J A D O S - T R A P S

JADOS bietet dem Programmierer zahlreiche nützliche Unterprogramme aus den Bereichen Stringverarbeitung, Dateibearbeitung, Dialogführung usw. . Der Aufrufmechanismus dieser Unterprogramme geschieht wie bei den Routinen des Grundprogramms über einen Trap. Während das Grundprogramm den Trap #1 verwendet, ist bei JADOS der Trap #6 zu benutzen. Im Register D7 muß die Nummer des Unterprogramms übergeben werden. Weitere Parameter werden ebenfalls in Registern ausgetauscht. In den einzelnen Beschreibungen werden diese Parameter exakt erklärt.

Da im JADOS alle Variabeln relativ zum Register A6 adressiert werden, sollte dieses in eigenen Programmen nicht benutzt werden. Beim Aufruf eines Unterprogramms setzt JADOS nämlich A6 auf den erforderlichen Wert.

Die Beschreibung der Unterprogramme ist in Anlehnung an das Grundprogramm nach einem einheitlichen Schema aufgebaut. Hierbei werden folgende Informationen gegeben:

TRAP-Nummer	: Unter dieser Nummer im Register D7 kann das Unterprogramm aufgerufen werden.
Name	: Name des Unterprogramms
Befehlsgruppe	: Gruppe, zu der das Unterprogramm gehört
Kurzbeschreib.	: Kurze Erklärung, was das Unterprogramm leistet
Eingabereg.	: Hier sind alle Register verzeichnet, die beim Aufruf mit den richtigen Werten zu laden sind. Nicht benutzte Bits sind zu Null zu setzen.
Ausgabereg.	: Hier werden die Register verzeichnet, die Ergebnisse nach dem Aufruf beinhalten. Die Registerbreite wird mit angegeben.
Zerstörte Reg.	: Hier werden die Register verzeichnet, die durch das Unterprogramm zerstört werden. Sie sollten deshalb gegebenenfalls vorher gerettet werden.
Ab Version	: Das Unterprogramm steht ab dieser Version zur Verfügung
Änderungen	: Wichtige Änderungen zu vorhergehenden Versionen werden hier aufgeführt.

Es folgt eine ausführliche Beschreibung des Unterprogramms mit Hinweisen und Programmierbeispielen. Für die Richtigkeit und Fehlerfreiheit der Beispiele wird keine Garantie übernommen.

Nach den Einzelbeschreibungen werden noch verschiedene Zusammenstellungen nach Nummern, alphabetisch und nach Gruppen sortiert aufgelistet.

TRAP-Nummer : 0
Name : getleng
Befehlsgruppe : Stringhandling
Kurzbeschreib.: Länge eines Textes in Bytes ermitteln

Eingabereg. : a0.l = Zeiger auf den Textanfang
Ausgabereg. : d1.w = Anzahl der Bytes
Zerstörte Reg.: keine
Ab Version : 1.2
Änderungen : keine

Der Befehl GETLENG beginnt ab der Startadresse mit dem Zählen der Bytes bis eine binäre Null gefunden wird durch die das Ende eines Textes markiert wird. Da als Ergebnis ein Wortregister verwendet wird, kann die Länge von Texten bis maximal 65535 Bytes korrekt ermittelt werden.

Beispiel:

```
start:  lea     text(pc),a0
        moveq   #0,d7
        trap    #6
        rts
text:   dc.b    'Hallo',0
```

Im Beispiel wird die Länge des Textes "Hallo" ermittelt. Das Ergebnis beträgt hier 5 Bytes.

```
TRAP-Nummer    : 1
Name           : getname
Befehlsgruppe  : Dialogführung
Kurzbeschreib.: Dateinamen erfragen und Dateisteuerblock initialisieren

Eingabereg.    : a0.l = Zeiger auf einen Hinweistext
                 a1.l = Zeiger auf einen Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = kein Fehler
                        $ff = Name falsch oder Eingabe abgebrochen (ESC)
Zerstörte Reg.: keine
Ab Version     : 1.2
Änderungen     : keine
```

Der Befehl GETNAME zeigt zunächst einen Hinweistext für den Benutzer an, der im Zusammenhang mit der anschließend folgenden Eingabe eines Dateinamens stehen sollte. Nach der Eingabe wird ein Dateisteuerblock angelegt, der eingegebene Name wird in Großbuchstaben umgewandelt und in den Steuerblock eingetragen. Danach können alle Dateioperationen mit diesem Steuerblock ausgeführt werden.
Die Eingabe des Namens kann vom Benutzer mit ESC abgebrochen werden. In diesem Fall wird ein Fehlercode erzeugt und kein Steuerblock angelegt. Falls die Eingabe des Benutzers nicht als gültiger Dateiname interpretiert werden kann, wird ebenfalls ein Fehlercode erzeugt.

Beispiel:

```
        start:  lea     text(pc),a0
                lea     fcb(pc),a1
                moveq   #1,d7
                trap    #6
                rts
        text:   dc.b    'Bitte Dateinamen eingeben: ',0
                ds      0
        fcb:    ds.b    48
```

Im Beispiel wird der Hinweistext "Bitte Dateinamen eingeben: " angezeigt. Der Cursor steht hinter dem Text und der Benutzer kann nun den Namen eingeben. Nach korrekter Eingabe enthält "fcb" den eingegebenen Namen.

TRAP-Nummer : 2
Name : getstadd
Befehlsgruppe : Dialogführung
Kurzbeschreib.: Startadresse für die Ablage von Texten erfragen

Eingabereg. : a2.l = Vorgegebene Startadresse
Ausgabereg. : a0.l = Eingegebene Startadresse
Zerstörte Reg.: keine
Ab Version : 1.2
Änderungen : keine

Der Befehl GETSTADD zeigt einen festen Text an, der den Benutzer auffordert, eine Startadresse einzugeben. Die vorgegebene Adresse wird ebenfalls angezeigt. Falls die Eingabe mit ESC abgebrochen wird oder die eingegebene Adresse kleiner ist als die vorgegebene Adresse, dann wird die vorgegebene Adresse als eingegebene Adresse übernommen; ansonsten wird die eingegebene Adresse übernommen.

Beispiel:

```
start:  lea     $10000,a2
        moveq   #2,d7
        trap    #6
        rts
```

Im Beispiel wird der Text "Welche Startadresse ? (Default = $010000)" angezeigt. Der Cursor steht hinter dem Text und der Benutzer kann nun eine Adresse angeben. Falls die Eingabe abgebrochen wird oder die Adresse kleiner ist als $10000, dann enthält a0 den Wert $10000.

```
TRAP-Nummer    : 3
Name           : lese
Befehlsgruppe  : Texteingabe
Kurzbeschreib.: Einlesen einer Textzeile über Tastatur

Eingabereg.    : d2.w = maximale Anzahl der Textzeichen
                 a1.l = Zeiger auf den Textpuffer
Ausgabereg.    : d0.b = Fehlercode
                          0 = Eingabe in Ordnung
                        $1b = Abbruch mit ESC
Zerstörte Reg.: keine
Ab Version     : 1.2
Änderungen     : Ab V 2.0: Kompletten Pufferinhalt anzeigen mit ^J
                 Ab V 2.1: Pufferinhalt byteweise anzeigen mit ^F
                           Zeichen im Puffer löschen mit ^G
                           Leerzeichen im Puffer einfügen mit ^V
                 Ab V 3.5: Abbruch nur noch mit ESC, nicht mehr mit ^A oder ^C
```

Der Befehl LESE dient zum Einlesen eines einzeiligen Textes über die Tastatur. Die Anzahl der Zeichen wird als Eingangsparameter vorgegeben. Der Textpuffer ist vom Anwender zu definieren. Die Eingabe wird mit RETURN korrekt beendet. Mit ESC kann die Eingabe abgebrochen werden. Dies wird im Fehlercode angezeigt. Bei der Texteingabe gibt es einige Korrekturmöglichkeiten. Mit BACKSPACE wird das Zeichen vor dem Cursor gelöscht und der Cursor nach links gezogen. Mit ^J wird der komplette Pufferinhalt angezeigt. Mit ^F wird das Zeichen unter dem Cursor aus dem Puffer sichtbar gemacht und der Cursor nach rechts gezogen. Mit ^G wird das Zeichen unter dem Cursor aus dem Puffer gelöscht. Mit ^V wird ein Leerzeichen in den Puffer eingefügt.

Intern in JADOS wird dieses Unterprogramm nur noch bei der Eingabe von Dateinamen und Startadressen verwendet. Die Kommandoeingabe geschieht mit dem moderneren Unterprogramm LINEEDIT.

Beispiel:

```
        start:  lea     text(pc),a1
                moveq   #10,d2
                moveq   #3,d7
                trap    #6
                rts
        text:   dc.b    'Hallo',0,0,0,0,0,0
```

Im Beispiel kann ein bis 10 Zeichen umfassender Text eingegeben werden. Im Puffer befindet sich bereits der Text "Hallo", der mit ^F oder ^J zeichenweise oder komplett abgerufen werden kann. Der Textpuffer muß immer um ein Zeichen größer definiert werden als die Zahl der Zeichen, die in D2 übergeben wird.

TRAP-Nummer : 4
Name : loadtext
Befehlsgruppe : Dialogführung
Kurzbeschreib.: Menügeführtes Einlesen von Textdateien

Eingabereg. : a2.l = vorgegebene Mindeststartadresse
Ausgabereg. : d0.b = Fehlercode
0 = Texte fehlerfrei geladen
1 = Funktion abgebrochen ohne Textladen
$ff = Fehler beim Dateizugriff
Zerstörte Reg.: keine
Ab Version : 1.2
Änderungen : Ab V 2.0: Die Namen der Textdateien können aus einer Antwortdatei entnommen werden.

Der Befehl LOADTEXT ist eine recht komplexe Funktion zum benutzergeführten Einlesen von Textdateien in den Hauptspeicher. Zuerst wird vom Benutzer die Startadresse im Hauptspeicher abgefragt, wobei eine Mindestadresse vorgegeben werden kann. Anschließend erscheint ein kleines Menü, bei dem der Benutzer auswählen kann, ob er eine Textdatei laden (Menüpunkt 1), die Texte aus einer Antwortdatei (Menüpunkt 2) laden oder die Funktion beenden will (Menüpunkt 9). Beim Laden aus einer Antwortdatei werden jeweils der Name der Textdatei und die Endadresse angezeigt.

Beispiel:

```
start:  lea     $10000,a2       * Mindeststartadresse
        moveq   #4,d7           * LOADTEXT
        trap    #6
        cmp.b   #0,d0
        bne.s   fehler          * Befehl nicht korrekt
        moveq   #!getstx,d7     * Startadresse abfragen
        trap    #1
        movea.l d0,a0
        moveq   #!lst,d7
        trap    #1
        moveq   #7,d7
        trap    #6
fehler: rts
```

Im Beispiel wird zunächst der Befehl LOADTEXT aufgerufen, wobei als Mindeststartadresse $10000 vorgegeben wird. Nach korrekter Beendigung des Befehls wird die Startadresse aus dem Grundprogramm abgefragt. Anschließend wird die Ausgabe auf den Drucker umgelenkt und mit dem Befehl SCHREIBE ausgedruckt.

TRAP-Nummer : 5
Name : motoroff
Befehlsgruppe : Laufwerke
Kurzbeschreib.: Diskettenmotoren abschalten

Eingabereg. : keine
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 1.2
Änderungen : keine

Der Befehl MOTOROFF schaltet die Antriebsmotoren der Diskettenlaufwerke ab. Bei einigen Floppycontrollern ist dies nötig. Modernere Controller schalten die Motoren automatisch ab.

Beispiel:

```
        start:  moveq   #5,d7
                trap    #6
                rts
```

TRAP-Nummer : 6
Name : response
Befehlsgruppe : Dialogführung
Kurzbeschreib.: Warten bis Leertaste gedrückt wird

Eingabereg. : keine
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 1.2
Änderungen : keine

Der Befehl RESPONSE zeigt den Hinweistext "Leertaste drücken" an und wartet dann solange bis die Leertaste gedrückt wird.

Beispiel:

```
start:  moveq   #6,d7
        trap    #6
        rts
```

TRAP-Nummer : 7
Name : schreibe
Befehlsgruppe : Textausgabe
Kurzbeschreib.: text anzeigen

Eingabereg. : a0.l = Zeiger auf Textanfang
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 1.2
Änderungen : keine

Der Befehl SCHREIBE gibt einen Text, der mit einer binären Null abgeschlossen ist, über die Schnittstelle CO2 aus (siehe Grundprogramm). Normalerweise ist die Bildschirmanzeige aktiviert. Durch vorheriges Umschalten auf den Drucker mit LST kann der Text aber auch ausgedruckt werden.

Beispiel:

```
        start:  lea     text(pc),a0
                moveq   #!crt,d7
                trap    #1
                moveq   #7,d7
                trap    #6
                rts
        text:   dc.b    'Hallo',0
```

Im Beispiel wird der Text "Hallo" angezeigt.

```
TRAP-Nummer    : 8
Name           : strgcomp
Befehlsgruppe  : Stringhandling
Kurzbeschreib.: Strings vergleichen

Eingabereg.    : a0.l = Zeiger auf den ersten String
                 a1.l = Zeiger auf den zweiten String
Ausgabereg.    : d0.b = Vergleichsergebnis
                          0 = Strings sind gleich
                        $ff = Strings sind ungleich
Zerstörte Reg.: keine
Ab Version     : 1.2
Änderungen     : keine
```

Der Befehl STRGCOMP vergleicht zwei Textstrings, die mit einer binären Null abgeschlossen sind.

Beispiel:

```
        start:  lea     str1(pc),a0
                lea     str2(pc),a1
                moveq   #8,d7
                trap    #6
                cmp.b   #0,d0
                bne.s   ungl
        gleich: lea     gltext(pc),a0
                bra.s   ausgab
        ungl:   lea     ugltxt(pc),a0
        ausgab: moveq   #7,d7
                trap    #6
                rts
        str1:   dc.b    'Hallo',0
        str2:   dc.b    'HALLO',0
        gltext: dc.b    'Strings sind gleich',13,10,0
        ugltxt: dc.b    'Strings sind ungleich',13,10,0
```

Im Beispiel werden die Textstrings str1 und str2 auf Gleichheit geprüft. Das Ergebnis des Vergleichs wird im Klartext angezeigt.

```
TRAP-Nummer    : 9
Name           : tload
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Textdatei laden

Eingabereg.    : a0.l = Adresse ab der der Text geladen werden soll
                 a1.l = Zeiger auf einen Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = Textladen war erfolgreich
                        $ff = nicht erfolgreich
                 a0.l = Adresse auf das erste Byte hinter dem geladenen Text
Zerstörte Reg.: keine
Ab Version     : 1.2
Änderungen     : Ab V 2.0: Wesentlich höhere Ladegeschwindigkeit
```

Mit dem Befehl TLOAD wird eine Textdatei in den Hauptspeicher geladen. Im Unterschied zu LOADTEXT erfolgt hier keinerlei Benutzerführung. Im Register A0 wird die Adresse im Hauptspeicher übergeben, ab der der Text geladen werden soll. Nach dem Ladevorgang zeigt A0 auf das nächste Byte hinter dem geladenen Text. Durch fortgesetzte TLOAD-Befehle können so mehrere Texte hintereinander geladen werden, ohne daß jeweils die Adresse neu berechnet werden muß. Nach dem Laden wird auch die Grundprogrammvariable STXTXT auf die Startadresse gesetzt. Vor dem Aufruf der Funktion muß ein Dateisteuerblock eingerichtet sein.

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7          * FILLFCB
                trap    #6
                cmp.b   #0,d0
                bne.s   fehler
                lea     $10000,a0
                moveq   #9,d7           * TLOAD
                trap    #6
                bra.s   ende
        fehler: lea     fehl(pc),a0
                moveq   #7,d7           * SCHREIBE
                trap    #6
        ende:   rts
        filnam: dc.b    'BEDIEN.TXT',0
                ds      0
        fehl:   dc.b    'Dateiname falsch',13,10,0
                ds      0
        fcb:    ds.b    48
```

Im Beispiel wird der Name "BEDIEN.TXT" in den Dateisteuerblock fcb eingetragen. Bei einem eventuellen Fehler erfolgt eine Meldung, ansonsten wird die Textdatei ab Adresse $10000 geladen. Wichtig ist, das der Dateiname in Großbuchstaben vorliegt. Dies kann durch den Befehl UPPERCAS sichergestellt werden, wenn er vor der Funktion FILLFCB benutzt wird.

```
TRAP-Nummer    : 10
Name           : tsave
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Textdatei abspeichern

Eingabereg.    : a0.l = Zeiger auf den Textanfang
                 a1.l = Zeiger auf einen Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = Text ist als Datei abgespeichert
                          5 = Massenspeicher ist voll
                          6 = Inhaltsverzeichnis ist voll
                        $ff = Fehler beim Zugriff auf das Laufwerk
Zerstörte Reg.: keine
Ab Version     : 1.2
Änderungen     : Ab V 2.0: Erheblich höhere Geschwindigkeit
```

Der Befehl TSAVE speichert einen im Hauptspeicher befindlichen und mit einer binären Null abgeschlossenen Text auf den Massenspeicher. Vorher muß noch ein Dateisteuerblock eingerichtet werden.

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7          * FILLFCB
                trap    #6
                cmp.b   #0,d0
                bne.s   fehler
                lea     $10000,a0
                moveq   #10,d7          * TSAVE
                trap    #6
                bra.s   ende
        fehler: lea     fehl(pc),a0
                moveq   #7,d7           * SCHREIBE
                trap    #6
        ende:   rts
        filnam: dc.b    'BEDIEN.TXT',0
                ds      0
        fehl:   dc.b    'Dateiname falsch',13,10,0
                ds      0
        fcb:    ds.b    48
```

Im Beispiel wird der Name "BEDIEN.TXT" in den Dateisteuerblock fcb eingetragen. Bei einem eventuellen Fehler erfolgt eine Meldung, ansonsten wird der Text ab der Adresse $10000 gespeichert. Wichtig ist, das der Dateiname in Großbuchstaben vorliegt. Dies kann durch den Befehl UPPERCAS sichergestellt werden, wenn er vor der Funktion FILLFCB benutzt wird.

TRAP-Nummer : 11
Name : uppercas
Befehlsgruppe : Stringhandling
Kurzbeschreib.: Klein- in Großbuchstaben wandeln

Eingabereg. : a0.l = Zeiger auf Textstring
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 1.2
Änderungen : keine

Der Befehl UPPERCAS wandelt alle in einem Textstring vorkommenden Kleinbuchstaben in Großbuchstaben um. Der Text muß mit einer binären Null abgeschlossen sein.

Beispiel:

```
        start:  lea     text(pc),a0
                moveq   #11,d7          * UPPERCAS
                trap    #6
                moveq   #7,d7           * SCHREIBE
                trap    #6
                rts
        text:   dc.b    'Dies ist ein Text',0
```

Im Beispiel wird der Text "Dies ist ein Text" umgewandelt in "DIES IST EIN TEXT". Anschließend wird der gewandelte Text angezeigt.

```
TRAP-Nummer    : 12
Name           : wrblank
Befehlsgruppe  : Textausgabe
Kurzbeschreib.: Eine Anzahl Leerzeichen ausgeben

Eingabereg.    : d1.w = Zahl der Leerzeichen
Ausgabereg.    : keine
Zerstörte Reg.: keine
Ab Version     : 1.2
Änderungen     : keine
```

Der Befehl WRBLANK gibt ab der aktuellen Cursorposition die in D1 angegebene Anzahl von Leerzeichen (ASCII-Wert $20) aus.

Beispiel:

```
        start:  lea     text1(pc),a0
                moveq   #7,d7           * SCHREIBE
                trap    #6
                moveq   #3,d1
                moveq   #12,d7          * WRBLANK
                trap    #6
                lea     text2(pc),a0
                moveq   #7,d7           * SCHREIBE
                trap    #6
                rts
        text1:  dc.b    'Name:',0
        text2:  dc.b    'NDR-Klein-Computer',13,10,0
```

Im Beispiel wird der Gesamttext "Name: NDR-Klein-Computer" angezeigt.

TRAP-Nummer : 13
Name : wrint
Befehlsgruppe : Textausgabe
Kurzbeschreib.: Positive 16-bit-Zahl ausgeben

Eingabereg. : d0.w = auszugebende 16-bit-Zahl
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 1.2
Änderungen : keine

Mit dem Befehl WRINT wird eine Zahl zwischen 0 und 65535 dezimal angezeigt. Dabei werden grundsätzlich 6 Zeichen linksbündig ausgegeben, wobei nach rechts Leerzeichen eingesetzt werden.

Beispiel:

```
        start:  move    #135,d0
                moveq   #13,d7
                trap    #6
                rts
```

Im Beispiel wird die Zahl "135 " angezeigt.

TRAP-Nummer : 14
Name : close
Befehlsgruppe : Dateihandling
Kurzbeschreib.: Datei schließen

Eingabereg. : a1.l = Zeiger auf Dateisteuerblock
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 1.2
Änderungen : keine

Der Befehl CLOSE schließt eine geöffnete Datei. Nach Schreiboperationen auf die Datei wird das Inhaltsverzeichnis aktualisiert. Auf eine geschlossene Datei kann nicht mehr zugegriffen werden.

Beispiel:

```
        start:  lea     fcb(pc),a1
                moveq   #14,d7          * CLOSE
                trap    #6
                rts
        fcb:    ds.b    48
```

Im Beispiel wird der Dateisteuerblock fcb geschlossen.

```
TRAP-Nummer    : 15
Name           : copyfile
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Datei kopieren

Eingabereg.    : a0.l = Startadresse des Kopierpuffers
                 a1.l = Zeiger auf den Steuerblock der zu kopierenden Datei
                 a2.l = Zeiger auf den Steuerblock der Kopie
Ausgabereg.    : d0.b = Fehlercode
                          0 = Kopie ist erfolgt
                          2 = Datei nicht gefunden
                          5 = Massenspeicher ist voll
                          6 = Inhaltsverzeichnis ist voll
                        $ff = Fehler beim Zugriff auf den Massenspeicher
Zerstörte Reg.: keine
Ab Version     : 1.2
Änderungen     : Ab V 2.0: Hoher Geschwindigkeitsgewinn durch Benutzung des
                           Userbereichs. Falls die Kopie bereits existiert,
                           wird sie überschrieben. Neu ist das Register A0.
```

Der Befehl COPYFILE kopiert eine einzelne Datei. Dazu wird der komplette Userbereich als Zwischenpuffer benutzt. Bei Kopien auf demselben Laufwerk ist kein Diskettenwechsel möglich. Falls der Name der Kopie schon existiert, wird er ohne Rückfragen überschrieben. Die Startadresse des Zwischenpuffers wird als Eingangsparameter vorgegeben, die Endadresse hängt von der Lage des Userstacks ab und wird von der Funktion automatisch ermittelt.

Beispiel:

```
        start:  lea     dfile(pc),a0
                lea     fcb2(pc),a1
                moveq   #18,d7          * Steuerblock der Zieldatei anlegen
                trap    #6
                cmp.b   #0,d0
                bne.s   fehler
                movea.l a1,a2           * A2 zeigt auf Steuerblock der Kopie
                lea     sfile(pc),a0
                lea     fcb1(pc),a1
                moveq   #18,d7          * Steuerblock der Quelldatei anlegen
                trap    #6
                cmp.b   #0,d0
                bne.s   fehler
                lea     buf(pc),a0      * Startadresse des Kopierpuffers
                moveq   #15,d7          * COPYFILE aufrufen
                trap    #6
        fehler: rts
        sfile:  dc.b    'QUELLE',0
        dfile:  dc.b    'ZIEL'0
                ds      0
        fcb1:   ds.b    48
        fcb2    ds.b    48
        buf:
```

Im Beispiel werden erst die Dateisteuerblöcke angelegt, anschließend wird die Kopierfunktion aufgerufen.

```
TRAP-Nummer    : 16
Name           : create
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Neue Datei einrichten

Eingabereg.    : a1.l = Zeiger auf Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = Datei eingerichtet
                          5 = Massenspeicher ist voll
                          6 = Inhaltsverzeichnis ist voll
                        $ff = Fehler beim Zugriff auf den Massenspeicher
Zerstörte Reg.: keine
Ab Version     : 1.2
Änderungen     : keine
```

Der Befehl CREATE legt eine neue Datei an und trägt sie in das Inhaltsverzeichnis ein. Anschließend erhält die Datei den Status "offen", so daß auf sie zugegriffen werden kann. Nach den Zugriffen muß die Datei mit CLOSE geschlossen werden, damit das Inhaltsverzeichnis aktualisiert wird. Falls die Datei schon existiert, wird sie lediglich geöffnet (siehe OPEN).

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7          * Dateisteuerblock anlegen
                trap    #6
                cmp.b   #0,d0
                bne.s   fehler
                moveq   #16,d7          * Datei anlegen oder nur öffnen
                trap    #6
        fehler: rts
        filnam: dc.b    'DATEI',0
                ds      0
        fcb:    ds.b    48
```

Im Beispiel wird zuerst ein Steuerblock für die Datei mit dem Namen "DATEI" angelegt. Anschließend wird der Befehl CREATE aufgerufen. Falls die Datei noch nicht existiert, wird sie jetzt in das Inhaltsverzeichnis eingetragen.

```
TRAP-Nummer    : 17
Name           : erase
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Datei löschen

Eingabereg.    : a1.l = Zeiger auf Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                        0 = Datei gelöscht
                        2 = Datei nicht vorhanden
                      $ff = Fehler beim Zugriff auf den Massenspeicher
Zerstörte Reg.: keine
Ab Version     : 1.2
Änderungen     : Ab V 2.0: Hoher Geschwindigkeitszuwachs. Löschvorgang wird
                           jetzt auf allen Laufwerken und nicht nur auf dem
                           aktuellen richtig ausgeführt.
```

Der Befehl ERASE löscht eine Datei. Dazu wird im Inhaltsverzeichnis ein entsprechender Eintrag abgelegt. In der Spurtabelle werden alle zu der Datei gehörenden Spuren als frei markiert. Daher kann eine einmal gelöschte Datei praktisch nicht mehr rekonstruiert werden.

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7          * Dateisteuerblock anlegen
                trap    #6
                cmp.b   #0,d0
                bne.s   fehler
                moveq   #17,d7          * Befehl ERASE aufrufen
                trap    #6
        fehler: rts
        filnam: dc.b    'BRIEF.TXT',0
                ds      0
        fcb:    ds.b    48
```

Im Beispiel wird für die Datei "BRIEF.TXT" ein Dateisteuerblock angelegt. War dies erfolgreich, dann wird der Befehl ERASE aufgerufen und die Datei damit gelöscht.

```
TRAP-Nummer    : 18
Name           : fillfcb
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Dateisteuerblock anlegen

Eingabereg.    : a0.l = Zeiger auf Dateinamen
                 a1.l = Zeiger auf Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = Steuerblock angelegt
                        $ff = Text ist kein Dateiname
Zerstörte Reg.: keine
Ab Version     : 1.2
Änderungen     : keine
```

Der Befehl FILLFCB initialisiert einen Speicherbereich von 48 Bytes und trägt den Dateinamen ein, falls der Name formal korrekt ist. Der Dateiname sollte unbedingt nur aus Großbuchstaben bestehen, damit die Datei vom Kommandointerpreter erreicht werden kann. Durch Benutzung von Kleinbuchstaben kann aber erreicht werden, daß eine Datei dem direkten Zugriff des Benutzers entzogen wird, was manchmal nützlich sein kann.

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7          * Dateisteuerblock anlegen
                trap    #6
                rts
        filnam: dc.b    'BRIEF.TXT',0
                ds      0
        fcb:    ds.b    48
```

Im Beispiel wird ein Dateisteuerblock angelegt und der Name "BRIEF.TXT" wird eingetragen.

```
TRAP-Nummer    : 19
Name           : open
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Datei öffnen

Eingabereg.    : a1.l = Zeiger auf Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = Datei geöffnet
                        $ff = Datei nicht vorhanden
Zerstörte Reg.: keine
Ab Version     : 1.2
Änderungen     : keine
```

Der Befehl OPEN öffnet eine Datei zum Lesen und Schreiben. Die Funktion ist zwingend erforderlich, um Lese- und Schreiboperationen durchzuführen. Dabei werden alle wichtigen Einträge aus dem Inhaltsverzeichnis in den Steuerblock übernommen.

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7          * Dateisteuerblock anlegen
                trap    #6
                cmp.b   #0,d0
                bne.s   fehler
                moveq   #19,d7          * Befehl OPEN aufrufen
                trap    #6
        fehler: rts
        filnam: dc.b    'BRIEF.TXT',0
                ds      0
        fcb:    ds.b    48
```

Im Beispiel wird für die Datei "BRIEF.TXT" ein Dateisteuerblock angelegt. War dies erfolgreich, dann wird der Befehl OPEN aufgerufen und die Datei damit für Lese- und Schreiboperationen freigegeben.

```
TRAP-Nummer    : 20
Name           : readrec
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Einen Sektor lesen

Eingabereg.    : a1.l = Zeiger auf Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = Sektor gelesen
                          1 = Dateiende erreicht
                         99 = Ende des Userbereichs erreicht
                        $ff = Fehler beim Zugriff auf den Massenspeicher
Zerstörte Reg.: keine
Ab Version     : 1.2
Änderungen     : keine
```

Der Befehl READREC liest einen Sektor einer geöffneten Datei und positioniert den Sektorzeiger auf den nächsten Sektor. Die Speicheradresse wird ebenso erhöht. Durch fortgesetzte READREC-Aufrufe kann so eine komplette Datei gelesen werden. Bei Erreichen des Dateiendes wird ein Fehlercode erzeugt, ebenso wenn die Speicheradresse das Ende des Userbereichs überschreiten würde.

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7          * Dateisteuerblock anlegen
                trap    #6
                cmp.b   #0,d0
                bne.s   fehler
                moveq   #19,d7          * Befehl OPEN aufrufen
                trap    #6
                cmp.b   #0,d0
                bne.s   fehler
                moveq   #20,d7
                trap    #6
        fehler: rts
        filnam: dc.b    'BRIEF.TXT',0
                ds      0
        fcb:    ds.b    48
```

Im Beispiel wird für die Datei "BRIEF.TXT" ein Dateisteuerblock angelegt. War dies erfolgreich, dann wird der Befehl OPEN aufgerufen und die Datei damit für Lese- und Schreiboperationen freigegeben. Anschließend wird der erste Sektor gelesen.

```
TRAP-Nummer    : 21
Name           : rename
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Datei umbenennen

Eingabereg.    : a1.l = Zeiger auf Dateisteuerblock mit altem Dateinamen
                 a2.l = Zeiger auf Dateisteuerblock mit neuem Dateinamen
Ausgabereg.    : d0.b = Fehlercode
                          0 = Umbenennung durchgeführt
                          2 = Datei nicht vorhanden
                          3 = Neuer Name existiert bereits
                          4 = Laufwerke sind unterschiedlich
                        $ff = Fehler bei Zugriff auf den Massenspeicher
Zerstörte Reg.: keine
Ab Version     : 1.2
Änderungen     : keine
```

Der Befehl RENAME ändert den Namen einer schon existierenden Datei. Dazu müssen zwei Dateisteuerblöcke angelegt werden, nämlich einer mit dem alten Namen und einer mit dem neuen Namen. Falls schon eine Datei mit dem neuen Namen existiert, wird ein Fehlercode erzeugt. Die Umbenennung kann auch nur auf ein und demselben Laufwerk erfolgen.

Beispiel:

```
        start:  lea     dfile(pc),a0
                lea     fcb2(pc),a1
                moveq   #18,d7          * Steuerblock der neuen Datei anlegen
                trap    #6
                cmp.b   #0,d0
                bne.s   fehler
                movea.l a1,a2
                lea     sfile(pc),a0
                lea     fcb1(pc),a1
                moveq   #18,d7          * Steuerblock der alten Datei anlegen
                trap    #6
                cmp.b   #0,d0
                bne.s   fehler
                moveq   #21,d7          * RENAME aufrufen
                trap    #6
        fehler: rts
        sfile:  dc.b    'QUELLE',0
        dfile:  dc.b    'ZIEL'0
                ds      0
        fcb1:   ds.b    48
        fcb2    ds.b    48
        buf:
```

Im Beispiel werden erst die Dateisteuerblöcke angelegt, anschließend wird die Renamefunktion aufgerufen.

TRAP-Nummer : 22
Name : setdta
Befehlsgruppe : Dateihandling
Kurzbeschreib.: Speichertransferadresse setzen

Eingabereg. : a0.l = Speichertransferadresse
a1.l = Zeiger auf Dateisteuerblock
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 1.2
Änderungen : keine

Die Speichertransferadresse wird im Dateisteuerblock verwaltet. Bei sequentiellen Dateizugriffen wird sie automatisch eingerichtet. Nach Aufruf der Funktion OPEN oder CREATE ist die Speichertransferadresse gleich dem Anfang des Userbereichs. Mit dem Befehl SETDTA kann sie verändert werden.

Beispiel:

```
start:  lea     $10000,a0
        lea     fcb(pc),a1
        moveq   #22,d7
        trap    #6
        rts
fcb:    ds.b    48
```

Im Beispiel wird angenommen, daß der Dateisteuerblock fcb bereits initialisiert wurde. Die Speichertransferadresse wird auf $10000 gesetzt.

```
TRAP-Nummer    : 23
Name           : writerec
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Einen Sektor schreiben

Eingabereg.    : a1.l = Zeiger auf Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = Sektor geschrieben
                          5 = Massenspeicher ist voll
                        $ff = Fehler beim Zugriff auf den Massenspeicher
Zerstörte Reg.: keine
Ab Version     : 1.2
Änderungen     : keine
```

Der Befehl WRITEREC schreibt einen Sektor auf eine geöffnete Datei und positioniert den Sektorzeiger auf den nächsten Sektor. Die Speicheradresse wird ebenso erhöht. Wenn über das bisherige Dateiende hinaus geschrieben wird, dann wird gegebenenfalls eine neue Spur angefordert und das Dateiende verschoben. Wenn kein Platz mehr auf dem Massenspeicher ist, dann wird ein Fehlercode erzeugt. Wenn die Datei vergrößert wurde, werden die Einträge in das Inhaltsverzeichnis geändert. Dazu ist aber der Aufruf CLOSE notwendig.

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7          * Dateisteuerblock anlegen
                trap    #6
                cmp.b   #0,d0
                bne.s   fehler
                moveq   #19,d7          * Befehl OPEN aufrufen
                trap    #6
                cmp.b   #0,d0
                bne.s   fehler
                moveq   #23,d7
                trap    #6
        fehler: rts
        filnam: dc.b    'BRIEF.TXT',0
                ds      0
        fcb:    ds.b    48
```

Im Beispiel wird für die Datei "BRIEF.TXT" ein Dateisteuerblock angelegt. War dies erfolgreich, dann wird der Befehl OPEN aufgerufen und die Datei damit für Lese- und Schreiboperationen freigegeben. Anschließend wird der erste Sektor geschrieben.

TRAP-Nummer : 24
Name : getversi
Befehlsgruppe : System
Kurzbeschreib.: Versionsnummer lesen

Eingabereg. : keine
Ausgabereg. : d0.l = Versionsnummer als Textzeichen
Zerstörte Reg.: keine
Ab Version : 1.2
Änderungen : keine

Der Befehl GETVERSI liefert die Versionsnummer des gerade installierten JADOS. Die Versionsnummer wird als Folge von vier ASCII-Zeichen geliefert, nicht als binärer Wert.

Beispiel:

```
        start:  moveq   #24,d7
                trap    #6
                lea     buf(pc),a0
                move.l  d0,(a0)
                clr.b   4(a0)
                moveq   #7,d7           * SCHREIBE
                trap    #6
                rts
        buf:    ds.b    5
```

Im Beispiel wird die Versionsnummer abgerufen, in einen Textpuffer kopiert und mit einer binären Null abgeschlossen. Anschließend wird die Versionsnummer angezeigt.

TRAP-Nummer : 25
Name : getparm
Befehlsgruppe : System
Kurzbeschreib.: Zeiger auf Kommandoparameter liefern

Eingabereg. : keine
Ausgabereg. : a0.l = Zeiger auf Kommandoparameter
Zerstörte Reg.: keine
Ab Version : 1.2
Änderungen : keine

Der Befehl GETPARM liefert einen Zeiger auf die Parameter, die beim Aufruf einer Programm- oder Batchdatei eingegeben wurden. Dadurch kann der gesamte Parameterstring untersucht werden. Alle Kleinbuchstaben sind aber schon vorher durch JADOS umgewandelt worden.

Beispiel:

```
        start:  moveq   #25,d7          * GETPARM
                trap    #6
                moveq   #7,d7           * SCHREIBE
                trap    #6
                rts
```

Das obige Prográmmelchen sei als Datei "SHOWPARM.68K" vorhanden. Wenn nun im Kommandointerpreter "showparm jados ist toll" eingegeben wird, dann zeigt das Programm "JADOS IST TOLL" an.

TRAP-Nummer : 26
Name : hardcopy
Befehlsgruppe : Textausgabe
Kurzbeschreib.: Bildschirminhalt ausdrucken

Eingabereg. : keine
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

Der Befehl HARDCOPY druckt den momentanen Bildschirminhalt auf den Drucker aus. Es wird aber nur der Textmodus unterstützt, d.h. alle mit CO2 oder CO geschriebenen Zeichen, die sich im Bildschirmpuffer befinden.

Beispiel:

```
start:  moveq   #26,d7
        trap    #6
        rts
```

Im Beispiel wird der aktuelle Bildschirminhalt ausgedruckt.

TRAP-Nummer : 27
Name : bell
Befehlsgruppe : Akustik
Kurzbeschreib.: Glockenton erzeugen

Eingabereg. : keine
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

Der Befehl BELL erzeugt mittels der Baugruppe SOUND einen Glockenton. Die Adresse der SOUND-Baugruppe liegt im JADOS auf $FFFFFF50. Sie kann aber in der Datei CONFIG.SYS umdefiniert werden.

Beispiel:

```
start:  moveq   #27,d7
        trap    #6
        rts
```

Im Beispiel wird ein Glockenton erzeugt.

TRAP-Nummer : 28
Name : beep
Befehlsgruppe : Akustik
Kurzbeschreib.: Tonerzeugung

Eingabereg. : d1.l = Tonhöhe in Hertz (20...5000)
d2.l = Tondauer in msek (0...8000)
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

Der Befehl BEEP erzeugt einen Ton einstellbarer Frequenz und einstellbarer Dauer über die Baugruppe SOUND. Die Adresse der SOUND-Baugruppe liegt im JADOS auf $FFFFFF50. Sie kann aber in der Datei CONFIG.SYS umdefiniert werden.

Beispiel:

```
start:  move.l  #1000,d1
        move.l  #1000,d2
        moveq   #28,d7
        trap    #6
        rts
```

Im Beispiel wird ein Ton von 1000 Hz für eine Sekunde erzeugt.

TRAP-Nummer : 29
Name : errnoise
Befehlsgruppe : Akustik
Kurzbeschreib.: Kurzen Fehlerton erzeugen

Eingabereg. : keine
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

Der Befehl ERRNOISE erzeugt mittels der Baugruppe SOUND einen Fehlerton. Die Adresse der SOUND-Baugruppe liegt im JADOS auf $FFFFFF50. Sie kann aber in der Datei CONFIG.SYS umdefiniert werden.

Beispiel:

```
start:  moveq   #29,d7
        trap    #6
        rts
```

Im Beispiel wird ein kurzer Ton erzeugt, der zur Markierung von Fehlern dienen kann.

TRAP-Nummer : 30
Name : sound
Befehlsgruppe : Akustik
Kurzbeschreib.: Ansprechen der SOUND

Eingabereg. : a0.l = Adresse der Soundtabelle
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

Die Funktion SOUND arbeitet genauso wie im Grundprogramm, allerdings liegt die IO-Adresse nicht fest sondern ist einstellbar. Die normale Einstellung ist Adresse $FFFFFF50. In der Datei CONFIG.SYS kann aber auch eine andere Adresse eingestellt werden. Natürlich muß dann auch die Adresse auf der Baugruppe geändert werden.

Dies war nötig, da die Druckerbaugruppe nicht vollständig ausdekodiert ist/war. Dadurch kam es zu Störungen, wenn die Druckerschnittstelle angesprochen wurde.

Beispiel:

```
        start:  lea     sndtab(pc),a0
                moveq   #30,d7
                trap    #6
                rts
        sndtab: dc.b    $de,$01,$dd,$01,$be,$00,$00,$f8
                dc.b    $10,$10,$10,$00,$0a,$08,$00,$00
```

Im Beispiel wird ein rhythmisches Klanggebilde erzeugt.

TRAP-Nummer : 31
Name : inport
Befehlsgruppe : System
Kurzbeschreib.: Port einlesen unabhängig von CPU-Typ

Eingabereg. : d1.l = Basisportadresse
Ausgabereg. : d0.b = Eingelesener Wert
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

Der NDR-Computer mit 68000er unterstützt drei CPU-Typen, nämlich den 68008, den 68000/68010 und den 68020. Die Prozessoren unterscheiden sich unter anderem in der Breite des Datenbusses. Dieser ist 8, 16 oder 32 Bit breit. Durch die Technik des Bussplitting ergeben sich Einflüsse auf die Adressierung der Peripheriebaugruppen. Es ist daher notwendig, die Adressen beim 68000 mit zwei und beim 68020 mit vier zu multiplizieren. Programme, die auf allen Prozessoren laufen sollen, müssen dies berücksichtigen.

Der Befehl INPORT bietet eine Möglichkeit, unabhängig vom CPU-Typ Ports einzulesen. Als Adresse wird die beim 68008 übliche verlangt. Die tatsächliche Adresse wird berechnet. Leider ist diese Methode wesentlich langsamer als das direkte einlesen vom Port. Für viele Anwendungen reicht dies aber vollkommen aus.

Beispiel:

```
        start:  move.l  #$ffffff68,d1
                moveq   #31,d7
                trap    #6
                rts
```

Im Beispiel wird ein Wert von der Tastatur eingelesen.

TRAP-Nummer : 32
Name : outport
Befehlsgruppe : System
Kurzbeschreib.: Auf Port schreiben unabhängig von CPU-Typ

Eingabereg. : d0.b = Wert, der auf Port geschrieben wird
d1.l = Basisportadresse
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

Der NDR-Computer mit 68000er unterstützt drei CPU-Typen, nämlich den 68008, den 68000/68010 und den 68020. Die Prozessoren unterscheiden sich unter anderem in der Breite des Datenbusses. Dieser ist 8, 16 oder 32 Bit breit. Durch die Technik des Bussplitting ergeben sich Einflüsse auf die Adressierung der Peripheriebaugruppen. Es ist daher notwendig, die Adressen beim 68000 mit zwei und beim 68020 mit vier zu multiplizieren. Programme, die auf allen Prozessoren laufen sollen, müssen dies berücksichtigen.

Der Befehl OUTPORT bietet eine Möglichkeit, unabhängig vom CPU-Typ auf Ports zu schreiben. Als Adresse wird die beim 68008 übliche verlangt. Die tatsächliche Adresse wird berechnet. Leider ist diese Methode wesentlich langsamer als das direkte schreiben auf ein Port. Für viele Anwendungen reicht dies aber vollkommen aus.

Beispiel:

```
        start:  move.l  #$ffffff49,d1
                move.b  #1,d0
                bsr.s   out
                move.l  #$ffffff48,d1
                move.b  #12,d0
                bsr.s   out
                move.l  #$ffffff49,d1
                clr.b   d0
                bsr.s   out
                move.b  #1,d0
                bsr.s   out
                rts
        out:    moveq   #32,d7
                trap    #6
                rts
```

Im Beispiel wird das Zeichen zum Seitenvorschub (12) auf den Druckerport ausgegeben.

TRAP-Nummer : 33
Name : movelbin
Befehlsgruppe : Speicherhandling
Kurzbeschreib.: Speicherbereich kopieren (linksseitig)

Eingabereg. : d1.w = Anzahl der Bytes
a1.l = Zeiger auf Bereich, der kopiert wird
a2.l = Zeiger auf Bereich, in den kopiert wird
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

Der Befehl MOVELBIN kopiert einen Speicherbereich. Dabei wird bei den niedrigen Adressen begonnen. Es kann daher auch in demselben Puffer kopiert werden, wenn man von rechts nach links kopieren will.

Beispiel:

```
start:  lea     $d0000,a1
        lea     $80000,a2
        move    #$ffff,d1
        moveq   #33,d7
        trap    #6
        rts
```

Im Beispiel wird der Speicherbereich von $D0000 bis $DFFFF nach $80000 kopiert.

TRAP-Nummer : 34
Name : moveltxt
Befehlsgruppe : Speicherhandling
Kurzbeschreib.: Textbereich im Speicher kopieren

Eingabereg. : a1.l = Zeiger auf Bereich, der kopiert wird
a2.l = Zeiger auf Bereich, in den kopiert wird
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

Der Befehl MOVELTXT kopiert einen Speicherbereich, der mit Text belegt ist, in einen anderen Bereich. Der zu kopierende Bereich muß mit einer binären Null abgeschlossen sein.

Beispiel:

```
start:  lea     $400,a1
        lea     $10000,a2
        moveq   #34,d7
        trap    #6
        rts
```

Im Beispiel wird der ab Adresse $400 abgelegte Text auf den Bereich ab $10000 kopiert.

TRAP-Nummer : 35
Name : moverbin
Befehlsgruppe : Speicherhandling
Kurzbeschreib.: Speicherbereich kopieren (rechtsseitig)

Eingabereg. : d1.w = Anzahl der Bytes
a1.l = Zeiger auf Bereich, der kopiert wird
a2.l = Zeiger auf Bereich, in den kopiert wird
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

Der Befehl MOVERBIN kopiert einen Speicherbereich. Dabei wird bei den höheren Adressen begonnen. Es kann daher auch in demselben Puffer kopiert werden, wenn man von links nach rechts kopieren will.

Beispiel:

```
start:  lea     $d0000,a1
        lea     $80000,a2
        move    #$ffff,d1
        moveq   #35,d7
        trap    #6
        rts
```

Im Beispiel wird der Speicherbereich von $D0000 bis $DFFFF nach $80000 kopiert.

```
TRAP-Nummer    : 36
Name           : wrtcmd
Befehlsgruppe  : System
Kurzbeschreib.: Kommando in Kommandopuffer schreiben

Eingabereg.    : a0.l = Zeiger auf Kommando
Ausgabereg.    : keine
Zerstörte Reg.: keine
Ab Version     : 2.0
Änderungen     : keine
```

Mit dem Befehl WRTCMD wird ein JADOS-Kommando mit Parametern in einen Puffer geschrieben. Nach Beendigung des Programms wird dieser Puffer vom Kommandointerpreter automatisch abgearbeitet. Falls gerade eine Batchdatei bearbeitet wird, bleibt das Kommando unwirksam.

Beispiel:

```
        start:  lea     cmdtxt(pc),a0
                moveq   #36,d7
                trap    #6
                rts
        cmdtxt: dc.b    'LDIR *.68K',0
```

Das obige Programm schreibt ein Kommando in den Kommandopuffer. Nach dem Ende des Programms wird dieses Kommando abgearbeitet.

TRAP-Nummer : 37
Name : gtretcod
Befehlsgruppe : System
Kurzbeschreib.: Returncode abfragen

Eingabereg. : keine
Ausgabereg. : d0.l = Returncode
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

In JADOS existiert ein Speicherbereich von 4 Bytes, auf den Programme schreiben können oder die von Programmen gelesen werden können. Dies kann z.B. dafür benutzt werden, einen Returncode abzusetzen, der dann von anderen Programmen gelesen werden kann, um darauf zu reagieren. Der Initialisierungswert ist gleich Null.

Der Befehl GTRETCOD liest den Returncode.

Beispiel:

```
start:  moveq   #37,d7
        trap    #6
        rts
```

Im Beispiel wird der Returncode gelesen, den ein anderes Programm abgesetzt hat.

TRAP-Nummer : 38
Name : stretcod
Befehlsgruppe : System
Kurzbeschreib.: Returncode absetzen

Eingabereg. : d0.l = Returncode
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

In JADOS existiert ein Speicherbereich von 4 Bytes, auf den Programme schreiben können oder die von Programmen gelesen werden können. Dies kann z.B. dafür benutzt werden, einen Returncode abzusetzen, der dann von anderen Programmen gelesen werden kann, um darauf zu reagieren. Der Initialisierungswert ist gleich Null.

Der Befehl STRETCOD setzt den Returncode.

Beispiel:

```
start:  moveq   #1,d0
        moveq   #38,d7
        trap    #6
        rts
```

Im Beispiel wird der Returncode 1 abgesetzt, um z.B. einen Fehlerzustand anzuzeigen.

```
TRAP-Nummer    : 39
Name           : moveline
Befehlsgruppe  : Speicherhandling
Kurzbeschreib.: Text zeilenweise aus Speicher lesen

Eingabereg.    : d2.w = Größe des Zeilenspeichers
                 a0.l = Zeiger auf Quelltext vor dem kopieren
                 a2.l = Zeiger auf Zeilenspeicher
Ausgabereg.    : d0.b = Fehlercode
                          0 = Zeile gelesen
                        $ff = Textende erreicht
                 a0.l = Zeiger auf Quelltext nach dem kopieren
Zerstörte Reg.: keine
Ab Version     : 2.0
Änderungen     : keine
```

Der Befehl MOVELINE dient zur Bearbeitung von Texten. Dabei wird eine mit carriage return abgeschlossene Textzeile in einen Zeilenspeicher kopiert und dort mit einer binären Null abgeschlossen. Die Größe des Zeilenspeichers wird berücksichtigt, so daß kein unkontrolliertes Überschreiben stattfindet. Der Zeiger in den Quelltext wird laufend verändert. Fortgesetzte Aufrufe von MOVELINE bewirken, daß Zeile für Zeile gelesen wird.

Beispiel:

```
        start:  lea     $400,a0         * Zeiger auf Quelltext
        loop:   lea     buf(pc),a2      * Zeiger auf Zeilenspeicher
                move    #80,d2          * Größe des Zeilenspeichers
                moveq   #39,d7
                trap    #6              * Eine Zeile lesen
                cmp.b   #$ff,d0
                beq.s   ende            * Textende erreicht
                exgl.l  a0,a2
                moveq   #!lst,d7
                trap    #1              * Ausgabe auf Drucker umlenken
                moveq   #7,d7
                trap    #6              * Zeile ausdrucken
                moveq   #!crlf,d7
                trap    #1              * Zeilenvorschub
                moveq   #!crt,d7
                trap    #1              * Ausgabe auf Bildschirm umlenken
                move.b  #'X',d0
                moveq   #!co2,d7
                trap    #1              * X anzeigen
                exg.l   a0,a2
                bra.s   loop
        ende:   rts
        buf:    ds.b    80
```

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

```
TRAP-Nummer    : 40
Name           : wraddr
Befehlsgruppe  : Textausgabe
Kurzbeschreib.: Adresse sedezimal anzeigen

Eingabereg.    : d0.l = Adresse
Ausgabereg.    : keine
Zerstörte Reg.: keine
Ab Version     : 2.0
Änderungen     : keine
```

Der Befehl WRADDR zeigt eine Adresse sedezimal auf sechs Stellen an.

Beispiel:

```
        start:  moveq   #!getbasis,d7
                trap    #1              * Anfangsadresse des Grundprogramms
                moveq   #40,d7          *   ermitteln
                trap    #6
                rts
```

Im Beispiel wird zunächst die Anfangsadresse des Grundprogramms ermittelt. Danach wird die Adresse angezeigt.

TRAP-Nummer : 41
Name : ci
Befehlsgruppe : Texteingabe
Kurzbeschreib.: Zeichen von der Tastatur einlesen

Eingabereg. : keine
Ausgabereg. : d0.b = eingelesener ASCII-Wert
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

Der Befehl CI liest ein Zeichen von der Tastatur ein. Wenn ^P eingegeben wird, dann wird der Bildschirminhalt ausgedruckt (Hardcopy).

Beispiel:

```
        start:  moveq   #41,d7
                trap    #6
                rts
```

TRAP-Nummer : 42
Name : getparm1
Befehlsgruppe : System
Kurzbeschreib.: 1. Kommandoparameter holen

Eingabereg. : keine
Ausgabereg. : a0.l = Zeiger auf 1. Parameter
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

In JADOS können jedem Kommando bis zu vier Parameter mitgegeben werden. Die Funktion GETPARM (Nr. 25) liefert einen Zeiger auf den kompletten Parameterstring. Aus diesem String bildet JADOS vier jeweils mit einer binären Null abgeschlossene Strings mit den einzelnen Parametern. Dadurch können Programme einfacher auf die Parameter zugreifen.

Der Befehl GETPARM1 liefert einen Zeiger auf den ersten Parameter.

Bespiel:

```
start:  moveq   #42,d7
        trap    #6
        moveq   #7,d7
        trap    #6
        rts
```

Im Beispiel wird der 1. Parameter geholt und angezeigt.

TRAP-Nummer : 43
Name : getparm2
Befehlsgruppe : System
Kurzbeschreib.: 2. Kommandoparameter holen

Eingabereg. : keine
Ausgabereg. : a0.l = Zeiger auf 2. Parameter
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

In JADOS können jedem Kommando bis zu vier Parameter mitgegeben werden. Die Funktion GETPARM (Nr. 25) liefert einen Zeiger auf den kompletten Parameterstring. Aus diesem String bildet JADOS vier jeweils mit einer binären Null abgeschlossene Strings mit den einzelnen Parametern. Dadurch können Programme einfacher auf die Parameter zugreifen.

Der Befehl GETPARM2 liefert einen Zeiger auf den zweiten Parameter.

Bespiel:

```
        start:  moveq   #43,d7
                trap    #6
                moveq   #7,d7
                trap    #6
                rts
```

Im Beispiel wird der 2. Parameter geholt und angezeigt.

```
TRAP-Nummer    : 44
Name           : fileload
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Komplette Datei lesen

Eingabereg.    : a0.l = Zeiger auf Adresse, ab der die Datei abgelegt wird
                 a1.l = Zeiger auf den Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = Datei geladen
                          2 = Datei nicht gefunden
                         99 = Userspeicher voll
                        $ff = Fehler beim Zugriff auf den Massenspeicher
Zerstörte Reg.: keine
Ab Version     : 2.0
Änderungen     : keine
```

Mit dem Befehl FILELOAD wird eine komplette Datei vom Massenspeicher in den Arbeitsspeicher geladen. Dies geschieht wesentlich schneller als in einer Programmierschleife mit dem Befehl READREC.

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7
                trap    #6              * Dateisteuerblock anlegen
                cmp.b   #0,d0
                bne.s   fehler
                lea     buf(pc),a0
                moveq   #44,d7
                trap    #6              * Datei laden
        fehler: rts
        filnam: dc.b    'BEISPIEL.TXT',0
                ds      0
        fcb:    ds.b    48
        buf:
```

Im Beispiel wird die Datei "BEISPIEL.TXT" geladen.

```
TRAP-Nummer    : 45
Name           : filesave
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Komplette Datei abspeichern

Eingabereg.    : d1.w = Anzahl der Sektoren minus 1 zu je 1K
                 a0.l = Zeiger auf Adresse, ab der gespeichert wird
                 a1.l = Zeiger auf den Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = Datei gespeichert
                          5 = Massenspeicher voll
                          6 = Inhaltsverzeichnis voll
                        $ff = Fehler beim Zugriff auf den Massenspeicher
Zerstörte Reg.: keine
Ab Version     : 2.0
Änderungen     : keine
```

Mit dem Befehl FILESAVE wird ein Bereich aus dem Arbeitsspeicher als komplette Datei auf den Massenspeicher gebracht. Dies geschieht wesentlich schneller als in einer Programmierschleife mit dem Befehl WRITEREC.

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7
                trap    #6              * Dateisteuerblock anlegen
                cmp.b   #0,d0
                bne.s   fehler
                moveq   #4,d1           * 4 entspricht 5 Sektoren
                lea     buf(pc),a0
                moveq   #45,d7
                trap    #6              * Datei Speichern
        fehler: rts
        filnam: dc.b    'BEISPIEL.TXT',0
                ds      0
        fcb:    ds.b    48
        buf:
```

Im Beispiel wird "BEISPIEL.TXT" in den Dateisteuerblock eingetragen. Dann werden ab Adresse "buf" 5 Sektoren (in d1 wird 4 übergeben !) abgespeichert.

TRAP-Nummer : 46
Name : getparm3
Befehlsgruppe : System
Kurzbeschreib.: 3. Kommandoparameter holen

Eingabereg. : keine
Ausgabereg. : a0.l = Zeiger auf 3. Parameter
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

In JADOS können jedem Kommando bis zu vier Parameter mitgegeben werden. Die Funktion GETPARM (Nr. 25) liefert einen Zeiger auf den kompletten Parameterstring. Aus diesem String bildet JADOS vier jeweils mit einer binären Null abgeschlossene Strings mit den einzelnen Parametern. Dadurch können Programme einfacher auf die Parameter zugreifen.

Der Befehl GETPARM3 liefert einen Zeiger auf den dritten Parameter.

Bespiel:

```
start:  moveq   #46,d7
        trap    #6
        moveq   #7,d7
        trap    #6
        rts
```

Im Beispiel wird der 3. Parameter geholt und angezeigt.

TRAP-Nummer : 47
Name : getparm4
Befehlsgruppe : System
Kurzbeschreib.: 4. Kommandoparameter holen

Eingabereg. : keine
Ausgabereg. : a0.l = Zeiger auf 4. Parameter
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

In JADOS können jedem Kommando bis zu vier Parameter mitgegeben werden. Die Funktion GETPARM (Nr. 25) liefert einen Zeiger auf den kompletten Parameterstring. Aus diesem String bildet JADOS vier jeweils mit einer binären Null abgeschlossene Strings mit den einzelnen Parametern. Dadurch können Programme einfacher auf die Parameter zugreifen.

Der Befehl GETPARM4 liefert einen Zeiger auf den vierten Parameter.

Bespiel:

```
start:  moveq   #47,d7
        trap    #6
        moveq   #7,d7
        trap    #6
        rts
```

Im Beispiel wird der 4. Parameter geholt und angezeigt.

```
TRAP-Nummer    : 48
Name           : loadpart
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Datei lesen solange Arbeitsspeicher reicht

Eingabereg.    : a0.l = Zeiger auf Anfang des Speichers
                 a1.l = Zeiger auf den Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = Datei teilweise geladen
                          1 = Datei komplett geladen
                        $ff = Fehler beim Zugriff auf den Massenspeicher
                 d2.w = Anzahl der geladenen Sektoren
Zerstörte Reg.: keine
Ab Version     : 2.0
Änderungen     : keine
```

Der Befehl LOADPART lädt eine Datei ganz oder teilweise in den Arbeitsspeicher. Der Dateisteuerblock muß vorher initialisiert werden und die Datei muß geöffnet sein (Befehl OPEN). Es werden soviele Sektoren gelesen wie Arbeitsspeicher vorhanden ist.

Ein komplexes Beispiel wird bei der Funktion SAVEPART gegeben.

```
TRAP-Nummer    : 49
Name           : savepart
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Arbeitsspeicher auf Massenspeicher bringen

Eingabereg.    : d2.w = Anzahl der Sektoren zu je 1K
                 a0.l = Adresse, ab der gespeichert wird
                 a1.l = Zeiger auf den Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = gespeichert
                          5 = Massenspeicher voll
                        $ff = Fehler beim Zugriff auf den Massenspeicher
Zerstörte Reg.: keine
Ab Version     : 2.0
Änderungen     : keine
```

Der Befehl SAVEPART speichert aus dem Arbeitsspeicher eine Anzahl Sektoren auf den Massenspeicher. Der Dateisteuerblock muß vorher initialisiert werden und die Datei muß geöffnet oder angelegt sein (Befehl OPEN oder CREATE).

Beispiel:

```
        *================================
        * SCHNELLES KOPIERPROGRAMM
        *  Eingangsreg.
        *    a0.l = Ladeadresse
        *    a1.l = Zeiger auf Quell-FCB
        *    a2.l =  Zeiger auf Ziel-FCB
        *  Ausgangsreg.
        *    d0.b = Fehlercode
        *             0 = ok
        *             2 = Quelle nicht gef.
        *             5 = Disk voll
        *             6 = Directory voll
        *           $ff = Undef. Fehler
        *================================

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

                exg.l    a1,a2           * JA --> Ziel anlegen
                moveq    #16,d7
                trap     #6
                cmp.b    #0,d0
                bne.s    copyend         * Fehler beim Anlegen des Ziels
```

```
copyloop:
        exg.l   a1,a2
        moveq   #48,d7          * LOADPART
        trap    #6
        move    d0,d3
        cmp.b   #255,d0
        beq     coperr1
        exg.l   a1,a2
        moveq   #49,d7          * SAVEPART
        trap    #6
        cmp.b   #5,d0
        beq     copdf           * Massenspeicher voll
        cmp.b   #0,d0
        beq.s   copsok
        bra     coperr1         * Zugriffsfehler
copsok:
        cmp.b   #1,d3           * Dateiende erreicht ?
        bne.s   copyloop        * NEIN --> weiter kopiern
        moveq   #14,d7          * Kopieren fertig: CLOSE aufrufen
        trap    #6
        clr     d0
        bra     copyend

copdf:                          * Massenspeicher ist voll
        exg.l   a1,a2
        moveq   #14,d7
        trap    #6
        move    #5,d0
        bra.s   copyend

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

copserr:                        * Quelle nicht gefunden
        moveq   #2,d0
copyend:
        movem.l (a7)+,d1-d5/a0-a3
        rts
```

Bei diesem Beispiel handelt es sich weitgehend um die interne Funktion COPYFILE. Zuerst wird die zu kopierende Datei, die sogenannte Quelldatei, geöffnet. Ist sie nicht vorhanden, wird ein Fehlercode erzeugt und zum Ende gesprungen. Ist si vorhanden, dann wird die Datei angelegt, dis die Kopie werden soll. In der Schleife copyloop: wird die Quelldatei solange geladen bis sie komplett geladen ist oder das Ende des Arbeitsspeichers erreicht ist. Anschließend wird dieser Bereich des Arbeitsspeichers auf die Zieldatei gebracht. Die Schleife wird sooft durchlaufen bis die Quelldatei komplett gelesen wurde. Am Schluß wird die Zieldatei geschlossen.

TRAP-Nummer : 50
Name : catalog
Befehlsgruppe : Dateihandling
Kurzbeschreib.: Inhaltsverzeichnis anzeigen

Eingabereg. : a0.l = Zeiger auf Text mit Dateimuster
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 2.0
Änderungen : keine

Der Befehl CATALOG zeigt das Inhaltsverzeichnis des spezifizierten Laufwerks an. Es werden nur die Dateien angezeigt, die auf das eingegebene Suchmuster passen. Falls das Suchmuster leer ist, wird dies wie "*.*" interpretiert.

Beispiel:

```
        start:  lea     muster(pc),a0
                moveq   #50,d7
                trap    #6
                rts
        muster: dc.b    '2:*.68K',0
```

Im Beispiel werden alle Dateien vom Typ 68K auf Laufwerk 2 angezeigt.

```
TRAP-Nummer    : 51
Name           : floppy
Befehlsgruppe  : Laufwerke
Kurzbeschreib.: Zugriff auf Disketten, Ramdisk, Harddisk

Eingabereg.    : d1.w = Kommando
                        0 = Steprate in d3 setzen
                        1 = Sektor lesen
                        2 = Sektor schreiben
                 d2.b = Sektornummer (ab 1)
                 d3.b = Spurnummer (ab 0)
                          oder
                        Steprate (0..7)
                 d4.w = Laufwerkscode, Dichte, Seitenauswahl
                        Ramdisk:    d4 = 0
                          oder
                        Disketten:  das niederwertige Nibble enthält die Nummer
                                    des Laufwerks, das höherwertige enthält
                                    Dichtecode und Seitennummer. Für die Nummer
                                    der Laufwerke gilt:
                                    $x1 = Diskette 1
                                    $x2 = Diskette 2
                                    $x4 = Diskette 3
                                    $x8 = Diskette 4
                          oder
                        Festplatte: das niederwertige Byte ist $ff. Das höher-
                                    wertige Byte enthält die Nummer der Parti-
                                    tion 0..25
                 a0.l = Ablageadresse zum Lesen/Schreiben
Ausgabereg.    : d0.b = Fehlercode
                          0 = Befehl ausgeführt
                        $ff = Zugriffsfehler
Zerstörte Reg.: keine
Ab Version     : 2.1
Änderungen     : Ab V 3.0: Ramdiskzugriff möglich
                 Ab V 3.5: Festplattenzugriff möglich
```

Der Befehl FLOPPY ist die zentrale Routine für den Zugriff auf Massenspeicher. Sie entspricht zunächst dem gleichnamigen Befehl des Grundprogramms. Dazu kommt aber noch die Motorsteuerung. Im Unterschied zum Grundprogramm unterstützt FLOPPY auch die anderen Massenspeicher und zwar eine Ramdisk und eine SCSI-Festplatte. In JADOS wird die Ramdisk wie ein Diskettenlaufwerk behandelt. Allerdings ist hierbei keine Motorsteuerung nötig. Der der Ramdisk zugehörige Speicherbereich ist in Spuren und Sektoren unterteilt. Jede Spur besitzt fünf Sektoren zu je 1 KByte. Die Zahl der Spuren hängt von der Größe der Ramdisk ab. Dabei sind minimal 3 Spuren und maximal 256 Spuren möglich. Auch die Festplatte wird wie eine Diskette aufgeteilt. Die komplette Platte wird in Partitionen zu je 2,5 MByte unterteilt. Eine Platte von 20 MByte ergibt acht Partitionen. Die maimale Zahl der Partitionen beträgt 26, dies entspricht einer Platte von 65 MByte. Jede Partition wird wie eine einseitige Diskette mit 256 Spuren zu je 10 Sektoren zu je 1 KByte verwaltet.

Mit FLOPPY kann ein Sektor gelesen oder ein Sektor beschrieben werden. Bei den Disketten kann auch noch die Steprate eingestellt werden.

Beispiel:

```
        start:  moveq   #1,d1           * Lesen
                moveq   #1,d2           * Sektor 1
                moveq   #3,d3           * Spur 3
                move    #$21,d4         * Diskette 1
                lea     buf(pc),a0
                moveq   #51,d7
                trap    #6

                clr.b   d3              * Spur 0
                clr     d4              * Ramdisk
                lea     buf(pc),a0
                moveq   #51,d7
                trap    #6

                move    #$00ff,d4       * Festplattenpartition A
                lea     buf(pc),a0
                moveq   #51,d7
                trap    #6

                move    #$03ff,d4       * Festplattenpartition D
                lea     buf(pc),a0
                moveq   #51,d7
                trap    #6
                rts
        buf:
```

Im Beispiel wird von verschiedenen Massenspeichern jeweils die Spurtabelle eingelesen.

```
TRAP-Nummer    : 52
Name           : drivecod
Befehlsgruppe  : Laufwerke
Kurzbeschreib.: Laufwerksnummer umkodieren

Eingabereg.    : d4.w = Laufwerksnummer
                            0 = Ramdisk
                         1.. 4 = Diskette 1 bis 4
                         5..30 = Festplattenpartition A bis Z
Ausgabereg.    : d4.w = Laufwerksnummer so wie es Befehl FLOPPY verlangt
                        aber ohne Dichte und Seitennummer
Zerstörte Reg.: keine
Ab Version     : 2.1
Änderungen     : Ab V 3.0: Ramdisk zugelassen
                 Ab V 3.5: Festplatte zugelassen
```

Die logische Nummer eines Laufwerks (0..30) wird in ein Format umkodiert, das vom Befehl FLOPPY verlangt wird, der zentralen Routine für den Zugriff auf Massenspeicher.

Beispiel:

```
        start:  move    #5,d4
                moveq   #52,d7
                trap    #6
                moveq   #1,d1
                moveq   #1,d2
                clr     d3
                lea     buf(pc),a0
                moveq   #51,d6
                trap    #6
                rts
        buf:
```

Im Beispiel soll die Spurtabelle der Festplattenpartition A gelesen werden. Dazu wird die Nummer des Laufwerks, hier gleich 5, umkodiert. Anschließend wird die zentrale Routine für den Massenspeicherzugriff, FLOPPY, aufgerufen.

```
TRAP-Nummer    : 53
Name           : getdrive
Befehlsgruppe  : Laufwerke
Kurzbeschreib.: Nummer des aktuellen Laufwerks ermitteln

Eingabereg.    : keine
Ausgabereg.    : d0.w = Nummer des aktuellen Laufwerks
                            0 = Ramdisk
                          1.. 4 = Diskette 1 bis 4
                          5..30 = Festplattenpartition A bis Z
Zerstörte Reg.: keine
Ab Version     : 2.1
Änderungen     : Ab V 3.0: Ramdisk möglich
                 Ab V 3.5: Festplatte möglich
```

Der Befehl GETDRIVE liefert die Nummer des aktuellen Laufwerks.

Beispiel:

```
        start:  moveq   #53,d7
                trap    #6              * Nummer ermitteln
                moveq   #13,d7
                trap    #6              * und anzeigen
                rts
```

Im Beispiel wird die Nummer des aktuellen Laufwerks ermittelt und angezeigt.

```
TRAP-Nummer    : 54
Name           : setdrive
Befehlsgruppe  : Laufwerke
Kurzbeschreib.: Nummer des aktuellen Laufwerks einstellen

Eingabereg.    : d0.w = Nummer des aktuellen Laufwerks
                            0 = Ramdisk
                         1.. 4 = Diskette 1 bis 4
                         5..30 = Festplattenpartition A bis Z
Ausgabereg.    : keine
Zerstörte Reg.: keine
Ab Version     : 2.1
Änderungen     : Ab V 3.0: Ramdisk möglich
                 Ab V 3.5: Festplatte möglich
```

Der Befehl SETDRIVE stellt die Nummer des aktuellen Laufwerks ein.

Beispiel:

```
        start:  move    #5,d0           * Festplattenpertition A
                moveq   #54,d7
                trap    #6              * einstellen
                rts
```

Im Beispiel wird die Festplattenpartition A eingestellt.

TRAP-Nummer : 55
Name : gttraptb
Befehlsgruppe : System
Kurzbeschreib.: Startadresse der Traptabelle liefern

Eingabereg. : keine
Ausgabereg. : a0.l = Startadresse der Traptabelle
Zerstörte Reg.: keine
Ab Version : 2.1
Änderungen : keine

In JADOS existiert eine Tabelle mit den Adreßoffsets aller Traps relativ zur Anfangsadresse der Tabelle. Die Adreßeinträge sind 4 Bytes lang. Der Befehl GTTRAPTB liefert die Adresse des Tabellenanfangs. Damit kann die tatsächliche Adresse jedes Traps ermittelt werden. Eine mögliche Anwendung besteht darin, einen Trap durch einen eigenen zu ersetzen, indem der Adreßeintrag manipuliert wird. Dies funktioniert allerdings nur bedingt, da JADOS intern die Traps mit BSR xxx aufruft, was wesentlich schneller ist. Eine weitere Anwendung kann sein, in der Einzelschrittverarbeitung auch einen JADOS-Trap zu untersuchen.

Insgesamt gesehen dient der Befehl momentan nur zu Experimentierzwecken.

Beispiel:

```
        start:  moveq   #55,d7
                trap    #6              * Adresse ermitteln
                move.l  a0,d0           * Adresse speichern
                lea     bell(pc),a1     * Adresse einer eigenen Routine
                suba.l  d0,a1           * Relativer Offset zur Tabelle
                move.l  a1,20(a0)       * Routine motoroff ersetzen
                rts
        bell:   moveq   #27,d7          * Eigene Routine
                trap    #6
                rts
```

Im Beispiel soll die Routine motoroff ersetzt werden durch eine Routine, die das Klingelzeichen ertönen läßt. Dazu wird zuerst die Adresse der Tabelle ermittelt. Von der Adresse der eigenen Klingelroutine wird die Adresse der Tabelle subtrahiert. Anschließend wird dieser Adreßoffset an den 6. Eintrag plaziert, was der Routine motoroff entspricht (Offset 20).

```
TRAP-Nummer    : 56
Name           : blockread
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Anzahl von Sektoren lesen

Eingabereg.    : d2.w = Anzahl der Sektoren
                 a0.l = Adresse, ab der der Sektorinhalt abgelegt wird
                 a1.l = Zeiger auf Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = Sektoren gelesen
                          1 = Dateiende erreicht
                         99 = Ende des Userbereichs erreicht
                        $ff = Fehler beim Zugriff auf den Massenspeicher
Zerstörte Reg.: keine
Ab Version     : 2.1
Änderungen     : keine
```

Der Befehl BLOCKREAD liest eine Folge von Sektoren einer geöffneten Datei. Da hierbei die Spurtabelle im Arbeitsspeicher verbleibt, ist die Abarbeitung des Befehls wesentlich schneller als eine Schleife über READREC.

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7
                trap    #6              * Dateisteuerblock anlegen
                cmp.b   #0,d0
                bne.s   fehler          * Fehler im Dateinamen
                moveq   #19,d7
                trap    #6              * Datei mit OPEN öffnen
                cmp.b   #0,d0
                bne.s   fehler          * Fehler beim Öffnen
                lea     $10000,a0
                moveq   #4,d2
                moveq   #56,d7
                trap    #6              * 4 Sektoren einlesen
                movea.l 44(a1),a0       * Aktuelle Transferadresse ermitteln
                clr.b   (a0)            * Binäre Null als Textendekennung
                moveq   #14,d7
                trap    #6              * Datei mit CLOSE schließen
                lea     $10000,a0
                moveq   #7,d7
                trap    #6              * Text anzeigen
        fehler: rts
        filnam: dc.b    'DOSTOOL.BAK',0
                ds      0
        fcb:    ds.b    48
```

Im Beispiel sollen die ersten vier Sektoren der Datei "DOSTOOL.BAK" eingelesen und dann angezeigt werden. Dazu wird zuerst der Dateisteuerblock angelegt. Dann wird die Datei geöffnet. Wenn dies geklappt hat, dann werden die ersten vier Sektoren ab Adresse $10000 eingelesen. Der Text wird dann mit einer binären Null abgeschlossen und angezeigt.

```
TRAP-Nummer    : 57
Name           : blockwrite
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Anzahl von Sektoren schreiben

Eingabereg.    : d2.w = Anzahl der Sektoren
                 a0.l = Adresse, ab der gespeichert wird
                 a1.l = Zeiger auf Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = Sektoren geschrieben
                          5 = Massenspeicher voll
                        $ff = Fehler beim Zugriff auf den Massenspeicher
Zerstörte Reg.: keine
Ab Version     : 2.1
Änderungen     : keine
```

Der Befehl BLOCKWRITE schreibt eine Folge von Sektoren auf eine geöffnete Datei. Da hierbei die Spurtabelle im Arbeitsspeicher verbleibt, ist die Abarbeitung des Befehls wesentlich schneller als eine Schleife über WRITEREC.

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7
                trap    #6              * Dateisteuerblock anlegen
                cmp.b   #0,d0
                bne.s   fehler          * Fehler im Dateinamen
                moveq   #16,d7
                trap    #6              * Datei mit CREATE anlegen oder öffnen
                cmp.b   #0,d0
                bne.s   fehler          * Fehler beim Öffnen
                lea     $10000,a0
                moveq   #20,d2
                moveq   #57,d7
                trap    #6              * 20 Sektoren schreiben
                moveq   #14,d7
                trap    #6              * Datei mit CLOSE schließen
        fehler: rts
        filnam: dc.b    'DOSTOOL.BAK',0
                ds      0
        fcb:    ds.b    48
```

Im Beispiel wird der ab Adresse $10000 stehende Speicherinhalt über 20 Sektoren in die Datei "DOSTOOL.BAK" gespeichert. Nach Anlegen des Dateisteuerblocks wird die Datei angelegt/geöffnet. Wenn dies geklappt hat, dann werden die ersten 20 Sektoren ab Adresse $10000 geschrieben.

TRAP-Nummer : 58
Name : getladdr
Befehlsgruppe : System
Kurzbeschreib.: Startadresse des Userbereichs ermitteln

Eingabereg. : keine
Ausgabereg. : a0.l = Startadresse
Zerstörte Reg.: keine
Ab Version : 2.1
Änderungen : keine

Der Befehl GETLADDR liefert die Startadresse des Userbereichs. Auf diese Adresse werden die Programme des Typs 68K geladen und gestartet. Die tatsächliche Adresse hängt davon ab, ob eine Bankbootkarte eingesetzt wird oder nicht. Wenn eine Bankbootkarte einsetzt wird, dann liegt die Adresse bei $400.

Beispiel:

```
        start:  moveq   #58,d7
                trap    #6              * Startadresse ermitteln
                move.l  a0,d0
                moveq   #40,d7
                trap    #6              * und anzeigen
                rts
```

Im Beispiel wird die Startadresse ermittelt und angezeigt.

```
TRAP-Nummer    : 59
Name           : skipchar
Befehlsgruppe  : Stringhandling
Kurzbeschreib.: Zeichen überspringen

Eingabereg.    : d1.b = Code des zu überspringenden Zeichens
                 a0.l = Zeiger auf Text vor dem Aufruf
Ausgabereg.    : a0.l = Zeiger auf Text nach dem Aufruf
Zerstörte Reg.: keine
Ab Version     : 2.1
Änderungen     : keine
```

Mit dem Befehl SKIPCHAR wird ein bestimmtes Zeichen in einem Text übersprungen. Damit können z.B. Leerzeichen übergangen werden.

Beispiel:

```
        start:  lea     text(pc),a0
                moveq   #7,d7
                trap    #6              * Text anzeigen
                move.b  #' ',d1         * Leerzeichen
                moveq   #59,d7
                trap    #6              * überspringen
                moveq   #7,d7
                trap    #6              * und erneut anzeigen
                rts
        text:   dc.b    '     Hugo ist dumm',13,10,0
```

Im Beispiel wird zunächst ein Text angezeigt. Dann werden die führenden Leerzeichen übersprungen und der Text erneut angezeigt.

TRAP-Nummer : 60
Name : delete
Befehlsgruppe : Stringhandling
Kurzbeschreib.: Teilstring in einem String löschen

Eingabereg. : d1.w = Anzahl der zu löschenden Zeichen
a0.l = Zeiger auf Stringposition, ab der gelöscht werden soll
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 2.1
Änderungen : keine

Der Befehl DELETE löscht aus einem String eine Anzahl von Zeichen. Der Rest des Strings wird nach links geschoben, d.h. in Richtung niedrigerer Adressen. Der String muß mit einer binären Null abgeschlossen sein. Werden mehr Zeichen angegeben als der String lang ist, dann wird auf die Stringposition A0 eine binäre Null geschrieben.

Beispiel:

```
        start:  lea     text(pc),a0
                moveq   #7,d7
                trap    #6              * Text anzeigen
                adda.l  #9,a0           * Position von "nicht" adressieren
                moveq   #6,d1           * 6 Zeichen löschen
                moveq   #60,d7
                trap    #6              * DELETE aufrufen
                lea     text(pc),a0
                moveq   #7,d7
                trap    #6              * und erneut anzeigen
                rts
        text:   dc.b    'Hugo ist nicht dumm',13,10,0
```

Im Beispiel wird zunächst der Text "Hugo ist nicht dumm" angezeigt. Dann wird der Teilstring "nicht" entfernt und der Text erneut angezeigt. Armer Hugo !

TRAP-Nummer : 61
Name : insert
Befehlsgruppe : Stringhandling
Kurzbeschreib.: Teilstring in einen String einfügen

Eingabereg. : d2.w = Maximal erlaubte Restlänge des Strings
a0.l = Zeiger auf Stringposition, ab der eingefügt werden soll
a1.l = Zeiger auf einzufügenden Teilstring
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 2.1
Änderungen : keine

Der Befehl INSERT fügt in einen String einen anderen String ein. Der Rest des Strings wird nach rechts geschoben, d.h. in Richtung höherer Adressen. Beide Strings müssen mit einer binären Null abgeschlossen sein. Ist der einzufügende String länger als die Restlänge, dann wird der Befehl nicht ausgeführt.

Beispiel:

```
        start:  lea     text(pc),a0
                lea     intext(pc),a1
                moveq   #7,d7
                trap    #6              * Text anzeigen
                adda.l  #9,a0           * Position von "dumm" adressieren
                moveq   #8,d2           * 8 Zeichen Restlänge
                moveq   #61,d7
                trap    #6              * INSERT aufrufen
                lea     text(pc),a0
                moveq   #7,d7
                trap    #6              * und erneut anzeigen
                rts
        text:   dc.b    'Hugo ist dumm        ',13,10,0
        intext: dc.b    'nicht ',0
```

Im Beispiel wird zunächst der Text "Hugo ist dumm" angezeigt. Dann wird der Teilstring "nicht" eingefügt und der Text erneut angezeigt. Hugo ist wieder rehabilitiert.

TRAP-Nummer : 62
Name : ramtop
Befehlsgruppe : System
Kurzbeschreib.: Ende des Userbereichs ermitteln

Eingabereg. : keine
Ausgabereg. : a0.l = Zeiger auf Ende des Userbereichs
Zerstörte Reg.: keine
Ab Version : 2.1
Änderungen : keine

Das Ende des Userbereichs wird durch den Userstackpointer minus einer kleinen Reserve von 1 KByte angezeigt. Da der Userbereich zugunsten residenter Programme und COM-Programmen schrumpfen kann, verschiebt sich das Ende entsprechend.

Beispiel:

```
        start:  moveq   #62,d7
                trap    #6              * Userbereichsende ermitteln
                move.l  a0,d0
                moveq   #40,d7
                trap    #6              * und anzeigen
                moveq   #!crlf,d7
                trap    #1              * Zeilenvorschub
                move.l  a7,d0           * Stackpointer ermitteln
                moveq   #40,d7
                trap    #6              * und anzeigen
                moveq   #!crlf,d7
                trap    #1              * Zeilenvorschub
                rts
```

Im Beispiel werden das Ende des Userbereichs sowie der aktuelle Stackpointer angezeigt.

TRAP-Nummer : 63
Name : dummy
Befehlsgruppe : keine
Kurzbeschreib.: nicht belegter Befehl

Eingabereg. : keine
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 3.0
Änderungen : Bis V 3.5: Routine zum Ansprechen der Smartwatch
Ab V 3.5: Nur Dummy Einsprungadresse

Der Befehl ist zur Zeit nicht belegt.

TRAP-Nummer : 64
Name : dummy
Befehlsgruppe : keine
Kurzbeschreib.: nicht belegter Befehl

Eingabereg. : keine
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 3.0
Änderungen : Bis V 3.5: Routine zum Ansprechen der Smartwatch
Ab V 3.5: Nur Dummy Einsprungadresse

Der Befehl ist zur Zeit nicht belegt.

TRAP-Nummer : 65
Name : getuhr
Befehlsgruppe : Uhren
Kurzbeschreib.: Datum und Uhrzeit lesen (unabhängig von Uhrentyp)

Eingabereg. : a0.l = Zeiger auf Ergebnispuffer, Werte in BCD
0(a0) = Stunde
1(a0) = Minute
2(a0) = Tag
3(a0) = Monat
4(a0) = Jahr
5(a0) = Wochentag
6(a0) = Sekunde
7(a0) = 100tel Sekunde
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 3.0
Änderungen : keine

Im NDR-Klein-Computer werden zur Zeit zwei verschiedene Uhren unterstützt. Es sind dies die Uhrenbaugruppe mit dem Uhren-IC E-050 und die Smartwatch. Beide werden vom Grundprogramm bereits unterstützt. Der Befehl GETUHR liest Datum und Uhrzeit über das Grundprogramm. Zusätzlich wird eine manuelle Uhr verwaltet. Da im Inhaltsverzeichnis der Dateien auch ein Datum geführt wird, aber nicht alle Benutzer über eine Uhr verfügen, kann das Datum auch manuell eingegeben werden. Mit GETUHR kann das Datum wieder ausgelesen werden, die Uhrzeit ist auf 00:00:00 eingestellt.

Beispiel:

```
start:  moveq   #65,d7
        trap    #6
        rts
```

Im Beispiel wird die Uhrzeit gelesen.

```
TRAP-Nummer    : 66
Name           : setuhr
Befehlsgruppe  : Uhren
Kurzbeschreib.: Datum und Uhrzeit setzen (unabhängig von Uhrentyp)

Eingabereg.    : a0.l = Zeiger auf Puffer, Werte in BCD
                        0(a0) = Stunde
                        1(a0) = Minute
                        2(a0) = Tag
                        3(a0) = Monat
                        4(a0) = Jahr
                        5(a0) = Wochentag
                        6(a0) = Sekunde
                        7(a0) = 100tel Sekunde
Ausgabereg.    : d0.b = Fehlercode
                          0 = Zeit gesetzt
                        $ff = Werte falsch
Zerstörte Reg.: keine
Ab Version     : 3.0
Änderungen     : keine
```

Im NDR-Klein-Computer werden zur Zeit zwei verschiedene Uhren unterstützt. Es sind dies die Uhrenbaugruppe mit dem Uhren-IC E-050 und die Smartwatch. Beide werden vom Grundprogramm bereits unterstützt. Der Befehl SETUHR setzt Datum und Uhrzeit über das Grundprogramm. Zusätzlich wird eine manuelle Uhr verwaltet. Da im Inhaltsverzeichnis der Dateien auch ein Datum geführt wird, aber nicht alle Benutzer über eine Uhr verfügen, kann das Datum auch manuell eingestellt werden.

Beispiel:

```
        start:  lea     buf(pc),a0
                moveq   #66,d7
                trap    #6
                rts
        buf:    dc.b    $19,$57,$27,$11,$89,$00,$00,$00
```

Im Beispiel werden Datum und Uhrzeit auf den Wert eingestellt, an dem diese Funktion beschrieben wurde.

TRAP-Nummer : 67
Name : date
Befehlsgruppe : Uhren
Kurzbeschreib.: Datum lesen und als Klartext aufbereiten

Eingabereg. : a0.l = Zeiger auf Ablagepuffer
Ausgabereg. : a0.l = Zeiger hinter das letzte Textzeichen
Zerstörte Reg.: keine
Ab Version : 3.0
Änderungen : keine

Der Befehl DATE liest die Uhrenwerte aus und bereitet die Informationen zum Datum im Klartext auf. Der Text wird mit einer binären Null abgeschlossen. Das Format der Datumsausgabe ist wie folgt:

wo, tt.mm.jj

wobei gilt:

wo = Wochentag (MO,DI,MI,DO,FR,SA,SO)
tt = Tag (01..31)
mm = Monat (01..12)
jj = Jahr (00..99)

Beispiel:

```
        start:  lea     buf(pc),a0
                moveq   #67,d7
                trap    #6
                lea     buf(pc),a0
                moveq   #7,d7
                trap    #6
                rts
```

Im Beispiel wird das Datum gelesen und angezeigt.

TRAP-Nummer : 68
Name : time
Befehlsgruppe : Uhren
Kurzbeschreib.: Uhrzeit lesen und als Klartext aufbereiten

Eingabereg. : a0.l = Zeiger auf Ablagepuffer
Ausgabereg. : a0.l = Zeiger hinter das letzte Textzeichen
Zerstörte Reg.: keine
Ab Version : 3.0
Änderungen : keine

Der Befehl TIME liest die Uhrenwerte aus und bereitet die Informationen zur Uhrzeit im Klartext auf. Der Text wird mit einer binären Null abgeschlossen. Das Format der Uhrzeitausgabe ist wie folgt:

hh:mm:ss

wobei gilt:

hh = Stunde (00..23)
mm = Minute (00..59)
ss = Sekunde (00..59)

Beispiel:

```
        start:  lea     buf(pc),a0
                moveq   #68,d7
                trap    #6
                lea     buf(pc),a0
                moveq   #7,d7
                trap    #6
                rts
```

Im Beispiel wird die Uhrzeit gelesen und angezeigt.

TRAP-Nummer : 69
Name : datim
Befehlsgruppe : Uhren
Kurzbeschreib.: Datum und Uhrzeit lesen und als Klartext aufbereiten

Eingabereg. : a0.l = Zeiger auf Ablagepuffer
Ausgabereg. : a0.l = Zeiger hinter das letzte Textzeichen
Zerstörte Reg.: keine
Ab Version : 3.0
Änderungen : keine

Der Befehl DATIM liest die Uhrenwerte aus und bereitet die Informationen zu Datum und Uhrzeit im Klartext auf. Der Text wird mit einer binären Null abgeschlossen. Das Format der Datumsausgabe ist wie folgt:

wo, tt.mm.jj hh:mm:ss

wobei gilt:

wo = Wochentag (MO,DI,MI,DO,FR,SA,SO)
tt = Tag (01..31)
mm = Monat (01..12)
jj = Jahr (00..99)
hh = Stunde (00..23)
mm = Minute (00..59)
ss = Sekunde (00..59)

Beispiel:

```
        start:  lea     buf(pc),a0
                moveq   #69,d7
                trap    #6
                lea     buf(pc),a0
                moveq   #7,d7
                trap    #6
                rts
```

Im Beispiel werden Datum und Uhrzeit gelesen und angezeigt.

```
TRAP-Nummer    : 70
Name           : setdatim
Befehlsgruppe  : Uhren
Kurzbeschreib.: Datum und Uhrzeit einstellen

Eingabereg.    : a1.l = Zeiger auf Text mit Datum und Uhrzeit
Ausgabereg.    : d0.b = Fehlercode
                          0 = Datum/Uhrzeit eingestellt
                        $ff = Formatfehler
Zerstörte Reg.: keine
Ab Version     : 3.0
Änderungen     : keine
```

Der Befehl SETDATIM nimmt die im Klartext vorliegenden Informationen über Datum und Uhrzeit und überträgt sie in die Register der Uhr. Der Text muß in einem bestimmten Format vorliegen und die Werte müssen sinnvoll sein, ansonsten wird ein Fehler signalisiert und die Werte der Uhr werden nicht geändert. Das Format ist wie folgt:

```
wo, tt.mm.jj  hh:mm:ss
```

wobei gilt:

```
wo = Wochentag (MO,DI,MI,DO,FR,SA,SO)
tt = Tag (01..31)
mm = Monat (01..12)
jj = Jahr (00..99)
hh = Stunde (00..23)
mm = Minute (00..59)
ss = Sekunde (00..59)
```

Beispiel:

```
        start:  lea     buf(pc),a1
                moveq   #70,d7
                trap    #6
                rts
        buf:    dc.b    'DI, 28.11.89  20:28:00',0
```

Im Beispiel werden Datum und Uhrzeit auf den Wert eingestellt, an dem diese Funktion beschrieben wurde.

```
TRAP-Nummer    : 71
Name           : fileinfo
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Informationen über eine Datei liefern

Eingabereg.    : a1.l = Zeiger auf Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = Informationen sind gültig
                        $ff = Informationen sind nicht verfügbar
                 d1.l = Dateilänge in Bytes
                 d2.l = Datum in BCD-Format
                          Bit  0.. 7 = Tag
                          Bit  8..15 = Monat
                          Bit 16..23 = Jahr
                          Bit 24..31 = 0
                 d3.b = Attributbyte
Zerstörte Reg.: keine
Ab Version     : 3.0
Änderungen     : keine
```

Der Befehl FILEINFO liefert nähere Informationen zu einer Datei. Es sind dies Dateilänge, Datum und Attribut.

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7
                trap    #6              * Dateisteuerblock initialisieren
                cmp.b   #0,d0
                bne.s   fehler
                moveq   #71,d7
                trap    #6              * FILEINFO
                cmp.b   #0,d0
                bne.s   fehler
                move.l  d1,d0
                moveq   #73,d7
                trap    #6              * Dateilänge in Bytes anzeigen
        fehler: rts
        filnam: dc.b    'BEISPIEL.DAT',0
                ds      0
        fcb:    ds.b    48
```

Im Beispiel wird ein Dateisteuerblock mit dem Namen "BEISPIEL.DAT" angelegt. Von dieser Datei werden die Informationen abgerufen und die Dateilänge in Bytes wird angezeigt.

```
TRAP-Nummer    : 72
Name           : diskinfo
Befehlsgruppe  : Laufwerke
Kurzbeschreib.: Informationen zum Laufwerk liefern

Eingabereg.    : d4.b = Laufwerksnummer
                        0 = Ramdisk
                      1.. 4 = Diskette 1..4
                      5..30 = Festplattenpartition A..Z
                 a0.l = Zeiger auf Ergebnispuffer
Ausgabereg.    : d0.b = Fehlercode
                        0 = Informationen sind gültig
                      $ff = Informationen sind nicht verfügbar
                 a0.l = Zeiger auf Informationen (32-Bit-Werte)
                       0(a0) = Anzahl belegter Spuren
                       4(a0) = Anzahl freier Spuren
                       8(a0) = Anzahl belegter Einträge
                      12(a0) = Anzahl freier Einträge
                      16(a0) = Anzahl belegter Sektoren
                      20(a0) = Anzahl freier Sektoren
                      24(a0) = Mindestanzahl freier Bytes
Zerstörte Reg.: keine
Ab Version     : 3.0
Änderungen     : Ab V 3.5: Festplattenpartitionen werden unterstützt
```

Der Befehl DISKINFO untersucht das Inhaltsverzeichnis und die Spurtabelle des gewünschten Laufwerks. Als Ergebnis werden Informationen über den Grad der Belegung und der freien Kapazitäten geliefert. Dazu muß ein Ergebnispuffer von mindestens 28 Bytes bereitgestellt werden.

Beispiel:

```
        start:  move.b  #6,d4
                lea     buf(pc),a0
                moveq   #72,d7
                trap    #6              * DISKINFO
                cmp.b   #0,d0
                bne.s   fehler
                move.l  8(a0),d0
                moveq   #73,d7
                trap    #6              * Anzahl belegter Einträge anzeigen
                moveq   #!crlf,d7
                trap    #1              * Zeilenvorschub
                move.l  12(a0),d0
                moveq   #73,d7
                trap    #6              * Anzahl freier Einträge anzeigen
        fehler: rts
        buf:    ds.b    28
```

Im Beispiel werden die Informationen zur Festplattenpartition B abgerufen. Danach werden die Zahl der belegten und freien Einträge in das Inhaltsverzeichnis angezeigt.

TRAP-Nummer : 73
Name : wrlint
Befehlsgruppe : Textausgabe
Kurzbeschreib.: Positive 32-bit-Zahl ausgeben

Eingabereg. : d0.l = auszugebende 32-bit-Zahl
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 3.0
Änderungen : keine

Mit dem Befehl WRLINT wird eine Zahl zwischen 0 und $2^{32}-1$ dezimal angezeigt. Dabei werden grundsätzlich 11 Zeichen linksbündig ausgegeben, wobei nach rechts Leerzeichen eingesetzt werden.

Beispiel:

```
        start:  move    #135,d0
                moveq   #73,d7
                trap    #6
                rts
```

Im Beispiel wird die Zahl "135 " angezeigt.

```
TRAP-Nummer    : 74
Name           : directory
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Inhaltsverzeichnis liefern

Eingabereg.    : d1.w = Größe des Ablagepuffers
                 d2.b = Auswahlattribut (bitmapped)
                        Bit 0 = Dateilänge
                        Bit 1 = Datum
                        Bit 2 = Schreibschutzattribut
                        Ein gesetztes Bit wählt die Information aus. Der
                        Dateiname wird immer geliefert
                 d3.w = Anzahl der Spalten für die Ausgabe
                 a0.l = Zeiger auf Ablagepuffer
                 a1.l = Zeiger auf Dateisuchmuster
Ausgabereg.    : d0.b = Fehlercode
                          0 = Informationen sind gültig
                        $ff = Informationen sind nicht gültig
Zerstörte Reg.: keine
Ab Version     : 3.0
Änderungen     : keine
```

Der recht komplexe Befehl DIRECTORY liefert Informationen über die eingetragenen Dateien eines Laufwerks. Das Laufwerk und das Suchmuster für die Dateien werden im Register A1 im Klartext vorgegeben. Im Muster sind die Jokerzeichen "*" und "?" zugelassen. Die Informationen werden nicht direkt angezeigt, sondern in den Ablagepuffer geschrieben. Zeilen werden mit Carriage Return Linefeed beendet, der Puffer mit einer binären Null abgeschlossen. Die Zahl der Spalten wird vorgegeben. Zu den Informationen gehört immer der Name einer Datei. Darüberhinaus wird vorgegeben, ob zusätzlich die Dateilänge, das Datum oder das Attribut berücksichtigt werden sollen.

Beispiel:

```
        start:  lea     buf(pc),a0
                lea     nam(pc),a1
                move    #$1000,d1
                clr.b   d2
                moveq   #1,d3
                moveq   #74,d7
                trap    #6
                rts
        nam:    dc.b    '*.68K',0
        buf:
```

Im Beispiel werden alle Dateien vom Typ 68K, die auf dem aktuellen Laufwerk vorhanden sind, ausgewählt. Es werden nur die Namen berücksichtigt und zwar ein Name pro Zeile. Da in einer Festplattenpartition 256 Dateien vorkommen können und pro Eintrag 14 Bytes nötig sind, beträgt die Puffergröße etwa 4 K.

TRAP-Nummer : 75
Name : table
Befehlsgruppe : Residente Programme
Kurzbeschreib.: Verzeichnis der residenten Programme

Eingabereg. : a0.l = Zeiger auf Ablagepuffer
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 3.0
Änderungen : keine

Der Befehl TABLE liefert ein Verzeichnis aller residenten Programme. Die Ausgabe wird in einen Puffer gestellt und nicht direkt angezeigt. Sie kann so weiterverarbeitet werden. Je Eintrag wird eine Zeile verwendet. Der Puffer wird mit einer binären Null beendet. Da zur Zeit bis zu 20 Einträge vorkommen können, ist die Puffergröße auf 512 Bytes zu dimensionieren.

Beispiel:

```
        start:  lea     buf(p),a0
                moveq   #75,d7
                trap    #6
                lea     buf(pc),a0
                moveq   #7,d7
                trap    #6
                rts
        buf:
```

Im Beispiel werden die Informationen über alle residenten Programme in den Puffer "buf" übernommen und anschließend angezeigt.

```
TRAP-Nummer    : 76
Name           : respinfo
Befehlsgruppe  : Residente Programme
Kurzbeschreib.: Liefert Informationen über residente Programme

Eingabereg.    : keine
Ausgabereg.    : a0.l = Zeiger auf Tabelle der residenten Programme mit 20 Ein-
                        trägen zu je 16 Bytes. Je Eintrag gilt das Format:
                          0(a0) = Name (8 Bytes)
                          8(a0) = Anfangsadresse im Speicher (4 Bytes)
                         12(a0) = Länge des Programms in KByte (2 Bytes)
                         14(a0) = Programmtyp
                                    0 = COM-Programm
                                   >0 = 68K-Programm
                 a1.l = Zeiger auf Tabellenindex des letzten Eintrags (2 Bytes)
                 a2.l = Zeiger auf Adresse des Userstacks (4 Bytes)
Zerstörte Reg.: keine
Ab Version     : 3.0
Änderungen     : keine
```

Der Befehl RESPINFO liefert alle Informationen, die nötig sind, residente Programme zu verwalten. Dazu gehören die Tabelle der eingetragenen Programme, sowie der Index des letzten Eintrags in die Tabelle und der Stackpointer.

Beispiel:

```
        start:  moveq   #76,d7
                trap    #6
                rts
```

```
TRAP-Nummer    : 77
Name           : lineedit
Befehlsgruppe  : Texteingabe
Kurzbeschreib.: Editor für eine Textzeile

Eingabereg.    : d2.w = Länge des Zeilenpuffers
                 a1.l = Zeiger auf Zeilenpuffer
Ausgabereg.    : d0.b = Fehlercode
                          0 = Ende mit <Return>
                          1 = Abbruch mit <ESC>
                          2 = Abbruch mit <^C>
                          5 = Ende durch <CURSOR AUF>
                         24 = Ende durch <CURSOR AB>
Zerstörte Reg.: keine
Ab Version     : 3.0
Änderungen     : keine
```

Der Befehl LINEEDIT beinhaltet einen recht komfortablen Editor für eine Textzeile. Folgende Kommandos sind erlaubt:

```
^A    = nach Wortanfang links
^C    = Abbruch mit Fehlercode
^D    = Cursor rechts
^E    = Abbruch mit Code für Cursor auf
^F    = nach Wortanfang rechts
^G    = Zeichen löschen und Rest nach links
TAB   = nach Wortanfang rechts
RET   = Zeile korrekt beenden
^S    = Cursor links
^T    = Rest der Zeile löschen
^U    = Ein Leerzeichen einfügen
^X    = Abbruch mit Code für Cursor ab
^V    = Toggle für Einfügemodus (Default = AUS)
BSP   = Zeichen links löschen und Rest nach links
DEL   =               dito
ESC   = Abbruch mit Fehlercode
```

Beispiel:

```
        start:  lea     buf(pc),a1
                move    #79,d2
                moveq   #77,d7
                trap    #6
                rts
        buf:    ds.b    80
```

```
TRAP-Nummer    : 78
Name           : getcpu
Befehlsgruppe  : System
Kurzbeschreib.: CPU-Typ liefern

Eingabereg.    : keine
Ausgabereg.    : d0.l = CPU-Typ
                        1 = 68008
                        2 = 68000
                        4 = 68020
Zerstörte Reg.: keine
Ab Version     : 3.0
Änderungen     : keine
```

Der Befehl GETCPU liefert den Typ der im Computer eingesetzten CPU.

Beispiel:

```
        start:  moveq   #78,d7
                trap    #6
                cmp.l   #1,d0
                beq.s   cpu08
                cmp.l   #2,d0
                beq.s   cpu00
                cmp.l   #4,d0
                beq.s   cpu20
                lea     tundef(pc),a0
                bra.s   out
        cpu08:  lea     tcpu08(pc),a0
                bra.s   out
        cpu00:  lea     tcpu00(pc),a0
                bra.s   out
        cpu20:  lea     tcpu20(pc),a0
        out:    moveq   #7,d7
                trap    #6
                rts
        tundef: dc.b    'CPU unbekannt',13,10,0
        tcpu08: dc.b    'CPU 68008',13,10,0
        tcpu00: dc.b    'CPU 68000',13,10,0
        tcpu20: dc.b    'CPU 68020',13,10,0
```

Im Beispiel wird der CPU-Typ ermittelt und im Klartext angezeigt.

TRAP-Nummer : 79
Name : getclock
Befehlsgruppe : Uhren
Kurzbeschreib.: Uhrentyp liefern

Eingabereg. : keine
Ausgabereg. : d0.l = Uhrentyp
0 = Keine Uhr
1 = Hardwareuhr
Zerstörte Reg.: keine
Ab Version : 3.0
Änderungen : Bis V 3.5: Zwischen E050 und Smartwatch wurde unterschieden

Der Befehl GETCLOCK liefert den Typ der eingesetzten Uhr. Da ab V 3.5 aber alle Hardwareuhren vom Grundprogramm verwaltet werden, ist die Funktion nur noch nützlich, um zu erkennen, ob überhaupt eine Uhr im System ist.

Beispiel:

```
        start:  moveq   #79,d7
                trap    #6
                cmp.l   #0,d0
                beq.s   manu
                lea     tclock(pc),a0
                bra.s   out
        manu:   lea     tmanu(pc),a0
        out:    moveq   #7,d7
                trap    #6
                rts
        tclock: dc.b    'Uhr ist vorhanden',13,10,0
        tmanu:  dc.b    'Keine Uhr im System',13,10,0
```

Im Beispiel wird der Uhrentyp ermittelt und das Ergebnis im Klartext angezeigt.

```
TRAP-Nummer    : 80
Name           : getgrund
Befehlsgruppe  : System
Kurzbeschreib.: Anfangsadresse des Grundprogramms liefern

Eingabereg.    : keine
Ausgabereg.    : d0.l = Anfangsadresse des Grundprogramms
                 a0.l = Anfangsadresse des Grundprogramms
Zerstörte Reg.: keine
Ab Version     : 3.0
Änderungen     : keine
```

Der Befehl GETGRUND liefert die Anfangsadresse des Grundprogramms.

Beispiel:

```
        start:  moveq   #80,d7
                trap    #6
                cmp.l   #0,d0
                beq.s   noboot
                lea     tboot(pc),a0
                bra.s   out
        noboot: lea     tnobotpc),a0
        out:    moveq   #7,d7
                trap    #6
                rts
        tboot:  dc.b    'Bootkarte ist vorhanden',13,10,0
        tnobot: dc.b    'Keine Bootkarte im System',13,10,0
```

Im Beispiel wird die Anfangsadresse des Grundprogramms ermittelt. Davon abhängig wird angezeigt, ob das System über eine Bootkarte verfügt.

```
TRAP-Nummer    : 81
Name           : getsound
Befehlsgruppe  : System
Kurzbeschreib.: Portadresse der SOUND-Karte liefern

Eingabereg.    : keine
Ausgabereg.    : d0.l = Basisadresse der SOUND bezogen auf CPU 68008
                 a0.l = Basisadresse der SOUND bezogen auf CPU 68008
Zerstörte Reg.: keine
Ab Version     : 3.0
Änderungen     : keine
```

Der Befehl GETSOUND liefert die Basisadresse der Baugruppe SOUND. Die Adresse gilt für die CPU 68008 und ist mit dem CPU-Typ zu multiplizieren.

Beispiel:

```
        start:  moveq   #81,d7
                trap    #6
                move.l  d0,d2
                moveq   #78,d7
                trap    #6
                moveq   #!muls32,d7
                trap    #1
                moveq   #40,d7
                trap    #6
                rts
```

Im Beispiel wird die Basisadresse der SOUND ermittelt, mit dem CPU-Typ multipliziert und die echte Adresse angezeigt.

```
TRAP-Nummer    : 82
Name           : getdisks
Befehlsgruppe  : Laufwerke
Kurzbeschreib.: Informationen über Laufwerksparameter liefern

Eingabereg.    : keine
Ausgabereg.    : a0.l = Zeiger auf Tabelle mit Laufwerksparametern. Es werden
                        8 Laufwerke unterstützt.
                          0 = Ramdisk
                          1 = Diskette 1
                          2 = Diskette 2
                          3 = Diskette 3
                          4 = Diskette 4
                          5 = Festplatte
                          6 = nicht belegt
                          7 = nicht belegt
                        Je Laufwerk sind 16 Bytes vorhanden. Sie bedeuten:
                          Byte      0 = Status des Laufwerks
                                        0 = undefiniert
                                        1 = gültig
                                        2 = nicht vorhanden
                          Byte      1 = Zahl der Oberflächen
                          Byte  2.. 3 = Zahl der Zylinder
                          Byte      4 = Zahl der Sektoren je Spur
                          Byte      5 = Dichtecode
                          Byte  6.. 7 = Nummer der Directoryspur
                          Byte  8.. 9 = Freie Kapazität in KByte
                          Byte 10..11 = Nummer des letzten Bytes in Spurtabelle
                          Byte 12..13 = Zahl der freien Spuren
                          Byte 14..15 = Nummer des ersten Bytes in Spurtabelle
Zerstörte Reg.: keine
Ab Version     : 3.0
Änderungen     : keine
```

Der Befehl GETDISKS liefert einen Zeiger auf die Tabelle der Laufwerksparameter. Alle ein Laufwerk betreffenden Informationen werden aus dieser Tabelle geholt. Damit ist es möglich, auch eigene Anpassungen vorzunehmen, etwa ein anderes Diskettenformat einzurichten. Die Parameter der Ramdisk werden von JADOS teilweise berechnet, da sie von der Größe der Ramdisk abhängen. Hier sollten keine Änderungen vorgenommen werden. Auch Änderungen an der Einstellung der Festplatte sind tunlichst zu vermeiden.

Beispiel:

```
        start:  moveq   #82,d7
                trap    #6
                adda.l  #32,a0          * Zeiger auf Diskette 2
                move    #40,2(a0)       *   40 Zylinder
                move    #380,8(a0)      *   380 KByte
                move    #320,10(a0)     *
                move    #76,12(a0)      *   76 freie Spuren
                rts
```

Im Beispiel werden die Parameter des 2. Diskettenlaufwerks so verändert, daß JADOS ein 40-Spur-Laufwerk verwalten kann.

```
TRAP-Nummer    : 83
Name           : chmode
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Schreibschutzattribut einer Datei ändern

Eingabereg.    : d1.b = Attribut
                        0 = Schreibschutz setzen
                        1 = Schutz aufheben
                 a1.l = Zeiger auf Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = Attribut übernommen
                          2 = Datei nicht vorhanden
                        $ff = Fehler beim Zugriff auf den Massenspeicher
Zerstörte Reg.: keine
Ab Version     : 3.0
Änderungen     : keine
```

Der Befehl CHMODE verändert das Schreibschutzattribut einer Datei. Bei gesetztem Schreibschutz kann mit den JADOS-Kommandos und fast allen Traps eine Datei weder überschrieben noch gelöscht werden. Die einzige Ausnahme bildet der Befehl FLOPPY.

Beispiel:

```
        start:  lea     nam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7
                trap    #6              * Dateisteuerblock anlegen
                cmp.b   #0,d0
                bne.s   fehler
                clr     d1
                moveq   #83,d7
                trap    #6
        fehler: rts
        nam:    dc.b    'JADOS.SYS',0
                ds      0
        fcb:    ds.b    48
```

Im Beispiel wird die Datei "JADOS.SYS" gegen Überschreiben und Löschen gesichert.

TRAP-Nummer : 84
Name : gtcmdtab
Befehlsgruppe : System
Kurzbeschreib.: Zeiger auf Tabelle der internen Kommandos liefern

Eingabereg. : keine
Ausgabereg. : a0.l = Startadresse der Kommandotabelle, je Eintrag:
Byte 0 = Kommandoname, mit Null abgeschlossen
Byte 6 = Offset relativ zum Tabellenstart
a1.l = Endadresse der Kommandotabelle
Zerstörte Reg.: keine
Ab Version : 3.0
Änderungen : keine

In JADOS existiert eine Tabelle mit den internen Kommandos. Je Eintrag ist der Kommandoname und der Offset des zugehörigen Programmteils bezüglich der Startadresse der Tabelle vorhanden. Der Name besteht aus 6 Zeichen inklusive der Endenull. Man kann damit eine Übersicht der Kommandos geben oder Kommandonamen ändern.

Beispiel:

```
        start:  moveq   #84,d7
                trap    #6
                clr     d2
        loop:   moveq   #7,d7
                trap    #6              * Name anzeigen
                addq    #1,d2
                cmp     #10,d2
                bge.s   newl            * Zeilenvorschub
                moveq   #2,d1
                moveq   #12,d7
                trap    #6
                bra.s   weiter
        newl:   moveq   #!crlf,d7
                trap    #1
                clr     d2
        weiter: adda.l  #10,a0
                cmpa.l  a1,a0
                blt.s   loop
                rts
```

Im Beispiel wird die Tabelle adressiert und dann werden alle Kommandonamen angezeigt. Dabei werden je Zeile 10 Kommandos aufgeführt.

```
TRAP-Nummer    : 85
Name           : getdpath
Befehlsgruppe  : Laufwerke
Kurzbeschreib.: Pfadangaben liefern

Eingabereg.    : keine
Ausgabereg.    : a0.l = Zeiger auf Laufwerkspfade
                        0..30 = gültige Laufwerke
                        $ff   = Ende der Tabelle
Zerstörte Reg.: keine
Ab Version     : 3.0
Änderungen     : keine
```

In JADOS kann ein Zugriffspfad auf maximal 8 Laufwerke eingerichtet werden. Der Pfad wird in einem kleinen Puffer verwaltet. Die Nummer $ff kennzeichnet das Ende der Laufwerksangaben. Der Befehl GETDPATH liefert die Startadresse des Puffers.

Beispiel:

```
        start:  moveq   #85,d7
                trap    #6
                move.l  #-1,d0          * Pfade löschen
                move.l  d0,(a0)
                move.l  d0,4(a0)
                move.b  d0,8(a0)
                move.l  #$05060708,(a0)
                rts
```

Im Beispiel werden die aktuellen Zugriffspfade gelöscht. Anschließend werden die Festplattenpartitionen A bis D in den Zugriff übernommen.

TRAP-Nummer : 86
Name : clrovwrt
Befehlsgruppe : System
Kurzbeschreib.: Speicherschutz reaktivieren

Eingabereg. : keine
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 3.0
Änderungen : keine

Beim Lesen von Dateien aus einem Massenspeicher in den Arbeitsspeicher wird normalerweise das Ende des Userbereichs geprüft, damit die dahinterliegenden Betriebssystemteile nicht zerstört werden. Manchmal ist diese automatische Prüfung aber nicht erwünscht. Sie kann mit dem Befehl SETOVWRT vorübergehend aufgehoben werden. Mit CLROVWRT wird die Prüfung wieder eingeschaltet. Auch nach Rückkehr in den Kommandointerpreter ist die Prüfung wieder aktiviert. Bei der Verwendung des Befehls SETOVWRT ist Vorsicht angebracht !

Ein Beispiel ist in der Beschreibung des Befehls SETOVWRT angegeben.

TRAP-Nummer : 87
Name : setovwrt
Befehlsgruppe : System
Kurzbeschreib.: Speicherschutz aufheben

Eingabereg. : keine
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 3.0
Änderungen : keine

Beim Lesen von Dateien aus einem Massenspeicher in den Arbeitsspeicher wird normalerweise das Ende des Userbereichs geprüft, damit die dahinterliegenden Betriebssystemteile nicht zerstört werden. Manchmal ist diese automatische Prüfung aber nicht erwünscht. Sie kann mit dem Befehl SETOVWRT vorübergehend aufgehoben werden. Mit CLROVWRT wird die Prüfung wieder eingeschaltet. Auch nach Rückkehr in den Kommandointerpreter ist die Prüfung wieder aktiviert. Bei der Verwendung des Befehls SETOVWRT ist Vorsicht angebracht !

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7
                trap    #6              * Dateisteuerblock anlegen
                cmp.b   #0,d0
                bne.s   fehler          * Fehler im Dateinamen
                moveq   #19,d7
                trap    #6              * Datei mit OPEN öffnen
                cmp.b   #0,d0
                bne.s   fehler          * Fehler beim Öffnen
                moveq   #87,d7
                trap    #6              * Speicherschutz aufheben !!
                lea     buf(pc),a0
                moveq   #4,d2
                moveq   #56,d7
                trap    #6              * 4 Sektoren einlesen
                movea.l 44(a1),a0       * Aktuelle Transferadresse ermitteln
                clr.b   (a0)            * Binäre Null als Textendekennung
                moveq   #14,d7
                trap    #6              * Datei mit CLOSE schließen
                lea     buf(pc),a0
                moveq   #7,d7
                trap    #6              * Text anzeigen
        fehler: rts
        filnam: dc.b    'DOSTOOL.BAK',0
                ds      0
        fcb:    ds.b    48
        buf:    ds.b    4100
```

Das obige Programm sei als COM-Datei vorhanden. Da die COM-Datei oberhalb des Userbereichs abläuft, scheitert das Lesen vom Massenspeicher auf den im Programm definierten Puffer normalerweise. Durch Aufheben des Speicherschutzes gelingt das Laden dennoch.

TRAP-Nummer : 88
Name : asctonum
Befehlsgruppe : Laufwerke
Kurzbeschreib.: Wandelt Laufwerksbezeichner "0".."4","A".."Z" in Nummer 0..30

Eingabereg. : d0.b = Laufwerksbezeichner in ASCII
Ausgabereg. : d0.b = Laufwerksnummer binär
Zerstörte Reg.: keine
Ab Version : 3.5
Änderungen : keine

Der Befehl ASCTONUM wandelt einen Laufwerksbezeichner im Klartext ("0".."4" und "A".."Z") in eine binäre Zahl um (0..30).

Beispiel:

```
start:  move.b  #'A',d0
        moveq   #88,d7
        trap    #6
        rts
```

TRAP-Nummer : 89
Name : numtoasc
Befehlsgruppe : Laufwerke
Kurzbeschreib.: Wandelt Laufwerksnummer 0..30 in Bezeichner "0".."4","A".."Z"

Eingabereg. : d0.b = Laufwerksnummer binär
Ausgabereg. : d0.b = Laufwerksbezeichner in ASCII
Zerstörte Reg.: keine
Ab Version : 3.5
Änderungen : keine

Der Befehl NUMTOASC wandelt eine Laufwerksnummer (0..30) in einen Laufwerksbezeichner ("0".."4" und "A".."Z").

Beispiel:

```
        start:  move.b  #5,d0
                moveq   #89,d7
                trap    #6
                rts
```

```
TRAP-Nummer    : 90
Name           : gethdisk
Befehlsgruppe  : Laufwerke
Kurzbeschreib.: Liefert Informationen über Festplattenpartitionen

Eingabereg.    : keine
Ausgabereg.    : a0.l = Zeiger auf Partitionstabelle. Es gibt 26 Partitionen.
                        Je Partition sind 8 Bytes vorhanden:
                          Byte 0..1: Status der Partition
                                     0 = nicht initialisiert
                                     1 = gültig
                                     2 = nicht vorhanden
                          Byte 2..3: Erster Sektor der SCSI-Platte
                          Byte 4..7: Reserve
Zerstörte Reg.: keine
Ab Version     : 3.5
Änderungen     : keine
```

Die komplette Festplatte wird in gleich große Partitionen aufgeteilt. Die allen Partitionen gemeinsamen Parameter können mit GETDISKS untersucht werden. Es werden insgesamt bis zu 26 Partitionen unterstützt. Die tatsächliche Zahl hängt nur von der Kapazität der Festplatte ab. Pro Partition existiert ein Status, der besagt, ob die Partition existiert.

Eine SCSI-Platte wird intern in logische Blöcke aufgeteilt, die unter JADOS zu je 1 KByte - also wie ein Sektor auf der Diskette - formatiert sind. Die Zahl der physikalischen Köpfe, Zylinder und Sektoren sind dem Betriebssystem unbekannt. Es benutzt nur die durchgehende Blocknummer. Aus Verwaltungsgründen wird eine Partition wie eine einseitige Diskette mit 256 Spuren und 10 Sektoren je Spur angesprochen. Die tatsächliche Blocknummer, ab der eine Partition beginnt, ist ebenfalls in der Tabelle enthalten. Der Befehl GETHDISK liefert einen Zeiger auf diese Tabelle. Damit kann man untersuchen, welche Partitionen existieren.

Beispiel:

```
        start:  moveq   #90,d7
                trap    #6
                cmp     #1,64(a0)
                bne.s   ende
                lea     texist(pc),a0
                moveq   #7,d7
                trap    #6
        ende:   rts
        texist: dc.b    'Es gibt mehr als 8 Partitionen',13,10,0
```

Im Beispiel wird untersucht, ob die Partition "I" existiert. Wenn ja, dann wird ein Text angezeigt.

```
TRAP-Nummer    : 91
Name           : loadauto
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Lädt Sequenz von Dateien aus einer Antwortdatei

Eingabereg.    : a0.l = Zeiger auf Name der Antwortdatei
                 a2.l = Startadresse im Arbeitsspeicher
Ausgabereg.    : d0.b = Fehlercode
                          0 = Alle Dateien geladen
                        $ff = Fehler beim Laden
Zerstörte Reg.: keine
Ab Version     : 3.5
Änderungen     : keine
```

Der Befehl LOADAUTO ist hervorgegangen aus dem Befehl LOADTEXT. Während LOADTEXT benutzergeführt arbeitet, funktioniert LOADAUTO ohne Benutzereingriff. Aus einer Antwortdatei, in der pro Zeile ein Dateiname steht, wird Datei für Datei hintereinander in den Arbeitsspeicher geladen. Alle diese Dateien müssen Textdateien sein, da ansonsten das Aneinanderketten nicht funktioniert.

Beispiel:

```
        start:  lea     antfil(pc),a0
                lea     buf(pc),a2
                moveq   #91,d7
                trap    #6
                cmp.b   #0,d0
                bne.s   fehler
                moveq   #!lst,d7
                trap    #1
                moveq   #7,d7
                trap    #6
                moveq   #!crt,d7
                trap    #1
        fehler: rts
        antfil: dc.b    'DOSLINK',0
                ds      0
        buf:
```

Im Beispiel werden alle in "DOSLINK" verzeichneten Dateien geladen und ausgedruckt.

TRAP-Nummer : 92
Name : cmdexec
Befehlsgruppe : System
Kurzbeschreib.: Internes JADOS-Kommando oder 68K-Datei ausführen

Eingabereg. : a0.l = Zeiger auf Kommandostring
Ausgabereg. : keine
Zerstörte Reg.: keine
Ab Version : 3.5
Änderungen : keine

Mit dem Befehl CMDEXEC kann aus einem Programm heraus auf alle internen Kommandos von JADOS zugegriffen werden. Auch 68K-Dateien können auf diese Weise benutzt werden. Wegen der Speicher- und Stackverwaltung können COM-Dateien und BAT-Dateien nicht ausgeführt werden; dies ist verriegelt. Da einige JADOS-Kommandos den Userspeicher als Zwischenpuffer benutzen und die 68K-Dateien an den Anfang des Userspeichers geladen werden, darf der Befehl CMDEXEC nur von COM-Programmen benutzt werden. Damit können Programme, die als COM-Datei laufen, ein leistungsfähiges Interface zum Betriebssystem anbieten.

```
TRAP-Nummer    : 93
Name           : setrec
Befehlsgruppe  : Dateihandling
Kurzbeschreib.: Wahlfreier Zugriff auf einen logischen Sektor einer Datei

Eingabereg.    : d1.w = Logische Sektornummer (0..65535)
                 a1.l = Zeiger auf Dateisteuerblock
Ausgabereg.    : d0.b = Fehlercode
                          0 = positioniert
                          1 = Dateiende erreicht
                        $ff = Fehler beim Zugriff auf den Massenspeicher
Zerstörte Reg.: keine
Ab Version     : 3.5
Änderungen     : keine
```

Der Befehl SETREC erlaubt es, wahlfrei in Dateien zu positionieren. Zusammen mit READREC und WRITEREC kann ein wahlfreies Zugriffssystem auf Dateien aufgebaut werden. Hierbei muß beachtet werden, daß READREC und WRITEREC vor dem Zugriff die logische Sektornummer inkrementieren !

Beispiel:

```
        start:  lea     filnam(pc),a0
                lea     fcb(pc),a1
                moveq   #18,d7
                trap    #6              * Dateisteuerblock anlegen
                cmp.b   #0,d0
                bne.s   fehler          * Fehler im Dateinamen
                moveq   #19,d7
                trap    #6              * Datei mit OPEN öffnen
                cmp.b   #0,d0
                bne.s   fehler          * Fehler beim Öffnen
                lea     buf(pc),a0
                moveq   #22,d7
                trap    #6              * Transferadresse festlegen
                move    26(a1),d1       * Letzten Sektor einstellen
                sub     #1,d1
                moveq   #93,d7
                trap    #6              * Sektor positionieren
                moveq   #20,d7
                trap    #6              * Sektor einlesen
                moveq   #14,d7
                trap    #6              * Datei mit CLOSE schließen
                lea     buf(pc),a0
                moveq   #7,d7
                trap    #6              * Text anzeigen
        fehler: rts
        filnam: dc.b    'DOSTOOL.BAK',0
                ds      0
        fcb:    ds.b    48
        buf:
```

Im Beispiel wird der letzte Sektor einer Textdatei angezeigt.

Sortierung nach TRAP-Nummern Teil 1

TRAP Nr.	Befehls-name	Kurzbeschreibung	Ab Vers	Befehlsgruppe
0	getleng	Textlänge ermitteln	1.2	Stringhandling
1	getname	Dateinamen erfragen	1.2	Dialogführung
2	getstadd	Startadresse erfragen	1.2	Dialogführung
3	lese	Einlesen einer Textzeile	1.2	Texteingabe
4	loadtext	Menügeführtes Laden von Texten	1.2	Dialogführung
5	motoroff	Diskettenmotoren abschalten	1.2	Laufwerke
6	response	Auf Leertastendruck warten	1.2	Dialogführung
7	schreibe	Text anzeigen	1.2	Textausgabe
8	strgcomp	Zwei Textstrings vergleichen	1.2	Stringhandling
9	tload	Textdatei laden	1.2	Dateihandling
10	tsave	Textdatei speichern	1.2	Dateihandling
11	uppercas	Klein- in Großbuchstaben wandeln	1.2	Stringhandling
12	wrblank	Anzahl Leerzeichen ausgeben	1.2	Textausgabe
13	wrint	Positive 16-Bit-Zahl ausgeben	1.2	Textausgabe
14	close	Datei schließen	1.2	Dateihandling
15	copyfile	Datei kopieren	1.2	Dateihandling
16	create	Datei anlegen oder öffnen	1.2	Dateihandling
17	erase	Datei löschen	1.2	Dateihandling
18	fillfcb	Dateisteuerblock anlegen	1.2	Dateihandling
19	open	Datei öffnen	1.2	Dateihandling
20	readrec	Sektor lesen	1.2	Dateihandling
21	rename	Datei umbenennen	1.2	Dateihandling
22	setdta	Transferadresse einstellen	1.2	Dateihandling
23	writerec	Sektor schreiben	1.2	Dateihandling
24	getversi	Version ermitteln	1.2	System
25	getparm	Kommandoparameter ermitteln	1.2	System
26	hardcopy	Bildschirminhalt ausdrucken	2.0	Textausgabe
27	bell	Glockenton erzeugen	2.0	Akustik
28	beep	Tonerzeugung	2.0	Akustik
29	errnoise	Fehlerton erzeugen	2.0	Akustik
30	sound	SOUND-Baugruppe ansprechen	2.0	Akustik
31	inport	Port einlesen für alle CPU's	2.0	System
32	outport	Auf Port schreiben für alle CPU's	2.0	System
33	movelbin	Speicherbereich nach links kopieren	2.0	Speicherhandling
34	moveltxt	Textspeicher nach links kopieren	2.0	Speicherhandling
35	moverbin	Speicherbereich nach rechts kopieren	2.0	Speicherhandling
36	wrtcmd	Kommando in Kommandopuffer schreiben	2.0	System
37	gtretcod	Returncode abfragen	2.0	System
38	stretcod	Returncode setzen	2.0	System
39	moveline	Textzeile lesen	2.0	System
40	wraddr	Adresse ausgeben	2.0	Textausgabe
41	ci	Zeichen von Tastatur einlesen	2.0	Zeicheneingabe
42	getparm1	1. Kommandoparameter ermitteln	2.0	System
43	getparm2	2. Kommandoparameter ermitteln	2.0	System
44	fileload	Datei komplett laden	2.0	Dateihandling
45	filesave	Datei komplett speichern	2.0	Dateihandling
46	getparm3	3. Kommandoparameter ermitteln	2.0	System
47	getparm4	4. Kommandoparameter ermitteln	2.0	System

TRAP Nr.	Befehls-name	Kurzbeschreibung	Ab Vers	Befehlsgruppe
48	loadpart	Datei solange lesen wie Speicher da	2.0	Dateihandling
49	savepart	Speicherinhalt in Datei schreiben	2.0	Dateihandling
50	catalog	Inhaltsverzeichnis anzeigen	2.0	Dateihandling
51	floppy	Zugriff auf Massenspeicher	2.1	Laufwerke
52	drivecod	Laufwerksnummer umkodieren	2.1	Laufwerke
53	getdrive	Aktuelles Laufwerk ermitteln	2.1	Laufwerke
54	setdrive	Aktuelles Laufwerk einstellen	2.1	Laufwerke
55	gttraptb	Anfangsadresse der Traptabelle liefern	2.1	System
56	blockread	Anzahl Sektoren lesen	2.1	Dateihandling
57	blockwrit	Anzahl Sektoren schreiben	2.1	Dateihandling
58	getladdr	Anfangsadresse des Userbereichs liefern	2.1	System
59	skipchar	Zeichen überspringen	2.1	Stringhandling
60	delete	Teilstring in String löschen	2.1	Stringhandling
61	insert	Teilstring in String einfügen	2.1	Stringhandling
62	ramtop	Ende des Userbereichs ermitteln	2.1	System
63	---			
64	---			
65	getuhr	Uhrenregister unabhängig von Uhr lesen	3.0	Uhren
66	setuhr	Uhrenregister unabhängig von Uhr setzen	3.0	Uhren
67	date	Datum ausgeben	3.0	Uhren
68	time	Uhrzeit ausgeben	3.0	Uhren
69	datim	Datum und Uhrzeit ausgeben	3.0	Uhren
70	setdatim	Datum und Uhrzeit einstellen	3.0	Uhren
71	fileinfo	Dateiinformationen ermitteln	3.0	Dateihandling
72	diskinfo	Laufwerksbelegung ermitteln	3.0	Laufwerke
73	wrlint	Positive 32-Bit-Zahl ausgeben	3.0	Textausgabe
74	directory	Inhaltsverzeichnis ermitteln	3.0	Dateihandling
75	table	Verzeichnis der residenten Programme	3.0	Residente Prog.
76	respinfo	Informationen über residente Programme	3.0	Residente Prog.
77	lineedit	Editor für eine Textzeile	3.0	Texteingabe
78	getcpu	CPU-Typ ermitteln	3.0	System
79	getclock	Uhrentyp ermitteln	3.0	System
80	getgrund	Anfangsadresse des Grundprogramms	3.0	System
81	getsound	Basisadresse der SOUND ermitteln	3.0	System
82	getdisks	Parametertabelle der Laufwerke	3.0	Laufwerke
83	chmode	Schreibschutzattribut ändern	3.0	Dateihandling
84	gtcmdtab	Adresse der Kommandotabelle liefern	3.0	System
85	getdpath	Pfadangaben ermitteln	3.0	Laufwerke
86	clrovwrt	Speicherschutz reaktivieren	3.0	System
87	setovwrt	Speicherschutz aufheben	3.0	System
88	asctonum	Laufwerksbezeichner in Nummer wandeln	3.5	Laufwerke
89	numtoasc	Nummer in Laufwerksbezeichner wandeln	3.5	Laufwerke
90	gethdisk	Partitionstabelle der Festplatte	3.5	Laufwerke
91	loadauto	Dateien aus Antwortdatei laden	3.5	Dateihandling
92	cmdexec	Kommandointerface zu JADOS	3.5	System
93	setrec	Wahlfrei auf Sektor positionieren	3.5	Dateihandling

Sortierung nach TRAP-Nummern Teil 2

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
0	getleng	a0.l	d1.w
1	getname	a0.l-a1.l	d0.b
2	getstadd	a2.l	a0.l
3	lese	d2.w/a1.l	d0.b
4	loadtext	a2.l	d0.b
5	motoroff	keine	keine
6	response	keine	keine
7	schreibe	a0.l	keine
8	strgcomp	a0.l-a1.l	d0.b
9	tload	a0.l-a1.l	d0.b
10	tsave	a0.l-a1.l	d0.b
11	uppercas	a0.l	keine
12	wrblank	d1.w	keine
13	wrint	d0.w	keine
14	close	a1.l	keine
15	copyfile	a0.l-a2.l	d0.b
16	create	a1.l	d0.b
17	erase	a1.l	d0.b
18	fillfcb	a0.l-a1.l	d0.b
19	open	a1.l	d0.b
20	readrec	a1.l	d0.b
21	rename	a1.l-a2.l	d0.b
22	setdta	a0.l-a1.l	keine
23	writerec	a1.l	d0.b
24	getversi	keine	d0.l
25	getparm	keine	a0.l
26	hardcopy	keine	keine
27	bell	keine	keine
28	beep	d1.l-d2.l	keine
29	errnoise	keine	keine
30	sound	a0.l	keine
31	inport	d1.l	d0.b
32	outport	d0.b/d1.l	keine
33	movelbin	d1.w/a1.l-a2.l	keine
34	moveltxt	a1.l-a2.l	keine
35	moverbin	d1.w/a1.l-a2.l	keine
36	wrtcmd	a0.l	keine
37	gtretcod	keine	d0.l
38	stretcod	d0.l	keine
39	moveline	d2.w/a0.l/a2.l	d0.b
40	wraddr	d0.l	keine
41	ci	keine	d0.b
42	getparm1	keine	a0.l
43	getparm2	keine	a0.l
44	fileload	a0.l-a1.l	d0.b
45	filesave	d1.w/a0.l-a1.l	d0.b
46	getparm3	keine	a0.l
47	getparm4	keine	a0.l

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
48	loadpart	a0.l-a1.l	d0.b
49	savepart	d2.w/a0.l-a1.l	d0.b
50	catalog	a0.l	keine
51	floppy	d1.w/d2.b/d3.b/d4.w/a0.l	d0.b
52	drivecod	d4.w	d4.w
53	getdrive	keine	d0.w
54	setdrive	d0.w	keine
55	gttraptb	keine	a0.l
56	blockread	d2.w/a0.l-a1.l	d0.b
57	blockwrit	d2.w/a0.l-a1.l	d0.b
58	getladdr	keine	a0.l
59	skipchar	d1.b/a0.l	a0.l
60	delete	d1.w/a0.l	keine
61	insert	d2.w/a0.l-a1.l	keine
62	ramtop	keine	a0.l
63	---		
64	---		
65	getuhr	a0.l	keine
66	setuhr	a0.l	d0.b
67	date	a0.l	a0.l
68	time	a0.l	a0.l
69	datim	a0.l	a0.l
70	setdatim	a1.l	d0.b
71	fileinfo	a1.l	d0.b/d1.l/d2.l/d3.b
72	diskinfo	d4.b/a0.l	d0.b/a0.l
73	wrlint	d0.l	keine
74	directory	d1.w/d2.b/d3.w/a0.l-a1.l	d0.b
75	table	a0.l	keine
76	respinfo	keine	a0.l-a2.l
77	lineedit	d2.w/a1.l	d0.b
78	getcpu	keine	d0.l
79	getclock	keine	d0.l
80	getgrund	keine	d0.l/a0.l
81	getsound	keine	d0.l/a0.l
82	getdisks	keine	a0.l
83	chmode	d1.b/a1.l	d0.b
84	gtcmdtab	keine	a0.l-a1.l
85	getdpath	keine	a0.l
86	clrovwrt	keine	keine
87	setovwrt	keine	keine
88	asctonum	d0.b	d0.b
89	numtoasc	d0.b	d0.b
90	gethdisk	keine	a0.l
91	loadauto	a0.l/a2.l	d0.b
92	cmdexec	a0.l	keine
93	setrec	d1.w/a1.l	d0.b

Sortierung nach dem Namen Teil 1

TRAP Nr.	Befehls-name	Kurzbeschreibung	Ab Vers	Befehlsgruppe
88	asctonum	Laufwerksbezeichner in Nummer wandeln	3.5	Laufwerke
28	beep	Tonerzeugung	2.0	Akustik
27	bell	Glockenton erzeugen	2.0	Akustik
56	blockread	Anzahl Sektoren lesen	2.1	Dateihandling
57	blockwrit	Anzahl Sektoren schreiben	2.1	Dateihandling
50	catalog	Inhaltsverzeichnis anzeigen	2.0	Dateihandling
83	chmode	Schreibschutzattribut ändern	3.0	Dateihandling
41	ci	Zeichen von Tastatur einlesen	2.0	Zeicheneingabe
14	close	Datei schließen	1.2	Dateihandling
86	clrovwrt	Speicherschutz reaktivieren	3.0	System
92	cmdexec	Kommandointerface zu JADOS	3.5	System
15	copyfile	Datei kopieren	1.2	Dateihandling
16	create	Datei anlegen oder öffnen	1.2	Dateihandling
67	date	Datum ausgeben	3.0	Uhren
69	datim	Datum und Uhrzeit ausgeben	3.0	Uhren
60	delete	Teilstring in String löschen	2.1	Stringhandling
74	directory	Inhaltsverzeichnis ermitteln	3.0	Dateihandling
72	diskinfo	Laufwerksbelegung ermitteln	3.0	Laufwerke
52	drivecod	Laufwerksnummer umkodieren	2.1	Laufwerke
17	erase	Datei löschen	1.2	Dateihandling
29	errnoise	Fehlerton erzeugen	2.0	Akustik
71	fileinfo	Dateiinformationen ermitteln	3.0	Dateihandling
44	fileload	Datei komplett laden	2.0	Dateihandling
45	filesave	Datei komplett speichern	2.0	Dateihandling
18	fillfcb	Dateisteuerblock anlegen	1.2	Dateihandling
51	floppy	Zugriff auf Massenspeicher	2.1	Laufwerke
79	getclock	Uhrentyp ermitteln	3.0	System
78	getcpu	CPU-Typ ermitteln	3.0	System
82	getdisks	Parametertabelle der Laufwerke	3.0	Laufwerke
85	getdpath	Pfadangaben ermitteln	3.0	Laufwerke
53	getdrive	Aktuelles Laufwerk ermitteln	2.1	Laufwerke
80	getgrund	Anfangsadresse des Grundprogramms	3.0	System
90	gethdisk	Partitionstabelle der Festplatte	3.5	Laufwerke
58	getladdr	Anfangsadresse des Userbereichs liefern	2.1	System
0	getleng	Textlänge ermitteln	1.2	Stringhandling
1	getname	Dateinamen erfragen	1.2	Dialogführung
25	getparm	Kommandoparameter ermitteln	1.2	System
42	getparm1	1. Kommandoparameter ermitteln	2.0	System
43	getparm2	2. Kommandoparameter ermitteln	2.0	System
46	getparm3	3. Kommandoparameter ermitteln	2.0	System
47	getparm4	4. Kommandoparameter ermitteln	2.0	System
81	getsound	Basisadresse der SOUND ermitteln	3.0	System
2	getstadd	Startadresse erfragen	1.2	Dialogführung
65	getuhr	Uhrenregister unabhängig von Uhr lesen	3.0	Uhren
24	getversi	Version ermitteln	1.2	System
84	gtcmdtab	Adresse der Kommandotabelle liefern	3.0	System
37	gtretcod	Returncode abfragen	2.0	System
55	gttraptb	Anfangsadresse der Traptabelle liefern	2.1	System

TRAP Nr.	Befehls-name	Kurzbeschreibung	Ab Vers	Befehlsgruppe
26	hardcopy	Bildschirminhalt ausdrucken	2.0	Textausgabe
31	inport	Port einlesen für alle CPU's	2.0	System
61	insert	Teilstring in String einfügen	2.1	Stringhandling
3	lese	Einlesen einer Textzeile	1.2	Texteingabe
77	lineedit	Editor für eine Textzeile	3.0	Texteingabe
91	loadauto	Dateien aus Antwortdatei laden	3.5	Dateihandling
48	loadpart	Datei solange lesen wie Speicher da	2.0	Dateihandling
4	loadtext	Menügeführtes Laden von Texten	1.2	Dialogführung
5	motoroff	Diskettenmotoren abschalten	1.2	Laufwerke
33	movelbin	Speicherbereich nach links kopieren	2.0	Speicherhandling
39	moveline	Textzeile lesen	2.0	System
34	moveltxt	Textspeicher nach links kopieren	2.0	Speicherhandling
35	moverbin	Speicherbereich nach rechts kopieren	2.0	Speicherhandling
89	numtoasc	Nummer in Laufwerksbezeichner wandeln	3.5	Laufwerke
19	open	Datei öffnen	1.2	Dateihandling
32	outport	Auf Port schreiben für alle CPU's	2.0	System
62	ramtop	Ende des Userbereichs ermitteln	2.1	System
20	readrec	Sektor lesen	1.2	Dateihandling
21	rename	Datei umbenennen	1.2	Dateihandling
76	respinfo	Informationen über residente Programme	3.0	Residente Prog.
6	response	Auf Leertastendruck warten	1.2	Dialogführung
49	savepart	Speicherinhalt in Datei schreiben	2.0	Dateihandling
7	schreibe	Text anzeigen	1.2	Textausgabe
70	setdatim	Datum und Uhrzeit einstellen	3.0	Uhren
54	setdrive	Aktuelles Laufwerk einstellen	2.1	Laufwerke
22	setdta	Transferadresse einstellen	1.2	Dateihandling
87	setovwrt	Speicherschutz aufheben	3.0	System
93	setrec	Wahlfrei auf Sektor positionieren	3.5	Dateihandling
66	setuhr	Uhrenregister unabhängig von Uhr setzen	3.0	Uhren
59	skipchar	Zeichen überspringen	2.1	Stringhandling
30	sound	SOUND-Baugruppe ansprechen	2.0	Akustik
38	stretcod	Returncode setzen	2.0	System
8	strgcomp	Zwei Textstrings vergleichen	1.2	Stringhandling
75	table	Verzeichnis der residenten Programme	3.0	Residente Prog.
68	time	Uhrzeit ausgeben	3.0	Uhren
9	tload	Textdatei laden	1.2	Dateihandling
10	tsave	Textdatei speichern	1.2	Dateihandling
11	uppercas	Klein- in Großbuchstaben wandeln	1.2	Stringhandling
40	wraddr	Adresse ausgeben	2.0	Textausgabe
12	wrblank	Anzahl Leerzeichen ausgeben	1.2	Textausgabe
13	wrint	Positive 16-Bit-Zahl ausgeben	1.2	Textausgabe
23	writerec	Sektor schreiben	1.2	Dateihandling
73	wrlint	Positive 32-Bit-Zahl ausgeben	3.0	Textausgabe
36	wrtcmd	Kommando in Kommandopuffer schreiben	2.0	System
63	---			
64	---			

Sortierung nach dem Namen Teil 2
==========

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
88	asctonum	d0.b	d0.b
27	bell	keine	keine
28	beep	d1.l-d2.l	keine
56	blockread	d2.w/a0.l-a1.l	d0.b
57	blockwrit	d2.w/a0.l-a1.l	d0.b
50	catalog	a0.l	keine
83	chmode	d1.b/a1.l	d0.b
41	ci	keine	d0.b
14	close	a1.l	keine
86	clrovwrt	keine	keine
92	cmdexec	a0.l	keine
15	copyfile	a0.l-a2.l	d0.b
16	create	a1.l	d0.b
67	date	a0.l	a0.l
69	datim	a0.l	a0.l
60	delete	d1.w/a0.l	keine
74	directory	d1.w/d2.b/d3.w/a0.l-a1.l	d0.b
72	diskinfo	d4.b/a0.l	d0.b/a0.l
52	drivecod	d4.w	d4.w
17	erase	a1.l	d0.b
29	errnoise	keine	keine
71	fileinfo	a1.l	d0.b/d1.l/d2.l/d3.b
44	fileload	a0.l-a1.l	d0.b
45	filesave	d1.w/a0.l-a1.l	d0.b
18	fillfcb	a0.l-a1.l	d0.b
51	floppy	d1.w/d2.b/d3.b/d4.w/a0.l	d0.b
79	getclock	keine	d0.l
78	getcpu	keine	d0.l
82	getdisks	keine	a0.l
85	getdpath	keine	a0.l
53	getdrive	keine	d0.w
80	getgrund	keine	d0.l/a0.l
90	gethdisk	keine	a0.l
58	getladdr	keine	a0.l
0	getleng	a0.l	d1.w
1	getname	a0.l-a1.l	d0.b
25	getparm	keine	a0.l
42	getparm1	keine	a0.l
43	getparm2	keine	a0.l
46	getparm3	keine	a0.l
47	getparm4	keine	a0.l
81	getsound	keine	d0.l/a0.l
2	getstadd	a2.l	a0.l
65	getuhr	a0.l	keine
24	getversi	keine	d0.l
84	gtcmdtab	keine	a0.l-a1.l
37	gtretcod	keine	d0.l
55	gttraptb	keine	a0.l

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
63	---		
64	---		
26	hardcopy	keine	keine
31	inport	d1.l	d0.b
61	insert	d2.w/a0.l-a1.l	keine
3	lese	d2.w/a1.l	d0.b
77	lineedit	d2.w/a1.l	d0.b
91	loadauto	a0.l/a2.l	d0.b
48	loadpart	a0.l-a1.l	d0.b
4	loadtext	a2.l	d0.b
5	motoroff	keine	keine
33	movelbin	d1.w/a1.l-a2.l	keine
39	moveline	d2.w/a0.l/a2.l	d0.b
34	moveltxt	a1.l-a2.l	keine
35	moverbin	d1.w/a1.l-a2.l	keine
89	numtoasc	d0.b	d0.b
19	open	a1.l	d0.b
32	outport	d0.b/d1.l	keine
62	ramtop	keine	a0.l
20	readrec	a1.l	d0.b
21	rename	a1.l-a2.l	d0.b
76	respinfo	keine	a0.l-a2.l
6	response	keine	keine
49	savepart	d2.w/a0.l-a1.l	d0.b
7	schreibe	a0.l	keine
70	setdatim	a1.l	d0.b
54	setdrive	d0.w	keine
22	setdta	a0.l-a1.l	keine
87	setovwrt	keine	keine
93	setrec	d1.w/a1.l	d0.b
66	setuhr	a0.l	d0.b
59	skipchar	d1.b/a0.l	a0.l
30	sound	a0.l	keine
38	stretcod	d0.l	keine
8	strgcomp	a0.l-a1.l	d0.b
75	table	a0.l	keine
68	time	a0.l	a0.l
9	tload	a0.l-a1.l	d0.b
10	tsave	a0.l-a1.l	d0.b
11	uppercas	a0.l	keine
40	wraddr	d0.l	keine
12	wrblank	d1.w	keine
13	wrint	d0.w	keine
23	writerec	a1.l	d0.b
36	wrtcmd	a0.l	keine
73	wrlint	d0.l	keine

Sortierung nach Gruppen und TRAP-Nummern

Gruppe: Akustik

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
27	bell	keine	keine
28	beep	d1.l-d2.l	keine
29	errnoise	keine	keine
30	sound	a0.l	keine

Gruppe: Dateihandling

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
9	tload	a0.l-a1.l	d0.b
10	tsave	a0.l-a1.l	d0.b
14	close	a1.l	keine
15	copyfile	a0.l-a2.l	d0.b
16	create	a1.l	d0.b
17	erase	a1.l	d0.b
18	fillfcb	a0.l-a1.l	d0.b
19	open	a1.l	d0.b
20	readrec	a1.l	d0.b
21	rename	a1.l-a2.l	d0.b
22	setdta	a0.l-a1.l	keine
23	writerec	a1.l	d0.b
44	fileload	a0.l-a1.l	d0.b
45	filesave	d1.w/a0.l-a1.l	d0.b
48	loadpart	a0.l-a1.l	d0.b
49	savepart	d2.w/a0.l-a1.l	d0.b
50	catalog	a0.l	keine
56	blockread	d2.w/a0.l-a1.l	d0.b
57	blockwrit	d2.w/a0.l-a1.l	d0.b
71	fileinfo	a1.l	d0.b/d1.l/d2.l/d3.b
74	directory	d1.w/d2.b/d3.w/a0.l-a1.l	d0.b
83	chmode	d1.b/a1.l	d0.b
91	loadauto	a0.l/a2.l	d0.b
93	setrec	d1.w/a1.l	d0.b

Gruppe: Dialogführung

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
1	getname	a0.l-a1.l	d0.b
2	getstadd	a2.l	a0.l
4	loadtext	a2.l	d0.b
6	response	keine	keine

Gruppe: Laufwerke

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
5	motoroff	keine	keine
51	floppy	d1.w/d2.b/d3.b/d4.w/a0.l	d0.b
52	drivecod	d4.w	d4.w
53	getdrive	keine	d0.w
54	setdrive	d0.w	keine
72	diskinfo	d4.b/a0.l	d0.b/a0.l
82	getdisks	keine	a0.l
85	getdpath	keine	a0.l
88	asctonum	d0.b	d0.b
89	numtoasc	d0.b	d0.b
90	gethdisk	keine	a0.l

Gruppe: Residente Programme

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
75	table	a0.l	keine
76	respinfo	keine	a0.l-a2.l

Gruppe: Speicherhandling

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
33	movelbin	d1.w/a1.l-a2.l	keine
34	moveltxt	a1.l-a2.l	keine
35	moverbin	d1.w/a1.l-a2.l	keine

Gruppe: Stringhandling

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
0	getleng	a0.l	d1.w
8	strgcomp	a0.l-a1.l	d0.b
11	uppercas	a0.l	keine
59	skipchar	d1.b/a0.l	a0.l
60	delete	d1.w/a0.l	keine
61	insert	d2.w/a0.l-a1.l	keine

Gruppe: System

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
24	getversi	keine	d0.l
25	getparm	keine	a0.l
31	inport	d1.l	d0.b
32	outport	d0.b/d1.l	keine
36	wrtcmd	a0.l	keine
37	gtretcod	keine	d0.l
38	stretcod	d0.l	keine
39	moveline	d2.w/a0.l/a2.l	d0.b
42	getparm1	keine	a0.l
43	getparm2	keine	a0.l
46	getparm3	keine	a0.l
47	getparm4	keine	a0.l
55	gttraptb	keine	a0.l
58	getladdr	keine	a0.l
62	ramtop	keine	a0.l
78	getcpu	keine	d0.l
79	getclock	keine	d0.l
80	getgrund	keine	d0.l/a0.l
81	getsound	keine	d0.l/a0.l
84	gtcmdtab	keine	a0.l-a1.l
86	clrovwrt	keine	keine
87	setovwrt	keine	keine
92	cmdexec	a0.l	keine

Gruppe: Textausgabe

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
7	schreibe	a0.l	keine
12	wrblank	d1.w	keine
13	wrint	d0.w	keine
26	hardcopy	keine	keine
40	wraddr	d0.l	keine
73	wrlint	d0.l	keine

Gruppe: Texteingabe

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
3	lese	d2.w/a1.l	d0.b
77	lineedit	d2.w/a1.l	d0.b

Gruppe: Uhren

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
65	getuhr	a0.l	keine
66	setuhr	a0.l	d0.b
67	date	a0.l	a0.l
68	time	a0.l	a0.l
69	datim	a0.l	a0.l
70	setdatim	a1.l	d0.b
68	time	a0.l	a0.l

Gruppe: Zeicheneingabe

TRAP Nr.	Befehls-name	Eingabe-Register	Ausgabe-Register
41	ci	keine	d0.b

INHALTSVERZEICHNIS

Vorbemerkungen 2

Beschreibung der Funktionen

getleng	0	3
getname	1	4
getstadd	2	5
lese	3	6
loadtext	4	7
motoroff	5	8
response	6	9
schreibe	7	10
strgcomp	8	11
tload	9	12
tsave	10	13
uppercas	11	14
wrblank	12	15
wrint	13	16
close	14	17
copyfile	15	18
create	16	19
erase	17	20
fillfcb	18	21
open	19	22
readrec	20	23
rename	21	24
setdta	22	25
writerec	23	26
getversi	24	27
getparm	25	28
hardcopy	26	29
bell	27	30
beep	28	31
errnoise	29	32
sound	30	33
inport	31	34
outport	32	35
movelbin	33	36
moveltxt	34	37
moverbin	35	38
wrtcmd	36	39
gtretcod	37	40
stretcod	38	41
moveline	39	42
wraddr	40	43
ci	41	44
getparm1	42	45
getparm3	43	46
fileload	44	47
filesave	45	48
getparm3	46	49
getparm4	47	50

loadpart	48	51
savepart	49	52
catalog	50	54
floppy	51	55
drivecod	52	57
getdrive	53	58
setdrive	54	59
gttraptb	55	60
blockread	56	61
blockwrit	57	62
getladdr	58	63
skipchar	59	64
delete	60	65
insert	61	66
ramtop	62	67
dummy	63	68
dummy	64	69
getuhr	65	70
setuhr	66	71
date	67	72
time	68	73
datim	69	74
setdatim	70	75
fileinfo	71	76
diskinfo	72	77
wrlint	73	78
directory	74	79
table	75	80
respinfo	76	81
lineedit	77	82
getcpu	78	83
getclock	79	84
getgrund	80	85
getsound	81	86
getdisks	82	87
chmode	83	88
gtcmdtab	84	89
getdpath	85	90
clrovwrt	86	91
setovwrt	87	92
asctonum	88	93
numtoasc	89	94
gethdisk	90	95
loadauto	91	96
cmdexec	92	97
setrec	93	98

Sortierung nach TRAP-Nummern Teil 1 99

Sortierung nach TRAP-Nummern Teil 2 101

Sortierung nach dem Namen Teil 1 103

Sortierung nach dem Namen Teil 2 105

Sortierung nach Gruppen und TRAP-Nummern
Gruppe Akustik 107
Gruppe Dateihandling 107
Gruppe Dialogführung 108
Gruppe Laufwerke 108
Gruppe Residente Programme 108
Gruppe Speicherhandling 109
Gruppe Stringhandling 109
Gruppe System 110
Gruppe Textausgabe 110
Gruppe Texteingabe 111
Gruppe Uhren 111
Gruppe Zeicheneingabe 111