Der NDR-Klein-Computer

HANDBUCH

zum 680XX Grundprogramm

Version 7.0

2007 Jens Mewes

NKC

Die Legende lebt

1. Einführung

1.1. Vorwort (1)

Es sind nun schon einige Jahre ins Land gegangen und doch gibt es noch Einige die ihren NKC hegen und pflegen und natürlich auch benutzen. Da in der letzten Zeit einige neue Baugruppen für den NKC entwickelt wurden, habe ich mich an die Arbeit gemacht, das bewährte Grundprogramm anzupassen und einige wenige Bugs zu beseitigen. Grundlage hierfür war das hervorragend dokumentierte Grundprogramm von Ralph Dombrowski in der Version 6.3. Ebenso diente das zugehörige Handbuch als Grundlage für dieses Werk. Die jetzt vorliegende Grundprogramm Version 7.0 steht z.Zt. für die Prozessoren 68008 und 68000 zur Verfügung, eine Adaption für den 68020 wird wahrscheinlich bald folgen. Auch ich bin für Anregungen und Fehlermeldungen immer dankbar, da sich doch bei so einem Projekt der eine oder andere Fehler einschleichen kann. Eine Gewährleistung in irgendeiner Form kann allerdings nicht übernommen werden. Ich wünsche allen NKC-Usern weiterhin viel Freude mit Ihren Rechnern und hoffe, dass das neue Grundprogramm seinen Teil dazu beiträgt.

In diesem Sinne: **NKC - Die Legende lebt.**

Jens Mewes aka. DerInder Hamburg, September 2007

1.2. Vorwort (2)

Seit nun ungefähr 2 Jahren wird das bewährte Grundprogramm des NDR-Klein- Computers (NKC) ständig verbessert und überarbeitet. Dabei werden immer auch die neuesten Baugruppen mit einbezogen. Mit der nun vorliegenden Version 6.2 ist ein neuer Standard geschaffen worden, der für alle CPUs der 68000-Serie gleich ist. Es wurden viele neue Unterprogramme geschaffen und alte Programme verbessert. Die Benutzerfreundlichkeit ist noch einmal durch neue Editor- und Assemblerbefehle erhöht worden. Ein Problem der neuen Versionen war bisher die Anpassung des Handbuchs an die vielen verbesserten Möglichkeiten. Auch dieser Mangel ist mit dem Erscheinen dieser Dokumentation entgültig behoben. Es baut auf dem Handbuch der Version 4.3 auf, ist aber vollständig überarbeitet worden. Eine Einteilung in Kapitel und eine übersichtliche Dokumentation der Unterprogramme und Funktionen hat die Handlichkeit wesentlich verbessert. Informationen können nun leichter und schneller gefunden werden. Ich hoffe, dass nun das neue Grundprogramm mit seinen neuen Möglichkeiten von vielen Anwendern voll genutzt wird. Außerdem wünsche ich viel Spaß beim Erstellen oder der Benutzung von Programmen. Für Anregungen und Fehlermeldungen bin ich natürlich immer dankbar, da ich trotz gründlichster Arbeit natürlich keine Fehlerfreiheit garantieren kann.

Ralph Dombrowski Glückstadt, Juli 1989

1.3. 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 Bayrischen Fernsehen und von Radio Bremen ausgestrahlt. Es werden bald auch die Regionalsender anderer Bundesländer die Sendung in ihr Programm aufnehmen.
Der NKC wurde von Rolf-Dieter Klein Anfang der achtziger Jahre entwickelt. Zuerst lief er nur mit einer Z80-CPU und ein paar kleinen Baugruppen. Inzwischen ist er aber zu einem sehr leistungsfähigen System angewachsen, zu dem es nicht nur verschiedene CPUs und viele Baugruppen gibt, sondern auch ein mittlerweile recht großes Software-Angebot. Er ist nicht nur sehr gut zum Erlernen der Computer-Technologie geeignet, sondern lässt sich auch von fortgeschrittenen Programmierern für nahezu alle Anwendungsgebiete einsetzen. Auf Grund der großen Flexibilität kann er auch an viele andere Computersysteme angeschlossen werden und diese unterstützen. Durch sein offenes System wird er auch in Zukunft immer wieder auf den neuesten technischen Stand gebracht und kann so nicht altern.
In der letzen Zeit ist auch das Software-Angebot sehr beachtlich gestiegen. Es gibt nun eine große Auswahl von Betriebssystemen, die für jeden Geschmack das richtige bieten. Dazu gehören so bekannte Systeme wie OS/9, CP/M68K oder CP/M. Es sind aber auch Betriebssysteme direkt für den NKC entstanden, die speziell für ihn gemacht sind und seine "Eigenheiten" unterstützen. Dazu gehören JADOS oder SYSTEM-DOS. Mit diesen vielen Systemen ist auch eine große Software-Unterstützung gewährleistet.
Da es für den NKC sehr viele 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.

1.4. Wozu dient das Grundprogramm?

Jeder Computer weiß eigentlich gar nichts. Damit er überhaupt in Betrieb genommen werden kann, benötigt er ein mehr oder weniger großes Programm, das fest im Computer (z.B. auf Eproms) vorhanden ist. Bei einigen Computern ist es nur dazu gedacht, ein großes Programm von Diskette zu laden. Im NKC ist beim Betrieb mit einem Prozessor der 68000-Serie das Grundprogramm vorhanden, das sehr leistungsfähig ist und die wichtigsten Funktionen zum Betrieb eines Computers enthält. Es ist nicht nur dazu gedacht, ein Betriebssystem von Diskette oder Festplatte zu laden, sondern es kann auch ohne weitere Programme betrieben werden. Wer nicht das OS/9-System besitzt, benötigt das Grundprogramm auf jeden Fall für den Betrieb des Rechners, falls er nicht ein eigenes System geschrieben hat.
Das Grundprogramm enthält einen eigenen Programm-Editor, der auch zum Erstellen von Briefen geeignet ist. Weiterhin ist ein Assembler vorhanden, der sehr schnell und komfortabel die mit dem Editor erstellten Programme übersetzt. Auch sind Funktionen zum Testen von Programmen, Programmieren von Eproms oder Ansehen von Speicherbereichen vorhanden. All diese Funktionen sind direkt und ohne Programmierkenntnisse über eine Menüsteuerung verfügbar. Viele weitere Funktionen können von Programmen aus erreicht werden. Die Beschreibung aller Menüfunktionen ist im Kapitel 3 erklärt. Die Beschreibung der Programme, die über den Assembler erreicht werden können, folgt in Kapitel 4.
Das Grundprogramm ist nicht nur zum direkten Betrieb gedacht. Es ist auch Grundlage für nahezu alle Betriebssysteme. So greifen z.B. JADOS und CP/M68K auf viele Grundprogrammfunktionen zu. Auch die meisten Anwenderprogramme nutzen in irgendeiner Form Unterprogramme des Grundprogramms.

1.5. Literatur und Information

Da der NKC-Computer selber aufgebaut werden kann und ein offenes System ist, muss er natürlich durch die verschiedensten Informationen begleitet werden. Dazu gibt es eine ganze Reihe von Büchern oder Zeitschriften. Hier sind die Bücher aufgeführt, die nicht nur direkt für den Aufbau nötig sind, sondern auch die, die einen Einblick in die Programmiersprache Assembler bieten.

Bücher

Rolf-Dieter-Klein
"Rechner Modular"
Der NDR-Klein-Computer selbstgebaut und programmiert
ISBN 3-7723-8721-7
Franzis-Verlag München
Auf diesem Buch baut die NDR-Serie auf.

Werner Hilf und Anton Nausch
"M68000 Familie"
ISBN 3-921803-16-0
TE-WI-Verlag
Dieses Buch beschreibt sehr ausführlich den Befehlssatz der 68000-Serie und ist ein sehr guter Nachschlagewerk für Assembler- Programmierer.

"MC68020" und "MC68881" User's Manual
MOTOROLA
Dieser beiden Bücher sind für die Spezialisten unter den Programmierern gedacht und zeigen alle Funktionen des professionellen 68020 und 68881.

Zeitschriften

Die Zeitschriften "MC" und "ELO" des Franzis-Verlags
Dort sind viele Artikel über den NKC erschienen, die beim Aufbau und zum Betrieb des Computers sehr hilfreich sind. Es sind auch Sonderhefte erschienen.

Die firmeneigene Zeitschrift "LOOP"
der Firma GRAF, die alle zwei Monate die neuesten Informationen über den NKC herausbringt.
Dort werden auch viele Verbesserungen und Beispielprogramme abgedruckt.

Video-Kassetten

Lizenzierte Originalkassetten für den privaten Gebrauch sind für die Systeme VHS, BETA und VIDEO 2000 erhältlich.

Internet

Es gibt inzwischen diverse Internetseiten zum NKC, hier seien stellvertretend nur einige genannt:

www.drcrazy.de/nkc	Diese Seite ist die Haupt-Anlaufstelle, da es hier auch ein umtriebiges Forum gibt.
www.schuetz.thtec.org	Auf dieser Seite gibt es sehr viele der NKC-Dokumentationen, sowie die diversen Neuentwicklungen an Hard- und Software.
www.hehlhans.de/ndr1.htm	Diese Seite gibt einen Überblick über viele der NKC-Baugruppen.

2. Inbetriebnahme

2.1. Hardware-Vorraussetzungen

Für den Betrieb des Grundprogramms ist es nötig, dass die GDP64K, GDP64HS oder GDP-FPGA vorhanden ist. Außerdem werden mindesten 8 KByte RAM benötigt. Ein Tastaturanschluss (KEY) muss natürlich auch vorhanden sein.
Dass auch eine CPU und eine Speicherbaugruppe vorhanden ist, sollte selbstverständlich sein. Beim 68000 sind zwei Speicherbaugruppen und beim 68020 sind vier Speicherbaugruppen nötig.

Mehr Baugruppen werden nicht für den Betrieb des Grundprogramms benötigt, obwohl mit dieser Minimalausstattung natürlich nicht sehr viel gemacht werden kann. Eine Übersicht von Erweiterungen erfolgt in Kapitel 2.3.

2.2. Software-Vorraussetzungen

Es ist keinerlei Software für den Betrieb des Grundprogramms nötig, da das Grundprogramm selbst die Grundlage aller anderen Programme bildet. Allerdings sollte ein Diskettenbetriebssystem vorhanden sein, damit ein größeres Software-Angebot zur Verfügung steht.

2.3. Empfohlene Erweiterungen

Für den NKC sind natürlich eine Menge von Baugruppen und Programmen vorhanden. Hier erfolgt nun eine Aufstellung aller Baugruppen und vieler Programme. Die wichtigsten dieser Produkte werden kurz erklärt.

Hardware:

ACRTC	- Wird nicht vom Grundprogramm unterstützt
AD-Baugruppen	- Es sind Analog-Digital-Wandler erhältlich, die für Hobby-Bastler sehr nützlich sind.
BANKBOOT	- Durch diese Karte kann das Grundprogramm in hinteren Speicherbereichen betrieben werden. Dadurch ist vorne Platz für RAM, was Vorrausetzung für den Betrieb von z.B. CP/M68K ist. Beim 68020 ist diese Baugruppe nicht nötig.
CAS oder CAS2	- Baugruppe zum Anschluss eines Kassettenrecorders. Nur für den Anfänger gedacht.
CENT2	- Diese Baugruppe wird für den Anschluss eines Druckers benötigt. Ein Drucker ist für den fortgeschrittenen Programmierer und Anwender unverzichtbar.
COL256	- Farbgrafikkarte, die durch ein großes GRAFIK-Paket direkt vom Grundprogramm angesteuert werden kann.
DA-Baugruppen	- Es sind Digital-Analog-Wandel erhältlich, die z.B. für Robotersteuerungen verwendet werden können.

FLO2 oder FLO3	- Zum Betrieb eines oder mehrerer Diskettenlaufwerke unbedingt nötig. Auf die Benutzung von Diskettenlaufwerken kann heute nicht mehr verzichtet werden, da erst mit diesen die große Programmvielfalt offen steht.
GDP64K, GDP64HS	- Zum Anschluss eines Bildschirms. Die GDP64HS ist durch einige Zusatzfunktionen sehr zu empfehlen.
GDP-FPGA	- Die FPGA Variante der GDP bietet die Möglichkeit einen VGA-Monitor anzuschließen, Anschlüsse für PS2 Tastatur und Maus, sowie eine SER kompatible Schnittstelle (weitere Schnittstellen sind möglich).
GIDE	- Zum Anschluss von (max. 2) IDE Laufwerke (PATA)
HARDCOPY-MAUS	- Zum Anschluss einer Maus und zum Auslesen des GDP-Bildes. Diese Baugruppe ist nicht nötig beim Betrieb einer GDP64HS und der KEY3.
HEX	- Ein-Ausgabe-Einheit (nicht unterstützt)
IOE oder IOE2	- Baugruppe zum Anschluss eigener Peripherie.
KEY, KEY2 oder KEY3	- Zum Anschluss einer Tastatur, die vorhanden sein muss. (KEY3 auch für Mausanschluss)
KEY4	- Zum Anschluss einer PS2 Tastatur und Maus
PROMER oder PROMER2	- Zum Programmieren von Eproms. Für die Benutzer gedacht, die eigene Programme dauerhaft im Computer einbauen wollen.
RELAIS	- Baugruppe, auf der 8 Relais vorhanden sind.
ROA64 oder ROA256	- Für den weiteren Speicherausbau nötig. Ein größerer Speicher ist Vorraussetzung für die Arbeit mit komfortablen Programmen und sehr zu empfehlen.
R256DYN	- Dynamische Speicherbaugruppe. Ist preiswert, erfordert aber viele Wartezeit der CPU. Deshalb für schnellere Systeme nicht zu empfehlen.
RTC	- Uhrbaugruppe mit Dallas-Chip
SASI	- Baugruppe zur Ansteuerung einer Festplatte (wird nicht unterstützt).
SCSI	- Mit einer Modifikation der SASI-Baugruppe kann eine SCSI-Festplatte angesprochen werden, die vom Grundprogramm unterstützt wird.
SER	- Serielle Baugruppe.
SER2	- Serielle Baugruppe mit 2 Anschlüssen
SOUND	- Für Musikausgabe
SPRACHE	- Für einfache Sprachausgabe
SRAMDISK	- Solid-State RAM-Disk
UHR	- Uhrenbaugruppe (wird nicht mehr vertrieben) Alternativ SMART-WATCH

Die meisten Baugruppen haben eine feste Adresse, auf der sie betrieben werden sollen. Die Baugruppen werden vom Grundprogramm auch nur unterstützt, wenn sie sich auf diesen Adressen befinden.
Folgende Adressen sind für die verschiedenen Baugruppen vorgesehen.

Zu den Abkürzungen :

AD16x8	8 Bit Analog-Digital-Wandler mit 16 Kanälen
AD1x10	10 Bit Analog-Digital-Wandler
AD/DA	Analog-Digital- und Digital-Analog-Wandler
ARCTC	Farbgrafik-Baugruppen (nicht unterstützt)
BANKBOOT	Baugruppe zum Betreiben von RAM auf Adresse 0 (Bei 68008 und 68000/10)
CAS	Anschluss eines Kassettenrecorders zur Datensicherung
CLUT	Farbtabelle für COL256 und ACRTC
COLOR	Port für Farberweiterung der GDP
COL256	Farbgrafik-Baugruppe COL256 (RAM auf $EC000)
DA2x8	8 Bit Digital-Analog-Wandler mit 2 Kanälen
FARBT	Farbtabelle für Farberweiterung GDP
FLO	Diskettenansteuerung (FLO2 und FLO3)
GDP(HS)(FPGA)	GDP-Grafik-Baugruppe(n)
GIDE	IDE-Anschluss
HARDM	Hardcopy-Maus-Baugruppe
HEX	HEXIO-Baugruppe (nicht unterstützt)
KEY	Tastatur-Baugruppe
RELAIS	Baugruppe mit 8 Relais
RTC	Uhrenbaugruppe mit Dallas Chip
PROMER	Eprom-Programmierer
SASI	Baugruppe zur Festplattenansteuerung (nicht unterstützt)
SCSI	Baugruppe zur Ansteuerung der SCSI-Schnittstelle
SER	Serielle Ein-Ausgabeeinheit
SER2	Serielle Ein-Ausgabeeinheit mit 2 Kanälen
SER4	Ausgabeeinheit für mehrere serielle Schnittstellen (nicht unterstützt)
SOUND	SOUND-Baugruppe
SPRACH	Sprachausgabe-Baugruppe
SRAMDISK	RAM-Disk mit Batteriepuffer
UHR	Uhrenbaugruppe (wird nicht mehr vertrieben)

Übersicht der Baugruppenadressen:

$	0x	1x	2x	3x	4x	5x	6x	7x
0	HEX			IOE	SOUND (alt)	SOUND	GDP Seite	GDP
1	HEX			IOE	SOUND (alt)	SOUND	GDP-HS	GDP
2	HEX							GDP
3	HEX							GDP
4	SCSI (alt)		SCSI					GDP
5	SCSI (alt)		SCSI					GDP
6	SCSI (alt)	GIDE	SCSI					GDP
7	SCSI (alt)	GIDE	SCSI					GDP
8		GIDE			CENT		KEY	GDP
9		GIDE			CENT		KEY	GDP
A		GIDE						GDP
B		GIDE						GDP
C		GIDE	SRAMDISK					GDP
D		GIDE	SRAMDISK					GDP
E		GIDE	SRAMDISK					GDP
F		GIDE	SRAMDISK					GDP

$	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
0	PROMER1/2	SER2(4)	COLOR		FLO	AD/DA	AD16x8	SER
1	PROMER1/2	SER2(4)			FLO	AD/DA	AD16x8	SER
2	PROMER1/2	SER2(4)	FARBT		FLO	AD/DA	AD16x8	SER
3	PROMER2	SER2(4)	FARBT		FLO	AD/DA	AD16x8	SER
4	PROMER2	SER2(4)	CLUT		FLO	AD/DA	AD16x8	
5	PROMER2	SER2(4)	CLUT		FLO	AD/DA	AD16x8	
6	PROMER2	SER2(4)	CLUT		FLO	AD/DA	AD16x8	
7	PROMER2	SER2(4)			FLO	AD/DA	AD16x8	
8	HARDM	SER2(4)	ACRTC		BANKBOOT	SPEAK	AD16x8	DA2x8
9	HARDM	SER2(4)	ACRTC		RELAIS	SPEAK	AD16x8	DA2x8
A	HARDM	SER2(4)			CAS	SPEAK	AD16x8	UHR RTC
B	HARDM	SER2(4)			CAS	SPEAK	AD16x8	UHR RTC
C	HARDM	SER2(4)	COL256		SASI (alt)	SPEAK	AD16x8	AD1x10
D	HARDM	SER2(4)	COL256		SASI (alt)	SPEAK	AD16x8	AD1x10
E	HARDM	SER2(4)	COL256		SASI (alt)	SPEAK	AD16x8	UHR (E050)
F	HARDM	SER2(4)	COL256		SASI (alt)	SPEAK	AD16x8	

Software:

Zum Grundprogramm sollte bei der Ausstattung mit mindestens einem Diskettenlaufwerk auch ein Diskettenbetriebssystem vorhanden sein. Zu empfehlen ist für den NKC das Betriebssystem JADOS, das preiswert und sehr leistungsfähig ist. Es wird auch direkt vom Editor des Grundprogramms unterstützt. Für ein großes Software-Angebot sorgt das teure Betriebssystem CP/M 68K. Wer allerdings wirklich professionell arbeiten will, sollte sich OS/9 anschaffen, welches ein MULTI-USER MULTI-TASKING System ist. Für dieses ist aber auch eine Harddisk nötig.

Die meisten NKC-Besitzer benutzen allerdings das JADOS-System.

Empfehlungen

Ich empfehle als Erweiterung zur Grundkonfiguration mit GDP und KEY zusätzlich den Anschluss eines oder mehrerer Diskettenlaufwerke mit dem entsprechenden Betriebssystem und die Verwendung eines Druckers. Außerdem sollten mindestens 256 KByte RAM vorhanden sein.

2.4. Verschiedene CPUs

Das 68000-Grundprogramm ist für alle drei CPUs der 68000-Serie erhältlich. Dieses Handbuch ist für alle Prozessoren gedacht. Es enthält deshalb an den Stellen, an den Abweichungen zwischen den Grundprogrammen bestehen, entsprechende Hinweise.

Das Grundprogramm für den 68000 kann auch mit dem 68010 betrieben werden, wenn im Quelltext des Grundprogramms, das auf Diskette erhältlich ist, an den gekennzeichneten Stellen die Änderungen durchgeführt werden. Anschließend muss es neu assembliert und auf Eproms gebrannt werden.

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

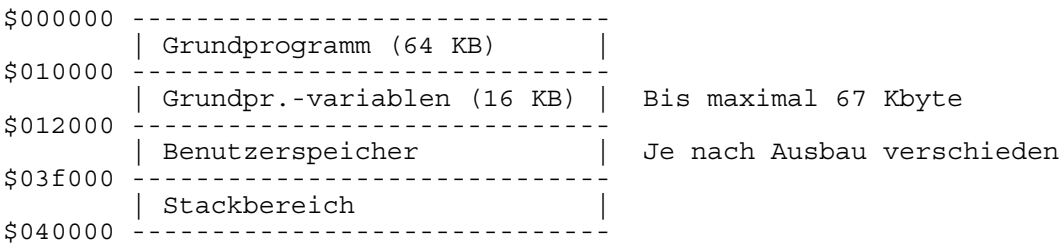
2.5. Speicheraufbau

Es gibt zwei verschiedene Möglichkeiten, wie das Grundprogramm betrieben werden kann. Es muss aber direkt hinter dem Grundprogramm mindestens ein Speicherbaustein vorhanden sein. Der weitere Speicheraufbau ist egal, wobei es aber vorteilhaft ist, möglichst wenig getrennte Speicherbereiche zu haben. Am günstigsten ist ein großer RAM-Bereich.

Betrieb ohne BANKBOOT-Karte

Wenn keine BANKBOOT-Karte vorhanden ist, so muss das Grundprogramm auf der Adresse Null beginnen.

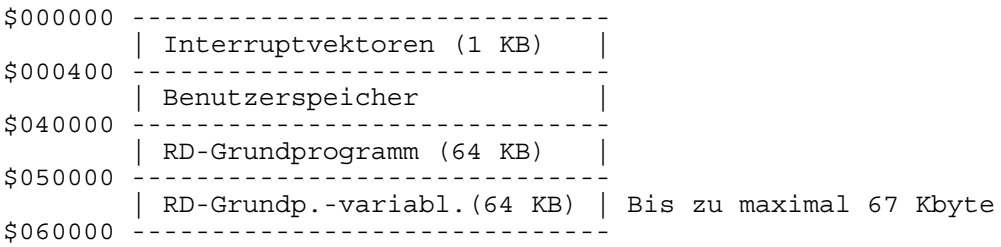
Beispiel:



Betrieb mit BANKBOOT-Karte

Beim Betrieb mit der BANKBOOT-Karte oder beim 68020 muss das Grundprogramm nicht auf der Adresse Null liegen. Es kann auf jeder beliebigen Adresse eingesetzt werden. Dabei muss auf der Adresse Null aber RAM vorhanden sein.
Ein möglichst großer Speicherbereich sollte ab der Adresse Null vorhanden sein. Das Grundprogramm sollte dann weit hinten liegen. Die Adresse ist dabei aber egal. Es wird für den 68008 die Adresse $0E0000 empfohlen. Beim 68000 sollte das Grundprogramm auf der Adresse $1e0000 liegen und beim 68020 auf $3e0000. Es ist aber auch hier darauf zu achten, dass direkt hinter dem Grundprogramm RAM vorhanden ist.

Beispiel:



2.6. Voreinstellungen

Bevor das Grundprogramm gestartet werden kann, sollten auf der KEY-Karte noch ein paar Voreinstellungen durchgeführt werden. Die DIL-Schalter haben nämlich seit der Version 6.0 eine feste Bedeutung. Damit das Grundprogramm richtig funktioniert, müssen die Schalter richtig gestellt werden.

Schalternummer	Stellung AUS	Stellung EIN
0	Reserviert	Reserviert
1	Normaler Beginn	Autoboot beim Einschalten
2	Keine Uhrenbaugruppe	Uhrenbaugruppe vorhanden
3	Alte GDP64K	Neue GDP64HS
4	Keine HARDDISK	SCSI-HARDDISK vorhanden
5	Keine IDE-Disk	IDE-Disk vorhanden
6	Keine GDP-FPGA	GDP-FPGA vorhanden
7	Reserviert	Reserviert

Wenn man möchte, dass beim ersten Einschalten ein Autoboot erfolgt, so muss der Schalter 1 eingeschaltet werden.

Ist eine Uhrenbaugruppe vorhanden, so muss Schalter 2 eingeschaltet werden. Wenn auch noch die SMART-Watch erkannt wird, so wird immer diese angesprochen.

Soll die SMART-WATCH eingebaut werden, so sollte darauf geachtet werden, dass sie nur unter RAM betrieben wird. Außerdem darf sie nicht im RAM-Bereich direkt hinter dem Grundprogramm sowie am Ende des ersten RAM-Bereichs hinter dem Grundprogramm (Stackbereich) betrieben werden.
Wenn weitere Software benutzt werden soll, die direkt auf die SMART-WATCH zugreift, so muss auch hier darauf geachtet werden, dass das Programm nicht mit der Uhr in Konflikt gerät, sonst kann es zu einem Systemabsturz kommen.

Ab Version 7.0 kann auch die RTC mit Dallas-Chip eingesetzt werden.
Mit dieser Uhr steht ein NVRAM (Batterie gepuffertes RAM) zur Verfügung, in dem die Systemeinstellungen gespeichert werden. Diese Einstellungen überschreiben auch die Einstellungen des DIL-Schalters, so dass alle Grundeinstellungen des Grundprogramms Softwaremäßig vorgenommen werden können.

Wenn die neue GDP64HS im System eingebaut ist, so sollte der Schalter 3 eingeschaltet werden, damit die neuen Möglichkeiten wie Hardscroll oder Bildschirmauslesen genutzt werden können.

Seit der Version 6.2 ist jetzt auch über das Grundprogramm der Betrieb einer SCSI-HARDDISK möglich. Die Beschreibung der SCSI-HARDDISK und der verfügbaren Programme erfolgt in Kapitel 4.2. Um die Routinen der SCSI-HARDDISK aber aufrufen zu können bzw. um von der SCSI-HARDDISK booten zu können, muss aber der DIL-Schalter 4 auf ein geschaltet werden.

Schalter 5 dient zur Kennung, dass eine GIDE Baugruppe vorhanden ist. Diese ist für den Anschluss von IDE (PATA) Laufwerken. IDE-Festplatten werden direkt vom Grundprogramm (ab Version 7.0) unterstützt.

Der Schalter 6 ist (ab Version 7.0) die Kennung für die GDP-FPGA Baugruppe, dieser sollte gesetzt werden, da diese Baugruppe über ein, zu den Standart GDP64 Baugruppen, abweichendes Timing hat (60 Hz Bildfrequenz).

2.7. Der erste Start

2.7.1. Grundprogramm auf Eproms

Wenn die DIL-Schalter eingestellt sind, so müssen nur noch die Eproms richtig eingesetzt werden. Dazu werden beim 68008 alle Eproms in der richtigen Reihenfolge hintereinander auf der richtige Adressen eingesetzt, die auf der ROA eingestellt werden muss. Die Einstellung der Adressen ist dem Handbuch der ROA zu entnehmen.
Beim 68000 und 68020 muss darauf geachtet werden, dass die Eproms in der richtigen Reihenfolge auf der richtigen BUS-Seite eingesetzt werden. Die Anordnung der Adressen ist dem jeweiligen Handbuch zu entnehmen. Hinter dem Grundprogramm muss dann aber auf jeder BUS-Seite mindestens ein RAM-Baustein vorhanden sein.

Sind alle Einbaumaßnahmen durchgeführt, so sollte noch einmal alles gründlich überprüft werden, damit keine Kurzschlüsse entstehen. Auch auf den richtigen Sitz der Eproms sollte geachtet werden.

Nun kann der Start beginnen. Schaltet man den Computer ein, so muss nach kurzer Zeit die Copyrightmeldung erscheinen und danach die erste Menüseite bzw. das einseitige Menü. Ist dies nicht der Fall, so sollte einmal die RESET-Taste gedrückt werden. Passiert dann noch nichts, so muss der Computer ausgeschaltet werden und die Fehlersuche kann beginnen.

Funktioniert soweit alles richtig, so kann begonnen werden, die Tastatur auszuprobieren. Dazu ist in Kapitel 3 beschrieben, was mit dem Grundprogramm gemacht werden kann.

2.7.2. Grundprogramm auf Diskette

Auf der ersten Diskette des Grundprogramms befindet sich ein File, das bereits das übersetzte Grundprogramm enthält. Nur dieses File ist für die folgenden Aktionen interessant. Alle anderen Files der Disketten sind in dem File LIESMICH.TXT beschrieben und sollen uns hier nicht weiter interessieren.

Zuerst einmal muss sichergestellt sein, dass das Grundprogramm, das auf Eproms im System vorhanden sein muss, nicht auf der Adresse Null liegt, da das neue Grundprogramm dann nicht gestartet werden kann. Dazu ist beim 68008 und beim 68000 die BANKBOOT-Karte nötig; beim 68020 kann die alte Eprom-Version einfach nach hinten gesteckt werden.

Jetzt muss der erste Speicherbereich groß genug sein, um das neue Grundprogramm mit 64 KByte und die Systemvariablen, die für den ersten Test ungefähr 5 KByte benötigen, aufzunehmen.

Das neue Grundprogramm sollte dann von JADOS aus mit dem Befehl TLOAD (auch wenn es kein Text ist) auf eine Adresse geladen werden, die etwa 70-80 KByte vor dem Ende des ersten Speicherbereichs liegt. Für den ersten Test reicht der Bereich aus, später sollte hinter dem Grundprogramm noch etwa 50 KByte RAM liegen, damit die Symboltabelle genügend Platz hat.

Ist das Grundprogramm geladen, so genügt ein Druck auf die RESET-Taste und die Copyrightmeldung und die erste Menüseite sollten erscheinen.

2.8. Fehlermöglichkeiten

Funktioniert der Computer nicht, so sollten zuerst folgende Dinge überprüft werden.

Fehlermöglichkeit	Ursache und Abhilfe
- Der Computer funktioniert mit einer älteren Grundprogrammversion und jetzt nicht mehr.	- Die DIL-Schalter müssen richtig eingestellt werden. Adresse der SMART-WATCH ändern.
- Das Grundprogramm läuft nicht an.	- Die Eproms stecken verkehrt in der ROA. Hinter dem Grundprogramm befindet sich kein RAM. Das Monitorkabel ist lose. Das Grundprogramm befindet sich auf einer falschen Adresse (z.B. IO-Bereich).
- Es ist keine Tastatureingabe möglich.	- Das Tastaturkabel ist lose.

Natürlich können noch viele andere Fehler auftreten, die dann aber nichts mit dem Grundprogramm zu tun haben, sondern von anderen Baugruppen ausgelöst werden. Anleitungen zur Fehlerbehebung sind dem jeweiligen Handbuch zu entnehmen.

3. Grundprogramm - Menü

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 Eingabe 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 Buchstaben 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 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.

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

Beim Starten des Grundprogramms erscheint zunächst ein kurzer Begrüßungs-Bildschirm, danach das Menü. Hierher kehrt das Grundprogramm auch nach einem RESET oder nach Eintritt einer Ausnahmebedingung wie z.B. einem INTERUPT oder einem ADRESSFEHLER zurück.

3.1.1. A = 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 Minuszeichens 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 ÄNDERN 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 Stings 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
```
1KB 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 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 9CR $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. B = Starten

Es erscheint ein Eingabefenster, in dem die Adresse eines Programms eingegeben werden soll. Dabei können auch Symbole (Namen) verwendet. 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 Adressangabe 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. C = Ansehen

Um schnell einen Überblick über die Werte in einem bestimmten Speicherbereich 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 von 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 einen 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 hexadezimaler 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. D = 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 Grundprogrammmenü 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 einen 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 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.

3.1.5. E = 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.
'T'	Die Autotab-Funktion ist aktiv.
'S'	Es wird Seitenweise gescrollt.
'Z'	Es wird Zeilenweise gescrollt.

Nun folgt die Aufstellung aller verfügbaren 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 vorn geblättert, d.h. es wird der Bildschirmausschnitt weiter hinten im Text sichtbar.

CTRL-D

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

CTRL-E

Der Cursor wird um eine Zeile nach oben bewegt. Befindet es sich schon 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 von Zeilenanfang an gerechnet.

CTRL-J

Die Hilfe-Seite, d.h. die Seite mit den Erläuterungen 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 welcher Adresse 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-KD

Block ausdrucken. Diese Operation ist nur möglich, wenn ein Block markiert ist.

CTRL-KF

Editoreinstellungen speichern. Mit diesem Befehl werden die aktuellen Editoreinstellungen als Datei auf ein JADOS-Laufwerk gespeichert (siehe auch Kapitel 3.3.1.2. Editordaten).

CTRL-KG

Editoreinstellungen laden. Mit diesem Befehl werden die aktuellen Editoreinstellungen aus einer Datei von einem JADOS-Laufwerk geladen (siehe auch Kapitel 3.3.2.2. Editordaten).

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 Datei 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-Namenskonversion 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 haben.

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 man 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 demselben 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-KV

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

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 (<RETURN>). Bei der 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-O

Der Cursor wird auf die vorherige TAB-Position gesetzt.

CTRL-P

Der Zeichensatz wird von deutsch auf amerikanisch umgeschaltet oder umgekehrt. Die aktuelle Einstellung wird in der Status-Zeile des Editors angezeigt. Bei deutschen 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 nächst Taste wählt die gewünschte Funktion aus.

CTRL-QA

Dieser Befehl ermöglicht es, Zeichenfolgen im gesamten Text zu ersetzten. Zuerst wird eingegeben, welche Zeichenfolge ersetzt werden soll. Dann muss die neue Zeichenfolge, die die alte ersetzten soll, eingegeben werden. Wird die Zeichenfolge im Text gefunden, wird der Cursor dahinter gesetzt. Durch Eingabe eines J (=Ja) auf der Tastatur wird die Ersetzung ausgeführt und die nächste Übereinstimmung gesucht. Bei Eingabe eines N (=Nein) wird nicht ersetzt und die nächste übereinstimmende Textstelle 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 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-QD

Der Cursor wird ans Ende der aktuellen Zeile gesetzt.

CTRL-QE

Der Cursor wird links Oben im sichtbaren Fenster positioniert

CTRL-QF

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

Der Cursor wird an den Anfang der aktuellen Zeile gesetzt.

CTRL-QT

Die aktuelle Zeile wird ab der Cursorposition umgebrochen. Der Cursor steht anschließend am Anfang der neuen Zeile.

CTRL-QV

Die aktuelle Zeile wird mit der darauf Folgenden zu einer Zeile verschmolzen.

CTRL-QX

Der Cursor wird auf die unterste Zeile des aktuellen Fensters positioniert.

CTRL-R

Der Bildschirmausschnitt wandert um 15 Zeilen zu 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 restlichen Zeichen der Zeile werden nach rechts geschoben. Das letzte Zeichen geht verloren, wenn die Zeile durch das Einfügen mehr als 80 Zeichen hat.

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 (CR / RETURN) 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 um eine Zeile nach unten gesetzt. Befindet er sich in der letzten Zeile, wird der Bildschirmausschnitt verschoben.

CTRL-Y

Die Zeile in 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 verschiedenen Stellen im Speicher stehen. Mit diesem Befehl wird ein neuer Textstart eingestellt. An der betreffenden Stelle muss aber tatsächlich ein Text vorhanden sein (siehe Kapitel 4.2. TEXTSTART). Es folgt dann ein Sprung in den betreffenden Text. Der Ausgangstext wird nicht verändert.

ESC C

Alle Tabulatoren werden gelöscht.

ESC D

Macro-Text definieren. Nach Eingabe dieses Befehls wird die aktuelle Belegung der Macro-Texte mit der ihnen zugeordneten Nummer angezeigt. Zum Ändern eines Textes muss man zuerst die zugeordnete Nummer eingeben. Hierauf erscheint ein Eingabefeld für den neuen Text. Nach Bestätigung der Eingabe mit CR wird der Text gespeichert und zum Editor-Fenster zurückgekehrt (siehe auch ESC 0-9).

ESC I

Die Standard Tabulatoren werden wiederhergestellt. Diese sind jeweils in acht Zeichen Abstand zueinander von linken Rand gesehen.

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

Die Standardbelegung der Macro-Texte wird wieder hergestellt.

ESC N

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

ESC O

Der Inhalt des sichtbaren Bildschirmausschnitts wird auf dem Drucker ausgegeben. Dabei erfolgt keinerlei 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 (Seitenweise) zu sehen, werden immer 15 Zeilen gleichzeitig vor- oder zurückgeblättert. Bei einem Z (Zeilenweise) werden die Zeilen nacheinander bewegt. Mit ESC S schaltet man die Scrollarten 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.

ESC W

An der aktuellen Cursorposition wird ein Tabulator gesetzt, oder falls schon Einer vorhanden ist, gelöscht.

ESC 0-9

Ein Macro-Text (nicht zu verwechseln mit der MACRO-Funktion des Assemblers) wird an der aktuellen Cursor-Position geschrieben. Die Belegung der einzelnen Macro-Texte kann jederzeit mit der Funktion ESC D geändert werden.
Die Standard-Belegung ist folgende:

ESC 0	move
ESC 1	clr
ESC 2	add
ESC 3	sub
ESC 4	bsr
ESC 5	bra
ESC 6	dbra
ESC 7	tst
ESC 8	trap
ESC 9	rts

3.1.6. F = 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 (ehemaligen) 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 (68881/2) 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 Programms 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 die 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 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 Werte dann im Speicher ablegen. Er ist auch in der Lage, Variablen bzw. Konstanten 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 sein. Sie müssen durch Hochkommas eingefasst werden.
DC.W Wert, Wert, 'AB', …	Hier werden Werte als Wörter (2 Bytes) abgelegt. Die Größenangabe kann entfallen (DC). Bei Texten werden nur die letzten beiden Buchstaben berücksichtigt.
DC.L Wert, Wert, 'ABCD'	Die Werte werden als Langworte (4 Bytes) abgelegt. Texte werden nur mit den letzten vier Buchstaben abgelegt.

DC.W und DC.L müssen auf einer Wortgrenze beginnen, da die 680X0-CPUs nicht auf ungerade Adressen zugreifen. Das ist auch bei der 68020-CPU zu beachten, damit die unter ihr geschriebenen Programme für den Einsatz unter anderen 680X0-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 Programms FPUWERT benutzt werden. Dazu muss aber eine FPU im System vorhanden sein.

DC.S Wert, Wert	Eine Floating-Point-Zahl wird in Single-Density (einfache 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 in die Tabelle eingetragen. In diesem Fall kann 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 zu 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 Befehle werden vom
      MOVEQ #100, D2          * DIS-Assembler übersetzt.
      DEBUGAN                       * Ab hier 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 dann 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(4)  * 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.
DS 0	Dies ist eine Sonderform des DS-Befehls, die bewirkt, dass alle drauffolgenden Werte auf eine gerade Adresse (Wortgrenze) zu liegen kommen. Hiermit wird je nach aktueller Adresse, der PC um 0 oder 1 erhöht.

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 den ersten Durchlauf abgebrochen.

EQU

Mit dieser Anweisung kann jedem Symbol ein Wert zugewiesen werden, wie es mit dem Befehl ZUWEIS möglich ist. Dann wird jedes Mal, wenn das Symbol verwendet wird, der dem Symbol zugewiesene 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 beliebiger Stelle im Listing einsetzten und dadurch viel Tipparbeit sparen. Außerdem können Werte übergeben werden, sodass Variationen möglich sind.

Eingeleitet wird die Definition 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 (z.B.: .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 arbeitenden Prrogrammteilen 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
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, TEXT0(PC),           *  Text  0  ausgeben,  aber
nicht D0 laden,
      RTS                                 * D0 bleibt unverändert ($21)

TEXT0:
      DC.B 'Hallo', 0

TEXT1:
      DC.B 'Texttext', 0
```

Es soll eine Schleife realisiert werden:

```
MACRO WARTE                          * Name des Macros.
      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 erfolg 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 umdefiniert 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 SYMBOL den Wert 5
SYMBOL NEWEQU 7         * mit EQU käme hier eine Fehlermeldung
                        * Ab jetzt hat 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 die 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 alleine 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-Anweisungen auf Null.

RSSET

Setzt den Zähler für die RS-Anweisungen 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 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 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 ein 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 durch SYMCLR wieder gelöscht wurden.

USR

Einige der vorstehend beschriebenen Befehle können die Ausgabe des Listings auf vom Grundprogramm definierten Schnittstellen lenken. Will man aber die Ausgabe auf ein selbst gebautes Peripheriegerät oder ins RAM lenken, seht dafür der Befehl USR zur Verfügung. Er lenkt die CO2-Routine um. Allerdings muss gemäß der Beschreibung von CO2 ein JMP-Befehl umgelenkt werden, denn sonst wird trotzdem auf dem 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 Durchlauf des Assemblers gegeben, ist noch kein Wert im RAM abgelegt.

CTRL-S stoppt den Assembliervorgang (wichtig für Bildschirmausgabe), CTRL-Q setzt ihn fort.

Mit SPACE (Leertaste) kann die Ausgabegeschwindigkeit umgeschaltet werden, wenn die GDP64HS vorhanden ist.

Für fortgeschrittene Programmiere:

Ab Version 6.0 liest das Grundprogramm den Text 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.

3.1.7. G = Bibliothek

Im Speicher des NKC können mehrere Programme an verschiedenen Stellen abgelegt sein, auf die der Benutzer jederzeit zugreifen kann. Allerdings währe es schwierig, wenn man sich alle Startadressen der Programme merken müsste. Man kann deshalb Programme in einer bestimmten 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 im Kapitel 5.7 erläutert.

Damit man die erörterten Programme einfach aufrufen 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 fester Adresse) 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 vor- und zurückblättern. 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.

3.1.8. H = Speichern Disk – JADOS

Dieser Menüpunkt dient zum Speichern von Daten oder Einstellungen auf einem Datenträger. Dieser Menüpunkt ist nur verfügbar, wenn JADOS geladen ist!

3.1.9.1. Datei

Es wird ein Speicherbereich als Datei auf dem Datenträger gespeichert.
Als Erstes muss man das Laufwerk sowie den Dateinamen angeben, hierbei sind die Konventionen von JADOS zu beachten.
Als Zweites wird die Quelladresse abgefragt, dies ist die Adresse auf der ein Programm oder Text beginnt.
Als Drittes muss man die Anzahl der Sektoren (je 1024 Bytes) angeben die gespeichert werden sollen.

3.1.9.2. Editordaten

Hiermit werden die aktuellen Editor-Einstellungen als Datei auf einem Datenträger gespeichert.
Auch hier muss man das Laufwerk und den Dateinamen gemäß JADOS-Konvention angeben.
Eine Editordaten-Datei bekommt eine besondere Kennung mitgegeben, die sie als solche ausweist.

3.1.9.3. Druckerdaten

Hiermit werden die aktuellen Druckereinstellungen (siehe DRUCKERSTEUERUNG) als Datei auf einen Datenträger gespeichert. Wie gewohnt muss man auch hier das Laufwerk und den Dateinamen gemäß JADOS-Konvention eingeben. Auch die Druckerdaten-Datei enthält eine Kennung zum ausweisen.

3.1.9. I = Laden Disk – JADOS

Dieser Menüpunkt dient zum Laden von Daten oder Einstellungen von einem Datenträger. Dieser Menüpunkt ist nur verfügbar, wenn JADOS geladen ist!

3.3.2.1. Datei

Es wird eine Datei vom Datenträge in einen Speicherbereich gelesen.
Als Erstes muss man das Laufwerk sowie den Dateinamen angeben, hierbei sind die Konventionen von JADOS zu beachten.
Als Zweites wird die Zieladresse abgefragt, dies ist die Adresse auf der ein Programm oder Text beginnen soll.

3.3.2.2. Editordaten

Hiermit werden Editor-Einstellungen aus einer Datei gelesen.
Auch hier muss man das Laufwerk und den Dateinamen gemäß JADOS-Konvention angeben.
Eine Editordaten-Datei bekommt eine besondere Kennung mitgegeben, die sie als solche ausweist.

3.3.2.3. Druckerdaten

Hiermit werden die aktuellen Druckereinstellungen (siehe DRUCKERSTEUERUNG) aus einer Datei gelesen. Wie gewohnt muss man auch hier das Laufwerk und den Dateinamen gemäß JADOS-Konvention eingeben. Auch die Druckerdaten-Datei enthält eine Kennung zum ausweisen.

3.1.10. J = Inhalt Disk

Es wird das Inhaltsverzeichnis des gewählten Laufwerks ausgegeben. Hierbei sind die Selektionsmöglichkeiten von JADOS zulässig.

3.1.11. K = Löschen Datei

Eine oder mehrere Dateien auf dem gewählten Laufwerk werden gelöscht.

3.1.12. L = Kopieren Datei

Eine oder mehrere Dateien werden kopiert.

3.1.13. M = Umbenennen Datei

Eine Datei wird umbenannt.

3.1.14. N = Booten

Mit diesem Menüpunkt kann ausgewählt werden, von wo ein Betriebssystem, das sich auf Diskette oder Festplatte befindet, gestartet werden soll. Dabei kann man zwischen vier Diskettenlaufwerken, SCSI-Festplatte, IDE-Festplatte oder SRAM-Disk wählen.

3.1.14.1. Floppy 1

Es 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 seht kein NOP am Anfang, geht es zurück ins Grundprogramm.

3.1.14.2. Floppy 2

Siehe "3.3.4.1. Floppy 1"

3.1.14.3. Floppy 3

Siehe "3.3.4.1. Floppy 1"

3.1.14.4. Floppy 4

Siehe "3.3.4.1. Floppy 1"
Es ist zu beachten, dass die Floppy 4 nicht erreichbar ist, wenn die SRAM-Disk vorhanden ist.

3.1.14.5. SCSI-Disk

Es wird der erste Sektor der SCSI-Disk 0 ins RAM gelesen, danach geht es weiter wie unter "Floppy 1" beschrieben. Allerdings wird im Register D0.L eine Eins als Harddisk-Kennung übergeben.

3.1.14.6. IDE-Disk

Diese Funktion dient zum Booten von einem IDE-Laufwerk. Dieses Laufwerk muss als Masterlaufwerk mit der GIDE-Baugruppe verbunden sein.
Es gibt zwei verschiedene Boot-Möglichkeiten. Zuerst wird ein Sektor (512 Bytes) gelesen, wenn an erster Stelle ein NOP steht, wird in den NKC-Modus geschaltet. Das bedeutet, dass immer zwei Sektoren á 512 Bytes gelesen und geschrieben werden und dass die SCSI-Routinen auf die IDE-Schnittstelle umgeleitet werden. Hierdurch wird erreicht, dass Programme die die SCSI-Hardware unterstützen, Problemlos auch mit IDE-Laufwerken arbeiten (z.B. JADOS). Hierbei sind allerdings ein paar Einschränkungen des IDE-Befehlssatzes gegenüber dem SCSI-Befehlssatz zu beachten (siehe Kapitel 4.2. unter IDEDISK), auch ist ein gemischter Betrieb von IDE- und SCSI-Laufwerken dann nicht möglich.
Die zweite Boot-Variante bietet die Möglichkeit mit einen Master-Boot-Rekord (MBR) zu arbeiten und z.B. unterschiedliche Betriebssysteme von verschiedenen Partitionen zu laden. Auch hier wird zuerst einen Sektor (512 Bytes) geladen und dann geschaut ob als 3. Wort ein NOP steht. Wenn dass der Fall ist wird das Programm ab der Adresse Null gestartet. Eine Umleitung der SCSI-Routinen erfolgt hierbei nicht.
Auch bei dieser Funktion wird in D0.L eine Eins, als Kennung übergeben.

3.1.14.7. SRAM-Disk

Sofern eine SRAMDISK im NKC vorhanden ist, wird dieser Menüpunkt statt der Floppy 4 eingeblendet. Das booten erfolgt analog zu zum booten von einer Floppy. Zu beachten ist das abweichende Format der SRAMDISK, dieses ist auch von der Kapazität des Laufwerks abhängig. Folgenden Aufbau hat die SRAMDISK:

Kapazität	Anzahl	
KB	Sektoren	Spuren
512	8	64
1024	8	128
1536	16	96
2048	16	128

Ein Betriebssystem (z.B. JADOS) muss an diese Aufteilung natürlich angepasst (gepatcht) werden.

3.1.15. O = EPROM lesen

Zunächst muss das EPROM und damit auch die Baugruppe wie bei EPROM Programmieren ausgewählt werden. Dabei ist der Lesealgorithmus bei Promer1 und Promer2 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.

3.1.16. P = 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 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 vo 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 µs	12,75 Volt	6,25 Volt
N	=	Promer2 27512	100 µs	12,75 Volt	6,25 Volt
O	=	Promer2 27010	100 µs	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 die Programmierfassung eingesetzt 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 welchen 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, welcher Prozessor für den anderen NKC verwendet wird. Es sind die Eingaben 1, 2 und 4 möglich. Die Angabe von 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 geprüft. 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 aber trotzdem programmiert werden. Der Programmiervorgang kann mit ESC abgebrochen werden. 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.

3.1.17. Q = Speicherbereiche

Es werden auf dem Bildschirm alle verfügbaren Speicherbereiche angezeigt. Die durchgehenden Flächen sind dabei RAM-Bereiche. Die schraffierten Bereiche 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 Variablen und die Symboltabelle verwendet werden kann und immer direkt hinter dem Grundprogramm beginnt.

3.1.18. R = 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 Drucksteuerung 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 angegeben 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 ausgegeben wird.

D = Schriftart

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

E = Druckrichtung

Normalerweise wird ein Ausdruck immer bidirektional ausgedruckt, was bedeutet, dass der Druckkopf 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 unidirektionalen 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 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 erst 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 ein 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 rücksetzten

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 den 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 Einstellung 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 mach die erste Seite der Druckersteuerung wieder sichtbar.

Z = Zurück

Hiermit wird die Druckersteuerung verlassen.

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

3.1.19. S = IO lesen

Da auf den IO-Adressen sehr oft veränderliche Daten liegen, die man einmal ansehen möchte, ist die Funktion vorhanden. ANSEHEN ist dafür nicht 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 eingegeben werden. Dann wird der Wert von dieser Adresse gelesen. Die Ausgabe erfolgt dabei immer in binärer, dezimaler und hexadezimaler Form. Durch drücken der Taste D wird auf einen Modus geschaltet, der den Wert dauernd (alle 20 / 16.67 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.

3.1.20. T = 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 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.

3.1.21. U = Einzelschritt

Umfangreiche Programme funktionieren meist nicht auf Anhieb fehlerfrei. Die groben Fehler sind aber leicht zu entdecken, nur bei 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 Text-Editor 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 68881/68882.
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 Programms eingegeben wird. Danach wird der erste Befehl auf dem Bildschirm ausgegeben. Nach drücken der RETURN Taste wird der 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 dem sie kennzeichnenden Buchstaben ausgegeben, was die Übersichtlichkeit stark erhöht.
Mit Hilfe der nachfolgend Aufgelisteten Befehle kann EINZELSCHRITT auf sehr komfortable Weise bedient werden. Sie werden im Übrigen auch beim Start des EINZELSCHRITT in einer Kurzübersicht angezeigt. Der Aufruf der Befehle erfolgt durch Eingabe des für den Befehl angeführten Buchstaben.

B = Bis ADRESSE/BEREICH ausführen

Das Programm wird soweit ausgeführt, bis der Befehl, dessen Adresse angegeben wurde, ausgeführt wird. 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/JSR 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 abgeschaltet 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)

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) setzten

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 Menüpunk während des Einzelschritts erreichen. Welche Eingaben zulässig sind, wird beim Unterprogramm GRUND im 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 der Statuszeilen sichtbar ist, arbeitet die Funktion T.

J = Befehl einzeln/Unterbrechen bei JMP/JSR (Nur beim 68020)

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 allerdings geboten, da es zu Stack-Fehlern 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 setzten

Alle Daten- und Adressregister sind direkt manipulierbar. Beim 68020-Prozessor können auch die Register 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 werden, 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 der unteren Zeile ausgegeben (beim 68020 Grundprogramm nicht auf der zweiten Seite). 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 soweit 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 der 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.

3.1.22. V = System Konfiguration

Dieser Menüpunkt dient zum einstellen und abspeichern einiger Systemkonfigurationen. Er kann nur dann benutzt werden, wenn eine UHR3 mit DS12887 Chip im System ist, ansonsten bekommt man nur einen Hinweistext. Beim erstmaligen Aufruf dieser Funktion sind die Konfigurationswerte mit den Standardeinstellungen vorbelegt.

A = DIP-Key

Die Einstellung des DIP-Keys auf der KEY-Platine kann hiermit überschrieben werden. Das bedeutet, dass diese hier gespeicherten Einstellungen beim Systemstart verwendet werden und nicht die des DIP-Keys.

Bedeutung der Bits:

```
7   6   5   4   3   2   1   0
|   |   |   |   |   |   |   |___Reserviert
|   |   |   |   |   |   |_______Autoboot beim Starten des Rechners
|   |   |   |   |   |___________Uhrenbaugruppe vorhanden
|   |   |   |   |_______________GDP64HS vorhanden
|   |   |   |___________________SCSI-HARDDISK vorhanden
|   |   |_______________________IDE-Disk vorhanden
|   |___________________________GDP-FPGA vorhanden
|_______________________________Reserviert
```

B = Autoboot

Mit dieser Einstellung bestimmt man die Reihenfolge, mit der nach einem bootfähigen Medium gesucht wird. Diese Funktion ist natürlich nur sinnvoll, wenn das Bit 1 des DIP-Keys gesetzt ist. Folgende Werte sind Möglich:

00 = Floppy 1
01 = Floppy 2
02 = Floppy 4
03 = SRAMDISK oder falls die nicht vorhanden ist Floppy 4
04 = SCSI-Harddisk
05 = IDE-Harddisk

Die Reihenfolge in der die Werte eingetragen werden, ist die Reihenfolge in der gesucht wird.

C = SER

Hiermit wird die "alte" SER-Karte konfiguriert. Die Werte sind identisch mit den Werten beim SIINIT Befehl (siehe Kapitel 4.2).

Beispiele:

$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

D = SER2 Kanal A

Die hier aufgeführten Werte werden in folgender Reihenfolge in die Register des UARTs für Kanal A der SER2 Baugruppe übertragen:

Baudrate
Parity, Anzahl Bits
Stopp Bits
Setze Extent Bit für Rx
Setze Extent Bit für Tx
Starte Kanal

Typische Einstellungen sind (jeweils ohne Parität, 8 Datenbits und 1 Stoppbit):

2400 Baud: $88, $13, $07, $90, $b0, $15
9600 Baud: $bb, $13, $07, $80, $a0, $15
19200 Baud: $cc, $13, $07, $80, $a0, $15
28800 Baud: $66, $13, $07, $80, $a0, $15
57600 Baud: $77, $13, $07, $80, $a0, $15
115000 Baud: $88, $13, $07, $80, $a0, $15

Weitere Einstellungen sind dem Datenblatt des XR88C681 zu entnehmen.

E = SER2 Kanal B

Diese Funktion ist identisch mit D, nur dass die Werte in die Register für Kanal B übertragen werde.

S = Speichern und Zurück

Nur mit dieser Funktion werden die obigen Werte in das NVRAM des Uhren-Chips gespeichert. Anschließend wird zum Hauptmenü zurückgekehrt.

Z = Zurück

Es wird zum Hauptmenü zurückgekehrt, hierbei werden keine Werte gespeichert.

3.1.23. W = Alter Text

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 werden. Es kann eine Adresse angewählt werden, auf der sich bereits ein Text befindet. Dieser "Alte Text" wird beim Aufruf nicht verändert und kann anschließend im Editor bearbeitet werden. Diese Funktion ist im Editor mit "ESC A" erreichbar.

3.1.24. X = Neuer Text

Auf der einzugebenden Adresse wird 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 "Neuer Text"-Adresse zerstört werden, erfolgt vor der Einrichtung des neuen Textstarts eine Sicherheitsabfrage. Diese Funktion ist im Editor mit "ESC N" erreichbar.

3.1.25. Y = 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.

3.1.26. Z = Zurück

Dieser Menüpunkt ist nur verfügbar, wenn das Grundprogramm als Unterprogramm aufgerufen wurde (siehe GRUND). Mit Eingabe von Z wird das Grundprogrammmenü beendet und zum aufrufenden Programm zurückgekehrt.

3.1.27. 1 = 40 Zeichen/Zeile

Wenn auf 40 Zeichen/Zeile umgeschaltet wird, erfolgen alle Bildschirmausgaben der Routinen CO2, CO und CHAR (Symbole ausgeben, Editor, Assembler) mit der größeren Schrift.

3.1.28. 2 = 80 Zeichen/Zeile

Alle Texte werden mit 80 Zeichen pro Zeile ausgegeben.

3.1.29. 3 = Debug-Info an

Die Debug-Infomation wird eingeschaltet.
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, 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).

3.1.30. 4 = Debug-Info aus

Die Debug-Infomation wird ausgeschaltet.

3.1.31. 5 = Nur Fehlerausgabe

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

3.1.32. 6 = Ausgabe auf CRT

Der Assembler gibt das gesamte Programm mit den dazugehörigen Adressen, Befehlen und Kommentaren auf den Bildschirm aus.

3.1.33. 7 = Ausgabe auf LST

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

3.1.34. 8 = Auf LST ohne LF

Die Ausgabe des Assemblers erfolgt auf dem Drucker, wobei aber das Zeichen $0A (Linefeed) unterdrückt wird, da einige Drucker es automatisch einfügen.

3.1.35. 9 = Uhr stellen

Wenn eine Uhren-Baugruppe im System vorhanden ist, kann sie mit dieser Funktion gestellt werden. Nach dem Aufruf steht der Cursor rechts im Feld des Wochentages. Zur Eingabe eines neuen Wertes muss man den alten erst mit BS (Backspace) löschen. Der Wochentag wird hier als Zahl eingegeben, also 0 = Montag, 1 = Dienstag usw. Eingaben werden mit Enter bestätigt.

4. Die Unterprogramme

Im Grundprogramm sind, wie schon oft erwähnt, eine ganze Menge von Programmen vorhanden, die für den Programmierer sehr leicht zu erreichen sind. Dabei ist im Assembler schon eine Funktion vorhanden, die ihn erkennen lässt, ob ein solches Unterprogramm aufgerufen werden soll. Die Namen dieser Programme sind nämlich in einer internen Tabelle abgelegt.

Sie können mit zwei verschiedenen Methoden aufgerufen werden.

1) Aufruf mit JSR @NAME

Die Unterprogramme haben, wie man leicht aus den Tabellen erkennt, alle einen bestimmten Namen. Durch diesen Namen kann der Assembler die Adresse des jeweiligen Programms ermitteln. Allerdings funktionieren die Programme dann eventuell nicht auf anderen Computern. Diese Methode ist nur gedacht, wenn man schnell ein kleines Programm schreiben will, das dann später nicht mehr benötigt wird. Außerdem muss darauf geachtet werden, dass das Adressregister A5.L mit dem richtigen Wert belegt ist. (Siehe Funktion SETA5)

Beispiel :

```
JSR @SCHREITE
```

2) 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.

Es wird bei dieser Methode im Register D7.W die Zahl abgelegt, die der Programmnummer entspricht. Dann wird der Befehl TRAP #1 ausgeführt. Beim Aufruf springt das Programm auf eine feste Adresse, wo das Grundprogramm die Zahl interpretiert und das gewünschte Programm aufruft. Dabei wird auch gleich das Register A5.L gesetzt. Den genauen Ablauf beim TRAP-Befehl kann man in einem Assembler-Bauch nachlesen.

Beispiel:

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

4.1. Tabellen der Unterprogramme

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

Zu beachten ist, dass beim Aufruf über TRAP zusätzlich immer die Register D7 und A6 zerstört werden. Der CPU-Status (USER- oder SUPERVISOR-Modus) bleibt beim Aufruf erhalten. Die Flags bleiben so erhalten, wie das Grundprogramm-Unterprogramm sie hinterlässt.

4.1.1 Sortierung nach TRAP-Nummer Teil 1

TRAP Nr.	Befehlsname	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	Schildkrötenneustart	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	Hexadezimale Ausgabe 2 Stellen (1 Byte)	3.1	Wertausgabe
42	PRINT4X	Hexadezimale Ausgabe 4 Stellen (2 Bytes)	3.1	Wertausgabe
43	PRINT6X	Hexadezimale Ausgabe 6 Stellen (3 Bytes)	3.1	Wertausgabe
44	PRINT8X	Hexadezimale 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
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 fest setzen (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 Arbeitsspeicher 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 Bildwiederholsp. lesen	4.0	Zeichenausgabe

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 Unterpr. 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 Standartwerte	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 dis-assemblieren	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	Harddisk
142	HARDTEST	Prüft, ob eine Harddisk vorhanden ist	6.2	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
149	SETFPZ	FP-Variable Z setzen	6.1	68020-FPU
150	SETSER	Auswahl des aktuellen SER Kanals	7.0	SER-Baugruppe
151	GETSER	Aktuellen SER Kanal feststellen	7.0	SER-Baugruppe
152	SETS2I	SCSI nach IDE Umleitung setzen / löschen	7.0	Harddisk

153	GETS2I	Status der SCSI nach IDE Umleitung lesen	7.0	Harddisk
154	IDETEST	Testet, ob ein IDE-Laufwerk vorhanden ist	7.0	Harddisk
155	IDEDISK	Befehl an die IDE-Disk geben und ausführen	7.0	Harddisk
156	SRDISK	Befehl an die SRAMDISK geben u. ausführen	7.0	FLO-Baugruppe
157	SETF2S	Flo4 nach SRD Umleitung setzen / löschen	7.0	FLO-Baugruppe
158	GETF2S	Flo4 nach SRAMDISK Umleitung lesen	7.0	FLO-Baugruppe
159	GETSRD	Größe der SRAMDISK abfragen	7.0	FLO-Baugruppe
160	SETSYS	Systemeinstellungen ins NVRAM speichern	7.0	System-Routinen
161	GETSYS	Systemeinstellungen aus NVRAM holen	7.0	System-Routinen
162	PATCH	Trap-Aufrufe umleiten/definieren	7.0	System-Routinen

4.1.2. Sortierung nach TRAP-Nummer Teil 2

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
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	KEINE
12	CI	KEINE	d0.l	KEINE
13	CSTS	KEINE	d0.l/Flags	KEINE
14	RI	KEINE	d0.l/Carry/Flags	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	KEINE
24	COS	d0.w	d0.w	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/Carry/(d0.l/d1.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
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
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	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	KEINE
124	GRUND	d0.w	KEINE	KEINE
125	HARDCOPY	Verschiedene Ein- und Aus	gaberegister	d7
126	GRAFIK	Verschiedene Ein- und Aus	gaberegister	d7/a6
127	GDPVERS	KEINE	d0.l	KEINE
128	SER	d0.b	KEINE	KEINE
129	CO2SER	KEINE	KEINE	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/d3/a0.l/a1.l)	Carry(a1.l/(d1-d7))	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.l)	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
150	SETSER	d0.b	d0	KEINE
151	GETSER	KEINE	d0.b	KEINE

152	SETS2I	d0.b	d0	KEINE
153	GETS2I	KEINE	d0.b	KEINE
154	IDETEST	d4.b	d0.l/a0.l/Carry	KEINE
155	IDEDISK	d1.b/d4.b/(d2/d3/a0.l)	d0.l/Carry	KEINE
156	SRDISK			KEINE
157	SETF2S	d0.b	KEINE	KEINE
158	GETF2S	KEINE	d0.b	KEINE
159	GETSRD	KEINE	d0	KEINE
160	SETSYS	a0.l	d0	KEINE
161	GETSYS	a0.l	d0/a0.l	KEINE
162	PATCH	d0/a0.l	d0/a0.l	KEINE

4.1.3. Sortierung nach dem Namen Teil 1

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

54	GETERR	Fehlercode lesen	3.1	Assembler
158	GETF2S	Flo4 nach SRAMDISK Umleitung lesen	7.0	FLO-Baugruppe
122	GETFLOAT	FP-Zahl in ASCII in BCD wandeln	5.0	68020-FPU
76	GETFLOP	FLOPPY-Format feststellen	4.0	FLO-Baugruppe
95	GETK	Schildkrötenrichtung lesen	4.0	Schildkrötengrafik
100	GETLINE	Zeilenadresse Bildwiederholsp. lesen	4.0	Zeichenausgabe
87	GETNEXT	Offset für Symboltabelle lesen	4.0	Symboltabelle
68	GETORG	Default-ORG-Adresse lesen	3.1	Assembler
59	GETRAM	Ram-Bereich testen	3.1	System-Routinen
153	GETS2I	Status der SCSI nach IDE Umleitung lesen	7.0	Harddisk
151	GETSER	Aktuellen SER Kanal feststellen	7.0	SER-Baugruppe
98	GETSN	Seriennummer Grundprogramm lesen	4.0	System-Routinen
159	GETSRD	Größe der SRAMDISK abfragen	7.0	FLO-Baugruppe
66	GETSTX	Textstart-Adresse lesen	3.1	Editor
86	GETSYM	Symboltabellenstart lesen	4.0	Symboltabelle
161	GETSYS	Systemeinstellungen aus NVRAM holen	7.0	System-Routinen
115	GETUHR	Uhrzeit und Datum lesen	4.0	Echtzeituhren
90	GETVAR	Adresse des Arbeitsspeicher lesen	4.0	System-Routinen
97	GETVERS	Versionsnummer Grundprogramm lesen	4.0	System-Routinen
78	GETXOR	XOR-Modus GDP lesen	4.0	Grafik
103	GETXY	Aktuelle GDP-Position lesen	4.0	Grafik
126	GRAFIK	Großes GRAFIK-Paket für GDP und COL	6.0	Grafik
39	GRAPOFF	Schildkrötengrafik ausschalten	3.1	Schildkrötengrafik
124	GRUND	Grundprogramm als Unterpr. aufrufen	6.0	System-Routinen
125	HARDCOPY	Ansteuerung der HARDCOPY-Baugruppe	6.0	Grafik
141	HARDDISK	Befehl an Harddisk geben und ausführen	6.2	Harddisk
142	HARDTEST	Prüft, ob eine Harddisk vorhanden ist	6.2	Harddisk
3	HEBE	Schildkrötenspur ausschalten	3.1	Schildkrötengrafik
47	HIDE	Schildkröte ausblenden	3.1	Schildkrötengrafik
155	IDEDISK	Befehl an die IDE-Disk geben und ausführen	7.0	Harddisk
154	IDETEST	Testet, ob ein IDE-Laufwerk vorhanden ist	7.0	Harddisk
93	KORXY	Schildkrötenkoordinaten lesen	4.0	Schildkrötengrafik
22	LO	Zeichen über Centronics ausgeben	3.1	Zeichenausgabe
50	LST	CO2 auf Drucker lenken	3.1	Zeichenausgabe
117	LSTS	Drucker bereit ?	4.0	Zeichenausgabe
8	MOVETO	Position GDP setzen	3.1	Grafik
72	MULS32	32-Bit Multiplikation	4.0	Berechnungen
27	NEWPAGE	Schreibseite und Leseseite GDP setzen	3.1	Grafik
52	NIL	CO2 ins Leere lenken	3.1	Zeichenausgabe
162	PATCH	Trap-Aufrufe umleiten/definieren	7.0	System-Routinen
15	PO	Zeichen über CAS schreiben	3.1	CAS-Baugruppe
41	PRINT2X	Hexadezimale Ausgabe 2 Stellen (1 Byte)	3.1	Wertausgabe
46	PRINT4D	Dezimale Ausgabe ohne Vorzeichen	3.1	Wertausgabe
42	PRINT4X	Hexadezimale Ausgabe 4 Stellen (2 Bytes)	3.1	Wertausgabe
43	PRINT6X	Hexadezimale Ausgabe 6 Stellen (3 Bytes)	3.1	Wertausgabe
45	PRINT8B	Binäre Ausgabe 8 Bit (1 Byte)	3.1	Wertausgabe
70	PRINT8D	Dezimale Ausgabe ohne Vorzeichen	4.0	Wertausgabe
44	PRINT8X	Hexadezimale Ausgabe 8 Stellen (4 Bytes)	3.1	Wertausgabe
121	PRINTFP0	Wert in FP0 in ASCII wandeln	5.0	68020-FPU
71	PRINTV8D	Dezimale Ausgabe mit Vorzeichen	4.0	Wertausgabe
64	PROGZGE	Frei definierbares Zeichen ausgeben	3.1	Textausgabe
145	PRTFP0	Wert in FP0 in ASCII wandeln	6.1	68020-FPU

84	PRTSYM	Symboltabelle ausgeben	4.0	Symboltabelle
88	PUTNEXT	Offset für Symboltabelle setzen	4.0	Symboltabelle
69	PUTORG	Default-ORG-Adresse setzen	3.1	Assembler
67	PUTSTX	Textstart-Adresse setzen	3.1	Editor
11	READ	Text vom Bildschirm einlesen	3.1	Texteingabe
123	READAUS	Texteingabe mit vorheriger Ausgabe	6.0	Texteingabe
132	RELAIS	Relais auf RELAIS-Baugruppe setzen	6.0	RELAIS-Baugruppe
133	RELAISIN	Port der RELAIS-Baugruppe lesen	6.0	RELAIS-Baugruppe
118	RELAN	Relais auf CAS2 einschalten	4.0	CAS-Baugruppe
119	RELAUS	Relais auf CAS2 ausschalten	4.0	CAS-Baugruppe
14	RI	Zeichen von CAS lesen	3.1	CAS-Baugruppe
96	RND	Zufallsgenerator	4.0	Berechnungen
19	SCHR16TEL	Schildkröte schreitet	3.1	Schildkrötengrafik
1	SCHREITE	Schildkröte schreitet	3.1	Schildkrötengrafik
4	SENKE	Schildkrötenspur einschalten	3.1	Schildkrötengrafik
128	SER	CI, LO oder FLOPPY auf SER lenken	6.0	SER-Baugruppe
7	SET	Schildkröte absolut setzen	3.1	Schildkrötengrafik
91	SETA5	RAM-Zeiger A5 wieder gültig setzen	4.0	System-Routinen
79	SETCOLOR	Farbcode GDP setzen	4.0	Grafik
102	SETCURXY	Cursorposition setzen	4.0	Zeichenausgabe
111	SETDA	DA-Wandler setzen	4.0	AD/DA-Baugruppen
134	SETDA12	DA-Wandler auf AD/DA-Baugruppe setzen	6.0	AD/DA-Baugruppen
53	SETERR	Fehlercode setzen	3.1	Assembler
157	SETF2S	Flo4 nach SRD Umleitung setzen / löschen	7.0	FLO-Baugruppe
58	SETFIG	Figur fest setzen (einfrieren)	3.1	Figurgrafik
34	SETFLIP	Umschaltrate für Autoflip setzen	3.1	Grafik
147	SETFPX	FP-Variable X setzen	6.1	68020-FPU
148	SETFPY	FP-Variable Y setzen	6.1	68020-FPU
149	SETFPZ	FP-Variable Z setzen	6.1	68020-FPU
55	SETPASS	Laufnummer setzen	3.1	Zeichenausgabe
37	SETPEN	Schreiber in GDP auf SCHREIBEN	3.1	Grafik
152	SETS2I	SCSI nach IDE Umleitung setzen / löschen	7.0	Harddisk
150	SETSER	Auswahl des aktuellen SER Kanals	7.0	SER-Baugruppe
160	SETSYS	Systemeinstellungen ins NVRAM speichern	7.0	System-Routinen
116	SETUHR	Uhrzeit und Datum setzen	4.0	Echtzeituhren
77	SETXOR	XOR-Modus GDP setzen	4.0	Grafik
48	SHOW	Schildkröte einblenden	3.1	Schildkrötengrafik
104	SI	Zeichen von serieller Schnittstelle	4.0	SER-Baugruppe
138	SI2	Zeichen von serieller Schnittstelle	6.1	SER-Baugruppe
108	SIINIT	SER initialisieren (Baudrate, Parität)	4.0	SER-Baugruppe
23	SIN	Sinus*256 berechnen	3.1	Berechnungen
106	SISTS	Zeichen von SER da ?	4.0	SER-Baugruppe
25	SIZE	Zeichengröße einstellen (für CO2..)	3.1	Zeichenausgabe
105	SO	Zeichen an serielle Schnittstelle	4.0	SER-Baugruppe
107	SOSTS	SER bereit zur Ausgabe ?	4.0	SER-Baugruppe
114	SOUND	SOUND-Generator setzen	4.0	Sprache und Sound
112	SPEAK	Sprachausgabe mit 5 Parametern	4.0	Sprache und Sound
113	SPEAK1	Sprachausgabe nur Phonemcode	4.0	Sprache und Sound
156	SRDISK	Befehl an die SRAMDISK geben u. ausführen	7.0	FLO-Baugruppe
137	SUCHBIBO	Bibliothekseinträge suchen	6.1	System-Routinen
85	SYMCLR	Symboltabelle löschen	4.0	Symboltabelle
28	SYNC	Vertikal-Synchronimpuls GDP abfragen	3.1	Synchronisation

139	SYSTEM	System-Informationen lesen	6.1	System-Routinen
140	UHRPRINT	Uhrzeit und Datum in ASCII wandeln	6.2	Echtzeituhren
51	USR	CO2 auf Benutzerschnittstelle lenken	3.1	Zeichenausgabe
18	WAIT	Warten, bis GDP fertig	3.1	Grafik
29	WERT	Ganzzahligen Wert berechnen	3.1	Berechnungen
10	WRITE	Text auf dem Bildschirm ausgeben	3.1	Textausgabe
6	WRITELF	Text auf dem Bildschirm ausgeben	6.0	Textausgabe
30	ZUWEIS	Symbol in Symboltabelle eintragen	3.1	Symboltabelle
143		RESERVIERT		
144		RESERVIERT		

4.1.4. Sortierung nach dem Namen Teil 2

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

101	GETCURXY	KEINE	d1.l/d2.l	KEINE
54	GETERR	KEINE	d0.w	KEINE
158	GETF2S	KEINE	d0.b	KEINE
122	GETFLOAT	a0.l/a1.l	a0.l/Carry	KEINE
76	GETFLOP	d4.b	d0.w/d1.b/d4.w/Carry	KEINE
95	GETK	KEINE	d0.w	KEINE
100	GETLINE	d0.b	a0.l-a2.l	d0
87	GETNEXT	KEINE	d0.l	KEINE
68	GETORG	KEINE	d0.l	KEINE
59	GETRAM	a0.l	d0.l/a0.l/a1.l/Carry	KEINE
153	GETS2I	KEINE	d0.b	KEINE
151	GETSER	KEINE	d0.b	KEINE
98	GETSN	KEINE	d0.l	KEINE
159	GETSRD	KEINE	d0	KEINE
66	GETSTX	KEINE	d0.l	KEINE
86	GETSYM	KEINE	d0.l/a0.l	KEINE
161	GETSYS	a0.l	d0/a0.l	KEINE
115	GETUHR	a0.l	a0.l	KEINE
90	GETVAR	KEINE	d0.l/a0.l	KEINE
97	GETVERS	KEINE	d0.l	KEINE
78	GETXOR	KEINE	d0.l	KEINE
103	GETXY	KEINE	d1.l/d2.l	KEINE
126	GRAFIK	Verschiedene Ein- und Aus	gaberegister	d7/a6
39	GRAPOFF	KEINE	KEINE	KEINE
124	GRUND	d0.w	KEINE	KEINE
125	HARDCOPY	Verschiedene Ein- und Aus	gaberegister	d7
141	HARDDISK	d1.b/d4.b/(d2/d3/a0.l)	d0.l/Carry	KEINE
142	HARDTEST	d4.b	d0.l/Carry	KEINE
3	HEBE	KEINE	KEINE	KEINE
47	HIDE	KEINE	d0.b	KEINE
155	IDEDISK	d1.b/d4.b/(d2/d3/a0.l)	d0.l/Carry	KEINE
154	IDETEST	d4.b	d0.l/a0.l/Carry	KEINE
93	KORXY	KEINE	d1.l/d2.l	KEINE
22	LO	d0.b	KEINE	KEINE
50	LST	KEINE	KEINE	KEINE
117	LSTS	KEINE	d0.l/Flags	KEINE
8	MOVETO	d1.w/d2.w	KEINE	KEINE
72	MULS32	d0.l/d2.l	d0.l/d1.l	KEINE
27	NEWPAGE	d0.b/d1.b	d0.b	KEINE
52	NIL	KEINE	KEINE	KEINE
162	PATCH	d0/a0.l	d0/a0.l	KEINE
15	PO	d0.b	Carry	KEINE
41	PRINT2X	d0.b/a0.l	a0.l	KEINE
46	PRINT4D	d0.w/a0.l	a0.l	d0
42	PRINT4X	d0.w/a0.l	a0.l	KEINE
43	PRINT6X	d0.l/a0.l	a0.l	KEINE
45	PRINT8B	d0.b/a0.l	a0.l	KEINE
70	PRINT8D	d0.l/a0.l	a0.l	d0
44	PRINT8X	d0.l/a0.l	a0.l	KEINE
121	PRINTFP0	d0.w/a0.l/fp0.x	a0.l	KEINE
71	PRINTV8D	d0.l/a0.l	a0.l	d0
64	PROGZGE	a0.l	KEINE	KEINE

145	PRTFP0	d0.w/a0.l/fp0.x	a0.l/Carry	KEINE
84	PRTSYM	KEINE	KEINE	KEINE
88	PUTNEXT	d0.w	KEINE	KEINE
69	PUTORG	d0.l	KEINE	KEINE
67	PUTSTX	d0.l	a0.l	KEINE
11	READ	d0.b/d1.w-d3.w/a0.l	d4.l/d5.l/a0.l	KEINE
123	READAUS	d0.b/d1.w-d3.w/a0.l	d4.l/d5.l/a0.l	KEINE
132	RELAIS	d0.b/a0.l	a0.l	KEINE
133	RELAISIN	a0.l	d0.b/a0.l	KEINE
118	RELAN	KEINE	KEINE	KEINE
119	RELAUS	KEINE	KEINE	KEINE
14	RI	KEINE	d0.l/Carry/Flags	KEINE
96	RND	d0.w	d0.l	KEINE
19	SCHR16TEL	d0.w	KEINE	d0
1	SCHREITE	d0.w	KEINE	d0
4	SENKE	KEINE	KEINE	KEINE
128	SER	d0.b	KEINE	KEINE
7	SET	d1.w-d3.w	d0.w	KEINE
91	SETA5	KEINE	a5.l	KEINE
79	SETCOLOR	d0.b	d0.b	KEINE
102	SETCURXY	d1.b/d2.b	KEINE	KEINE
111	SETDA	d1.b/d2.b	KEINE	KEINE
134	SETDA12	d0.w/d1.w	KEINE	KEINE
53	SETERR	d0.w	KEINE	KEINE
157	SETF2S	d0.b	KEINE	KEINE
58	SETFIG	KEINE	KEINE	KEINE
34	SETFLIP	d0.b/d1.b	KEINE	KEINE
147	SETFPX	fp0.x	KEINE	KEINE
148	SETFPY	fp0.x	KEINE	KEINE
149	SETFPZ	fp0.x	KEINE	KEINE
55	SETPASS	d0.w	KEINE	KEINE
37	SETPEN	KEINE	KEINE	KEINE
152	SETS2I	d0.b	d0	KEINE
150	SETSER	d0.b	d0	KEINE
160	SETSYS	a0.l	d0	KEINE
116	SETUHR	a0.l	a0.l	KEINE
77	SETXOR	d0.w	d0.b	KEINE
48	SHOW	KEINE	d0.b	KEINE
104	SI	KEINE	d0.l	KEINE
138	SI2	KEINE	d0.l	KEINE
108	SIINIT	d0.b/d1.b	KEINE	KEINE
23	SIN	d0.w	d0.w	KEINE
106	SISTS	KEINE	d0.l/Flags	KEINE
25	SIZE	d0.b	KEINE	KEINE
105	SO	d0.b	KEINE	KEINE
107	SOSTS	KEINE	d0.l/Flags	KEINE
114	SOUND	a0.l	KEINE	KEINE
112	SPEAK	a0.l	KEINE	KEINE
113	SPEAK1	a0.l	KEINE	KEINE
156	SRDISK			KEINE
137	SUCHBIBO	d0.w/(d2/d3/a0.l/a1.l)	Carry(a1.l/(d1-d7))	d0/(d1-d2)
85	SYMCLR	KEINE	a0.l	KEINE

28	SYNC	KEINE	d0.w/Flags	KEINE
139	SYSTEM	KEINE	d0.l	KEINE
140	UHRPRINT	d0.w/a0.l	a0.l/Carry	KEINE
51	USR	KEINE	KEINE	KEINE
18	WAIT	KEINE	KEINE	KEINE
29	WERT	a0.l	d0.l/d1.l/a0.l/Carry	d2/d3/a1-a3
10	WRITE	d0.b/d1.w/d2.w/a0.l	KEINE	KEINE
6	WRITELF	d0.b/d1.w/d2.w/a0.l	KEINE	KEINE
30	ZUWEIS	a0.l	a0.l/Carry/(d0.l/d1.l)	d2-d3/a1-a2
143		RESERVIERT		
144		RESERVIERT		

4.1.5. Sortierung nach Gruppen und TRAP-Nummer

Gruppe1: AD/DA-Baugruppen

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
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.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
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.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
23	SIN	d0.w	d0.w	KEINE
24	COS	d0.w	d0.w	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.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
14	RI	KEINE	d0.l/Carry/Flags	KEINE
15	PO	d0.b	Carry	KEINE
118	RELAN	KEINE	KEINE	KEINE
119	RELAUS	KEINE	KEINE	KEINE

Gruppe5: CLUT-Baugruppe

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
130	CLUTINIT	KEINE	KEINE	KEINE
131	CLUT	a0.l	KEINE	KEINE

Gruppe6: DIS-Assembler

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
136	DISASS	d0.l/a0.l	KEINE	KEINE

Gruppe7: Echtzeituhren

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
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.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
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.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
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.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
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
156	SRDISK			KEINE
157	SETF2S	d0.b	KEINE	KEINE
158	GETF2S	KEINE	d0.b	KEINE
159	GETSRD	KEINE	d0	KEINE

Gruppe11: Grafik

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
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	d0.b	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 Aus	gaberegister	d7
126	GRAFIK	Verschiedene Ein- und Aus	gaberegister	d7/a6
127	GDPVERS	KEINE	d0.l	KEINE

Gruppe12: RELAIS-Baugruppe

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
132	RELAIS	d0.b/a0.l	a0.l	KEINE
133	RELAISIN	a0.l	d0.b/a0.l	KEINE

Gruppe13: Schildkrötengrafik

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
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: Harddisk

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
141	HARDDISK	d1.b/d4.b/(d2/d3/a0.l)	d0.l/Carry	KEINE
142	HARDTEST	d4.b	d0.l/Carry	KEINE
152	SETS2I	d0.b	d0	KEINE
153	GETS2I	KEINE	d0.b	KEINE
154	IDETEST	d4.b	d0.l/a0.l/Carry	KEINE
155	IDEDISK	d1.b/d4.b/(d2/d3/a0.l)	d0.l/Carry	KEINE

Gruppe15: SER-Baugruppe

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
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	KEINE	KEINE
138	SI2	KEINE	d0.l	KEINE
150	SETSER	d0.b	d0	KEINE
151	GETSER	KEINE	d0.b	KEINE

Gruppe16: Sprache und Sound

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
112	SPEAK	a0.l	KEINE	KEINE
113	SPEAK1	a0.l	KEINE	KEINE
114	SOUND	a0.l	KEINE	KEINE

Gruppe17: Symboltabelle

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
30	ZUWEIS	a0.l	a0.l/Carry/(d0.l/d1.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.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
28	SYNC	KEINE	d0.w/Flags	KEINE
35	DELAY	d0.l	KEINE	d0

Gruppe19: System-Routinen

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
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/d3/a0.l/a1.l)	Carry(a1.l/(d1-d7))	d0/(d1-d2)
139	SYSTEM	KEINE	d0.l	KEINE
160	SETSYS	a0.l	d0	KEINE
161	GETSYS	a0.l	d0/a0.l	KEINE
162	PATCH	d0/a0.l	d0/a0.l	KEINE

Gruppe20: Textausgabe

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
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.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
11	READ	d0.b/d1.w-d3.w/a0.l	d4.l/d5.l/a0.l	KEINE
123	READAUS	d0.b/d1.w-d3.w/a0.l	d4.l/d5.l/a0.l	KEINE

Gruppe22: Wertausgabe

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
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.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
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	KEINE
129	CO2SER	KEINE	KEINE	KEINE

Gruppe24: Zeicheneingabe

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
12	CI	KEINE	d0.l	KEINE
13	CSTS	KEINE	d0.l/Flags	KEINE
31	CIINIT2	KEINE	KEINE	KEINE
32	CI2	KEINE	d0.l/Carry	KEINE

Gruppe25: 68020-FPU

TRAP Nr.	Befehlsname	Eingabe-Register	Ausgabe-Register	Zerstörte Reg.
121	PRINTFP0	d0.w/a0.l/fp0.x	a0.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 sehr nützliches Mittel, um Programme einfach schreiben zu können. Sie sind fest im Eprom vorhanden und dadurch natürlich 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 Revisionsänderung 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 ist nach einem einheitlichen Schema aufgebaut. Jedes Unterprogramm hat einen Kopf, durch den die wichtigsten Informationen sehr schnell abgelesen werden können. Jeder Kopf enthält folgende Informationen:

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, so sollten sie immer auf Null gesetzt sein.

Ausgaberegister: Alle Ausgaberegister sind unter diesem Punkt aufgeführt. Zwei zusätzliche Informationen sind das Carry-Flag, das z.B. angeben kann, ob ein Programm richtig ausgeführt wurde, oder ob ein Fehler auftrat. Durch das Carry-Flag können direkt nach Unterprogrammende Sprünge mit den Befehlen BCC oder BCS durchgeführt werden, wodurch sich eventuell 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, so bedeutet dies, 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 dem Punkt "Beschreibung".

Zerstörte Register: Hier sind alle Register aufgeführt, die durch das Unterprogramm zerstört werden können. Diese Register sollten also vorm Programmaufruf gerettet werden, falls sie noch benötigt werden. Hier erfolgt keine Längenangabe. Es muss damit gerechnet werden, dass das gesamte Langwort zerstört werden könnte. Die Angaben über die zerstörten Register gelten erst ab Version 6.2. In vorherigen Versionen können eventuell auch mehr Register zerstört werden.

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

Änderungen zu 4.3:
Änderungen zu 6.1:
Änderungen zu 6.3: Wenn Änderungen zu vorherigen Grundprogrammversionen vorhanden sind, die die Kompatibilität beeinträchtigen könnten, sind sie hier aufgeführt. Wenn zur Version 6.3 Änderungen vorhanden sind, so gelten diese auch für die Version 6.2, 6.1, 6.0 und 4.3.

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

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

Die Beispiele sind sehr einfach gehalten. Für die Unterprogrammaufrufe wird immer die TRAP-Funktion gewählt. Auch wenn nur ein Byte in ein Register geladen werden müsste, so wird oftmals 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 diesen Fällen eigentlich nicht nötig ist.

In der Zeitschrift LOOP sind bereits sehr viele kleine Programme erschienen, die die Funktionen des Grundprogramms ausnutzen. Für größere Programme sollte man sich diese Beispiele ansehen. Außerdem ist das Listing des Grundprogramms auf Diskette erhältlich, wodurch man alle Informationen der einzelnen Unterprogramme direkt ablesen kann. Da das Listing kommentiert ist, kann man die Programme auch leicht verstehen. Außerdem gibt es zur 68000-Serie bereits viele Bücher, die die Programmiersprache ASSEMBLER leicht verständlich erklären. Ein Verständnis der Programmiersprache ist Vorraussetzung für den direkten Umgang mit den Grundprogramm-Routinen.

TRAP-Nummer: 1
Befehlsname: SCHREITE
Befehlgruppe: 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
Änderungen zu 6.3: 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)

Beschreibung :

Der Befehl SCHREITE bewegt die Schildkröte auf dem Bildschirm in die Richtung, in die sie gerade 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
```

Komplexes 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 Bereich. Der Wert sollte aber 4096 nicht überschreiten, sonst erscheint die Schildkröte wieder im Bildbereich.

Bei allen Befehlen der Schildkrötengraphik, werden automatisch zwei Bildseiten verwendet. Diese beiden Bildseiten werden alle 20ms umgeschaltet. Auf der Bildseite 0 wird die Graphik dargestellt und auf der Bildseite 1 die Schildkröte, die im folgenden auch manchmal Zeiger genannt wird. Der Zeiger besteht dabei aus einem Dreieck, dessen Spitze in die aktuelle Blickrichtung 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 speziellen 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
Änderungen zu 6.3:	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)		

Beschreibung:

Die Schildkröte wird entsprechend dem Wert in D0.W gedreht. Eine positive Zahl 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      * Uhrzeiger drehen.
        TRAP    #1
        RTS
```

Die Schildkröte dreht sich um 30 Grad im Uhrzeigersinn.

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

Komplexes 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
        MOVE    #!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:

Wird ein Winkel angegeben, der größer als 359 Grad oder kleiner als 0 Grad ist, so wird er 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 des Zeigers bezieht, nicht dagegen auf den Bildschirmrand. Dabei gehorcht der Zeiger der in LOGO üblichen Weise.

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
Änderungen zu 6.3:	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)

Beschreibung:

Die Schildkröte hinterlässt ab sofort keine Schreibspur mehr beim Bewegen. Damit kann man isolierte Figuren zeichnen. Der Befehl gilt nur für die Schildkröte nicht für Grafik allgemein.

Zu dem Befehl gehört auch das Gegenstück SENKE.

Einfaches Beispiel:

Nach diesem Programm hinterlässt die Schildkröte keine Schreibspur mehr.

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

Komplexes 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
Änderungen zu 6.3:	NEIN

Siehe auch:	SCHREITE(1)	DREHE(2)	HEBE(3)
	SET(7)	SCHR16TEL(19)	FIRSTTIME(36)
	GRAPOFF(39)	HIDE(47)	SHOW(48)
	AUFXY(92)	KORXY(93)	AUFK(94)
	GETK(95)		

Beschreibung:

Der Schreibstift wird gesenkt. Von da an wird eine Schreibspur hinterlassen, wenn sich der Zeiger (sprich: 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
```

Komplexes 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
Änderungen zu 6.3:	NEIN
Siehe auch:	FIGUR(57) SETFIG(58)

Beschreibung:

Dieser Befehl entspricht in seiner Funktion dem Befehl FIGUR. Er hat aber den großen 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 1 bis 256 hat. Dy ist die Y-Vergrößerung mit dem selben Bereich.

Beispiel:

Dx = 5
Dy = 9

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

Die hexadezimale Darstellung hat den großen Vorteil, dass man die Vergrößerung sofort erkennen kann.

Da die Funktion des Befehls ansonsten mit dem FIGUR-Befehl gleich ist, wird auf diese Funktion (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 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    0,0,2,4,4,6,10  * Rechteck
```

Komplexes Beispiel:

Ein Rechteck wird erst in X- und dann in Y-Richtung gedehnt. Dies ist mit dem FIGUR-Befehl nicht möglich. Auch hier wird immer die alte Figur zuerst 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 (16,67 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 (16,67 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
```

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
Änderungen zu 6.3:	NEIN
Siehe auch:	WRITE(10) PROGZGE(64)

Beschreibung:

Die Funktion ist fast identisch mit dem Befehl WRITE. Die beiden Ausnahmen sind:

1.) Der Text wird nicht vorgelöscht, weshalb man nur auf leere Bildteile 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 sehr leicht ausgegeben werden, ohne dass der WRITE-Befehl mehrmals hintereinander aufgerufen werden 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 nähere Informationen gilt 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 auf 360-Grad-Bereich beschränkt
Zerstörte Register:	KEINE
Ab Version:	3.1
Änderungen zu 4.3:	NEIN
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	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)		

Beschreibung:

Damit kann die Schildkröte direkt auf eine absolute Position gesetzt werden. Die Schildkröte 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)

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
```

Komplexes 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
        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
Änderungen zu 6.3:	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)		

Beschreibung:

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

Als Parameter steht in Register D1.W die X-Koordinate. Dabei ist die Schreibstelle auf dem Bildschirm sichtbar, wenn die Koordinate im Bereich 0 bis 511 liegt, sonst zeichnet er im Unsichtbaren. Der gesamte Bildpunktraum ist 4096 Punkte groß. Also entspricht die Koordinate X=4096 der Koordinate 0 und X=4097 dem Wert 1.

Im Register D2.W steht die Y-Koordinate. Dabei ist hier der sichtbare Bereich von 0 bis 255 und es gibt ebenfalls 4096 verschiedene Koordinaten. Negative Werte sind auch 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
```

Komplexes Beispiel:

Der Bildschirm wird mit weißer Farbe 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
        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
Änderungen zu 6.3:	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)		

Beschreibung:

Der Graphik-Prozessor zeichnet eine Linie von der zuletzt eingestellten Position zu der neuen Position, die beim DRAWTO-Befehl angegeben wird.

Register D1.W erhält dazu 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              * Linie von X=0
        CLR     D2              * und Y=0
        MOVEQ   #!MOVETO,D7     * Startpunkt festlegen
        TRAP    #1
        MOVE    #511,D1         * bis X=511
        MOVE    #255,D2         * und Y=255
        MOVEQ   #!DRAWTO,D7     * ergibt Diagonale
        TRAP    #1
        RTS
```

Komplexes 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    #360-1,D3       * 360 Grad ist ein Kreis
SCHLEIFE:
        MOVE    D3,D0
        MOVEQ   #!SIN,D7        * Sinus ist X-Koordinate
        TRAP    #1
        MOVE    D0,D1
        ADD     #256,D1         * In Bildschirmmitte
        MOVE    D3,D0
        MOVEQ   #!COS,D7        * Cosinus ist Y-Koordinate
        TRAP    #1
        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.

Will man eine Linie löschen, so kann man mit dem Befehl ERAPEN die Schreibart "Löschen" vorwählen. Mit SETPEN wird sie wieder in den Normalzustand gesetzt.

TRAP-Nummer:	10
Befehlsname:	WRITE
Befehlsgruppe:	Textausgabe
Kurzbeschreibung:	Text mit frei wählbarer Größe auf dem 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
Änderungen zu 6.3:	NEIN
Siehe auch:	WRITELF(6) PROGZGE(64)

Beschreibung:

Ein Befehl, mit dem man Texte auf dem Bildschirm ausgeben kann. Dazu wird im Register D0.B die Textgröße festgelegt, in Register D1.W die X-Koordinate (0 bis 511) und in 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.

1.) 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 (Mit Abstand zur nächsten Zeile).

Wenn Dx oder Dy den Maximalwert von 16 annehmen sollen, so muss man anstelle der Zahl 16 den Wert 0 einsetzen. Dies ist ein Sonderfall, und ist vom Hersteller des Graphikprozessors (EF 9366) so bestimmt.

Beispiel:

Dx = 1
Dy = 1

=> g = 1*16 + 1

Also ist g=16. Man kann den Wert auch hexadezimal darstellen, es ergibt sich g=$11. Dort 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).

2.) 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 ganzen Textzeilen kann man dann ganz leicht durch mehrere WRITE-Befehle erreichen, deren Y-Koordinaten sich unterscheiden. Eigene Experimente verschaffen aber wohl schnell Aufschluss. Eine weitere Möglichkeit mehrere Zeilen auszugeben, ist der WRITELF-Befehl, der oben erklärt 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, so kann die Fehlermeldung "Nicht auf Wortgrenze" erscheinen. Dann muss man den Befehl

```
DS 0
```

vor den neuen Befehl schreiben. Damit wird automatisch ein leeres Byte eingefügt, wenn die nachfolgende Adresse ungerade sein sollte. 680XX-Befehle dürfen nämlich nur auf geraden Adressen zu liegen kommen.

Diese Angaben über Textablage und Endekennungen gelten auch für andere Unterprogramme. Eine ausführliche Beschreibung der Befehle DC und DS liefert das Kapitel 3.2.2. (Assembler).

TRAP-Nummer:	11
Befehlsname:	READ
Befehlsgruppe:	Texteingabe
Kurzbeschreibung:	Liest einen Text mit maximaler Länge 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 = Wirkliche Anzahl Zeichen d5.l = Letztes eingegebenes Zeichen a0.l = Adresse der Endekennung
Zerstörte Register:	KEINE
Ab Version:	3.1
Änderungen zu 4.3:	Die Eingabe kann nur durch ein CR (Carriage Return) oder durch Vollschreiben des Fensters beendet werden. Jedes Controlzeichen, das nicht zur Cursorsteuerung benutzt wird, wird ignoriert. Es sind D4.L und D5.L gültig.
Änderungen zu 6.1:	Wenn CR gedrückt wird, so 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.
Änderungen zu 6.3:	NEIN
Siehe auch:	READAUS(123)

Beschreibung:

Einlesen eines Textes von der Tastatur. Damit kann man Eingabefelder erzeugen, wie sie auch im Grundprogramm, z.B. zur Eingabe von "Adresse" oder "Name" 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. Damit wird auch die Breite des Textfensters bestimmt.
Das Register A0.l schließlich erhält die Adresse des Textbuffers, in demnach dem Aufruf der eingegebene Text abgelegt sein wird.

Nach dem Aufruf steht in Register D4.L die wirkliche Anzahl der eingegebenen 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 eingeben wurden, als in Register D3.W als Maximum zulässig waren. Im Buffer steht an letzter Stelle eine 0 (Code 00), um das Ende des eingegebenen Textes einfach erkennen zu können. Daher muss für den Buffer immer ein Byte mehr reserviert werden, als es die Anzahl in D3.W angibt. A0.L zeigt nach dem Aufruf auf dieses Endekennung.

Das Textfenster kann mit vielen Befehlen, die denen des Editors entsprechen, bearbeitet werden. Dadurch ist eine sehr komfortable Eingabe möglich.

Folgende Kommandos gibt es :

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 hinten geschoben. Das letzte Zeichen verschwindet.
Ctrl-V:	Der automatische Einfügemodus wird an- oder ausgeschaltet. Der Einfügemodus hängt mit dem des Editors zusammen. Wenn also hier umgeschaltet wird, so wird auch für den Editor umgeschaltet und umgekehrt.
Ctrl-G:	Das Zeichen an der Cursorposition wird gelöscht und alle rechts liegenden Zeichen rücken auf.

DEL:	Das Zeichen links des Cursors wird gelöscht und alle Zeichen ab der Cursorposition rücken auf.
Ctrl-T:	Alle Zeichen ab der Cursorposition werden gelöscht.
Ctrl-A:	Der Cursor wird am Textanfang positioniert.
Ctrl-F:	Der Cursor wird hinter dem letzten Zeichen positioniert.
Ctrl-P:	Der Zeichensatz wird umgeschaltet. Auch diese Funktion ist mit der des Editors gekoppelt.
CR:	Das Textfenster wird verlassen und damit die Eingabe beendet. Dabei werden alle Zeichen im Fenster belassen, egal wo der Cursor sich befindet. Im Register d5.L wird der Wert $0D oder 13 abgelegt.

Einfaches Beispiel:

Es kann ein maximal 12 Zeichen langer Text 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 Reserve)
```

Bemerkung:

Wenn X=0,1 oder Y=0,1 wird ein Teil der Umrahmung abgeschnitten da der Textrahmen nicht bei der angegebenen Koordinate anfängt, sondern zwei Punkte weiter links und weiter unten. Das gleiche gilt natürlich auch, wenn die Werte für X oder Y zu groß werden.

TRAP-Nummer:	12
Befehlsname:	CI
Befehlsgruppe:	Zeicheneingabe
Kurzbeschreibung:	Ein Zeichen wird von der Tastatur eingelesen
Eingaberegister:	KEINE
Ausgaberegister:	d0.l = Zeichen im ASCII-Code Flags
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, wodurch sehr leicht Terminalbetrieb möglich ist.
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN
Siehe auch:	CSTS(13) CIINIT2(31) CI2(32)

Beschreibung:

Ein Zeichen wird von der Tastatur gelesen. Dabei wird solange gewartet, bis das Zeichen auch 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, so ist bei jeder Eingabe über diese Routine eine HARDCOPY-Funktion möglich. Dazu muss die Tastenkombination CTRL-@ gedrückt werden. Anschließend muss noch die genaue Funktion ausgewählt werden. Es gibt dabei drei verschiedene Möglichkeiten :

Taste :

"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 einen 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 kann ebenfalls sehr leicht erfolgen. Dazu muss man in die Speicherzelle $2b (IOSTATB) den Wert 6 schreiben. Dann erfolgt beim Aufruf ein Sprung auf die Adresse $18 (USERCI). Dort muss dann 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, denn dieses verwendet bei der Umschaltung die gleiche Adresse, jedoch mit dem Wert 0 in D0.W.

Wenn man die Routine wieder normalisieren will, so muss man den Wert 0 auf die Adresse $2b schreiben. Die Adressen beziehen sich auf den Anfang des Variablenbereichs, der mit GETVAR oder aus dem Register A5.L erfahren 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 einfach nur ein Zeichen von der Tastatur gelesen.

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

Hier wird solange 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, so werden 10 Schritte ausgeführt und eine kleine 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. Wer will kann das Programm zu einem richtigen kleinen CAD-System ausbauen.

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
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
Änderungen zu 6.3: NEIN

Siehe auch: CI(12) CIINIT2(31) CI2(32)

Beschreibung:

Eng verwandt mit dem CI-Befehl ist der Befehl CSTS. Damit ist es möglich abzufragen, ob eine Taste gedrückt wurde oder nicht. Im Register D0.L erscheint eine 0, wenn keine Taste gedrückt wurde, sonst der Wert $ff. Das Zeichen wird aber nicht eingelesen, sondern muss dann mit dem Befehl CI geholt werden. Auch bleibt der Status solange erhalten, bis das Zeichen mit CI geholt wurde. Die FLAGS sind auch entsprechend dem Wert gesetzt, wodurch direkt nach dem Aufruf gleich ein Sprung ausgeführt werden kann.

Wenn die Routine CI auf die serielle Karte gelenkt wird, so auch diese Routine. Dadurch kann sehr leicht ein Terminalbetrieb aufgebaut werden.

Ebenso ist ein Umlenken auf eine Benutzerroutine vorgesehen. Dieses Umlenken funktioniert gemeinsam mit der Umlenkung der CI-Routine und kann dort nachgelesen werden. Allerdings wird hier die Adresse $1E (USERCSTS) angesprungen. Auch hier wird der Wert 2 in Register D0.W übergeben. Die Adresse bezieht sich auch hier wieder auf den Anfang des Variabelnbereichs.

Auch hier muss auf die Konvention der CSTS-Routine geachtet werden.

Einfaches Beispiel:

Dieses kleine Programm prüft, ob eine Tast 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:
        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   #!CSTS,D7       * Fragen, ob Taste gedrückt
        BEQ.S   START           * Nein, dann zurück
        RTS                     * Ja, dann aufhören
```

TRAP-Nummer: 14
Befehlsname: RI
Befehlsgruppe: CAS-Baugruppe
Kurbeschreibung: Es wird ein Zeichen von der CAS-Baugruppe gelesen

Eingaberegister: KEINE
Ausgaberegister: d0.l = Einhält das Zeichen, das gelesen wurde
Flags
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 CTRL-C abgebrochen werden.
Änderungen zu 6.3: NEIN

Siehe auch: PO(15) RELAN(118) RELAUS(119)

Beschreibung:

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

Mit der Tastenkombination CTRL-C 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 CTRL-C abgebrochen wird.

```
START:
        LEA     BUFFER(PC),A0   * Zieladresse in A0
SCHLEIFE:
        MOVEQ   #!RI,D7
        TRAP    #1              * Zeichen lesen
        BCS.S   ENDE            * Abbruch mit CTRL-C
        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 immer ein unbedeutender Datenstrom den eigentlichen Daten vorweggeschickt.

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

Mit "ANSEHEN" kann man sich z.B. den Inhalt des Buffers dann anschauen.

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 CTRL-C abgebrochen werden.
Änderungen zu 6.3: NEIN

Siehe auch: RI(14) RELAN(118) RELAUS(119)

Beschreibung:

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

Auch dieser Befehl kann mit CTRL-C abgebrochen werden. Dann ist der Wert in d0 als Rückgabe undefiniert und das CARRY-Flag ist gesetzt.

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 sind zwei Programme gezeigt, 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         *
        BCS.S   SCHRENDE        * Abbruch
        TRAP    #1              * Zeichen ausgeben
        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     SCHRLP1         * bis das Zeichen "NUL" erscheint
SCHRENDE:
        RTS                     * Dann Ausgabe beendet
```

```
AUSBUF:
        DC.B    'HALLO',0       * Text
        DS      0               * Angleich auf Even-Adresse


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         * dem 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 Aufzeichnungsvorgang mit "SCHREIBE" geht bei diesem Beispiel sehr schnell, daher meldet sich das Menü auch sofort wieder.

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

In der Praxis muss man dieses einfache Programm noch um eine Prüfmöglichkeit erweitern. Denn treten beim Lesen Fehler auf, so muss eine Kontrollmöglichkeit vorhanden sein.

Meist wird dazu eine Prüfsumme verwendet. Das heißt man summierte alle Datenbytes auf, verwendet aber nur z.B. 16 Bits des Ergebnis und gibt dieses Ergebnis 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 selbst realisiert, wenn man Texte oder Daten aufzeichnet. Ferner ist es nützlich, die Anzahl der Datenbytes mit am Anfang aufzuzeichnen, um auch beliebige Daten abspeichern zu können, die auch z.B. 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
Änderungen zu 6.3:	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)		

Beschreibung:

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

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

Ab Version 6.1 wird auch der HARDSCROLL-Wert auf Null gesetzt.

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, so kann auch ein interner Befehl dafür benutzt werden (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
Änderungen zu 6.3:	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)		

Beschreibung:

Damit kann eine unsichtbare Seite gelöscht werden. Der Vorgang dauert hierbei 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
```

Komplexes Beispiel:

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

```
START:
        MOVEQ   #1,D6           * Schreibseite
        MOVEQ   #0,D7           * Leseseite
        MOVEQ   #0,D5           * Zähler für SIN, COS
SCHLEIFE:
        MOVE    D6,D0           * Schreibseite
        MOVE    D7,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
        MOVEQ   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,D7           * Lese- und Schreibseite vertauschen
        ADD     #10,D5          * 10 Grad dazu
        MOVEQ   #!CSTS
        TRAP    #1
        BEQ.S   SCHLEIFE        * Und wiederholen, bis Zeichen eingegeben wurde
        RTS
```

Wenn man die Graphik schneller haben will, so 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 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 der CLPG-Befehl Zeit beansprucht.

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
Änderungen zu 6.3:	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)		

Beschreibung:

Immer wenn man direkt Daten auf die GDP-Ports ausgeben will, muss man solange warten, bis der GDP den letzten Befehl ausgeführt hat. Dies kann man direkt mit dem WAIT-Befehl tun.

Einfaches Beispiel:

Nur warten bis GDP fertig ist.

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

Komplexes Beispiel:

Der Buchstabe "A" wird in allen möglichen 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              * Size merken
        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
SYNC:
        MOVEQ   #!SYNC,D7       * 20 ms (16,67 ms) Warten
        TRAP    #1
        BEQ.S   SYNC
        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
Änderungen zu 6.3:	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)

Beschreibung:

Damit ist es möglich um 1/16 Bildpunkt zu schreiten. Dieser Befehl wird z.B. für Kreise mit unterschiedlichen Radien benötigt.

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 gegebenem Radius gezeichnet werden:

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

U = 2 * PI * R

Wobei PI = 3.141592 ... , R = Radius und U = Umfang.

Das Kreisprogramm habe 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      * Um 1 Grad drehen
        TRAP    #1
        DBRA    D3,SCHLEIFE
        RTS
```

Es gilt nun ANZAHL zu berechnen. Der Umfang ergibt sich zu:

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 und es ergibt sich:

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, so sollte mit der SINUS- und COSINUS-Funktion gearbeitet werden.

TRAP-Nummer:	20
Befehlsname:	CLRSCREEN
Befehlsgruppe:	Zeichenausgabe
Kurzbeschreibung:	Löscht den Bildschirm und bereitet ihn zur Ausgabe 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
Änderungen zu 6.3:	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)	

Beschreibung:

Der Bildschirm wird gelöscht. Dabei werden alle vier Bildseiten gelöscht. Ferner wird die Textausgabe vorbereitet und es erscheint ein blinkendes Cursorfeld links oben auf dem Bildschirm. Der interne Bildschirmspeicher wird gelöscht. Außerdem wird der HARDSCROLL-Wert wieder auf Null gesetzt, was bedeutet, dass der Bildschirm wieder so liegt, als würde er ohne HARDSCROLL betrieben.
Immer wenn man die Ausgaberoutinen CO und CO2 verwenden will, muss man mit CLRSCREEN die Ausgabe 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
```

Komplexes Beispiel:

Der Bildschirm wird gelöscht und ein Text wird 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 CO',$d,$a
        DC.B    'und CLRSCREEN',0
```

Der Cursor blinkt danach direkt hinter der letzten Textstelle.

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 sind vorhanden. Es erfolgt Keine Zeichensatzanpassung bei der Ausgabe.
Änderungen zu 6.1:	Hardscroll ist auch mit Cursordarstellung möglich.
Änderungen zu 6.3:	NEIN

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)	

Beschreibung:

Damit wird ein Zeichen im Textmode ausgegeben. Das Zeichen steht zuvor in Register D0.B. Dabei können auch Steuerzeichen ausgegeben werden.

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

Bevor man CO verwenden kann, muss CLRSCREEN aufgerufen worden sein, sonst erscheint kein Cursor auf dem Bildschirm.

Die Zeichenbreite wird bei der Ausgabe so gewählt, wie man sie zuvor im OPTIONEN-Menue eingestellt hat (40 oder 80 Zeichen pro Zeile). Auch eine nachträgliche Einstellung mit SIZE ist möglich.

Seite der 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 ein Register auf der GDPHS-Karte verändert und die oberste Zeile wird gelöscht. Der HARDSCROLL-Modus darf ab dieser Version auch mit Cursordarstellung getrieben werden. Weiterhin sind in diesem Modus alle Befehle erlaubt, die nur in der Cursorzeile Veränderungen verursachen. 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 muss 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 reiner Ausgabe mit CO2, CHAR oder CO möglich und kann deshalb 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 besitzen sind Sonderzeichen, wie "[" 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, so erfolgt ein Sprung zur nächsten Zeile und das Zeichen erscheint links auf der nächsten Zeile. Wenn ein Zeichen auf der letzten Zeile ganz rechts ausgegeben wird, so erfolgt ein Scrollvorgang des Bildschirms. Der Bildschirm wird dabei um eine Zeile nach oben verschoben.

Einfaches Beispiel:

Es werden alle normalen 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-Menue auf 80 Zeilen pro Zeile umgeschaltet hat.

CTRL-Befehle

Hier werden Codes im Bereich 0 bis $1F als Befehl interpretiert. Dabei gilt der Code $1B als besonderer Code, da dort 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 soviel wie Rückwärtsschritt bedeutet. Befindet sich der Cursor schon auf der linken Seite des Bildschirmes, so wandert er eine Zeile nach oben und kommt ganz rechts zum Vorschein. Wenn der Cursor in der obersten Zeile ganz links war, so kann man ihn nicht mehr weiter bewegen.

Code $09: CTRL-I

Der Cursor wird ein Zeichen nach rechts verschoben. War der Cursor ganz rechts auf der Zeile, so erfolgt ein Sprung in die nächste Zeile. War der Cursor dabei auch auf der letzten Zeile, so wird der Bildschirminhalt 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) war, so wandert er nicht weiter nach unten, sondern der gesamte Bildschirminhalt wird um eine Zeile nach oben verschoben. Diese Funktion nennt man auch 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 war, so verändert er seine Position nicht weiter.

Code $0C: CTRL-L

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

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

Code $0D: CTRL-M

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

Code $16: CTRL-V

Der Cursor wird eine Zeile nach unten transportiert. Wenn er auf der untersten Zeile stand, so wird er aber nicht weiter nach unten gesetzt. Es erfolgt auch kein Scroll-Vorgang.

Code $1A: CTRL-Z

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

Code $1E: CTRL-^

Der Cursor wandert in die linke obere Ecke, der Bildschirm bleibt aber erhalten. Der Befehl wird auch HOME genannt.

ESC-Befehle

Bei den ESC-Befehlen werden mehrere Codes übertragen. Auch sie dienen der Cursor- und Bildschirmsteuerung. Zum Test wird zunächst, wie angegeben die ESC-Taste gedrückt und dann die Zeichensequenz.

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 dabei die linke obere Ecke des Bildschirms.

Code $1B $24: ESC $

Der Zeichensatz wird auf Deutsch umgestellt. Dabei wird seit Version 6.0 die Zeichensatzanpassung nicht mehr von dieser Routine durchgeführt. Dadurch ist gemischter Ausgabebetrieb mit deutschem und amerikanischem Zeichensatz möglich. Die Auswertung der eingegebenen Zeichen muss von den Anwenderprogrammen selbst durchgeführt werden.

Code $1B $25: ESC %

Der Zeichensatz wird wieder auf amerikanisch umgestellt. Dabei gelten die gleichen Änderungen zu Version 4.3 wie bei 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 wird keine weitere Anpassung vorgenommen.

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. Dies entspricht dem Befehl 'O' des DRUCKMENÜS (Siehe auch Kapitel 3.4.3.). Außerdem wird der Zeichensatz bei der Ausgabe umgeschaltet. Dies entspricht dem NDR-Zeichensatz im Druckmenü.

Code $1B $51: ESC Q

Ein Leerzeichen wird an der aktuellen Cursorstelle eingefügt. Dabei wandern alle auf der Zeile befindlichen Zeichen um eins nach rechts. Wenn ganz rechts ein Zeichen stand, so geht es verloren.

Code $1B $52: ESC R

Die Zeile, in der sich der Cursor befindet wird gelöscht, und alle Zeilen darunter werden um eine Zeile nach oben geschoben.
Dieser 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 und die restliche Zeile wandert um eine Zeichen-Stelle 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

Dieser Befehl ist seit Version 6.0 vorhanden. Damit kann der HARDSCROLL-Modus ausgewählt 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 wieder 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
Kurzbeschreibung:	Zeichen über Centronix an Drucker ausgeben
Eingaberegister:	d0.b = Zeichen, das ausgegeben werden soll
Ausgaberegister:	KEINE
Zerstörte Register:	KEINE
Ab Version:	3.1
Änderungen zu 4.3:	Eine Umlenkung auf die serielle Schnittstelle ist möglich.
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	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)	

Beschreibung:

Das Zeichen, das in Register D0.B steht, wird über die Druckerschnittstelle ausgegeben. Dabei erfolgt die Ausgabe 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 eine vollständige Zeile ausgibt. Deshalb muss normalerweise immer noch $0D und $0A hinterhergeschickt werden.

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

Komplexes Beispiel:

Der aktuelle Editortext wird auf dem Drucker ausgegeben. Dabei wird keine Zeichensatzumschaltung vorgenommen. Soll diese mit durchgeführt werden, so muss mit dem DRCKMENÜ 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 obigen Programm ist es möglich einen Text, der z.B. mit dem Editor eingegeben wurde, auf dem Drucker auszugeben. Wenn man das Programm startet, so druckt es sich zunächst selbst aus.

Will man einen anderen Text ausgeben, so muss man entweder einen neuen Text mit dem Editor eingeben, oder mit der Funktion TEXTSTART (OPTIONEN-Menue) einen neuen Textanfang auswählen.

TRAP-Nummer:	23
Befehlsname:	SIN
Befehlsgruppe:	Berechnungen
Kurzbeschreibung:	Berechnet den Sinus*256 eines Wertes
Eingaberegister:	d0.w = Wert, für den der Sinus berechnet werden soll
Ausgaberegister:	d0.w = Sinus des Wertes
Zerstörte Register:	KEINE
Ab Version:	3.1
Änderungen zu 4.3:	NEIN
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN

Siehe auch: COS(24) WERT(29) MULS32(72) DIVS32(73) ADJ360(83) RND(96)

Beschreibung:

Damit steht die SIN-Funktion zur Verfügung. Der Sinus wird dabei 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 zu sehen.

Komplexes 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, dadurch 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 Stopp 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
```

Bemerkung:

Die Fouriersynthese erfüllt wichtige Aufgaben in der Technik. Hier soll aber nicht weiter darauf eingegangen werden, sondern dies sei nur ein Beispiel für die Anwendung der schnellen SIN-Funktion.

TRAP-Nummer:	24
Befehlsname:	COS
Befehlsgruppe:	Berechnungen
Kurzbeschreibung:	Berechnet den Cosinus*256 eines Wertes
Eingaberegister:	d0.w = Wert, für den der Cosinus berechnet werden soll
Ausgaberegister:	d0.w = Cosinus des Wertes
Zerstörte Register:	KEINE
Ab Version:	3.1
Änderungen zu 4.3:	NEIN
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN

Siehe auch: SIN(23) WERT(29) MULS32(72) DIVS32(73) ADJ360(83) RND(96)

Beschreibung:

Für die COS-Routine gilt das gleiche wie schon bei SIN. Im Register D0.W wird der Wert des Parameters in Grad übergeben und das Ergebnis steht dann nach dem Aufruf wieder im Register D0.W. Der Wert ist dabei cos(x)*256 und 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 vom Mittelpunkt eines Kreises zu den Kreisrandpunkten eine Linie 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 Cosinus genauer als 1 Grad benötigt, so muss man die Zwischenpunkte aproximieren. Dies wird z.B. im Grafik-Paket bei den Kreisen und Ellipsen so gemacht.

TRAP-Nummer:	25
Befehlsname:	SIZE
Befehlsgruppe:	Zeichenausgabe
Kurzbeschreibung:	Stellt die 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 sein sollte.
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	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)	

Beschreibung:

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 genauso, 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 unsichtbar.

Diese beiden Codes werden auch im OPTIONEN-Menue eingestellt, wenn man die Zeichengröße festsetzt. Andere Größen sind prinzipiell möglich, jedoch ergeben sich dabei Bildausschnitte, ggf. auch in der Zeilenrichtung. Eigene Versuche schaffen hier aber schnell Aufschluß.

Ist der HARDSCROLL eingeschaltet, und wird dann eine Schriftgröße eingestellt, die eine Y-Vergrößerung ungleich 1 enthält, so 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 auch 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, so erscheint ein ganz großer Cursor auf dem Bildschirm und auch der Text ist gedehnt. Man sieht aber nur einen Bildausschnitt. Wenn der Cursor ganz oben steht, so wird er abgeschnitten, denn der Start liegt ein halbes Zeichen oberhalb. Auch erscheinen auf dem Bildschirm nur ca. 12 Zeilen. Die restlichen Zeilen sind weiter unten unsichtbar. Will man sie sehen, so kann man mit CTRL-Z den Bildschirm nach oben schieben. Es ist jetzt möglich den Cursor in ein unsichtbares Gebiet zu schieben. Will man ihn wieder sehen, so gibt man z.B. CTRL-Q, SHIFT-R ein und der Cursor ist links oben.

Diese Darstellungsart ist z.B. gut, wenn man eine Gruppe von Personen Einzelheiten bei der Eingabe zeigen will.

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

TRAP-Nummer:	26
Befehlsname:	CMD
Befehlsgruppe:	Grafik
Kurzbeschreibung:	Befehl an GDP direkt
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
Änderungen zu 6.3:	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)		

Beschreibung:

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

Eine genaue Liste alle Befehle findet man bei der GDP-Beschreibung des EF9366. Hier aber 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 setzen (hell oder dunkel).
$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, und die Codes $80 bis $FF für Kurzvektoren. Die Vektoren sind aber über DRAWTO oder FIGUR-Befehl 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 auf. Damit kann man inverse Darstellung vorbereiten, wenn man danach Zeichen oder Graphik im Löschmode (CMD mit D0=1 oder ERAPEN) ausgibt.

Komplexes 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 ms (16,67 ms) 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 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
```

Durch den Befehl SYNC wird erreicht, dass der Text ohne Flimmern über den Bildschirm wandert. Zur Probe kann man die drei Zeilen weglassen.

Will man die Schrift schnell bewegen, so addiert man einfach einen größeren Wert auf D1 und D2. Man kann den Text auch auf zwei Bildschirmseiten darstellen und dann ohne SYNC arbeiten. Dann kann man die Schrift sehr schnell bewegen, 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
Änderungen zu 6.3:	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)		

Beschreibung:

Die GDP-Baugruppe besitzt vier Bildseiten mit 512 x 256 Punkten. Wenn man die Bildseite auswählen will, so kann man den Befehl NEWPAGE verwenden. 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 und das Bild auf dem Bildschirm stammt von der Leseseite. Damit ist es also möglich, auf einer unsichtbaren Seite Bilder zu erstellen und dabei eine andere Seite zu betrachten. Dies wird benötigt, um 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, so 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
```

Es wird ein Kreis dargestellt, der immer größer wird.

```
START:
        MOVEQ   #!FIRSTTIME,D7  * Turtle 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 zuerst initialisiert wird, und dann wird sie mit "HIDE" abgestellt. Dann erst kann man die zweite Bildseite auch mitverwenden.

Eine andere Möglichkeit ist es auch, das alte Bild zu löschen, indem die Graphik mit ERAPEN neu gezeichnet wird.

Komplexes Beispiel:

Es wird wieder ein Kreis vergrößert, allerdings wird er auch noch im Kreis auf dem Bildschirm bewegt.

```
START:
        MOVEQ   #!FIRSTTIME,D7  * Turtle 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
        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 dann dazu verwendet werden, den Bildwechsel mit der Graphikerstellung zu synchronisieren, um damit keine störenden Bildschnitte mehr zu erhalten.

TRAP-Nummer:	28
Befehlsname:	SYNC
Befehlsgruppe:	Synchronisation
Kurzbeschreibung:	Fragt den 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
Änderungen zu 6.3:	NEIN
Siehe auch:	DELAY(35)

Beschreibung:

Damit ist es möglich, den Vertikal-Synchronisationsimpuls des GDP abzufragen. Immer wenn ein Durchgang erfolgte, 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 sehr schnell hintereinander abfragt und sich immer noch in der vertikalen Austastlücke befindet. Dies ist nötig um auch Zeitsynchronisationen zu ermöglichen. Wenn man SYNC dauernd abfragt, so liefert es alle 20ms (16,67 ms bei der GDP-FPGA) 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       * V-Sync abfragen
        TRAP    #1
        BEQ.S   START           * Wiederholen, bis SYNC da ist
        RTS
```

Komplexe Beispiele:

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*20ms (60*16,67ms bei GDP-FPGA)
WARTE:
        MOVEQ   #!SYNC,D7       * Messen
        TRAP    #1
        BEQ.S   WARTE           * Bis 20 ms (16,67 ms)
        DBRA    D3,WARTE        * Bis 1 sec
        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              * Modul-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
```

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 Line
        TRAP    #1
        MOVE    #255,D2         * Y = 255
        MOVEQ   #!DRAWTO,D7     * Ende der Line
        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-Positition
        CMP     #512,D1
        BNE.S   SCHLEIFE        * Wenn rechter Rand erreicht, dann Ende
        RTS
```

Wenn man die SYNC-Befehl einmal weglässt, so sieht man den gleichen Vorgang, jedoch erscheint die Linie unterbrochen und sie wandert auch viel schneller über den Bildschirm. Die Unterbrechungen kommen daher, weil man nun Lösch- und Schreibvorgang überlagert sieht.

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
Änderungen zu 6.3:	NEIN

Siehe auch:

Beschreibung:

Jetzt kommt ein sehr komplexer Befehl. Damit ist es möglich Ausdrücke zu berechnen. Dabei können Symbole, wie auch mathematische Zeichen verwendet werden. Dieser Befehl wird im Grundprogramm dazu verwendet, um z.B. im Assembler arithmetische Ausdrücke zu berechnen. Ebenfalls dient der Befehl der Eingabe von Startadressen etc., wo immer es gilt eine Zahl aus einer Formel oder einer einfachen Eingabe zu ermitteln. Dabei können auch Symbole verwendet werden, die z.B. im Assembler definiert wurden.

Für die Parameter gilt folgendes:

A0.L muss 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 auftrat.

Für den Fehlercode in D0.L gilt:

		Für das Carry-Flag gilt:	
0	Syntax-Fehler aufgetreten	1	gesetzt
1	Bytewert geladen	0	rückgesetzt
2	Wortwert geladen	0	rückgesetzt
3	Langwortwert geladen	0	rückgesetzt
4	-reserviert-		
5	Symbolaufruf ohne Definition des Symbols	0	rückgesetzt

Als Operationen sind zulässig:

+	Addition 32 Bit
-	Subtraktion 32 Bit
*	Multiplikation
/	Division
\	Modulo-Division Das Vorzeichen des Ergebnis entspricht dem Vorzeichen des Divisors.
( )	Klammerrechnung
!	Oder-Verknüpfung 32 Bit
&	Und-Verknüpfung 32 Bit

*	Steht für den Programmstand beim Assembler
?	Ist die aktuelle Textanfangsadresse
'...'	Text, wird als ASCII-Wert(e) geladen
~	Nicht-Operation
-	Vorzeichen einer Zahl
$nnnnnnnn	Sedezimale Zahl
ddddddddd	Dezimale Zahl
%bbbbbbb	Binäre Zahl
@symbol	Adresse eines festen Symbols
!symbol	Index eines festen Symbols, für TRAP-Befehl

Alle Operationen werden mit einer Genauigkeit von 32 Bit durchgeführt.

Die nachfolgenden Zusätze stehen hinter einem Ausdruck (ganz hinten):

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

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 hexadezimal $11 oder dezimal 17. Man kann das überprüfen, wenn man das Programm im Einzelschritt durchläuft.

Komplexes Beispiel:

Es kann ein Ausdruck eingegeben werden, der dann berechnet und ausgegeben wird. Wenn man nur <RETURN> drückt endet das Programm.

```
START:
        LEA     BUFFER(PC),A0   * Text eingeben
        MOVEQ   #$22,D0         * Größe
        MOVEQ   #2,D1           * X=2
        MOVE    #128,D2         * Y=128
        MOVEQ   #30,D3          * Anzahl der Zeichen
        MOVEQ   #!READ,D7
        TRAP    #1              * Zeichen lesen
        TST.B   D4              * Ende, wenn nur <RETURN>
        BEQ.S   ENDE
        LEA     BUFFER,A0       * Dann auswerten
        MOVEQ   #!WERT,D7
        TRAP    #1              * Wert jetzt in D0.L
        BCS.S   START           * Bei Fehler
        CMP     #5,D1
        BEQ.S   START           * Wenn undefiniert
        LEA     BUFFER(PC),A0   * Wieder Adresse laden
        MOVE.B  #'$',(A0)+      * Kennung, dass hexadezimal
        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\d0923
!SCHREITE
@SCHREITE
?+1
3*(4+5)
5!12
'A'
'AB'
%1010
```

Man beachte, dass die Ausgabe hier hexadezimal ist. Wenn man eine dezimale Ausgabe will, so kann man PRINT4D oder PRINTV8D verwenden. Dabei muss man aber den Buffer mit Leerzeichen auffüllen, sonst bleibt eventuell der Rest der vorherigen Zeile stehen, falls die neue Ausgabe weniger Stellen besitzt als die alte Ausgabe.

Achtung! Zwischen den Operationen dürfen keine Leerzeichen stehen. Dies war nötig, da man z.B. im Assembler sonst nicht unterscheiden könnte, ob das Zeichen "*" als Kommentar oder als Operation gemeint ist.

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
Eingaberegister:	a0.l = Zeigt auf die Symboldefinition
Ausgaberegister:	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
Zerstörte Register:	d2/d3/a1/a2 bei Fehler auch d0 und d1
Ab Version:	3.1
Änderungen zu 4.3:	NEIN
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN
Siehe auch:	PRTSYM(84) SYMCLR(85) GETSYM(86) GETNEXT(87) PUTNEXT(88)

Beschreibung:

Damit ist es möglich, einem Symbol einen Wert zuzuordnen. Dies geschieht in Form einer Zuweisung:

SYMBOL := wert

Auf der rechten Seite steht ein beliebiger Ausdruck, wie er bei dem Befehl WERT möglich war. Auf der linken Seite ein Name, den man selbst gewählt hat. Diesem Namen wird dann der Wert zugeordnet. Man muss dabei immer eine Zuweisung ausführen. Wenn keine Zuweisung im Text steht, so wird ein CARRY=1 zurückgemeldet.

Bei einer Fehleingabe wird die Zuweisung damit nicht ausgeführt.

A0.L muss auf diese Definition zeigen. Wird die Zuweisung korrekt ausgeführt, so steht der Wert des Symbols in D0.L und das Attribut (wie D1.L bei WERT) steht in D1.L. Wird das Symbol nicht korrekt in die Symboltabelle eingetragen, so werden eventuell d0 und d1 zerstört.

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 hexadezimal $4B.

Komplexes Beispiel:

Hier kann man Werte berechnen lasse und Symbole in die Symboltabelle eintragen. Erfolgt keine Wertzuweisung, so 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 mal 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              * Hexadezimale 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 einmal 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
Änderungen zu 6.3: NEIN

Siehe auch: CI(12) CSTS(13) CI2(32)

Beschreibung:

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

STXTXT * Ist die Adresse des Textanfangs
AKTTXT * Ist die Adresse der aktuellen Textstelle
ETXTXT * Ist die Adresse auf das Textende (Zeichen 0)

Die Speicheraufteilung eines einfachen Textes ist dabei so:

```
            --------------------------
STXTXT---->|HALLO!!! DIES IST EIN   |
            |TEXT IM SPEICHER. ER    |
AKTTXT---->|WURDE MIT DEM EDITOR    |
            |ERSTELLT UND GESPEICHRT.|
ETXTXT---->|0                       |
            --------------------------
```

Die 0 am Textende besitzt dabei 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. Beide zeigen dann auf den Textanfang.

Der Textanfang ist durch das OPTIONEN-Menue oder im Editor umdefinierbar

Einfaches Beispiel:

CIINIT2 aufrufen, wodurch CI2 initialisiert wird.

```
START:
        MOVEQ   #!CIINIT2,D7    * Textstart setzen
        TRAP    #1
        RTS
```

Komplexes Beispiel:

Der momentane 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 die Endekennung
        MOVEQ   #!CO2,D7        * Zeichen ausgeben
        TRAP    #1
        BRA.S   SCHLEIFE        * Wiederholen
ENDE:
        RTS
```

Mit diesem Beispiel wird der Textinhalt auf dem Bildschirm ausgegeben. Wenn man dabei nur das Programm eingegeben hat, und den TEXTSTART nicht verändert hat, so gibt es sich selbst auf dem Bildschirm 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
Änderungen zu 6.3:	NEIN

Siehe auch: CI(12) CSTS(13) CIINIT2(31)

Beschreibung:

Damit 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, dem Textstart zurückgesetzt werden. Das Ergebnis des Aufrufs ist im Register D0.L (.B) abgelegt. Wenn das Textende erreicht ist, also das Zeichen 0 (Codierung 0) gefunden wurde, so wird zusätzlich das CARRY-Flag gesetzt. Es genügt aber auch einfach auf 0 abzufragen. Ferner wird beim Textende auch eine Speicherzelle mit dem Namen ERRFLAG auf den Wert 2 gesetzt. Diese Speicherzelle wird dazu verwendet, eine evtl. unterdrückte Ausgabe mit CO2 wieder einzuschalten. Im Assembler wird diese Funktion dazu verwendet ein ausgeschaltetes Listing (1 = Nur Fehlerausgabe) im Fehlerfall einzuschalten, um den Fehler auf den Bildschirm zu bringen. Jetzt gibt es noch eine zweite Möglichkeit Texte einzulesen. Wenn man auf die Speicherzelle $2B relativ zum Variablenanfang den Wert 5 schreibt, so werden die Eingaben über einen Sprungvektor gelesen. Wenn man den Sprungvektor der auf Adresse $18 steht, durch einen Sprung auf eine eigene Routine umsteuert, so kann man Eingaben auch durch eine eigene Routine erzeugen. Wenn CIINIT2 aufgerufen wurde, so wird beim nächsten CI2-Aufruf im Register D0.L der Wert 1 an die USERCI-Routine übergeben, sonst der Wert 0. Dies ist nötig, um Mehrpassvorgänge auch mit USERCI zu steuern, z.B. bei CP/M68K, um die Quelle von der Diskette mehrfach zu lesen. Der Sprungvektor USERCI ($2B) ist nach dem RESET auf die Routine CI geschaltet.

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 eine feste Funktion.
Änderungen zu 6.3:	NEIN

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)	

Beschreibung:

Zunächst besitzt CO2 die gleiche Bedeutung, wie der CO-Befehl, nämlich die Ausgabe eines Zeichens, das im Register D0.B steht auf die Console. Aber es gibt noch mehr Möglichkeiten. Zunächst kann die Ausgabe auf einen Drucker (LST), auf eine eigene Routine (USR), ins Leer (NIL) oder auf die serielle Karte (CO2SER) gelenkt werden.

Bei der CO-Routine ist die Ausgabe auf dem Bildschirm genau beschrieben.

Ab Version 6.1 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 wird. Ist CO2 auf USERCO geschaltet, so muss die Routine dafür sorgen, dass Carry den richtigen Wert enthält.

Ferner erfolgt die Ausgabe nur, wenn die Speicherzelle PASSFLAG den Wert 2 enthält. Dieser Wert ist normalerweise aber voreingestellt. Die Abfrage 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 Unterdrückte Ausgabe, dabei wird ERRFLAG geprüft und nur wenn es 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 $A) wird unterdrückt. Bei manchen Druckern ist dies nötig.
5 Ausgabe auf USERCO. Dabei steht ein Sprungvektor auf der Adresse $24 relativ zum Variablenanfang. Der Sprung wird nach dem RESET auf die Routine CO geschaltet.
6 Die Ausgabe wird auf die serielle Schnittstelle umgelenkt. Diese muss aber zuvor mit SIINIT auf die gewünschten Werte eingestellt werden.

Es gibt aber noch eine bedeutende Funktion. Wenn man den Code $01 (CTRL-A) im Register D0.B beim Aufruf der CO2-Routine angibt, so werden alle nachfolgenden Zeichen bis zum Line-Feed (Code $0A) in einem Buffer (AUSBUF) zwischengespeichert. Danach erfolgt die Ausführung.

Dazu gibt es folgende Befehle (adr und wert können beliebige Ausdrücke sein, siehe WERT, so z.B. auch 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 dies ursprünglich im Sprachkonzept nicht enthalten 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 und 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, von beliebigen Programmiersprachen aus die Werte 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-Bespiel:

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 auf Wert 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 wird 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 nur ein 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-Menue die Ausgabe auf den Drucker umstellt.

Komplexes 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
Änderungen zu 6.3:	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)		

Beschreibung:

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 wenn man Befehle der Schildkrötensprache ausführt.

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

Dazu übergibt man beim Aufruf zwei Parameter. Im Register D0.B steht die Wechselrate für die Zweiseitenumschaltung und im Register D1.B die Wechselrate für die Vierseitenumschaltung. Der Wert 0 bedeutet, keine Umschaltung und jeder Wert <> 0 ist die Umschaltrate in 20 Millisekunden-Schritten.

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 sec einstellen
        MOVEQ   #!SETFLIP,D7    * Umschaltrate setzen
        TRAP    #1
        RTS
```

Komplexes Beispiel:

Eine Buchstabenreihe wandert scheinbar über den Bildschirm. Diese Täuschung kommt dadurch, dass auf alle Bildseiten geschrieben wurde. Dabei sind die Buchstaben aber jeweils versetzt. Durch das Umschalten der Seiten kommt ein Bewegungseffekt zustande.

```
START:
        MOVE    #!SYSTEM,D7     * System-Informationen holen
        TRAP    #1
        AND     #7,D0           * Nur CPU 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 Null
        MOVEQ   #!NEWPAGE,D7    * Seite einstellen
        TRAP    #1
        MOVEQ   #'A',D0         * Zeichen 'A'
        MOVEQ   #!CMD,D7        * Ausgeben
        TRAP    #1
        ADDQ    #1,D3           * Schreibseite um eins weiter
        DBRA    D4,SCHLEIFE
        CLR     D0              * Keine Zwei-Seiten-Umschaltung
        MOVEQ   #5,D1           * Vier-Seiten-Umschaltung alle 1/10 Sekunde
        MOVEQ   #!SETFLIP,D7    * Seitenumschaltung einschalten
        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 bewegenden Pfeilen etc. Im allgemeinen benötigt man aber nur zwei Seiten.

TRAP-Nummer:	35
Befehlsname:	DELAY
Befehlsgruppe:	Synchronisation
Kurzbeschreibung:	Warteschleife in 1/10-tel 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
Änderungen zu 6.3:	Anpassung für GDP-FPGA aufgenommen
Siehe auch:	SYNC(28)

Beschreibung:

Die Routine DELAY gestattet quarzgenaue Verzögerungen im 100 Millisekundenraster. Dabei wird während der Verzögerungszeit das automatische Seitenumschalten aufrechterhalten, so dass sich beim Ablauf keine Störungen bei der Bildschirmumschaltung durch SETFLIP ergeben. Dies ist insbesondere bei Bildern mit der Schildkrötengraphik wichtig. Im Register D0.L wird die Anzahl der 1/10 sec angegeben. Wenn die Zahl gleich Null ist, so erfolgt keine Ausführung des Befehls.

Beispiel:

Das Programm wartet eine Sekunde.

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

Komplexes Beispiel:

Es wird ein Quadrat und ein Kreis gezeichnet. Dann wird alle zwei Sekunden die Darstellung 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   #20,D0
        MOVEQ   #!DELAY,D7      * 2 Sekunden 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   #20,D0
        MOVEQ   #!DELAY,D7      * 2 Sekunden warten
        TRAP    #1
        MOVEQ   #!CLR,D7        * Bildschirm löschen
        TRAP    #1
        MOVEQ   #!CSTS,D7       * Wiederhole, 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
Änderungen zu 6.3:	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)

Beschreibung:

Dadurch wird die Schildkrötengraphik initialisiert. Die Schildkröte wird in die Bildmitte gesetzt, und die Bildseitenautomatik für die Umschaltung wird aktiviert. Ferner wird die Schreib- und Leseseite auf 0 gestellt. FIRSTTIME wird auch automatisch aufgerufen, wenn man einen der Schildkrötenbefehle verwendet. Es ist also nur in Sonderfällen nötig, diesen Befehl zu verwenden. Ein Fall ist z.B., wenn man mit mehreren Ebenen arbeitet und die Schildkröte auf dem Bildschirm nicht haben möchte, dann ist folgende Sequenz interessant:

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 dann auch nicht am Anfang gesetzt, da ein Setzen nur erfolgt wenn AUTOFLIP aufgerufen wird und das geschieht nur bei den anderen Schildkrötenbefehlen. HIDE jedoch setzt den Wert von FLIP auf 0, somit findet keine automatische Bildseitenumschaltung statt, und dann wird auch bei späterem Aufruf von AUTOFLIP die Schildkröte nicht eingeblendet.

Wenn man AUTOFLIP später verwenden will, um alle Bildseiten nacheinander darzustellen, so muss man mit GRAPOFF, die Schildkrötengraphik ausschalten.

Komplexes Beispiel:

Es wird auf allen vier Seiten ein Dreieck ausgegeben, das aber jeweils gedreht ist. Später werden die Seiten dann 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 Null
        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    * Schildkröte ausschalten
        TRAP    #1
        CLR     D0
        MOVEQ   #3,D1           * Vier-Seiten-Umschaltung
        MOVEQ   #!SETFLIP,D7    * Alle 60 ms umschalten
        TRAP    #1
        MOVEQ   #!CI,D7         * Auf Taste warten
        TRAP    #1
        RTS
```

TRAP-Nummer:	37
Befehlsname:	SETPEN
Befehlsgruppe:	Grafik
Kurzbeschreibung:	Der 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
Änderungen zu 6.3:	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)		

Beschreibung:

Damit wird der Graphikprozessor in den Schreibmode mit gesenktem Schreibstift versetzt. All 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 also vorhergehende Befehle rücksetzen, die einen Löschvorgang bewirken.
Ein komplexes Beispiel folgt bei dem nächsten Befehl.

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
Änderungen zu 6.3:	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)		

Beschreibung:

Schreibstift auf Löschen umschalten. Dabei wird der Stift auch gesenkt. Nun kann man dunkle Linie auf hellem Grund schreiben oder bestehende Grafikteile löschen.
Das Gegenstück zu diesem Befehl ist SETPEN.

Einfaches Beispiel:

Schreibstift auf LÖSCHEN schalten.

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

Komplexes Beispiel:

Der Buchstabe "A" wandert über den Bildschirm.

```
START:
        MOVE    #!SYSTEM,D7     * System-Informationen holen
        TRAP    #1
        AND     #7,D0           * Nur CPU 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 (16,67 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:	Die Schildkrötengrafik wird ausgeschaltet

Eingaberegister:	KEINE
Ausgaberegister:	KEINE
Zerstörte Register:	KEINE

Ab Version:	3.1
Änderungen zu 4.3:	NEIN
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	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)		

Beschreibung:

Mit dem Befehl wird die Schildkrötengrafik ausgeschaltet, das heißt die Schildkröte wird von da an nicht mehr auf die Bildebene Eins gezeichnet. Sonst ändern sich keine Einstellungen. Wenn man danach einen Schildkrötenbefehl, wie SCHREITE gibt, so beginnt die Schildkröte wieder in der Mitte des Bildschirms zu zeichnen.

Intern wird die Variable FIRST mit dem Wert 1 belegt.

Einfaches Beispiel:

Die Schildkrötengrafik wird ausgeschaltet.

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

Komplexes 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   #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 an 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
Änderungen zu 6.3:	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)

Beschreibung:

Damit kann man eine Befehlssequenz direkt an den GDP ausgeben. Ferner stellt man zuvor mit Register D0.B die Schriftgröße ein (wie Register 3 des EF9366), in D1.W steht die X-Koordinate und in D2.W die Y-Koordinate. Das Register A0.L enthält die Adresse, bei der sich die Daten befinden. Eine binäre 0 (Code 00) beendet den Text. Man kann also damit nicht den Befehl "Auf Schreiben schalten" mit dem Code 00 an den GDP ausgeben, sondern muss dies ggf. mit SETPEN oder mit CMD direkt tun. Der Befehl 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
        MOVEQ   #50,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 den prinzipielle Aufbau eines Kurzvektors:

```
B7   B6   B5   B4   B3   B2   B1   B0
1    --Dx---   --Dy---   --Richtung--
```

Am ersten Bit 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. Da der GDP hierbei nur in 45 Grad-Schritten arbeitet, kann man Dx und Dy immer mit dem gleichen Wert belegen. 00 bedeutet dabei kein 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
```

Einfaches Beispiel:

Es wird ein kleiner Achteck ausgegeben.

```
START:
        LEA     DATEN(PC),A0    * Adresse der Daten
        MOVEQ   #$22,D0         * Textgröße
        MOVEQ   #50,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:	Hexadezimale 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
Änderungen zu 6.3:	NEIN
Siehe auch:	PRINT4X(42) PRINT6X(43) PRINT8X(44) PRINT8B(45) PRINT4D(46) PRINT8D(70) PRINTV8D(71)

Beschreibung:

Hexadezimale Ausgabe eines Wertes. Dabei werden zwei hexadezimale Ziffern ausgegeben. Der Wert steht zuvor im Register D0.B. Im Register A0.L ist die Zieladresse für die hexadezimalen Ziffern angegeben. Dabei wird das Ergebnis im ASCII-Code abgelegt. A0.L zeigt anschließend auf den nächsten freien Platz und dort ist eine 0 (Code 00) abgelegt, so dass man den Text mit den Befehlen wie WRITE direkt ausgeben kann.

Einfaches Beispiel:

Es wird der Text '1A',0 im Zielbuffer abgelegt.

```
START:
        LEA     BUFFER(PC),A0   * Zielbufferadresse
        MOVEQ   #$1A,D0         * Wert
        MOVEQ   #!PRINT2X,D7    * Ausgabe
        TRAP    #1
        RTS

BUFFER:
        DS.B    4               * Zielbuffer
```

Komplexes Beispiel:

Es wird von $00 bis $FF gezählt, und die Zahl wird ausgegeben.

```
START:
        CLR     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
        MOVEQ   #2,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    4               * eigentlich nur 3 Bytes nötig.
```

Das Programm zählt hexadezimal von 0 bis $FF und gibt das Ergebnis alle 1/5 Sekunde auf dem Bildschirm aus. 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:	Hexadezimale 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
Änderungen zu 6.3:	NEIN
Siehe auch:	PRINT2X(41) PRINT6X(43) PRINT8X(44) PRINT8B(45) PRINT4D(46) PRINT8D(70) PRINTV8D(71)

Beschreibung:

Hexadezimale Ausgabe eines Wertes. Dabei werden vier hexadezimale Ziffern ausgegeben. Der Wert steht zuvor im Register D0.W. Im Register A0.L ist die Zieladresse für die hexadezimalen Ziffern angegeben. Dabei wird das Ergebnis im ASCII-Code abgelegt. A0.L zeigt anschließend auf den nächsten freien Platz und dort ist eine 0 (Code 00) abgelegt, so dass man den Text mit den Befehlen wie WRITE direkt ausgeben kann.

Einfaches Beispiel:

Im Zielbuffer wird der Text '12AB',0 abgelegt.

```
START:
        LEA     BUFFER(PC),A0   * Zielbufferadresse
        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:	Hexadezimale Ausgabe 6 Stellen
Eingaberegister:	d0.l = Auszugebender Wert (aber nur 3 Byte) 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
Änderungen zu 6.3:	NEIN
Siehe auch:	PRINT2X(41) PRINT4X(42) PRINT8X(44) PRINT8B(45) PRINT4D(46) PRINT8D(70) PRINTV8D(71)

Beschreibung:

Hexadezimale Ausgabe eines Wertes. Dabei werden sechs hexadezimale Ziffern ausgegeben. Der Wert steht zuvor im Register D0.L. Im Register A0.L ist die Zieladresse für die hexadezimalen Ziffern angegeben. Dabei wird das Ergebnis im ASCII-Code abgelegt. A0.L zeigt anschließend auf den nächsten freien Platz und dort ist eine 0 (Code 00) abgelegt, so dass man den Text mit den Befehlen wie WRITE direkt ausgeben kann.

Einfaches Beispiel:

Im Zielbuffer wird der Text '12ABCD',0 abgelegt.

```
START:
        LEA     BUFFER(PC),A0   * Zielbufferadresse
        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:	Hexadezimale 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
Änderungen zu 6.3:	NEIN
Siehe auch:	PRINT2X(41) PRINT4X(42) PRINT6X(43) PRINT8B(45) PRINT4D(46) PRINT8D(70) PRINTV8D(71)

Beschreibung:

Hexadezimale Ausgabe eines Wertes. Dabei werden acht hexadezimale Ziffern ausgegeben. Der Wert steht zuvor im Register D0.L. Im Register A0.L ist die Zieladresse für die hexadezimalen Ziffern angegeben. Dabei wird das Ergebnis im ASCII-Code abgelegt. A0.L zeigt anschließend auf den nächsten freien Platz und dort ist eine 0 (Code 00) abgelegt, so dass man den Text mit den Befehlen wie WRITE direkt ausgeben kann.

Einfaches Beispiel:

Im Zielbuffer wird der Text '1234ABCD',0 abgelegt.

```
START:
        LEA     BUFFER(PC),A0   * Zielbufferadresse
        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
Änderungen zu 6.3:	NEIN
Siehe auch:	PRINT2X(41) PRINT4X(42) PRINT6X(43) PRINT8X(44) PRINT4D(46) PRINT8D(70) PRINTV8D(71)

Beschreibung:

Binäre Ausgabe eines Wertes. Dabei werden acht Dualstellen ausgegeben. Der Wert steht zuvor im Register D0.B. Im Register A0.L ist die Zieladresse für die dualen Ziffern angegeben. Dabei wird das Ergebnis im ASCII-Code abgelegt. A0.L zeigt anschließend auf den nächsten freien Platz und dort ist eine 0 (Code 00) abgelegt, so dass man den Text mit den Befehlen wie WRITE direkt ausgeben kann.

Einfaches Beispiel:

Im Zielbuffer wird der Text '00011010',0 abgelegt.

```
START:
        LEA     BUFFER(PC),A0   * Zielbufferadresse
        MOVEQ   #$1A,D0         * Wert
        MOVEQ   #!PRINT8B,D7    * Ausgabe
        TRAP    #1
        RTS

BUFFER:
        DS.B    10              * Zielbuffer
```

Komplexes Beispiel:

Es werden die Zahlen von 0 bis 255 dual ausgegeben.

```
START:
        CLR     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
        MOVEQ   #2,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 1 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
Änderungen zu 6.3:	NEIN
Siehe auch:	PRINT2X(41) PRINT4X(42) PRINT6X(43) PRINT8X(44) PRINT8B(45) PRINT8D(70) PRINTV8D(71)

Beschreibung:

Dezimale Ausgabe eines Wertes. Der Wert steht zuvor im Register D0.W. Im Register A0.L ist die Zieladresse für die dezimalen Ziffern angegeben. Dabei wird das Ergebnis im ASCII-Code abgelegt. A0.L zeigt anschließend auf den nächsten freien Platz und dort ist eine 0 (Code 00) abgelegt, so dass man den Text mit den Befehlen wie WRITE direkt ausgeben kann. Achtung die Ausgabe erfolgt ohne Vorzeichen. Wenn man Zweierkomplement-Zahlen verwendet, so muss man bei einer negativen Zahl das Vorzeichen selbst mit 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 übereinander schreibt den Zielbuffer mit Leerzeichen auffüllen und die Endekennung eventuell neu setzen.

Einfaches Beispiel:

Im Zielbuffer wird der Text '4666',0 abgelegt.

```
START:
        LEA     BUFFER(PC),A0   * Zielbufferadresse
        MOVE.W  #$123A,D0       * Wert
        MOVEQ   #!PRINT4d,D7    * Ausgabe
        TRAP    #1
        RTS

BUFFER:
        DS.B    6               * Zielbuffer
```

Komplexes Beispiel:

Es wird von 500 bis Null gezählt, und die Zahlen werden ausgegeben.

```
START:
        MOVE    #500,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
        MOVEQ   #2,D1           * X-Position
        MOVE    #128,D2         * Y-Position
        MOVEQ   #!WRITE,D7      * Auf den Bildschirm
        TRAP    #1
        MOVE    #2,D0          * 1/5 Sek warten
        MOVEQ   #!DELAY,D7
        TRAP    #1
        DBRA    D4,SCHLEIFE    * Wiederholen
        RTS

BUFFER:
        DS.B    6               * Ablageadresse
```

Das Programm zählt von 500 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
Änderungen zu 6.3:	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)

Beschreibung:

Wenn man den Befehl eingibt, so wird die automatische Seitenumschaltung abgestellt, und die Schildkröte wird 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 die Lese- und Schreibseite sowie den XOR-Mode.

Einfaches Beispiel:

Es wird eine kurze Linie gezeichnet und dann die Schildkrötengrafik abgestellt.

```
START:
        MOVEQ   #10,D0
        MOVEQ   #!SCHREITE,D7   * 10 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, so empfiehlt sich folgende Methode:

```
START:
        MOVEQ   #!FIRSTTIME,D7  * Schildkröte an
        TRAP    #1
        MOVEQ   #!HIDE,D7       * Nicht zeigen
        TRAP    #1
        MOVEQ   #10,D0
        MOVEQ   #!SCHREITE,D7   * 10 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 nun die Taste "F=Flip" betätigt, so werden beide Bildseiten umgeschaltet, aber keine 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
Änderungen zu 6.3:	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)		

Beschreibung:

Eine durch HIDE ausgeschaltete Bildschirmseitenumschaltung wird wieder rückgängig gemacht. Die Schildkröte wird dann wieder eingeblendet.

Als Ausgabe erhält man in D0.B den Wert, der vom Programm HIDE auf Port $60 der GDP geschrieben wird. Dieser Wert enthält die Lese- und Schreibseite sowie den XOR-Mode.

Einfaches Beispiel:

Nachdem eine Linie gezogen ist, passiert nichts mehr, da HIDE und SHOW sich aufheben.

```
START:
        MOVEQ   #10,D0
        MOVEQ   #!SCHREITE,D7   * 10 Schritte gehen
        TRAP    #1
        MOVEQ   #!HIDE,D7       * Abschalten
        TRAP    #1
        MOVEQ   #!SHOW,D7       * Wiederanschalten
        TRAP    #1
        RTS
```

Komplexes Beispiel:

Alle Sekunde wechselt die Darstellung zwischen "Schildkröte" sichtbar und "Schildkröte" unsichtbar.

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
Änderungen zu 6.3:	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)	

Beschreibung:

Die CO2-Ausgabe ist auf verschiedene Ausgabegeräte umstellbar. Mit dem Aufruf von CRT wird die Ausgabe wieder auf den Bildschirm gestellt. Dieser Befehl ist insbesondere für höhere Programmiersprachen wichtig, um die Ausgaben auf unterschiedliche Geräte zu steuern. Mit CRT wird aber meist die Ausgabe zurückgesetzt.

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
Änderungen zu 6.3:	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)	

Beschreibung:

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   #!LST,D7
        TRAP    #1              * umschalten auf den Drucker
        RTS
```

Die Funktion ist genauso wie im Optionenmenü, 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
Kurzschreibung:	Schaltet CO2 auf den Benutzervektor um
Eingaberegister:	KEINE
Ausgaberegister:	KEINE
Zerstörte Register:	KEINE
Ab Version:	3.1
Änderungen zu 4.3:	NEIN
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	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)	

Beschreibung:

Die Ausgabe des CO2-Befehls wird auf die Benutzerschnittstelle umgelenkt.

Es wird dann 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).

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
Änderungen zu 6.3:	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)	

Beschreibung:

Alle Ausgaben mit dem Befehl CO2 werden unterdrückt, es sein denn, die Variable ERRFLAG wurde umbesetzt (siehe CO2-Beschreibung). Diese 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 Optionenmenue "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
Änderungen zu 6.3: NEIN

Siehe auch: GETERR(54) ASSEMLE(65) GETORG(68)
SETORG(69) ASSERR(120)

Beschreibung:

Damit wird die Variable ERRFLAG gesetzt. Im Register D0.W steht der Wert. Wenn ERRFLAG einen Wert ungleich 0 annimmt, so wird eine evtl. mit NIL unterdrückte CO2-Ausgabe wieder freigegeben. Diese Funktion ist speziell 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   #1,D0
        MOVEQ   #!SETERR,D7     * Fehler setzen, dadurch Ausgabe wieder an
        TRAP    #1
        MOVEQ   #'A',D0
        MOVEQ   #!CO2,D7        * 'A' ausgeben
        TRAP    #1
        RTS
```

Der Buchstabe "A" erscheint auf dem Bildschirm, obwohl durch den Aufruf von NIL die Ausgabe abgeschaltet wurde. Wenn man den Aufruf von SETERR weglässt, so erscheint keine Meldung.

Wenn man nach einem Aufruf von SETERR mit D0.W <>0, anschließend SETERR mit einem Wert von 0 in D0.W aufruft, so 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 zu 4.3:	NEIN
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN

Siehe auch: SETERR(53) ASSEMLE(65) GETORG(68) SETORG(69) ASSERR(120)

Beschreibung:

Der Inhalt der Variablen ERRFLAG wird ins 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. Dann wird auch ERRFLAG auf 2 gesetzt. Wichtig ist dabei, dass eine dann nachfolgende Ausgabe mit CO2 wieder erfolgt, falls sie mit NIL unterdrückt wurde, und 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 dadurch die Ausgabe
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
Änderungen zu 6.3:	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)	

Beschreibung:

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. Der Wert 2 wird sonst automatisch vom Grundprogramm vorbelegt, um die Ausgabe freizuschalten. Wichtig ist dieser Befehl für Übersetzer, die beim ersten Durchlauf noch keine Ausgabe auf dem Bildschirm erzeugen sollen. Dann wird z.B. PASSFLAG = 1 gesetzt. Erst beim zweiten Durchlauf wird z.B. beim Assembler eine Ausgabe erzeugt. Dann wird PASSFLAG durch den Befehl SETPASS auf 2 gesetzt und die Ausgabe durchgeführt.

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:	56
Befehlsname:	EDIT
Befehlsgruppe:	Editor
Kurzbeschreibung:	Der Editor wird aufgerufen
Eingaberegister:	KEINE
Ausgaberegister:	d0.l = 0 oder -1, wenn Carry nicht gesetzt, sonst ohne Funktion Carry = Bei Ende mit ^KA gesetzt, sonst rückgesetzt
Zerstörte Register:	Alle außer a5
Ab Version:	3.1
Änderungen zu 4.3:	Definition der Ausgaberegister. Eine Rückmeldung des Editors, wie er beendet wurde, gibt es erst seit 6.1
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN
Siehe auch:	GETSTX(66) PUSTX(67)

Beschreibung:

Damit kann man den TEXT-EDITOR als Unterprogramm aufrufen und so Eingaben vom Benutzer verlangen. Der Text wird beginnend bei der Adresse, die durch STXTXT bestimmt ist, abgelegt. Diese Adresse kann man ebenfalls verändern, um so ein freies Textgebiet verwenden zu können.

Der Editor liefert jetzt auch eine Kennung zurück, da man ihn jetzt mit verschiedenen Befehlen verlassen kann.

1) Verlassen mit CTRL-KX
Das Register D0.L enthält den Wert 0 und das CARRY-Flag ist rückgesetzt.

2) Verlassen mit CTRL-KQ
Das Register D0.L enthält den Wert -1 oder $FFFFFFFF und das CARRY-Flag
ist rückgesetzt.

3) Verlassen mit CTRL-KA
Das CARRY-Flag ist gesetzt (Register d0 ist ohne Funktion).

Eine genaue Beschreibung des Editors liefert Kapitel 3.2.1.

Einfaches Beispiel:

Es wird der Editor aufgerufen.

```
START:
        MOVEQ   #!EDIT,D7       * Editor aufrufen
        TRAP    #1
        RTS
```

Komplexes Beispiel:

Es wird ein neuer Textstart festgelegt und ein Text kann bearbeitet werden. Der alte Textstart wird aber wieder eingestellt. Außerdem wird der neue Text über CO2 auf dem Bildschirm ausgegeben.

```
START:
        MOVEQ   #!GETSTX,D7     * Alte Textadresse holen
        TRAP    #1
        MOVE.L  D0,-(A7)        * Merken für später
        LEA     TEXT(PC),A0
        MOVE.L  A0,D0           * Adresse nach D0.L
        MOVEQ   #!PUTSTX,D7     * Neue Adresse einstellen
        TRAP    #1
        MOVEQ   #!EDIT,D7       * Editor aufrufen
        TRAP    #1
        MOVEQ   #!CLRSCREEN,D7  * Bildschirm für CO2 vorbereiten
        TRAP    #1
        MOVEQ   #!CIINIT2,D7    * Pointer für CI2 auf Textanfang
        TRAP    #1
SCHLEIFE:
        MOVEQ   #!CI2,D7        * Zeichen lesen
        TRAP    #1
        BCS.S   ENDE            * Ende, wenn letztes Zeichen
        MOVEQ   #!CO2,D7        * Zeichen ausgeben
        TRAP    #1
        BRA.S   SCHLEIFE        * Wiederholen
ENDE:
        MOVE.L  (A7)+,D0        * Alte Textadresse
        MOVEQ   #!PUTSTX,D7     * wieder einstellen
        TRAP    #1
        RTS

TEXT:
        DC.B    'Hallo',0       * Defaulttext
                                * Ende ist offen
```

Nach dem Start des Programms erscheint der Text "Hallo" auf dem Bildschirm und man befindet sich im Editor. Jetzt kann man beliebige Texte eingeben und wenn man den Editor verlässt, so wird der Text auf dem Bildschirm ausgegeben. Wichtig ist, dass man die 0 bei TEXT: nicht als Abschluss vergisst, sonst ist das Textende nicht definiert. Anstelle von CO2 kann man auch LO aufrufen und damit den eingegebenen Text auf dem Drucker ausgeben.

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
Änderungen zu 6.3: NEIN

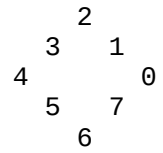
Siehe auch: FIGURXY(5) SETFIG(58)

Beschreibung:

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, so erhält man die kleinste Figur. Mit anderen Werten kann man die Größe jeweils mit dem Wert multiplizieren, der angegeben wird. 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 dabei wie folgt:



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, so 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, so 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, so muss man den FIGURXY-Befehl verwenden, der die gleiche Funktion hat.

Einfaches Beispiel:

Es wird ein deformiertes Achteck gezeichnet.

```
START:
        LEA     FIGUR(PC),A0
        MOVEQ   #2,D0
        MOVEQ   #100,D1
        MOVEQ   #100,D2
        MOVEQ   #!FIGUR,D7
        TRAP    #1
        RTS

FIGUR:
        DC.B    6,5,0,7,2,2,1,4,3,6,10
```

Komplexe 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
SCHLEIFE:
        MOVEQ   #3,D0           * Vergrößerung
        MOVEQ   #!FIGUR,D7      * Figur zeichnen
        TRAP    #1
SYNC:
        MOVEQ   #!SYNC,D7       * 20 ms (16,67 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
```

Eine besondere Möglichkeit ist es aber auch die Adresse zu wechseln, um so sich selbst bewegende Figuren über den Bildschirm bewegen zu können.

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 fest setzen (einfrieren)

Eingaberegister: KEINE
Ausgaberegister: KEINE
Zerstörte Register: KEINE

Ab Version: 3.1
Änderungen zu 4.3: NEIN
Änderungen zu 6.1: NEIN
Änderungen zu 6.3: NEIN

Siehe auch: FIGURXY(5) FIGUR(57)

Beschreibung:

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, verschwindet die alte Figur nicht von dem Bildschirm.

Einfaches Beispiel:

Es werden zwei Figuren gezeichnet, da die erste fest gesetzt wird.

```
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 fest setzen
        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 mehr mehrmals erkannt. Der Such-Bereich ist auf die CPU angepasst.
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN

Siehe auch:	GETBASIS(89)	GETVAR(90)	SETA5(91)
	GETVERS(97)	GETSN(98)	GRUND(124)
	SUCHBIBO(136)	SYSTEM(139)	SETSYS(160)
	GETSYS(161)	PATCH(162)	

Beschreibung:

Dies ist ein Befehl, der einen Speicherbereich ermitteln kann, 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 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 erfahren will, so muss man GETRAM solange 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 Debug anschalten). Nach dem Aufruf des Befehls GETRAM erscheinen in den Registern A0.L und A1.L die Werte.

Komplexes 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
        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
Änderungen zu 6.3:	NEIN

Siehe auch:

MOVETO(8)	DRAWTO(9)	CLR(16)
CLPG(17)	WAIT(18)	CMD(26)
NEWPAGE(27)	SETFLIP(34)	SETPEN(37)
ERAPEN(38)	CMDPRIN(40)	SETXOR(77)
GETXOR(78)	SETCOLOR(79)	GETCOLOR(80)
GETXY(103)	HARDCOPY(125)	GRAFIK(126)
GDPVERS(127)		

Beschreibung:

Dieses Unterprogramm steuert die automatische Bildseitenumschaltung. Dabei ist die Umschaltrate von den Variablen FLIP und FLIP1 abhängig. Die Variablen kann man mit dem Befehl SETFLIP belegen. Sie werden aber auch automatisch durch die Schildkrötenbefehle besetzt. Alle 20 ms (16,67 ms bei GDP-FPGA) werden die Variablen abgefragt. FLIP steuert dabei die Umschaltung zwischen zwei Bildseiten. Ist FLIP = 1, so wird alle 20ms 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, so wird alle 20ms zwischen den vier Bildseiten umgeschalten. Dabei wird von 0 auf 1, von 2 und 3 und von 3 auf 0 geschaltet. Ein Wert größer 1 bewirkt eine Multiplikation mit jeweils 20ms. Also 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 beliebiger Taste
        RTS
```

Die Schildkröte bleibt weiterhin eingeblendet. Wenn man AUTOFLIP nicht verwendet, so ist nur eine Seite sichtbar, jedoch ist nicht einmal definiert welche Seite. Mit NEWPAGE könnte man aber 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:	Die Cursordarstellung ist auch bei HARDSCROLL möglich.
Änderungen zu 6.3:	NEIN

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)	

Beschreibung:

Einschalten des Cursors. Normalerweise wird dies automatisch durchgeführt, wenn man die Routine CI aufruft. Wenn man aber eine große Schleife durchläuft, und will den Cursor auf dem Bildschirm sehen, so muss man diese Routine verwenden. Bevor ein Aufruf von CO, CO2 oder CI erfolgt, muss man aber den Cursor mit CURSAUS wieder ausschalten. Ferner muss man mit AUTOFLIP die Seiteumschaltung aufrechterhalten (in DELAY z.B. wird automatisch 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 CO).

Einfaches Beispiel:

Der Cursor blinkt 5 Sekunden lang und wird dann ausgeschaltet. Abbruch des Programms erfolgt 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
        MOVEQ   #!DELAY,D7      * warten
        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 ins Grundprogramm wird der Cursor wieder 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:	Der Cursor kann auch bei HARDSCROLL ausgeschaltet werden
Änderungen zu 6.3:	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)	CHAR(63)
CURON(81)	CUROFF(82)	CRLF(99)
GETLINE(100)	GETCURXY(101)	SETCURXY(102)
LSTS(117)	CO2SER(129)	

Beschreibung:

Der Cursor, der zuvor mit CURSEIN eingeschaltet wurde, wird wieder abgestellt.

Anwendung:
Normalerweise wird z.B. bei CSTS kein Cursor auf den Bildschirm gebracht. Es gibt aber Fälle bei denen man in einer Schleife durch CSTS abfragt, ob ein Zeichen angekommen ist und es dann erst mit CI einliest (z.B. das 68000 FORTH von Laboratory Microsystems für CP/M68K tut es so).

Auch dieser Befehl kann genau wie CURSEIN mit HARDSCROLL betrieben werden.

Komplexes 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:	Eine Ausgabe mit HARDSCROLL ist möglich.
Änderungen zu 6.1:	Es kann immer mit HARDSCROLL gearbeitet werden.
Änderungen zu 6.3:	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)
	CURON(81)	CUROFF(82)	CRLF(99)
	GETLINE(100)	GETCURXY(101)	SETCURXY(102)
	LSTS(117)	CO2SER(129)	

Beschreibung:

Ausgabe eines Zeichens, das im Register D0.B steht auf den Bildschirm. Dabei wird wie bei CO verfahren, jedoch sind Steuerfunktionen nicht verwirklicht. Damit können alle Zeichen von 0 bis $FF verwendet werden. Da es sich um eine interne Funktion handelt ist aber bei der Anwendung Vorsicht geboten. Man sollte wenn nicht anders möglich 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
Änderungen zu 6.3: NEIN

Siehe auch: WRITELF(6) WRITE(10)

Beschreibung:

Damit ist es möglich, einen programmierbaren Zeichengenerator zu verwenden. Man definiert sich dazu in einer 8*5 Matrix die Zeichen und kann sie mit diesem Befehl in der aktuelle Schriftgröße auf dem Bildschirm ausgeben.

Dazu zeigt das Register A0.L auf den Datenblock für das Zeichen. Die Ausgabe erfolgt an der aktuellen Koordinatenstelle, die man mit MOVETO zuvor noch einstellen kann. Als Größe wird die aktuelle Schriftgröße verwendet, die gerade im Register 3 der GDP-Karte steht.

Nach dem Aufruf befindet sich die aktuelle Schreibposition des GDP genau an der richtigen Stelle für das nächste Zeichen, so dass mehrere PROGZGE-Aufrufe hintereinander 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, so erscheint es als Punkt hell auf dem Bildschirm.

Komplexes Beispiel:

Das OHM-Zeichen wird an der Stelle 100,125 in der Größe $55 ausgegeben.

```
START:
        MOVEQ   #100,D1         * X-Position
        MOVEQ   #125,D2         * Y-Position
        MOVEQ   #!MOVETO,D7     * Koordinaten setzen
        TRAP    #1
        MOVE    #!SYSTEM,D7     * System-Informationen holen
        TRAP    #1
        AND     #7,D0           * Nur CPU lassen
        MULS    #$FF73,D0       * GDP-Register 3 (Size)
        MOVEA.L D0,A0
        MOVE.B  #$55,(A0)       * Größe einstellen
        LEA     ZEICHEN(PC),A0  * Adresse der Daten
        MOVEQ   #!PROGZGE,D7    * Zeichen ausgeben
        TRAP    #1
        RTS

Zeichen:
        DC.B %10011110          * Ohm-Zeichen
        DC.B %11100001
        DC.B %00000001
        DC.B %11100001
        DC.B %10011110
```

Es soll nicht verschwiegen werden, dass der Bildaufbau dabei relativ langsam erfolgt. Daher sollte man einer vektoriellen Darstellung den Vorzug geben, wann immer sie möglich ist.

Die Routine wird intern für die Darstellung der deutschen Sonderzeichen verwendet, da eine vektorielle Darstellung aufgrund der vielen Schriftgrößen zu aufwendig wäre.

TRAP-Nummer:	65
Befehlsname:	ASSEMBLE
Befehlsgruppe:	Assembler
Kurzbeschreibung:	Es wird der Assembler 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. Ein Abbruch durch CTRL-C ist möglich. Beim ersten Durchlauf wird nichts im RAM abgelegt.
Änderungen zu 6.1:	Es gibt immer eine Macrotabelle mit einigen Einträgen für die LINK-Befehle.
Änderungen zu 6.3:	NEIN
Siehe auch:	SETERR(53) GETERR(54) GETORG(68) PUTORG(69) ASSERR(120)

Beschreibung:

Damit kann der Assembler als Unterprogramm aufgerufen werden. Dieser 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 und automatisch zu übersetzen.

Die Übersetzung beginnt beim aktuellen Textstart (STXTXT), der z.B. mit dem Befehl PUTSTX verändert werden kann.

Der Programmcode wird an der Standart-ORG-Adresse abgelegt. Dies kann mit dem Befehl ORG verändert werden.

Wurde der Assemblerlauf mit CTRL-C abgebrochen, so ist das CARRY-Flag gesetzt. Außerdem wird der Fehlerzähler (ERRCNT) um eins erhöht, so dass mindestens ein Fehler vorliegt.

Eine ausführliche Beschreibung des Assemblers bietet Kapitel 3.2.2.

Komplexes Beispiel:

Bei Start des Programms wird zunächst der Assembler aufgerufen und 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   #!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 lassen
NEU:
        DS.W    100             * Hier kommt Code hin
```

TRAP-Nummer: 66
Befehlsname: GETSTX
Befehlsgruppe: Editor
Kurzbeschreibung: Die 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
Änderungen zu 6.3: NEIN

Siehe auch: EDIT(56) PUTSTX(67)

Beschreibung:

Damit kann man die aktuelle Textadresse 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
        TRAP    #1
        ADD.L   #1024*5,D0
        MOVEQ   #!PUTSTX,D7
        TRAP    #1
        RTS
```

Komplexes Beispiel:

Die aktuelle Textadresse wird kurzfristig auf die Adresse von NEU gelegt. Dort kann ein Text eingegeben werden. Beim Verlassen des Editors wird aber 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
        MOVEQ   #!PUTSTX,D7     * Einstellen
        TRAP    #1
        MOVEQ   #!EDIT,D7       * Editor aufrufen
        TRAP    #1
        MOVE.L  (A7)+,D0        * Alte Adresse
        MOVEQ   #!PUTSTX,D7     * Wie einstellen
        TRAP    #1
        RTS

NEU:
        DC.B 0                  * Leerer Text dort
```

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
Änderungen zu 6.3: NEIN

Siehe auch: EDIT(56) GETSTX(66)

Beschreibung:

Damit kann man den Textanfang umdefinieren, genauso 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, so passiert zunächst noch nichts aufregendes. Erst wenn man anschließend in den Editor zurückgeht, so fehlt der Programmtext. In der unteren Zeile sieht man, das der Textanfang nun auf einer neuen Adresse liegt. Wenn man den alten Programmtext wieder sehen will, so kann man dies z.B. mit dem OPTIONEN-MENÜ durch Eingabe eines neuen Textstartes tun, oder man schreibt nochmals das obere Programm, allerdings mit dem alten Adresse.

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
Änderungen zu 6.3: NEIN

Siehe auch: SETERR(53) GETERR(54) ASSEMBLE(65)
PUTORG(69) ASSERR(120)

Beschreibung:

Damit kann man den Defaultwert des Programmzählers für die Codeablage durch den Assembler abfragen. Im Register D0.L steht dann das Ergebnis. Dieser Wert wird nur durch die PUTORG-Anweisung aber nicht durch den ORG-Befehl im Assembler verändert.

Komplexes Beispiel:

Die voreingestellte Übersetzungsadresse wird ausgegeben.

```
START:
        MOVEQ   #!GETORG,D7     * Übersetzungsadresse Assembler holen
        TRAP    #1
        LEA     BUFFER(PC),A0   * Ziel für Ausgabe
        MOVE    #!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
```

Normalerweise erscheint der Wert $11C00 auf dem Bildschirm. Dorthin wird nämlich der Assemblercode gelegt, wenn man kein 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
Änderungen zu 6.3:	NEIN
Siehe auch:	SETERR(53) GETERR(54) ASSEMBLE(65) SETORG(68) ASSERR(120)

Beschreibung:

Damit kann man den Defaultwert für die Assembler-Codeablage ändern. Im Register D0.L steht dann die neue Adresse. Dies 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.B    200             * Freiraum für Macro-Tabelle
ENDE:
```

Wenn man das Programm startet, so passiert noch gar nichts. Wenn man aber dann den Assembler erneut startet, so legt er den neuen Programmcode nicht auf der normalen Adresse ab, sondern direkt hinter den alten Programmcode. Wenn man dann das Programm nochmals startet, so 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
Änderungen zu 6.3:	NEIN
Siehe auch:	PRINT2X(41) PRINT4X(42) PRINT6X(43) PRINT8X(44) PRINT8B(45) PRINT4D(46) PRINTV8D(71)

Beschreibung:

Damit kann eine 32-Bit-Größe, die im Register D0.L steht, dezimal ausgegeben werden. Die Ausgabe erfolgt dabei 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.

Komplexes 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
        MOVEQ   #2,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
Änderungen zu 6.3:	NEIN
Siehe auch:	PRINT2X(41) PRINT4X(42) PRINT6X(43) PRINT8X(44) PRINT8B(45) PRINT4D(46) PRINT8D(70)

Beschreibung:

Damit kann eine 32-Bit-Größe, die im Register D0.L steht, dezimal ausgegeben werden. Die Ausgabe erfolgt dabei mit 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.

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
        MOVEQ #2,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
```

Komplexes Beispiel:

Man kann Rechenaufgaben, sogar mit Klammerrechnung eingeben und das Ergebnis wird dann nach der Eingabe von <RETURN> auf dem Bildschirm angezeigt. Dabei erfolgt die Rechnung natürlich nur mit ganzen Zahlen (siehe WERT-Befehl). Das Programm wird abgebrochen, wenn nur <RETURN> gedrückt wird.

```
START:
        LEA BUFFER(PC),A0       * Zieladresse für Text
        MOVEQ #$22,D0           * Schriftgröße
        MOVEQ #2,D1             * X-Position
        MOVEQ #100,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 wurde
        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 #2,D1             * X-Position
        MOVEQ #120,D2           * Y-Position
        MOVEQ #!WRITE,D7        * Ausgabe auf Schirm
        TRAP #1
        BRA.S START             * Wiederholen

ENDE:
        RTS

BUFFER:
        DS.B 32
```

TRAP-Nummer:	72
Befehlsname:	MULS32
Befehlsgruppe:	Berechnungen
Kurzbeschreibung:	Es werden zwei 32-Bit-Zahlen multipliziert
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.
Änderungen zu 6.3:	NEIN
Siehe auch:	SIN(23) COS(24) WERT(29) DIVS32(73) ADJ360(83) RND(96)

Beschreibung :

Der 68000/8 besitzt eine eingebaute Multiplikation, die aber nur mit 16-Bit Größen arbeiten kann. Da man für bestimmte Rechnungen aber auch eine 32-Bit-Multiplikation benötigt, wurde diese Routine hinzugefügt. Der Multiplikand wird im Register D0.L übergeben, der Multiplikator im Register D2.L. Das Ergebnis erscheint 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. Beim 68020 ist der Befehl aus Kompatibilitätsgründen auch vorhanden. Es wird aber die interne Multiplikation ausgeführt, die wesentlich schneller ist.

Einfaches Beispiel:

Im Register D0.L erscheint nach dem Aufruf die Zahl $7FFFFFFE. Man kann das Ergebnis z.B. 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 der eingebauten 68000/8 Multiplikation (MULU, MULS) sollte man bei zeitkritischen Problemen versuchen mit den 16-Bit-Operationen auszukommen. Beim 68020 erfolgt die Multiplikation natürlich mit höchster 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.
Änderungen zu 6.3:	NEIN
Siehe auch:	SIN(23) COS(24) WERT(29) MULS32(72) ADJ360(83) RND(96)

Beschreibung :

Der 68000/8 besitzt eine eingebaute Division, die aber nur mit 16 Bit arbeitet. Daher wurde diese Routine dazugenommen, die auch mit 32-Bit-Werten arbeiten kann. Dazu wird im Register D0.L der Divisor abgelegt und im Register D2.L steht der Dividend. Das Ergebnis der Division steht anschließend im Register D0.L und der Rest der Division im Register D1.L. Die Division wird vorzeichenbehaftet durchgeführt. Das Vorzeichen des Restes (D1.L) entspricht dem Vorzeichen des Divisors (D0.L vorher). Beim 68020 wird diese Operation durch einen internen Befehl durchgeführt.

Operation: D0.L / D2.L -> D0.L REST D1.L

Einfaches Beispiel:

Wenn man das Programm im Einzelschritt durchläuft, so 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 D0.L / Rest D1.L
```

Da die Division mit DIVS32 langsamer erfolgt als mit der eingebauten 68000/8 Division (DIVU, DIVS) sollte man bei zeitkritischen Problemen versuchen mit den 16-Bit-Operationen auszukommen. Beim 68020 erfolgt die Division natürlich mit höchster 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.
Änderungen zu 6.3:	NEIN
Siehe auch:	FLOPPY(75) GETFLOP(76) SRDISK(156) SETF2S(157) GETF2S(158) GETSRD(159)

Beschreibung :

Dieser Befehl ist für den Anschluss einer Floppy gedacht. 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, so dass man sie normalerweise nicht benötigt, wenn man mit der Floppy arbeiten will.

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 von frischen Disketten benötigt man ein eigenes Programm, das nicht im Grundprogramm enthalten ist.

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 auftrat
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.
Änderungen zu 6.3:	Wenn flo2srd gesetzt ist, werden die Zugriffe auf Floppy4 auf SRDISK umgeleitet.
Siehe auch:	FLINIT(74) GETFLOP(76) SRDISK(156) SETF2S(157) GETF2S(158) GETSRD(159)

Beschreibung :

Diese Routine ist für den Betrieb mit der FLO2-Baugruppe bestimmt. Damit kann man alle Operation durchführen, die der FLOPPY-Controler zur Verfügung stellt. Außerdem ist eine Funktion vorhanden, die den letzten aufgetretenen Fehler zurückgibt.

Immer wenn ein Fehler auftrat, so wird im Register D0.L der Wert $FFFFFFFF (-1) zurückgegeben. Gleichzeitig ist das CARRY-Flag gesetzt.

Weiterhin kann die FLOPPY auf die serielle Karte gelenkt werden. Dabei sind aber nur die Funktionen SEKTOR LESEN und SEKTOR SCHREIBEN möglich.

Wenn die flo2srd Variable (siehe SETF2S) gesetzt ist, werden alle Zugriffe auf Floppy4 auf den Befehl SRDISK umgeleitet. Dabei sind aber nur die Funktionen SEKTOR LESEN und SEKTOR SCHREIBEN möglich.

Registerbelegung:

A0.L Ist die Adresse für Ziel oder Quelle

Dieses Register wird immer 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, so muss im Register D3.B die Steprate stehen (0-7) und eventuell das SSO-Bit (Bit 7). Bei diesem 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, so werden immer 1024 Bytes gelesen, egal welche Werte in D4.B eingestellt sind. Außerdem werden vor dem Lesen der 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 den Routinen SI und 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, so 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 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. Dieser 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. enthalten sind.

6 = Restore

Durch diesen Befehl wird der Kopf des Laufwerks auf die Spur Null gefahren. (Register d4.b benötigt

7 = Spur suchen

Mit diesem Befehl kann der Kopf auf eine bestimmte Spur gefahren werden. Dazu muss im Register D3.B die richtige 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. (Keine weiteren Register benötigt)

9 = Schreibkopf eine Spur nach außen
Der Kopf des Laufwerks wird um eine Spur nach außen hin gefahren. (Keine weiteren Register benötigt).

10 = Letzten Fehler lesen
Wurde bei einem Zugriff auf das Laufwerk vom Controler ein Fehler entdeckt, so 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, so handelte es sich bei dem Befehl, bei dem der Fehler auftrat um einen Befehl aus der Gruppe 1, sonst war es ein Befehl aus der Gruppe 2 oder 3. Wurde auf die serielle Karte zugegriffen, so sollte dieser Befehl nicht aufgerufen werden, da die Werte dann noch vom letzten Zugriff auf die FLOPPY stammen, da sie in dem Falle nicht verändert werden. Die Bedeutung des einzelnen Bits kann dem FLO-Handbuch entnommen werden. Die FLAGS sind entsprechend dem Wert in D0.B gültig.

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:
0...k (k=34,39,76,79 je nach Laufwerk)
Die Anzahl der Spuren hängt von der Art des Laufwerks ab. Bei dem Befehl D1.W = 0 (Steprate setzen) hat dieses Register eine besondere Funktion. Hier wird angegeben, welchen Wert die Steprate haben soll. Diese gibt an, wie schnell der Laufwerkskopf positioniert werden soll.

0 - 3 = Stepraten bei Mini-Laufwerken
4 - 7 = Stepraten bei Maxi-Laufwerken für Mini-Laufwerken
Der kleinere Wert ist dabei 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 benutzt 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.

Soll auch auf die Seite 1 eines Laufwerks zugegriffen werden, so 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 Bit 0 für Laufwerk 1, Bit 1 für Laufwerk 2 usw. Es ist aber auch eine andere Codierung denkbar.
4 : Dichte (1 = Einfache Dichte)
5 : Laufwerk (1 = Minilaufwerk)
6 : Motorsteuerung (0 = Motor ein)
7 : Seitenauswahl (1 = Zugriff auf Seite 1 ist möglich)
Dieses Bit muss normalerweise immer auf 1 gesetzt werden. Zusätzlich muss bei einem Zugriff auf Seite 1 das Bit 7 der Steprate gesetzt sein.

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.L übergeben (nicht implementierter Befehl), so wird auch in D0.L 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 wurde. Dazu wird eine 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 A
      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 überträgt die richtige Anzahl von Bytes.
Abschließend noch eine Zusammenstellung gebräuchlicher Dichtecodes für Laufwerk A. Am Laufwerk muss die Laufwerksnummer eingestellt werden. A entspricht auch der Nr 1, D der Nr 4.

8 Zoll einfache Dichte:	$11
8 Zoll doppelte Dichte:	$01
5 1/4 Zoll einfache D.:	$31
5 1/4 Zoll doppelte D.:	$21

Für 3 Zoll und 3 1/2-Zoll-Laufwerke gibt es jeweils aus dem oberen Beispiel einen gültigen Code.

Wenn man die Rückseite eines Doppelkopflaufwerks ansprechen will, so addiert man einfach den Wert $80 auf den Laufwerks-Dichtecode. Damit der Zugriff auch 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, wenn nicht richtig formatiert
Zerstörte Register:	KEINE
Ab Version:	4.0
Änderungen zu 4.3:	Der Befehl wird nicht mehr abgebrochen, wenn keine Diskette vorhanden ist.
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	Es ist eine maximale Wartezeit vorgegeben, wenn innerhalb dieser keine Antwort erfolgt, wird ein Fehler ausgegeben. Die flo2srd Variable wird berücksichtigt.

Siehe auch:	FLINIT(74)	FLOPPY(75)	SRDISK(156)
	SETF2S(157)	GETF2S(158)	GETSRD(159)

Beschreibung :

Damit ist es möglich, bei einer formatierten Diskette den Dichtecode automatisch zu ermitteln. Dazu wird nur im Register D4.B der Laufwerkscode angegeben und nach dem Aufruf steht im Register D$.B der Laufwerkscode und der Dichtecode, so 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 vorwiegend zum automatischen Start eines Programms auf der Spur 0 gedacht. Außerdem wird das Carry-Flag gesetzt, wenn ein Fehler auftrat und im Register D0.W erscheint der Wert $FFFF sonst der Wert 0. Der Fehler, der auftrat kann aus den Bits in d0.b ermittelt werden, die dem Statusregister der FLO-Baugruppe entsprechen. Wenn innerhalb der vorgegebenen Wartezeit keine Rückmeldung von der FLO-Baugruppe kommt, wird der Befehl mit einer Fehlermeldung abgebrochen. Bei gesetzter flo2srd Variablen (siehe SETF2S), wird der Befehl sofort mit einem OK beendet.

Einfaches Beispiel:

Mit diesem Programm ist es möglich, den ersten Sektor einer Diskette zu lesen, unabhängig davon, ob Mini- oder Maxilaufwerk ob einfache oder doppelte Dichte.

```
START:
      MOVEQ #1,D4             * Laufwerk A
      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 eine Weile (< 1 Sekunde) braucht, um die Dichte zu ermitteln, sollte man GETFLOP nicht vor jedem FLOPPY-Befehl verwenden, sondern nur einmal am Anfang eines Programms.

Beispiel für Codierung der Laufwerknummern:

Laufwerk A Seite 0 $01
Laufwerk B Seite 1 $82
Laufwerk C Seite 1 $84
Laufwerk D 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
Änderungen zu 6.3:	NEIN

Siehe auch:

MOVETO(8)	DRAWTO(9)	CLR(16)
CLPG(17)	WAIT(18)	CMD(26)
NEWPAGE(27)	SETFLIP(34)	SETPEN(37)
ERAPEN(38)	CMDPRIN(40)	AUTOFLIP(60)
GETXOR(78)	SETCOLOR(79)	GETCOLOR(80)
GETXY(103)	HARDCOPY(125)	GRAFIK(126)
GDPVERS(127)		

Beschreibung :

Dies ist ein Befehl, der für eine GDP-Erweiterung gedacht ist. 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 ist 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.

Komplexes Beispiel:

```
START:
      CLR D1                  * Linie von 0,0
      CLR D2
      MOVEQ #!MOVETO,D7
      TRAP #1
      MOVEQ #100,D1           * nach 100,100 zeichnen
      MOVEQ #100,D2
      MOVEQ #!DRAWTO,D7
      TRAP #1
      MOVEQ #1,D0
      MOVEQ #!SETXOR,D7       * XOR-Modus anschalten
      TRAP #1
      CLR D1
      CLR D2
      MOVEQ #!MOVETO,D7       * Auf Position 0,0
      TRAP #1
      MOVE #!SYSTEM,D7  * System-Informationen lesen
      TRAP #1
      AND #7,D0         * Nur CPU 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
Änderungen zu 6.3:	NEIN

Siehe auch:

MOVETO(8)	DRAWTO(9)	CLR(16)
CLPG(17)	WAIT(18)	CMD(26)
NEWPAGE(27)	SETFLIP(34)	SETPEN(37)
ERAPEN(38)	CMDPRIN(40)	AUTOFLIP(60)
SETXOR(77)	SETCOLOR(79)	GETCOLOR(80)
GETXY(103)	HARDCOPY(125)	GRAFIK(126)
GDPVERS(127)		

Beschreibung :

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
Änderungen zu 6.3:	NEIN

Siehe auch:

MOVETO(8)	DRAWTO(9)	CLR(16)
CLPG(17)	WAIT(18)	CMD(26)
NEWPAGE(27)	SETFLIP(34)	SETPEN(37)
ERAPEN(38)	CMDPRIN(40)	AUTOFLIP(60)
SETXOR(77)	GETXOR(78)	GETCOLOR(80)
GETXY(103)	HARDCOPY(125)	GRAFIK(126)
GDPVERS(127)		

Beschreibung :

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 als Farbe weiß gedacht, und so wird er vom Grundprogramm auch nach dem RESET eingestellt. Wenn man selbst eine Farberweiterung bauen will, so 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, so ist die jeweilige Farbe aktiviert. Wenn das Bit "ton" auf 0 liegt, so wird die Farbe geschrieben, wenn es auf 1 liegt, so 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
```

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
Änderungen zu 6.3:	NEIN

Siehe auch: MOVETO(8), DRAWTO(9), CLR(16), CLPG(17), WAIT(18), CMD(26), NEWPAGE(27), SETFLIP(34), SETPEN(37), ERAPEN(38), CMDPRIN(40), AUTOFLIP(60), SETXOR(77), GETXOR(78), SETCOLOR(79), GETXY(103), HARDCOPY(125), GRAFIK(126), GDPVERS(127)

Beschreibung :

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. 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.
Änderungen zu 6.3:	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)	CUROFF(82)	CRLF(99)
	GETLINE(100)	GETCURXY(101)	SETCURXY(102)
	LSTS(117)	CO2SER(129)	

Beschreibung:

Damit wird die interne Variable "CURON" auf 1 gesetzt. Von nun an erscheint ein Cursor bei der Ausgabe durch CO, CO2 und CHAR. Die Variable "CURON" wird normalerweise 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 gesagt wird, dass er z.B. bei CI gezeigt wird.

Einfaches Beispiel:

Die automatische Bildseitenumschaltung wird aktiviert und der Cursor wird automatisch dargestellt.

```
START:
        MOVEQ   #10,D0          * Flip einschalten
        CLR     D1
        MOVEQ   #!SETFLIP,D7
        TRAP    #1
        MOVEQ   #!CURON,D7      * Cursor einschalten, falls er aus war
        TRAP    #1
        RTS
```

Auf dem Bildschirm erscheint ein blinkender Cursor. Allerdings ist die Position nicht definiert und in Sonderfällen kann er auch unsichtbar bleiben, denn CLRSCREEN wurde in unserem Beispiel nicht aufgerufen. Jedoch verwendet der Assembler auch CLRSCREEN und so sind die meisten Variablen schon ok.

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.
Änderungen zu 6.3:	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)	CRLF(99)
GETLINE(100)	GETCURXY(101)	SETCURXY(102)
LSTS(117)	CO2SER(129)	

Beschreibung:

Damit kann der Cursor ausgeschaltet werden, wenn man z.B. bei einer Eingabe keinen Cursor auf dem Bildschirm sehen will. Dazu wird intern die Variable "CURON" auf 0 gesetzt (Siehe auch CUROFF).

Einfaches Beispiel:

Der Buchstabe "A" wird ausgegeben und 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, so 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
Änderungen zu 6.3:	NEIN
Siehe auch:	SIN(23) COS(24) WERT(29) MULS32(72) DIVS32(73) RND(96)

Beschreibung:

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. Die Eingabe kann dabei auch negative Werte in der Zweierkomplementdarstellung verarbeiten. 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 jetzt 3&1
        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
Änderungen zu 6.3:	NEIN
Siehe auch:	ZUWEIS(30) SYMCLR(85) GETSYM(86) GETNEXT(87) PUTNEXT(88)

Beschreibung:

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   #!PRTSYI,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.
Änderungen zu 6.3:	NEIN
Siehe auch:	ZUWEIS(30) PRTSYM(84) GETSYM(86) GETNEXT(87) PUTNEXT(88)

Beschreibung:

Damit kann man die Symboltabelle löschen. Dieser Befehl ist wichtig, z.B. für automatische Übersetzer, 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 das Menü "4 = Symbole" aufruft, so 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
Änderungen zu 6.3:	NEIN
Siehe auch:	ZUWEIS(30) PRTSYM(84) SYMCLR(85) GETNEXT(87) PUTNEXT(88)

Beschreibung:

Im Register A0.L und D0.L wird die Adresse des Symboltabellenstarts zurückgegeben. Wenn man keine Symbole verwendet, so kann man z.B. in eigenen Programmen dorthin seine Variable legen. Danach sollte man aber sicherheitshalber mit SYMCLR die Symboltabelle löschen, um keine Fehlinformation 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
Änderungen zu 6.3:	NEIN
Siehe auch:	ZUWEIS(30) PRTSYM(84) SYMCLR(85) GETSYM(86) PUTNEXT(88)

Beschreibung:

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 Variable 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 auf der nächste Einträge 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 holen
        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
Änderungen zu 6.3:	NEIN
Siehe auch:	ZUWEIS(30) PRTSYM(84) SYMCLR(85) GETSYM(86) GETNEXT(87)

Beschreibung:

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, so wird beim Erzeugen von neuen Symbolen 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 Eintrage 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
Änderungen zu 6.3:	NEIN

Siehe auch: GETRAM(59) GETVAR(90) SETA5(91)
GETVERS(97) GETSN(98) GRUND(124)
SUCHBIBO(136) SYSTEM(139) SETSYS(160)
GETSYS(161) PATCH(162)

Beschreibung:

Da man das Grundprogramm ab Version 4.0 verschieben kann, gibt es einen Befehl, mit dem man die aktuelle Anfangsadresse des Grundprogramms ermitteln kann. Das Ergebnis erscheint in Register A0.L und D0.L. Der Wert ist normalerweise = 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. Dies 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
Änderungen zu 6.3:	NEIN

Siehe auch:	GETRAM(59)	GETBASIS(89)	SETA5(91)
	GETVERS(97)	GETSN(98)	GRUND(124)
	SUCHBIBO(136)	SYSTEM(139)	SETSYS(160)
	GETSYS(161)	PATCH(162)	

Beschreibung:

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, wird der Wert davon abweichen. Dies ist 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
Änderungen zu 6.3:	NEIN

Siehe auch: GETRAM(59) GETBASIS(89) GETVAR(90) GETVERS(97) GETSN(98) GRUND(124) SUCHBIBO(136) SYSTEM(139) SETSYS(160) GETSYS(161) PATCH(162)

Beschreibung:

Im Grundprogramm der Version größer 4.0 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 normalerweise nicht für eigene Zwecke verwenden. Wenn man mit dem TRAP-Befehl arbeitet, so wird das Register A5.L automatisch zurückgestellt, wenn man aber JSR verwendet, so verlässt sich das Grundprogramm darauf, dass der Inhalt von A5.L gültig ist. Wenn man A5.L verändert, so kann man mit dem Aufruf SETA5 den Wert wieder restaurieren.

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
Änderungen zu 6.3:	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)

Beschreibung:

Damit kann man die Schildkröte auf eine absolute Koordinate positionieren. Dabei 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, die Richtung der Schildkröte wird nicht verändert.

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
Änderungen zu 6.3:	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)		

Beschreibung:

Damit kann man den aktuellen Standpunkt der Schildkröte ermitteln. Im Register D1.W wird die X-Koordinate geliefert und im Register D2.W die Y-Koordinate. Dabei wird beidesmal der Wertebereich 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, 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
Änderungen zu 6.3:	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)

Beschreibung:

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 und 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
        MOVE    #180,D0
        MOVEQ   #!AUFK,D7       * Nun absoluten Winkel einstellen
        TRAP    #1
        RTS
```

Komplexes 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 -150 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
Änderungen zu 6.3:	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)

Beschreibung:

Damit lässt sich die aktuelle Bildrichtung 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, und dann 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 oder hexadezimal $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 Zufallswertes
Ausgaberegister:	d0.l = Zufallwert
Zerstörte Register:	KEINE
Ab Version:	4.0
Änderungen zu 4.3:	NEIN
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN

Siehe auch:	SIN(23)	COS(24)	WERT(29)
	MULS32(72)	DIVS32(73)	ADJ360(83)

Beschreibung:

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.W zurückgeliefert. Wenn z.B. im Register D0.W der Wert 6 übergeben wurde, so erhält man eine Zufallszahl zwischen 0 und 5. Beim Wert 0 in D0.W wird als Ausgabe eine Bereich von 0 bis $FFFF ausgegeben.

Komplexes 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
Änderungen zu 6.3:	NEIN

Siehe auch:	GETRAM(59)	GETBASIS(89)	GETVAR(90)
	SETA5(91)	GETSN(98)	GRUND(124)
	SUCHBIBO(136)	SYSTEM(139)	SETSYS(160)
	GETSYS(161)	PATCH(162)	

Beschreibung:

Damit wird die auf Adresse $40C enthaltene Versionsnummer des Grundprogramms im Register D0.L übergeben. Version 6.2 liefert zum Beispiel den Wert $620.

Dieser Befehl ist wichtig, wenn man Programme konstruiert, die von Spezialitäten 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
Änderungen zu 6.3:	NEIN

Siehe auch: GETRAM(59) GETBASIS(89) GETVAR(90) SETA5(91) GETVERS(97) GRUND(124) SUCHBIBO(136) SYSTEM(139) SETSYS(160) GETSYS(161) PATCH(162)

Beschreibung:

Damit wird der Wert auf 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
```

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 USERCO2 immer auf 1 gesetzt
Zerstörte Register:	d0
Ab Version:	4.0
Änderungen zu 4.3:	NEIN
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	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)	

Beschreibung:

Damit wird 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, so wird der Bildschirminhalt um eins 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.

Komplexes Beispiel:

Der Buchstabe "B" steht direkt unter dem Buchstaben "A".

```
START:
        MOVEQ   #!CLRSCREEN,D7  * Bildschirm für CO2 vorbereiten
        TRAP    #1
        MOVEQ   #'A',D0         * Buchstaben 'A' ausgeben
        MOVEQ   #!CO2,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 = Zeilennummer (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
Änderungen zu 6.3:	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)	

Beschreibung:

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, Formularen oder Steuerungen realisieren.

Komplexes Beispiel:

Nach Start des Programms erscheint der Cursor links oben. Nun kann man beliebig auf dem Bildschirm Texte schreiben. Dazu kann man auch die Sonderfunktionen des CO2-Befehls (siehe CO2) verwenden. Am Schluss gibt man CTRL-D (Erst die Taste CTRL drücken, dann dazu die Taste D) ein. Der Bildschirminhalt wird auf den Drucker ausgegeben. Mit CTRL-C wird das Programm abgebrochen.

```
START:
        MOVEQ   #!CLRSCREEN,D7  * Bildschirm für CO2 vorbereiten
        TRAP    #1
SCHLEIFE:
        MOVEQ   #!CI,D7         * Zeichen lesen
        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
        BRA.S   SCHLEIFE        * Weitere Eingaben
ENDE:
        RTS
```

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
Änderungen zu 6.3:	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)	

Beschreibung:

Diese Routine ist im Zusammenhang mit CO2, CO und CHAR zu benutzten.

Im Register D1.L steht 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.

Komplexes Beispiel:

Mit diesem Programm kann man auf dem Bildschirm Zeichen eingeben. Wenn man die Tasten CTRL-D drückt, so erscheint die aktuelle Zeile, auf der man sich gerade befindet auf der untersten (25-ten) Zeile, blinkend. Das Blinken kommt daher, weil die Zeile nur in eine Bildseite eingeschrieben wurde.

Mit CTRL-C kann das Programm 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   * Adressen der Cursorzeile lesen
        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
Änderungen zu 6.3:	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)	

Beschreibung:

Auch dieses Routine funktioniert im Zusammenhang mit CO2, CO und CHAR.

Damit 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 20sten 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
Änderungen zu 6.3:	NEIN

Siehe auch:

MOVETO(8)	DRAWTO(9)	CLR(16)
CLPG(17)	WAIT(18)	CMD(26)
NEWPAGE(27)	SETFLIP(34)	SETPEN(37)
ERAPEN(38)	CMDPRIN(40)	AUTOFLIP(60)
SETXOR(77)	GETXOR(78)	SETCOLOR(79)
GETCOLOR(80)	HARDCOPY(125)	GRAFIK(126)
GDPVERS(127)		

Beschreibung:

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.

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 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.
Änderungen zu 6.3:	Unterstützung für die SER2 integriert.

Siehe auch:	SO(105)	SISTS(106)	SOSTS(107)
	SIINIT(108)	SER(128)	SI2(138)
	SETSER(150)	GETSER(151)	

Beschreibung:

Eingabe eines Zeichens über die aktuelle serielle Schnittstelle. Es wird ein Zeichen von einer seriellen Schnittstelle 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 solange gewartet bis ein Zeichen angekommen ist.

Wie serielle Baugruppe miteinander verbunden werden und wie man sie einsetzen kann ist im Handbuch der SER beschrieben, weshalb hier auf Anschlussbelegungen verzichtet wird.

Komplexes Beispiel:

Wenn man das Programm startet, werden alle ankommenden Zeichen auf dem Bildschirm ausgegeben. Dabei wurde hier eine Baudrate von 9600 Baud eingestellt. Wenn man andere Baudraten haben will, so 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     * Wert 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 CURSEIN, CURSAUS 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
Änderungen zu 6.3:	Unterstützung für die SER2 integriert.

Siehe auch:	SI(104)	SISTS(106)	SOSTS(107)
	SIINIT(108)	SER(128)	SI2(138)
	SETSER(150)	GETSER(151)	

Beschreibung:

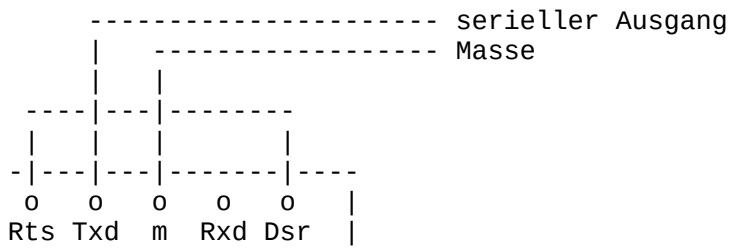
Ausgabe eines Zeichens über eine serielle Schnittstelle. Das Zeichen steht zuvor im Register D0.B und wird mit Hilfe der SER-/SER2-Baugruppe ausgegeben. Die Ausgabe erfolgt bei der SER aber nur, wenn der Eingang DSR auf +12V liegt, sonst wird die Ausgabe gesperrt und das Programm wartet bis die Bedingung 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.

Komplexes Beispiel:

Die serielle Baugruppe initialisiert und 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   #$5A,D0         * Testwert
        MOVEQ   #!SO,D7         * Zeichen ausgeben
        TRAP    #1
        MOVEQ   #!CSTS,D7       * Abbruch durch Taste ?
        TRAP    #1
        BEQ.S   SCHLEIFE        * Nein, dann wiederholen
        RTS
```

Wenn der Eingang DSR nicht beschaltet wird, so 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
Änderungen zu 6.3:	Unterstützung für die SER2 integriert.

Siehe auch:	SI(104)	SO(105)	SOSTS(107)
	SIINIT(108)	SER(128)	SI2(138)
	SETSER(150)	GETSER(151)	

Beschreibung:

Damit kann man prüfen, ob ein Zeichen an der aktuellen 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.

Komplexes Beispiel:

Mit diesem Programm arbeitet der Computer wie ein Terminal. Zeichen, die auf der Tastatur eingegeben werden, werden über die serielle Schnittstelle ausgegeben und 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
```

Wenn man die Leitungen RxD mit TxD verbindet und RTS mit DSR, so kann man auf dem Bildschirm schreiben, denn die Zeichen, die man eintippt, erscheinen auf dem Bildschirm. Dies ist auch der beste Test für die serielle Schnittstelle.

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
Änderungen zu 6.3:	Unterstützung für die SER2 integriert.

Siehe auch:	SI(104)	SO(105)	SISTS(106)
	SIINIT(108)	SER(128)	SI2(138)
	SETSER(150)	GETSER(151)	

Beschreibung:

Im Register D0.L erscheint der Wert $FFFFFFFF, wenn der Ausgabebuffer des aktuellen 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.

Komplexes Beispiel:

Es wird mit der Schildkrötengrafik ein Kreis gezeichnet und 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
```

Wenn man die Befehle "JSR @SOSTS" und "BEQ.S SCHLEIFE" weglässt, so wird der Kreis nur sehr langsam durchlaufen, denn dann wartet der Computer jedes Mal auf die Beendigung der Ausgabe.

Diesen Effekt kann man aber z.B. bei 9600 Baud nicht mehr so deutlich sehen, denn sobald der Computer zum Durchlauf der Schleife eine ähnlich große Zeit wie für Übertragungsdauer 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 ausgegeben 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 a0.l = Parameteradresse für SER2
Ausgaberegister:	KEINE
Zerstörte Register:	KEINE
Ab Version:	4.0
Änderungen zu 4.3:	NEIN
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	Unterstützung für die SER2 integriert.

Siehe auch: SI(104) SO(105) SISTS(106) SOSTS(107) SER(128) SI2(138) SETSER(150) GETSER(151)

Beschreibung:

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 Control-Register und im Register D1.B steht der Wert für das Command-Register. Die Abweichungen der FPGA-SER sind bei den Registern angegeben.

Auch ist es möglich die SER2-Karte hiermit zu initialisieren. Dazu muss das Bit #4 in D0.B auf 0 gesetzt werden, dies ist normalerweise die Kennung für einen externen Takt. Da dieser bei der SER-Karte aber nicht genutzt wird, wird es hier als Kennung für die SER2-Karte "missbraucht". Für die Initialisierung der SER2-Karte wird in A0.L die Adresse eines Parameterblocks übergeben.

Control-Register (D0.B):

Bits

0 - 3 Ausgabegeschwindigkeit einstellen
- 0000 16x externer Takt
- 0001 50 Baud
- 0010 75 Baud
- 0011 109.92 (115200) Baud
- 0100 134.58 (57600) Baud
- 0101 150 (38400) 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

Die Werte in Klammern beziehen sich auf die FPGA Variante der SER-Karte.
Bit 0 dient bei der SER2 zur Auswahl des zu initialisierenden Kanals. 0 = Kanal A, 1 = Kanal B.

4 Dieses Bit dient normalerweise zum umschalten zwischen internem und externem Takt (0 = externer Takt, 1 = interner Takt). Da die Standart SER-Karte nicht über einen externen Takt verfügt, wird dieses Bit zur Auswahl der SER-Karte verwendet.
1 = SER-Karte, 0 = SER2-Karte

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

0 0 = DTR auf 1-Signal
1 = Rxd und Txd freigeben

1 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 diesen beiden Tabellen kann der serielle Baustein programmiert werden. Dazu werden aus den binären Muster die gewünschten Bytes zusammengesetzt. Hier noch ein paar 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

Parameteradresse SER2 (a0.l)

Byte

0 Baudrate
1 Parity, Anzahl Bits
2 Stop Bits
3 Extent Bit fuer Rx
4 Extent Bit fuer Tx

Gebräuchliche Werte für die SER2 sind:

```
dc.b $88, $13, $07, $90, $b0, $15        2400 Baud, 8 Bit, keine Parität, 1 Stop
dc.b $bb, $13, $07, $80, $a0, $15        9600 Baud, 8 Bit, keine Parität, 1 Stop
dc.b $cc, $13, $07, $80, $a0, $15        19200 Baud, 8 Bit, keine Parität, 1 Stop
dc.b $66, $13, $07, $80, $a0, $15        28800 Baud, 8 Bit, keine Parität, 1 Stop
dc.b $77, $13, $07, $80, $a0, $15        57600 Baud, 8 Bit, keine Parität, 1 Stop
dc.b $88, $13, $07, $80, $a0, $15        115000 Baud, 8 Bit, keine Parität, 1 Stop
```

Für die genaue Ermittlung der Byte-Werte siehe das SER2 Handbuch, oder das 88XR661 Datenblatt.

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
```

SER2 Kanal B mit 19200 Baud, 8 Datenbits, 1 Stoppbit und ohne Parität initialisieren.

```
START:
        MOVEQ   #$01,D0         * SER2, Kanal B
        LEA S2BD19K2(PC),A0     * Parameteradresse laden
        MOVEQ #!SIINIT,D7       * Werte an serielles Interface
        TRAP #1
        RTS

S2BD19K2:                       * SER2 19k2,N,8,1
        dc.b $cc, $13, $07, $80, $a0, $15
```

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
Änderungen zu 6.3:	NEIN
Siehe auch:	GETAD10(110 SETDA(111) SETDA12(134) GETAD12(135)

Beschreibung:

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.

Komplexes 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
Änderungen zu 6.3: NEIN

Siehe auch: GETAD8(109) SETDA(111) SETDA12(134) GETAD12(135)

Beschreibung:

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.

Achtung! Ggf. sind drei Wait-Zyklen beim 68008 einzuschalten, um die Startpulsbreite zu garantieren, sonst ergeben sich Messfehler.

Komplexes Beispiel:

Bis zum Tastendruck wird alle 1/10 Sekunde der aktuelle Wert am AD-Wandler 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
Änderungen zu 6.3:	NEIN
Siehe auch:	GETAD8(109) GETAD10(110) SETDA12(134) GETAD12(135)

Beschreibung:

Damit kann man die Baugruppe DA8 x 2 ansprechen, die zwei Digital/Analog-Umsetzer beinhaltet. 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 8Bit-Größe verarbeiten. Die Wandlung erfolgt in ca. 800 ns (Nanosekunden).

Komplexes Beispiel:

Wenn man den Kanal 0 auf die X-Ablenkung eines Skops legt und den Kanal 1 auf die Y-Ablenkung, so 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
```

Komplexes Beispiel:

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:
        MOVE    #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 bei der Erzeugung des Bildes durch den GDP (Graphik Display Prozessor) passiert. Ein Beispiel für das Rasterscan-Verfahren auf dem Skop sei hier mal als Anregung gezeigt.

Wenn man das Programm startet, so 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
Änderungen zu 6.3: NEIN

Siehe auch: SPEAK1(113) SOUND(114)

Beschreibung:

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 angegeben werden. Das Register A0.L erhält dazu die Adresse des Parameterblocks. Das erste Wort des Parameterblocks enthält die Anzahl der Phoneme. Dann folgend 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 kann 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
Änderungen zu 6.3: NEIN

Siehe auch: SPEAK(112) SOUND(114)

Beschreibung:

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 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 folgen 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 Parameterblocks
Ausgaberegister: KEINE
Zerstörte Register: KEINE

Ab Version: 4.0
Änderungen zu 4.3: NEIN
Änderungen zu 6.1: NEIN
Änderungen zu 6.3: NEIN

Siehe auch: SPEAK(112) SPEAK1(113)

Beschreibung:

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 erscheint ein rhythmisches Klanggebilde am Ausgang des Lautsprechers.

```
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
Änderungen zu 6.3: UHR3 Unterstützung hinzugefügt

Siehe auch: SETUHR(116) UHRPRINT(140)

Beschreibung:

Mit dieser Funktion können die Uhrenbaugruppe mit dem IC E050-16,die SMART-WATCH, sowie die Dallas DS12887 angesprochen werden. Dabei wird die Uhrzeit sowie das Datum gelesen.

Es werden dabei 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, so dass 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.

Dabei kann man das Programm natürlich auch ergänzen, um Datum, Jahr und Wochentag auszugeben.

```
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   * Adressse Ausgabetext
        MOVEQ   #$33,D0         * Schriftgröße
        MOVEQ   #10,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 Byte verlangt, da auch 100-tel Sekunden gesetzt werden können.
Änderungen zu 6.1: NEIN
Änderungen zu 6.3: UHR3 Unterstützung hinzugefügt

Siehe auch: GETUHR(115) UHRPRINT(140)

Beschreibung:

Auch mit diesem Befehl wird entweder die Uhr auf der Uhrenbaugruppe, die SMART-WATCH oder die Dallas DS12887 angesprochen.

Mit diesem Befehl wird die Uhrzeit und das Datum eingestellt. Bei der Uhrenbaugruppe werden die 1/10 und 1/100 Sekunden ignoriert.

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
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
Änderungen zu 6.3:	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)	

Beschreibung:

Damit kann man prüfen, ob der Drucker das letzte Zeichen schon ausgegeben hat und bereit für den Empfang eines neuen Zeichens ist. Mit diesem Befehl ist es möglich den Drucker im Hintergrundbetrieb laufen zu lassen, und in einem Hauptprogramm etwas anderes zu tun. Wenn der Drucker bereit ist, so wird im Register D0.L der Wert $FF übergeben, sonst der Wert 0.

Ist die Routine auf die serielle Karte gelenkt, so wird entweder der Wert $FFFFFFFF oder 0 zurückgegeben.

Komplexes Beispiel:

Während der Kreis gezeichnet wird, wird auch die Druckausgabe durchgeführt. Wenn man den Drucker anhält, so wird das Kreisprogramm dennoch nicht unterbrochen. Da kein LF (Code: $A) ausgeben wird, erfolgt die Ausgabe immer auf einer 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           * Letztes 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
Änderungen zu 6.3: NEIN

Siehe auch: RI(14) PO(15) RELAUS(119)

Beschreibung:

Damit kann man das Relais (falls vorhanden) auf der CAS2-Baugruppe einschalten. Diese 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
Änderungen zu 6.3: NEIN

Siehe auch: RI(14) PO(15) RELAN(118)

Beschreibung:

Damit kann man das Relais (falls vorhanden) auf der CAS2-Baugruppe ausschalten. 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
Änderungen zu 6.3: NEIN

Siehe auch: SETERR(53) GETERR(54) ASSEMBLE(65)
GETORG(68) PUTORG(69)

Beschreibung:

Die Anzahl der Fehler, die bei einer Übersetzung durch den Assembler auftraten werden im Register D0.L übergeben. Die Maximalzahl ist dabei $FFFF. 0 bedeutet, es trat kein Fehler auf. Dieser Befehl wird von höheren Programmiersprachen verwendet, um festzustellen ob Fehler bei einem Assemblerlauf aufgetreten sind und dann das Programm nicht zu starten.

Komplexes 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     * Alter Textstart wieder
        TRAP    #1
        RTS
        DS.B    1000            * Platz für Macro-Tabelle

BUFFER:
        DC.B    'ORG ?+$2000',0 * Adresse muss eingestellt werden
```

Wenn man das Programm startet, so meldet sich der Texteditor mit einem leeren Feld. Man kann nun ein Assemblerprogramm eingeben. Wenn man CTRL-K und dann X betätigt, so wird automatisch der Assembler gestartet, wenn man einen Fehler im Assemblerprogramm hat, so wird automatisch wieder der Texteditor aufgerufen und man kann den Fehler beseitigen. Es meldet sich erst dann wieder das Menü, wenn man einen korrekten Assemblertext eingibt. 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 natürlich nicht mehr nötig, da es im Editor den Befehl ^KA gibt. Es soll hier auch nur als Beispiel dienen.

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 bei 68020
Änderungen zu 4.3:	----
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN
Siehe auch:	GETFLOAT(122) PRTFP0(145) FPUWERT(146) SETFPX(147) SETFPY(148) SETFPZ(149)

Beschreibung:

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 der FPU stehen, in eine ASCII-Darstellung gebracht werden. Dazu muss im Register FP0.X der Wert vorliegen und Register A0.L muss auf einen Buffer zeigen, der genügen Stellen enthält, damit alle Zeichen abgelegt werden können. Nach dem Aufruf zeigt A0.L dann auf das Ende, 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, kann 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 FPSS 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.

```
VZ.ZZZZZZZZZZZZZZZZeVEEE
||||                ||
||||                |- Drei Stellen für den Exponenten
||||                -- Vorzeichen des Exponenten, wenn negativ
|||------------------- 16 Nachkommastellen
||-------------------- Dezimalpunkt
|--------------------- Eine Vorkommastelle
---------------------- Vorzeichen, falls negativ
```

Es wird im Gegensatz zum Befehl PRTFP0 nicht geprüft, ob im Register FP0 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.P #-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
```

Komplexes 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 bei 68020
Änderungen zu 4.3:	----
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN
Siehe auch:	PRTFP0(121) PRTFP0(145) FPUWERT(146) SETFPX(147) SETFPY(148) SETFPZ(149)

Beschreibung:

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 und 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 auftrat, so zeigt A0.L direkt auf den Fehler. Das CARRY-Flag ist gesetzt, wenn ein Fehler auftrat.

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, so dass bei größeren Zahlen der Aufruf dieses Programms sinnlos ist, da das Ergebnis völlig falsch ist.

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-Darstelllung wird in das Register FP0 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 (Ist nicht unbedingt nötig)

ZIEL:                           * Dort steht nachher FP-Zahl
        DS.P    1
```

Komplexes 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, so 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 nur <RETURN> gedrückt
        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 ZIEL(PC),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 aufffüllen, 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 Vorne
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
Zerstörte Register:	KEINE
Ab Version:	6.0
Änderungen zu 4.3:	----
Änderungen zu 6.1:	Wenn CR gedrückt wird, so 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.
Änderungen zu 6.3:	NEIN
Siehe auch:	READ(11)

Beschreibung:

Dieser Befehl arbeitet ähnlich dem READ-Befehl, deshalb werden hier nur die Unterschiede aufgeführt.

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 steht, ein Text abgelegt, der mit einer Null enden muss. Wird dort nur eine Null abgelegt, so ist der Befehl READAUS mit dem Befehl READ fast identisch. Ein weiterer Unterschied ist nämlich 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 und hinter diesem Text steht der Cursor. Mit diesem kann wie bei READ der Text bearbeitet werden. Wird <RETURN> gedrückt, so wird der alte Text ausgegeben und hinter dem Text sind dann wieder 11 Zeichen Platz. Der Cursor steht dann auch wieder hinter dem Text. Ist das Textfenster leer und es wird <RETURN> gedrückt, so 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
        TST     D4
        BNE.S   START           * Weiter, wenn Buffer nicht leer ist
        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 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
Änderungen zu 6.3:	Verwendung der Funktion -22 geändert

Siehe auch:	GETRAM(59)	GETBASIS(89)	GETVAR(90)
	SETA5(91)	GETVERS(97)	GETSN(98)
	SUCHBIBO(136)	SYSTEM(139)	SETSYS(160)
	GETSYS(161)	PATCH(162)	

Beschreibung:

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 immer ins Grundprogramm gesprungen werden muss.

Ist D0.W positiv, so wird das Grundprogramm immer als Ganzes aufgerufen, d.h. es erscheint das Grundprogrammenü.

Wenn D0.W negativ ist, so werden die einzelnen Menüpunkte aufgerufen. Dabei sind die Programme in der gleichen Reihenfolge angeordnet wie beim einseitigen Menü. Wenn eine Bildschirmausgabe erfolgt, so wird vorher der Bildschirm gelöscht. Einige dieser Programme, wie Editor oder Assembler können auch direkt aufgerufen werden. Dazu stehen eigenen 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 dies 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	Speichern Disk
-9	Laden Disk
-10	Inhalt Disk
-11	Löschen Datei
-12	Kopieren Datei
-13	Umbenennen Datei
-14	Booten
-15	Eprom programmieren
-16	Eprom lesen
-17	Speicherbereiche ausgeben
-18	Druckmenü
-19	IO lesen
-20	IO setzen
-21	Einzelschritt aufrufen
-22	System Konfiguration einstellen
-23	Alten Textstart festlegen
-24	Neuen Textstart festlegen
-25	Symboltabelle löschen
-26	CO auf 80 Zeichen einstellen
-27	CO auf 40 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:

Grundprogrammmenü aufrufen.

```
START:
        MOVEQ   #0,D0           * Ganzes Menü
        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.b = Auswahl des Programms und Spezialfunktion Zusätzlich noch verschiedene Register, die vom Unterprogramm abhängen
Ausgaberegister: Ausgaberegister sind vom Unterprogramm abhängig
Zerstörte Register: d7

Ab Version: 6.0
Änderungen zu 4.3: ----
Änderungen zu 6.1: NEIN
Änderungen zu 6.3: NEIN

Siehe auch:

MOVETO(8)	DRAWTO(9)	CLR(16)
CLPG(17)	WAIT(18)	CMD(26)
NEWPAGE(27)	SETFLIP(34)	SETPEN(37)
ERAPEN(38)	CMDPRIN(40)	AUTOFLIP(60)
SETXOR(77)	GETXOR(78)	SETCOLOR(79)
GETCOLOR(80)	GETXY(103)	GRAFIK(126)
GDPVERS(127)		

Beschreibung:

Dies ist kein einzelnes Unterprogramm, sondern es ist eine Programmsammlung. Dabei stehen 16 Unterprogramme zur Verfügung, die zum Betrieb mit der HARDCOPY-MAUS-Baugruppe benötigt werden.

Da die Unterprogramme verschiedene Ein- und Ausgaberegister verlangen, sind sie aufgeteilt, wobei die Angaben wieder im Kopf aufgeführt sind. Die Auswahl des Programms erfolgt dabei durch das Register D0.B. Dabei werden nur die untersten 4 Bit für das Programm berücksichtig. Die anderen Bits haben teilweise noch verschiedene Funktionen. Im folgenden ist die Unterprogrammnummer identisch mit der Zahl in D0.B.

Unterprogrammnummer: 0
Programmfunktion: Mausposition relativ abfragen

Eingaberegister: KEINE
Ausgaberegister: d0.b = 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.B 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.

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
        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.b = 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 dann die Bewegung der Maus seit der letzten Abfrage addiert. Auch wird in D0.B der Wert der Maustasten übergeben.

Durch diese Funktion kann (wenn D1 und D2 nicht zerstört werden) immer die aktuelle Mausposition erfahren werden, wenn das Programm dauernd aufgerufen wird.

Komplexes Beispiel:

Es wird die Mausbewegung seit der letzten Abfrage auf D1.W und D2.W addiert und in D0.B wird der Wert der Maustasten übergeben. Die aktuelle Mausposition wird auf dem Bildschirm ausgegeben. Durch Maustastendruck wird das Programm abgebrochen.

```
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 rücksetzen

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, so zählen die Register der Maus wieder von Null an. Dieser Befehl sollte am Anfang eines Programms benutzt werden, damit die Register der Maus definiert sind.

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.

Komplexes 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 (16,67 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.

Komplexes 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.b = 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, so 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 dann die Mausbewegung seit der letzten Abfrage und setzt das Fadenkreuz dann auf diese neue Position. Dabei wird allerdings keine Randüberwachung durchgeführt. Die neue Position wird in D1.W und D2.W zurückgegeben und in D0.B sind die Werte der Maustasten vorhanden.

Einfaches Beispiel:

Das Fadenkreuz kann mit der Maus bewegt werden. Das Programm endet, wenn eine 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, so kann hiermit der Wert des Ports gelesen werden. Da der Port nur 8 Bit groß ist, so 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, so wird diese Funktion abgeschaltet, damit es nicht zu Schwierigkeiten kommt. Das Programm könnte ja die Funktion CTRL-@ für irgendetwas benutzen.

Mit dieser Funktion kann aber 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, so 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 ==Y Hardcopy aus
        MOVEQ   #!HARDCOPY,D7
        TRAP    #1
        RTS
```

Unterprogrammnummer: 8
Programmfunktion: Bildschirmdaten im Speicher abgelegen

Eingaberegister: d0.b = Bits 4-7 haben Spezialfunktion
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 GDP bzw. 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 dabei, 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 Bild übereinander gelegt werden.
Bit 6 0 = Das Bild wird direkt übernommen.
1 = Das Bild wird invers im Speicher abgelegt.
Bit 5 0 = Es erfolgt keine Veränderung.
1 = Das Bild wird an der spiegelverkehrt in Y-Richtung im Speicher abgelegt.
Bit 4 0 = Es erfolgt keine Veränderung.
1 = Das Bild wird spiegelverkehrt in X-Richtung im Speicher abgelegt.

Die Funktionen können beliebig kombiniert werden.

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 wird (Bits 4-7 auf Null), so wird die gleiche Funktion des GRAFIK-Pakets aufgerufen, die wesentlich schneller arbeitet. Es ist dann aber kein Fadenkreuz zu sehen; die Funktion wird aber trotzdem ausgeführt.

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
```

Komplexes Beispiel:

Ein ausgefülltes Quadrat wird gezeichnet und abgespeichert. Danach wird das Quadrat noch einmal dazu abgespeichert. Der Bildschirm wird gelöscht und das Bild aus beiden Quadraten wird 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
        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 Screen bringen

Eingaberegister: d0.b = Bits 4-7 haben Spezialfunktion
a0.l = Gerade Adresse der Bildschirmdaten
Ausgaberegister: KEINE

Dies 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 gleiche Funktion, nur Bit 7 weicht etwas ab. Ist dieses Bit nicht gesetzt, so werde die Daten normal auf den Bildschirm gebracht (so, wie der GDP eingestellt ist). Ist das Bit aber auf 1 gesetzt, so wird der XOR-Modus eingeschaltet, und die Bildschirmfarbe wechselt dort, wo ein Punkt gesetzt wird (siehe Handbuch GDPHS).

Einfaches Beispiel:

Der Bildschirm 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
```

Komplexes Beispiel:

Ein Kreis wird gezeichnet. Das Bild wird abgespeichert. Dann wird es gespiegelt abgespeichert und mit dem XOR-Modus geladen. Dann wird es noch einmal gespiegelt geladen.

```
START:
        MOVEQ   #8,D0           * Kreis ausgefüllt
        MOVEQ   #100,D1         * X-Position
        MOVEQ   #100,D2         * Y-Position
        MOVEQ   #100,D3         * Radius des Kreises
        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
        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.b = Bits 4-7 haben Spezialfunktion
a0.l = Gerade Adresse für Hilfsspeicher
a1.l = Adresse der Druckerbefehle
Ausgaberegister: KEINE

Um auch direkt Bilder vom Bildschirm auf den Drucker ausgeben zu können, ist dieses Programm gedacht.

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. Nachdem nun 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 $FF.B als Endekennung stehen. Wenn am Anfang einer Zeile diese Befehle an den Drucker geschickt worden sind, werden die Bildpunkte übertragen. Dabei werden immer 8 Punkte in einem Byte gleichzeitig übertragen. Diese Bildpunkte stehen untereinander. So werden also 512 Bytes 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 :

```
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
```

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
```

Unterprogrammnummer: 11
Programmfunktion: Hardcopy auf 24-Nadel Drucker

Eingaberegister: d0.b = Bits 4-7 haben Spezialfunktion
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
```

Unterprogrammnummer: 12
Programmfunktion: Standarthardcopy auf 9-Nadel Drucker

Eingaberegister: d0.b = Bits 4-7 haben Spezialfunktion
a0.l = Gerade Adresse für Hilfsspeicher
Ausgaberegister: KEINE

Dieser Befehl entspricht dem Programm 10, wobei allerdings keine Druckerbefehle angegeben werden, da fest im Grundprogramm vorhandene verwendet werden. Die oben aufgeführte Tabelle für den 9-Nadel Betrieb ist identisch mit der im Grundprogramm. Die Befehle sind für Epson-kompatible Drucker geeignet.

Unterprogrammnummer: 13
Programmfunktion: Standarthardcopy auf 24-Nadel Drucker

Eingaberegister: d0.b = Bits 4-7 haben Spezialfunktion
a0.l = Gerade Adresse für Hilfsspeicher
Ausgaberegister: KEINE

Dieser Befehl entspricht dem Programm 11, wobei allerdings keine Druckerbefehle angegeben werden, da fest im Grundprogramm vorhandene verwendet werden. Die 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 Hilfsspeicher
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 Spezialfunktion. 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 Hilfsspeicher
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 Spezialfunktion. 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.b = Auswahl des Programms Zusätzlich noch verschiedene Register, die vom 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.
Änderungen zu 6.3:	NEIN

Siehe auch:

MOVETO(8)	DRAWTO(9)	CLR(16)
CLPG(17)	WAIT(18)	CMD(26)
NEWPAGE(27)	SETFLIP(34)	SETPEN(37)
ERAPEN(38)	CMDPRIN(40)	AUTOFLIP(60)
SETXOR(77)	GETXOR(78)	SETCOLOR(79)
GETCOLOR(80)	GETXY(103)	HARDCOPY(125)
GDPVERS(127)		

Beschreibung:

Auch dieser Aufruf beinhaltet wie HARDCOPY eine ganze Menge von Funktionen. Es sind alles Programme, die für Grafik- und Textausgabe gedacht sind. Die Besonderheit bei diesen Programmen ist, dass die Ausgabe für die GDP-Karte und die COL256 geschrieben sind. Durch einen der unten beschriebenen Befehle kann ausgewählt werden, ob auf die GDP oder die COL ausgegeben werden soll. Normalerweise ist die Ausgabe auf die GDP gelenkt.

Wichtig ist für die Ausgabe auf der COL, dass der Ausgang benutzt wird, der eine Auflösung von 512*256 Punkten ermöglicht, da die Routinen dafür ausgelegt sind. Dadurch sind zwar "nur" 16 Farben darstellbar, allerdings ist die Auflösung genau so wie bei der GDP, wodurch die Funktionen alle nahezu kompatibel sind.

Auch hier muss in D0.B entschieden werden, welches Unterprogramm aufgerufen wird. Es gibt 32 Unterprogramm 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 sind. 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 funktionieren. Dies 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 könnten.

Alle Zeichen-Funktionen, bei den 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, verwendet werden. 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 jedes Mal 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, so müssen D1.W und D2.W im Bereich von 0 bis 511 liegen.

Einfaches Bespiel:

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 viele Linien hintereinander gezeichnet werden. Dabei ist aber der Endpunkt der erste Linie der Anfangspunkt der nächsten Linie usw. Die Koordinate werden dazu in einer Tabelle übergeben. A0.L muss auf diese Tabelle zeigen. Nach dem Aufruf zeigt A0.L genau hinter die Tabelle, wodurch sehr 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 jetzt noch die Kantenlänge angegeben werden. Dieser Wert sollte aber 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 Eck
d3.w = Positive Kantenlänge
Ausgaberegister: KEINE

Dieses 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 muss jetzt noch die Kantenlänge angegeben werden. Dieser Wert sollte aber 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.

```
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

Dieses Programm ähnelt sehr stark 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

Damit auch Kreise sehr schnell gezeichnet werden können, ist dieses Programm vorhanden. Dabei 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 umrandet 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
```

Komplexes 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: 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.

Unterprogrammnummer: 11
Programmfunktion: Fläche füllen

Eingaberegister: d1.w = X-Koordinate des Anfangspunktes
d2.w = Y-Koordinate des Anfangspunktes
Ausgaberegister: Carry = Gesetzt, wenn Fehler auftrat

Um geschlossene Flächen füllen zu können, ist dieser Befehl nötig. 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 gefüllt 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 selbe Farbe hat, wie die aktuelle Malfarbe, so wird der Befehl nicht ausgeführt und das CARRY-Flag wird gesetzt.

Komplexes Beispiel:

Nach dem Löschen des Bildschirms werden viele kleine Kreise gezeichnet. Dann wird vom Mittelpunkt aus die verfügbare Fläche gefüllt. Dies 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 und 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 wurde bzw. den Wert 0 in D1.L und D2.L falls vorher noch keine neue Seite eingestellt wurde, denn diese Werte sind 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 Befehle wie ERAPEN und SETPEN aus, sondern es können 16 verschiedene Farben 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 (1,3,5,7,...). Dadurch kann auch dieser Aufruf für GDP und COL gleich sein.

In D2.B kann 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 der Tabelle entnommen werden kann. 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

Diese Funktion fragt die mit Befehl 15 eingestellten Werte ab. Genau diese werden zurückgeliefert. In D1.L steht 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 auftrat

Dieser Befehl funktioniert mit der COL und mit der neuen GDPHS. Wenn die alte GDP-Karte vorhanden ist, so wird das CARRY-Flag gesetzt und das Programm wird 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 wurde, auf der anderen Seite. Der Bildschirm ist somit 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, so erscheinen natürlich auch alle späteren Ausgabe versetzt, so dass die Koordinaten, wenn sie normal erscheinen sollen, vor dem Aufruf umgerechnet werden müssen. Die Routine CO macht dies, 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 wird
        TRAP    #1
        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 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

Um einen Punkt auslesen zu könne, ist dieses Programm gedacht. 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, so wird das CARYY-Flag gesetzt und das Programm beendet. Ansonsten ist das CARRY-Flag rückgesetzt und in D3.B wird die Farbe zurückgeliefert.

Bei der GDP wird die Farbe zurückgeliefert (0 = Schwarz/1 = Weiß).

Bei der COL hingeben wird der Farbwert zurückgegeben, der der Tabelle der Funktion 15 entspricht. Dieser Wert muss dann gegebenenfalls in die Nummer der Farbe umgerechnet werden.

Durch dieses 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-Koodinate
        MOVEQ   #100,D2         * Y-Koodinate
        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 auftrat

Dieses Programm funktioniert genau so, wie die HARDCOPY-Funktion beim Programm HARDCOPY, wenn die GDP als Ausgabe gewählt wurde. Allerdings wird sie nicht mit der HARDCOPY-MAUS-Baugrupe, sondern mit der GDPHS durchgeführt, wodurch sie sehr viel schneller ist. Außerdem gibt es keine Extra-Funktionien. 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 Wert 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

Dies 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.

Komplexes 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

Dieses 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 identisch ist aber, 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, so ist diese Funktion aktiviert. Dabei muss dann 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.

Komplexes 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, so 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
```

Komplexes Beispiel:

Ein blauer Kreis wird mit einem zyan farbene 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                     * Kreis ist jetzt in blau auf der COL zu sehen

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 dabei 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.

Einfaches Beispiel:

Der Bildschirm wird gelöscht und 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     * Texteadresse
        MOVEQ   #10,D1          * X-Koordinate
        MOVE    #400,D2         * Y-Koodinate
        MOVEQ   #$33,D3         * Textgröße
        MOVEQ   #23,D0          * Kennung
        MOVEQ   #!GRAFIK,D7     * Text ausgeben
        TRAP    #1
        RTS

TEXT:                           * Ausgabetext
        DC.B    'Dies ist ein Test-Test.',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

Dieser Befehl entspricht in seiner Funktion in etwa dem Programm PROGZGE. Auch hier kann ein frei definierbares Zeichen ausgegeben werden. In A0.L steht dabei 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 direkt hintereinander durchgeführt werden, wodurch sehr leicht Texte ausgegeben werden könne. Es muss dann nur die Adresse des auszugebenden Zeichens geändert werden.

Komplexes Beispiel:

Es wird der Text "aha" ausgegeben.

Wenn längere Texte ausgegeben werden sollen, so wird natürlich in der Praxis eine Schleife verwendet, in der aus einer Tabelle alle Adressen geholt werden, so dass 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(A1),A1      * Adresse des "h"
        MOVEQ   #10,D1          * X-Anfangskoordinate
        MOVEQ   #250,D2         * Y-Koordinate
        MOVEQ   #$55,D3         * Schriftgröße
        MOVEQ   #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   A0,A1           * 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 nicht jedes Programm eine eigene Zeichentabelle anlegen muss, gibt es dieses Programm. In A0.L wird die Adresse der Tabelle zurückgeliefert, die alle Standart 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 sind. 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 auf allen Adressen betrieben werden kann, die als unteres Wort $C000 (Bei 68000 und 68020 entsprechend multipliziert) haben, und verschiedene Programme jeweils andere Adressen besitzen, kann mit diesem Befehl die Adresse geändert werden. Sie muss in D1.L übergeben werden. Intern wird das untere Wort auf $C000 gesetzt. Die Adresse darf beim 68000 und 68020 nicht angepasst werden, sondern muss der Wert sein, der auch auf der COL eingestellt ist. Die Anpassung erfolgt intern. Voreingestellt ist der Wert $EC000. Ist die GRAFIK-Funktion auf die GDP-Karte geschaltet, so wird der Befehl ignoriert.

Einfaches Beispiel:

Auf alte COL-Standart-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 GDP und COL ist diese Funktion gedacht. In D1.B wird ausgewählt, welche Karte angesprochen werden soll. Eine 0 schaltet die GDP-Karte an und eine 1 die COL. Dabei werden bei der jeweiligen Karte alle Parameter wie Seiten, Farbe und Scrollwert zur Sicherheit 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

Um jederzeit feststellen zu können, wo die Grafik-Funktionen ausgegeben werden, ist dieser Befehl gedacht. In D1.L wird dabei der Wert zurückgegeben, der mit Funktion 30 eingestellt wurde (0 = GDP, 1 = COL). 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 Flags
Zerstörte Register:	KEINE
Ab Version:	6.0
Änderungen zu 4.3:	----
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN

Siehe auch:	MOVETO(8)	DRAWTO(9)	CLR(16)
	CLPG(17)	WAIT(18)	CMD(26)
	NEWPAGE(27)	SETFLIP(34)	SETPEN(37)
	ERAPEN(38)	CMDPRIN(40)	AUTOFLIP(60)
	SETXOR(77)	GETXOR(78)	SETCOLOR(79)
	GETCOLOR(80)	GETXY(103)	HARDCOPY(125)
	GRAFIK(126)		

Beschreibung:

Durch diese Routine kann man erfahren, welche GDP im System vorhanden ist. Dabei gibt es die Auswahl zwischen GDP und GDPHS. Es wird die Information zurückgegeben, die auch mit den DIL-Schaltern auf der KEY eingestellt wurden. 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
```

Komplexes 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/2 lenken.		
Eingaberegister:	D0.B = Wert für Umlenkung der einzelnen Programme		
Ausgaberegister:	Carry = Gesetzt, wenn keine SER/2 vorhanden.		
Zerstörte Register:	KEINE		
Ab Version:	6.0		
Änderungen zu 4.3:	----		
Änderungen zu 6.1:	NEIN		
Änderungen zu 6.3:	Unterstützung für die SER2 integriert.		
Siehe auch:	SI(104)	SO(105)	SISTS(106)
	SOSTS(107)	SIINIT(108)	SI2(138)
	SETSER(150)	GETSER(151)	

Beschreibung:

Mit diesem Programm können verschiedene andere Unterprogramme auf eine der seriellen Karten 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 setzt. Ist keine serielle Karte vorhanden, 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

Einfaches Beispiel:

CI und CSTS werden 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
Änderungen zu 6.3:	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)	

Beschreibung:

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 (SER) 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
Änderungen zu 6.3: NEIN

Siehe auch: CLUT(131)

Beschreibung:

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
Änderungen zu 6.3: NEIN

Siehe auch: CLUTINIT(130)

Beschreibung:

Mit diesem Programm kann man die 255 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. Dadurch können mehrere Tabellen direkt hintereinander stehen und das Programm kann in einer Schleife mehrfach hintereinander 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. Die drei Bytes haben einen Bereich von 0 bis 63. Das Ende der Tabelle wird mit $FF markiert.

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
Änderungen zu 6.3:	NEIN
Siehe auch:	RELAISIN(133)

Beschreibung:

Hiermit können die Relais der RELAIS-Baugruppe gesetzt bzw. rückgesetzt werden. Dabei muss in A0.L die Adresse der Baugruppe angegeben werden. Dabei darf die Adresse nicht mit dem CPU-Wert multipliziert werden, da das Programm dies selber durchführt. Eine Karte mit einer bestimmten Adresse hat also beim 68008 die gleiche Adresse wie beim 68000 oder 68020. In d0.b muss jetzt noch angegeben werden welche Relais gesetzt werden sollen und welche rückgesetzt 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 werden 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 rücksetzen
        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
Änderungen zu 6.3:	NEIN
Siehe auch:	RELAIS(132)

Beschreibung:

Auf jeder RELAIS-Baugruppe gibt 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
Änderungen zu 6.3:	NEIN
Siehe auch:	GETAD8(109) GETAD10(110) SETDA(111) GETAD12(135)

Beschreibung:

Die ADDA-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 aber nur 12 Bit aus.

Für die richtige Ausgangsspannung ist aber 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
Änderungen zu 6.3: NEIN

Siehe auch: GETAD8(109) GETAD10(110) SETDA(111) SETDA12(134)

Beschreibung:

Dieses Programm bedient den AD-Teil der ADDA-Baugruppe. Dort stehen 16 Kanäle zur Verfügung, die einzeln angesteuert werden können. In D1.B muss dabei 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. Dies ist die normale Betriebsart. Wenn die CPU aber keine WAIT-Leitung besitzt, so kann mit einer 1 in diesem Bit, 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 Null als Langwort als Endekennung.
Änderungen zu 6.3:	NEIN

Siehe auch:	GETRAM(59)	GETBASIS(89)	GETVAR(90)
	SETA5(91)	GETVERS(97)	GETSN(98)
	GRUND(124)	SYSTEM(139)	SETSYS(160)
	GETSYS(161)	PATCH(162)	

Beschreibung:

In diesem Programm sind drei Fälle enthalten, 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, so wird das CARRY-Flag gesetzt, ansonsten ist es rückgesetzt. 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 Eintrag gefunden wird)
D3.L = Zweiter Teil des Namen
D4.L = Startadresse des Programms (Bei relokativen Programmen bereits berechnet)
D5.L = Länge des Programms in Bytes
D6.B = Relokativ Byte
D7.B = CPU-Wert (0, dann für alle CPUs geeignet, sonst Wert der System-CPU)

D0.W = 1

Die Funktion ist fast die gleich 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. Dabei wird von der Adresse $400 der ganze Speicher durchsucht. Die Tabelle wird ab der Adresse abgelegt, die in A1.L übergeben wird. A1.L muss unbedingt eine 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.

TRAP-Nummer:	137
Befehlsname:	DISASS
Befehlsgruppe:	DIS-Assembler
Kurzbeschreibung:	Disassembliert einen Befehl
Eingaberegister:	d0.l = Adresse des Befehle a0.l = Gerade Zieladresse für Text
Ausgaberegister:	KEINE
Zerstörte Register:	KEINE
Ab Version:	6.1
Änderungen zu 4.3:	----
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN
Siehe auch:	----

Beschreibung:

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. Dies 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 Wert der Adresse übergeben. Für den Zielbuffer sollten beim 68008 und 68000 etwa 60 Zeichen und beim 68020 ca. 100 Zeichen reserviert werden. Der Ausgabetexte 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
Änderungen zu 6.3:	SER2 Unterstützung hinzugefügt.

Siehe auch:	SI(104)	SO(105)	SISTS(106)
	SOSTS(107)	SIINIT(108)	SER(128)

Beschreibung:

Dieser 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 ja keine Daten geschickt werden, und so auch eine Abfrage, ob Daten vorhanden sind, nicht sinnvoll ist. Man muss sich also bei der jeweiligen Anwendung überlegen, welche Routine man verwenden will. In D0.L wird das gelesene Zeichen zurückgeliefert, wobei natürlich nur maximal die untersten 8 Bit interessant sind. Es wird immer so lange gewartet, bis wirklich ein Byte angekommen ist. Wird dieser Befehl bei aktiver SER2 aufgerufen, wird auf den Befehl SI umgeleitet, da die SER2 kein Handshake unterstützt.

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.
Änderungen zu 6.3:	Neue Baugruppen aufgenommen.

Siehe auch: GETRAM(59) GETBASIS(89) GETVAR(90) SETA5(91) GETVERS(97) GETSN(98) GRUND(124) SUCHBIBO(136) SETSYS(160) GETSYS(161) PATCH(162)

Beschreibung:

Nach dem Aufruf dieses Programms enthält D0.L einige Informationen, die für gewisse Anwendungen recht interessant sein können. Dabei sind die Informationen bitweise kodiert. Hier nun die Funktion der einzelnen Bits.

Bits	Funktion
0	1 = CPU 68008 ist im System vorhanden
1	1 = CPU 68000 oder 68010
2	1 = CPU 68020
3	1 = Die GDPHS ist vorhanden
4	0 = Über das GRAFIK-Programm wird die GDP angesprochen 1 = Über das GRAFIK-Programm wird die COL angesprochen
5-6	%00 = Keine Uhr vorhanden %01 = Uhrenbaugruppe vorhanden %10 = Smartwatch vorhanden %11 = Dallas Uhr
7	1 = Das Programm läuft im Einzelschritt
8	1 = Der Hardscroll bei CO ist eingeschaltet
9	1 = Das einseitige Menü ist die aktuelle Menüform
10	1 = Die Hardcopyfunktion bei CI ist auch außerhalb des Grundprogramms verfügbar
11-14	Gibt an welche Programme auf die serielle Karte gelenkt sind. Bit 11 : CI und CSTS Bit 12 : LO und LSTS Bit 13 : FLOPPY Bit 14 : Reserviert
15	1 = Es ist eine Harddisk vorhanden
16	1 = Key3 vorhanden
17	1 = SER vorhanden
18	1 = SER2 vorhanden
19	1 = GIDE vorhanden
20	1 = SRAMDISK vorhanden
21	1 = GDP-FPGA 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:	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:	----
Änderungen zu 6.3:	NEIN
Siehe auch:	GETUHR(115) SETUHR(116)

Beschreibung:

Dieses Programm wandelt das Datum und die Uhrzeit in eine lesbare Form. Dabei wird auch der Wochentag und das Datum ausgegeben. In D0.W wird angegeben, wie viele Zeichen der Wochentag maximal haben darf. Bei einer Null wird kein Wochentag mit ausgegeben. Bei einer Zahl wird so lange ausgegeben, bis die maximale Anzahl übertragen wurde, oder bis der Wochentag übertragen wurde. Hinter dem Wochentag wird dann 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:

```
Wochentag  Ta.Mo.Ja  St:Mi:Se
|         | |  |  | | |  |  |-- Sekunden (2 Stellen)
|         | |  |  | | |  |----- Minuten (2 Stellen)
|         | |  |  | | |-------- Stunden (2 Stellen
|         | |  |  | |---------- 2 Leerzeichen
|         | |  |  |------------ Jahr (2 Stellen)
|         | |  |--------------- Monat (2 Stellen)
|         | |------------------ Tag (2 Stellen)
|         |-------------------- 2 Leerzeichen (Auch wenn D0.W = 0)
|----------------------------- Wochentag (Anzahl der Stellen in D0.W)
```

Einfaches Beispiel:

Die Uhrzeit wird mit voll ausgeschriebenem Wochentage 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
        MOVE    #400,D1         * X-Position
        MOVEQ   #0,D2           * Y-Position
        MOVEQ   #!WRITE,D7      * Text ausgeben
        TRAP    #1
ENDE:
        RTS
ZIEL:
        DS.B    30
```

TRAP-Nummer:	141
Befehlsname:	HARDDISK
Befehlsgruppe:	Harddisk
Kurzbeschreibung:	Befehl an die 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 welche verlangt werden
Ausgaberegister:	d0.l = Fehlercode CARRY = 1, wenn d0.l <> 0
Zerstörte Register:	KEINE
Ab Version:	6.2
Änderungen zu 4.3:	----
Änderungen zu 6.1:	----
Änderungen zu 6.3:	Zugriffe können auf die IDE-Disk umgeleitet werden.
Siehe auch:	HARDTEST(142) SETS2I(152) GETS2I(153) IDETEST(154) IDEDISK(155)

Beschreibung:

ACHTUNG !!!

Zur direkten Benutzung einer SCSI-Harddisk ist nur der fortgeschrittene Programmierer in der Lage. 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.

Hiermit kann jede SCSI-Harddisk angesprochen werden. Es wird dabei 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 sehr viele verschiedene Befehle ausgeführt werden können, werden diese extra aufgeführt. Gemeinsam ist allen Befehlen, dass in D1.W der Befehl ausgewählt werden kann. Welche Nummern mit Funktionen 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 mit einer SCSI-Schnittstelle 8 Laufwerke verwaltet werden können. Dabei entspricht ein gesetztes Bit 0 dem Laufwerk 0, ein gesetztes Bit 1 dem Laufwerk 1 usw. Die Kodierung ist also genau so wie beim Diskettenlaufwerk.

Nach dem Aufruf steht in D0.L ein Fehlercode. Ist keine HARDDISK vorhanden, so 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 auch verschieden ist. In der Beschreibung über die SCSI-Schnittstelle ist aber eine Tabelle der Returncodes aufgeführt. Wenn kein Fehler auftrat, so ist aber D0.L immer Null und das CARRY-Flag ist zurückgesetzt. Das CARRY-Flag ist aber bei einem Wert ungleich -1 immer zurückgesetzt. In D0.L wird übrigens auch eine -1 zurückgegeben, wenn D1.W beim Aufruf einen falschen Wert enthalten hat (Falscher Befehl).

Wenn die scsi2ide Variable (siehe SETS2I) gesetzt ist werden alle Aufrufe auf die IDEDISK Routine umgeleitet.

Jetzt folgt die detaillierte Beschreibung der einzelnen Befehle. Register werden nicht zerstört und es werden außer in D0.L auch 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 aber 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 = Quelladresse

Dieser Befehl liest bis zu 256 hintereinander liegende Sektoren in den Speicher. Dabei wird in D2.L der Anfangssektor angegeben. Der kleinste Sektor ist Sektor 0 und 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 dabei für 256 Sektoren. Wenn von einem DOS Sektoren geladen werden sollen, die hintereinander liegen, so sollten diese immer in einem Stück eingelesen werden, da dies sehr viel schneller geht, als das mehrmalige Einlesen eines Sektors. In A0.L schließlich muss noch 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 = Zieladresse

Dieses Programm funktioniert genau so, wie die Funktion 1 READ. Allerdings werden die Sektoren geschrieben und nicht gelesen.

Komplexes 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           * 1  Sektor
        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 Laufwerk befinden sich nicht nur die Datenbytes, sondern auch noch Prüfbytes, die normalerweise vom Benutzer nicht benötigt werden. Sollen sie aber doch mit gelesen werden, so muss dieser 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

Dieser 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

Dieser Befehl veranlasst die HARDDISK gewisse Einstellungen zu ändern. Wie dies 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. Dies 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

Dieses Programm ist das Gegenstück zum Befehl 5. Er ist dafür gedacht, viele Einstellungen der HARDDISK zurückzulesen, um so z.B. festzustellen wie groß ein Sektor ist 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, so werden keine weiteren Daten übergeben, 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-Descriptors
        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. Dies passiert z.B. wenn das Laufwerk gerade eben erst eingeschaltet worden ist. Um abzufragen, ob das Laufwerk Daten übertragen kann, ist dieses Programm vorhanden.

Einfaches Beispiel:

Es wird so lange gewartet, bis das Laufwerk fertig 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: Laufwerk 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, so passiert nichts. Einige Laufwerke haben eine automatische Parkeinrichtung, so dass dann dieser Befehl nicht notwendig ist. Ein DOS sollte ihn aber auf jeden Fall zur Verfügung stellen.

Einfaches Beispiel:

Laufwerk 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. Dies ist das Gegenstück zu Funktion 10.

Einfaches Beispiel:

Laufwerk 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, ist diese Funktion vorhanden. Mit ihr können dann 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 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,D2       * 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 internen 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 gelesen bzw. geschrieben wurde. Bei einigen Festplatten kann dieses Buffer ausgelesen oder beschrieben werden. Dies ist normalerweise nicht notwenig, sondern nur zu Testzwecken gedacht. Die Daten innerhalb des Buffers 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, wie viel 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

Dieser Befehl schreibt die angegebene Anzahl von Bytes in den Buffer des Laufwerks. Dabei gilt das gleiche wie bei Funktion 13. Wenn mehr Daten übertrage werden sollen, als in den Buffer passen, so wird automatisch wenn der Buffer voll ist mit der Datenübertragung aufgehört.

Nummer des Befehls: 15
Befehlsname: Reserve (SB)
Kurzbeschreibung: Laufwerk reservieren

Eingaberegister: d2.w = 3rd PTY/3rd PTY Device ID/Extend/Reservatin ID
d3.w = Länge der Daten
a0.l = Adresse der zu übertragenden Daten

Dieser Befehl ist nur in wirklich sehr seltenen Fällen von Bedeutung. Beim NDR-Computer wird er wohl nie benutzt werden. Bei einer SCSI-Schnittstelle können nämlich auch mehrere Rechner auf eine Festplatte zugreifen. Dann kann ein Rechner ein Teil der Festplatte für sich reservieren, so dass keine anderer auf diesen Teil zugreifen kann.

Was eventuell wichtig sein könnte, ist dass auch ein Rechner bei gleichzeitig ablaufenden Programmen Reservierungen vornehmen kann. Dies ist für Multi-Tasking-System 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 noch die Länge der zu übertragenden Daten angegeben werden, die die Informationen über reservierte Bereiche enthalten und A0.L gibt die Quelleadresse 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. Dabei wird in D2.W der selbe Wert übergeben, wie bei Funktion 15 und 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, so 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, so 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.

Dieser 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.

Komplexes Beispiel:

Es werden die Sektoren 0 bis 100 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. Dies 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 wird mit dem Prüfvorgang angefangen und die in D3.W gewünschte Anzahl von Sektoren wird 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 anderen 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.

Komplexes 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

Damit werden verschiedene interne Zähler eines Laufwerks gelesen. Es werden dabei 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 werden, werden sie zurückgesetzt und 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               * Unkorregierte 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. Dies ist für jedes Betriebssystem notwendig, damit es weis, wie viele Sektoren zur Verfügung stehen. In D3.B wird dabei ausgewählt, ob die Sektoranzahl des gesamten Laufwerks ausgegeben werden soll, oder der letzte Sektor eines gewissen Tracks. Enthält D3.B den Wert 0, so muss D2.L auf Null gesetzt sein und es wird die Nummer des letzten Sektors des Laufwerks innerhalb der Daten zurückgeliefert. Enthält D3.B den Wert 1, so 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 ist es für ein Programm wichtig, mit welchem Laufwerk es arbeitet, damit es eventuelle Spezial-Befehle oder wichtige 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   #23,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

Jedes Laufwerk enthält intern ein Liste alle defekter Sektoren. Diese Liste ist unterteilt in Sektoren, die vom Hersteller eingegeben wurde, da sie bei der Herstellung entstanden und solchen die später vom Benutzer hinzugefügt wurden, da sie im Laufe der Zeit fehlerhaft wurden. D2.B gibt nun 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 jeder weiterführenden Beschreibung einer SCSI-Harddisk. In D3.W wird die Anzahl der maximal zurückgeschickten Daten angegeben. Ist die Liste länger, so 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 einige Sektoren Defekte aufweisen können, ist es möglich 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 auch 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 näher 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. Dazu ist dieser Befehl gedacht. Man muss dabei nicht wie bei einer Diskette jedes Byte einzeln übertragen, sondern nur gewisse 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 und die Bits 2 bis 0 geben das Format der List der Defekte an. Die Bedeutung dieser Bits kann jeder weiterführenden SCSI-Beschreibung entnommen werden. In D3.W wird der INTERLEAVE-Faktor angegeben und in A0.L die Quelladresse der List 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 eventuell einige Laufwerke zusätzliche Befehle ausführen können oder in 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, so muss in A0.L die Quell- oder Zieladresse angegeben werden.

Einfaches Beispiel:

Der Befehl TEST UNIT READY wird direkt 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               * Opearation Code
        DC.B    0               * LUN / Reserviert
        DC.B    0,0,0           * Reserviert
        DC.B    0               * Reserviert / Flag / Link
```

TRAP-Nummer:	142
Befehlsname:	HARDTEST
Befehlsgruppe:	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:	----
Änderungen zu 6.3:	SCSI nach IDE Umleitung integriert
Siehe auch:	HARDDISK(141) SETS2I(152) GETS2I(153)
	IDETEST(154) IDEDISK(155)

Beschreibung:

ACHTUNG !!!

Zur direkten Benutzung einer SCSI-Harddisk ist nur der fortgeschrittene Programmierer in der Lage. 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.

Dieses Programm ist dazu gedacht, dass ein Anwenderprogramm testen kann, ob eine bestimmte SCSI-Harddisk vorhanden ist, da ja mit den DIL-Schaltern auf der KEY nur festgelegt wird, dass mindestens eine verfügbar ist. Wird das angesprochene Laufwerk gefunden, so 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 dazu 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, so wird in D0.L der Fehlercode, den die Harddisk liefert, zurückgegeben, außerdem ist das CARRY-Flag zurückgesetzt, wenn der Wert in D0.L Null ist. Ansonsten ist das CARRY-Flag gesetzt und D0.L enthält den Wert -1.

Wenn die scsi2ide Variable (siehe SETS2I) gesetzt ist wird diese Routine auf IDETEST umgeleitet.

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 bei 68020
Änderungen zu 4.3:	----
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN
Siehe auch:	PRINTFP0(121) GETFLOAT(122) FPUWERT(146) SETFPX(147) SETFPY(148) SETFPZ(149)

Beschreibung:

Dieser Befehl ist fast identisch mit der Funktion 121 (PRINTFP0). Sie wurde nur eingeführt, weil sie etwas mehr kann und die Funktionen von PRTFP0 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, so dass die Zahl so kurz wie möglich wird. Außerdem wird erkannt, wenn es keine Zahl ist, die ausgegeben werden soll. In jedem besseren Handbuch zum 68881 (FPU) ist beschrieben, dass es einigen Formate der Zahlendarstellung gibt, die einen Überlauf darstellen oder den Wert KEINE ZAHL. Dies erkennt die Routine und gibt dann entsprechend der Kennung den Text "Keine Zahl" oder "Überlauf" aus. Dies kann man sehr leicht bei der Registeranzeige der FPU-Register beim Einzelschritt sehen, bevor das erste Mal auf ein Register zugegriffen wurde.

Die beiden folgenden Beispiele sind die gleichen, wie bei PRINTFP0, allerdings kann man bei einem Vergleich die verschiedene Ausgaben erkennen.

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.P #-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
```

Komplexes 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
        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 auftrat
Zerstörte Register:	KEINE
Ab Version:	6.1 bei 68020
Änderungen zu 4.3:	----
Änderungen zu 6.1:	Es werden auch Werte aus der Symboltabelle verarbeitet.
Änderungen zu 6.3:	NEIN
Siehe auch:	PRTFP0(121) GETFLOAT(122) PRTFP0(145) SETFPX(147) SETFPY(148) SETFPZ(149)

Beschreibung:

Dies ist nun 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. Weiterhin sind die natürlich Klammerausdrücke erlaubt. Die Schachtelungstiefe ist beliebig und 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, so 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, so 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 die 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
```

Komplexes 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.L 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 bei 68020
Änderungen zu 4.3:	----
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN
Siehe auch:	PRTFP0(121) GETFLOAT(122) PRTFP0(145) FPUWERT(146) SETFPY(148) SETFPZ(149)

Beschreibung:

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 bei 68020
Änderungen zu 4.3:	----
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN
Siehe auch:	PRTFP0(121) GETFLOAT(122) PRTFP0(145) FPUWERT(146) SETFPX(148) SETFPZ(149)

Beschreibung:

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 bei 68020
Änderungen zu 4.3:	----
Änderungen zu 6.1:	NEIN
Änderungen zu 6.3:	NEIN
Siehe auch:	PRTFP0(121) GETFLOAT(122) PRTFP0(145) FPUWERT(146) SETFPX(148) SETFPY(149)

Beschreibung:

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
```

TRAP-Nummer:	150
Befehlsname:	SETSER
Befehlsgruppe:	SER-Baugruppe
Kurzbeschreibung:	Auswahl des aktuellen SER Kanals
Eingaberegister:	d0.b = Nummer des SER-Kanals
Ausgaberegister:	d0
Zerstörte Register:	KEINE
Ab Version:	7.0
Änderungen zu 4.3:	----
Änderungen zu 6.1:	----
Änderungen zu 6.3:	----

Siehe auch:	SI(104)	SO(105)	SISTS(106)
	SOSTS(107)	SIINIT(108)	SER(128)
	SI2(138)	GETSER(151)	

Beschreibung:

Mit dieser Funktion wird der aktuelle serielle Kanal fürs senden oder empfangen eingestellt. Die Kanalnummer wird in d0.b übergeben.

Gültige Kanalnummern sind:

1 = "alte" SER-Karte mit 6551
2 = 1. Kanal der SER2-Karte
3 = 2. Kanal der SER2-Karte

Nach korrekter Ausführung des Befehls wird im Register d0 die Kanalnummer zurückgegeben. Im Fehlerfall (Kanal nicht vorhanden) steht in d0 $ffff.

TRAP-Nummer:	151		
Befehlsname:	GETSER		
Befehlsgruppe:	SER-Baugruppe		
Kurzbeschreibung:	Aktuellen SER Kanal feststellen.		
Eingaberegister:	KEINE		
Ausgaberegister:	d0.b = Nummer des aktiven SER-Kanals		
Zerstörte Register:	KEINE		
Ab Version:	7.0		
Änderungen zu 4.3:	----		
Änderungen zu 6.1:	----		
Änderungen zu 6.3:	----		
Siehe auch:	SI(104)	SO(105)	SISTS(106)
	SOSTS(107)	SIINIT(108)	SER(128)
	SI2(138)	SETSER(150)	

Beschreibung:

Diese Funktion gibt in d0.b die Nummer des aktiven seriellen Kanals zurück

Folgende Kanalnummern sind möglich:

0 = keine serielle Karte vorhanden.
1 = "alte" SER-Karte mit 6551
2 = 1. Kanal der SER2-Karte
3 = 2. Kanal der SER2-Karte

TRAP-Nummer:	152
Befehlsname:	SETS2I
Befehlsgruppe:	Harddisk
Kurzbeschreibung:	SCSI nach IDE Umleitung setzen / löschen.
Eingaberegister:	d0.b
Ausgaberegister:	d0
Zerstörte Register:	KEINE
Ab Version:	7.0
Änderungen zu 4.3:	----
Änderungen zu 6.1:	----
Änderungen zu 6.3:	----
Siehe auch:	HARDTEST(140) HARDDISK(141) GETS2I(153) IDETEST(154) IDEDISK(155)

Beschreibung:

Diese Funktion dient zum setzten oder löschen der SCSI nach IDE Umleitung.

Wird in d0.b eine 1 übergeben, wird die Umleitung aktiviert. Das bedeutet, dass alle Aufrufe des HARDDISK Befehls, zu dem IDEDISK Befehl und die HARDTEST Aufrufe auf IDETEST umgeleitet werden. Des weiteren werden immer 2 IDE Sektoren (512 Byte) zu einem NKC konformen Sektor (1024 Byte) zusammengefasst.
Bei Übergabe einer 0 in d0 wird die Umleitung deaktiviert.

Nach erfolgreicher Ausführung des Befehls steht in d0 eine 0. Im Fehlerfall (IDE nicht verfügbar) steht in d0 ein $ffff

TRAP-Nummer:	153
Befehlsname:	GETS2I
Befehlsgruppe:	Harddisk
Kurzbeschreibung:	Status der SCSI nach IDE Umleitung lesen.
Eingaberegister:	KEINE
Ausgaberegister:	d0.b
Zerstörte Register:	KEINE
Ab Version:	7.0
Änderungen zu 4.3:	----
Änderungen zu 6.1:	----
Änderungen zu 6.3:	----
Siehe auch:	HARDTEST(140) HARDDISK(141) SETS2I(152) IDETEST(154) IDEDISK(155)

Beschreibung:

Der Status der SCSI nach IDE Umleitung wird in d0.b zurückgegeben.
D0.B = 1 Umleitung aktiv
D0.B = 0 keine Umleitung

TRAP-Nummer:	154
Befehlsname:	IDETEST
Befehlsgruppe:	Harddisk
Kurzbeschreibung:	Testet, ob ein IDE-Laufwerk vorhanden ist.
Eingaberegister:	d4.b = Auswahl des Laufwerks
Ausgaberegister:	d0.l = Fehlercode CARRY = 1, wenn kein Laufwerk vorhanden ist a0.l = Adresse der Laufwerksinformationen
Zerstörte Register:	KEINE
Ab Version:	7.0
Änderungen zu 4.3:	----
Änderungen zu 6.1:	----
Änderungen zu 6.3:	----
Siehe auch:	HARDTEST(140) HARDDISK(141) SETS2I(152) GETS2I(153) IDEDISK(155)

Beschreibung:

Dieses Programm ist dazu gedacht, dass ein Anwenderprogramm testen kann, ob eine bestimmte IDE-Harddisk vorhanden ist, da ja mit den DIL-Schaltern auf der KEY nur festgelegt wird, dass mindestens eine verfügbar ist. Wird das angesprochene Laufwerk gefunden, so werden außerdem noch die wichtigsten Daten des Laufwerks geladen.
In D0 steht ein 0 wenn das Laufwerk vorhanden ist und das CARRY-Flag ist zurückgesetzt, ansonsten eine -1 und das CARRY-Flag ist gesetzt.
Im Register A0.L wird die Adresse des Speichers angegeben, auf dem die Laufwerksdaten liegen.

Im Register D4.B muss dazu 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.

Wenn die scsi2ide Variable (siehe SETS2I) aktiv ist werden Aufrufe von HARDTEST auf diese Routine umgeleitet.

TRAP-Nummer:	155
Befehlsname:	IDEDISK
Befehlsgruppe:	Harddisk
Kurzbeschreibung:	Befehl an die IDE-Disk 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 welche verlangt werden
Ausgaberegister:	d0.l = Fehlercode CARRY = 1, wenn d0.l <> 0
Zerstörte Register:	KEINE
Ab Version:	7.0
Änderungen zu 4.3:	----
Änderungen zu 6.1:	----
Änderungen zu 6.3:	----
Siehe auch:	HARDTEST(140) HARDDISK(141) SETS2I(152) GETS2I(153) IDETEST(154)

Beschreibung:

Diese Routine ist analog zur HARDDISK Routine aufgebaut und dient zum Zugriff auf IDE Festplatten, die über die GIDE Baugruppe angeschlossen sind. Es werden gegenüber der HARDDISK Routine nur 5 Kommandos und 2 Laufwerke (Master und Slave) unterstützt.
Wenn die scsi2ide Variable (siehe SETS2I) gesetzt ist werden die HARDDISK Aufrufe auf diese Routine umgeleitet. Dies hat auch zur Folge, dass immer 2 Sektoren (1024 Byte) gelesen/geschrieben werden, damit werden IDE-Laufwerke NKC konform bedient. Ansonsten werden die Laufwerke im Standart IDE-Modus benutzt, d.h. die Sektoren haben eine Größe von 512 Byte.

Die verschiedenen Befehle werden über D1.W ausgewählt.
D2 und D3 haben je nach Befehl verschiedene Funktionen. In D4.B wird das Laufwerk ausgewählt, das angesprochen werden soll, da mit einer IDE-Schnittstelle 2 Laufwerke verwaltet werden können. Dabei entspricht ein gesetztes Bit 0 dem Laufwerk 0 (Master), ein gesetztes Bit 1 dem Laufwerk 1 (Slave). Die Kodierung ist also genau so wie beim Diskettenlaufwerk.

Nach dem Aufruf steht in D0.L ein Fehlercode. Ist kein Laufwerk vorhanden, so erhält D0.L den Wert -1 und das CARRY-Flag ist gesetzt. Ansonsten wird in D0.L der Fehlercode des Laufwerks zurückgeliefert, der sehr viele verschiedene Werte annehmen kann und je nach Befehl auch verschieden ist. In der Beschreibung über die IDE-Schnittstelle ist aber eine Tabelle der Returncodes aufgeführt. Wenn kein Fehler auftrat, so ist aber D0.L immer Null und das CARRY-Flag ist zurückgesetzt. Das CARRY-Flag ist aber bei einem Wert ungleich -1 immer zurückgesetzt. In D0.L wird übrigens auch eine -1 zurückgegeben, wenn D1.W beim Aufruf einen falschen Wert enthalten hat (Falscher Befehl).

Jetzt folgt die detaillierte Beschreibung der einzelnen Befehle. Register werden nicht zerstört und es werden außer in D0.L auch keine Werte zurückgegeben.

Nummer des Befehls: 1
Befehlsname: Read
Kurzbeschreibung: Bis zu 256 Sektoren lesen

Eingaberegister: d2.l = Anfangssektor (Nur 21 Bits genutzt)
d3.b = Anzahl der Sektoren
a0.l = Zieladresse

Dieser Befehl liest bis zu 256 hintereinander liegende Sektoren in den Speicher. Dabei wird in D2.L der Anfangssektor angegeben. Der kleinste Sektor ist Sektor 0 und 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 dabei für 256 Sektoren. Wenn von einem DOS Sektoren geladen werden sollen, die hintereinander liegen, so sollten diese immer in einem Stück eingelesen werden, da dies sehr viel schneller geht, als das mehrmalige Einlesen eines Sektors. In A0.L schließlich muss noch die Zieladresse im Ram angegeben werden.

Nummer des Befehls: 2
Befehlsname: Write
Kurzbeschreibung: Bis zu 256 Sektoren schreiben

Eingaberegister: d2.l = Anfangssektor (Nur 21 Bits genutzt)
d3.b = Anzahl der Sektoren
a0.l = Zieladresse

Dieses Programm funktioniert genau so, wie die Funktion 1 READ. Allerdings werden die Sektoren geschrieben und nicht gelesen.

Nummer des Befehls: 8
Befehlsname: Test Unit Ready
Kurzbeschreibung: Testen, ob Laufwerk bereit ist

Eingaberegister: KEINE

Es kann vorkommen, dass das Laufwerk nicht bereit ist, einen Befehl auszuführen. Dies passiert z.B. wenn das Laufwerk gerade eben erst eingeschaltet worden ist. Um abzufragen, ob das Laufwerk Daten übertragen kann, ist dieses Programm vorhanden.

Nummer des Befehls: 22
Befehlsname: Read Capacity
Kurzbeschreibung: Größe des Laufwerks lesen

Eingaberegister: d3.b = PMI-Bit
a0.l = Zieladresse für Daten

Mit diesem Befehl kann man die Größe einer Festplatte feststellen. Dies ist für jedes Betriebssystem notwendig, damit es weis, wie viele Sektoren zur Verfügung stehen.
In D3.B wird dabei ausgewählt, ob die Sektoranzahl des gesamten Laufwerks ausgegeben werden soll, oder der letzte Sektor eines Tracks. Enthält D3.B den Wert 0, so wird die Nummer des letzten Sektors des Laufwerks zurückgeliefert. Enthält D3.B den Wert 1, so wird bei den Daten der letzte Sektor eines Tracks zurückgeliefert. 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,D3           * Nummer des letzten Sektors holen
        MOVEQ   #22,D1          * Kennung für READ CAPACITY
        MOVEQ   #1,D4           * Laufwerk 0 (Master)
        LEA     ZIEL(PC),A0     * Ziel für Daten
        MOVE    #!IDEDISK,D7    * Befehle ausführen
        TRAP    #1
        ....                    * Fehlerbehandlung
        RTS

ZIEL:
        DS.L    1               * Ziel für größte Sektornummer
        DS.L    1               * Ziel für Blocklänge
```

Nummer des Befehls: 24
Befehlsname: Inquiry
Kurzbeschreibung: Daten des Laufwerks lesen (Name/Versionsnummer..)

Eingaberegister: a0.l = Zieladresse für Daten

Manchmal ist es für ein Programm wichtig, mit welchem Laufwerk es arbeitet, damit es eventuelle Spezial-Befehle oder wichtige 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.

TRAP-Nummer: 156
Befehlsname: SRDISK
Befehlsgruppe: FLO-Baugruppe
Kurzbeschreibung: Befehl an die SRAMDISK übergegeben und ausführen.

Eingaberegister:
Ausgaberegister:
Zerstörte Register: KEINE

Ab Version: 7.0
Änderungen zu 4.3: ----
Änderungen zu 6.1: ----
Änderungen zu 6.3: ----

Siehe auch: FLINIT(74) FLOPPY(75) GETFLOP(76)
SETF2S(157) GETF2S(158) GETSRD(159)

Beschreibung:

Diese Routine ist für den Zugriff auf die SRAMDISK Baugruppe. Sie ist analog zum FLOPPY Befehl aufgebaut, allerdings werden nur die Kommandos Sektor lesen und Sektor schreiben unterstützt.

Wenn die Variable flo2srd (siehe SETF2S) gesetzt ist, werden die Aufrufe der Routine FLOPPY für das Laufwerk 4, auf diese Routine umgeleitet.

Die SRAMDISK wird je nach Kapazität in folgende logische Blöcke aufgeteilt:

Kapazität	Anzahl	
KB	Sektoren	Spuren
512	8	64
1024	8	128
1536	16	96
2048	16	128

Jeder Sektor hat die Größe von 1024 Byte.
Die Sektoren werden mit 1 bis 8 (16) angegeben, die Spuren mit 0 bis 64-1 (96-1, 128-1)
Diese Adressierung entspricht der von Disketten.
Alternativ kann man auch nur eine Sektornummer angeben. Hierbei muss D3.L (Spuren) auf 0 gesetzt werden. In D2.L wird dann eine Sektornummer von 1 bis zur max. Kapazität (512, 1024, 1536 oder 2048) angegeben.

Registerbelegung:

A0.L Ist die Adresse für Ziel oder Quelle
Dieses Register wird immer 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.

D1.W Befehlscode:
In diesem Register wird festgelegt, welcher Befehl ausgeführt werden soll.
1 = Sektor lesen
Es werden 1024 Bytes gelesen. Dabei müssen die Register d2 und d3.b richtig belegt sein. Im Register A0.L wird die Zieladresse übergeben.

2 = Sektor schreiben
Es werden 1024 Bytes geschrieben.
Dabei müssen die Register d2 und d3.b richtig belegt sein. Im Register A0.L wird die Quelladresse übergeben.

D2 Sektor: 1...n (n=8 oder 16 je nach Laufwerk)
Hier wird der Sektor angegeben, auf den zugegriffen werden soll. Dabei hängt die Anzahl der Sektoren pro Spur von der Größe der SRAMDISK ab.
Alternativ 1...n (n=512, 1024, 1536 oder 2048 je nach Laufwerk)

D3.B Spur: 0...k (k=63, 95 oder 127 je nach Laufwerk)
Die Anzahl der Spuren hängt von der Art des Laufwerks ab.

TRAP-Nummer:	157
Befehlsname:	SETF2S
Befehlsgruppe:	FLO-Baugruppe
Kurzbeschreibung:	Floppy 4 nach SRAMDISK Umleitung setzen / löschen.
Eingaberegister:	d0.b
Ausgaberegister:	KEINE
Zerstörte Register:	KEINE
Ab Version:	7.0
Änderungen zu 4.3:	----
Änderungen zu 6.1:	----
Änderungen zu 6.3:	----

Siehe auch:	FLINIT(74)	FLOPPY(75)	GETFLOP(76)
	SRDISK(156)	GETF2S(158)	GETSRD(159)

Beschreibung:

Mit diesem Befehl wird die Variable flo2srd gesetzt oder gelöscht.

Zum setzen der Variablen wird im Register d0.b eine 1 übergeben. Alsdann werden alle Zugriffe auf die Floppy4 mit dem Befehl FLOPPY(75) auf die Routine SRDISK(156) umgeleitet. Hierbei sind allerdings nur die Funktionen Sektor lesen und Sektor schreiben verfügbar.

Zum löschen der Variablen wird im Register d0.b eine 0 übergeben. Danach erfolgen alle Zugriffe auf Floppy4 in gewohnter Weise.

Die Variable flo2srd wird automatisch gesetzt, wenn beim Bootvorgang die SRAMDISK gefunden wird.

TRAP-Nummer:	158
Befehlsname:	GETF2S
Befehlsgruppe:	FLO-Baugruppe
Kurzbeschreibung:	Status der Floppy4 nach SRAMDISK Umleitung lesen.
Eingaberegister:	KEINE
Ausgaberegister:	d0.b
Zerstörte Register:	KEINE
Ab Version:	7.0
Änderungen zu 4.3:	----
Änderungen zu 6.1:	----
Änderungen zu 6.3:	----

Siehe auch: FLINIT(74) FLOPPY(75) GETFLOP(76)
SRDISK(156) SETF2S(157) GETSRD(159)

Beschreibung:

Hiermit wird der Zustand der flo2srd Variablen abgefragt.

Die Rückgabe erfolgt im Register d0.b.

Eine 1 bedeutet, dass der Floppy4 nach SRAMDISK Umleitung aktiv ist.

Bei einer 0 ist die Umleitung deaktiviert, das bedeutet, dass Zugriffe auf das Floppy Laufwerk 4 normal möglich sind.

TRAP-Nummer:	159
Befehlsname:	GETSRD
Befehlsgruppe:	FLO-Baugruppe
Kurzbeschreibung:	Größe der SRAMDISK abfragen.
Eingaberegister:	KEINE
Ausgaberegister:	d0
Zerstörte Register:	KEINE
Ab Version:	7.0
Änderungen zu 4.3:	----
Änderungen zu 6.1:	----
Änderungen zu 6.3:	----

Siehe auch: FLINIT(74) FLOPPY(75) GETFLOP(76)
SRDISK(156) SETF2S(157) GETF2S(158)

Beschreibung:

Hiermit wird die Größe der SRAMDISK in d0 übergeben.

Wenn die SRAMDISK nicht vorhanden ist wird in d0 eine 0 übergeben, ansonsten die Größe in Kilobyte -1.

TRAP-Nummer:	160
Befehlsname:	SETSYS
Befehlsgruppe:	System-Routinen
Kurzbeschreibung:	Systemeinstellungen ins NVRAM speichern

Eingaberegister:	a0.l
Ausgaberegister:	d0
Zerstörte Register:	KEINE

Ab Version:	7.0
Änderungen zu 4.3:	----
Änderungen zu 6.1:	----
Änderungen zu 6.3:	----

Siehe auch:

GETRAM(59)	GETBASIS(89)	GETVAR(90)
SETA5(91)	GETVERS(97)	GETSN(98)
GRUND(124)	SUCHBIBO(136)	SYSTEM(139)
GETSYS(161)	PATCH(162)	

Beschreibung:

Hiermit wird ein Datenblock mit Systemdaten ins NVRAM geladen, diese bleiben auch nach Ausschalten des Computers erhalten und werden beim Starten des Computers verwendet.
In A0.L wird die Adresse eines xx Byte langen Datenblocks übergeben. Dieser hat folgenden Aufbau:

Byte	Daten
0-1	Kontrollsumme, diese wird von der Routine berechnet.
2	DIP-Key, hiermit wird die (Hardware) DIP Einstellung überschrieben
3	Reserve
4-7	Autoboot-Reihenfolge 0-2 = Floppy 1 bis 3 3 = Floppy 4 oder falls vorhanden SRAMDISK 4 = SCSI Harddisk 5 = IDE Harddisk
8-9	SER Einstellung
10-15	SER2 Kanal A
16-21	SER2 Kanal B

d0 = 0 Daten OK
d0 = -1 keine Uhr mit Dallas-Chip vorhanden

TRAP-Nummer:	161
Befehlsname:	GETSYS
Befehlsgruppe:	System-Routinen
Kurzbeschreibung:	Systemeinstellungen aus NVRAM holen

Eingaberegister:	a0.l
Ausgaberegister:	d0/a0.l
Zerstörte Register:	KEINE

Ab Version:	7.0
Änderungen zu 4.3:	----
Änderungen zu 6.1:	----
Änderungen zu 6.3:	----

Siehe auch:

GETRAM(59)	GETBASIS(89)	GETVAR(90)
SETA5(91)	GETVERS(97)	GETSN(98)
GRUND(124)	SUCHBIBO(136)	SYSTEM(139)
SETSYS(160)	PATCH(162)	

Beschreibung:

Mit dieser Routine wird ein Datenblock mit Systemdaten aus dem NVRAM in den Speicher geladen.
In A0.L wird die Adresse für den xx Byte langen Datenblock (siehe SETSYS) angegeben.

d0 =	0 Daten OK
d0 = -1	keine Uhr mit Dallas-Chip vorhanden
d0 = -2	Daten sind nicht korrekt

TRAP-Nummer:	162
Befehlsname:	PATCH
Befehlsgruppe:	System-Routinen
Kurzbeschreibung:	Trap-Aufrufe umleiten/definieren
Eingaberegister:	d0/a0.l
Ausgaberegister:	d0/a0.l
Zerstörte Register:	KEINE
Ab Version:	7.0
Änderungen zu 4.3:	----
Änderungen zu 6.1:	----
Änderungen zu 6.3:	----

Siehe auch:	GETRAM(59)	GETBASIS(89)	GETVAR(90)
	SETA5(91)	GETVERS(97)	GETSN(98)
	GRUND(124)	SUCHBIBO(136)	SYSTEM(139)
	SETSYS(160)	GETSYS(161)	

Beschreibung:

Achtung: diese Routine ist nur bei Systemen funktionsfähig, die RAM ab Adresse 0 haben!

Das Grundprogramm verfügt über die Möglichkeit 192 Trap-Aufrufe zu verwalten. Die Aufrufe bis zur Nummer 162 sind die hier beschriebenen Grundprogramm-Routinen. Mit diesem Befehl kann man nun einen Aufruf auf einen eigene Routine legen.
Dazu wird in D0 die Trapnummer und in A0.L die Adresse der Routine angegeben. Nach dem Aufruf enthält A0.L die Adresse der bisherigen Routine, die diese Trapnummer hatte (wichtig für eine Restaurierung der Trap-Tabelle) und in D0 steht eine 0 wenn der Aufruf erfolgreich war, ansonsten eine -1.
Besonders vorsichtig muss man sein wenn man bestehende Grundprogramm Routinen ersetzt, da evtl. auch andere Programme (z.B. JADOS) auf diese zugreifen.
Nach einem System-Reset ist die Standart Trap-Tabelle wieder hergestellt.

Beispiel:

```
WRITETXT EQU 170        * WRITETXT bekommt die Nummer 170

START:
 MOVE #WRITETXT, D0     * Trapnummer der Routine
 LEA WRTEXT, A0         * Adresse der Routine
 MOVE #!PATCH, D7
 TRAP #1                *  ab  jetzt  steht  die  Routine  für  Trapaufrufe  zur
Verfügung

 LEA TEXT(PC), A0       * auszugebender Text
 MOVE #WRITETXT, D7     * Nummer der neuen Routine. Achtung: ohne !
 TRAP #1

 RTS


WRTEXT:                 * Routine zum ausgeben von Texten bis zur 0
 MOVEM.L D0/D7, -(A7)   * Register sichern
WRTEXT1:
 MOVE.B (A0)+, D0       * Byte holen
 BEQ.S WRTEX            * wenn 0, dann Ende
 MOVE #!CO2, D7         * CO2
 TRAP #1                * aufrufen
 BRA.S WRTEXT1          * und weitere Zeichen ausgeben
WRTEX:
 MOVEM.L (A7)+, D0/D7   * gesicherte Register zurück
 RTS

TEXT:                   * auszugebender Text
 DC.B 'Hallo Welt',0
```

5. Ergänzende Informationen

Wenn man die vorangegangenen Kapitel sorgfältig durchgelesen hat, so kann man mit dem Grundprogramm schon wirklich sehr gut arbeiten. Die notwendigen Informationen zum System sollten jetzt eigentlich bekannt sein.

Dieses Kapitel ist für diejenigen Benutzer und Programmierer gedacht, die sich sehr intensiv 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 so ein Listing in welcher Form auch immer anzuschaffen, um die nun folgenden Informationen genau benutzen zu können. Es gibt nämlich noch eine ganze Fülle von Dingen, die einem bei der Programmierung helfen und noch nicht angesprochen wurden.

5.1. Allgemeiner Aufbau und Funktion

Das Grundprogramm ist in drei Teile aufgebaut, die auch im Speicher getrennt voneinander vorhanden sind und deshalb unterschieden werden müssen. Auch im Listing sind diese Teile getrennt.

Der erste Teil ist der Bereich der Exceptions. Er liegt zwar mit im Eprom, wird aber nach dem Start und bei jedem RESET auf die Adresse 0 geschrieben, falls sich das Grundprogramm nicht auf dieser Adresse befinden sollte. Der Bereich ist genau 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 erfolgte. Dadurch, dass er sich im RAM befindet, können die Adressen vom Benutzer verändert werden.

Der zweite Bereich ist der Programmbereich. Er erstreckt sich von der 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 Werts**	**Bedeutung**
$400	Ein Langwort	Kennung, dass das Grundprogramm vorhanden ist. Dort steht der Wert $5aa58001.
$404	Ein Langwort	Dort ist der Beginn des Variabelenbereichs relativ zum Anfang der Eproms angegeben (Normalerweis $10000).
$408	Ein Langwort	Adresse des Kaltstarts relalativ zum Anfang der Eproms.
$40C	Ein Langwort	Versionsnummer (Version 6.2 = $620)
$410	Ein Langwort	Seriennummer (nicht genutzt)
$414	Ein Langwort	CPU-Kennung (1=68008, 2=68000, 4=68020)
$418	Zwei Langworte	Reserve (2 mal 0.l)
$420	2 Worte	Sprung auf TRAP-Mechanismus mit RTS-Abschluß.
$424	2 Worte	Sprung auf Kaltstart.
$428	4 Worte	2 mal Sprung auf Adresse 0 als Reserve.

Der dritte Bereich sind die Variablen, die normalerweise direkt hinter den Grundprogrammeproms liegen.

5.2. Die Variablen

Zu den Variablen ist eigentlich nicht sehr viel zu sagen. Sie liegen immer direkt hinter dem Grundprogramm, es sei denn der Wert der Adresse RAMSTART ($404) im ROM wird geändert.

Wichtig ist eigentlich nur, dass der Variabelenaufbau für die wichtigsten Variablen immer gleich ist und dass die Länge des Variabelenbereichs nicht fest ist.

Die Länge des Variabelenbereichs 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 sehr leicht dem Listing des Grundprogramms entnommen werden.

5.3. Exceptions und Interrupts

Eigentlich 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 irgendetwas zu steuern. Die anderen Exceptions treten in unregelmäßigen Abständen gewollt oder ungewollt auf.

Nun zunächst einmal zu den Interrupts. Hier soll nicht näher darauf eingegangen werden, was genau Interrupts sind und was man mit ihnen machen kann, hier soll nur der grundsätzliche Ablauf im Betrieb mit dem Grundprogramm erklärt werden. Nähere Informationen sind in verschiedenen Büchern vorhanden.

Um sich nicht zu sehr in die Programmierung einzumischen, wird die Interruptbehandlung nicht vom Grundprogramm durchgeführt. Dadurch bleiben alle Möglichkeiten der Anwendung erhalten. Das einzige, was das Grundprogramm macht, ist die Bereitstellung von festen Adressen, wo die CPU "hinspringt" oder besserer gesagt die Programmausführung beginnt, wenn ein Interrupt auftritt. Diese Adressen sind immer gleich. Diese Adressen liegen im RAM direkt hinter dem Grundprogramm.

Beim 68008 stehen dabei nur die Ebenen 2,5 und 7 zur Verfügung, was nicht am Grundprogramm liegt, sondern am Aufbau der CPU.

Jede Interruptadresse hat eine Länge von 6 Bytes, wodurch genau Platz für einen JMP-Befehl ist. Auf diesen Adressen wird nämlich genau nur dieser Befehl eingesetzt, der einen Sprung auf die eigene Routine durchführt. Bei jedem Neustart des Grundprogramms stehen dort Sprünge auf eine interne Grundprogrammroutine, damit unerwünscht auftretende Interrupts abgefangen werden. Die interne Routine zeigt auf dem Bildschirm an, dass ein Interrupt auftrat und führt dann zurück ins Grundprogramm. Diese festen Adressen im RAM sind dem Listing des Grundprogramms zu entnehmen.

Alle weiteren Exceptions, die sonst noch auftreten können, wie BUS-Fehler ADRESS-Error o.ä. werden auch auf diese Routine geführt und am Bildschirm angezeigt. Dann erscheint oben auf dem Bildschirm die Information, welche Exception auftrat und unten werden die Register ausgegeben, die beim 68020 wie beim Einzelschritt umgeschaltet werden können. Mit einem Tastendruck aufs "M" wird das Grundprogramm dann neu gestartet, wobei die Variablen neu initialisiert werden. Mit "W" gelangt man auch ins Grundprogramm zurück, allerdings wird nur der Stack neu eingerichtet, während alle anderen Variablen und Sprungadressen nicht verändert werden. Diese Funktion kann bei "harmlosen" Exceptions wie CHK, DIVISION DURCH NULL o.ä. verwendet werden, um zum Beispiel eine Umlenkung der CO2-Routine nicht zurückzusetzen.

Wenn das Grundprogramm nicht auf der Adresse 0 liegt, so stehen diese ganzen Adressen im RAM von der Adresse 0 an, so dass sie auch sehr leicht geändert werden können, wobei sie dann auf spezielle Routinen gelenkt werden könnten.

Die letzte Gruppe von Exceptions sind die vom Benutzer direkt gewollten und im Programm angegebenen Exceptions. Dies sind die TRAP-Aufrufe. Alle TRAP-Vektoren sind ins RAM hinterm Grundprogramm geführt und werden von dort weiter verzweigt. Die Adressen der TRAPs sind aus dem Listing zu entnehmen.

Die TRAPs 1 und 6 sind übrigens reserviert.

Eine weitere Besonderheit ist beim TRAP #1 vorgesehen. Es stehen jetzt alle Adressen der TRAP-Unterprogramme auf den Adressen der NON-AUTO-Vektoren, die auf der Adresse $100 beginnen. Wenn das Grundprogramm nicht auf der Adresse 0 liegt, so 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 genüg, einfach auf die entsprechende Adresse eine neue Sprungadresse zu schreiben. Der TRAP-Mechanismus holt sich nämlich aus dem RAM die Adresse und spring zu dem entsprechenden Unterprogramm. Dabei steht im RAM direkt die Adresse der Routine. Beim Initialisieren wird die Basisadresse des Grundprogramms nämlich schon berücksichtigt. Wenn eine Sprungadresse umgelenkt wird, so wird nicht nur beim TRAP die neue Routine angesprungen, sondern es wird dann z.B. auch bei der WERT-Routine oder beim Assembler beim Ausdruck @name diese neue Adresse zurückgegeben, weshalb bei der Verwendung dieser Funktion etwas Vorsicht geboten ist. Wird nur die TRAP-Funktion benutzt, so kann eigentlich nichts schief gehen.

Damit auch für den Anwender noch NON-AUTO-Vektoren zur Verfügung stehen, sind die Vektoren 180 bis 192 für Anwendungen frei gehalten. Dem Grundprogramm stehen damit aber immer noch 180 zur Verfügung.

Die Vektoren von 0 bis $400 werden bei jedem Start oder RESET neu initialisiert. Die Vektoren hinter dem Grundprogramm im Ram werden nur initialisiert, wenn keine POWER-ON-Erkennung vorhanden ist. Dadurch bleiben umgelenkte Routinen auch nach einem RESET umgelenkt.

5.4. Die Symboltabelle

Weil der Aufbau der Symboltabelle sehr wichtig für Programme ist, die größere Datenmengen sortieren oder verwalten wollen, oder auf die 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. Sie ist eine 16 Bit Zahl ohne Vorzeichen. Das bedeutet, dass ein Symboltabellenbereich von 64 KByte oder 3640 Einträgen zur Verfügung steht. 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.

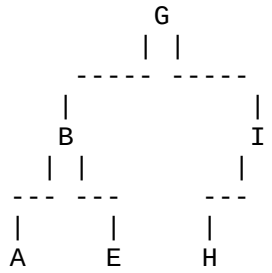
Die zweite Variable ist eigentlich nicht direkt eine Variable, sondern die Anfangsadresse der Tabelle. Auf dieser Adresse liegt der erste Eintrag. Der Anfang der Tabelle kann durch die Funktion GETSYM ermittelt werden. Mit jedem Eintrag wird die Symboltabelle 18 Bytes länger, da jeder Eintrag im Grundprogramm immer 18 Bytes lang ist. Bei eigenen Routinen können die Einträge auch andere Längen haben, da der nächste Eintrag immer relativ zum Symboltabellenanfang angegeben wird. Wenn eine andere Länge der Einträge gewählt wird, so dürfen aber nicht immer die internen Grundprogrammroutinen verwendet werden.

Nun aber zum direkten Aufbau der Tabelle.
Jeder Eintrag enthält folgende Daten.

Name	**Länge**	**Funktion**
KLEINER	1 Wort	Zeiger auf einen Eintrag, der einen Namen hat, der im Alphabet weiter vorne steht.
GROESSER	1 Wort	Zeiger auf einen Eintrag, der einen Namen hat, der im Alphabet weiter hinten steht.
DATENWERT	1 Langwort	Wert des Variablen
ATTRIBUT	1 Wort	Dieses Wort enthält nähere Informationen über den Wert. 0 = Wert ist ungültig Der Ausdruck, durch den der Wert berechnet werden sollte, enthielt einen Syntax-Fehler. 1 = Bytewert 2 = Wortwert 3 = Langwort 4 = Reserviert 5 = Undefiniertes Symbol Der Wert sollte durch ein Symbol definiert werden, das selbst nicht definiert war.
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 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 all diese Dinge hält, so bleibt die Symboltabelle auch bei direkten Manipulationen kompatibel zum Grundprogramm, und man kann 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 auch selber 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 noch 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, so 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 auch die nötigen Informationen liefert. Falls man nicht immer selber rechnen möchte, kann man mit ihr sehr 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 direkt 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, so wird der Cursor bei Routinen wie CI automatisch dargestellt. (siehe CURON, CUROFF, CURSEIN, CURSAUS).

5.6. Speicherbereiche

In Kapitel 2.5. wurde zwar schon kurz auf die Speicheraufteilung eingegangen, aber hier sollen noch einmal alle Speicherbereiche aufgeführt werden.

1) Von 0 - $400 befindet sich die Exceptiontabelle, die im RAM liegt, wenn das Grundprogramm nicht auf der Adresse 0 vorhanden ist.

2) Der Editorbereich ist frei verschiebbar und benötigt, je nach Textumfang mehr oder weniger RAM.

3) Hinter dem Editor wird jedes Mal beim Assemblieren eine Tabelle angelegt, die Daten über die verwendeten MACROs des Assemblers enthält. Es sind aber auch Einträge vorhanden, wenn keine MACROs im Programm definiert sind.

4) Hinter der MACRO-Tabelle wird eine wenig 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, so 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 5.2. und 5.4.).

7) Der Stack liegt normalerweise am Ende des ersten Speicherbereichs hinter dem Grundprogramm. Manche Betriebssysteme verlegen ihn aber (z.B. JADOS).

5.7. Bibliotheks-Funktion

Das Grundprogramm enthält eine Bibliotheks-Funktion, die sehr einem sehr leicht ermöglicht, viele verschiedene Programme auf verschiedenen Adressen im Speicher oder Eproms zu haben und die dann sehr leicht zu finden. Im Menü gibt es dazu die bekannte Funktion, die alle Einträge auflistet. Hier soll gezeigt werden, wie ein solcher Eintrag aussehen muss, damit ein Programm gefunden werden kann.

Zuerst einmal muss ein Bibliotheks-Eintrag auf einer 1-Kbyte-Grenze beginnen, da ansonsten zu viel Speicher durchsucht werden muss. Ein Eintrag hat dabei folgende Form:

DC.L	$55AA0180	Kennung, dass Eintrag vorhanden ist
DC.B	'Name '	Name des Programms (genau 8 Zeichen)
DC.L	Startadresse	Gibt die Startadresse des Programms an Dieser Wert 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 aber normalerweise nicht ausgewertet.
DC.B	Reloaktiv-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 hängen bleibt, wenn es nicht für die CPU geeignet ist. Dabei sind vier Werte zulässig. 0 = Das Programm ist für alle CPUs geeignet 1 = Das Programm ist nur für den 68008 geeignet 2 = Nur für 68000/68010 4 = Nur für 68020
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.