
```
   JJJJJ        A        DDDDDD         OOO         SSSSS
   JJJJJ       A A       DD    D      O   O      SS    S
     JJ       AA AA      DD    DD    OO   OO     SS
     JJ       A   A      DD     D    OO   OO      SSSSS
     JJ      AAAAAAA     DD    DD    OO   OO          SS
JJ   JJ      AA   AA     DD    D      O   O      S    SS
 JJJJJ       AA   AA     DDDDDD        OOO        SSSSS
```

DAS DISKETTEN-BETRIEBSSYSTEM

FÜR DIE PROZESSOREN 68008 , 68000 UND 68020 IM NDR-KLEIN-COMPUTER

KLAUS JANßEN
KREFELD

VERSION 2.1

Graf Elektronik Systeme GmbH

VORWORT ZUR VERSION 2.1

Seitdem im Frühjahr 1986 die JADOS-Version 1.2 auf den Markt kam, ist ein gutes Jahr vergangen, und ich freue mich, nunmehr Version 2.1 präsentieren zu können.

Die neue Version dient im wesentlichen dazu, alle bislang bekanntgewordenen Unzulänglichkeiten von Version 2.0 auszumerzen und den Bedienungskomfort sowie die Sicherheit bei Floppyzugriffen spürbar zu erhöhen. Als besonderes Feature sei hier erwähnt, daß nun bis zu 20 Programme resident im Hauptspeicher gehalten werden können.

In diesem Zusammenhang möchte ich mich für die zahlreichen und anregenden Zuschriften von JADOS-Anwendern bedanken. Namentlich erwähnen möchte ich hier R. Bäcker, A. Granel, M. Husemann und besonders U. Koch.

Sehr herzlich bedanke ich mich bei meiner Ehefrau Claudia, die mir bei der Erstellung des Handbuchs tatkräftig geholfen hat und noch immer Verständnis für meine zeitverschlingende JADOS-Pflege aufbringt,

Krefeld, im März 1987 Klaus Janßen

INHALTSVERZEICHNIS

1) A L L G E M E I N E S

1.1) EINLEITUNG

JADOS ist ein Betriebssystem für die Prozessorfamilie 680xx im NDR-Klein-Computer und unterstützt den Betrieb mit ein bis vier Minidiskettenlaufwerken (5 1/4 '' oder 3 1/2 '') im Format NDR80.

Die vorliegende Version 2.1 ist vollständig aufwärts kompatibel zu den Versionen 1.2.und 2.0. Das bis dato schon recht leistungsfähige JADOS wurde stark verbessert, sowohl im **Funktionsumfang** als auch in den Punkten **Schnelligkeit** und **Bedienungskomfort.**

1.2) ANDERUNGEN ZU VERSION 2.00

Folgende Schwächen der Version 2.0 wurden beseitigt:

- Das Rückschreiben des FCB erfolgt jetzt nur noch nach Schreibvorgängen. Die Probleme mit dem Schreibschutz sind damit aufgehoben.
- Sämtliche Floppyzugriffe sind jetzt stärker abgesichert.
- JADOS arbeitet nun auch mit anderen Bootprogrammen zusammen, z.B. dem FLOBOOT von A. Granel.
- COM-Dateien können nun fehlerfrei aufgerufen werden.
- Der Default-Textstart beim ASS-Kommando liegt jetzt um 16 KByte hinter der Ladeadresse.
- Die formalen Parameter in Batchdateien beziehen sich jetzt auch auf Teilstrings und Kommandos.
- Das Dateisuchmuster beim DIR-Befehl ist nun verbessert.
- Das Ein - und Ausschalten der Laufwerksmotoren ist verbessert. Das lästige "Ruckeln" der Laufwerke ist damit verschwunden.
- Der Urlader lädt nun "bedingungslos" von der Disk. Die Problematik mit Stackfehlern ist beseitigt.
- Ein einmal geladenes JADOS kann nun uneingeschränkt mit der Bibliotheksfunktion gestartet werden; auch nach einem RESET.

Neu hinzugekommen sind:

- Die Laufwerke 3: und 4: sind jetzt eingebunden. Sie müssen aber ebenfalls Minilaufwerke im NDR 80 - Format sein.

- Programme können nun mit dem INST-Kommando resident in den Hauptspeicher geladen werden.

- Die residenten Programme (bis 20) werden normal im Kommandointerpreter gestartet.

- Mit dem TABLE-Kommando werden die residenten Programme aufgelistet.

- Die Kommando - und Dateinameneingabe ist wesentlich komfortabler geworden. So können jetzt einzelne Zeichen gelöscht, eingefügt und einzeln aus dem Puffer abgerufen werden.

Weggefallen sind:

- Das interne Kommando ECHO ist jetzt ohne Wirkung, aber aus Kompatibilitätsgründen noch vorhanden.

- Das SYS-Kommando ist nun kein internes Kommando mehr.

Neue Unterprogramme:

- "getdrive" und "setdrive" zur Behandlung des Standardlaufwerks

- "ramtop" zur Abfrage des Endes des Benutzerspeichers

- "gttraptb" zum Ersetzen von JADOS-Unterprogrammen durch eigene

- "blockread", "blockwrite" zum schnellen Lesen und Schreiben von Teilen einer Datei

- "floppy" mit verbessertem Einschaltverhalten des Laufwerksmotors

- "delete", "insert" und "skipchar" zur Stringverarbeitung

Neue externe Kommandos

- DISKCOPY zum schnellen Kopieren ganzer Disketten

- GP, ein komfortables Interface zum Grundprogramm

- SYS zum Kopieren des Systems mit Versionsanzeige und Sicherheitscheck

Verbesserte externe Kommandos

- FORMAT jetzt für maximal 4 Laufwerke und schnelles Prüflesen
- MORE jetzt auch mit deutschen Umlauten
- BRDRUCK in völlig neuem Gewand mit stark erweiterten Funktionen
- ROMSTART findet jetzt alle Bibliothekseinträge

Quellcodes zwecks Anpassung für:

- BRDRUCK
- GP

In letzter Minute noch geändert:

- Für Computerkonfigurationen OHNE BOOTKARTE gilt bezüglich der Ladeadresse:

  Bis Grundprogrammversion 4.3 ist die Ladeadresse unverändert $0C000.
  Ab Grundprogrammversion 5.0 ist die Ladeadresse jetzt $14000.

1.3) LEISTUNGSUMFANG

* Laden, Speichern und Löschen von Dateien, Texten und Programmen
* Die Bereiche gelöschter Dateien stehen wieder uneingeschränkt zur Verfügung
* Koppeln von Texten; manuell oder über Antwortdatei
* Ändern von Dateinamen
* Selektives Inhaltsverzeichnis
* Laufwerksumschaltung über Tastatur
* Automatische Abarbeitung einer Startdatei nach dem Booten
* Volle Unterstützung des Assemblers von R.-D. Klein
* Volle Unterstützung des Editors von R.-D. Klein
* Hardcopy des Textinhalts
* Superschnelle Textausgabe mit Vor- und Rückwärtsblättern
* Formatieren im NDR80-Format
* Sehr komfortable Druck-Funktion
* Laden von Programmen mit Autostart
* Unterstützung der CPUs 68008, 68000 und 68020 !!
* Läuft mit und ohne BANKBOOT-Karte !!
* Installieren residenter Programme
* Superschnelles Starten residenter Programme
* Superschnelles Kopieren kompletter Disketten (unter 70 sek.)
* über 60 Unterprogramme für den Programmierer
* Komplette Kommandoabläufe können mit Hilfe von Kommandodateien automatisiert werden; ähnlich SUBMIT bei CP/M
* Deutsche Bedienerführung und Fehlermeldungen
* Umfangreiches, klar verständliches deutsches Handbuch
* Sehr günstiger Preis !!!!!

2) I N B E T R I E B N A H M E

2.1) GRUNDVORAUSSETZUNGEN

Für den Betrieb von JADOS sind folgende Baugruppen unbedingt erforderlich:

2.1.1) Konfiguration ohne Bootkarte

- CPU-Karte mit 68008-, 68000- oder 68020-Prozessor.
- Speicherkarten mit Grundprogramm von R.-D. Klein ab *Version 4.3* ab Adresse $00000.
- RAM-Speicher von mindestens *96 KByte* lückenlos ab Speicherplatz $08000.
- Grafikcontroller GDP64.
- Monitor für Darstellung von 80 Zeichen.
- Diskettencontroller FLO2 oder FDC.
- Tastaturinterface KEY.
- ASCII-Tastatur
- Ein Mini-Diskettenlaufwerk doppelseitig, 80 Spuren pro Seite (z.B. TEAC FD 55F).

2.1.2) Konfiguration mit Bootkarte

- CPU-Karte mit 68008-, 68000- oder 68020-Prozessor.
- Speicherkarten mit Grundprogramm von R.-D. Klein ab *Version 4.3* auf einer Adresse ungleich 0.
- RAM-Speicher von mindestens *16 KByte* lückenlos hinter dem Grundprogramm und 64 KByte ab Adresse $00000.
- Bankbootkarte(n) mit Booteprom
- Grafikcontroller GDP64.
- Monitor für Darstellung von 80 Zeichen.
- Diskettencontroller FLO2 oder FDC.
- Tastaturinterface KEY.
- ASCII-Tastatur
- Ein Mini-Diskettenlaufwerk doppelseitig, 80 Spuren pro Seite (z.B. TEAC FD 55F).

2.2) SINNVOLLE ERWEITERUNGEN

Für den Betrieb unbedingt empfehlenswert sind folgende Erweiterungen:

- Centronicsschnittstelle für einen Drucker.
- Matrixdrucker (vorteilhaft ist EPSON oder dazu Kompatibler).
- Speichererweiterung auf 256 KByte im ersten zusammenhängenden Speicherbereich.
- Zweites Diskettenlaufwerk.
- SOUND-Baugruppe.

2.3) STARTVORGANG

Es gibt zwei Möglichkeiten JADOS zu starten, nämlich über den Menüpunkt Floppy Start im Grundprogramm oder über die Bibliotheksfunktion.

2.3.1) Floppy Start

Nach dem Einschalten des Computers oder wenn JADOS im Hauptspeicher überschrieben wurde, muß JADOS von der Systemdiskette geladen werden.

Wurde JADOS bereits installiert und soll erneut gestartet werden, so kann es mit Floppy Start gestartet werden, ohne daß es zu einer Mehrfachinstallation kommt.

Es muß die Systemdiskette in das Laufwerk 1 eingelegt und folgendes Menü im Grundprogramm angewählt werden:

```
1 = Speichern CAS
2 = Laden CAS
3 = Pruefen CAS
4 = Floppy Start
W = weiter
```

Mit dem *Menüpunkt* 4 wird JADOS gestartet.

Nach erfolgreichem Startvorgang meldet sich JADOS mit:

```
J A D O S   VERSION 2.10   CPU 680XX
(C) 1985/86/87 BY KLAUS JANßEN

FREIER BENUTZERSPEICHER (KBYTE) : XXX
1>
```

Es werden die jeweils vorhandene CPU und der tatsächlich vorhandene freie Benutzerspeicher angezeigt.

Das Zeichen 1> mit dem blinkenden Cursor zeigt an, daß JADOS bereit ist, Kommandos entgegenzunehmen.

Der Startvorgang kann beschleunigt werden, indem das Original-Bootprogramm durch z.B. FLOBOOT von A. Granel ersetzt wird, welches den Umweg über die Grundprogramm-Menüs erspart.

2.3.2) Bibliotheksstart

JADOS besitzt die Bibliothekskennung und kann daher auch über die Bibliotheksfunktion gestartet werden, sofern es sich im Hauptspeicher befindet. Dies erfolgt sehr viel schneller als der Ladevorgang von Diskette. Seit Version 2.1 funktioniert dies nun auch nach einem RESET oder einer Ausnahmeverarbeitung.

2.3.3) Der AUTOSTART-Mechanismus

Nach jedem Floppy Start von der Systemdiskette sucht JADOS nach einer Datei AUTOEXEC.BAT auf Laufwerk 1. Diese Kommandodatei (siehe auch Kapitel 5.3.3 - Stapelverarbeitung) wird, falls sie existiert, automatisch gestartet. Damit kann der Benutzer Systeminitialisierungen durchführen, ein Anwendungsprogramm starten oder z.B. einige externe Kommandos resident in den Hauptspeicher laden.

Dieser Mechanismus wird nur ausgeführt, wenn JADOS komplett von der Systemdiskette geladen wurde, d.h nicht nach einem Bibliotheksstart.

2.3.4) Fehlersituationen

Es gibt verschiedene Fehlersituationen, die beim Laden oder Starten von JADOS auftreten können.

Beim Urladevorgang können folgende Ereignisse eintreten:

- Das Grundprogramm meldet sich wieder. Dann war kein System auf der Diskette.
- Der Urlader meldet: FALSCHES FORMAT. Dann liegt keine Minidiskette mit Double Density - Formatierung vor.
- Der Urlader meldet: ZU WENIG SPEICHER. Dann ist der erste zusammenhängende Speicherbereich zu klein. Der Speicherbereich muß mindestens 64 KByte umfassen.
- Der Urlader meldet: LESEFEHLER BEIM BOOTEN. Dann ist die Systemdiskette nicht mehr in Ordnung. **Deshalb ist es wichtig, sich eine Sicherheitskopie der Systemdiskette anzulegen !**

Beim Start des JADOS können folgende Meldungen vorkommen:

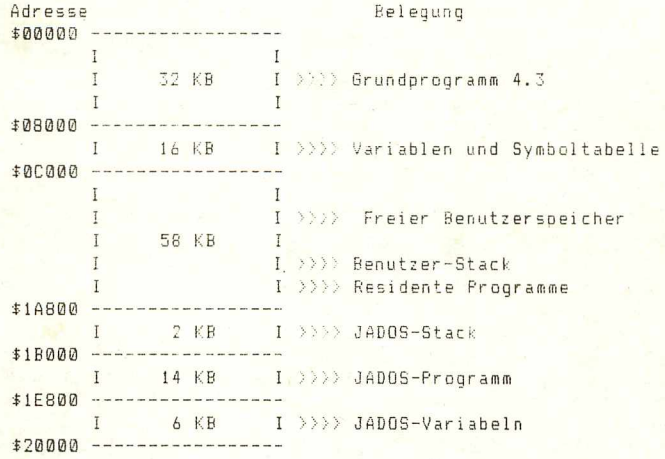
- JADOS meldet: ZU WENIG SPEICHER !! Dann ist kein freier Benutzerspeicher vorhanden.

JADOS führt seit V 2.10 eine eigene Ausnahmeverarbeitung durch. Die Fehler "ADDRESS ERROR", "ILLEGAL INSTRUCTION" usw. werden von JADOS abgefangen und führen zu einem automatischen Kaltstart des JADOS aus dem Hauptspeicher. Dies funktioniert aber nur, wenn eine Bootkarte eingesetzt wird und ab Adresse 0 RAM liegt.

3) D I E S P E I C H E R V E R W A L T U N G

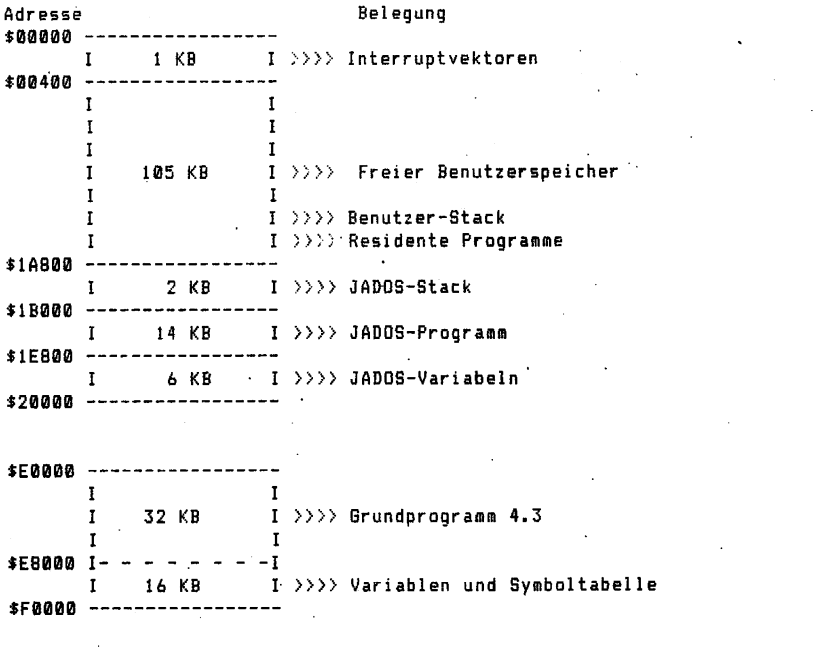
JADOS verwaltet den ersten zusammenhängenden Speicherbereich.
Dabei werden zwei verschiedene Speichermodelle unterschieden,
abhängig davon, ob eine Bankbootkarte verwendet wird oder nicht.

Die folgende Abbildung zeigt das Speichermodell einer typischen
Konfiguration mit 96 KByte RAM ohne Einsatz der Bankbootkarte:



Auf Adresse $00000 bis $07FFF liegt das Grundprogramm. Es belegt
32 KByte Speicherplatz.
Auf Adresse $08000 bis $0BFFF werden die Variablen des Grundprogramms
sowie die sogenannte Symboltabelle verwaltet. Dafür stehen insgesamt
16 KByte RAM zur Verfügung. Für die Symbole allein sind etwa 13 KByte
reserviert, was etwa 720 Symbolen entspricht.
Am Ende des 1. zusammenhängenden Speicherbereichs liegt das JADOS
mit seinen Variablen und einem eigenen 2 KByte tiefen Stack.
Ab Adresse $0C000 beginnt der Benutzerspeicher. **Die Adresse $0C000
ist gleichzeitig die Ladeadresse für die .68K-Dateien.**
Der Benutzerspeicher besteht aus freiem Benutzerspeicher, dem Stack
und dem Speicher für die residenten Programme. Die Größe des freien
Speichers hängt von der Anzahl und der Größe residenter Programme ab
und wird von JADOS dynamisch verwaltet.

Die folgende Abbildung zeigt das Speicherabbild einer typischen Konfiguration mit Bankbootkarte, 128 KByte Ram ab Adresse **$00000**, dem Grundprogramm auf Adresse **$E0000** sowie 16 KByte RAM ab Adresse **$E8000**.



Bei dieser Konfiguration sind zwei Speicherbereiche interessant.
Im 1. Speicherbereich liegen ab Adresse **$00000** die Interruptvektoren.
Ab Adresse **$00400** beginnt der JADOS-Benutzerspeicher. **$00400 ist die Ladeadresse für .68K-Dateien.**
Das JADOS-System selbst belegt wieder den hinteren Teil des ersten Speicherbereichs und umfaßt derzeit etwa 22 KByte.

Im zweiten Speicherbereich liegt das Grundprogramm. Daran schließt sich der Bereich der Grundprogrammvariablen und der Symboltabelle an. Wer übrigens mit den weiter oben erwähnten 720 Symbolen nicht auskommt, der kann den RAM-Speicher hinter dem Grundprogramm auf 32 KByte ausbauen und hat dann Platz für fast 1500 Symbole.

4) DIE DATEIVERWALTUNG

Die in den Abschnitten 'INHALTSVERZEICHNIS' und 'SPURTABELLE' angegebenen Informationen richten sich vor allem an diejenigen Benutzer, die sich für den internen Aufbau von JADOS interessieren. Der nicht daran Interessierte kann diese Abschnitte überspringen.

4.1) DATENTRÄGERFORMAT

Datenträger sind doppelseitige Disketten mit 96 Spuren pro Zoll. Das Diskettenformat ist wie folgt (mc - Format) :

- doppelseitig
- doppelte Dichte
- Miniformat 5 1/4 Zoll oder 3 1/2 Zoll
- 80 Spuren pro Seite
- 5 Sektoren pro Spur
- 1024 Byte pro Sektor

Andere Formate werden nicht unterstützt.

4.2) DISKETTENKAPAZITÄT

JADOS verwaltet *160 logische Spuren* pro Diskette. Dabei entsprechen den Spuren 0 - 79 die physikalischen Spuren 0 - 79 auf Seite 0 und den Spuren 80 - 159 die physikalischen Spuren 79 - 0 auf Seite 1.
Es gilt folgende Einteilung:

Spuren	Sektoren	Inhalt	Kapazität
0 - 2	1 - 5	Betriebssystem	15 KByte
3	1	Spurtabelle	1 KByte
3	2 - 5	Inhaltsverzeichnis	4 KByte
4 - 159	1 - 5	Benutzerdateien	780 KByte

4.3) INHALTSVERZEICHNIS

Pro Datei werden 32 Bytes Informationen abgelegt. Da für das Inhaltsverzeichnis 4 Sektoren zur Verfügung stehen, kann JADOS *128 Dateien pro Diskette* verwalten.

Für jede Datei ist folgender Datensatz eingetragen:

Byte-Nr	Inhalt
0 - 1	Kennung, z.B. $E5E5 -> kein Eintrag $0000 -> gültiger Eintrag $FFFF -> gelöschter Eintrag
2 - 9	Dateiname
12 - 14	Namenserweiterung bzw. Dateityp
16 - 17	Startspur der Datei
18 - 19	Endesektor, d.h. letzter Sektor der letzten Dateispur
22 - 25	Ladeadresse
26 - 27	Dateilänge, gemessen in Sektoren

Die Bytes Nummer 10,11 und 15 sind intern reserviert. Die Bytes Nummer 20 bis 21 und 28 bis 31 sind für eine spätere Erweiterung reserviert und werden bisher nicht ausgenutzt. Eine sinnvolle Erweiterung wäre z.B. das Datum des letzten schreibenden Zugriffs auf die Datei.

4.4) SPURTABELLE

Bei sehr einfachen Betriebssystemen werden Dateien fortlaufend Spur für Spur abgespeichert. Dies ermöglicht zwar ein schnelles Laden und Speichern der Dateien, hat aber einen wesentlichen Nachteil:
Wenn eine schon vorhandene Datei vergrößert werden soll, so paßt die Ergänzung häufig nicht mehr auf den der Datei zugewiesenen Bereich. Für die gesamte Datei (einschließlich Erweiterung) muß in diesem Fall ein Bereich gesucht werden, der ausreichend groß ist. Meistens liegt ein solcher Bereich hinter der letzten Datei. Der der ursprünglichen Datei (vor der Erweiterung) zugewiesene Bereich wird in der Regel gelöscht und steht einer neuen Datei zur Verfügung. Da Dateien nun aber immer nur fortlaufend, d.h. an einem Stück gespeichert werden können, darf diese neue Datei auf keinen Fall größer als der frei gewordene Bereich sein. Da es in der Praxis aber nicht immer möglich ist, eine Datei "herbeizuzaubern", die genau in die frei gewordene Lücke paßt oder nur wenig kleiner ist, kann es häufig vorkommen, daß ein gelöschter Bereich von einer sehr viel kleineren Datei belegt wird oder überhaupt nicht mehr belegt wird, da man zufällig nur noch größere Dateien zu speichern hat. Dabei geht dann sehr viel Diskettenkapazität verloren, was eine recht unökonomische Speicherplatznutzung darstellt. In ungünstigen Fällen (nach vielen Löschvorgängen) paßt keine Datei mehr auf die Diskette, obgleich vielleicht nur wenige Prozent der Diskettenkapazität ausgenutzt sind.

JADOS verwaltet die Diskettenkapazität wesentlich effizienter. Jede Datei belegt eine oder mehrere komplette Spuren. Umfangreichere Dateien müssen dabei nicht unbedingt auf aufeinanderfolgenden Spuren untergebracht sein. Soll z.B. eine Datei vergrößert werden, so sucht JADOS automatisch nach freien Bereichen (z.B. einen durch Löschen einer Datei freigeworden Bereich) und speichert die Dateiergänzung darauf ab. Auf diese Weise können alle auf der Diskette vorhandenen Spuren auch wirklich ausgenutzt werden.
Im Inhaltsverzeichnis ist eingetragen, auf welcher Spur eine Datei beginnt. Da größere Dateien an verschiedenen Stellen auf der Diskette gespeichert sein können, wird in einer sogenannten *Spurtabelle* festgehalten, wo es weitergeht. Und zwar werden hier für jede Spur einer Datei jeweils die Nachfolgerspur (also in Richtung auf das Dateiende) und die Vorgängerspur (also in Richtung auf den Dateianfang) eingetragen.

Die Spurtabelle ist folgendermaßen aufgebaut:

Byte	Inhalt	Zusatzbedeutung
0 - 1	Vorgängerspur	$E5E5 -> freie Spur
		$FFFF -> kein Vorgänger
		4 - 159 -> Nummer der Vorgängerspur
2 - 3	Nachfolgerspur	$E5E5 -> freie Spur
		$FFFF -> kein Nachfolger
		4 - 159 -> Nummer der Nachfolgerspur

Der *Vorteil* dieses Verfahrens liegt darin, daß vorhandene Dateien problemlos wachsen können und daß die durch Löschen einer Datei frei gewordenen Spuren wieder beliebig zur Verfügung stehen. Ein kleiner *Nachteil* besteht darin, daß das Laden und Speichern etwas länger dauern kann.

4.5) DATEISTEUERBLOCK

Jede Datei wird von einem Dateisteuerblock, im folgenden mit FCB (engl. file control block) abgekürzt, verwaltet.
Es ist dies ein Datensatz, der beim öffnen oder Anlegen einer Datei gebildet wird, und beim Schließen der Datei wieder unwirksam wird.

Der Datensatz ist folgendermaßen aufgebaut:

Byte	Inhalt	Zusatzbedeutung
0 - 1	Laufwerksnummer	1 oder 2
2 - 9	Dateiname	
10 - 11	Intern reserviert	
12 - 14	Namenserweiterung bzw. Dateityp	
15	Intern reserviert	
16 - 17	Startspur der Datei	
18 - 19	Endesektor, d.h. letzter Sektor der letzten Dateispur	
20 - 21	Für spätere Erweiterungen reserviert	
22 - 25	Ladeadresse	
26 - 27	Dateilänge,gemessen in Sektoren	
28 - 31	Für spätere Erweiterungen reserviert	
32 - 33	Sektor des Inhaltsverzeichnisses	
34 - 35	Startbyte im Sektor des Inhaltsverzeichnisses	
36 - 37	Dateistatus	0 -> geschlossen
		1 -> geöffnet, kein Directory-Update
		2 -> geöffnet, mit Directory-Update
38 - 39	Aktuelle Dateispur	
40 - 41	Aktueller Dateisektor	
42 - 43	Letzte Dateispur	
44 - 47	Aktuelle Lade- bzw. Speicheradresse	

5) D I E K O M M A N D O S

5.1) EINFÜHRUNG

5.1.1) Der Kommandointerpreter

Der Benutzer kommuniziert mit JADOS über Kommandos, die in der Regel über die Tastatur eingegeben werden. Die Auswertung besorgt der im JADOS eingebaute Kommandointerpreter. Dieser wird sofort nach dem Start von JADOS aktiv und zeigt mit einem Prompt die Bereitschaft des Systems an, Kommandos entgegenzunehmen.

Das Prompt besteht aus der Nummer des aktuellen Laufwerks und dem Zeichen ">". Nach dem Start erscheint also

```
1>
```

Viele Kommandos benötigen Parameter - in der Regel Dateinamen - um ihre Aufgabe zu erfüllen. Diese Parameter können vom erfahrenen Anwender direkt hinter dem Kommando, abgesetzt durch ein Leerzeichen, eingegeben werden.
Wer sich die Anzahl und Bedeutung der Parameter aber nicht merken will, braucht nur die Kommandos eingeben. JADOS fragt dann alle benötigten Parameter mit erklärenden Hinweisen ab.

JADOS unterscheidet zwischen internen und externen Kommandos. Die Internen sind fest in JADOS eingebaut. Die externen Kommandos sind Namen von Programmdateien, die nach dem Aufruf von der Diskette geladen und dann automatisch abgearbeitet werden.

5.1.2) Dateinamen

Ein Dateiname besteht aus drei Komponenten, der Laufwerksnummer, dem eigentlichen Namen und dem Dateityp. Zwischen der Laufwerksnummer und dem Namen muß das Trennzeichen ":" und zwischen dem Namen und dem Dateityp das Trennzeichen "." stehen.

Wird die Laufwerksnummer nicht angegeben, dann nimmt JADOS an, daß das aktuelle Laufwerk verwendet werden soll. Das aktuelle Laufwerk wird durch das Systemprompt angezeigt. Mögliche Laufwerksnummern sind 1:,2:,3:oder 4:.

Der Name umfaßt 1 bis 8 Zeichen, wobei kein Unterschied zwischen Klein- und Großbuchstaben besteht. Die meisten Sonderzeichen werden zwar akzeptiert, sollten aber nicht verwendet werden.

Der Dateityp umfaßt 0 bis 3 Zeichen, ansonsten gilt das für den Namen Gesagte.
Die Dateitypen .68K, .COM und .BAT sind von JADOS mit festen Bedeutungen belegt.

Beispiele für gültige Dateinamen:

```
FORMAT.68K
2:FORMAT.68K
FORMAT
```

5.1.3) Zeileneditor

Bei der Eingabe von Kommandos und Dateinamen über die Tastatur sind folgende Möglichkeiten gegeben:

* Mit der <- bzw. der DEL-Taste kann das Zeichen links vom Cursor gelöscht werden. Der Cursor rückt dabei um eine Stelle nach links.

* Mit der RETURN-Taste wird die Eingabe ordnungsgemäß beendet.

* Mit Ctrl C kann die Eingabe abgebrochen werden. Geschieht dies während der Eingabe eines Kommandos, so wird ein "Warmstart" ausgeführt, d.h. JADOS initialisiert sich neu. Während der Eingabe eines Dateinamens bewirkt Ctrl C den Abbruch des Kommandos.

* In JADOS gibt es Zeichenpuffer für Kommandos und Dateinamen. Mit Ctrl J kann der komplette Pufferinhalt abgerufen werden. Wenn man z.B. ein vorher eingegebenes Kommandos wiederholen möchte, so braucht man nur Ctrl J und RETURN eingeben. Mit Ctrl F kann der Puffer zeichenweise abgerufen werden.

* Mit Ctrl P kann eine Hardcopy des Textinhalts des Bildschirms auf den Drucker ausgegeben werden.

* Mit Ctrl G wird ein Zeichen im Puffer gelöscht.

* Mit Ctrl V wird ein Leerzeichen im Puffer eingefügt.

5.1.4) Ladeadresse

Der Begriff "Ladeadresse" wird im folgenden häufiger vorkommen. Darunter wird eine Speicheradresse verstanden, ab der ein Text oder ein Programm von Diskette geladen wird.
Bei Systemen ohne Bankbootkarte ist die Ladeadresse für Programme gleich $C000 und bei Systemen mit Bankbootkarte gleich $0400.

5.2) DIE INTERNEN KOMMANDOS

5.2.1) Übersicht

In der folgenden alphabetischen Übersicht sind alle internen JADOS-Kommandos aufgelistet. Dabei wird folgende Nomenklatur verwendet:

[...] -> Die in den eckigen Klammern angegebenen Parameter sind optionell, d.h. man kann sie beim Kommandoaufruf mit eingegeben.

Kommando	Parameter	Bedeutung
Ctrl A	--	Kaltstart
Ctrl C	--	Warmstart
1:	--	Umschaltung auf Laufwerk 1
2:	--	Umschaltung auf Laufwerk 2
3:	--	Umschaltung auf Laufwerk 3
4:	--	Umschaltung auf Laufwerk 4
ASS	[dateiname]	Assemblierung
BELL	--	Glockenton
CLS	--	Bildschirm löschen
COPY	[datei1 datei2]	Datei kopieren
DIR	[dateiname]	Inhaltsverzeichnis anzeigen
DIRP	[dateiname]	Inhaltsverzeichnis ausdrucken
EDIT	[dateiname]	Textdatei eingeben oder ändern
ERA	[dateiname]	Datei löschen
FF	--	Seitenvorschub am Drucker
INST	[dateiname]	Programm resident installieren
PAUSE	[beliebiger Text]	Stapelverarbeitung anhalten
PRINT	[dateiname]	Textdatei ausdrucken
REM	[beliebiger Text]	Kommentar in Stapelverarbeitung
REN	[datei-alt datei-neu]	Datei umbenennen
TABLE	--	Residente Programme auflisten
TLOAD	[adresse dateiname]	Textdateien auf bestimmte Adresse laden
TSAVE	[dateiname adresse]	Text ab bestimmter Adresse abspeichern
TYPE	[dateiname]	Textdatei auf Bildschirm anzeigen

5.2.2) Kaltstart mit Ctrl A

Mit der Tastenkombination Ctrl A wird ein Kaltstart durchgeführt. Alle residenten Programme sind dann verloren.

5.2.3) Warmstart mit Ctrl C

Mit der Tastenkombination Ctrl C wird ein sogenannter Warmstart des Betriebssystems veranlaßt. JADOS reagiert darauf wie folgt:

- Der Bildschirm wird gelöscht.
- Das Copyright, die Versionsnummer, die CPU und der freie Benutzerspeicher werden angezeigt.
- Das Laufwerk 1 wird zum aktuellen Laufwerk
- Alle Tastaturpuffer werden gelöscht.
- Der amerikanische Zeichensatz wird eingeschaltet.

Die residenten Programme bleiben erhalten.

5.2.4) Laufwerkumschaltung mit 1: ... 4:

Das Systemprompt zeigt die Nummer des aktuellen Laufwerks an. JADOS bietet vier Kommandos zur Umschaltung des aktuellen Laufwerks an:

```
1:   -->  Laufwerk 1 wird aktuell
2:   -->  Laufwerk 2 wird aktuell.
3:   -->  Laufwerk 3 wird aktuell.
4:   -->  Laufwerk 4 wird aktuell.
```

5.2.5) ASS

ASS ist die Abkürzung für *assemble*. Damit kann ein Quellprogramm in ablauffähigen Maschinencode übersetzt werden. JADOS verwendet den im Grundprogramm enthaltenen Assembler.

Die Arbeitsweise des Kommandos hängt davon ab, ob beim Aufruf ein Parameter übergeben wird.

5.2.5.1) Mit Parameter

Diese Kommandoart wird hauptsächlich bei kleinen Programmen verwendet, die nur aus einem Quellprogrammodul bestehen.

Der übergebene Parameter ist der Name der Quellprogrammdatei. Diese Textdatei wird auf die Ladeadresse für Programme plus $4000 geladen. Damit bleiben 16 KByte Speicherplatz für den Maschinencode. Nach einem fehlerfreien Assemblerlauf wird der Maschinencode unter dem eingegebenen Namen abgespeichert, wobei als Dateityp automatisch .68K eingesetzt wird.

5.2.5.2) Ohne Parameter

Diese Kommandoart bietet dem Anwender wesentlich mehr Möglichkeiten.

Nach der Kommandoeingabe erscheint folgender Text:

```
A S S E M B L E R

!!  KEINE ORG- ODER OFFSETANWEISUNG BENUTZEN  !!
!!  STARTADRESSE IST KONFIGURATIONSABHÄNGIG   !!
!!  NORMAL: $C000   -  MIT BOOTKARTE: $0400   !!
```

Beim Assembliervorgang müssen drei Speicherbereiche verwaltet werden, die Symboltabelle, der erzeugte Maschinencode und der Quelltext. Der Maschinencode wird ab Adresse $C000 bzw. $0400 abgelegt, daher dürfen *keine ORG- oder OFFSET-Anweisungen* benutzt werden. Die Startadresse der Symboltabelle liegt hinter dem Grundprogramm und umfaßt in der Normalkonfiguration etwa 13 KByte und in der Bootkartenkonfiguration je nach Speicherausbau zwischen 5 KByte und 29 KByte. Dabei entsprechen 13 KByte etwa 700 Symbolen !!

ACHTUNG !!
Wenn die Symboltabelle in der Normalkonfiguration die Adresse $C000 überschreitet, dann wird sie durch den abgelegten Maschinencode zerstört.

Nach dem Hilfetext wird der Benutzer aufgefordert, die Startadresse des Quelltextes einzugeben:

```
QUELLPROGRAMME LADEN
WELCHE STARTADRESSE ? (DEFAULT = $00XXXX )
```

Der Quelltext wird hinter den Maschinencode geladen. Bei einer Fehleingabe oder wenn kein Wert eingegeben wurde, dann wählt JADOS automatisch den DEFAULT - Wert. Dieser liegt 16 KByte hinter dem Start des Maschinen codes, also $10000 bzw. $4400.

Die folgende übersicht zeigt die Lage von Symboltabelle, Maschinencode und Quelltext in Abhängigkeit von der Konfiguration:

	NORMALKONFIGURATION	BOOTKARTENKONFIGURATION
Symboltabelle	$00BA4E	$0EBA4E
Maschinencode	$00C000	$000400
Quelltext (Default)	$010000	$004400

Nach Eingabe der Startadresse wird der Benutzer aufgefordert, das Quellprogramm zu laden. Dabei können mehrere Quellmodule in Folge geladen werden, die dann als Ganzes assembliert werden. Damit kann der Programmierer kleine Quellmodule wie Bausteine zu einem großen Programm verbinden.
Das Laden der Quellmodule geschieht menügeführt:

```
1 - TEXT LADEN
2 - automatisch
9 - beenden
```

Mit 9 kann der Ladevorgang abgebrochen werden.

Mit 1 oder Return wird eine Textdatei geladen:

```
WELCHE TEXTDATEI LADEN ?
```

Der Benutzer kann hier den Namen des Quellprogrammes eingeben. Nach jedem Ladevorgang erscheint obiges Menü erneut.

Mit 2 kann eine automatische Antwortdatei geladen werden, die die Namen aller Quellmodule des Programms enthält. Diese Antwortdatei wird mit dem normalen Editor erstellt und benötigt keinen speziellen Dateityp. Beim Ladevorgang werden alle Namen und zusätzlich die Endadressen angezeigt. Diese Funktion ist besonders bei großen Programmen mit vielen Quellmodulen sehr hilfreich.

Danach erscheint ein Ausgabemenü, in dem der Bediener die Art der Ausgabe des Assemblers festlegen kann:

```
A U S G A B E
 1 - NUR FEHLER
 2 - Bildschirm
 3 - Drucker
 9 - beenden
```

Mit 1 oder Return wird festgelegt, daß nur Fehlermeldungen auf dem Bildschirm erscheinen.
Mit 2 wird festgelegt, daß das komplette Listing einschließlich Fehlermeldungen auf dem Bildschirm ausgegeben wird. Das Listing kann mit CTRL S angehalten und mit CTRL Q fortgesetzt werden.
Mit 3 wird festgelegt, daß das komplette Listing einschließlich Fehlermeldungen auf dem Drucker ausgegeben wird.
Mit 9 kann das gesamte Kommando abgebrochen werden.

Nun erfolgt der Assembliervorgang. Falls Fehler aufgetreten sind, wird das Maschinenprogramm nicht gespeichert. Bei fehlerfreien Maschinenprogrammen wird der Bediener aufgefordert:

```
NAMEN DER PROGRAMMDATEI (.68K) EINGEBEN:
```

Es muß nur der Name eingegeben werden. Der Dateityp .68K wird automatisch vergeben. Dateien dieses Typs können dann im Kommandointerpreter von JADOS durch Eingeben des Namens geladen und gestartet werden.

5.2.6) BELL

Hiermit kann ein Glockenton erzeugt werden.

Das Kommando ist für den Stapelverarbeitungsbetrieb gedacht, um den Benutzer auf Eingaben oder dergleichen aufmerksam zu machen.

Voraussetzung dafür sind die Baugruppe **SOUND** und ein Lautsprecher. Die Baugruppe wird vom Grundprogramm unterstützt.

Leider ergeben sich Schwierigkeiten mit dem Drucker, wenn die SOUND-Baugruppe auf die vom Grundprogramm vorgesehene Adresse $FFFFFF40 eingestellt wird. Deshalb adressiert JADOS diese Baugruppe auf der Adresse $FFFFFF50.

Die genannten I/O-Adressen beziehen sich auf das System mit 68008-CPU. Die entsprechenden Adressen für die anderen CPUs der 68000-Familie werden von JADOS automatisch angepaßt.

5.2.7) CLS

Mit diesem Kommando werden der Bildschirm gelöscht und der 80-Zeichen-Modus eingestellt. Dies ist auch für Stapelverarbeitungsbetrieb interessant.

5.2.8) COPY

COPY dient zum Kopieren von Dateien. Das Kommando kann mit oder ohne Parameter aufgerufen werden.

a) mit Parameter:

1. Parameter: Name der Quelldatei
2. Parameter: Name der Zieldatei

b) ohne Parameter:

Nach der Kommandoeingabe wird gefragt:

```
WELCHE DATEI KOPIEREN ?
```

Hier ist der Name der zu kopierenden Datei einzugeben. Danach wird der Name der Kopie erfragt:

```
NAME DER KOPIE ?
```

Nach erfolgreichem Kopiervorgang meldet JADOS:

```
DATEI KOPIERT .
```

5.2.9) DIR / DIRP

Mit DIR bzw. DIRP wird das Inhaltsverzeichnis einer Diskette ausgegeben.

DIR --> Anzeige auf dem Bildschirm.
DIRP --> Ausdruck auf dem Drucker.

Es werden jeweils der Name, der Dateityp und die Größe einer Datei angegeben. Am Schluß werden die Anzahl der aufgelisteten Dateien, deren Gesamtgröße und die Anzahl der freien Spuren und Sektoren der Diskette angegeben. Das folgende Beispiel zeigt einen Ausschnitt aus der Entwicklungsdiskette für dieses Handbuch:

```
                INHALTSVERZEICHNIS VON LAUFWERK 2
--NAME-- TYP  LAENGE     --NAME-- TYP  LAENGE     --NAME-- TYP  LAENGE
TRAP0005 JHB  18         TRAP0613 JHB  14         TRAP1421 JHB  15
TRAP2230 JHB  10         TRAP3141 JHB  14         TRAP4250 JHB  16
COMASS   JHB  15         COMREST  JHB  1
  ANZAHL DATEIEN:  8       ANZAHL SEKTOREN:  103
  FREIE SEKTOREN:  375     FREIE SPUREN:     53
```

a) ohne Parameter

Es wird das komplette Inhaltsverzeichnis der im aktuellen Laufwerk eingelegten Diskette ausgegeben.

b) mit Parameter

Hier können das Laufwerk sowie bestimmte Dateien selektiert werden. Der Parameter wird wie ein Dateiname eingegeben. Alle auf den Namen passende Dateien werden dann aufgelistet. Es können auch sogenannte Jokerzeichen (im EDV-Jargon "Wildcards" genannt) eingegeben werden. Dabei ist das Zeichen "?" ein Joker für einen Buchstaben und das Zeichen "*" ein Joker für alle Buchstaben.

Beispiele:

```
DIR *.68K        --> Alle Dateien vom Typ .68K
DIR DOS*.*       --> Alle Dateinamen, die mit "DOS" anfangen
DIR BRIEF.TXT    --> Nur die Datei "BRIEF.TXT"
DIR 2:*.?68      --> Alle Dateien auf Laufwerk 2, deren Typname
                     mit "68" endet
```

5.2.10) EDIT

EDIT dient zum Eingeben und/oder Ändern von Textdateien. Dabei wird der im Grundprogramm enthaltene bildschirmorientierte Editor von R.-D. Klein verwendet.

Die Arbeitsweise von EDIT hängt davon ab, ob ein Dateiname als Parameter übergeben wird.

5.2.10.1) Mit Parameter

Falls die Datei bereits existiert, wird sie automatisch von der Diskette geladen. Der Benutzer kann dann den Text ändern. Nach dem Editieren wird der Text automatisch unter dem gleichen Namen abgespeichert.

Falls die Datei noch nicht existiert, kann der Benutzer einen neuen Text eingeben. Nach der Eingabe wird der Text automatisch unter dem eingegebenen Namen abgespeichert.

5.2.10.2) Ohne Parameter

Nach der Eingabe von EDIT erscheint folgendes Menü:

```
T E X T E D I T O R
 1 - neuen Text erstellen
 2 - ALTEN TEXT ÄNDERN
 9 - beenden
```

Mit dem *Menüpunkt 1* kann ein völlig neuer Text erstellt werden. Nach der Eingabe verläßt der Bediener den Editor wie gewohnt mit CTRL KX und erhält folgende Aufforderung:

```
NAMEN DER NEUEN TEXTDATEI EINGEBEN:
```

Jetzt kann dem erstellten Text ein Name zugewiesen werden, unter dem er auf der Diskette abgespeichert wird. Mit CTRL C kann das Abspeichern verhindert werden.

Mit dem *Menüpunkt 2* oder Return kann ein bereits erstellter Text geändert werden. Dazu wird der Bediener gefragt:

```
WELCHE TEXTDATEI ÄNDERN ?
```

Der Bediener kann nun den Namen der zu ändernden Textdatei eingeben. Diese wird von der Diskette geladen und kann dann mit dem Editor bearbeitet werden. Nach Verlassen des Editors mit CTRL KX wird dem Benutzer der Name der ursprünglichen Textdatei angezeigt und dann gefragt:

```
NAMEN DER GEÄNDERTEN TEXTDATEI EINGEBEN:
```

Gibt der Benutzer den gleichen Namen ein, so wird die ursprüngliche Textdatei durch die geänderte ersetzt. Gibt er einen anderen Namen ein, so wird der geänderte Text unter diesem Namen abgespeichert, während die ursprüngliche Textdatei erhalten bleibt. Gibt er Ctrl C ein, so wird nichts abgespeichert.

Mit dem *Menüpunkt 9* kann das Kommando beendet werden.

ACHTUNG ! !

Folgende Hinweise sind unbedingt zu beachten:

- Der Editor von R.-D. Klein kann nur Texte bis 64 KByte bearbeiten. Wird diese Grenze überschritten, so produziert der Editor nur noch "Schrott". Die Reaktionen von JADOS können dann auch fehlerhaft sein.

- Der Benutzer ist dafür verantwortlich, daß der Text nicht größer wird als der freie Benutzerspeicher. Keine Probleme gibt es, wenn der freie Benutzerspeicher größer als 64KByte ist.

- Falls bei der Eingabe des Namens der geänderten Textdatei ein Fehler auftritt, so wird der Text nicht gespeichert. Hier kann sich der Benutzer mit dem TSAVE-Kommando helfen, wobei die Frage nach der Ladeadresse mit RETURN zu quittieren ist.

5.2.11) ERA

Mit ERA kann eine Datei gelöscht werden. Die freigewordenen Bereiche auf der Diskette können von anderen Dateien wieder belegt werden. Seit Version 2.0 wird der Löschvorgang sehr schnell ausgeführt.

a) mit Parameter

Die als Parameter übergebene Datei wird gelöscht.

b) ohne Parameter

Der Benutzer wird nach dem Namen der zu löschenden Datei gefragt:

```
WELCHE DATEI LöSCHEN ?
```

Der Löschvorgang wird anschließend bestätigt:

```
DATEI GELöSCHT
```

5.2.12) FF

Hiermit kann ein Seitenvorschub auf dem Drucker ausgelöst werden. Dies ist vor allem beim Ausdruck mehrerer Textdateien über eine Stapelverarbeitung nützlich.

5.2.13) INST

Mit INST wird ein Programm resident in den Hauptspeicher geladen. Der dafür nötige Speicherplatz geht auf Kosten des freien Benutzerspeichers. Es können maximal 20 Programme resident installiert werden. Dateien, die nicht vom Typ 68K oder COM sind, werden zurückgewiesen.

a) mit Parameter

Das als Parameter angegebene Programm wird installiert.

b) ohne Parameter

JADOS fragt:

WELCHES PROGRAMM INSTALLIEREN?

Dieses Programm wird dann resident geladen.

Hinweis: Ein residentes Programm wird "ganz normal" im Kommandointerpreter durch Eingabe des Namens aufgerufen.

5.2.14) PAUSE

Mit diesem Kommando kann der Ablauf einer Stapelverarbeitung angehalten werden. JADOS gibt dabei folgende Meldung aus:

Wenn fertig, bitte eine Taste betätigen ...

JADOS wartet dann, bis der Benutzer eine beliebige Taste betätigt.

5.2.15) PRINT

Mit PRINT kann eine Textdatei auf den Drucker ausgegeben werden. Dabei wird automatisch nach jeweils 65 Zeilen ein Seitenvorschub erzeugt.

Dem Kommando kann der Name der Textdatei als Parameter übergeben werden. Bei Aufruf des Kommandos ohne Parameter fragt JADOS:

```
WELCHE TEXTDATEI AUSGEBEN ?
```

Der Name der Textdatei kann dann eingegeben werden.
Die Ausgabe kann mit Ctrl C abgebrochen werden.

5.2.16) REM

Dieses Kommando dient zum Einbringen von Kommentarzeilen in Stapelverarbeitungsdateien.

5.2.17) REN

REN dient als Abkürzung für *rename*. Mit diesem Kommando kann der Name einer schon existierenden Datei geändert werden. Die nötigen Parameter können wieder wahlweise beim Aufruf mitgegeben werden oder werden von JADOS erfragt.

a) mit Parameter:

1. Parameter: Alter Name
2. Parameter: Neuer Name

b) ohne Parameter:

Nach der Kommandoeingabe fragt JADOS:

```
WELCHE DATEI UMBENENNEN ?
```

Hier wird der alte Name eingegeben. Danach erscheint die Frage nach dem neuen Namen:

```
NEUER NAME ?
```

Nach der Umbenennung bestätigt JADOS den Vorgang

```
DATEI UMBENANNT .
```

5.2.18) TABLE

TABLE dient zum Auflisten aller residenten Programme. Die Ausgabe ist wie folgt (die Adressen und Namen sind nur Beispiele):

```
VERZEICHNIS DER RESIDENTEN PROGRAMME
--NAME--        ADRESSE         TYP
DISKCOPY        $29000          68K
GP              $28400          COM
DISASS          $25400          68K
```

5.2.19) TLOAD

TLOAD dient zum Laden einer oder mehrerer Textdateien in den Hauptspeicher zwecks weiterer Bearbeitung mit dem Grundprogramm. Eine mögliche Anwendung wäre das Laden eines Quellprogramms, um es im Grundprogramm mit der Option DEBUG EIN zu assemblieren und dann eine Einzelschrittverarbeitung durchzuführen.

a) mit Parameter

1. Parameter: Speicheradresse, ab der der Text geladen werden soll
2. Parameter: Name der Textdatei

b) ohne Parameter

Dies bietet dem Benutzer mehr Möglichkeiten als der Aufruf mit Parametern. Zuerst wird die Startadresse erfragt

```
WELCHE STARTADRESSE ? (DEFAULT = $00XXXX)
```

Die angegebene Adresse kann mit RETURN bestätigt werden oder aber man gibt die gewünschte Adresse ein.

Nach Eingabe der Startadresse wird der Benutzer aufgefordert, den Quelltext zu laden. Dabei können mehrere Quelltexte in Folge geladen werden, die dann als ein Text zusammengebunden werden. Damit kann man große Texte erzeugen.
Das Laden der Quelltexte geschieht menügeführt:

```
1 - TEXT LADEN
2 - automatisch
9 - beenden
```

Mit 9 kann der Ladevorgang abgebrochen werden.

Mit 1 oder Return wird eine Textdatei geladen:

```
WELCHE TEXTDATEI LADEN ?
```

Der Benutzer kann hier den Namen der Textdatei eingeben. Nach jedem Ladevorgang erscheint obiges Menü erneut.

Mit 2 kann eine automatische Antwortdatei geladen werden, die die Namen aller Quelltexte enthält. Diese Antwortdatei wird mit dem normalen Editor erstellt und benötigt keinen speziellen Dateityp.
Beim Ladevorgang werden alle Namen und zusätzlich die Endadressen angezeigt.

5.2.20) TSAVE

TSAVE dient zum Abspeichern von Texten aus dem Hauptspeicher auf Diskette.

a) mit Parameter

1. Parameter: Name, unter dem der Text gespeichert werden soll
2. Parameter: Speicheradresse, ab der der Text gespeichert werden soll

b) ohne Parameter

Der Benutzer wird zuerst nach dem Namen der Textdatei gefragt:

```
NAMEN DER TEXTDATEI EINGEBEN:
```

Nach der Eingabe des Namens erscheint die Frage nach der Startadresse:

```
WELCHE STARTADRESSE ? (DEFAULT = $00XXXX)
```

5.2.21) TYPE

Mit TYPE kann eine Textdatei auf den Bildschirm ausgegeben werden.

Dem Kommando kann der Name der Textdatei als Parameter mitgegeben werden. Bei Aufruf des Kommandos ohne Parameter fragt JADOS:

```
WELCHE TEXTDATEI AUSGEBEN ?
```

Der Name der Textdatei kann dann eingegeben werden.
Mit Ctrl C kann die Ausgabe abgebrochen werden.

5.3) DIE EXTERNEN KOMMANDOS

Bei den externen Kommandos handelt es sich um Namen von Dateien. Dabei muß zwischen Programmdateien (Typ .68K oder .COM) und Kommandodateien (Stapelverarbeitung ; Typ .BAT) unterschieden werden.

Daß es sich um ein externes Kommando handelt, erkennt JADOS am Dateityp, d.h. an der Namenserweiterung .68K, .COM oder .BAT. Diese Namenserweiterung braucht der Benutzer jedoch nicht einzugeben. Angenommen, der Benutzer hat die Programmdatei DRUCKE.68K entwickelt und abgespeichert und möchte diese nun aufrufen. Er gibt dazu lediglich DRUCKE ein. JADOS sucht nun nach der Datei DRUCKE.68K, lädt diese und startet sie automatisch. Selbstverständlich darf der Benutzer einem externen Kommando nicht den Namen eines internen Kommandos geben, wie z.B. PRINT. JADOS würde sogleich das interne Kommando starten.

Die Reihenfolge, in der JADOS externe Kommandos sucht, ist:
SPEICHERRESIDENT --> .68K --> .COM --> .BAT

JADOS sucht also zuerst nach einem residenten Programm, danach nach einer Datei vom Typ .68K, danach vom Typ .COM und zuletzt vom Typ .BAT. Findet JADOS nichts, dann meldet es:

FALSCHES KOMMANDO .

Selbstverständlich darf der Benutzer auch keine gleichen Namen für externe Kommandos benutzen, z.B. DRUCKE.68K und DRUCKE.BAT, denn JADOS würde als Kommando nur DRUCKE.68K finden und ausführen.

5.3.1) Programmdateien

Programmdateien bestehen aus ausführbarem Maschinencode. JADOS erkennt solche Dateien ausschließlich am Dateityp.

5.3.1.1) Programme vom Typ .68K

Diese Programmdateien werden von JADOS auf die Ladeadresse **$C000** bzw. **$0400** geladen und automatisch gestartet. Die Startadresse entspricht dabei der Ladeadresse.

Falls ein Programm den sogenannten Bibliotheksvorspann besitzt, so wird die Startadresse vom entsprechenden Bibliothekseintrag bestimmt. Es werden jedoch nur solche Programme gestartet, die als relokativ gekennzeichnet sind, die also nicht an eine feste Startadresse gebunden sind.

Jedes Programm muß mit einem "RTS"-Befehl beendet werden, damit JADOS wieder die Programmkontrolle erhält.

5.3.1.2) Programme vom Typ .COM

Seit Version 2.0 steht der Programmtyp .COM zur Verfügung. Diese Programmdateien werden aber nicht auf die Ladeadresse geladen, sondern möglichst weit hinten im Benutzerspeicher. Dadurch bleibt der Speicher ab der Ladeadresse frei.

.COM-Dateien müssen voll relokativ sein, um fehlerfrei ablaufen zu können. Die Ladeadresse liegt nämlich nicht fest, sondern wird von JADOS automatisch gewählt. Variablenspeicher darf nur auf dem Stack oder per ds.b-Anweisung eingerichtet werden!

Der Dateityp .COM wird durch Umbenennen einer .68K-Datei erzeugt.

5.3.2) Residente Programme

Seit Version 2.1 besteht die Möglichkeit, bis zu 20 Programme speicherresident zu halten (siehe Kommando INST). Hierbei spielt der Programmtyp eine wichtige Rolle. Programme vom Typ COM werden direkt auf der Adresse gestartet, ab der sie installiert sind. Programme vom Typ 68K werden vorher auf die Ladeadresse kopiert. Die Programme müssen dabei kleiner als 256 KByte sein, da sonst der Kopierbefehl nicht ordnungsgemäß funktioniert.

5.3.3) Stapelverarbeitung

Bisher konnten Kommandos nur über die Tastatur eingegeben werden. Seit Version 2.0 ist nun ein Mechanismus verfügbar, der eine Reihe von Kommandos, die in einer Textdatei abgelegt sind, automatisch abarbeitet. Immer wiederkehrende Kommandofolgen, z.B. bei einer Datensicherung, können auf diese Weise bequem und sicher ablaufen.

Dies entspricht der SUBMIT-Funktion des bekannten Betriebssystems CP/M 68K.

Die Kommandodateien sind ganz normale Textdateien, die mit dem Editor erstellt werden. Sie müssen vom Dateityp **.BAT** sein, damit JADOS sie als Kommandodateien erkennen kann.

Jede Textzeile darf ein JADOS-Kommando enthalten. Dabei können sämtliche interne und externe Kommandos verwendet werden. Lediglich die Verwendung von Kommandodateien ist eingeschränkt, da eine Verschachtelung von Kommandodateien nicht zulässig ist. Nur das letzte Kommando einer Stapelverarbeitungsdatei darf eine Kommandodatei sein ! Auf diese Weise können Stapelverarbeitungsdateien verkettet werden.

Die Kommandos werden der Reihe nach abgearbeitet, bis das letzte Kommando bearbeitet worden ist. Der Benutzer kann die Stapelverarbeitung unterbrechen, indem er eine beliebige Taste betätigt. JADOS fragt dann:

```
STAPELVERARBEITUNG ABBRECHEN (J/N) ?
```

Durch Eingabe von "J" wird die Stapelverarbeitung komplett abgebrochen, ansonsten wird sie korrekt fortgeführt.

Eine Kommandodatei darf Kommandos mit formalen Parametern enthalten. Diese formalen Parameter heißen %1, %2, %3 und %4. Beim Aufruf einer Kommandodatei können bis zu 4 Parameter mitgegeben werden, die dann die formalen Parameter ersetzen. Dadurch wird eine hohe Flexibilität erreicht.
Ein Beispiel soll dies verdeutlichen:

Eine Textdatei (z.B. mit dem Namen dateixyz) soll viermal ausgedruckt werden. Man könnte nun viermal das Kommando PRINT dateixyz über die Tastatur eingeben und dazwischen jedesmal den Drucker auf eine neue Seite einstellen.
Eine Kommandodatei erfüllt dies wesentlich eleganter. Um die Kommandodatei allgemeingültig zu halten, d.h. auf Textdateien unterschiedlichen Namens anwendbar zu machen, muß die Kommandodatei mit einem formalen Parameter arbeiten. Die folgende Kommandosequenz erfüllt alle Anforderungen:

```
FF
PRINT %1
FF
PRINT %1
FF
PRINT %1
FF
PRINT %1
FF
FF
```

Angenommen, die Kommandodatei habe den Namen DRUCKE4.BAT. Wird nun DRUCKE4 dateixyz eingegeben, so ersetzt JADOS den formalen Parameter %1 durch den aktuellen Parameter dateixyz.

Soll nun eine Textdatei mit dem Namen dateiabc viermal ausgedruckt werden, so muß man lediglich DRUCKE4 dateiabc eingeben. Die Kommandodatei konnte dabei unverändert übernommen werden. Damit hat sich der kleine Arbeitsaufwand, die Kommandodatei zu erstellen, bereits gelohnt!!

Die folgende Kommandodatei, mit deren Hilfe eine Datensicherung aller Dateien, die zur Entwicklung von JADOS dienten, ausgeführt wird, möge abschließend als Beispiel einer nützlichen Anwendung der Stapelverarbeitung dienen:

```
bell
rem Quelle in 1: und-Ziel in 2:
pause
copy 1:dosmain.asm 2:dosmain.210
copy 1:dostabs.asm 2:dostabs.210
copy 1:dostool.asm 2:dostool.210
copy 1:dosfile.asm 2:dosfile.210
copy 1:dosbatch.asm 2:dosbatch.210
copy 1:dosgrund.asm 2:dosgrund.210
copy 1:doslink.ans 2:doslink.ans
copy 1:dossave.bat 2:dossave.bat
copy 1:dosinit.asm 2:dosinit.210
copy 1:dosinit.68k 2:dosinit.68k
copy 1:jados210.68k 2:jados210.68k
bell
rem Ende des Backups
```

Nach Aufruf der Kommandodatei "DOSSAVE" ertönt ein akustisches Signal. Danach fordert JADOS den Benutzer auf, die Quellendiskette in Laufwerk 1 und die Zieldiskette in Laufwerk 2 einzulegen sowie die Leertaste zu betätigen. Danach werden alle Dateien, die zur JADOS-Entwicklung dienten von Laufwerk 1 auf Laufwerk 2 kopiert.

Der 1. große Vorteil von Kommandodateien ist, daß sie immer wieder verwendet werden können. Eine Kommandofolge muß nur einmal erstellt werden. Dadurch erspart man sehr viel "Tipparbeit". Der 2. große Vorteil ist, daß die Kommandos automatisch abgearbeitet werden. In dieser Zeit braucht der Benutzer nicht unbedingt am Computer zu sitzen. Der 3. große Vorteil ist, daß man sich über die Kommandoabfolge keine Gedanken mehr machen muß. Dies hat man ja bei der Erstellung der Kommandodatei bereits getan.

Ein weiteres Beispiel soll zeigen, wie eine Kommandodatei vorteilhaft zur Entwicklung kleiner Programme eingesetzt werden kann:

```
EDIT %1.ASM
ASS %1.ASM
%1
```

6) BESCHREIBUNG DER UNTERPROGRAMME

In JADOS sind 63 *Unterprogramme* für den Programmierer verfügbar.
Bei geladenem Betriebssystem können Anwendungsprogramme diese Unterprogramme mit dem Befehl **TRAP #6** aufrufen.
In **Register D7** muß die Nummer des Unterprogramms übergeben werden.

ACHTUNG !!
Der Variablenbereich von JADOS bezieht sich auf das Register A6. Dieses wird beim Aufruf eines JADOS-Unterprogramms automatisch auf den richtigen Wert gebracht. Der ursprüngliche Wert von A6 geht dabei verloren.

6.1) NUMERISCHE ÜBERSICHT

Nummer	Name	Register
0	getleng	A0 -> D1
1	getname	A0, A1 -> D0.B
2	getstadd	A2 -> A0
3	lese	D2, A1 -> D0.B
4	loadtext	A2 -> D0.B
5	motoroff	
6	response	
7	schreibe	A0
8	strgcomp	A0, A1 -> D0.B
9	tload	A0, A1 -> D0.B, A0
10	tsave	A0, A1 -> D0.B
11	uppercas	A0
12	wrblank	D1
13	wrint	D0
14	close	A1
15	copyfile	A0, A1, A2 -> D0.B
16	create	A1 -> D0.B
17	erase	A1 -> D0.B
18	fillfcb	A0, A1 -> D0.B
19	open	A1 -> D0.B
20	readrec	A1 -> D0.B
21	rename	A1, A2 -> D0.B
22	setdta	A0, A1
23	writerec	A1 -> D0.B
24	getversi	-> D0.L
25	getparm	-> A0
26	hardcopy	
27	bell	
28	beep	D1.L, D2.L
29	errnoise	
30	sound	A0
31	inport	D1.L -> D0.B
32	outport	D0.B, D1.L
33	movelbin	D1, A1, A2
34	moveltxt	A1, A2
35	moverbin	D1, A1, A2

Nummer	Name	Register
36	wrtcmd	A0
37	gtretcod	-> D0.L
38	stretcod	D0.L
39	moveline	D2, A0, A2 -> D0.B, A0
40	wraddr	D0.L
41	ci	-> D0.B
42	getparm1	-> A0
43	getparm2	-> A0
44	fileload	A0, A1 -> D0.B
45	filesave	D1, A0, A1 -> D0.B
46	getparm3	-> A0
47	getparm4	-> A0
48	loadpart	A0, A1 -> D0.B, D2
49	savepart	D2, A0, A1 -> D0.B
50	catalog	A0
51	floppy	D1, D2, D3, D4, A0
52	drivecode	D4 -> D4
53	getdrive	-> D0
54	setdrive	D0
55	gttraptb	-> A0
56	blockread	D2, A0, A1 -> D0.B
57	blockwrite	D2, A0, A1 -> D0.B
58	getladdr	-> A0
59	skipchar	D1.B, A0 -> A0
60	delete	D1, A0
61	insert	D2, A0, A1
62	ramtop	-> A0

6.2) ALPHABETISCHE ÜBERSICHT

Name	Nummer	Kurzbeschreibung
beep	28	Erzeugt Ton mit wählbarer Frequenz und Dauer (Soundbaugruppe !)
bell	27	Erzeugt Glockenton (Soundbaugruppe !)
blockread	56	Liest Folge von Sektoren von Disk
blockwrite	57	Schreibt Folge von Sektoren auf Disk
catalog	50	Schreibt Disketteninhaltsverzeichnis über die CO2-Schnittstelle
ci	41	Liest ASCII-Wert von Tastatur; Ctrl P erzeugt Text-Hardcopy vom Bildschirm
close	14	Schließt eine geöffnete Datei
copyfile	15	Kopiert eine Datei
create	16	Legt eine Datei an
delete	60	Löscht Anzahl Zeichen in einem String
drivecode	52	Wandelt Laufwerksnummer in binären Code
erase	17	Löscht eine Datei
errnoise	29	Erzeugt Fehlerton (Soundbaugruppe !)
fileload	44	Liest komplette Datei von Diskette
filesave	45	Schreibt komplette Datei auf Diskette
fillfcb	18	Trägt Dateinamen in Dateisteuerblock ein
floppy	51	Verbesserte "FLOPPY"-Routine
getdrive	53	Liefert Nummer des aktuellen Laufwerks
getladdr	58	Liefert Ladeadresse
getleng	0	Ermittelt Länge eines Textes
getname	1	Erfragt Dateinamen und trägt diesen in den Dateisteuerblock ein

Name	Nummer	Kurzbeschreibung
getparm	25	Liefert Parameter in der Kommandozeile
getparm1	42	Liefert 1. Parameter der Kommandozeile
getparm2	43	Liefert 2. Parameter der Kommandozeile
getparm3	46	Liefert 3. Parameter der Kommandozeile
getparm4	47	Liefert 4. Parameter der Kommandozeile
getstadd	2	Erfragt Startadresse zum Laden/Speichern in/von Hauptspeicher von/auf Diskette
getversi	24	Informiert über die Version von JADOS
gtretcod	37	Liest 4-Byte Usercode
gttraptb	55	Liefert Adresstabelle der Unterprogramme
hardcopy	26	Druckt Bildschirminhalt (nur Text)
inport	31	Liest Byte von Port (gilt für alle CPUs der 68000-Familie)
insert	61	Fügt Teilstring in String ein
lese	3	Liest einen Textstring von der Tastatur
loadpart	48	Liest Teil einer Datei von Diskette
loadtext	4	Lädt menügesteuert Texte von Diskette in den Hauptspeicher, wobei die Texte hintereinander abgelegt werden
motoroff	5	Schaltet Diskettenmotoren ab
movelbin	33	Verschiebt binären Speicherbereich von links
moveline	39	Verschiebt Textzeile in Speicherpuffer
moveltxt	34	Verschiebt Textbereich von links
moverbin	35	Verschiebt binären Speicherbereich von rechts
open	19	öffnet eine Datei
outport	32	Gibt Byte auf Port aus (gilt für alle CPUs der 68000-Familie)
ramtop	62	Liefert Ende des freien Benutzerspeichers
readrec	20	Liest einen Dateisektor
rename	21	Benennt eine Datei um
response	6	Wartet auf Tastendruck
savepart	49	Schreibt Teil einer Datei auf Diskette
schreibe	7	Schreibt einen Text über die CO2-Schnittstelle
setdrive	54	Verändert Nummer des aktuellen Laufwerks
setdta	22	Verändert die aktuelle Lade/Speicher - Transfer-Adresse
skipchar	59	überspringt Zeichen in einem String
sound	30	Steuert Soundbaugruppe auf Adresse $FFFFFF50 an
stretcod	38	Setzt 4-Byte Usercode ab
strgcomp	8	Vergleicht zwei Textstrings
tload	9	Lädt eine komplette Textdatei
tsave	10	Speichert eine komplette Textdatei
uppercas	11	Wandelt alle Kleinbuchstaben in einem Text in Großbuchstaben
wraddr	40	Schreibt eine 24-Bit Adresse hexadezimal über die CO2-Schnittstelle
wrblank	12	Schreibt eine Anzahl Leerzeichen über die CO2-Schnittstelle
wrint	13	Schreibt eine 16-Bit Dezimalzahl über die CO2-Schnittstelle
writerec	23	Schreibt einen Dateisektor
wrtcmd	36	Schreibt JADOS-Kommando in Kommandopuffer

6.3) BESCHREIBUNG DER EINZELNEN UNTERPROGRAMME

6.3.1) getleng 0

Hiermit kann die Länge eines Textes ermittelt werden.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.l	x	Startadresse des Textes
	D1.w	x	Anzahl der Zeichen des Textes

Hinweis: Das Textende muß durch eine binäre Null gekennzeichnet sein. Die Länge des Textes sollte nicht größer als 65535 Zeichen sein, da D1 nur als Wortregister verwendet wird. Es wird nur Register D1 verändert.

Beispiel :
```
* TEXTLÄNGENERMITTLUNG
*
START:
  LEA     TEXT(PC),A0
  MOVE    #0,D7
  TRAP    #6
  RTS

TEXT: DC.B 'Hallo',0
END
```

Im Beispiel wird die Länge des Textes 'Hallo' ermittelt. Das Ergebnis 5 steht nach dem TRAP-Aufruf in Register D1.

6.3.2) getname 1

Hiermit steht dem Programmierer eine Funktion zur Bedienerführung zur Verfügung. Es wird ein Text ausgegeben, der vom Bediener verlangt, einen Dateinamen einzugeben. Der Bediener kann nun einen bis zu 14 Zeichen umfassenden Namen eingeben. Die Eingabe kann mit CTRL A oder CTRL C abgebrochen werden. Ein solcher Abbruch wird im Register D0 mit dem Wert $FF angezeigt. Nach korrekter Eingabe werden alle Klein- in Großbuchstaben umgewandelt. Anschließend wird ein Dateisteuerblock initialisiert und mit dem eingegebenen Dateinamen ausgefüllt. Falls hierbei ein Fehler festgestellt wird, so wird ein Fehlertext ausgegeben und der Wert $FF in das Register D0 geschrieben.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.l	x	Adresse des Aufforderungstextes
	A1.l	x	Adresse eines Dateisteuerblocks
	D0.b	x	Fehlercode: 00 - kein Fehler $FF - Name falsch oder Abbruch mit CTRL A oder CTRL C

Hinweis: Alle Ausgaben werden über die CO2-Schnittstelle getätigt. Es wird nur Register D0 verändert.

```
Beispiel:   * DATEINAMEN ERFRAGEN
            *
            START:
              LEA       TEXT(PC),A0
              LEA       USERFCB(PC),A1
              MOVE      #1,D7
              TRAP      #6
              RTS

            TEXT:     DC.B 'Bitte Dateinamen eingeben: ',0
                      DS 0
            USERFCB:  DS.B 48
            END
```

Im Beispiel wird der Text 'Bitte Dateinamen eingeben: ' ausgegeben. Der Cursor blinkt neben dem Text, und der Benutzer kann nun einen Dateinamen eingeben. Der Dateisteuerblock *USERFCB* enthält nach dem TRAP-Aufruf den eingegebenen Dateinamen.

6.3.3) getstadd 2

Hiermit steht dem Programmierer eine Funktion zur Bedienerführung zur Verfügung. Es wird ein Text ausgegeben, der vom Bediener verlangt, eine Startadresse einzugeben. Der Bediener kann nun einen bis zu 10 Zeichen umfassenden Wert eingeben. Die Eingabe kann mit CTRL A oder CTRL C abgebrochen werden. Ein solcher Abbruch oder wenn der Wert kleiner ist als ein vorgegebener Wert führt dazu, daß als Startadresse der vorgegebene Wert zurückgegeben wird.

Parameter:

Register	Eing./Ausg.	Bedeutung
A0.1	x	Eingegebene Startadresse
A2.1	x	Vorgegebene Startadresse

Hinweis: Alle Ausgaben werden über die CO2-Schnittstelle getätigt. Es wird nur Register A0 verändert.

```
Beispiel:   * STARTADRESSE ERFRAGEN
            *
            START:
              LEA       $C000,A2
              MOVE      #2,D7
              TRAP      #6
              RTS
            END
```

Im Beispiel wird der Text 'WELCHE STARTADRESSE ? (DEFAULT = $00C000) ausgegeben.
Der Cursor blinkt neben dem Text, und der Benutzer kann nun einen Wert eingeben. Dies kann eine Zahl oder ein Symbol sein. Falls die die Eingabe nicht abgebrochen wurde und der Wert nicht kleiner als $C000 ist, steht nach dem TRAP-Aufruf im Register A0 der eingegebene Wert.

6.3.4) lese 3

Diese Funktion dient zum Einlesen eines Textes. Mit der RETURN-Taste wird die Eingabe korrekt abgeschlossen. Mit der <- Taste oder der DEL-Taste kann die Eingabe korrigiert werden.
Die Tastenkombination Ctrl J gibt den Textpufferinhalt ab der Cursorposition aus.
Mit CTRL A oder CTRL C kann die Eingabe abgebrochen werden. Dies erzeugt einen Fehlercode.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A1.l	x	Adresse des Textpuffers
	D2.w	x	Zeichenzahl
	D0.b	x	Zustandscode:
			00 - Eingabe ok
			01 - Abbruch mit CTRL A
			03 - Abbruch mit CTRL C

Hinweis: Alle Ausgaben werden über die CO2-Schnittstelle getätigt. Alle Steuerzeichen ($00 - $1F) werden ignoriert außer CTRL A, CTRL C, CTRL H (<-), CTRL M (RETURN) und CTRL J (LINEFEED) sowie CTRL F, CTRL G und CTRL V.
Es wird nur Register D0 verändert.

Die Funktion CTRL J steht seit Version 2.0 zur Verfügung. Die Funktionen CTRL F, CTRL G und CTRL V stehen seit Version 2.1 zur Verfügung.

Beispiel:

```
* TEXTZEILE EINLESEN
*
START:
  LEA      TEXTBUF(PC),A1
  MOVE     #10,D2
  MOVE     #3,D7
  TRAP     #6
  RTS

TEXTBUF:   DS.B 11
END
```

Im Beispiel kann ein bis zu 10 Zeichen umfassender Text eingegeben werden. Dieser Text steht nach dem TRAP-Aufruf in *TEXTBUF*.
Dabei wird das Textende automatisch mit $00 markiert. Deshalb muß der Textbuffer auch um ein Zeichen größer sein als die vorgegebene Textlänge.

6.3.5) loadtext 4

Dies ist eine sehr komplexe Funktion zum benutzergeführten Laden von Textmodulen von Diskette in den Speicher. Dies geschieht menügeführt. Die Startadresse des ersten Textes kann vom Benutzer eingegeben werden, wobei der Programmierer eine Mindeststartadresse vorgegeben kann.

* Seit Version 2.0 steht auch ein automatischer Textlademechanismus zur
* Verfügung. Hierbei wird der Name einer Textdatei eingeben, die
* wiederum die Namen einer oder mehrerer Textdateien enthält, die
* leztendlich geladen werden sollen.
* Neu ist auch die Ausgabe der jeweiligen Endadresse.

Parameter:

Register	Eing./Ausg.		Bedeutung
A2.l	x		Mindeststartadresse
D0.b		x	Zustandscode:
			00 - Texte geladen
			01 - Funktion abgebrochen, ohne daß ein Text geladen wurde
			$FF - Fehler beim Laden von Diskette (z.B. Datei nicht gefunden)

Hinweis: Alle Ausgaben werden über die CO2-Schnittstelle getätigt. Der geladene Text kann im Grundprogramm bearbeitet werden. Z.B. können mehrere Programmodule zu einem Quellprogramm zusammengefaßt und komplett assembliert werden. Dies geht besonders bequem mit der neuen automatischen Textladefunktion !!
Es werden keine Register außer D0 verändert.

Beispiel:

```
* MEHRERE TEXTDATEIEN EINLESEN UND DRUCKEN
*
START:
  LEA       $C000,A2
  MOVE      #4,D7     * loadtext
  TRAP      #6
  CMP.B     #0,D0
  BNE.S     FEHLER

  JSR       @GETSTX
  MOVEA.L   D0,A0
  JSR       @LST
  MOVE      #7,D7     * schreibe
  TRAP      #6
FEHLER:
  RTS
```

Im Beispiel können mehrere Textdateien von Diskette in den Hauptspeicher geladen werden. Nach dem Laden wird der Fehlercode abgefragt. Ist dieser Null, so wird der komplette Text über die Druckerschnittstelle ausgegeben.

6.3.6) motoroff 5

Hiermit können die Laufwerksmotoren abgeschaltet werden.

Parameter: keine !

Beispiel :
```
* MOTOREN ABSCHALTEN
*
START:
  MOVE     #5,D7
  TRAP     #6
  RTS
END
```

Im Beispiel werden die Motoren der angeschlossenen Laufwerke abgeschaltet.

6.3.7) response 6

Es erscheint der Text 'LEERTASTE DRÜCKEN'. Das Unterprogramm wartet nun in einer Schleife, bis die Leertaste gedrückt wird.

Parameter: keine !

Hinweis: Die Textausgabe erfolgt über die CO2-Schittstelle.
Es werden keine Register verändert.

Beispiel:
```
* WARTESCHLEIFE
*
START:
  MOVE    #6,D7
  TRAP    #6
  RTS
END
```

Im Beispiel wird der Text 'LEERTASTE DRÜCKEN ' ausgegeben. Erst nach Drücken der Leertaste wird die Warteschleife verlassen.

6.3.8) schreibe 7

Mit dieser Funktion kann ein Text über die CO2-Schnittstelle ausgegeben werden. Das Textende muß durch eine binäre Null gekennzeichnet werden. Das erste Zeichen wird auf die aktuelle Cursorposition gesetzt.

Parameter:

Register	Eing./Ausg.	Bedeutung
A0.l	x	Adresse des Textes

Hinweis: Alle Ausgaben erfolgen über die CO2-Schnittstelle.
Es werden keine Register verändert.

Beispiel:
```
* TEXT AUSGEBEN
*
START:
  LEA     TEXT(PC),A0
  MOVE    #7,D7
  TRAP    #6
  RTS

TEXT:   DC.B 'Hallo',0
END
```

Im Beispiel wird der Text 'Hallo' ausgegeben.

6.3.9) strgcomp 8

Mit dieser Funktion können zwei Textstrings auf Gleichheit überprüft werden. Beide Texte müssen mit einer binären Null abgeschlossen sein.

Parameter:

Register	Eing./Ausg.	Bedeutung
A0.l	x	Adresse des 1. Textstrings
A1.l	x	Adresse des 2. Textstrings
D0.b	x	Ergebnis: 00 - Strings sind gleich
		$FF - Strings sind ungleich

Hinweis: Es wird nur Register D0 verändert.

Beispiel:

```
* STRINGS VERGLEICHEN
*
START:
  LEA     STRING1(PC),A0
  LEA     STRING2(PC),A1
  MOVE    #8,D7
  TRAP    #6
  CMP.B   #0,D0
  BNE.S   UNGLEICH
GLEICH:
  LEA     GLTEXT(PC),A0
  BRA.S   AUSGABE
UNGLEICH:
  LEA     UGLTEXT(PC),A0
AUSGABE:
  MOVE    #7,D7
  TRAP    #6
 RTS
STRING1:  DC.B 'Hallo',0
STRING2:  DC.B 'HALLO',0
GLTEXT:   DC.B 'Strings sind gleich',13,10,0
          DS 0
UGLTEXT:  DC.B 'Strings sind ungleich',13,10,0
END
```

Im Beispiel werden die beiden Strings STRING1 und STRING2 auf Gleichheit geprüft. Bei Gleichheit wird der Text 'Strings sind gleich' ausgegeben, bei Ungleichheit der Text 'Strings sind ungleich'.

6.3.10) tload 9

Mit dieser Funktion kann eine Textdatei von Diskette in den Haupt-Speicher geladen werden. Wenn die Datei nicht existiert, oder der Hauptspeicher zu klein ist, so wird der Ladevorgang abgebrochen und eine Fehlermeldung ausgegeben. Die Ladeadresse wird in A0 vorgegeben. Nach dem Laden zeigt A0 auf das erste Byte nach dem Text. Die Variable "stxtxt" aus dem Grundprogramm zeigt auf die Ladeadresse.

* Seit Version 2.0 werden komplette Dateien erheblich schneller geladen.
* Applikationen, die "tload" benutzen, profitieren automatisch von
* diesem Geschwindigkeitsgewinn !

Parameter:

Register	Eing.	Ausg.	Bedeutung
A0.l	x	x	Ladeadesse. Nach dem Laden zeigt A0 auf erstes Byte nach dem Text.
A1.l	x		Adresse eines Dateisteuerblocks
D0.b		x	Zustandscode: 00 - Ladevorgang ok $FF - Abbruch wegen Fehler

Hinweis: Alle Ausgaben efolgen über die CO2-Schnittstelle.
Im Dateisteuerblock muß bereits der Dateiname eingetragen sein. Das Eintragen des Namens sollte nur mit der Funktion "fillfcb" erfolgen.
Es werden die Register A0 und D0 geändert.

Beispiel:

```
* TEXTDATEI EINLESEN
*
START:
* DATEINAMEN IN DATEISTEUERBLOCK EINTRAGEN
  LEA       DATEINAM(PC),A0
  LEA       USERFCB(PC),A1
  MOVE      #18,D7
  TRAP      #6
  CMP.B     #0,D0
  BNE.S     FEHLER
* DATEI VON DISKETTE LESEN
  LEA       $C000,A0
  MOVE      #9,D7
  TRAP      #6
  BRA.S     ENDE
FEHLER:
  LEA       FEHLTEXT(PC),A0
  MOVE      #7,D7
  TRAP      #6
ENDE:
  RTS

FEHLTEXT:  DC.B 'Dateiname falsch',13,10,0
           DS 0
DATEINAM:  DC.B '1:BEDIEN.TXT',0
           DS 0
USERFCB:   DS.B 48
END
```

Im Beispiel wird zunächst ein Dateiname in den Dateisteuerblock "USERFCB" eingetragen. Falls dabei ein Fehler auftritt, wird der Text 'Dateiname falsch' ausgegeben, sonst wird die Datei ab Adresse $C000 geladen. Wichtig ist, daß der Dateiname in Großbuchstaben vorliegt. Dies kann durch Aufruf des Unterprogramms "uppercase" (Nr. 11) vor dem Aufruf von "fillfcb" sichergestellt werden.

6.3.11) tsave 10

Dies ist das Gegenstück zu "tload". Hiermit kann ein Text auf Diskette gespeichert werden. Dazu muß bereits der Dateiname in den Dateisteuerblock eingetragen sein.

* Seit Version 2.0 werden komplette Dateien erheblich schneller gespei-
* chert. Applikationen, die "tsave" benutzen, profitieren automatisch
* von diesem Geschwindigkeitsgewinn !

Parameter:

Register	Eing./Ausg.		Bedeutung
A0.l	x		Startadresse des Textes
A1.l	x		Adresse des Dateisteuerblocks
D0.b		x	Zustandscode:
			00 - Texte gespeichert
			05 - Diskette voll
			06 - Inhaltsverzeichnis voll
			$FF - Fehler beim Diskettenzugriff

Hinweis: Alle Ausgaben erfolgen über die CO2-Schnittstelle. Falls die Diskette oder das Inhaltsverzeichnis der Diskette voll sind, wird eine entsprechende Fehlermeldung ausgegeben. Es wird nur Register D0 verändert.

Beispiel:

```
* TEXT AUF DISKETTE SPEICHERN
*
START:
* DATEINAMEN IN DATEISTEUERBLOCK EINTRAGEN
  LEA       DATEINAM(PC),A0
  LEA       USERFCB(PC),A1
  MOVE      #18,D7
  TRAP      #6
  CMP.B     #0,D0
  BNE.S     FEHLER
* DATEI AUF DISKETTE SPEICHERN
  LEA       $C000,A0
  MOVE      #10,D7
  TRAP      #6
  BRA.S     ENDE
FEHLER:
  LEA       FEHLTEXT(PC),A0
  MOVE      #7,D7
  TRAP      #6
ENDE:
  RTS

FEHLTEXT:  DC.B 'Dateiname falsch',13,10,0
           DS 0
DATEINAM:  DC.B '1:BEDIEN.TXT',0
           DS 0
USERFCB:   DS.B 48
END
```

Im Beispiel wird zunächst ein Dateiname in den Dateisteuerblock "USERFCB" eingetragen. Falls dabei ein Fehler auftritt, wird der Text 'Dateiname falsch' ausgegeben, sonst wird der ab Adresse $C000 im RAM stehende Text auf die Datei 'BEDIEN.TXT' gespeichert.

6.3.12) uppercas 11

Hiermit werden die Kleinbuchstaben 'a' bis 'z' ,die in einem Text vorkommen, in Großbuchstaben umgewandelt.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.l	x	Adresse des Textes

Hinweis: Der Text muß durch eine binäre Null abgeschlossen sein.

Beispiel:

```
* KLEIN- IN GROßBUCHSTABEN WANDELN
*
START:
  LEA       TEXT(PC),A0
  MOVE      #11,D7
  TRAP      #6
  MOVE      #7,D7
  TRAP      #6
  RTS
TEXT:       DC.B 'Dies ist ein Text',0
END
```

Im Beispiel wird der Text 'Dies ist ein Text' umgewandelt in 'DIES IST EIN TEXT'. Der umgewandelte Text wird anschließend ausgegeben.

6.3.13) wrblank 12

Hiermit können eine Anzahl Leerzeichen über die CO2-Schnittstelle ausgegeben werden.

Parameter:	Register	Eing./Ausg.	Bedeutung
	D1.w	x	Anzahl Leerzeichen

Beispiel:

```
* LEERZEICHEN AUSGEBEN
*
START:
  LEA       TEXT1(PC),A0
  MOVE      #7,D7
  TRAP      #6
  MOVE      #3,D1
  MOVE      #12,D7
  TRAP      #6
  LEA       TEXT2(PC),A0
  MOVE      #7,D7
  TRAP      #6
  RTS
TEXT1:      DC.B 'Name: ',0
TEXT2:      DC.B 'NDR-Klein-Computer',0
END
```

Im Beispiel wird der Text 'Name: NDR-Klein-Computer' ausgegeben.

6.3.14) wrint 13

Hiermit kann eine positive 16-bit Zahl dezimal ausgegeben werden.
Dabei werden immer 6 Zeichen linksbündig ausgegeben.

Parameter:

Register	Eing./Ausg.	Bedeutung
D0.w	x	16-bit Zahl

Beispiel:

```
* ZAHL AUSGEBEN
*
START:
  MOVE    #135,D0
  MOVE    #13,D7
  TRAP    #6
  RTS
END
```

Im Beispiel wird die Zahl '135 ' ausgegeben.

6.3.15) close 14

Diese Funktion schließt eine geöffnete Datei. Nach Schreiboperationen wird das Inhaltsverzeichnis aktualisiert.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A1.l	x	Adresse des Dateisteuerblocks

Hinweis: Bei einer bereits geschlossenen Datei wird die Funktion ignoriert.
Es werden keine Register verändert.

Beispiel:
```
* DATEI SCHLIEßEN
*
START:
  LEA      USERFCB(PC),A1
  MOVE     #14,D7
  TRAP     #6
  RTS
USERFCB:   DS.B 48
END
```

Im Beispiel wird der Dateisteuerblock "USERFCB" geschlossen.

6.3.16) copyfile 15

Mit dieser Funktion kann eine beliebige Datei kopiert werden.
Bei Kopien auf demselben Laufwerk ist kein Diskettenwechsel möglich.
Fehlerzustände werden über das Register D0 gemeldet.

ACHTUNG !!

* Seit Version 2.0 ist zu beachten, daß "copyfile" sehr viel schneller
* geworden ist (Faktor 10 bis 15). Dies wurde zum Teil dadurch
* erreicht, daß "copyfile" einen Teil des Benutzerspeichers belegt.
* Die Anfangsadresse muß dabei durch das Register A0 festgelegt
* werden. Die Endadresse wird durch den Stackpointer bestimmt. Dies
* ist von Applikationsprogrammen zu berücksichtigen.
* Neu ist seit Version 2.0 auch, daß existierende Dateien nicht mehr
* vorher gelöscht werden müssen.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.l	x	Anfangsadresse des Kopierspeichers. Die Endadresse wird durch A7 bestimmt.
	A1.l	x	Adresse des Dateisteuerblocks der zu kopierenden Datei
	A2.l	x	Adresse des Dateisteuerblocks der Kopie
	D0.b	x	Fehlerzustand: 00 - Kopie ok 02 - Datei nicht gefunden 05 - Diskette voll 06 - Inhaltsverzeichnis voll $FF - Sonstiger Diskettenfehler

Hinweis: Es wird nur Register D0 geändert.
Das Register A0 ist neu hinzugekommen.

Beispiel:

```
* DATEI KOPIEREN
*
START:
* DATEISTEUERBLOCK DER ZIELDATEI ANLEGEN
  LEA       DATEI2(PC),A0
  LEA       FCB2(PC),A1
  MOVE      #18,D7
  TRAP      #6
  CMP.B     #0,D0
  BNE.S     FEHLER

  MOVEA.L   A1,A2
* DATEISTEUERBLOCK DER QUELLDATEI ANLEGEN
  LEA       DATEI1(PC),A0
  LEA       FCB1(PC),A1
  MOVE      #18,D7
  TRAP      #6
  CMP.B     #0,D0
  BNE.S     FEHLER

* KOPIERFUNKTION AUFRUFEN
  LEA       LADDR(PC),A0
  MOVE      #15,D7
  TRAP      #6
  CMP.B     #0,D0
  BNE.S     FEHLER
  RTS

FEHLER:
  RTS

DATEI1:     DC.B 'QUELLE',0
DATEI2:     DC.B 'ZIEL',0
            DS 0
FCB1:       DS.B 48
FCB2:       DS.B 48
LADDR:

END
```

Im Beispiel werden zuerst die Dateisteuerblöcke angelegt. Wird dabei ein Fehler festgestellt, so wird zum Lable FEHLER: verzweigt. Hier könnte z.B. eine entsprechende Fehlermeldung erfolgen. Nach dem Anlegen der Dateisteuerblöcke wird die Ladeadresse festgelegt und anschließend die Kopierfunktion aufgerufen.

6.3.17) create 16

Mit dieser Funktion kann eine Datei in das Inhaltsverzeichnis eingetragen werden. Der zugehörige Dateisteuerblock erhält den Status "offen", d.h. auf die Datei kann zugegriffen werden.

Parameter:	Register	Eing./Ausg.		Bedeutung
	A1.l	x		Adresse des Dateisteuerblocks
	D0.b		x	Fehlerzustand:
				00 - Datei angelegt
				05 - Diskette voll
				06 - Inhaltsverzeichnis voll
				$FF - Sonstiger Diskettenfehler

Hinweis: Eine bereits existierende Datei wird zum Lesen und Schreiben geöffnet, siehe Funktion "open" (Nr. 19). Nur das Register D0 wird verändert.

Beispiel:

```
* DATEI ANLEGEN
*
START:
  LEA      USERFCB(PC),A1
  MOVE     #16,D7
  TRAP     #6
 RTS
USERFCB:   DS.B 48
END
```

Im Beispiel wird eine Datei im Inhaltsverzeichnis eingetragen. Der Dateisteuerblock muß bereits den Dateinamen enthalten, siehe auch "fillfcb".

6.3.18) erase 17

Mit dieser Funktion kann eine Datei gelöscht werden. Dazu wird ein entsprechender Eintrag im Inhaltsverzeichnis abgelegt. In der Spurtabelle werden alle zur Datei gehörigen Spuren als frei markiert. Eine einmal gelöschte Datei läßt sich in der Regel nicht mehr rekonstruieren, da die Spurzuordnung ebenfalls gelöscht wird.

* Seit Version 2.0 wird diese Funktion erheblich schneller ausgeführt.

ACHTUNG !!

* Vor Version 2.0 wurde der Löschvorgang nur auf dem aktuellen Laufwerk korrekt ausgeführt. Seit Version 2.0 ist dieser Fehler behoben.

Parameter:	Register	Eing./Ausg.		Bedeutung
	A1.l	x		Adresse des Dateisteuerblocks
	D0.b		x	Zustandscode:
				00 - Löschvorgang ausgeführt
				02 - Datei nicht gefunden
				$FF - Sonstiger Diskettenfehler

Hinweis: Es wird nur Register D0 verändert.

Beispiel:
```
* DATEI LöSCHEN
*
START:
* DATEINAMEN IN DATEISTEUERBLOCK EINTRAGEN
  LEA       DATEINAM(PC),A0
  LEA       USERFCB(PC),A1
  MOVE      #18,D7
  TRAP      #6
  CMP.B     #0,D0
  BNE.S     FEHLER
* DATEI VON DISKETTE LöSCHEN
  MOVE      #17,D7
  TRAP      #6
  BRA.S     ENDE
FEHLER:
  LEA       FEHLTEXT(PC),A0
  MOVE      #7,D7
  TRAP      #6
ENDE:
  RTS

FEHLTEXT:  DC.B 'Dateiname falsch',13,10,0
DATEINAM:  DC.B '1:BEDIEN.TXT',0
           DS 0
USERFCB:   DS.B 48
END
```

Im Beispiel wird zunächst ein Dateiname in den Dateisteuerblock "USERFCB" eingetragen. Falls dabei ein Fehler auftritt, wird der Text 'Dateiname falsch' ausgegeben, sonst wird die Datei gelöscht.

6.3.19) fillfcb 18

Hiermit können Laufwerk, Dateiname und Dateityp in einen Dateisteuerblock eingetragen werden.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.l	x	Adresse eines Textes mit dem Dateinamen
	A1.l	x	Adresse des Dateisteuerblocks
	D0.b	x	Zustandscode: 00 - Name eingetragen $FF - Name weist Fehler auf

Hinweis: Dateiname und Dateityp müssen als Großbuchstaben vorliegen. Dies kann mit der Funktion "uppercas" erreicht werden. Falls Kleinbuchstaben vorkommen, so kann diese Datei vom Kommandointerpreter nicht erreicht werden !
Es wird nur Register D0 verändert.

```
Beispiel:   * DATEINAMEN IN DATEISTEUERBLOCK EINTRAGEN
            *
            START:
              LEA       DATEINAM(PC),A0
              LEA       USERFCB(PC),A1
              MOVE      #18,D7
              TRAP      #6
              RTS
            DATEINAM:   DC.B '1:BEDIEN.TXT',0
                        DS 0
            USERFCB:    DS.B 48
            END
```

Im Beispiel wird die Datei "BEDIEN.TXT" in den Dateisteuerblock "USERFCB" eingetragen.

6.3.20) open 19

Diese Funktion öffnet eine vorhandene Datei zum Lesen und Schreiben.

Parameter: Register	Eing./Ausg.		Bedeutung
A1.l	x		Adresse des Dateisteuerblocks
D0.b		x	Fehlerzustand:
			$00 - Datei gefunden und geöffnet
			$FF - Datei nicht gefunden

Hinweis: Es wird nur Register D0 verändert.

```
Beispiel:   * DATEISTEUERBLOCK ANLEGEN
            *
            START:
              LEA       DATEINAM(PC),A0
              LEA       USERFCB(PC),A1
              MOVE      #18,D7
              TRAP      #6
              CMP.B     #0,D0
              BNE.S     FEHLER

            * DATEI ÖFFNEN
              MOVE      #19,D7
              TRAP      #6
              RTS
            FEHLER:
              RTS

            DATEINAM:   DC.B '1:BEDIEN.TXT',0
                        DS 0
            USERFCB:    DS.B 48
            END
```

Im Beispiel wird die Datei "BEDIEN.TXT" in den Dateisteuerblock "USERFCB" eingetragen. Falls dabei kein Fehler festgestellt wird, wird die Datei geöffnet.

6.3.21) readrec 20

Hiermit kann ein Sektor einer geöffneten Datei gelesen werden.
Die Ladeadresse wird nach erfolgtem Lesevorgang um die Sektorgröße erhöht.
Das Dateiende oder der Überlauf des Benutzerspeichers werden zuverlässig erkannt und mit einem Fehlercode gemeldet.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A1.l	x	Adresse des Dateisteuerblocks
	D0.b	x	Fehlerzustand:
			00 - Sektor gelesen
			01 - Dateiende erreicht
			99 - Speicherende erreicht
			$FF - Sonstiger Diskettenfehler

```
Beispiel:   * EINEN DATEISEKTOR LESEN
            *
            START:
              MOVE      #20,D7
              TRAP      #6
              RTS
            END
```

Im Beispiel wird ein Sektor einer bereits geöffneten Datei gelesen.

6.3.22) rename 21

Hiermit kann eine Datei umbenannt werden.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A1.l	x	Adresse des Dateisteuerblocks
	A2.l	x	Dateisteuerblock mit neuem Namen
	D0.b	x	Fehlerzustand:
			00 - Umbenennung durchgeführt
			02 - Datei nicht gefunden
			03 - Zielname existiert
			04 - Laufwerke unterschiedlich
			$FF - Sonstiger Fehler

Hinweis: Es wird nur Register D0 zerstört.

```
Beispiel:   * DATEI UMBENENNEN
            *
            START:
            * DATEISTEUERBLOCK DER ZIELDATEI ANLEGEN
              LEA       DATEI2(PC),A0
              LEA       FCB2(PC),A1
              MOVE      #18,D7
              TRAP      #6
              CMP.B     #0,D0
              BNE.S     FEHLER
```

```
  MOVEA.L  A1,A2
* DATEISTEUERBLOCK DER QUELLDATEI ANLEGEN
  LEA      DATEI1(PC),A0
  LEA      FCB1(PC),A1
  MOVE     #18,D7
  TRAP     #6
  CMP.B    #0,D0
  BNE.S    FEHLER

* UMBENENNUNGSFUNKTION AUFRUFEN
  MOVE     #21,D7
  TRAP     #6
  CMP.B    #0,D0
  BNE.S    FEHLER
  RTS

FEHLER:
  RTS

DATEI1:    DC.B 'QUELLE',0
DATEI2:    DC.B 'ZIEL',0
           DS 0
FCB1:      DS.B 48
FCB2:      DS.B 48
END
```

Im Beispiel werden zuerst die Dateisteuerblöcke angelegt. Wird dabei ein Fehler festgestellt, so wird zum Lable FEHLER: verzweigt. Hier könnte z.B. eine entsprechende Fehlermeldung erfolgen. Nach dem Anlegen der Dateisteuerblöcke wird die Umbenennungsfunktion aufgerufen.

6.3.23) setdta 22

Diese Funktion verändert die aktuelle Speichertransferadresse bei Lade- und Speicheroperationen.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.l	x	Neue Speichertransferadresse
	A1.l	x	Adresse des Dateisteuerblocks

Hinweis: Es werden keine Register verändert.

Beispiel:
```
* SPEICHERTRANSFERADRESSE ÄNDERN
*
START:
  MOVEA.L  #$10000,A0
  MOVE     #22,D7
  TRAP     #6
  RTS
END
```

Im Beispiel wird die Adresse $10000 zur neuen Speichertransferadresse. Dabei wird vorausgesetzt, daß das Register A1 auf einen Dateisteuerblock zeigt.

6.3.24) writerec 23

Hiermit kann ein Sektor auf eine geöffnete Datei geschrieben werden. Die Speichertransferadresse wird nach dem Schreibvorgang um die Sektorgröße erhöht.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A1.l	x	Adresse des Dateisteuerblocks
	D0.b	x	Fehlerzustand:
			00 - Sektor geschrieben
			05 - Diskette voll
			$FF - Sonstiger Diskettenfehler

Beispiel:
```
* EINEN DATEISEKTOR SCHREIBEN
*
START:
  MOVE     #23,D7
  TRAP     #6
  RTS
END
```

Im Beispiel wird ein Sektor auf eine geöffnete Datei geschrieben.

6.3.25) getversi 24

Mit dieser Funktion kann die Versionsnummer des JADOS erfragt werden.

Parameter:	Register	Eing./Ausg.	Bedeutung
	D0.1	x	Versionsnummer als ASCII-Zeichen

Hinweis: Die Versionsnummer wird als Folge von 4 ASCII-Zeichen zurückgegeben.
Es wird nur Register D0 verändert.

Beispiel:

```
* VERSIONSNUMMER ERFRAGEN
*
START:
  MOVE      #24,D7
  TRAP      #6

* VERSIONSNUMMER AUSGEBEN
  LEA       TEXT(PC),A0
  MOVE.L    D0,(A0)
  MOVE      #0,4(A0)
  MOVE      #7,D7
  TRAP      #6
  RTS
TEXT:       DS.B 6
END
```

Im Beispiel wird zuerst die Versionsnummer erfragt. Anschließend wird das Ergebnis ausgegeben.

6.3.26) getparm 25

Diese Funktion liefert einen Zeiger auf die Parameter, die beim Aufruf einer Programm- oder Stapeldatei eingegeben wurden.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.1	x	Zeiger auf Programmparameter

Hinweis: Es wird nur das Register A0 verändert.
Der Parametertext wird intern in Großbuchstaben gewandelt.

Beispiel:

```
* PROGRAMMPARAMETER AUSGEBEN
*
START:
  MOVE      #25,D7
  TRAP      #6

* VERSIONSNUMMER AUSGEBEN
  MOVE      #7,D7
  TRAP      #6
  RTS
END
```

Das obige Programm sei als Datei "SHOWPARM.68K" abgespeichert. Wenn nun im Kommandointerpreter "SHOWPARM JADOS IST PRIMA" eingegeben wird, so gibt das Programm "JADOS IST PRIMA" auf dem Bildschirm aus.

Alle folgenden Unterprogramme stehen erst seit Version 2.0 zur Verfügung

6.3.27) hardcopy 26

Diese Funktion druckt den momentanen Bildschirminhalt auf einen an die Parallelschnittstelle angeschlossenen Drucker aus.
Es wird nur der Textmodus unterstützt.

Parameter: keine

Hinweis: Falls der eingeschaltete Drucker off line ist, "hängt" der Computer in einer Warteschleife bis er wieder on line ist.

Beispiel:
```
* HARDCOPY VOM BILDSCHIRM
*
START:
  MOVE    #26,D7
  TRAP    #6
  RTS
END
```

Das obige Programm druckt den aktuellen Bildschirminhalt aus.

6.3.28) bell 27

Diese Funktion erzeugt einen Glockenton mittels der SOUND-Baugruppe. Die Baugruppe muß auf die I/O-Adresse $FFFFFF50 gelegt werden, da die vom Grundprogramm vorgesehene Adresse $FFFFFF40 zu Schwierigkeiten mit der Druckerschnittstelle führt.

Parameter: keine

Hinweis: Falls die Baugruppe SOUND nicht angeschlossen ist oder auf einer anderen I/O-Adresse liegt, führt dies zu keinerlei Problemen. Es wird lediglich die Funktion nicht ausgeführt.

Beispiel:
```
* GLOCKENTON
*
START:
  MOVE    #27,D7
  TRAP    #6
  RTS
END
```

Das obige Programm erzeugt einen Glockenton.

6.3.29) beep 28

Diese Funktion erzeugt einen Ton über die SOUND-Baugruppe.
Die Baugruppe muß auf die I/O-Adresse $FFFFFF50 gelegt werden, da die vom Grundprogramm vorgesehene Adresse $FFFFFF40 zu Schwierigkeiten mit der Druckerschnittstelle führt.
Die Tonhöhe und -dauer ist frei wählbar.

Parameter:

Register	Eing./Ausg.	Bedeutung
D1.l	x	Tonhöhe in Hertz (20 ... 5000)
D2.l	x	Tondauer in msec (0 ... 8000)

Hinweis: Falls die Baugruppe SOUND nicht angeschlossen ist, oder auf einer anderen I/O-Adresse liegt, führt dies zu keinerlei Problemen. Es wird lediglich die Funktion nicht ausgeführt. Es werden keine Register verändert.

Beispiel:

```
* TON ERZEUGEN
*
START:
  MOVE.L   #1000,D1
  MOVE.L   #1000,D2
  MOVE     #28,D7
  TRAP     #6
  RTS
END
```

Im Beispiel wird ein Ton von 1000 Hz für die Dauer von 1 s erzeugt.

6.3.30) errnoise 29

Diese Funktion erzeugt einen kurzen Ton über die SOUND-Baugruppe.
Die Baugruppe muß auf die I/O-Adresse $FFFFFF50 gelegt werden, da die vom Grundprogramm vorgesehene Adresse $FFFFFF40 zu Schwierigkeiten mit der Druckerschnittstelle führt.

Parameter: keine

Hinweis: Falls die Baugruppe SOUND nicht angeschlossen ist oder auf einer anderen I/O-Adresse liegt, führt dies zu keinerlei Problemen. Es wird lediglich die Funktion nicht ausgeführt. Register werden nicht verändert.

Beispiel:

```
* FEHLERTON
*
START:
  MOVE     #29,D7
  TRAP     #6
  RTS
END
```

Das obige Programm erzeugt einen kurzen Ton als akustische Fehlermeldung.

6.3.31) sound 30

Diese Funktion unterstützt die SOUND-Baugruppe in der gleichen Weise wie das Grundprogramm (Funktion Nr. 114).
Die Baugruppe muß auf die I/O-Adresse $FFFFFF50 gelegt werden, da die vom Grundprogramm vorgesehene Adresse $FFFFFF40 zu Schwierigkeiten mit der Druckerschnittstelle führt.

Parameter:

Register	Eing./Ausg.	Bedeutung
A0.l	x	Adresse der Soundtabelle

Hinweis: Falls die Baugruppe SOUND nicht angeschlossen ist oder auf einer anderen I/O-Adresse liegt, führt dies zu keinerlei Problemen. Es wird lediglich die Funktion nicht ausgeführt. Es werden keine Register verändert.

Beispiel:

```
* TON ERZEUGEN
*
START:
  LEA     SOUNDTAB(PC),A0
  MOVE    #30,D7
  TRAP    #6
  RTS

SOUNDTAB: DC.B $DE,$01,$DD,$01,$BE,$00,$00,$F8
          DC.B $10,$10,$10,$00,$0A,$08,$00,$00
END
```

Im Beispiel wird ein rhythmisches Klanggebilde erzeugt.

6.3.32) inport 31

Der NDR-Klein-Computer kann derzeit mit drei Prozessoren der 68000-Familie betrieben werden, dem 68008, dem 68000 und dem 68020. Die Prozessoren unterscheiden sich in der Breite des Datenbusses. Dies wirkt sich auf die Adressen der Peripheriebaugruppen aus. Wenn man die Adressen, die im 68008-System vorkommen, als Basisadressen bezeichnet, so muß man diese beim Einsatz des 68000 mit zwei und beim 68020 mit vier multiplizieren.
Zur Erstellung allgemeingültiger Applikationen für alle drei CPUs muß dies berücksichtigt werden.
Die hier angebotene Funktion unterstützt die CPU-unabhängige Portprogrammierung. Die jeweilige CPU wird automatisch erkannt und die Berechnung der tatsächlichen Portadresse ausgehend von der Basisadresse durchgeführt.

Mit inport kann ein Wert von einer I/O-Adresse eingelesen werden.

Parameter:	Register	Eing./Ausg.		Bedeutung
	D0.b		x	Eingelesener Wert von Port
	D1.l	x		Basis-Portadresse

Hinweis: Es wird nur Register D0 verändert.
Es soll nicht verschwiegen werden, daß diese Funktion 10 bis 20 mal langsamer ist als das direkte Einlesen vom Port. Für viele Anwendungen ist dies aber ausreichend.

Beispiel:
```
* PORTADRESSE EINLESEN
*
START:
  MOVE.L   #$FFFFFF68,D1
  MOVE     #31,D7
  TRAP     #6
  RTS
END
```

Im Beispiel wird ein Wert von der Tastatur eingelesen. Der Wert steht anschließend in D0.

6.3.33) outport 32

Der NDR-Klein-Computer kann derzeit mit drei Prozessoren der 68000-Familie betrieben werden, dem 68008, dem 68000 und dem 68020. Die Prozessoren unterscheiden sich in der Breite des Datenbusses. Dies wirkt sich auf die Adressen der Peripheriebaugruppen aus. Wenn man die Adressen, die im 68008-System vorkommen, als Basisadressen bezeichnet, so muß man diese beim Einsatz des 68000 mit zwei und beim 68020 mit vier multiplizieren.
Zur Erstellung allgemeingültiger Applikationen für alle drei CPUs muß dies berücksichtigt werden.
Die hier angebotene Funktion unterstützt die CPU-unabhängige Portprogrammierung. Die jeweilige CPU wird automatisch erkannt und die Berechnung der tatsächlichen Portadresse ausgehend von der Basisadresse durchgeführt.

Mit outport wird ein Wert auf ene I/O-Adresse geschrieben.

Parameter:

Register	Eing./Ausg.	Bedeutung
D0.b	x	Ausgabewert auf Port
D1.l	x	Basis-Portadresse

Hinweis: Es werden keine Register verändert.
Es soll nicht verschwiegen werden, daß diese Funktion 10 bis 20 mal langsamer ist als die direkte Ausgabe auf ein Port. Für viele Anwendungen ist dies aber akzeptabel.

Beispiel:

```
* WERT AUF PORT AUSGEBEN
*
START:
  MOVE.L   #$FFFFFF49,D1
  MOVE.B   #1,D0
  BSR      OUT
  MOVE.L   #$FFFFFF48,D1
  MOVE.B   #12,D0
  BSR      OUT
  MOVE.L   #$FFFFFF49,D1
  CLR.B    D0
  BSR      OUT
  MOVE.B   #1,D0
  BSR      OUT
  RTS
OUT:
  MOVE     #32,D7
  TRAP     #6
  RTS
END
```

Im Beispiel wird das Seitenvorschubzeichen auf die Druckerschnittstelle ausgegeben.

6.3.34) movelbin 33

Mit dieser Funktion kann ein Speicherbereich kopiert werden. Dabei wird bei niedrigen Adressen begonnen (von links !).

Parameter:

Register	Eing./Ausg.	Bedeutung
A1.l	x	Adresse der Quelle
A2.l	x	Adresse des Ziels
D1.w	x	Anzahl der Bytes (1 .. 65535)

Hinweis: Es werden keine Register verändert.

Beispiel:

```
* GRUNDPROGRAMM VERSCHIEBEN
*
START:
  MOVEA.L  #$E0000,A1
  MOVEA.L  #$20000,A2
  MOVE     #32768,D1
  MOVE     #33,D7
  TRAP     #6
  RTS
END
```

Im Beispiel wird das auf Adresse $E0000 liegende Grundprogramm in den RAM-Speicher ab Adresse $20000 kopiert.

6.3.35) moveltxt 34

Diese Funktion kopiert einen Textbereich im Hauptspeicher. Der Quelltext muß mit einer binären Null abgeschlossen sein.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A1.l	x	Adresse der Quelle
	A2.l	x	Adresse ds Ziels

Hinweis: Es werden keine Register verändert.

Beispiel:

```
* TEXT KOPIEREN
*
START:
  MOVEA.L  #$400,A1
  MOVEA.L  #$10000,A2
  MOVE     #34,D7
  TRAP     #6
  RTS
END
```

Das obige Programm kopiert einen ab Adresse $400 stehenden Text auf Adresse $10000.

6.3.36) moverbin 35

Mit dieser Funktion kann ein Speicherbereich kopiert werden. Dabei wird bei höheren Adressen begonnen (von rechts !!).

Parameter:	Register	Eing./Ausg.	Bedeutung
	A1.l	x	Adresse der Quelle
	A2.l	x	Adresse des Ziels
	D1.w	x	Anzahl der Bytes (1 .. 65535)

Hinweis: Es werden keine Register verändert.

Beispiel:

```
* GRUNDPROGRAMM VERSCHIEBEN
*
START:
  MOVEA.L  #$E0000,A1
  MOVEA.L  #$20000,A2
  MOVE     #32768,D1
  MOVE     #35,D7
  TRAP     #6
  RTS
END
```

Im Beispiel wird das auf Adresse $E0000 liegende Grundprogramm in den RAM-Speicher ab Adresse $20000 kopiert.

6.3.37) wrtcmd 36

Mit dieser Funktion kann ein JADOS-Kommando mit Parametern in den automatischen Kommandopuffer geschrieben werden. Sobald der Kommandointerpreter aktiv ist, wird der automatische Kommandopuffer abgearbeitet.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.1	x	Adresse des Kommandotextes

Hinweis: Falls gerade eine Stapelverarbeitung abläuft, ist die Funktion unwirksam, da die Stapelverarbeitung Priorität hat.

Beispiel:

```
* AUFRUF EINES KOMMANDOS AUS EINER ANWENDUNG
*
START:
  LEA       COMMTEXT(PC),A0
  MOVE      #36,D7
  TRAP      #6
  RTS
COMMTEXT:  DC.B 'DIR 2:*.68K',0
END
```

Das obige Programm schreibt ein JADOS-Kommando in den Kommandopuffer. Nach der Rückkehr in den JADOS-Kommandointerpreter wird das Kommando ausgeführt. Hier wird z.B. das Inhaltsverzeichnis aller .68K-Dateien auf Laufwerk 2: angezeigt.

6.3.38) gtretcod 37

In JADOS gibt es einen Speicherbereich von 4 Bytes, auf den Programme einen Wert schreiben können, der von anderen Programmen gelesen werden kann.

Die Funktion gtretcod liest diesen Wert.

Parameter:	Register	Eing./Ausg.	Bedeutung
	D0.1	x	Applikationscode

Hinweis: Es wird nur Register D0 geändert.

Beispiel:

```
* APPLIKATIONSCODE LESEN
*
START:
  MOVE      #37,D7
  TRAP      #6
  RTS
END
```

Im Beispiel wird der Code eingelesen, der zuletzt von einer anderen Anwendung geschrieben wurde. Bei Start initialisiert JADOS den Wert auf Null.

6.3.39) stretcod 38

In JADOS gibt es einen Speicherbereich von 4 Bytes, auf den Programme einen Wert schreiben können, der von anderen Programmen gelesen werden kann.

Die Funktion stretcod schreibt diesen Wert.

Parameter:

Register	Eing./Ausg.	Bedeutung
D0.l	x	Applikationscode

Hinweis: Es werden keine Register geändert.

Beispiel:

```
* APPLIKATIONSCODE SETZEN
*
START:
  MOVE.L   #1,D0
  MOVE     #38,D7
  TRAP     #6
  RTS
END
```

Im Beispiel wird der Wert 1 geschrieben, um z.B. einen Fehlerzustand anzuzeigen.

6.3.40) moveline 39

Diese Funktion dient zur Bearbeitung von Texten. Dabei wird bei jedem Aufruf eine mit <CR> <LF> abgeschlossene Textzeile in einen speziellen Pufferspeicher geschrieben und der Textzeiger auf die nächste Zeile positioniert.

Parameter:

Register	Eing./Ausg.		Bedeutung
A0.l	x	x	Adresse der Textzeile
A2.l	x		Adresse des Pufferspeichers
D2	x		Größe des Pufferspeichers
D0.b		x	Ergebniscode:
			0 - ok
			$FF - Textende erreicht

Hinweis: Es werden die Register D0 und A0 verändert.
Der Text im Pufferspeicher wird mit einer binären Null abgeschlossen.

Beispiel:

```
* TEXT AUSDRUCKEN
*
START:
  LEA      $400,A0
LOOP:
  LEA      BUF(PC),A2
  MOVE     #80,D2
  MOVE     #39,D7
  TRAP     #6
  CMP.B    #$FF,D0
  BEQ.S    ENDE
  EXG.L    A0,A2
```

```
        MOVE      #50,D7
        TRAP      #1
        MOVE      #7,D7
        TRAP      #6
        MOVE      #99,D7
        TRAP      #1
        MOVE      #49,D7
        TRAP      #1
        MOVE.B    #'X',D0
        MOVE      #33,D7
        TRAP      #1
        EXG.L     A0,A2
        BRA       LOOP
ENDE:
        RTS
BUF:  DS.B  80
END
```

Im Beispiel wird ein mit dem Editor erstellter Text, der ab Adresse $400 steht, zeilenweise ausgedruckt. Nach jeder Zeile wird ein "X" auf dem Bildschirm ausgegeben.

6.3.41) wraddr 40

Hiermit wird eine sedezimale Adresse auf 6 Stellen ausgegeben.

Parameter:	Register	Eing./Ausg.	Bedeutung
	D0.l	x	Adresse

Hinweis: Es wird auf die CO2-Schnittstelle ausgegeben. Register werden nicht verändert.

Beispiel:

```
* GRUNDPROGRAMMSTART AUSGEBEN
*
START:
        MOVE      #89,D7
        TRAP      #1
        MOVE      #40,D7
        TRAP      #6
        RTS
END
```

Im Beispiel wird die Startadresse des Grundprogramms ermittelt und auf die CO2-Schnittstelle ausgegeben.

6.3.42) ci 41

Hiermit kann die Tastaturleseroutine von JADOS verwendet werden. Immer wenn Ctrl P eingegeben wird, wird eine Hardcopy des Bildschirms ausgedruckt.

Parameter:

Register	Eing./Ausg.	Bedeutung
D0.b	x	Tastaturwert

Beispiel:

```
* TASTATUR EINLESEN
*
START:
  MOVE     #41,D7
  TRAP     #6
  RTS
END
```

Im Beispiel wird auf eine Tastatureingabe gewartet. Bei jeder Eingabe von Ctrl P wird eine Text-Hardcopy ausgedruckt.

6.3.43) getparm1 42

In JADOS können jedem Kommando bis zu vier Parameter mitgegeben werden. Die Funktion "getparm" (Nr. 25) liefert einen Zeiger auf den gesamten Parameterstring.
Aus dem kompletten Parameterstring bildet JADOS vier einzelne Strings, die jeweils mit einer binären Null abgeschlossen sind. Dadurch können Programme sehr einfach auf die einzelnen Parameter zugreifen, insbesondere wenn mehrere Parameter übergeben werden.
Die vorliegende Funktion liefert den ersten Parameter.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.l	x	Adresse des 1. Parameters

Hinweis: Es wird nur Register A0 verändert.

Beispiel:
```
* 1. PARAMETER AUSGEBEN
*
START:
  MOVE    #42,D7
  TRAP    #6
  MOVE    #7,D7
  TRAP    #6
  RTS
END
```

Im Beispiel wird der 1. Programmparameter über den aktuellen CO2-Kanal ausgegeben.

6.3.44) getparm2 43

In JADOS können jedem Kommando bis zu vier Parameter mitgegeben werden. Die Funktion "getparm" (Nr. 25) liefert einen Zeiger auf den gesamten Parameterstring.
Aus dem kompletten Parameterstring bildet JADOS vier einzelne Strings, die jeweils mit einer binären Null abgeschlossen sind. Dadurch können Programme sehr einfach auf die einzelnen Parameter zugreifen, insbesondere wenn mehrere Parameter übergeben werden.
Die vorliegende Funktion liefert den zweiten Parameter.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.l	x	Adresse des 2. Parameters

Hinweis: Es wird nur Register A0 verändert.

Beispiel:
```
* 2. PARAMETER AUSGEBEN
*
START:
  MOVE    #43,D7
  TRAP    #6
  MOVE    #7,D7
  TRAP    #6
  RTS
END
```

Im Beispiel wird der 2. Programmparameter über den aktuellen CO2-Kanal ausgegeben.

6.3.45) fileload 44

Mit dieser Funktion kann eine komplette Datei von Diskette geladen werden. Die Ladegeschwindigkeit ist seit Version 2.0 optimiert und um den Faktor 2 schneller als Programmschleifen mit der Funktion "readrec" (Nr. 20).

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.l	x	Ladeadresse
	A1.l	x	Adresse des Dateisteuerblocks
	D0.b	x	Fehlercode: 0 - ok
			2 - Datei nicht gefunden
			99 - Speicher voll
			$FF - Sonstiger Fehler

Hinweis: Es wird nur Register D0 verändert.

Beispiel:

```
* DATEI LADEN
*
START:
  LEA       NAM(PC),A0
  LEA       FCB(PC),A1
  MOVE      #18,D7
  TRAP      #6
  CMP.B     #0,D0
  BNE.S     FEHLER
  LEA       ADR(PC),A0
  MOVE      #44,D7
  TRAP      #6
FEHLER:
  RTS
NAM: DC.B 'BEISPIEL.TXT',0
     DS 0
FCB: DS.B 48
ADR:
END
```

Im Beispiel wird die Datei "BEISPIEL.TXT" in einen Dateisteuerblock eingetragen und dann komplett geladen.

6.3.46) filesave 45

Mit dieser Funktion kann eine komplette Datei auf Diskette gespeichert werden. Die Speichergeschwindigkeit ist seit Version 2.0 optimiert und um den Faktor 2 - 5 schneller als Programmschleifen mit der Funktion "writerec" (Nr. 23).

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.l	x	Ladeadresse
	A1.l	x	Adresse des Dateisteuerblocks
	D1.w	x	Anzahl der Sektoren minus 1
	D0.b	x	Fehlercode: 0 - ok
			5 - Diskette voll
			6 - Inhaltsverzeichnis voll
			$FF - Sonstiger Fehler

Hinweis: Es wird nur Register D0 verändert.

Beispiel:
```
* DATEI SPEICHERN
*
START:
  LEA       NAM(PC),A0
  LEA       FCB(PC),A1
  MOVE      #18,D7
  TRAP      #6
  CMP.B     #0,D0
  BNE.S     FEHLER
  MOVEA.L   #$800,A0
  MOVE      #5,D1
  MOVE      #45,D7
  TRAP      #6
FEHLER:
  RTS
NAM: DC.B 'BEISPIEL.TXT',0
     DS 0
FCB: DS.B 48
END
```

Im Beispiel wird die Datei "BEISPIEL.TXT" in einen Dateisteuerblock eingetragen. Anschließend wird der ab Adresse $800 abgelegte Text mit der Größe 5 KByte auf Diskette gespeichert.

6.3.47) getparm3 46

In JADOS können jedem Kommando bis zu vier Parameter mitgegeben werden. Die Funktion "getparm" (Nr. 25) liefert einen Zeiger auf den gesamten Parameterstring.
Aus dem kompletten Parameterstring bildet JADOS vier einzelne Strings, die jeweils mit einer binären Null abgeschlossen sind. Dadurch können Programme sehr einfach auf die einzelnen Parameter zugreifen, insbesondere wenn mehrere Parameter übergeben werden.
Die vorliegende Funktion liefert den dritten Parameter.

Parameter:

Register	Eing./Ausg.	Bedeutung
A0.l	x	Adresse des 3. Parameters

Hinweis: Es wird nur Register A0 verändert.

Beispiel:
```
* 3. PARAMETER AUSGEBEN
*
START:
  MOVE      #46,D7
  TRAP      #6
  MOVE      #7,D7
  TRAP      #6
  RTS
END
```

Im Beispiel wird der 3. Programmparameter über den aktuellen CO2-Kanal ausgegeben.

6.3.48) getparm4 47

In JADOS können jedem Kommando bis zu vier Parameter mitgegeben werden. Die Funktion "getparm" (Nr. 25) liefert einen Zeiger auf den gesamten Parameterstring.
Aus dem kompletten Parameterstring bildet JADOS vier einzelne Strings, die jeweils mit einer binären Null abgeschlossen sind. Dadurch können Programme sehr einfach auf die einzelnen Parameter zugreifen, insbesondere wenn mehrere Parameter übergeben werden.
Die vorliegende Funktion liefert den vierten Parameter.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.l	x	Adresse des 4. Parameters

Hinweis: Es wird nur Register A0 verändert.

Beispiel:

```
* 4. PARAMETER AUSGEBEN
*
START:
  MOVE    #47,D7
  TRAP    #6
  MOVE    #7,D7
  TRAP    #6
  RTS
END
```

Im Beispiel wird der 4. Programmparameter über den aktuellen CO2-Kanal ausgegeben.

6.3.49) loadpart 48

Mit dieser Funktion kann eine geöffnete Datei in den Speicher geladen werden. Der Ladevorgang wird beendet, wenn entweder das Dateiende oder das Speicherende erreicht ist.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.l	x	Ladeadresse
	A1.l	x	Adresse des Dateisteuerblocks
	D2.w	x	Anzahl der geladenen Sektoren
	D0.b	x	Ergebniscode: 0 - Teil geladen
			1 - Komplett geladen
			$FF - Fehler

Hinweis: Es werden die Register D0 und D2 geändert.
Diese Funktion sollte nur von erfahrenen Programmierern verwendet werden.

Beispiel: siehe Funktion "savepart" (Nr. 49)

6.3.50) savepart 49

Mit dieser Funktion kann auf eine geöffnete Datei geschrieben werden.
Der Speichervorgang erfolgt geschwindigkeitsoptimiert.

Parameter:

Register	Eing./Ausg.		Bedeutung
A0.l	x		Ladeadresse
A1.l	x		Adresse des Dateisteuerblocks
D2.w	x		Anzahl der Sektoren
D0.b		x	Ergebniscode: 0 - ok
			5 - Diskette voll
			$FF - Fehler

Hinweis: Es wird nur Register D0 geändert.
Diese Funktion sollte nur von erfahrenen Programmierern verwendet werden.

Beispiel:

```
* SCHNELLES KOPIERPROGRAMM
*
* a0: Ladeadresse
* a1: Zeiger auf Quell-FCB
* a2: Zeiger auf Ziel-FCB
* -> d0: Fehlercode
*      0 - ok
*    $FF - Undef. Fehler
*      2 - Quelle nicht gef.
*      5 - Disk voll
*      6 - Directory voll

COPYFILE:
  movem.l  d1-d5/a0-a3,-(a7)
* Quelle suchen
  move     #19,d7
  trap     #6
* Existiert Quelle ?
  cmp      #255,d0
* NEIN --> Fehler
  beq      copserr

* JA --> Ziel suchen
  exg.l    a1,a2
  move     #16,d7
  trap     #6
  cmp.b    #0,d0
  bne.s    copyend
```

```
copyloop:
   exg.l     a1,a2
* LOADPART
   move      #48,d7
   trap      #6
   move      d0,d3
   cmp.b     #255,d0
   beq       coperr1
   exg.l     a1,a2
* SAVEPART
   move      #49,d7
   trap      #6
   cmp.b     #5,d0
   beq       copdf
   cmp.b     #0,d0
   beq.s     copsok
   bra       coperr1
copsok:
* DATEIENDE ERREICHT ?
   cmp.b     #1,d3
* NEIN, DANN WEITER KOPIEREN
   bne.s     copyloop

* Beendet
   move      #14,d7
   trap      #6
   clr       d0
   bra       copyend

* DISKETTE VOLL
copdf:
   exg.l     a1,a2
   move      #14,d7
   trap      #6
   move      #5,d0
   bra.s     copyend

* SONSTIGER FEHLER
coperr1:
   exg.l     a1,a2
   move      #14,d7
   trap      #6
   move      #255,d0
   bra.s     copyend

* QUELLE NICHT GEFUNDEN
copserr:
   move      #2,d0
copyend:
   movem.l   (a7)+,d1-d5/a0-a3
   RTS

END
```

Bei diesem Beispiel handelt es sich weitgehend um die im JADOS enthaltene Routine "copyfile" (Nr. 15). Zuerst wird die Quelldatei gesucht. Ist sie vorhanden, dann wird die Zieldatei angelegt. In der Schleife COPYLOOP wird die Quelldatei bis zum Dateiende oder Speicherende eingelesen und auf die Zieldatei gespeichert. Wurde nur bis zum Speicherende eingelesen, so wird der nächste Block mit "loadpart" (Nr. 48) gelesen und mit "savepart" (Nr. 49) gespeichert, und zwar solange, bis in "loadpart" das Ende der Quelldatei erkannt wurde.

6.3.51) catalog 50

Diese Funktion dient zur Ausgabe des Inhaltsverzeichnisses einer Diskette über den aktuellen CO2-Kanal. Laufwerk und eine Dateispezifikation können als Parameter übergeben werden.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.1	x	Adresse einer Textzeile mit einer Dateispezifikation.

Hinweis: Es werden keine Register verändert.
Zeigt A0 auf einen "leeren" Text, dann wird der gesamte Inhalt des aktuellen Laufwerks ausgegeben.

Beispiel:

```
* INHALTSVERZEICHNIS ANZEIGEN
*
START:
  LEA       DATSPEZ(PC),A0
  MOVE      #50,D7
  TRAP      #6
  RTS
DATSPEZ:    DC.B '2:X*.68K',0
END
```

Im Beispiel werden alle Dateien vom Typ .68K, die mit einem "X" beginnen und sich auf Laufwerk 2 befinden, angezeigt.

Alle folgenden Unterprogramme stehen erst seit Version 2.1 zur Verfügung

6.3.52) floppy 51

Diese Funktion entspricht der gleichnamigen Routine des Grundprogramms. Im Unterschied dazu schaltet es erst die Laufwerkmotoren an und wartet vor dem Zugriff auf die Diskette 500 ms, falls die Motoren abgeschaltet waren. Liefen die Motoren bereits, so erfolgt der Zugriff unmittelbar. Dies funktioniert aber nur dann einwandfrei, wenn die Laufwerksmotoren mit der Funktion "motoroff" abgeschaltet wurden.
Entdeckt die Routine einen Zugriffsfehler, so wird ein 2. Versuch unternommen. Schlägt auch dieser fehl, so meldet die Funktion einen Fehler im Register D0 und gibt eine Fehlermeldung aus.

Parameter: Register	Eing./Ausg.		Bedeutung
D4.B	x		Laufwerkscode, Dichte, Seite
D3.W	x		Spurnummer
D2.W	x		Sektornumer
D1.W	x		0 = Steprate in D3 setzen
			1 = Lesen
			2 = Schreiben
A0.L	x		Ablageadresse zum Lesen/Schreiben
D0.B		x	Ergebniscode: 0 = ok
			$FF = Fehler

Hinweis: Es wird nur Register D0 verändert. Eine eventuelle Fehlerausgabe erfolgt über die CO2-Schnittstelle.

Beispiel:

```
START:
  MOVE      #1,D1          ** Bootsektor lesen **
  MOVE      #1,D2          * Lesen
  MOVE      #0,D3          * Sektor 1
  MOVE      #$21,D4        * Spur 0
  LEA       $D000,A0       * Mini, Laufwerk 1
  MOVE      #51,D7         * Ablageadresse
  TRAP      #6             * Funktion ausführen
  RTS
```

Im Beispiel wird der Bootsektor gelesen und auf Adresse $D000 abgelegt.

6.3.53) drivecod 52

Diese Funktion wandelt eine dezimale Laufwerksnummer in eine binär codierte um, so wie es die Routine "floppy" verlangt. So ergibt z.B. Laufwerk 2 den binären Wert 0000 0010 und Laufwerk 4 den Wert 0000 1000.

Parameter:	Register	Eing./Ausg.		Bedeutung
	D4.W	x	x	Als Eingabe wird die dezimale Laufwerksnummer verlangt. Als Ausgabe wird der binäre Wert geliefert

Hinweis: Im Zusammenhang mit den Laufwerken sind nur die dezimalen Werte 1 bis 4 interessant.

Beispiel:

```
START:                          * Alle sinnvollen binären Laufwerkscodes ausgeben
  MOVE      #3,D3               * Mit höchstem Laufwerk beginnen
LOOP:
  MOVE      d3,D4
  ADDQ      #1,D4
  BSR       OUT                 * Wandeln und ausgeben
  DBF       D3,LOOP
  RTS                           * Ende
OUT:                            * Binärwert ausgeben
  MOVE      #52,D7              * Laufwerkscode wandeln
  TRAP      #6
  MOVE      D4,D0
  MOVE      #13,D7              * Binärwert ausgeben
  TRAP      #6
  MOVE      #!CRLF,D7           * Zeilenvorschub
  TRAP      #1
  RTS
```

Im Beispiel werden die sinnvollen Laufwerksnummern 1 bis 4 in den binären Wert gewandelt und angezeigt.

6.3.54) getdrive 53

Diese Funktion liefert die Nummer des aktuellen Laufwerks.

Parameter:	Register	Eing./Ausg.	Bedeutung
	D0.W	x	Nummer des aktuellen Laufwerks

Hinweis: Es wird nur Register D0 verändert.

Beispiel:

```
START:                          * Nummer des aktuellen
                                  Laufwerks ausgeben
  MOVEQ     #53,D7              * Nummer des aktuellen
                                  Laufwerks holen
  TRAP      #6
  MOVEQ     #13,D7              * und ausgeben
  TRAP      #6
  RTS
```

Im Beispiel wird die Nummer des aktuellen Laufwerks angezeigt.

6.3.55) setdrive 54

Mit dieser Funktion wird die Nummer des aktuellen Laufwerks verändert.

Parameter:	Register	Eing./Ausg.	Bedeutung
	D0.W	x	Neue Laufwerksnummer

Hinweis: Es werden nur die erlaubten Laufwerksnummern 1 bis 4 akzeptiert.

```
Beispiel: START:                          * Aktuelles Laufwerk ändern
            MOVE        #4,D0             * Laufwerk 4 auswählen
            MOVEQ       #54,D7            * und setzen
            TRAP        #6
            RTS
```

Im Beispiel wird das Laufwerk 4 aktuell.

6.3.56) gttraptb 55

Diese Funktion liefert die Anfangsadresse der Adreßtabelle der JADOS-Funktionen. Diese Adressen sind 4 Byte lang und sind relativ zur Anfangsadresse der Adreßtabelle.
Der Anwender kann nun JADOS-Funktionen durch eigene ersetzen. Dieser Ersatz funktioniert allerdings nur, wenn die Funktionen mit TRAP #6 aufgerufen werden. JADOS selber ruft die Funktionen mit BSR xxx auf; somit kommt der Ersatz in JADOS selber nicht zur Geltung, wohl aber in Anwenderprogrammen.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.L	x	Anfangsadresse der TRAP-Tabelle

Hinweis: Diese Funktion sollte nur zu Experimentierzwecken verwendet werden.

```
Beispiel: START:
            MOVE        #55,D7            * Tabellenadresse holen
            TRAP        #6
            MOVE.l      A0,D0             * in D0 speichern
            ADDA.l      #20,A1            * den 6. Eintrag adressieren
                                            (motoroff)
            LEA         BELL(PC),A1       * Adresse der eigenen Routine
                                            holen
            SUBA.l      D0,A1             * Tabellenadresse abziehen
            MOVE.l      A1,(A0)           * und in Tabelle ablegen
            RTS

          BELL:                           * Eigene Routine
            MOVE        #27,D7            * Klingelzeichen
            TRAP        #6
            RTS
```

Im Beispiel wird die Routine "motoroff" durch eine eigene ersetzt, die ein Klingelzeichen ertönen läßt.

6.3.57) blockread 56

Mit dieser Funktion wird eine Folge von Sektoren gelesen. Der Lesevorgang erfolgt im Highspeed-Modus, d.h. die Spurtabelle verbleibt während des Lesevorgangs im Hauptspeicher.

Parameter:	Register	Eing./Ausg.	Bedeutung
	D2.w	x	Anzahl der zu lesenden Sektoren
	A0.L	x	Ablageadresse
	A1.L	x	Zeiger auf FCB
	D0.B	x	Ergebniscode: 0 = ok
			1 = Dateiende
			99 = Speicher voll
			$FF = Sonstiger Fehler

Hinweis: Vor dem Zugriff muß die Datei geöffnet sein.

```
Beispiel: START:
          LEA         NAM(PC),A0      * Dateinamen holen
          LEA         FCB(PC),A1      * FCB adressieren
          MOVEQ       #18,D7          * Datei in FCB eintragen
          TRAP        #6
          CMP.B       #0,D0           * Eintrag ok?
          BNE.S       ENDE            * Nein. > Dann Ende.
          MOVEQ       #19,D7          * Datei öffnen
          TRAP        #6
          CMP.B       #0,D0           * Datei öffnen?
          BNE.S       ENDE            * Nein. > Dann Ende.
          LEA         $10000,A0       * Ablageadresse
          MOVEQ       #04,D2          * 4 Sektoren lesen
          MOVEQ       #56,D7          * Funktion aufrufen
          TRAP        #6
          MOVEA.l     44(A1),A0       * Am Ende Textendezeichen
          CLR         (A0)            * für z.B. Textdatei
         ENDE:
          MOVEQ       #14,D7          * Datei schließen
          TRAP        #6
          RTS
         NAM:         DC.B '3:DOSTOOL.BAK',0
                      DS 0
         FCB:         DS.B 48
```

Im Beispiel werden die ersten 4 Sektoren der Datei "DOSTOOL.BAK" gelesen und ab $10000 abgelegt.

6.3.58) blockwrite 57

Mit dieser Funktion wird eine Folge von Sektoren gespeichert. Der Speichervorgang erfolgt im Highspeed-Modus, d.h. die Spurtabelle verbleibt während des Speichervorgangs im Hauptspeicher.

Parameter:	Register	Eing./Ausg.	Bedeutung
	D2.W	x	Anzahl der zu speichernden Sektoren
	A0.L	x	Ablageadresse
	A1.L	x	Zeiger auf FCB
	D0.B	x	Ergebniscode: 0 = okay 5 = Disk voll $FF = Sonstiger Fehler

Hinweis: Vor dem Zugriff muß die Datei angelegt oder geöffnet sein.

Beispiel:

```
      START:
        LEA             NAM(PC),A0      * Dateinamen holen
        LEA             FCB(PC),A1      * FCB adressieren
        MOVE            #18,D7          * Datei in FCB eintragen
        TRAP            #6
        CMP.B           #0,D0           * Eintrag okay?
        BNE.S           ENDE            * Nein. > Dann Ende
        MOVE            #16,D7          * Datei neu anlegen
        TRAP            #6
        CMP.B           #0,D0           * Datei angelegt?
        BNE.S           ENDE            * Nein. > Dann Ende
        LEA             $10000          * Ab Adresse $10000 speichern
        MOVE            #20,D2          * 20 Sektoren speichern
        MOVE            #57,D7
        TRAP            #6
      ENDE:
        MOVE            #14,D7          * Datei schließen
        TRAP            #6
        RTS
      NAM:              DC.B '3:DOSTOOL.KKK',0
                        DS 0
      FCB:              DS.B 48
```

6.3.59) getladdr 58

Diese Funktion liefert die Ladeadresse für 68K-Dateien.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.L	x	Ladeadresse

```
Beispiel:  START:
             MOVE      #58,D7          * Ladeadresse holen
             TRAP      #6
             MOVE.L    A0,D0           * und auf Bildschirm anzeigen
             MOVE      #40,D7          * in Hex
             TRAP      #6
             RTS
```

Im Beispiel wird die Ladeadresse angezeigt.

6.3.60) skipchar 59

Mit dieser Funktion kann ein bestimmtes Zeichen in einem String, z.B. das Leerzeichen, übersprungen werden.

Parameter:	Register	Eing./Ausg.		Bedeutung
	D1.B	x		zu überspringendes Zeichen
	A0.L	x	x	Zeiger auf Text

```
Beispiel:  START:
             LEA       TEXT(PC),A0     * Text adressieren
             MOVEQ     #7,D7           * und anzeigen
             TRAP      #6
             MOVE.B    #' ',D1         * Leerzeichen
             MOVEQ     #59,D7          * überspringen
             TRAP      #6
             MOVEQ     #7,D7           * und erneut anzeigen
             TRAP      #6
             RTS

             TEXT: DC.B '    Hugo ist nicht doof',13,10,0
```

Im Beispiel werden die führenden Leerzeichen übersprungen.

6.3.61) delete 60

Mit dieser Funktion kann eine Anzahl Zeichen aus einem String gelöscht werden. Der restliche String wird entsprechend nach links geschoben.

Parameter:	Register	Eing./Ausg.	Bedeutung
	D1.W	x	Anzahl der zu löschenden Zeichen
	A0.L	x	Zeiger auf Stringposition, ab der gelöscht werden soll

Hinweis: Der String muß mit einer binären Null abgeschlossen sein. Sollen mehr Zeichen gelöscht werden als der String lang ist, so wird auf die Position, die durch A0 gekennzeichnet ist, eine binäre Null geschrieben.

```
Beispiel: START:
          LEA       TEXT(PC),A0     * String holen
          MOVEQ     #7,D7           * und anzeigen
          TRAP      #6
          ADDA.L    #9,A0           * Position von "nicht" adressie-
                                      ren
          MOVEQ     #6,D7           * 6 Zeichen löschen
          MOVEQ     #60,D7
          TRAP      #6
          MOVEQ     #7,D7
          TRAP      #6
          RTS

          Text: DC.B 'Hugo ist nicht doof',13,10,0
```

Im Beispiel wird aus dem String "Hugo ist nicht doof" das Wort "nicht" entfernt. Hugo möchte mir das verzeihen!

6.3.62) **insert 61**

Diese Funktion fügt einen Teilstring in einen String ein.

Parameter: Register	Eing./Ausg.	Bedeutung
D2.W	x	Maximal erlaubte Restlänge des Strings
A0.L	x	String, in den eingefügt werden soll
A1.L	x	String, der eingefügt werden soll

Hinweis: String und Teilstring müssen mit einer binären Null abgeschlossen sein. Ist der Teilstring länger als die maximale Restlänge, so wird die Funktion nicht ausgeführt.

```
Beispiel:
        START:
          LEA       TEXT(PC),A0     * Text adressieren
          LEA       TEXTI(PC),A1    * Teilstring adressieren
          MOVEQ     #7,D7           * Text anzeigen
          TRAP      #6
          ADDA.L    #9,A0           * Auf 10. Zeichen positionieren
          MOVEQ     #8,D2           * Restlänge 8 Zeichen
          MOVEQ     #61,D7          * Einfügen
          TRAP      #6
          LEA       TEXT(PC),A0     * Gesamten Text adressieren
          MOVEQ     #7,D7           * und wieder anzeigen
          TRAP      #6
          RTS

          TEXT: DC.B 'Hugo ist doof         ',13,10,0
          TEXTI:DC.B 'nicht ',0
```

Im Beispiel wird Hugo rehabilitiert, indem in den Beleidigungstext das Wort "nicht" eingefügt wird. Hugo ist hoffentlich versöhnt.

6.3.63) ramtop

Diese Funktion liefert das Ende des freien Benutzerspeichers.

Parameter:	Register	Eing./Ausg.	Bedeutung
	A0.L	x	Zeiger auf Speicherende

Hinweis: Für den Benutzerstack sind 1 KByte berücksichtigt.

Beispiel:

```
START:
  MOVEQ     #62,D7          * Ende des Benutzerspeichers
                              laden
  TRAP      #6
  MOVE.L    A0,d0
  MOVEQ     #40,D7          * und anzeigen
  TRAP      #6
  MOVE      #!CRLF,D7       * mit Zeilenvorschub
  TRAP      #1
  MOVE.L    A7,D0           * Stackpointer holen
  MOVEQ     #40,D7          * und anzeigen
  TRAP      #6
  RTS
```

Im Beispiel werden das Ende des freien Benutzerspeichers und der aktuelle Wert des Benutzerstackpointers angezeigt. Am oberen Ende des Stacks liegen die Rückkehradresse ins JADOS und die Register D0-D7 und A0-A6.

A) A N H A N G

A.1) DIE HILFSPROGRAMME VON JADOS

Zum Lieferumfang von JADOS gehören zur Zeit neun Hilfsprogramme.

A.1.1) Übersicht

Die folgende Übersicht informiert kurz über die bereitgestellten Hilfsprogramme:

Dateiname	Bedeutung
1COPY.68K	Kopierprogramm für Systeme mit nur einem Laufwerk
BRDRUCK.68K	Komfortables Druckprogramm
DISKCOPY.68K	Programm zum schnellen Kopieren kompletter Disketten
DSAVE.68K	Speichern eines Bereichs des Hauptspeichers auf Diskette
FORMAT.68K	Formatierprogramm
GP.COM	Komfortables Interface zum Grundprogramm
MORE.68K	Programm zur Anzeige von Textdateien mit superschneller Bildschirmausgabe
ROMSTART.68K	Programm zum Auffinden und Starten von Bibliotheksprogrammen, die im RAM oder im EPROM gespeichert sind
SYS.68K	Programm zum Kopieren der Systemspuren

Zum Zwecke eigener Anpassungen sind die Programme GP und BRDRUCK auch im Quellcode veröffentlicht.

A.1.2) 1COPY

Zweck: Programm zum Kopieren von Dateien für Systeme mit nur einem Laufwerk
Aufruf: 1COPY
Codeumfang: 2 KByte
Startadresse: aktuelle Ladeadresse

Bedienungsablauf:
Das Programm meldet sich mit seinem Namen, der Versionsnummer und dem Copyright.

```
EIN-LAUFWERK-KOPIERER V 2.1   (C) 1986 by K.Janßen
```

Anschließend erscheint folgendes Menü:

```
-----BITTE AUSWAEHLEN-----
     1 ---> KOPIEREN
     9 ---> beenden
```

Mit Eingabe einer "1" oder Return wird der Kopiervorgang eingeleitet, mit Eingabe einer "9" wird das Programm beendet.

Ein normaler fehlerfreier Kopiervorgang läuft folgendermaßen ab. Erst wird nach dem Namen der Quelldatei, dann nach der Zieldatei gefragt:

```
Quelldateinamen eingeben:
Zieldateinamen eingeben:
```

Sind die Laufwerksangaben unterschiedlich, dann verhält sich 1COPY genauso wie das interne Kommando COPY. Sind die Laufwerksangaben dagegen gleich, fordert 1COPY dazu auf, die Quelldiskette einzulegen:

```
Quell-Diskette einlegen  -->
LEERTASTE DRÜCKEN
```

Anschließend wird die Quelldatei komplett in den Hauptspeicher geladen. Danach kommt die Aufforderung, die Zieldiskette einzulegen:

```
Ziel-Diskette einlegen   -->
LEERTASTE DRÜCKEN
```

Nach dem Kopieren erscheint in der Regel eine Erfolgsmeldung:

```
Datei kopiert
```

Danach erscheint erneut das obige Menü.

Folgende Fehlersituationen werden erkannt und gemeldet:

- Quelldatei nicht gefunden
- Diskette ist voll
- Inhaltsverzeichnis ist voll
- Undefinierter Diskettenfehler
- Falscher Dateiname

A.1.3) BRDRUCK

Zweck: Komfortables Druckprogramm für EPSON-Drucker oder Kompatible
Aufruf: BRDRUCK
Codeumfang: 5 KByte
Startadresse: aktuelle Ladeadresse

Bedienung: Nach dem Programmstart meldet sich folgender Bildschirm:

```
------------------------------------------------------------------------------
Druckprogramm für Epson-Drucker    V 2.1   (C) 1986,1987 by K. Janßen

          >>>> V O R E I N S T E L L U N G E N <<<<<

F-ormularlänge      [65  ]          B-eginn Seitenzahl   [1   ]
L-inker Rand        [8   ]          P-osition Seitenzahl [40  ]
K-opienzahl         [0   ]          C-ontrolzeichen      [#   ]
S-eitenauswahl      [1     - 9999]
1 = Schriftart      [Normal    ]    2 = Zeichensatz      [NDR     ]
3 = Zeilenabstand   [1/6"      ]    4 = Seitenzahlen     [AUS     ]
5 = Einzelbatt      [AUS       ]    6 = Seitenvorschub   [am Ende ]

                    >>>> S T A T U S <<<<

Datei:  DOSMAIN.ASM         Größe:  40307   Kopien:  0

                    >>>> A K T I O N E N <<<<

A-bbrechen                          N-ormieren des Druckers
D-rucken                            V-orschub auf nächste Seite
T-extdatei laden                    Z-eilenvorschub
------------------------------------------------------------------------------
```

Unter VOREINSTELLUNGEN können verschiedene Parameter eingestellt werden. Unter STATUS steht der aktuelle Dateiname sowie die jeweils noch zu druckende Anzahl Zeichen und Anzahl Kopien. Unter AKTIONEN stehen die zur Verfügung gestellten Programmaktionen.

Im Einzelnen:

VOREINSTELLUNGEN

"F" --> Einstellen der Formularlänge, d.h. Anzahl der Druckzeilen je Seite, nach der automatisch ein Seitenvorschub erfolgt.
Standardeinstellung: 65 Zeilen.
Erlaubter Bereich..: 10 bis 127.

"L" --> Einstellen des linken Randes, d.h. Anzahl der Spalten, die am Zeilenanfang frei gelassen werden.
Standardeinstellung: 8 Zeichen.
Erlaubter Bereich..: 0 bis 127.

"K" --> Anzahl der Druckkopien. So bedeutet die 0, daß nur ein Original gedruckt wird.
Standardeinstellung: 0 Kopien.
Erlaubter Bereich..: 0 bis 99.

"B" --> Einstellen der ersten Seitennummer wenn mit Seitenzahlen gedruckt wird.
Standardeinstellung: Seite 1.
Erlaubter Bereich..: 1 bis 999.

"P" --> Spaltenposition der Seitenzahl. Damit kann man einstellen, ob die Seitenzahl links, rechts oder in der Mitte gedruckt wird.
Standardeinstellung: Spalte 40.
Erlaubter Bereich..: 1 bis 240.

"C" --> Einstellen des Steuerzeichens für die Druckeroptionen. Das Steuerzeichen plus ein Aktionszeichen werden nicht gedruckt sondern dienen zur Einstellung einer speziellen Druckeroption. Die Druckeroptionen werden weiter unten erläutert.
Standardeinstellung: #
Erlaubte Zeichen...: alle druckbaren Zeichen.

"S" --> Hiermit kann man auswählen, welche Seiten der Textdatei ausgedruckt werden sollen. Somit kann man, falls erforderlich, nur einen Ausschnitt der Datei drucken. Die beiden Seitenzahlen sind nacheinander einzugeben und jeweils mit <RETURN> abzuschließen.
Standardeinstellung: 1 bis 999.
Erlaubter Bereich..: 1 bis 999.

"1" --> Auswahl der Schriftart. Mit jedem Tastendruck "1" wird eine andere Schriftart eingestellt.
Standardeinstellung..: Normal.
Weitere Einstellungen: Elite und Schmal.

"2" --> Auswahl des Zeichensatzes. Mit jedem Tastendruck "2" wird ein anderer Zeichensatz eingestellt.
Standardeinstellung..: NDR.
Weitere Einstellungen: Deutsch und ASCII.

"3" --> Auswahl des Zeilenabstandes. Mit jedem Tastendruck "3" wird ein anderer Zeilenabstand eingestellt.
Standardeinstellung..: 1/6 Zoll.
Weitere Einstellungen: 1/8" und 7/72".

"4" --> Einstellung, ob mit oder ohne Seitenzahlen gedruckt werden soll.
Standardeinstellung....: AUS.
Alternative Einstellung: AN.

"5" --> Einstellung, ob mit oder ohne Einzelblatteinzug gearbeitet werden soll.
Standardeinstellung....: AUS.
Alternative Einstellung: AN.

"6" --> Einstellung des Seitenvorschubs. Mit jedem Tastendruck "6" wird eine andere Einstellung aktiviert.
Standardeinstellung..: Seitenvorschub am Ende des Druckvorgangs.
Weitere Einstellungen: Kein Seitenvorschub und Seitenvorschub am Anfang des Druckvorgangs.

AKTIONEN

"N" --> Normieren des Druckers. Dies bewirkt das Gleiche wie Ein- und Ausschalten des Druckers; auch Fernreset genannt.

"V" --> Seitenvorschub auslösen.

"Z" --> Zeilenvorschub auslösen.

"A" --> Beenden des Programms.

"D" --> Starten des Druckvorgangs mit den aktuellen Einstellungen. Der Druckvorgang wird aber nur dann ausgelöst, wenn eine Textdatei geladen wurde.

"T" --> Textdatei laden.

WICHTIG !! Wenn man den Druckvorgang vorzeitig abbrechen will, so muß man Ctrl C eintippen.

Steuercodes: Im Editor können nur ASCII-Zeichen zwischen $20 und $7E eingegeben werden. Zur Druckersteuerung wird hier eine Zeichenkombination, bestehend aus einem Steuerbuchstaben - standardmäßig "#" - und einem Aktionszeichen verwendet. Die folgende Tabelle zeigt die verwendbaren Aktionszeichen und die entsprechenden Druckeraktionen:

Aktionszeichen	Druckeraktion
0	Einzelblatt AUS
1	Einzelblatt EIN
6	Zeilenabstand 1/6"
7	Zeilenabstand 7/72"
8	Zeilenabstand 1/8"
@	Fern-RESET
A	ASCII-Zeichensatz
B	Breitschrift EIN
b	Breitschrift AUS
C	Kursivschrift EIN
c	Kursivschrift AUS
D	Doppeldruck EIN
d	Doppeldruck AUS
E	Elite - Schriftart
F	Fettdruck EIN
f	Fettdruck AUS
G	Deutscher Zeichensatz
g	Glocke ertönen lassen
H	Halbe Druckgeschwindigkeit EIN
h	Halbe Druckgeschwindigkeit AUS
I	Indizierung EIN
i	Indizierung AUS
L	Linken Rand auf aktuellen Wert
l	Linken Rand auf Null
N	Normal - Schriftart (PICA)
P	Potenzierung EIN
p	Potenzierung AUS
R	Ein Zeichen zurück
S	Schmalschrift EIN
s	Schmalschrift AUS
U	Unterstreichen EIN
u	Unterstreichen AUS
V	Seitenvorschub
Z	Zeilenvorschub

A.1.4) **DISKCOPY**

Zweck: Programm zum schnellen Kopieren kompletter Disketten mit einem oder mehreren Laufwerken
Aufruf: DISKCOPY
Codeumfang: 3 KByte
Startadresse: aktuelle Ladeadresse

Bedienungsablauf:
Das Programm meldet sich mit seinem Namen, der Versionsnummer und dem Copyright.

```
DISKETTEN-KOPIERER V 2.1   (C) 1986 by K.Janßen
```

Anschließend erscheint folgendes Menü:

```
K O P I E R  -  M E N U E
1 --->  KOPIEREN
9 --->  beenden
```

Mit "1" oder RETURN wird der Kopiervorgang eingeleitet, mit "9" wird er beendet. Nach dem Starten werden Quell- und Ziellaufwerk abgefragt:

```
Bitte Quellaufwerk angeben (1..4,ESC=Abbruch)
Bitte Ziellaufwerk angeben (1..4,ESC=Abbruch)
```

Erlaubt sind jeweils die Laufwerksnummern 1 bis 4. Mit ESC kann das Programm abgebrochen werden.

Nachdem Quell - und Ziellaufwerk bestimmt sind, untersucht das Programm ob die Laufwerksnummern gleich oder ungleich sind.

a) ungleiche Laufwerksnummern:

Es erscheinen folgende Meldungen:

```
Quelldiskette einlegen in Laufwerk  x
 Zieldiskette einlegen in Laufwerk  y

Wenn fertig -->
LEERTASTE DRUECKEN
```

Nach Drücken der Leertaste wird die Quelldiskette auf die Zieldiskette kopiert. Dabei wird der freie Benutzerspeicher komplett ausgenutzt. Der Kopiervorgang efolgt in "Highspeed" und benötigt etwa 70 sec.

Nach erfolgreicher Kopie erscheint die Meldung:

```
Diskette kopiert !!
```

ansonsten die Fehlermeldung:

```
Fehler beim Kopieren -->> ABBRUCH !!
```

b) gleiche Laufwerksnummern

Es erscheint folgende Meldung:

```
Zahl der Diskettenwechsel: nn

Quelldiskette einlegen
Wenn fertig -->
LEERTASTE DRUECKEN
```

Nach Drücken der Leertaste liest das Programm den Inhalt der Quelldiskette soweit der freie Benutzerspeicher ausreicht. Anschießend erscheint die Meldung:

```
Zahl der Diskettenwechsel: nn-1

Zieldiskette einlegen
Wenn fertig -->
LEERTASTE DRUECKEN
```

Nach Drücken der Leertaste wird der Speicherinhalt auf die Zieldiskette geschrieben. Anschließend wird die Zahl der Diskettenwechsel dekrementiert und es erscheint wieder die Aufforderung, die Quelldiskette einzulegen. Dies erfolgt so oft, bis die Diskette komplett kopiert ist. Je mehr freien Benutzerspeicher man hat, desto geringer ist die Zahl der Diskettenwechsel.

Nach erfolgreicher Kopie erscheint die Meldung:

```
Diskette kopiert !!
```

ansonsten die Fehlermeldung:

```
Fehler beim Kopieren -->> ABBRUCH !!
```

A.1.5) **DSAVE**

Zweck: Programm zum Abspeichern eines Speicherbereichs (z.B. im EPROM) auf Diskette
Aufruf: DSAVE
Codeumfang: 1 KByte
Startadresse: aktuelle Ladeadresse

Bedienungsablauf:
Das Programm meldet sich mit seinem Namen, der Versionsnummer und dem Copyright.

```
DATA SAVE V 2.1   (C) 1986 by K.Janßen
```

a) mit Parametern:

1. Parameter: Name der Datei
2. Parameter: Startadresse
3. Parameter: Anzahl KByte

Nach dem Speichern meldet DSAVE:

```
Abgespeichert !!
```

Beispiel für einen Aufruf mit Parametern:

```
DSAVE GRUND43.ROM $E0000 32
```

b) ohne Parameter:

Erst wird nach dem Dateinamen gefragt:

```
Dateiname eingeben
```

Danach wird die Startadresse festgelegt:

```
WELCHE STARTADRESSE ? (DEFAULT = $000000)
```

Daran anschließend muß noch festgelegt werden, wieviel KByte abgespeichert werden sollen:

```
wieviel KByte speichern ?
```

Ist auch dies festgelegt, wird der so definierte Speicherbereich auf die Diskette geschrieben. DSAVE meldet dann:

```
Abgespeichert !!
```

Folgende Fehlermeldungen können auftauchen:

```
Kein Platz auf Diskette
Diskettenfehler beim Schreibzugriff
Fehler bei der Namenseingabe
Anzahl KByte muß zwischen 1 und 780 sein
```

A.1.6) FORMAT

Zweck: Programm zum Formatieren von Disketten
Aufruf: FORMAT
Codeumfang: 3 KByte
Startadresse: aktuelle Ladeadresse

Bedienungsablauf:
Nach dem Programmstart erscheint wie üblich das Copyright:

```
FORMATIERER V 2.20 für Format NDR80   (C) 1986 by K.Janßen
```

Danach wird dazu aufgefordert, die alte Diskette zu entnehmen und das Laufwerk auszuwählen:

```
Alte Diskette bitte entnehmen

----- Laufwerk auswählen -----
      1 ---> Laufwerk 1
      2 ---> Laufwerk 2
      3 ---> Laufwerk 3
      4 ---> Laufwerk 4
      9 ---> Beenden
```

Nach Wahl des Laufwerks wird die Laufwerksnummer bestätigt und dann der Benutzer aufgefordert, eine neue Diskette in das gewählte Laufwerk einzulegen:

```
Bitte neue Diskette einlegen

Formatieren starten (J/N) ?
```

Mit Eingabe von "J" für JA beginnt der Formatiervorgang, ansonsten erscheint wieder das Laufwerkmenü.

Der Formatiervorgang wird durch Anzeige der aktuellen Spurnummer dokumentiert:

```
Diskette wird formatiert   --  Spur:  XX
```

Die Spurnummer beginnt mit 00 und endet mit 79. Es werden beide Seiten formatiert.

An das Formatieren schließt sich ein Prüflesevorgang an:

```
Diskette wird geprüft      --  Spur:  XX
```

Wurde beim Prüflesen kein Fehler entdeckt, so erscheint die Meldung

```
Formatierung ok
```

Im Fehlerfall können folgende Meldungen erscheinen:

Fehler beim Formatieren
Laufwerksfehler
Zu wenig Speicher
Fehler beim Prüflauf entdeckt

Nach einem solchen Fehlerfall erscheint erneut das Eingangsmenü.
Seit V 2.20 werden 4 Laufwerke unterstützt. Der Prüflesevorgang benötigt nur noch die halbe Zeit. Nach dem Hochfahren der Laufwerkmotoren wird sicherheitshalber 500msec gewartet.

A.1.7) GP

Zweck: Interface zum Grundprogramm
Aufruf: GP
Codeumfang: 2 KByte
Startadresse: am Ende des freien Benutzerspeichers

Bedienungsablauf:

Nach dem Start meldet sich folgender Bildschirm:

```
------GRUNDPROGRAMM-INTERFACE V 2.00------

--------------  HAUPT - MENUE  -----------

 A = Aendern              I = EPROM progr.
 B = Starten              J = EPROM lesen
 C = Ansehen              K = Speicherbereiche
 D = Symboltabelle        L = Text drucken
 E = Editor               M = IO lesen
 F = Assembler            N = IO setzen
 G = Bibliothek           O = Einzelschritt
 H = Optionen             Z = BEENDEN
```

Durch Eingeben des entsprechenden Buchstabens wird die angezeigte Aktion ausgeführt. Die Bedienung ist ausführlich im Grundprogramm von H. Klein beschrieben.
Mit "Z" wird das Programm beendet und man befindet sich wieder im JADOS. Mit "H" wird ein Untermenue aufgerufen, indem man Optionen einstellen kann. Das entsprechende Menü sieht wie folgt aus:

```
------GRUNDPROGRAMM-INTERFACE V 2.00------

---------------  OPTIONEN  ---------------

 A = Nur Fehlerausg.   G = Symbole loeschen
 B = Ausgabe auf CRT   H = 40 Zeichen/Zeile
 C = Ausgabe auf LST   I = 80 Zeichen/Zeile
 D = wie C ohne LF     J = Debug-Info AN
 E = Textstart alt     K = Debug-Info AUS
 F = Textstart neu     Z = Grundmenue
```

Mit "A" bis "K" können die angezeigten Optionen eingestellt werden.
Mit "Z" kommt man wieder zum Hauptmenü zurück.

Dieses Programm steht als Quelltext zur Verfügung damit der Anwender Erweiterungen und Anpassungen an evtl. geänderte Grundprogramme machen kann. Das Programm benutzt direkte Einsprungadressen in das Original-Grundprogramm V 4.3 von H. Klein.

A.1.8) MORE

Zweck: Programm zur superschnellen Anzeige von Textdateien
Aufruf: MORE
Codeumfang: 2 KByte
Startadresse: aktuelle Ladeadresse

Bedienungsablauf:

Nach dem Programmstart erfolgt die Meldung:

SCHNELLE DATEIANZEIGE V 2.20 (C) 1986 by P.Gerl, A.Granel, K.Janßen

a) mit Parameter:

1. Parameter: Name der Textdatei

b) ohne Parameter:

Der Benutzer wird aufgefordert:

Bitte Dateinamen eingeben:

Danach wird die Textdatei in den Hauptspeicher geladen und die erste Seite Text angezeigt. Die Textausgabe ist "superschnell". Seit V 2.20 werden auch deutsche Sonderzeichen ausgegeben.

Mit jeder beliebigen Taste kann die Anzeige der nächsten Seite (etwa 22 Zeilen) erreicht werden. Am unteren Ende des Bildschirms erscheint jeweils eine Meldung, ob die Anfangsseite, die Schlußseite oder eine beliebige Seite dazwischen angezeigt wird. Mit der Taste "-" kann man eine halbe Seite (10 Zeilen) zurück !!

Wenn am unteren Bildschirmrand *** Ende *** angezeigt wird, so kann man mit dem nächsten Tastendruck wieder nach JADOS zurück.
Mit Ctrl C kann das Programm abgebrochen werden.

A.1.9) **ROMSTART**

Zweck: Programm zum Suchen und Starten von Bibliotheksprogrammen, die im Hauptspeicher stehen
Aufruf: ROMSTART
Codeumfang: 2 KByte
Startadresse: aktuelle Ladeadresse

Bedienungsablauf:
Nach dem Programmstart wird folgender Bildschirm angezeigt:

```
------------------------------------------------------------------------------
BIBLIOTHEKSFUNKTION  V 2.1    (C) 1986,1987 BY Klaus Janßen

   --NAME--     -START-     -LAENGE-
   INSPEC       03F420      001F92
   DISASS       044C20      002A86
   BRDRUCK      0480DC      001378
   JADOS        04B020      0035D4




"+" --> nächster Eintrag , "-" --> voriger Eintrag  "E" --> Tabellenende
"A" --> Tabellenanfang   , "S" --> Starten          ESC --> Abbrechen
------------------------------------------------------------------------------
```

Die Suche nach Bibliotheksprogrammen erfolgt in Schritten von 32 Byte. Es werden maximal 127 Einträge verwaltet, von denen jeweils 16 angezeigt werden. Die unterschiedliche Adreßkapazität der CPUs wird hierbei berücksichtigt.

Der Cursor zeigt auf den gerade aktuellen Eintrag.
Mit "+" wird der Cursor zum nächsten Eintrag bewegt. Dabei werden automatisch die nächsten 16 Einträge angezeigt, falls der Cursor am unteren Ende stand.
Mit "-" wird der Cursor zum vorigen Eintrag bewegt.
Mit "E" wird der Cursor zum letzten Eintrag überhaupt bewegt.
Mit "A" wird der Cursor zum ersten Eintrag bewegt.
Mit "S" wird das vom Cursor markierte Programm gestartet. JADOS selber läßt sich auf diese Weise nicht starten. Vor dem Start wird der Bildschirm gelöscht, die 2-Seiten-Umschaltung abgeschaltet und Seite 0 aktiv gesetzt.
Mit ESC kann das Programm abgebrochen werden.

A.1.10) **SYS**

Zweck: Programm zum Kopieren der Systemspuren, falls sich auf den Systemspuren ein JADOS-System befindet.
Aufruf: SYS
Codeumfang: 2 KByte
Startadresse: aktuelle Ladeadresse

Bedienungsablauf:

Nach dem Start wird der Bildschirm gelöscht, und es erscheint die Meldung:

```
SYSTEM-ÜBERTRAGUNG V 2.1    (C) 1987 by K.Janßen
```

a) mit Parameter:

1. Parameter: Nummer des Quellaufwerks
2. Parameter: Nummer des Ziellaufwerks

b) ohne Parameter:

Zuerst wird nach der Nummer des Quellaufwerks gefragt:

```
Bitte Quellaufwerk angeben (1..4,ESC=Abbruch):
```

Erlaubt sind 1 bis 4. ESC bewirkt den Abbruch des Programms.
Danach wird nach der Nummer des Ziellaufwerks gefragt:

```
Bitte Ziellaufwerk angeben (1..4,ESC=Abbruch):
```

Hier sind wieder 1 bis 4 erlaubt.

Nachdem Quell - und Ziellaufwerk bestimmt sind, untersucht das Programm ob die Laufwerksnummern gleich oder ungleich sind.

a) ungleiche Laufwerksnummern:

Es erscheinen folgende Meldungen:

```
Quelldiskette einlegen in Laufwerk  x
 Zieldiskette einlegen in Laufwerk  y

Wenn fertig -->
LEERTASTE DRUECKEN
```

Nach Drücken der Leertaste liest das Programm die Systemspuren der Quelldiskette und prüft, ob es sich um ein JADOS-System handelt. Wenn nicht, kommt die Meldung:

```
Kein SYSTEM vorhanden --> Abbruch !!
```

Wenn sich doch ein System auf den reservierten Spuren befindet, dann meldet SYS:

```
SYSTEM Version  2.10
```

und kopiert das System auf die reservierten Spuren der Zieldiskette.

b) gleiche Laufwerksnummern

Es erscheint folgende Meldung:

```
Quelldiskette einlegen in Laufwerk  x

Wenn fertig -->
LEERTASTE DRUECKEN
```

Nach Drücken der Leertaste liest das Programm die Systemspuren der Quelldiskette und prüft, ob es sich um ein JADOS-System handelt. Wenn nicht, kommt die Meldung:

```
Kein SYSTEM vorhanden --> Abbruch !!
```

Wenn sich doch ein System auf den reservierten Spuren befindet, dann meldet SYS:

```
SYSTEM Version  2.10

Zieldiskette einlegen in Laufwerk  x

Wenn fertig -->
LEERTASTE DRUECKEN
```

und kopiert das System auf die reservierten Spuren der Zieldiskette.

A.2) ABHILFE BEI FEHLERMELDUNGEN

- ABBRUCH WEGEN LESEFEHLER

  -> Im Kommando ASS und TLOAD erscheint diese Meldung, falls beim menügesteuerten Laden von Texten ein Fehler auftritt.

  Abhilfe: Neu versuchen !

- ACHTUNG !! DIESER TEXT IST GRÖSSER ALS 64 KBYTE
  ER KANN NICHT MIT DEM RDK-EDITOR BEARBEITET WERDEN

  -> Im Kommando EDIT erscheint diese Meldung, falls Textdateien bearbeitet werden sollen, die größer als 64 KByte sind.

  Abhilfe: Textdatei mit TLOAD laden und im Grundprogramm durch Eingabe einer sedezimalen Null an geeigneter Stelle verkleinern.

- BIBLIOTHEKSPROGRAMM NICHT RELOKATIV

  -> Beim Laden eines ausführbaren Programms mit Bibliotheksvorspann erscheint diese Meldung, falls das Programm als nicht relokativ gekennzeichnet ist.

  Abhilfe: Programm relokativ schreiben und dies im Bibliotheksvorspann kennzeichnen.
  Programm mit TLOAD auf die absolute Adresse laden, die im Bibliotheksvorspann steht und im Grundprogramm starten.

- BITTE MIT FLOPPYSTART BOOTEN

  -> Diese Meldung erscheint, wenn JADOS beim Start feststellt, daß es nicht auf der Adresse beginnt, die beim Floppystart festgelegt wird.

  Abhilfe: Floppystart benutzen

- DATEINAME EXISTIERT BEREITS

  -> Im Kommando REN erscheint diese Meldung, falls der neue Name bereits existiert.

  Abhilfe: Datei mit neuem Namen umbenennen oder löschen.

- DATEI NICHT GEFUNDEN

  -> Bei vielen Kommandos, die eine Datei bearbeiten, erscheint diese Meldung, falls die Datei nicht existiert.

  Abhilfe: Eine andere Diskette nehmen und den Versuch wiederholen.
  Schreibweise des Dateinamens überprüfen.

- DIES IST KEIN PROGRAMM

  -> Wenn die resident zu installierende Datei nicht die Endung .68K oder .COM besitzt, erscheint diese Meldung.

  Abhilfe: Nur Programmdateien resident installieren.

- DISKETTENFEHLER ODER SCHREIBSCHUTZ GESETZT !

  -> Falls ein Schreibzugriff auf eine schreibgeschützte Diskette stattfindet oder beim Lesen ein CRC- oder Datenfehler festgestellt wurde, erscheint diese Meldung.

  Abhilfe: Schreibschutz entfernen oder Diskette überprüfen.

- FALSCHE ORG-ANWEISUNGEN BENUTZT

  -> Im Kommado ASS erscheint diese Meldung, falls man ORG-Anweisungen benutzt.

  Abhilfe: Alle ORG-Anweisungen aus der Programmquelle entfernen.

- FALSCHES KOMMANDO

  -> Diese Meldung erscheint, wenn ein eingegebenes Kommando weder ein Internes noch ein Speicherresidentes ist und auch nicht als .68K-, .COM- oder .BAT-Datei existiert.

  Abhilfe: Eine andere Diskette einlegen und Kommando wiederholen.
  Schreibweise des Kommandonamens überprüfen.

- FEHLER BEI DER NAMENSEINGABE

  -> Bei vielen Kommandos, die die Eingabe eines Dateinamens erfordern, erscheint diese Meldung. Meist wurde dann gar kein Name eingegeben oder eine falsche Laufwerksangabe gemacht.

  Abhilfe: Namen korrekt eingeben.

- FEHLER BEIM SPEICHERN DER PROGRAMMDATEI

  -> Diese Meldung erscheint im Kommando ASS, falls die Abspeicherung des Maschinencodes fehlschlägt.

  Abhilfe: Diskette überprüfen.

- INSTALLATIONSTABELLE VOLL

  -> Die Meldung erscheint, wenn versucht wird, mehr als 20 Programme speicherresident zu installieren.

- KEIN PLATZ MEHR AUF DER DISKETTE

  -> Wenn die Diskette keine freien Spuren mehr enthält und eine neue Spur angelegt werden soll, dann erscheint diese Meldung.

  Abhilfe: Eine neue Diskette nehmen oder nicht mehr gebrauchte Dateien löschen.

- KEIN PLATZ MEHR IM INHALTSVERZEICHNIS

  -> Wenn die Diskette bereits 128 Dateieinträge enthält und eine neue Datei angelegt werden soll, dann erscheint diese Meldung.

  Abhilfe: Eine neue Diskette nehmen oder nicht mehr gebrauchte Dateien löschen.

- KEIN SPEICHERPLATZ MEHR

  -> Die Meldung erscheint, wenn JADOS beim Installieren eines residenten Programmes feststellt, daß der freie Benutzerspeicher nicht ausreicht.

- LADEFEHLER

  -> Falls im Kommando ASS beim Laden eines Quellprogramms ein Fehler auftritt, erscheint diese Meldung.

  Abhilfe: Diskette prüfen.

- PARAMETERFEHLER

  -> Falls das Kommando ASS mit einem ungültigen Parameter aufgerufen wird, erscheint diese Meldung.

  Abhilfe: Parameter korrekt eingeben.

- PROGRAMM NICHT VORHANDEN

  -> Beim Installieren eines residenten Programmes erscheint diese Meldung, wenn das Programm nicht vorhanden ist.

  Abhilfe: Schreibweise des Dateinamens prüfen oder andere Diskette nehmen.

- SPEICHER VOLL

  -> Diese Meldung erscheint immer dann, wenn eine Datei nicht mehr in den freien Benutzerspeicher paßt.

  Abhilfe: Benutzerspeicher vergrößern oder Datei verkleinern. Eventuell weniger Programme speicherresident halten.

- STAPELVERARBEITUNG ABBRECHEN (J/N) ?

  -> Falls bei der Abarbeitung einer Kommandodatei eine Taste gedrückt wird, erscheint diese Meldung. Der Benutzer kann sich dann entscheiden, ob er die Stapelverarbeitung abbrechen oder fortsetzen will.

- TEXT WEGEN SCHREIBFEHLER NICHT GESPEICHERT

  -> Diese Meldung erscheint im Kommando TSAVE, falls beim Speichern des Textes ein Fehler auftritt.

  Abhilfe: Diskette prüfen.

- UMBENENNNUNG NUR AUF GLEICHEM LAUFWERK !

  -> Im Kommando REN erscheint diese Meldung, falls für den alten und neuen Namen unterschiedliche Laufwerksangaben gemacht werden.

  Abhilfe: Gleiche Laufwerksangaben machen.

- UNDEFINIERTER DISKETTENFEHLER

  -> Diese Meldung kann erscheinen, falls ein undefinierter Diskettenfehler auftritt.

  Abhilfe: Diskette prüfen.

- UNDEFINIERTER LESEFEHLER

  -> Diese Meldung kann erscheinenm, falls beim Lesen von einer Diskette ein Fehler auftritt.

  Abhilfe: Diskette prüfen.

- ZU WENIG SPEICHER !!

  -> Diese Meldung erscheint nach dem Start von JADOS, falls kein freier Benutzerspeicher vorhanden ist.

  Abhilfe: Ersten zusammenhängenden Speicherbereich ausbauen.

A.3) AUSNAHMEVERARBEITUNG

Bei Computerkonfigurationen, die ab Adresse $0000 RAM-Speicher aufweisen, können die Einsprungadressen von Interrupts verändert werden. Davon macht JADOS seit Version 2.10 Gebrauch. Der Vorteil ist, daß das JADOS-System nicht verlassen wird, sondern die Programmkontrolle behält bzw. erhält. Es muß also nicht jedesmal neu gebootet werden. Allerdings wird ein sogenannter Kaltstart durchgeführt, bei dem alle speicherresidenten Programme verloren gehen.

Folgende Ausnahmesituationen werden erfaßt:

>>>> ADRESS-FEHLER <<<<

Dies tritt immer dann auf, wenn ein Wort- oder Langwortzugriff auf eine ungerade Adresse stattfindet.

>>>> FALSCHER BEFEHL <<<<

Dies tritt auf, wenn die CPU auf einen nicht vorhandenen Befehl (Opcode) trifft.

>>>> DIVISION durch 0 <<<<

Dies tritt auf, wenn der Divisor beim DIVS- oder DIVU-Befehl Null ist.

>>>> CHK-Befehl <<<<

Dies erscheint, wenn der CHK-Befehl benutzt wird und der Operand außerhalb der Grenzen liegt, gegen die geprüft wird.

>>>> TRAPV-Befehl <<<<

Dies erscheint, wenn der TRAPV-Befehl ausgeführt wird und das Überlauf-Flag gesetzt ist.

>>>> PRIVILEG-Verletzung <<<<

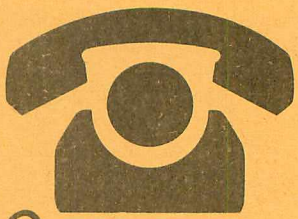
Wenn die CPU im User-Modus arbeitet, sind einige Befehle nicht zugelassen. Werden sie dennoch benutzt, so kommt es zur sogenannten Privilegverletzung. Das Grundprogramm und JADOS arbeiten im System-Modus, in dem alle Befehle erlaubt sind. Auch die Anwenderprogramme laufen im System-Modus, so daß es zur obigen Meldung normalerweise nicht kommen kann.

>>>> Befehlscode A Emulator <<<<

Dies erscheint, wenn die höchsten vier Bits eines Befehls 1010 oder sedezimal $A sind. Dies ist vom Hersteller der CPU für spätere Erweiterungen gedacht.

>>>> Befehlscode F Emulator <<<<

Dies erscheint, wenn die höchsten vier Bits eines Befehls 1111 oder sedezimal $F sind. Dies sind alle Befehle des arithmetischen Co-prozessors (z.B. MC68881).



Telefonservice
08 31- 62 11
jeden Mittwochabend
bis 20.00 Uhr

Graf Elektronik Systeme GmbH
Magnusstraße 13 · Postfach 1610
8960 Kempten (Allgäu)
Telefon: (08 31) 62 11
Teletex: 831804 = GRAF
Telex: 17 831804 = GRAF
Datentelefon: (08 31) 6 93 30

Verkauf:
Computervilla
Ludwigstraße 18 b
(bei Möbel-Krügel)
8960 Kempten-Sankt Mang
Telefon: 08 31 / 6 93 00

Geschäftszeiten: GES GmbH + Verkauf
Mo. - Do. 8.00 - 12.00 Uhr, 13.00 - 17.00 Uhr
Freitag 8.00 - 12.00 Uhr
Telefonservice

Filiale Hamburg
Ehrenbergstraße 56
2000 Hamburg 50
Telefon: (0 40) 38 81 51

Filiale München:
Georgenstraße 61
8000 München 40
Telefon: (0 89) 2 71 58 58

Öffnungszeiten der Filialen:
Montag – Freitag
10.00 – 12.00 Uhr, 13.00 – 18.00 Uhr
Samstag 10.00 – 14.00 Uhr

ges

Hinweis zum JADOS-Handbuch V2.1

Das Handbuch JADOS V2.1 besteht aus dem Handbuch JADOS V2.0 und den Zusatzblättern JADOS V2.1. Sie müssen jetzt die Seiten 43 bis 74 aus dem JADOS V2.0 Handbuch entnehmen und die restlichen Seiten dieses Handbuches wegwerfen (jedenfalls weiter weglegen, um sie mit den Seiten des JADOS V2.1 nicht zu vertauschen. Jetzt setzen Sie die Seiten 43 bis 74 in das neue Handbuch JADOS V2.1 ein. Damit ist ihr JADOS Handbuch V2.1 komplett.

Sollten Sie ein JADOS-Update bekommen haben, so erhalten sie nur die Zusatzblätter zum JADOS V2.1. Die Seiten 43 bis 74 müssen sie aus ihrem alten JADOS-Handbuch V2.0 entnehmen.

ACHTUNG!

Achten sie bitte darauf, daß sie beim Austauschen der Blätter nicht die Seiten 0 bis 42 und 75 bis xxx der beiden JADOS-Versionen vertauschen.