version	: 6.1 (Nur beim 68020-Grundprogramm).		
Änderungen zu 4.3	: ——		
Änderungen zu 6.1	: Es werden auch Werte aus der Symboltabelle verarbeitet.		
Siehe auch	: PRTFP0 (121)	GETFLOAT (122)	PRTFP0 (145)
	SETFPX (147)	SETFPY (148)	SETFPZ (149)

Das ist wieder ein sehr mächtiger Befehl. Er ermöglicht die komfortable Auswertung von arithmetischen Ausdrücken, wobei Integer- und Floating-Point-Zahlen verarbeitet werden können. Die Funktion ist ähnlich der des WERT-Befehls.

In A0.L wird die Adresse eines Textes übergeben, der einen arithmetischen Ausdruck in ASCII-Zeichen enthält. Der Ausdruck muß mit einer binären Null ($00) oder einem Leerzeichen abgeschlossen sein. Der Ausdruck selber darf alle Ziffern von 0 bis 9 sowie alle einfachen Operatoren wie +, -, *, /, \ und ^ enthalten. Außerdem sind Klammerausdrücke erlaubt. Die Schachtelungstiefe ist beliebig. Sie hängt nur von der maximalen Stackgröße ab. Weiterhin sind alle folgenden Funktionen erlaubt, wobei das Argument immer in Klammern stehen muß: ABS, ACOS, ASIN, ATAN, COS, COSH, INT, LG, LN, LOG, SIN, SINH, SQR, TAN, TANH.

Zusätzlich zu den Zahlen gibt es drei interne Variable, die mit X, Y, und Z angesprochen werden. Sie sind Floating-Point-Zahlen, die direkt in die arithmetischen Ausdrücke eingebaut werden können. Sie können mit den Befehlen SETFPX, SETFPY und SETFPZ verändert werden, wodurch leicht in Programmen Funktionen berechnet werden können. Sie wurden eingeführt, da die Variablen der Symboltabelle nur Integer-Werte annehmen können und deshalb nicht sehr brauchbar sind. Variablen der Symboltabelle können aber, wie beim WERT-Befehl auch mit eingebaut werden. Sie werden dann als LANGWORT-INTEGER-Zahlen berechnet. Wenn in der Symboltabelle die Symbole X, Y und Z auftreten, können diese nicht erreicht werden, da die drei internen Variablen Vorrang haben.

Beispiel arithmetische Ausdrücke:

- SIN(X)^2+COS(X)^2
- TANH(1/4)
- 1+2*4-7*(5-3)^7+LN(LN(LN(X+Y+Z)))

Leerzeichen zwischen den Zahlen sind nicht erlaubt !!!

Das Ergebnis der Auswertung wird in FP0.X zurückgegeben, wobei der Wert bei einem Fehler natürlich nicht zu gebrauchen ist. Außerdem liefert D1.L wie bei der WERT-Routine eine Fehlermeldung. Wenn D1.L ungleich Null ist, wird das CARRY-Flag gesetzt, sonst ist es Null. A0.L zeigt nach dem Aufruf entweder auf die Endekennung oder auf den Fehler.

Werte in D1.L:

0 = Syntaxfehler (z.B 5**7 oder 5-*3).
1 = Kein Fehler.
4 = Überlauf (Wert wurde durch eine Berechnung zu groß).
5 = Falsche Variable aus Symboltabelle oder nicht definierte Funktion benutzt.

Einfaches Beispiel:

Es wird der Wert des Ausdrucks SQR(25) berechnet. Das Ergebnis kann beim Einzelschritt im Register FP0.X überprüft werden.

```
START:
        LEA     WERT(PC),A0     * Adresse des arithmetischen Ausdrucks.
        MOVE    #!FPUWERT,D7    * Wert berechnen und in FP0.X ablegen.
        TRAP    #1
        RTS                     * Ergebnis in FP0.X
WERT:
        DC.B    'SQR(25)',0     * Arithmetischer Ausdruck.
```

Komplexeres Beispiel:

Es wird die Funktion Y = 127 * e^(-x/30) * sin(x) + 127 auf dem Bildschirm im Intervall [0,102.2] ausgegeben.

```
START:
        FMOVECR #$0F,FP1        * 0 ins Register FP1.X laden.
        MOVEQ   #0,D3           * Schleifenzähler.
SCHLEIFE:
        FMOVE.X FP1,FP0         * X Register FP0.X
        MOVE    #!SETFPX,D7     * Variable X setzen.
        TRAP    #1
        LEA     FUNKTION(PC),A0 * Funktionstext.
        MOVE    #!FPUWERT,D7    * Wert berechnen.
        TRAP    #1
        FMOVE.L FP0,D2          * Ins Register D2 für Bildschirmausgabe.
        MOVE    D3,D1           * X-Position auf dem Bildschirm.
        MOVEQ   #!MOVETO,D7     * Position einstellen.
        TRAP    #1
        MOVEQ   #127,D2         * Neue Y-Position.
        MOVEQ   #!DRAWTO,D7     * Linie ziehen, dadurch Fläche sichtbar.
        TRAP    #1
        FADD.X  #0.2,FP1        * X erhöhen.
        ADDQ    #1,D3           * Einen Punkt auf dem Screen nach rechts.
        CMP     #512,D3         * Bis ganz rechts.
        BNE.S   SCHLEIFE
        RTS
FUNKTION:                       * Hier steht Funktionstext.
        DC.B    '127*exp(-x/30)*cos(x)+127',0
```

TRAP-Nummer	: **147**		
Befehlsname	: **SETFPX**		
Befehlsgruppe	: 68020-FPU.		
Kurzbeschreibung	: FP-Variable X setzen.		
Eingaberegister	: FP0.X = Wert für Variable.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 6.1 (Nur beim 68020-Grundprogramm).		
Änderungen zu 4.3	: ——		
Änderungen zu 6.1	: Nein.		
Siehe auch	: PRTFP0 (121)	GETFLOAT (122)	PRTFP0 (145)
	FPUWERT (146)	SETFPY (148)	SETFPZ (149)

Mit dieser Funktion kann die interne Variable X der Routine FPUWERT mit einem beliebigen Wert belegt werden. Jeder Zugriff auf X bei FPUWERT gibt dann diesen Wert zurück. Übergeben wird die Zahl in FP0.X.

Einfaches Beispiel:

X wird mit dem Wert 0 belegt.

```
START:
        FMOVECR     #$0F,FP0        * 0 aus internem ROM der FPU nehmen.
        MOVE        #!SETFPX,D7     * Wert setzen.
        TRAP        #1
        RTS
```

TRAP-Nummer	: **148**		
Befehlsname	: **SETFPY**		
Befehlsgruppe	: 68020-FPU.		
Kurzbeschreibung	: FP-Variable Y setzen.		
Eingaberegister	: FP0.X = Wert für Variable.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 6.1 (Nur beim 68020-Grundprogramm).		
Änderungen zu 4.3	: ——		
Änderungen zu 6.1	: Nein.		
Siehe auch	: PRTFP0 (121)	GETFLOAT (122)	PRTFP0 (145)
	FPUWERT (146)	SETFPX (148)	SETFPZ (149)

Mit dieser Funktion kann die interne Variable Y der Routine FPUWERT mit einem beliebigen Wert belegt werden. Jeder Zugriff auf Y bei FPUWERT gibt dann diesen Wert zurück. Übergeben wird die Zahl in FP0.X.

Einfaches Beispiel:

Y wird mit dem Wert Pi belegt.

```
START:
        FMOVECR     #$00,FP0        * Pi aus internem ROM der FPU nehmen.
        MOVE        #!SETFPY,D7     * Wert setzen.
        TRAP        #1
        RTS
```

TRAP-Nummer	: 149		
Befehlsname	: **SETFPZ**		
Befehlsgruppe	: 68020-FPU.		
Kurzbeschreibung	: FP-Variable Z setzen.		
Eingaberegister	: FP0.X = Wert für Variable.		
Ausgaberegister	: Keine.		
Zerstörte Register	: Keine.		
Ab Version	: 6.1 (Nur beim 68020-Grundprogramm).		
Änderungen zu 4.3	: ——		
Änderungen zu 6.1	: Nein.		
Siehe auch	: PRTFP0 (121)	GETFLOAT (122)	PRTFP0 (145)
	FPUWERT (146)	SETFPX (148)	SETFPY (149)

Mit dieser Funktion kann die interne Variable Z der Routine FPUWERT mit einem beliebigen Wert belegt werden. Jeder Zugriff auf Z bei FPUWERT gibt dann diesen Wert zurück. Übergeben wird die Zahl in FP0.X.

Einfaches Beispiel:

Z wird mit dem Wert e belegt.

```
START:
        FMOVECR     #$0C,FP0        * e aus internem ROM der FPU nehmen.
        MOVE        #!SETFPZ,D7     * Wert setzen.
        TRAP        #1
        RTS
```

Kapitel 5 - Ergänzende Informationen

Wer die vorangegangenen Kapitel sorgfältig durchgelesen hat, kann mit dem Grundprogramm schon wirklich gut arbeiten. Die notwendigen Informationen zum System sollten jetzt eigentlich bekannt sein.

Dieses Kapitel ist für diejenigen Benutzer und Programmierer gedacht, die sich intensiver mit dem NKC beschäftigen möchten und dabei nicht auf die Routinen des Grundprogramms verzichten wollen. Es soll hier also ein tieferer Einblick in das System gewährt werden.

Da es sich beim NKC um ein offenes System handelt, d.h. daß alle Informationen über Hardware und Software erhältlich sind, wird natürlich auch ein Listing des Grundprogramms auf Diskette oder als Ausdruck vertrieben. Es empfiehlt sich deshalb für den fortgeschrittenen Programmierer, sich ein Listing zu beschaffen, um die folgenden Informationen verarbeiten zu können. Es gibt nämlich noch eine Fülle von Einzelheiten, die bei der Programmierung helfen können und bisher noch nicht besprochen worden sind.

5.1. Allgemeiner Aufbau und Funktion

Das Grundprogramm besteht aus drei Teilen, die auch im Speicher getrennt voneinander gehalten werden müssen. Auch im Listing sind die drei Teile getrennt.

Der erste Teil ist der Bereich der Exceptions. Er liegt zwar im Eprom, wird aber nach dem Start und bei jedem RESET auf die Adresse $0 geschrieben, falls das Grundprogramm nicht auf dieser Adresse liegt. Der Bereich der Exceptions ist 1 Kbyte lang und liegt auf den Adressen $0 bis $3ff einschließlich. Dieser Bereich muß vorhanden sein, da die CPU sich von dort die Adressen holt, falls Fehler beim Betrieb auftreten, ein Interrupt ausgelöst wird oder ein TRAP-Aufruf erfolgt. Weil sich der Bereich im RAM befindet, können die Adressen vom Benutzer verändert werden.

Der zweite Bereich ist der Programmbereich. Er erstreckt sich von Adresse $400 bis maximal Adresse $ffff. Dort stehen alle Programme, Texte und Daten, die für den Betrieb des Grundprogramms nötig sind. Dabei haben die ersten Adressen eine besondere Bedeutung:

Adresse	Länge des Wertes	Bedeutung
$400	1 Langwort	Kennung, daß das Grundprogramm vorhanden ist. Dort steht der Wert $5aa58001.
$404	1 Langwort	Dort ist der Beginn des Variablenbereichs relativ zum Anfang der Eproms angegeben (normalerweise $10000).
$408	1 Langwort	Adresse des Kaltstarts relativ zum Anfang der Eproms.
$40C	1 Langwort	Versionsnummer (Version 6.21 = $621).
$410	1 Langwort	Seriennummer (nicht benutzt).
$414	1 Langwort	CPU-Kennung (1 = 68008, 2 = 68000, 4 = 68020).
$418	1 Langwort	Anzahl der Worte des Grundprogramms vom Beginn der Eproms an.
$41C	1 Langwort	Checksumme. Diese wird wortweise gebildet (Anzahl aus Adresse $418) vom Anfang der Eproms an. Der Wert der Checksumme selber ist nicht enthalten.
$420	2 Worte	Sprung auf TRAP-Mechanismus mit RTS-Abschluß.
$424	2 Worte	Sprung auf Kaltstart.
$428	2 Worte	Sprung auf Init aller Variablen. Stack wird aber nicht eingerichtet. In d0 wird Stack zurückgeliefert.
$42C	2 Worte	Sprung auf Menüoberfläche.

Der dritte Bereich sind die Variablen, die normalerweise direkt hinter den Grundprogramm-Eproms liegen. Die Adressen der einzelnen Variablen können dem Listing entnommen werden.

5.2. Variablen

Zu den Variablen ist eigentlich nicht viel zu sagen. Sie liegen immer direkt hinter dem Grundprogramm, es sei denn, der Wert der Adresse RAMSTART ($404) im ROM wird geändert.

Von Bedeutung ist eigentlich nur, daß der Variablenaufbau für die wichtigsten Variablen immer gleich ist und daß die Länge des Variablenbereichs nicht fest vorgegeben ist.

Die Länge des Variablenbereichs hängt von der Länge der Symboltabelle ab, die direkt hinter der letzten Variablen beginnt. Die Symboltabelle kann aber nur maximal 64 Kbyte lang werden.

Die Adressen der einzelnen Variablen und deren Funktion kann man dem Listing des Grundprogramms entnehmen.

5.3. Exceptions und Interrupts

Im Prinzip sind Exceptions und Interrupts von der Ausführung her das Gleiche. Der Unterschied besteht nur darin, daß Interrupts meist in zeitlich gleichen Abständen ausgelöst werden, um z.B. eine Uhr zu simulieren oder um etwas zu steuern. Die anderen Exceptions treten in unregelmäßigen Abständen gewollt oder ungewollt auf.

Was die Interrupts angeht, soll hier nicht näher darauf eingegangen werden, was sie genau sind und was man mit ihnen machen kann. Hier soll nur der grundsätzliche Ablauf der Interruptverabeitung im Grundprogramm erklärt werden.

Die Interruptbehandlung wird nicht vom Grundprogramm selbst durchgeführt. Dadurch bleiben alle Varianten des Interrupts erhalten, die die Prozessoren 680xx zur Verfügung stellen. Das Grundprogramm stellt lediglich feste Adressen bereit, zu denen der Prozessor "springt" oder, besser gesagt, von denen aus die Programmausführung beginnt, wenn ein Interrupt aufgetreten ist. Diese Adressen sind immer gleich. Sie liegen im RAM direkt hinter dem Grundprogramm.

Für den Prozessor 68008 stehen dabei nur die Interruptebenen 2, 5 und 7 zur Verfügung, was nicht am Grundprogramm liegt, sondern am Aufbau der 68008-CPU.

Jede Interruptadresse hat eine Länge von 6 Bytes. Das ist genau der Platz für einen JMP-Befehl ($4EF9) und eine Adresse, nämlich der, von der aus das Programm fortgesetzt werden soll.

Bei jedem Start des Grundprogramms stehen dort Sprünge auf eine interne Grundprogrammroutine, damit unerwünscht auftretende Interrupts abgefangen werden können. Die interne Routine zeigt auf dem Bildschirm an, daß ein Interrupt aufgetreten ist, und führt dann in das Grundprogramm zurück. Die Interruptadresse steht im Listing des Grundprogramms.

Alle weiteren Exceptions, die sonst noch auftreten können, wie BUSERROR, ADRESSERROR o. ä. werden ebenfalls auf diese Routine geführt und am Bildschirm angezeigt. Dann erscheint oben auf dem Bildschirm die Information über die Art der Exception. Unten auf dem Bildschirm werden die Inhalte der Register der CPU ausgegeben, die bei der CPU68020 wie beim EINZELSCHRITT umgeschaltet werden können. Mit einem Tastendruck M wird das Grundprogramm neu gestartet, wobei die Variablen neu initialisiert werden. Mit W kann man auch in das Grundprogramm gelangen. Dabei wird allerdings nur der Stack neu eingerichtet, während alle anderen Variablen und Sprungadressen unverändert bleiben. Diese Funktion kann bei harmloseren Exceptions wie CHK, DIVISION DURCH NULL usw. verwendet werden, um z. B. eine Umlenkung der CO2-Routine nicht zurücksetzen zu müssen.

Wenn das Grundprogramm nicht auf der Adresse $0 liegt, stehen diese ganzen Adressen im RAM von der Adresse $0 an, sodaß auch sie leicht geändert werden können.

Die letzte Gruppe von Exceptions sind vom Benutzer gewollte oder im Programm angegebene Exceptions. Das sind die TRAP-Aufrufe. Alle TRAP-Vektoren sind in das RAM hinter das Grundprogramm geführt und werden von dort weiterverzweigt. Die Adressen der TRAPs sind dem Listing zu entnehmen.

Die TRAPs 1 (Grundprogramm) und 6 (Jados) sind übrigens reserviert.

Eine Besonderheit gibt es beim TRAP 1. Es stehen jetzt alle Adressen der TRAP-Unterprogramme auf den Adressen der NON-AUTO-Vektoren, die bei der Adresse $100 beginnen. Wenn das Grundprogramm nicht auf der Adresse $0 liegt, werden diese Adressen ins RAM kopiert und die richtige Adresse wird eingestellt. Jetzt können gezielt einzelne Unterprogramme umgelenkt werden, ohne daß der TRAP umgelenkt werden muß. Es genügt, einfach auf die entsprechende Adresse eine neue Sprungadresse zu schreiben. Der TRAP-Mechanismus holt sich aus dem RAM die Adresse und springt zu dem entsprechenden Unterprogramm, weil die Adresse der Routine dann bereits im RAM steht. Beim Initialisieren wird nämlich die Basisadresse des Grundprogramms schon berücksichtigt.

Wenn im übrigen eine Sprungadresse umgelenkt wird, wird nur beim TRAP die neue Routine angesprungen, d.h es wird bei der WERT-Routine oder beim Assembler beim Ausdruck @NAME die alte im Grundprogramm eingebaute Adresse zurückgegeben, weshalb bei der Verwendung einer umgelenkten TRAP #1-Funktion keine Probleme auftreten sollten.

Damit auch für den Anwender noch NON-AUTO-Vektoren zur Verfügung stehen, sind die Vektoren 180 bis 192 für Anwendungen frei gehalten. Für das Grundprogramm gibt es aber immer noch 180 NON-AUTO-Vektoren.

Die Vektoren von Adressen $0 bis $400 werden bei jedem START oder RESET neu initialisiert. Die hinter dem Grundprogramm stehenden Vektoren werden nur dann initialisiert, wenn keine POWER-ON-Kennung vorhanden ist. Dadurch bleiben umgelenkte Routinen auch nach einem RESET unverändert.

5.4. Symboltabelle

Weil der Aufbau der Symboltabelle sehr wichtig für Programme ist, die größere Datenmengen sortieren oder verwalten oder auf Daten direkt zugreifen, soll hier ein wenig näher auf den Aufbau der Tabelle eingegangen werden.

Für die Symboltabelle sind im Grundprogramm zwei Variablen vorgesehen. Die erste heißt SYMNEXT. Sie gibt eigentlich nur an, wie lang die Symboltabelle ist und auf welcher Adresse relativ zum Symboltabellenanfang der nächste Wert abgelegt werden soll. Die Variable ist eine 16 Bit Zahl ohne Vorzeichen. Demgemäß steht ein Symboltabellenbereich von 64 Kbyte oder 3640 Einträgen zur Verfügung. Zur Berechnung der nächsten Adresse darf die Variable nicht mit Befehlen wie EXT.L o. ä. auf eine Langwort-Größe gebracht werden, da dort ein Vorzeichen, das eigentlich gar nicht vorhanden ist, mit berücksichtigt wird. Der Wert von SYMNEXT kann übrigens sehr schnell mit den Befehlen GETNEXT gelesen und PUTNEXT gesetzt werden.

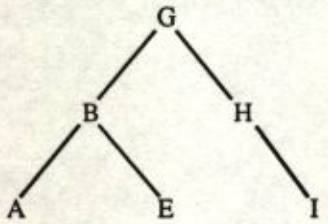
Die zweite Variable ist genau genommen keine Variable, sondern die Anfangsadresse der Tabelle. Auf dieser Adresse liegt der erste Eintrag. Der Anfang der Tabelle kann mit der Funktion GETSYM ermittelt werden. Mit jedem Eintrag wird die Symboltabelle 18 Bytes länger, da jeder Eintrag im Grundprogramm immer 18 Bytes lang ist. Bei eigenen Routinen können die Einträge auch andere Längen haben. Der nächste Eintrag wird nämlich immer relativ zum Symboltabellenanfang angegeben. Wenn eine andere Länge für die Einträge gewählt wird, dürfen aber nicht immer die internen Programmroutinen des Grundprogramms verwendet werden.

Nun zum Aufbau der Symboltabelle. Jeder Eintrag in der Tabelle enthält folgende Angaben:

Name	Länge	Funktion
KLEINER	1 Wort	Zeiger auf einen Eintrag mit einem Namen, der im Alphabet weiter vorne steht.
GROESSER	1 Wort	Zeiger auf einen Eintrag mit einem Namen, der im Alphabet weiter hinten steht.
DATENWERT	1 Langwort	Wert der Variablen.
ATTRIBUT	1 Wort	Dieses Wort enthält nähere Informationen über das Symbol: 0 = Wert ist ungültig. Der Ausdruck, für den der Wert berechnet werden sollte, enthielt einen Syntax-Fehler. 1 = Symbol hat eine Breite von einem Byte. 2 = Symbol hat eine Breite von einem Wort. 3 = Symbol hat eine Breite von einem Langwort. 4 = Reserviert. 5 = Undefiniertes Symbol.
NAME	8 Bytes	Name, der als erstes einen Buchstaben enthalten muß und dann Buchstaben und Ziffern. Außerdem sind die deutschen Sonderzeichen und das Zeichen _ erlaubt. Die Zeichen sind Großbuchstaben.

Durch die Zeigerstruktur wird ein geordneter binärer Baum aufgebaut, der weder ausgeglichen ist noch irgendwelchen Kriterien entspricht, die bei sehr schnellen Suchroutinen erfüllt sein müssen. Ein Suchvorgang ist aber trotzdem schneller als bei einer einfachen Liste, da immer nur ein Ast des Baumes durchsucht werden muß. Bei ungünstigen Daten (z.B. sind alle Symbole vor der Definition schon sortiert) entartet der Baum aber zu einer Liste und das Suchen dauert sehr viel länger. Hat ein Knoten des Baumes keinen Nachfolger, sind beide Zeiger auf Null gesetzt.

Beispiel für die Struktur der Symboltabelle:



Wenn man von oben ausgeht, ist der Name, der links unter einem Knoten steht, immer kleiner als der rechte Name.

Wenn man sich an diese Vorgaben hält, bleibt die Symboltabelle auch bei direkten Manipulationen kompatibel zum Grundprogramm. Man kann dann sehr schnell in eigenen Programmen direkt darauf zugreifen.

5.5. Aufbau des Bildschirmspeichers

Ein weiterer wichtiger Punkt ist der Bildschirmspeicher. Es gibt zwar viele Routinen, mit denen man auf den Speicher zugreifen kann und mit denen man auch die Adressen der Zeilen und Zeichen erfahren kann, aber manchmal möchte man auf den Speicher selbst zugreifen. Für diese speziellen Anwendungen ist dieses Kapitel gedacht.

Der Bildschirmspeicher beginnt immer auf der Adresse $259 relativ zum Anfang des Variablenspeichers. Dort sind die Daten aber nur nach einem CLRSCREEN-Kommando hintereinander abgelegt. Ansonsten bildet immer nur jede Zeile eine Einheit. Für jede Zeile sind genau 80 Zeichen reserviert. Um die Zeilen zu verwalten, gibt es zwei Tabellen.

Die Tabelle LINECNT ($229) enthält nacheinander die maximale Anzahl der Zeichen, die in einer Zeile vorhanden sind. Dabei werden Endeleerzeichen auch mitgezählt. Die Werte in der Tabelle sind jeweils ein Byte lang. Der erste Wert gibt also an, wie lang die erste Zeile ist, die ganz oben steht. Der zweite Wert enthält die Länge der zweiten Zeile usw.

Die zweite Tabelle LINEPTR ($241) sagt etwas über die Zeilenanordnung aus. Wenn z.B. eine Zeile eingefügt wird, wird nicht der ganze Bildschirmspeicher verschoben, sondern nur der Zeiger auf die jeweilige Zeile.

Der erste Eintrag in dieser Tabelle (jeweils Byte-Wert) sagt aus, wo sich im SCREEN-Bereich die erste Zeile befindet. Der zweite Wert zeigt auf die zweite Zeile usw. Dabei muß der Byte-Wert noch mit 80 multipliziert werden.

Diesen beiden Tabellen kann man eigentlich alle Informationen über den Bildschirmspeicher entnehmen. Im Grundprogramm befindet sich die Routine GETLINE, die die nötigen Informationen liefert. Falls man nicht immer selbst rechnen möchte, kann man mit ihr schnell auf den Speicher zugreifen.

Jetzt sind eigentlich nur noch zwei Variablen wichtig. Es sind dies CURX und CURY, die auf den Adressen $223 und $224 liegen. Dort ist die Position des Cursors und damit die aktuelle Zeichenausgabestelle angegeben. Die Werte entsprechen den Ein- und Ausgaben der Routinen GETCURXY und SETCURXY.

Für die Cursorsteuerung ist die Variable CURON ($222) verantwortlich. Wenn sie den Wert 1 hat, wird der Cursor bei Routinen wie CI automatisch dargestellt (siehe Kapitel 4.2. unter CURON, CUROFF, CURSEIN, CURSAUS).

5.6. Speicherbereiche

In Kapitel 2.5. wurde zwar schon kurz auf die Speicheraufteilung eingegangen, hier sollen trotzdem noch einmal alle Speicherbereiche aufgeführt werden.

1) Auf Adressen $0 - $400 befindet sich die Exceptiontabelle, die im RAM liegt, wenn das Grundprogramm nicht auf Adresse $0 liegt.

2) Der Editorbereich ist frei verschiebbar und benötigt, je nach Textumfang, mehr oder weniger RAM.

3) Hinter dem Editor wird bei jedem Assemblieren eine Tabelle angelegt, die Daten über die verwendeten MACROs des Assemblers enthält. Es sind aber auch einige wenige Daten vorhanden, wenn keine MACROs im Programm definiert sind (Init der Tabelle).

4) Hinter der MACRO-Tabelle wird etwas Speicher für die Bearbeitung der MACROs benötigt, da diese dort noch einmal vom Assembler beim Aufruf abgelegt werden. Je nach Länge der MACROs und der ineinander geschachtelten Aufrufe wird verschieden viel Platz benötigt.

5) Wenn die DEBUG-Funktion eingeschaltet ist, wird eine DEBUG-Tabelle hinter dem Speicher der MACRO-Tabelle angelegt. Diese Tabelle kann sehr viel Platz benötigen.

6) Der Speicher direkt hinter dem Grundprogramm wird für Variablen und die Symboltabelle verwendet (siehe vorstehend 5.2. und 5.4.).

7) Der Stack liegt normalerweise am Ende des ersten durchgehenden Speicherbereichs hinter dem Grundprogramm. Manche Betriebssysteme verlegen ihn aber (z. B. JADOS).

5.7. Bibliotheks-Funktion

Das Grundprogramm enthält eine Bibliotheksfunktion, die ermöglicht, Programme auf verschiedenen Adressen im Speicher oder in Eproms leicht zu finden. Im Menü gibt es dafür die BIBLIOTHEK, die Einträge in RAM oder in Eproms auflistet. Hier soll gezeigt werden, wie ein solcher Eintrag aussehen muß, damit ein Eintrag gefunden werden kann.

Zuerst einmal muß ein Bibliothekseintrag auf einer 1-Kbyte-Grenze beginnen, da sonst zu viel Speicher durchsucht werden müßte. Ein Eintrag muß folgende Form haben:

DC.L	$55AA0180	Kennung für den Eintrag.
DC.B	'Name '	Name des Programms (genau 8 Zeichen).
DC.L	Startadresse	Gibt die Startadresse des Programms an. Die Adresse muß absolut angegeben werden, wenn ein absolutes Programm vorliegt, oder relativ zur Adresse der Kennung (Anfang des Eintrags), wenn das Programm verschiebbar ist.
DC.L	Programmlänge	Hier kann die Länge des Programms in Bytes angegeben werden. Der Wert wird nicht ausgewertet.
DC.B	Relokativ-Byte	Eine Null in diesem Byte sagt aus, daß das Programm nicht verschiebbar ist. Dann muß die Startadresse absolut angegeben werden. Eine Eins gibt an, daß das Programm verschiebbar ist. Dann muß die Startadresse relativ zum Anfang des Eintrags angegeben werden.
DC.B	CPU-Byte	Hier kann festgelegt werden, für welche CPU das Programm geeignet ist. Dadurch wird vermieden, daß ein Programm nach dem Aufruf abstürzt oder hängenbleibt, wenn es für die ausführende CPU nicht geeignet ist. Es sind vier Werte zulässig: 0 = Programm ist für alle CPUs geeignet. 1 = Nur für die 68008-CPU. 2 = Nur für 68000/68010-CPUs. 4 = Nur für 68020-CPU.
DC.B	0,0	Reserviert.
DC.L	0,0	Abstand, wenn noch mehr Einträge folgen.

Hier können jetzt weitere Einträge folgen, die wieder mit $55AA0180 beginnen müssen. Die "Folge-Einträge" sind die einzigen Einträge, die nicht auf einer 1-Kbyte-Grenze beginnen müssen.