SYSTEM - Beschreibung

(DEBUG-Handbuch)

zu Genie-DOS

Reg.-Nr.: **83 - 032**

1.Auflage 0-100, Oktober 1983.

JHL Heicke, Postfach 100847, 5000 Köln 1

Vorwort

Das vorliegende Hand-Buch soll dem Programmierer die Möglichkeiten aufzeigen, wie er seine Programme mit Hilfe des Betriebssystems effektiver gestalten kann.

Es befaßt sich mit dem Aufbau des Inhaltsverzeichnisses, den Controllblöcken für Peripherie und Dateien, dem Ablauf und der Handhabung der von G-DOS zur Verfügung gestellten Systemroutinen und Restartbefehle.

Insbesondere die Beschreibung der Systemeinsprungpunkte wird Ihnen bei der Programmierung sehr hilfreich sein.

Absichtlich wurde als Einband ein Ringbuch gewählt, damit spätere Erweiterungen problemlos eingefügt oder Änderungen, falls nötig, ausgetauscht werden können.
Dies erlaubt dem Benutzer auch das Ablegen eigener Notizen an der betreffenden Stelle des Handbuchs.

Dieses Handbuch gibt keine Hilfestellung bei der Programmerstellung in Maschinenebene, sondern zeigt nur, wie die einzelnen Systemroutinen angesprungen werden müssen.
Kenntnisse der Z-80 Befehle und Ihre Wirkungsweise werden vorausgesetzt.

Aktueller Stand der Beschreibung ist G-DOS 3.0. Spätere und frühere Versionen werden sicherlich in einigen Punkten abweichen. Die Systemroutinen $DELIM, $FILLER, $MULT, $DIV und $HEXDE stehen erst ab Version 3.0 zur Verfügung. Ebenso war der Datumseintrag bisher nicht definiert.

Wir haben uns bemüht dieses Handbuch fehlerfrei zu gestalten. Bei Unklarheiten und abweichendem Verhalten der einzelnen Routinen sind wir für Anregungen und Hinweise dankbar.

Dem Handbuch beigelegt ist eine Registrierkarte, die uns erlaubt Ihnen die Korrekturen und Ergänzungen der nachfolgenden Auflage kostenfrei zu übersenden. abgeschickt am 18.12.83

Köln, Oktober 1983 J.H.L. Heicke

INHALTSVERZEICHNIS / DIRECTORY
==========

Inhaltsverzeichnis

Im Inhaltsverzeichnis werden neben den Dateinamen auch die der Datei und der Diskette zugehörigen Informationen abgelegt. Im einzelnen sind dies:

1) Speicherbelegungsplan (GAT-Sektor)

Der erste Sektor des Inhaltsverzeichnisses beinhaltet den Speicherbelegungsplan (Granule Allocation Table).
Genie-DOS ermöglicht die Aufteilung des Speichermediums in bis zu 192 Blöcke zu je maximal 8 Einheiten (Granule) mit normalerweise 5 Sektoren zu je 256 Bytes. Rein rechnerisch ergibt sich hieraus die Möglichkeit mit einem Inhaltsverzeichnis ein Speichermedium von bis zu 192*8*5 Sektoren, also 1920 kByte, zu organisieren. Ein 8-Zoll-Doppelkopflaufwerk ermöglicht in doppelter Dichte die Abspeicherung von 77*54 Sektoren, also 1039.5 kByte.
Bei Festplattenspeichern ergibt sich somit die Notwendigkeit, entweder die Einheitengröße von 5 Sektoren zu verändern, oder aber die Festplatte in mehrere logische Medien zu unterteilen. Die Festplatte verhält sich dann wie mehrere Diskettenlaufwerke.

Um eine gewisse Übereinstimmung zwischen logischer und realer Speicheraufteilung zu erreichen, wurde bei Genie-DOS nachfolgende Blockgrößen gewählt:
2 Einheiten im Block bei einseitigen 5-Zoll-Disketten in einfacher Dichte, hier entspricht ein Block einer Spur zu 10 Sektoren.
4 Einheiten im Block bei doppelseitigen 5-Zoll-Disketten in einfacher Dichte mit 2*10 Sektoren je Spur.
5-Zoll-Disketten werden in doppelter Dichte mit jeweils 18 Sektoren je Spur und Seite beschrieben. Hieraus bietet sich an, 3 bzw. 6 (bei doppelseitigen Disketten) Einheiten im Block zusammenzufassen, da dadurch nach jeweils 5 Spuren, also 90 (180) Sektoren, 6 Blöcke enden. Dies erleichtert die Berechnung der Lage physikalischer Sektoren bei direktem Diskettenzugriff (z.B. zur Rettung zerstörter Dateiinhalte).

Im unteren Bereich des GAT-Sektors sind Diskettenkennwort, -name und -datum gespeichert. Die letzten 32 Bytes sind zur Abspeicherung des AUTOmatischen Startbefehls reserviert.

2) Such-Index (HIT-Sektor)

Der zweite Sektor des Inhaltsverzeichnisses beinhaltet den Such-Index (Hashcode Index Table).
Jeder im Inhaltsverzeichnis eingetragene Dateiname wird in einen Hashcode von einem Byte verschlüsselt. Ergibt sich bei der Verschlüsselung zufällig 0, so wird der entsprechende Hashcode auf 1 erhöht. Somit entsprechen alle Bytes ungleich 0 einer Datei im Inhaltsverzeichnis.
Die Hashcodes sind in insgesamt 8 Reihen zu 32 Bytes angeordnet. Die Position innerhalb der Reihe entspricht dem Sektor des Dateieintrags. Da bereits 2 Sektoren des Inhaltsverzeichnisses vergeben sind (GAT und HIT) und das Inhaltsverzeichnis eine durch 5 teilbare Sektorenzahl haben muß, bleiben die letzten 4 Bytes frei (2+28=30 Sektoren, 6 Einheiten). In der ersten Reihe wird das letzte Byte dazu genutzt, um die Anzahl der die Mindestgröße des Inhaltsverzeichnisses (10 Sektoren, 2 Einheiten) überschreitenden Sektoren abzuspeichern.

Bei Zugriff auf das Inhaltsverzeichnis wird der Dateiname aufgrund eines übereinstimmenden Hashcodes mit dem Dateieintrag in dem entsprechenden Sektor und der entsprechenden Reihe verglichen. Somit wird nur dann ein Sektor des Inhaltsverzeichnisses gelesen, wenn dies aufgrund übereinstimmender Hashcodes erforderlich erscheint. Hashcodes dienen also dem schnelleren Zugriff.

Die relative Sektorposition des treffenden Hashcodes ergibt den Dateieintragszeiger (DEC = Directory Entry Code). Dieser Code dient bei allen weiter folgenden Dateizugriffen als Verbindung zwischen Datei-Control-Block und Inhaltsverzeichnis.

SS SD ST

10 SEK
2 EIB (10 Sektoren)
17 SBIV (17·10 = 170 Sektor)
2 AEIV (10 Sektoren)

3) Dateieintrag (FDE)

Die nachfolgenden Sektoren beinhalten die Dateieinträge in 8 Reihen zu 32 Bytes. Der Übersicht halber wird nachfolgend die jeweils logische Position der einzelnen Bytes angegeben.

Byte 00 bit 7 Erweiterungseintrag (FXDE) siehe dort 128
bit 6 Systemdatei 64
bit 5 null 32
bit 4 Eintrag besteht 16
bit 3 unsichtbar (INV) 8
bit 2,1,0 Zugriffsstufe 0 bis 7 4,2,4

Byte 01 bit 7 F-Flag: sperrt Dateiplatz-Freigabe
bit 6 E-Flag: sperrt Datei-Erweiterung
bit 5 B-Flag: Bearbeitungskennzeichen
bit 4,3,2,1,0 Tag des Datums bei Dateieröffnung

Byte 02 bit 7,6,5,4 Jahr des Datums - 1980
bit 3,2,1,0 Monat des Datums (0-15)

Byte 03 EOF-Byte

Das End-of-File-Byte zeigt hinter die Position des letzten zur Datei gehörenden Byte. Dies ist die niederwertige Adresse der EOF-Position (logische Schreibweise)

Byte 04 log

Die logische Satzlänge wird bei der Berechnung der Anzahl der Sätze der Datei benutzt, nicht jedoch bei Dateizugriffsfunktionen (0=256).

Byte 05 bis Byte 0C 8 Bytes für den Namen der Datei

Die fehlenden Zeichen werden durch Blanks rechts ergänzt.

Byte 0D bis Byte 0F 3 Bytes für den Typ der Datei

Die fehlenden Zeichen werden durch Blanks rechts ergänzt.

Byte 10,11 Codierung des Hauptkennworts

Dem aus bis zu 8 alphanumerischen Zeichen bestehenden Kennwort wird ein Doppelbyte zugeordnet.

Byte 12,13 Codierung des Bearbeiterkennworts

Dem aus bis zu 8 alphanumerischen Zeichen bestehenden Kennwort wird ein Doppelbyte zugeordnet.

Byte 14,15 EOF-Sektor

Dies sind die höherwertigen Adressen der EOF-Position. Bitte beachten Sie, daß hier im Gegensatz zum FCB die tatsächliche Anzahl der Sektoren der Datei angegeben wird. Ist das EOF-Byte gleich 0, so bedeutet dies, daß alle 256 Bytes des letzten Sektors zur Datei gehören.

Byte 16 Blockzuordnung

Dieses Byte nennt den Block (0-191) des Speicherbelegungsplans der den folgenden Dateiteil enthält. Ist dieses Byte FF, so folgt kein Zeiger auf zugeordnete Einheiten.

Byte 17 Einheitenzuordnung

Die drei oberen bits (bit 7,6,5) geben die Einheit innerhalb des Blocks an, die der Datei zugeordnet ist.
Die fünf unteren bits (bit 4,3,2,1,0) geben die Anzahl der weiteren Einheiten an, die der ersten Einheit unmittelbar folgen und der Datei zugeordnet sind.

Byte 18,19 sowie 1A,1B und 1C,1D entsprechend Byte 16,17

Byte 1E Erweiterungsmarkierung

Ist dieses Byte FF, so folgt kein Erweiterungseintrag. Ist dieses Byte FE, so beinhaltet das nachfolgende Byte den Zeiger (DEC) auf den Erweiterungseintrag.

Byte 1F Erweiterungszeiger

Die drei oberen bits (bit 7,6,5) ergeben die Reihe in der der Erweiterungseintrag zu finden ist.
Die fünf unteren bits (bit 4,3,2,1,0) ergeben die logische Sektornummer-2 im Inhaltsverzeichnis des den Erweiterungseintrag enthaltenden Sektors.

4) Erweiterungseintrag zum Dateientrag (FXDE)

Byte 00 bit 7 kennzeichnet Erweiterungseintrag
bit 6 null
bit 5 null
bit 4 aktive Zuordnung
bit 3,2,1,0 null

Byte 01 Rückverweis auf vorhergehenden DEC

Byte 02 bis 15 null

Byte 16 bis 1F wie bei Dateieintrag definiert

5) Kennworte

Genie-DOS benutzt drei Kennworte:

Das Diskettenkennwort erlaubt den Status der Diskette zu ändern. Dieses Kennwort wird immer dann benutzt, wenn keine bestimmte Datei angesprochen wird und ist in den Dateikennworten übergeordnet.

Das Hauptkennwort dient zur Festlegung der Zugriffsstufe auf eine bestimmte Datei.

Das Bearbeitungskennwort erlaubt die Überschreitung der Zugriffsstufe, nicht aber die Änderung derselben.

6) Datum

Sowohl die Diskette als auch alle Dateien sind mit einem Datumseintrag versehen. Der Datumseintrag der Datei ist in 13 bit codiert, das Diskettendatum wird im GAT-Sektor (D8-DF) in Klarschrift abgelegt.

7) Besondere Dateikennzeichen

Genie-DOS hat drei definierte Dateikennzeichen:

F-Flag verhindert die Freigabe des der Datei zugeordneten Speicherplatzes, wenn nicht alle Einheiten, die bereitgestellt wurden, benötigt werden. Dies verhindert das Zerreißen bei Dateien deren Größe sich während des Programmablaufs ändern kann.

E-Flag verhindert die Erweiterung der Datei über den bereits zugewiesenen Speicherplatz hinaus.

B-Flag zeigt eine Bearbeitung der Datei an. Dieses Kennzeichen wird bei einem Schreibvorgang vom System gesetzt.

8) Lage des Inhaltsverzeichnisses

Um einen gleichmäßig schnellen Zugriff auf alle im Inhaltsverzeichnis abgelegten Dateien zu gewährleisten, sollte das Inhaltsverzeichnis in der Mitte des verfügbaren Diskettenplatzes angelegt sein. Bei alten 35-Spur-Laufwerken befand sich das Inhaltsverzeichnis auf Spur 17. Bei 80-Spur-Doppelkopflaufwerken findet sich das Inhaltsverzeichnis ab Sektor 1440/5A0H wenn in doppelter Dichte insgesamt 2880 Sektoren bereitstehen. Andere Disketten-Betriebssysteme positionieren ihr Inhaltsverzeichnis auf Spur 0, dies ergibt sich aus der Nutzung eines speziellen Steuerbefehls für den Floppy-Disk-Controller. Bei Genie-DOS wird der Magnetkopf jedoch auf der aktuellen Spur belassen.

9) Lage der Systemprogramme im Inhaltsverzeichnis

Die ersten vier Reihen der ersten 8 Dateieintragssektoren sind für Systemdateien reserviert. Systemdateien werden aufgrund ihrer Position im Inhaltsverzeichnis geladen und nicht namentlich aufgerufen. Es sind mindestens zwei Systemdateien im Inhaltsverzeichnis eingetragen, nämlich GDOS/SYS (BOOT/SYS) und INHALT/SYS (DIR/SYS). Dadurch wird die Zahl der Dateieinträge auf Datendisketten auf maximal 222 begrenzt.

Alle anderen Systemdateien sind nur auf Systemdisketten notwendig, und von 0 bis 29 durchnumeriert. SYSO/SYS wird von GDOS/SYS bei Systemstart geladen und muß in Sektor 5 der Diskette beginnen. Die verbleibenden Systemdateien sind rund um das Inhaltsverzeichnis abgelegt. Die entsprechende Positionierung wurde gemäß Reihenfolge und Häufigkeit des Aufrufs gewählt.

Bei 80-Spur DD/DS wurde auch das Disk-BASIC in den Bereich um das Inhaltsverzeichnis eingebunden. Alle weiteren Programme werden oberhalb von SYSO/SYS abgelegt, somit werden die kritischen inneren Spuren der Diskette erst als letzte belegt.

Peripheriesteuerblock / DCB

Dateisteuerblock / FCB

Systemroutinen

Restartbefehle

Der DCB (Device Control Block, Steuerblock für Peripherie)

Die Byte-Lese-Routine und die Byte-Schreibe-Routine erwarten in Register DE den Zeiger auf einen DCB zur Steuerung der nachfolgenden Datenübertragung. Register B enthält dabei den Typ des DCB. Register A dient als Transportregister.

DCB+0 TYP des DCB

Wenn Bit 7 von DCB+0 gesetzt und Bit 6 rückgesetzt ist, handelt es sich um einen FCB, der in den nachfolgenden Bytes völlig anders definiert ist.

Wenn nur Bit 7 und Bit 6 gesetzt sind, bedeutet dies eine Umleitung der Datenübertragung auf eine andere Peripherie.

Die niederwertigen sechs Bits werden zur eigentlichen Typkennzeichnung benutzt.

Bit 0	(01)	Eingabeeinheit, CARRY wenn B<2
Bit 1	(02)	Ausgabeeinheit, ZERO wenn B=2
Bit 2	(04)	
Bit 3	(08)	
Bit 4	(10)	
Bit 5	(20)	

DCB+1 Treiberadresse (niederwertiges Byte)
DCB+2 Treiberadresse (höherwertiges Byte)

DCB+3 Hilfsspeicher für Treiber-Routine
DCB+4 Hilfsspeicher für Treiber-Routine

DCB+5 Zwischenspeicher für Daten

DCB+6 frei
DCB+7 frei

TYP	Peripherie	DCB-Adresse	Treiber	Systemcall	Name
1	Tastatur	04015H	003E3H	0002BH	$INKEY
7	Monitor	0401DH	00458H	00033H	$DISPL
6	Drucker	04025H	0058DH	0003BH	$PRT

Der FCB (File Control Block, Datensteuerblock):

Der Datensteuerblock wird nachfolgend kurz FCB genannt. Im Gegensatz zum DCB handelt es sich hier um ein Speichermedium, während der DCB nur Ein/Ausgabe-Medien beschreibt. Zur Unterscheidung wird im FCB das höchste Bit (Bit 7) des ersten Bytes (Byte 00) auf 1 gesetzt. Nachfolgend wird die Bedeutung der einzelnen Bytes, beginnend mit dem relativen Byte 00, beschrieben. Teilweise ist die Bedeutung der Bytes bereits im Kapitel Inhaltsverzeichnis, Abschnitt 3) Dateieintrag (FDE) erklärt. Im nachfolgenden Text wird gesondert darauf hingewiesen. Diese Bytes stimmen in ihrer Bedeutung im FDE und FCB überein.

Byte 00 bit 7 Markierung für FCB
bit 6 unbenutzt
bit 5 unbenutzt
bit 4 unbenutzt
bit 3 unbenutzt
bit 2 unbenutzt
bit 1 Ein/Ausgabe auf Disk ohne Dateieintrag
bit 0 Data-Adressmarke (DAM) wie Inhaltsverzeichnis

Byte 01 bit 7 Einzel-Byte Ein/Ausgabe
bit 6 RANDOM-ACCESS Bit (Keine Änderung des EOF)
bit 5 Sektor noch nicht gelesen
bit 4 Sektor noch nicht geschrieben
bit 3 Länge des FCB ist 32 bytes
bit 2,1,0 Zugriffsstufe 0 bis 7

Byte 02 bit 7 Systemoption ADE=N
bit 6 Systemoption ADF=N
bit 5 Bearbeitungskennzeichen
bit 4,3,2,1,0 Tag des Datums

Byte 03,04 Adresse des FCB-Sektor-Buffers

Bei Schreib-/Leseoperationen mit Speichermedien wird ein Puffer benötigt, der den Inhalt des aktuellen Sektors zwischenspeichert.

Byte 05 relative Schreib-/Leseposition innerhalb des Sektors

Zeigt bei Einzelbyte Ein-/Ausgabe auf das nächste zu bearbeitende Zeichen.

Byte 06 bit 7 unbenutzt
bit 6 unbenutzt
bit 5 unbenutzt
bit 4 unbenutzt
bit 3 unbenutzt
bit 2,1,0 Laufwerksnummer

Byte 07 Dateieintragszeiger

Zeigt auf den zutreffenden Hashcode im Hash-Index-Table.
(siehe Inhaltsverzeichnis, Abschnitt 2)

Byte 08 EOF-Byte

Das End-Of-File-Byte zeigt hinter die Position des letzten zur Datei gehörenden Bytes.

Byte 09 Logische Satzlänge

Dient zur Berechnung der Anzahl der Sätze der Datei.
(0=256 Bytes)

Byte 0A,0B Sektornummer

Nummer des aktuellen Sektors, relativ zum Anfang der Datei (oder Diskette, wenn Byte 00 bit 1 gesetzt).

Byte 0C,0D EOF-Sektor Nummer

Höherwertige Adressen der EOF-Position, ergibt mit dem EOF-Byte die relative Byte-Adresse des ersten nicht zur Datei gehörenden Bytes.

Byte 0E Blockzuordnung

GAT-Sektor Byte oder 0FFH (kein Eintrag), 0FEH
(siehe Inhaltsverzeichnis, Dateieintrag, Byte 16).

Byte 0F Einheitenzuordnug

bit 7,6,5 Starteinheit im Block (0-7)
bit 4,3,2,1,0 Anzahl der aufeinander folgenden Einheiten
(siehe Inhaltsverzeichnis, Dateieintrag, Byte 17).

Byte 10...1F je zwei Bytes für Erweiterungen
(wie Byte 0EH und 0FH)

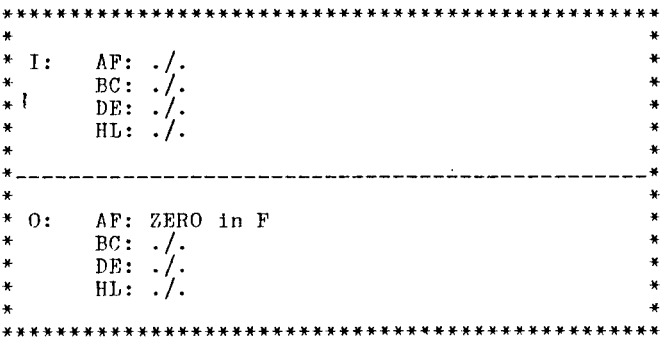
D A T E N S T E U E R B L O C K (F C B)

Adresse:	* BIT 7	* BIT 6	* BIT 5	* BIT 4	* BIT 3	* BIT 2	* BIT 1	* BIT 0	* Name:
FCB+00H	* FCB	*	*	*	*	*	* Disk	* DAM (IV)	* XFCB
FCB+01H	* Byte-I/O	* RANDOM	* lesen	*schreiben	* FCB 32	* Zugriffs-Stufe (0-7)			* XFCBA
FCB+02H	* ADE=N	* ADF=N	* BEA=J	* Tag des Datums (0-31)					* XFCBB
FCB+03H	* niederwertiges Byte der Startadresse des FCB-Sektorpuffers								* XFCBIO
FCB+04H	* höherwertiges Byte der Startadresse des FCB-Sektorpuffers								* XFCBIO+1
FCB+05H	* aktuelle Byte-Position im Sektor								* XFCBAB
FCB+06H	*	*	*	*	*	* Laufwerksnummer (Drive)			* XFCBDR
FCB+07H	* Dateieintragszeiger (DEC) ins Inhaltsverzeichnis (Position im HIT-Sektor)								* XFCBHT
FCB+08H	* End-of-File Byte								* XFCBEB
FCB+09H	* Logische Satzlänge								* XFCBLL
FCB+0AH	* Aktuelle Sektornummer (niederwertiges Byte)								* XFCBAS
FCB+0BH	* Aktuelle Sektornummer (höherwertiges Byte)								* XFCBAS+1
FCB+0CH	* End-of-File Sektor (niederwertiges Byte)								* XFCBES
FCB+0DH	* End-of-File Sektor (höherwertiges Byte)								* XFCBES+1

SYSTEMEINSPRUNGTABELLE
==========

4400H	$DOS	kehrt zur Befehlseingabe zurück
4403H	MENTRY	2-Byte Einsprungadresse nach $LOAD
4405H	$DOSCMD	ruft Maschinenprogramm oder DOS-Befehl auf
4408H	$NERROR	ret z ;sonst
4409H	$DERROR	Fehlermeldung ausgeben
440DH	$DEBUG	Systemmonitor aufrufen
4410H	$ENQUE	Interruptroutine einfügen
4413H	$DEQUE	Interruptroutine ausschalten
4416H	$RESEL	wählt aktuelles Laufwerk erneut an
4419H	$DOSCAL	wie $DOSCMD, kehrt aber zurück
441CH	$EXFIL	prüft und überträgt Dateinamen in FCB
441FH	MGDOS0	systemintern reserviertes Merkbyte
4420H	$INIT	initialisiert Datei und eröffnet FCB
4423H	MGDOS1	systemintern reserviertes Merkbyte
4424H	$OPEN	eröffnet FCB für existierende Datei
4427H	MGDOS2	systemintern reserviertes Merkbyte
4428H	$CLOSE	schließt FCB
442BH	MGDOS3	systemintern reserviertes Merkbyte
442CH	$KILL	löscht Dateieintrag
442FH	MGDOS4	systemintern reserviertes Merkbyte
4430H	$LOAD	lädt Maschinenprogramm
4433H	$RUN	startet Maschinenprogramm
4436H	$RDSEC	liest Sektor
4439H	$WRSEC	schreibt Sektor
443CH	$WRSECV	schreibt Sektor und prüft Inhalt
443FH	$POS0	setzt FCB-Sektornummer auf Zero
4442H	$POSBC	setzt FCB-Sektornummer auf (BC)
4445H	$POSDEC	decrementiert FCB-Sektornummer
4448H	$POSEOF	setzt FCB-Sektornummer ans Dateiende
444BH	$ALLOC	reserviert Einheit auf der Diskette für Datei
444EH	$POSRBA	positioniert FCB auf relative Byte-Adresse
4451H	$WREOF	Dateiende ins Inhaltsverzeichnis eintragen
4454H	$DELIM	prüft Delimiter bei Parametern
4457H	MGDOS5	systemintern reserviertes Merkbyte
4458H	$FILLER	vergleicht Füllwort/Parameter
445BH	$DRVSEL	wählt Laufwerk an
445EH	$DSKMNT	testet ob Disk in Laufwerk
4461H	$NAMENQ	erweitert den Befehlsvorrat um geladene Routine
4464H	$NAMDEQ	löscht Befehl aus erweitertem Befehlssatz
4467H	$PRINT	gibt Text an Monitor
446AH	$LPRINT	gibt Text an Drucker
446DH	$CONTIM	gibt 8 Bytes Uhrzeit
4470H	$CONDAT	gibt 8 Bytes Datum
4473H	$DEFEXT	ergänzt Typbezeichnung
4476H	$MULT	multipliziert A*HL=AHL
4479H	$DIV	dividiert HL/A=HL Rest A
447CH	$HEXDE	übergibt Hexadezimaldarstellung von DE
447FH	MGDOS6	systemintern reserviertes Merkbyte
4480H	DOSFCB	systeminterner FCB von 32 Bytes Länge

Rückkehr zum Betriebssystem ohne Fehleranzeige



Bei jedem Jump oder Call auf $RET überprüft das Betriebssystem seinen Status:

1 DOS-Status

Der Stackpointer wird auf den DOS-Stackbereich gesetzt.
Die Betriebssystem-Meldung wird angezeigt.
Das Betriebssystem wartet auf die nächste Befehlseingabe.
Der Status der BREAK-Taste richtet sich nach Systemoption AG.

2 DOS-CALL-Status

Der Stackpointer wird auf die Position zurückgesetzt, die bei Aufruf der nun beendeten Stufe bestand.

2.1 DOS-CALL ohne aktiven JOB

Bei DOS-CALL ohne JOB-Ausführung oder bei unterbrochener JOB-Ausführung (siehe DOS-Befehl 'CONT'), werden alle Register außer AF auf ihre früheren Werte zurückgesetzt. Im Flag-Register ist das ZERO-Flag gesetzt.

2.2 DOS-CALL mit unterbrochenem JOB

Bei Rückkehr aus DOS-CALL wird die JOB-Datei wieder aktiviert, wenn

ihre Ausführung mit CONT,J (bzw /./5J) unterbrochen war.

2.3 Beendigung des DOS-CALL-Status

Bei Rückkehr aus der ersten (also zu diesem Zeitpunkt einzigen) DOS-CALL-Stufe wird der DOS-CALL-Status des Betriebssystems gelöscht.

3 MINI-DOS-Status

Eine Rückkehr in diesen Status ist möglich:

a) nach Aufruf des Systemmonitors (DEBUG)
b) bei Beendigung einer *name-Routine
c) nach dem DOS-Befehl ´GO´

Der Stackpointer wird auf die Position zurückgesetzt, die vor Aufruf der durchlaufenen Programmebene bestand.
Die Betriebssystem-Meldung wird mit dem Zusatz "Mini-" ausgegeben.
Das Betriebssystem wartet auf die nächste Befehlseingabe.

4 RUN-ONLY-Status (Ablaufunterbrechung untersagt)

Bei RUN-ONLY kann an dieser Stelle keine manuelle Eingabe erfolgen.

4.1 RUN-ONLY ohne aktiven JOB

Bei RUN-ONLY ohne aktiven JOB wird die Meldung "RUN ONLY STOPPED" angezeigt. Die folgende Benutzereingabe wirkt sich wie der Befehl ´BOOT´ aus.

Anmerkung:
Die JOB-Datei ersetzt die manuelle Eingabe des nächstfolgenden Befehls.

Fehlerausgang zum DOS ohne Fehlermeldung

```
*************************************************
*                                               *
* I:   AF: beliebig                             *
*      BC: ./.                                  *
*      DE: ./.                                  *
*      HL: ./.                                  *
*                                               *
*-----------------------------------------------*
*                                               *
* O:   AF: CARRY in F (bei DOS-Call)            *
*      BC: .?.                                  *
*      DE: .?.                                  *
*      HL: .?.                                  *
*                                               *
*************************************************
```

Diese Routine erweitert die Möglichkeiten ein Programm abzuschliessen, wenn bei verschachtelten Aufrufen eine Fehlerbehandlung notwendig wird.

1 Fehleranzeige

Bei einen Jump oder Call zu dieser Adresse geht das Betriebssystem davon aus, daß der Fehler entweder bereits angezeigt wurde oder nicht angezeigt werden soll.

2 Unterschied zu $RET

Das Betriebssystem verhält sich bis auf zwei Ausnahmen wie bei $RET.

2.1 Verhalten bei JOB

Eine JOB-Ausführung wird abgebrochen.

2.2 Verhalten bei DOS-Call

Unter DOS-Call wird das Carry Flag gesetzt.

Normaler Ausgang zum DOS

```
**************************************************
*                                                *
* I:    AF: ./.                                  *
*       BC: ./.                                  *
*       DE: ./.                                  *
*       HL: ./.                                  *
*                                                *
*------------------------------------------------*
*                                                *
* O:    AF: ./.                                  *
*       BC: ./.                                  *
*       DE: ./.                                  *
*       HL: ./.                                  *
*                                                *
**************************************************
```

Ein Jump oder Call zu dieser Adresse führt zu dem selben Ergebnis wie bei Adresse 402DH.
Hierdurch ergibt sich die Möglichkeit bei Bedarf zu einer Anwenderroutine zwischen 402DH und 4400H zu verzweigen.

Ausführen eines DOS - Befehls

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: zeigt auf DOS - Befehl                *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z/NZ in F, Error in A                 *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: zeigt hinter DOS - Befehl             *
*                                                *
**************************************************
```

Diese Routine benutzt den Stack des Betriebsystems.
Das HL - Register zeigt auf einen Text, welcher mit ODH abgeschlossen und mit einem ausführbaren Befehl beginnen muß.
Dieser Text darf nicht länger als der Befehlspuffer des Betriebssystems sein, da er vor Befehlsausführung dorthin übertragen wird.
Nach Ausführung erfolgt die Rückkehr ins Betriebssystem.

Wenn der DOS-Befehl ein Programm aufruft, kann dieses seinen eigenen Stackbereich benutzen.
Die Rückkehr aus diesem Programm darf nur über einen der drei folgenden Aussprünge erfolgen: 402DH, 4030H oder 4409H.
Außerdem kann bei Programmende ein Zwei-Byte-Parameter nach 4403H und 4404H (17411, 17412) übergeben werden, dort steht ansonsten nach $LOAD und $RUN die Startadresse des geladenen Programms.

Fehlermeldung ausgeben, wenn NZ-Status

```
**************************************************
*                                                *
* I:   AF: A=Fehlercode                          *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: ./.                                   *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Error in A, NZ u. NC in F             *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: ./.                                   *
*                                                *
**************************************************
```

Diese Routine verhindert die Durchführung von $DERROR, wenn Z-Flag gesetzt ist. Bei NZ wird $DERROR wie nachfolgend beschrieben ausgeführt.

Im ACCU befindet sich der Fehlercode.

Bit 7 in A = 0 Programm beenden, Rückkehr zu DOS
Bit 7 in A = 1 Fehler ausgeben, zurück zum Programm

Falls Bit 7=0 wird die JOB-Bearbeitung abgebrochen.

Im DOS-CALL-Modus wird NZ und NC State im Flag-Register gesetzt. Der Fehlercode befindet sich im A-Register. Der Fehler wird nicht angezeigt.

Ansonsten wird die Fehlermeldung ausgegeben und es erfolgt ein Aussprung entsprechend 402DH.

Durch setzen von Bit 7 in A und einem Call nach 4409H kann man den dem Error-Code in A entsprechenden Fehler anzeigen und behält die Kontrolle über das Programm. Dabei wird nur das Flag-Register geändert.

Ändert man die zwei Bytes bei 4409H in F7 C9, so kann man diese Routine dazu benutzen, um bei einem Fehler in den Systemmonitor zu gelangen anstatt die Fehlermeldung auszugeben.

Fehlermeldung ausgeben

```
************************************************
*                                              *
* I:   AF: A=Fehlercode                        *
*      BC: ./.                                 *
*      DE: ./.                                 *
*      HL: ./.                                 *
*                                              *
*----------------------------------------------*
*                                              *
* O:   AF: Error in A, NZ u. NC in F           *
*      BC: ./.                                 *
*      DE: ./.                                 *
*      HL: ./.                                 *
*                                              *
************************************************
```

Im ACCU befindet sich der Fehlercode.

Bit 7 in A = 0 Programm beenden, Rückkehr zu DOS
Bit 7 in A = 1 Fehler ausgeben, zurück zum Programm

Falls Bit 7=0 wird die JOB-Bearbeitung abgebrochen.

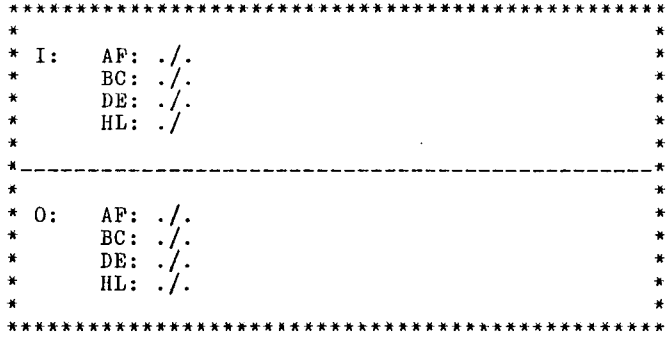
Im DOS-CALL-Modus wird NZ und NC State im Flag-Register gesetzt. Der Fehlercode befindet sich im A-Register. Der Fehler wird nicht angezeigt.

Ansonsten wird die Fehlermeldung ausgegeben und es erfolgt ein Aussprung entsprechend 402DH.

Durch setzen von Bit 7 in A und einem Call nach 4409H kann man den dem Error-Code in A entsprechenden Fehler anzeigen und behält die Kontrolle über das Programm. Dabei wird nur das Flag-Register geändert.

Ändert man die zwei Bytes bei 4409H in F7 C9, so kann man die Routine dazu benutzen, um bei einem Fehler in den Systemmonitor zu kommen ohne die Fehlermeldung auszugeben.

aufrufen des System-Monitors (DEBUG)



Für Benutzerprogramme bieten sich zwei Möglichkeiten an um den DEBUGGER aufzurufen:

(a) mit der RST 30H Funktion und
(b) mit einem CALL 440DH.

Wenn das PC - Register durch den DEBUGGER nicht geändert wurde, kann man mit G wieder ins aufrufende Programm zurückkehren. Dabei wird mit dem nächsten Befehl nach dem RST 30H oder dem CALL 440DH fortgefahren.

Die drei Bytes bei 440DH lassen sich selbstverständlich durch einen JP xxxx zu einem beliebigen vorher geladenen Monitor-Programm ersetzen.

Einfügen von Interruptroutinen

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: Adresse der Interrupt Routine         *
*      HL: ./.                                   *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: ./.                                   *
*                                                *
**************************************************
```

Diese Routine ändert die Register AF, BC, DE und HL.
Beim Einsprung in diese Routine steht in DE die Adresse der Interrupt - Routine ~~und zwar in der folgenden Weise:~~ die folgende Struktur hat

(a) die ersten zwei Bytes werden vom Betriebsystem als Folgezeiger gebraucht. Beim Einsprung in die Routine können hier beliebige Werte stehen.

(b) das dritte Byte ist die Anzahl der 25ms Intervalle, welche zwischen zwei Ausführungen der Routine vergehen sollen. Soll die Routine z.B. jede Sekunde ausgeführt werden, so muß das dritte Byte den Wert = 40 (28H) enthalten. Das Betriebssystem ändert dieses Byte nicht.

(c) das vierte Byte ist der Wert, um den zurückgezählt wird, bevor die Interruptroutine angesprungen wird.
Der Wert dieses Bytes muß größer sein als 0, aber kleiner oder gleich dem Wert im dritten Byte.
Das Betriebssystem vermindert diesen Wert alle 25ms.
Ist das Ergebnis nicht Null, so wird die Routine für diesen Interupt umgangen.
Ist das Ergebnis gleich Null, so wird das dritte Byte ins vierte transferiert, die Register HL, DE, BC und AF werden gesichert und die Benutzerroutine wird entsprechend dem fünften Byte aufgerufen.
Jedes Register, welches in der Routine gebraucht wird muß dort gesichert oder geholt werden.
Die Interrupts sind disabled und dürfen von der Benutzerroutine nicht enabled werden.

Während eine Benutzer - Interrupt - Routine in der Interrupt - Kette ist, darf sie nicht geändert werden, es sei denn, von einer Routine, welche mit Sicherheit die Interrupts disabled; die ersten beiden Bytes dürfen nicht geändert werden.

Herausnehmen einer Interrupt-Routine

```
*************************************************
*                                               *
* I:   AF: ./.                                  *
*      BC: ./.                                  *
*      DE: Adresse der Interrupt-Routine        *
*      HL: ./.                                  *
*                                               *
*-----------------------------------------------*
*                                               *
* O:   AF: .?.                                  *
*      BC: .?.                                  *
*      DE: .?.                                  *
*      HL: .?.                                  *
*                                               *
*************************************************
```

Diese Routine ändert die Register AF, BC, DE und HL.

Ein in der Interruptkette entsprechend dem Inhalt von DE eingefügte Routine wird aus der Kette herausgenommen.
Der Benutzer kann nun in der Routine Änderungen vornehmen.

erneutes Anwählen des aktuellen Laufwerkes

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: ./.                                   *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z/NZ in F, Error in A                 *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: ./.                                   *
*                                                *
**************************************************
```

Wenn die Diskettenlaufwerke rotieren, selektiert diese Routine das aktuelle Laufwerk erneut und läßt es ca. 2-3 Sekunden länger laufen.

Hierdurch kann man verhindern, daß bei längeren Datenübertragungen, während der Ausführung von programm-intern notwendigen Routinen, die Laufwerke abfallen und danach mit Zeitverlust erneut gestartet werden müssen.

Ausführen eines DOS - Befehls mit anschließender Rückkehr zum aufrufenden Programm

```
************************************************
*                                              *
* I:   AF: ./.                                 *
*      BC: ./.                                 *
*      DE: ./.                                 *
*      HL: zeigt auf DOS - Befehl              *
*                                              *
*----------------------------------------------*
*                                              *
* O:   AF: Z/NZ in F, Error in A               *
*      BC: ./.                                 *
*      DE: ./.                                 *
*      HL: zeigt hinter DOS - Befehl           *
*                                              *
************************************************
```

Die DOSCAL Routine benutzt nicht ihren eigenen Stack, sondern den des aufrufenden Programms. Bis auf AF werden alle Register auf diesen Stack geschoben und beim Rücksprung wieder zurückgeholt.
HL zeigt auf einen mit ODH abzuschliessenden Text, der die Länge des betriebssysteminternen Befehlspuffers nicht überschreiten darf.
Dieser Text muß mit einem in der DOS-Ebene gültigen Befehl beginnen.
Der DOS-CALL-Status des Betriebssystems wird gesetzt.
Das Betriebsystem sichert den aktuellen Stackpointer und führt den Befehl aus.
Diese Routine wird auch vom BASIC-Befehl CMD"befehl" angewandt.
Hierdurch ist auch der Aufruf eines Benutzerprogrammes erlaubt.
Jobfolgen sind auch unter DOSCAL erlaubt.

Der Benutzer muß sicherstellen, daß es zu keinerlei Speicherbelegungskonflikt kommen kann, und daß genügend Platz für den STACK vorhanden ist.

Geschachtelte Aufrufe sind hierbei möglich. Die Rückkehr von einer DOS-CALL Stufe erfolgt immer zur jeweils aufrufenden Stufe. Die Betriebssystemebene kann man somit als Stufe 0 betrachten.

Wenn die DOSCAL-Routine ein Programm aufruft, darf dieses seinen eigenen Stackbereich benutzen.
Die Rückkehr aus diesem Programm darf nur über einen der drei folgenden Aussprünge erfolgen: 402DH, 4030H oder 4409H.
Außerdem kann ein Zwei-Byte-Parameter nach 4403H und 4404H (17411, 17412) übergeben werden, dort steht ansonsten nach $LOAD und $RUN die Startadresse des geladenen Programms.

Übertragen eines Dateinamens

```
*************************************************
*                                               *
* I:   AF: ./.                                  *
*      BC: ./.                                  *
*      DE: FCB-Bereich für Dateiname            *
*      HL: Text der Dateiname enthält           *
*                                               *
*-----------------------------------------------*
*                                               *
* O:   AF: Z oder NZ                            *
*      BC: geändert                             *
*      DE: nicht geöffneter FCB                 *
*      HL: hinter Dateiname                     *
*                                               *
*************************************************
```

Diese Routine extrahiert einen Dateiname aus dem Text auf den HL zeigt, überträgt ihn in den Bereich der in DE angegeben wurde, und schließt ihn mit 03H ab.

Status:

(A) Wenn

(a) der erste Textcharakter ein Zeichen zwischen A-Ü oder 0-9 ist oder

(b) das erste Zeichen ein ´*´ und das nächste zwischen A-Ü oder 0-9 ist,
dann wird der Dateiname von dem in HL angegebenen Bereich in den in DE bestimmten übertragen, bis ein Zeichen gefunden wird, welches ungleich ´/´, ´.´, ´:´, A-Ü oder 0-9 ist oder bis 32 Byte übertragen wurden.

(B) Ist der Text kürzer als 32 Byte, so wird hinter das letzte in den FCB-Bereich übertragene Byte des Dateinamens ein 03H als Endzeichen eingefügt, ZERO-Status gesetzt und zurückgesprungen.

(C) ist der angegebene Text länger als 31 Byte, wird er als falsch angesehen bei folgenden Voraussetzungen:

(a) wenn das erste Zeichen nicht zulässig ist, oder das erste Zeichen ein ´*´ ist, aber das zweite unzulässig ist, wird ein Return mit NZ-Status durchgeführt.

(b) wenn das Endzeichen gleich 03H oder 0DH ist, so zeigt HL beim Aussprung auf dieses Byte, im andern Fall zeigt HL auf das dem Fehlerzeichen folgende nächste Byte.

Achtung: Das Betriebssystem überprüft hier nicht die Korrektheit des Dateinamens. Es überläßt dies den Routinen 4420H sowie 4424H.

Eröffnen eines FCB für eine neue oder existierende Datei

```
**************************************************
*                                                *
* I:   AF: wird zerstört                         *
*      BC: B enthält die logische Satzlänge      *
*      DE: zeigt auf FCB mit Dateiname           *
*      HL: I/O Puffer für FCB+03                 *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlercode  *
*          C=neue Datei, NC=alte Datei           *
*      BC: B wie oben, C unverändert             *
*      DE: offener FCB für I/O auf Datei         *
*      HL: Puffer wie im FCB angegeben           *
*                                                *
**************************************************
```

Diese Routine eröffnet einen FCB für eine neue oder bereits existierende Datei.

Nur Register AF wird von der Routine geändert. Alle anderen Register werden beim Aufruf gesichert und beim Verlassen wieder zurückgesetzt.

Tritt während der Ausführung der Routine kein Fehler auf, so wird beim Rücksprung das ZERO-Flag gesetzt. Andernfalls wird NZ gesetzt und A enthält dann den Fehlerkode. Bei ZERO gibt das CARRY-Flag Auskunft darüber, ob die Datei schon existierte.

1 Einsprungbedingungen

Folgende Bedingungen gelten für den Einsprung:

a) Register DE zeigt auf den zu öffnenden FCB, der zu diesem Zeitpunkt einen vollständigen Dateinamen enthalten soll.
b) Register HL zeigt auf einen 256 Byte langen Puffer, der zur sektorweisen Übertragung der Daten dient.
c) Register B enthält die logische Satzlänge (0=256)

2 Ablauf der Routine

2.1 Eröffnungsversuch mit $OPEN

Die $OPEN-Routine wird benutzt, um eine bereits existierende Datei zu erkennen.

Kann die $OPEN-Routine die angegebene Datei erfolgreich eröffnen, so sind keine weiteren Maßnahmen erforderlich und es erfolgt ein Rücksprung bei dem Z und NC gesetzt sind.

2.2 Eröffnen einer neuen Datei

Wird die Datei nicht gefunden, so wird sie angelegt.

2.2.1 Suchen eines freien Platzes

Wird eine Laufwerksnummer mit angegeben, so eröffnet die Routine die Datei, wenn auf der entsprechenden Diskette ein freier FDE ist, egal ob die Diskette voll ist oder nicht.

Ist keine Laufwerksnummer angegeben, so sucht das Betriebssystem, beginnend mit dem in der Systemoption A0 angegebenen Laufwerk nach einem freien FDE. Es wird dann auf dem erstmöglichen nicht-schreibgeschützten Laufwerk mit einem freien FDE die Datei eröffnet.

Wird kein freier FDE gefunden, so kann keine Datei eröffnet werden, und es wird mit einer Fehlermeldung zurückgesprungen.

2.2.2 Eintragung ins Inhaltsverzeichnis

Beim Anlegen einer Datei wird der freie FDE in einen FPDE umgewandelt. Desweiteren wird:

a) Name und Typ eingesetzt
b) ein angegebenes Kennwort codiert
c) von Register B die logische Satzlänge übertragen
d) EOF auf 0 gesetzt
e) die Datei als Nicht-System und Nicht-Unsichtbar definiert.
f) das aktuelle Datum eingetragen

Hinweis:
Die logische Satzlänge wird im Inhaltsverzeichnis nur zur Berechnung der Anzahl der Sätze abgespeichert. Jeder weitere Dateizugriff richtet sich nach der durch Register B übergebenen und im FCB abgelegten logischen Satzlänge.

Zu diesem Zeitpunkt überprüft das Betriebssystem weder ob auf der Diskette noch Platz ist, noch wird für die Datei mehr Platz reserviert.

2.3 Eröffnen des FCB

Nachdem die Datei angelegt wurde, ruft das Betriebssystem erneut die $OPEN-Routine auf, um die Datei zu öffnen. Ist alles erfolgreich abgeschlossen, wird mit Z und C zurückgesprungen.

Eröffnen eines FCB für eine bereits existierende Datei

```
**************************************************
*                                                *
* I:   AF: wird zerstört                         *
*      BC: B enthält die logische Satzlänge      *
*      DE: zeigt auf FCB mit Dateiname           *
*      HL: I/O Puffer für FCB+03                 *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlercode  *
*      BC: B wie oben, C unverändert             *
*      DE: offener FCB für I/O auf Datei         *
*      HL: Puffer wie im FCB angegeben           *
*                                                *
**************************************************
```

Nur Register AF wird durch diese Routine geändert.

1 Einsprungbedingungen

Folgende Bedingungen gelten für den Einsprung:

a) Register DE zeigt auf den zu öffnenden FCB, der zu diesem Zeitpunkt einen vollständigen Dateinamen enthalten soll.
b) Register HL zeigt auf einen 256 Byte langen Puffer, der zur sektorweisen Übertragung der Daten dient.
c) Register B enthält die logische Satzlänge (0=256)

2 Ablauf der Routine

2.1 Auffinden der gesuchten Datei

a) Der Dateiname wird auf Zulässigkeit überprüft. Im Fehlerfall wird die Routine schon hier mit entsprechendem Fehlercode abgebrochen.

b) wenn eine Laufwerksnummer angegeben wurde, so wird nur auf dem angegebenen Laufwerk gesucht, andernfalls beginnt die Suche nach der Datei auf Laufwerk 0.

c) wird die Datei nicht gefunden, so erfolgt ein Rücksprung mit Fehlermeldung.

2.2 Kennworte

a) sind Kennworte aktiviert, so tritt ein Fehler auf, wenn die Datei mit einem Kennwort versehen ist, welches weder mit dem Bearbeitungs- noch mit dem Hauptkennwort übereinstimmt.

b) für den Fall, daß die Kennworte vom System nicht abgefragt werden, daß die Datei kein Kennwort hat oder daß das Hauptkennwort übereinstimmt, wird die Zugriffsstufe auf VOLL gesetzt. Ansonsten wird zur Limitierung die im Inhaltsverzeichnis eingetragene Stufe genommen.

2.3 Veränderung des FCB

Der FCB wird umgeformt und enthält dann nicht mehr den Dateinamen, sondern Informationen über die Datei.
Diese Informationen werden benötigt, um die Zahl der Diskettenzugriffe herabzusetzten.

3 FCB nach $OPEN

Die $OPEN-Routine setzt

FCB+00 080H (bit7, FCB ist offen)
FCB+01 bit7 gesetzt, wenn Register B<>0
bit6 gesetzt, wenn Register B=0
bit5 gesetzt, weil Sektor noch nicht gelesen
bit4 nicht gesetzt, da sonst Schreiben bei $CLOSE
bit3 weil FCB-Länge 32 Bytes
FCB+05 000H relative Byteposition im Sektor
FCB+0C Sektorposition LSB
FCB+0D Sektorposition MSB

Die restlichen der 32 Bytes des FCB werden bestimmt durch

3.1 FCB-Daten gemäß Einsprungbedingungen

FCB+03 I/O Pufferadresse Register L
FCB+04 I/O Pufferadresse Register H
FCB+09 logische Satzlänge Register B

3.2 FCB-Daten aus dem Inhaltsverzeichnis

FCB+01 Bit 0-2 Zugriffsstufe gemäß Kennwort
FCB+02 wie FDE+01 (s. Inhaltsverzeichnis, Dateieintrag)
FCB+06 Laufwerksnummer
FCB+07 Dateieintragszeiger (s. Inhaltsverzeichnis, HIT-Sektor)
Dateiende:
FCB+08 EOF-Byte der Datei
FCB+0C LSB des EOF-Sektors
FCB+0D MSB des EOF-Sektors
FCB+0E Blockzuordnung
FCB+0F Einheitenzuordnung
nachfolgende 3 Doppelbytes (10-15) sind auf FFFF gesetzt oder gemäß den Erweiterungszeigern des Inhaltsverzeichnisses wie in FCB+0E,0F definiert. Die Bytes 16-1F werden von $OPEN auf FF gesetzt und erst später geändert.

diese Routine schließt einen FCB

```
*************************************************
*                                               *
* I:   AF: ./.                                  *
*      BC: ./.                                  *
*      DE: aktueller FCB                        *
*      HL: ./.                                  *
*                                               *
*-----------------------------------------------*
*                                               *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlercode *
*      BC: ./.                                  *
*      DE: FCB mit Dateiname                    *
*      HL: ./.                                  *
*                                               *
*************************************************
```

Diese Routine löst die Verbindung zwischen FCB und Datei.
Nur Register AF wird geändert.

1 Inhalt des FCB - Puffer

Wenn Bit 4 von FCB+01 gleich 1 ist, wird der FCB - Puffer wie bei Call $WRSEC auf die Diskette geschrieben.

2 Eintragung des Dateiendes

Ist das Dateiende im FCB anders als im FPDE, so wird die Eintragung im FPDE auf den aktuellen Stand gebracht.

a) wenn die Datei während der Bearbeitung vergrößert wurde, wird der neu hinzugekommene Bereich eingetragen.

b) wenn die Datei durch die Bearbeitung verkürzt wurde, wird der überschüssige Bereich nur dann freigegeben, wenn bei dieser Datei ADF=J gesetzt wurde.

3 FCB nach $CLOSE

Der FCB enthält nach Beendigung der Routine den im Inhaltsverzeichnis für diese Datei eingetragenen Namen und kann somit bei Wiedereröffnung sofort benutzt werden.

löschen eines Dateieintrages

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: aktueller FCB                         *
*      HL: ./.                                   *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z=fehlerfrei, NZ=Fehler A Fehlercode  *
*      BC: ./.                                   *
*      DE: genullter FCB                         *
*      HL: ./.                                   *
*                                                *
**************************************************
```

Diese Routine löscht einen Dateieintrag in derselben Weise wie das Betriebssystemkommando ´KILL´.
Nur Register AF wird verändert.

1 Einsprungbedingungen

Register DE zeigt auf einen offenen FCB
Der die Datei enthaltende Datenträger darf nicht schreibgeschützt sein.

2 Ablauf der Routine

Gemäß den im FCB enthaltenen Daten über Laufwerk und DEC wird der den Dateieintrag enthaltende Sektor des Inhaltsverzeichnisses geladen.
Ein Diskettenwechsel nach Dateieröffnung führt zur Löschung der dem DEC entsprechenden Datei!

2.1 Änderung des Dateieintrags

Nur Bit 4 des Dateieintrags und aller nachfolgenden Dateierweiterungseinträge wird gelöscht. Alle übrigen Informationen, also auch über Blockzuordnung, bleiben erhalten.

2.2 Such-Index (HIT-Sektor)

Im HIT-Sektor wird das Byte des Dateieintragszeigers (DEC) auf 00 gesetzt. Dieses Byte enthält den Dateinamen in verschlüsselter Form. Ohne korrekten Dateieintragszeiger kann die Datei nicht vom

Betriebssystem gefunden werden.

2.3 Speicherbelegungsplan (GAT-Sektor)

Im GAT-Sektor werden die Bits der bisher der Datei zugeordneten Blöcke wieder gesetzt. Diese Blöcke können nun wieder von anderen Dateien belegt werden.

3 FCB nach $KILL

Der FCB wird mit Nullen aufgefüllt. Alle bisher im FCB enthaltenen Informationen sind damit gelöscht. Bei Abbruch der Routine aufgrund eines Fehlers bleibt der FCB unverändert erhalten.

Laden eines Maschinenprogrammes ohne Ausführung

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: FCB mit Dateiname                     *
*      HL: ./.                                   *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z ohne Fehler, NZ Fehler, A Fehlerc.  *
*      BC: .?.                                   *
*      DE: geöffneter FCB                        *
*      HL: Startadresse des Maschinenprogramms   *
*                                                *
**************************************************
```

Diese Routine hat dieselbe Wirkungsweise wie der LIB-Befehl ´LOAD´.
Die Register AF, BC, HL werden von der Routine geändert.

1 Einsprungbedingungen

Register DE zeigt auf FCB, der den Dateinamen enthält.

2 Ablauf der Routine

Die Datei wird geöffnet.
Beginnend mit Sektor 0 Byte 00 werden die Sektoren der Datei nacheinander in den Betriebssystempuffer eingelesen und von dort in die entsprechenden Speicherstellen übertragen.

2.1 Auswertung der Ladeinformation

Die Übertragung der Daten in die einzelnen Speicherstellen wird durch eine Ladeinformation bewirkt, die aus vier Bytes besteht. Byte
00 Ladecode
Byte 01 Anzahl der Bytes nach Ladeadresse + 2
Byte 02 niederwertige Ladeadresse
Byte 03 höherwertige Ladeadresse

Gebräuchliche Ladecodes sind: 01 Daten
02 Startadresse
05 Kommentar

Ein Kommentar wird überlesen und nicht übertragen.

2.2 Startadresse

Bei Erreichen einer Startadresse wird der Ladevorgang beendet. Die Startadresse wird im Register HL und auf den Adressen 4403H - 4404H (17411 - 17412) abgelegt.

3 FCB nach $LOAD

Der FCB wird durch die Routine nicht geschlossen. Der Positionszeiger beinhaltet den Sektor, der die Startadresse enthält. Die Zugriffsstufe ist auf 5 gesetzt.

Hinweis:
Der Anwender ist selbst verantwortlich für das eventuelle Auftreten von Speicherbelegungskonflikten.

Laden und Starten eines Maschineprogrammes

```
************************************************
*                                              *
* I:   AF: ./.                                 *
*      BC: ./.                                 *
*      DE: FCB mit Dateiname                   *
*      HL: ./.                                 *
*                                              *
*----------------------------------------------*
*                                              *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlerkode *
*      BC: .?.                                 *
*      DE: geöfffneter FCB                     *
*      HL: unverändert                         *
*                                              *
************************************************
```

Diese Routine entspricht in ihrer Vorgehensweise der Ausführung einer CMD-Datei aus der DOS-Ebene.
Register AF sowie BC werden geändert, alle übrigen bleiben erhalten.

1 Einsprungbedingungen

Register DE zeigt auf FCB mit Dateiname.
Eine Typbezeichnung wie z.B. /CMD wird nicht automatisch ergänzt.

2 Fehlerbehandlung

Bei Auftreten eines Fehlers während des Öffnens oder Ladens wird nach $DERROR gesprungen.

3 Ablauf der Routine

Die Routine $LOAD wird ausgeführt.
Register HL wird restauriert.
Die Startadresse wird angesprungen.
Der FCB bleibt wie nach $LOAD.

Lesen eines Disksektors oder Übertragen eines logischen Satzes vom FCB - Puffer in einen Anwenderpuffer

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: offener FCB                           *
*      HL: Adresse des Anwenderpuffers           *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlercode  *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: ./.                                   *
*                                                *
**************************************************
```

Die Routine ändert das Register AF. Alle anderen Register werden beim Einsprung gesichert und beim Rücksprung mit den gesicherten Werten geladen.

1 Einsprungbedingungen

Register DE zeigt auf offenen FCB. Byte 00 und Byte 01 des FCB beeinflussen den weiteren Lesevorgang.
Register HL zeigt auf den Anwenderpuffer, in den die Daten zur weiteren Bearbeitung übertragen werden.

2 Ablauf der Routine

Der Ablauf der Routine wird durch die Statusbits des FCB beeinflußt.

2.1 Sektorzugriff

Wenn Bit 7 von FCB+01 nicht gesetzt ist, wird der mit den beiden Bytes des NEXT Feldes bezeichnete Sektor in den FCB - Puffer eingelesen.
Tritt kein Fehler auf oder ergibt sich der Fehlercode 6 (Sektormarkierung wie Inhaltsverzeichnis) so wird das NEXT Feld um 1 erhöht.
Wurde ein anderer Fehlercode erkannt, so wird das NEXT Feld nicht geändert. Dies ermöglicht dem Benutzer den gleichen Sektor nochmals zu lesen.

2.2 Satzzugriff

Wenn Bit 7 von FCB+01 gesetzt ist, wird ein logischer Satz von der im FCB abgelegten Länge vom FCB - Puffer in den Puffer transferiert, dessen Adresse im HL - Register steht. Nach jedem Byte welches übertragen wurde, wird der Positionszeiger (RBA) erhöht.
Wenn der FCB - Puffer keine zu übertragende Bytes mehr enthält, wird automatisch der nächste Dateisektor in den Puffer eingelesen, wonach die noch fehlenden Bytes übertragen werden. Tritt ein Fehler (einschließlich des Fehler 6) auf, wird der Transfer abgebrochen und der Positionszeiger um die bisher übertragenen Bytes erhöht.

2.3 Diskettenzugriff

Wenn Bit 1 von FCB+00 gesetzt ist, erlaubt die Routine einen direkten Diskettenzugriff, ohne daß dabei ein Dateieintrag im Inhaltsverzeichnis erfolgt.

3 Anmerkung

Bei gesetztem Bit 0 von FCB+00 wird das Lesen des Sektors nicht beeinflußt.
Bit 0 und Bit 1 von FCB+00 müssen vor dem Zugriff durch $BREAD und $BWRITE zurückgesetzt werden.

Schreiben eines Sektors
bzw. Übertragen eines logischen Satzes vom Anwenderpuffer (Arbeitspuffer) in den Disk-I/O-Puffer (FCB-Puffer).

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: aktueller FCB                         *
*      HL: Adresse des Anwenderpuffers           *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlercode  *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: ./.                                   *
*                                                *
**************************************************
```

Diese Systemroutine schreibt einen Sektor auf die Diskette oder transferiert einen logischen Satz von einem Benutzerpuffer in den FCB - Puffer.
Geändert wird das Register AF, die anderen Register werden beim Einsprung gesichert und beim Rücksprung mit den ursprünglichen Werten geladen.

1 Einsprungbedingungen

Register DE zeigt auf einen offenen FCB
Register HL zeigt auf den Anwenderpuffer, in dem die zu übertragenden Daten enthalten sind.

2 Ablauf der Routine

Der Ablauf der Routine richtet sich nach den Statusbits des FCB.

2.1 Sektorzugriff

Wenn Bit 7 von FCB+01 nicht gesetzt ist, wird der im NEXT-Feld angegebene Disksektor mit dem Inhalt des FCB - Puffers beschrieben. Der geschriebene Sektor wird nur überprüft, wenn VERIFY aktiviert ist.
Wenn kein Fehler auftritt und die aktuelle Byte-Position im Sektor gleich 0 ist, wird die Sektor-Position um 1 (also 256 Bytes) erhöht. Auf jeden Fall wird, wenn der Positionszeiger erhöht wurde und dabei über das Dateiende hinauskommt, oder wenn Bit 6 von FCB+01 nicht gesetzt ist, das Dateiende gleich dem Positionszeiger (letzter Sektor = NEXT-FELD, EOF-Byte=aktuelles Byte im Sektor) gesetzt.

Falls ein Fehler auftritt, wird der Positionszeiger nicht erhöht, sodaß der Benutzer den Sektor nochmals schreiben kann.

2.2 Satzzugriff

Wenn Bit 7 von Byte 01 des FCB gesetzt ist, wird ein logischer Satz von der im FCB eingetragenen Länge vom Benutzerpuffer, dessen Beginn in HL übergeben wurde, in den FCB - Puffer übertragen. Nach jedem übertragenen Byte wird der Positionszeiger erhöht. Wird dabei das Dateiende erreicht oder wenn Bit 6 von FCB+01 nicht gesetzt ist, so wird das Dateiende auf die durch NEXT-Feld und NEXT-Byte gegebene Sektorposition gesetzt (wie oben).
Wenn der FCB - Puffer keine weiteren Bytes mehr aufnehmen kann, so wird er in den zugehörigen Disksektor geschrieben, danach durch Lesen überprüft und der Transfer wird anschließend fortgeführt. Sobald ein Fehler auftritt wird der Transfer unterbrochen, wobei der Positionszeiger um die bis dahin übertragenen Bytes erhöht wird.

2.3 Diskettenzugriff

Wenn Bit 1 von FCB+00 gesetzt ist, erfolgt eine Ein/Ausgabe direkt auf der Diskette ohne Dateieintrag.

3 Adressmarke

Wenn Bit 0 von FCB+00 gleich 1 ist, so entspricht die Adressmarke des zu schreibenden Sektors der Adressmarke eines Inhaltsverzeichnis-Sektors (geschützter Sektor, Fehlercode 6).

3.1 Verify

Wenn nach dem Schreiben auf einen geschützten Sektor ein Verify durchgeführt wird, so wird zwar der Fehlercode 6 erzeugt, dem Benutzer jedoch nicht angezeigt.

Schreiben eines Sektors mit Überprüfung des Sektors
bzw. Übertragen eines logischen Satzes vom Anwenderpuffer
(Arbeitspuffer) in den Disk-I/O-Puffer (FCB-Puffer).

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: aktueller FCB                         *
*      HL: Adresse des Anwenderpuffers           *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlercode  *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: ./.                                   *
*                                                *
**************************************************
```

Diese Systemroutine schreibt einen Sektor auf die Diskette oder transferiert einen logischen Satz von einem Benutzerpuffer in den FCB - Puffer.
Geändert wird das Register AF, die anderen Register werden beim Einsprung gesichert und beim Rücksprung mit den ursprünglichen Werten geladen.

1 Einsprungbedingungen

Register DE zeigt auf einen offenen FCB
Register HL zeigt auf den Anwenderpuffer, in dem die zu übertragenden Daten enthalten sind.

2 Ablauf der Routine

Der Ablauf der Routine richtet sich nach den Statusbits des FCB.

2.1 Sektorzugriff

Wenn Bit 7 von FCB+01 nicht gesetzt ist, wird der im NEXT-Feld angegebene Disksektor mit dem Inhalt des FCB - Puffers beschrieben. Der geschriebene Sektor wird auch dann überprüft, wenn VERIFY nicht aktiviert ist.
Wenn kein Fehler auftritt und die aktuelle Byte-Position im Sektor gleich 0 ist, wird die Sektor-Position um 1 (also 256 Bytes) erhöht. Auf jeden Fall wird, wenn der Positionszeiger erhöht wurde und dabei über das Dateiende hinauskommt, oder wenn Bit 6 von FCB+01 nicht gesetzt ist, das Dateiende gleich dem Positionszeiger (letzter Sektor = NEXT-FELD, EOF-Byte=aktuelles Byte im Sektor) gesetzt.

Falls ein Fehler auftritt, wird der Positionszeiger nicht erhöht, sodaß der Benutzer den Sektor nochmals schreiben kann.

2.2 Satzzugriff

Wenn Bit 7 von Byte 01 des FCB gesetzt ist, wird ein logischer Satz von der im FCB eingetragenen Länge vom Benutzerpuffer, dessen Beginn in HL übergeben wurde, in den FCB - Puffer übertragen. Nach jedem übertragenen Byte wird der Positionszeiger erhöht. Wird dabei das Dateiende erreicht oder wenn Bit 6 von FCB+01 nicht gesetzt ist, so wird das Dateiende auf die durch NEXT-Feld und NEXT-Byte gegebene Sektorposition gesetzt (wie oben).
Wenn der FCB - Puffer keine weiteren Bytes mehr aufnehmen kann, so wird er in den zugehörigen Disksektor geschrieben, danach durch Lesen überprüft und der Transfer wird anschließend fortgeführt. Sobald ein Fehler auftritt wird der Transfer unterbrochen, wobei der Positionszeiger um die bis dahin übertragenen Bytes erhöht wird.

2.3 Diskettenzugriff

Wenn Bit 1 von FCB+00 gesetzt ist, erfolgt eine Ein/Ausgabe direkt auf der Diskette ohne Dateieintrag.

3 Adressmarke

Wenn Bit 0 von FCB+00 gleich 1 ist, so entspricht die Adressmarke des zu schreibenden Sektors der Adressmarke eines Inhaltsverzeichnis-Sektors (geschützter Sektor, Fehlercode 6).

3.1 Verify

Wenn nach dem Schreiben auf einen geschützten Sektor ein Verify durchgeführt wird, so wird zwar der Fehlercode 6 erzeugt, dem Benutzer jedoch nicht angezeigt.

Positioniert FCB auf Dateianfang

```
*************************************************
*                                               *
* I:   AF: ./.                                  *
*      BC: ./.                                  *
*      DE: aktueller FCB                        *
*      HL: ./.                                  *
*                                               *
*-----------------------------------------------*
*                                               *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlercode *
*      BC: ./.                                  *
*      DE: ./.                                  *
*      HL: ./.                                  *
*                                               *
*************************************************
```

Diese Systemroutine dient dazu, den Positionszeiger der Datei an den Anfang der Datei zu setzen.
Nur Register AF wird verändert. Alle anderen Register werden beim Einsprung gesichert und beim Rücksprung mit den gesicherten Daten geladen.

1 Einsprungbedingungen

Register DE zeigt auf offenen FCB

2 Ablauf der Routine

Wenn Bit 4 von FCB+01 gesetzt ist, so wird der im Puffer enthaltene Sektor mit $WRSEC auf Diskette übertragen.
Danach wird der Positionszeiger auf Null gesetzt.

3 FCB nach $POSO

Bit 5 von FCB+01 wird gesetzt, da der Sektor noch nicht gelesen wurde.
Bit 4 von FCB+01 ist rückgesetzt, um ein automatisches Schreiben zu verhindern.
Der Positionszeiger bestehend aus FCB+05, FCB+0A und FCB+0B ist auf NULL gesetzt.
Weitere Änderungen werden im FCB nicht vorgenommen.

Positioniert FCB auf Satznummer aus Register BC

```
************************************************
*                                              *
* I:   AF: ./.                                 *
*      BC: Relative Satznummer                 *
*      DE: aktueller FCB                       *
*      HL: ./.                                 *
*                                              *
*----------------------------------------------*
*                                              *
* O:   AF: Z=Fehlerfrei, NZ=Fehler, A=Fehlerkode*
*      BC: ./.                                 *
*      DE: ./.                                 *
*      HL: ./.                                 *
*                                              *
************************************************
```

Diese Routine ändert das Register AF. Alle anderen Register werden beim Einsprung gesichert und beim Rücksprung mit den gesicherten Werten geladen.

1 Einsprungbedingungen

Register DE zeigt auf offenen FCB
Register BC enthält neue Satznummer

2 Ablauf der Routine

Die Relative Byte Adresse der in Register BC angegebenen logischen Satznummer (0=erster Satz) wird für den neuen Positionszeiger mit Hilfe der logischen Satzlänge errechnet und im FCB eingetragen.

3 Pufferinhalt

Wenn die Sektornummer verändert wird, und wenn Bit 4 von FCB+01 gesetzt ist, wird der alte FCB-Pufferinhalt geschrieben und Bit 4 wird zurückgesetzt.
Bit 5 von FCB+01 wird gesetzt; es wird damit angezeigt, daß der neue Sektor noch nicht in den Puffer gelesen wurde.

Bei unveränderter Sektornummer ändern sich Bit 4 und Bit 5 von FCB+01 nicht.

Positioniert FCB auf vorhergehenden Satz

```
************************************************
*                                              *
* I:   AF: ./.                                 *
*      BC: ./.                                 *
*      DE: aktueller FCB                       *
*      HL: ./.                                 *
*                                              *
*----------------------------------------------*
*                                              *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlercode *
*      BC: ./.                                 *
*      DE: ./.                                 *
*      HL: ./.                                 *
*                                              *
************************************************
```

Diese Routine ändert das Register AF. Alle anderen Register werden beim Einsprung gesichert und beim Rücksprung mit den gesicherten Werten geladen.

1 Einsprungbedingungen

Register DE zeigt auf offenen FCB

2 Ablauf der Routine

Die Relative Byte Adresse für den neuen Positionszeiger wird mit Hilfe der logischen Satzlänge errechnet und im FCB eingetragen.

3 Pufferinhalt

Wenn die Sektornummer verändert wird, und wenn Bit 4 von FCB+01 gesetzt ist, wird der alte FCB-Pufferinhalt geschrieben und Bit 4 wird zurückgesetzt.
Bit 5 von FCB+01 wird gesetzt; es wird damit angezeigt, daß der neue Sektor noch nicht in den Puffer gelesen wurde.

Bei unveränderter Sektornummer ändern sich Bit 4 und Bit 5 von FCB+01 nicht.

Hinweis:
Die Routine $RDSEC erhöht automatisch den Positionszeiger.

Positioniert FCB auf Dateiende

```
*************************************************
*                                               *
* I:   AF: ./.                                  *
*      BC: ./.                                  *
*      DE: aktueller FCB                        *
*      HL: ./.                                  *
*                                               *
*-----------------------------------------------*
*                                               *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlercode *
*      BC: ./.                                  *
*      DE: ./.                                  *
*      HL: ./.                                  *
*                                               *
*************************************************
```

Diese Routine ändert das Register AF. Alle anderen Register werden beim Einsprung gesichert und beim Rücksprung mit den gesicherten Werten geladen.

1 Einsprungbedingungen

Register DE zeigt auf offenen FCB

2 Ablauf der Routine

Die Relative Byte Adresse für den neuen Positionszeiger wird vom Dateiende übernommen und im FCB eingetragen.

3 Pufferinhalt

Wenn die Sektornummer verändert wird, und wenn Bit 4 von FCB+01 gesetzt ist, wird der alte FCB-Pufferinhalt geschrieben und Bit 4 wird zurückgesetzt.
Bit 5 von FCB+01 wird gesetzt; es wird damit angezeigt, daß der neue Sektor noch nicht in den Puffer gelesen wurde.

Bei unveränderter Sektornummer ändern sich Bit 4 und Bit 5 von FCB+01 nicht.

Reserviert den für die Datei notwendigen Platz auf der Diskette

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: aktueller FCB                         *
*      HL: ./.                                   *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z=Fehlerfrei, NZ=Fehler A=Fehlercode  *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: ./.                                   *
*                                                *
**************************************************
```

Diese Routine erlaubt die Zuordnung von Diskettenplatz bei Beginn der Bearbeitung.
Nur Register AF wird verändert. Alle übrigen Register werden beim Einsprung gesichert und beim Rücksprung zurückgesetzt.

1 Einsprungbedingungen

Register DE zeigt auf offenen FCB.

2 Ablauf der Routine

Wenn der in den beiden höchstwertigen Bytes des Positionszeigers angegebene Sektor noch nicht der Datei zugewiesen ist, so werden entsprechend viele Einheiten der Diskette für die Datei reserviert. Die Einheiten werden im Speicherbelegungsplan gesperrt und im Dateieintrag eingetragen.

3 Anwendung

Für den Fall, daß die Dateigröße bekannt ist, ist es vorteilhaft, schon vor dem Schreibbeginn den benötigten Platz zu reservieren (wie z.B. beim Kopieren). Andernfalls muß bei jeder Dateierweiterung erneut auf das Inhaltsverzeichnis zugegriffen werden.

Positioniert FCB auf Relative Byte Adresse

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: C=Position im Sektor                  *
*      DE: aktueller FCB                         *
*      HL: Sektornummer                          *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlercode  *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: ./.                                   *
*                                                *
**************************************************
```

Diese Routine ändert das Register AF. Alle anderen Register werden beim Einsprung gesichert und beim Rücksprung mit den gesicherten Werten geladen.

1 Einsprungbedingungen

Register DE zeigt auf offenen FCB
Register HL enthält die Sektoradresse der RBA
Register C enthält die Byteadresse der RBA

2 Ablauf der Routine

Die Relative Byte Adresse für den neuen Positionszeiger wird im FCB eingetragen.
FCB+05 erhält den Wert aus Register C.
FCB+0A erhält den Wert aus Register L.
FCB+0B erhält den Wert aus Register H.

3 Pufferinhalt

Wenn die Sektornummer verändert wird, und wenn Bit 4 von FCB+01 gesetzt ist, wird der alte FCB-Pufferinhalt geschrieben und Bit 4 wird zurückgesetzt.
Bit 5 von FCB+01 wird gesetzt; es wird damit angezeigt, daß der neue Sektor noch nicht in den Puffer gelesen wurde.

Bei unveränderter Sektornummer ändern sich Bit 4 und Bit 5 von FCB+01 nicht.

Eintragen des Dateiendes ins Inhaltsverzeichnis

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: aktueller FCB                         *
*      HL: ./.                                   *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z=Fehlerfrei, NZ=Fehler A=Fehlercode  *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: ./.                                   *
*                                                *
**************************************************
```

Diese Routine erlaubt die Änderung des Dateiendezeigers (EOF) im Inhaltsverzeichnis.
Nur Register AF wird geändert. Alle anderen Register werden beim Einsprung gesichert und beim Rücksprung zurückgesetzt.

1 Einsprungbedingungen

Register DE zeigt auf offenen FCB.

2 Ablauf der Routine

Das im Inhaltsverzeichnis eingetragenen Dateiende wird gelesen und mit dem im FCB eingetragenen Dateiende verglichen.
Sind beide Eintragungen identisch, so endet die Routine.
Bei Abweichung wird die Eintragung vom FCB ins Inhaltsverzeichnis übertragen.

Erkennt und überliest Trennzeichen

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: zeigt auf Text                        *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z=Textende, C=Zeiger auf ´,´; A=34H   *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: zeigt auf nachfolgendes Zeichen       *
*                                                *
**************************************************
```

Diese Routine erlaubt das Lesen eines Textes bis zum nächsten Parameter.

1 Einsprungbedingungen

Register HL zeigt auf das erste Zeichen hinter einem Parameter oder Dateinamen.

2 Ablauf der Routine

Das durch Register HL adressierte Zeichen wird mit ODH, 2CH und 20H verglichen.
Wird keines dieser Zeichen vorgefunden, so verhält sich die Routine wie bei Leerzeichen beschrieben.

2.1 Leerzeichen

Mehrere aufeinanderfolgende Leerzeichen werden ignoriert und HL zeigt auf das den Leerzeichen folgende Zeichen.
CARRY ist gesetzt, ZERO ist rückgesetzt. Register A enthält Fehlercode 34H.

2.2 Komma

Die Routine erkennt das ´,´ als Trennzeichen, wenn es sofort angetroffen wird.
CARRY ist dann rückgesetzt, ZERO ist rückgesetzt. Register A

enthält Fehlercode 34H. Register HL zeigt auf das nachfolgende Zeichen.

2.3 Zeilenende

Ein 'ODH' am Textende wird durch das gesetzte ZERO - Flag angezeigt. Register A enthält den Wert ODH. Register HL wird nicht erhöht.

3 Anwendungshinweis

Bei Eingabe eines Befehls können hinter dem Befehl noch Parameter folgen. Die Routine $DELIM erkennt, ob Parameter vorhanden sind, und positioniert Register HL auf das erste signifikante Zeichen.

Vergleicht Füllwort/Parameter

```
*************************************************
*                                               *
* I:   AF: ./.                                  *
*      BC: zeigt auf Füllworttafel              *
*      DE: ./.                                  *
*      HL: zeigt auf Text                       *
*                                               *
*-----------------------------------------------*
*                                               *
* O:   AF: Z=Füllwort korrekt, NZ falsch        *
*      BC: aktueller Stand in Tafel             *
*      DE: ./.                                  *
*      HL: zeigt auf Text hinter Füllwort (Z)   *
*                          vor Füllwort (NZ)    *
*************************************************
```

Diese Routine erlaubt den Vergleich eines Füllwortes oder eines Parameters in einem Textstring mit den in einer Tafel abgelegten Parametern oder Füllworten.

1 Einsprungbedingungen

Register BC zeigt auf die Parametertafel. Hinter jedem Parameter muß OOH als Abschlußkennzeichen gesetzt sein. Register HL zeigt auf Eingabetext.

2 Ablauf der Routine

Ein durch Register HL adressierter Text wird byteweise mit dem durch Register BC gegebenen Speicherinhalt verglichen. Der Vergleich endet bei Abweichung mit NZ.
Bei Übereinstimmung werden die Doppelregister BC und HL jeweils um eins erhöht.
Zeigt Register BC auf eine Speicherstelle mit dem Inhalt OOH so wird der Vergleich mit gesetztem ZERO-Flag beendet.

3 Anwendungshinweis

Nachfolgende Bytes sind vom Anwenderprogramm auswertbar, da bei zutreffendem Vergleich BC hinter das Abschlußkennzeichen zeigt. Bei nicht zutreffendem Vergleich hat das Anwenderprogramm selber dafür Sorge zu tragen, daß BC wieder auf den nächstfolgenden Parameter positioniert wird.

HL wird bei nicht zutreffendem Vergleich wieder auf die Position zurückgesetzt, die vor Aufruf der Routine gültig war. Bei zutreffendem Vergleich zeigt HL hinter das gelesene Füllwort.

Selektiert das gewünschte Laufwerk

```
************************************************
*                                              *
* I:   AF: A=Laufwerksnummer                   *
*      BC: ./.                                 *
*      DE: ./.                                 *
*      HL: ./.                                 *
*                                              *
*----------------------------------------------*
*                                              *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlercode *
*      BC: ./.                                 *
*      DE: ./.                                 *
*      HL: ./.                                 *
*                                              *
************************************************
```

Diese Routine definiert das genannte Laufwerk als aktuelles Laufwerk.
Nur Register AF wird verändert. Alle anderen Register werden beim Einsprung gesichert und beim Rücksprung mit den gesicherten Werten geladen.

1 Einsprungbedingungen

Beim Einsprung enthält Register A die Nummer des zu selektierenden Laufwerkes.

2 Ablauf der Routine

Die Routine erledigt alle notwendigen Vorbereitungen, um im weiteren Verlauf auf das genannte Laufwerk zugreifen zu können.

2.1 Existenzprüfung

Ist das Laufwerk gemäß den Systemoptionen nicht definiert, so wird Fehlercode 20H ausgegeben.

2.2 Parametertafel

Die Spezifikation des gewählten Laufwerks wird in die Parametertafel für das aktuelle Laufwerk übertragen.

2.3 Floppydisk-Controller

Der Floppydisk-Controller wird in den gewünschten Modus versetzt. Die Routine kann also genutzt werden, um den Controller von Double-Density auf Single-Density, von 8-Zoll auf 5 1/4 - Zoll etc. umzuschalten.

2.4 Laufwerk

Abschließend wird das Laufwerk angewählt und wenn nötig neu gestartet.

3 Hinweis

Es wird nicht geprüft, ob die Datenausgabeeinheit existiert.

Prüft nach, ob im angegebenen Laufwerk eine Diskette ist

```
************************************************
*                                              *
* I:   AF: A=Laufwerksnummer                   *
*      BC: ./.                                 *
*      DE: ./.                                 *
*      HL: ./.                                 *
*                                              *
*----------------------------------------------*
*                                              *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlercode *
*      BC: ./.                                 *
*      DE: ./.                                 *
*      HL: ./.                                 *
*                                              *
************************************************
```

Diese Routine überprüft, ob sich eine zulässige Diskette im angegebenen Laufwerk befindet.
Nur Register AF wird verändert. Alle anderen Register werden gesichert und bei Rückkehr mit den alten Werten geladen.

1 Einsprungbedingungen

Register A enthält die Laufwerksnummer.

2 Ablauf der Routine

Die Routine durchläuft $DRVSEL und testet dann die im Laufwerk eingelegte Diskette.

2.1 Lesbarkeit

Die Routine testet nicht die Lesbarkeit der Diskette. Es wird anhand des Indexlochs festgestellt, ob sich die Diskette im Laufwerk mit ausreichender Geschwindigkeit dreht.

3 Fehlermeldung

Wenn keine zulässige Diskette (diese kann auch noch unformatiert sein) vorgefunden wird, so wird der Fehlerkode 08H im AKKU abgelegt.

Erweitert den Befehlsvorrat um eine geladene Routine

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: Zeiger auf Benutzerroutine            *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlercode  *
*      BC: .?.                                   *
*      DE: .?.                                   *
*      HL: .?.                                   *
*                                                *
**************************************************
```

Diese Routine erlaubt den LIB-Befehlssatz zu erweitern.

1 Einsprungbedingungen

Register HL zeigt auf eine Benutzerroutine im Speicher.
Dabei müssen die ersten zwölf Bytes dieser Routine wie folgt definiert sein:
a) die ersten vier Bytes sind für das Betriebssystem reserviert.
b) die nächsten acht Bytes enthalten den Namen der Routine, wobei fehlende Zeichen nach rechts mit Blanks aufgefüllt werden.

2 Ablauf der Routine

Die bereits aktivierten *name-Routinen werden auf die Existenz des Namens der Routine überprüft.

2.1 Fehlererkennung

Ist eine Routine mit demselben Namen schon vorhanden, so wird der Fehlerkode 53H ausgegeben und das NZ - Flag gesetzt. Andernfalls wird die Routine eingebunden und das Z - Flag gesetzt.

3 Aufruf der Routine

Die Routine kann jetzt mit *name aufgerufen werden.

3.1 Parameterübergabe

Immer wenn ein Kommando in der Form *name oder *name,parameter auszuführen ist, wird nach der betreffenden Routine gesucht und der HL - Zeiger auf die Parameter (falls vorhanden) gesetzt und zum dreizehnten Byte der Routine gesprungen.

3.2 Verlassen der Routine

Wenn die Routine verlassen wird, so sollte das über $RET, $DERROR oder $EXIT geschehen. Es können alle Register benutzt werden. Zusätzlich können in den zwei Bytes 4403H - 4404H Integerwerte geholt oder übergeben werden.

3.3 Fehlermeldung

Existiert der Name der Routine nicht, so wird der Fehlerkode 24H ausgegeben, das NZ - Flag gesetzt und zurückgesprungen.

Löscht Befehl aus erweiterter Befehlsliste

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: Zeiger auf Benutzerroutine            *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: Z=fehlerfrei, NZ=Fehler A=Fehlercode  *
*      BC: .?.                                   *
*      DE: .?.                                   *
*      HL: .?.                                   *
*                                                *
**************************************************
```

Ein mit $NAMENQ eingefügter Befehl kann mit dieser Routine wieder gelöscht werden. Der durch die Routine belegte Speicherplatz darf erst danach anderweitig genutzt werden.

1 Einsprungbedingungen

Register HL zeigt auf eine Benutzerroutine im Speicher.
Dabei müssen die ersten zwölf Bytes dieser Routine wie folgt definiert sein:

a) die ersten vier Bytes sind für das Betriebssystem reserviert.
b) die nächsten acht Bytes enthalten den Namen der Routine, wobei fehlende Zeichen nach rechts mit Blanks aufgefüllt werden.

2 Ablauf der Routine

Ist eine Routine mit diesem Namen nicht vorhanden, so wird der Fehlerkode 24H ausgegeben und das NZ - Flag gesetzt. Andernfalls wird die Routine gelöscht.

Wird die Routine jetzt nochmals mit *NAME aufgerufen, wird der Fehlerkode 42H ausgegeben.

$PRINT 4467H

Text auf dem Monitor ausgeben

```
*************************************************
*                                               *
* I:   AF: ./.                                  *
*      BC: ./.                                  *
*      DE: ./.                                  *
*      HL: Zeiger auf Text                      *
*                                               *
*-----------------------------------------------*
*                                               *
* O:   AF: A=03H  oder A=0DH                    *
*      BC: ./.                                  *
*      DE: ./.                                  *
*      HL: unverändert!                         *
*                                               *
*************************************************
```

Die Routine erlaubt die Ausgabe eines Textes auf den Monitor wahlweise mit oder ohne abschliessenden Zeilenvorschub.

Nur Register AF wird geändert. Alle anderen Register werden beim Einsprung gesichert und beim Rücksprung mit den gesicherten Werten geladen.

1 Einsprungbedingungen

Register HL zeigt auf einen mit 0DH oder 03H abzuschliessenden Text.

2 Ablauf der Routine

Der durch HL adressierte Text wird einschließlich 0DH aber ohne 03H auf dem Monitor ausgegeben.
Durch die Ausgabe von 0DH wird ein Zeilenvorschub bewirkt.

Der Akku enthält das Abschlusszeichen (03H oder 0DH).

Text auf dem Drucker ausgeben

```
************************************************
*                                              *
* I:   AF: ./.                                 *
*      BC: ./.                                 *
*      DE: ./.                                 *
*      HL: Zeiger auf Text                     *
*                                              *
*----------------------------------------------*
*                                              *
* O:   AF: A=03H  oder A=0DH                   *
*      BC: ./.                                 *
*      DE: ./.                                 *
*      HL: unverändert!                        *
*                                              *
************************************************
```

Die Routine erlaubt die Ausgabe eines Textes auf den Drucker wahlweise mit oder ohne abschliessenden Zeilenvorschub.

Nur Register AF wird geändert. Alle anderen Register werden beim Einsprung gesichert und beim Rücksprung mit den gesicherten Werten geladen.

1 Einsprungbedingungen

Register HL zeigt auf einen mit 0DH oder 03H abzuschliessenden Text.

2 Ablauf der Routine

Der durch HL adressierte Text wird einschließlich 0DH aber ohne 03H auf dem Drucker ausgegeben.
Durch die Ausgabe von 0DH wird ein Zeilenvorschub bewirkt.

Der Akku enthält das Abschlusszeichen (03H oder 0DH).

Uhrzeit in das Format hh:mm:ss umwandeln

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: Formatfeld (8 Bytes, für hh:mm:ss)    *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: .?.                                   *
*      BC: B=':'  C=00                           *
*      DE: 4040H (vor Uhrzeit)                   *
*      HL: zeigt hinter Formatfeld               *
*                                                *
**************************************************
```

Die aktuelle Uhrzeit, die in den Speicherstellen 4041H - 4043H abgelegt ist, wird in die Form hh:mm:ss konvertiert und in den acht Bytes auf die HL zeigt abgelegt.

Die Register AF, BC, DE, und HL sind danach geändert.

1 Einsprungbedingungen

Register HL zeigt auf ein 8 Byte langes Formatfeld.

2 Ablauf der Routine

Die aktuelle Uhrzeit wird von ihrer systeminternen 3 Byte Darstellung in "Klarschrift" umgewandelt.

Wegen der Reihenfolge der internen Darstellung zeigt Register DE anschliessend vor diese gepackte Darstellung.
Register HL zeigt hinter das Formatfeld.
Register B enthält das Trennzeichen.
Register A enthält das zuletzt übertragene Zeichen.

Beim Aussprung zeigt HL auf das nächste Byte nach diesem Feld.

Wenn der 25 ms - heartbeat kommt, wird (4040H) hochgezählt und wenn dann noch Bit7 von (37E0H) hoch ist, kann die Zeit der RTC nach (4041H-4043H) gelesen werden.
ee h

Datum in das Format tt.mm.jj umwandeln

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: Formatfeld (8 Bytes, für tt.mm.jj)    *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: .?.                                   *
*      BC: B='.'  C=00                           *
*      DE: 4043H (vor Datum)                     *
*      HL: zeigt hinter Formatfeld               *
*                                                *
**************************************************
```

Das aktuelle Datum, das in den Speicherstellen 4044H - 4046H abgelegt ist, wird in die Form tt.mm.jj konvertiert und in den acht Bytes auf die HL zeigt abgelegt.

Die Register AF, BC, DE, und HL sind danach geändert.

1 Einsprungbedingungen

Register HL zeigt auf ein 8 Byte langes Formatfeld.

2 Ablauf der Routine

Das aktuelle Datum wird von seiner systeminternen 3 Byte Darstellung in "Klarschrift" umgewandelt.

Wegen der Reihenfolge der internen Darstellung zeigt Register DE anschliessend vor diese gepackte Darstellung.
Register HL zeigt hinter das Formatfeld.
Register B enthält das Trennzeichen.
Register A enthält das zuletzt übertragene Zeichen.

Beim Aussprung zeigt HL auf das nächste Byte nach diesem Feld.

Ergänzt Typbezeichnung

```
**************************************************
*                                                *
* I:   AF: ./.                                   *
*      BC: ./.                                   *
*      DE: zeigt auf Dateinamen                  *
*      HL: zeigt auf Typ (3 Bytes)               *
*                                                *
*------------------------------------------------*
*                                                *
* O:   AF: siehe unten                           *
*      BC: ./.                                   *
*      DE: ./.                                   *
*      HL: HL-1                                  *
*                                                *
**************************************************
```

Diese Routine ergänzt im Dateinamen den angegebenen Typ, wenn der Dateiname noch keinen Typ enthielt.

Die Register AF und HL werden geändert.

1 Einsprungbedingungen

Register DE zeigt auf den Anfang eines Dateinamens.
Register HL zeigt auf eine 3 Byte lange Typbezeichnung.

2 Ablauf der Routine

Wenn der Dateiname, auf den DE zeigt, noch ohne Typbezeichnung ist, so werden in den Namen das Typ-Trennzeichen '/' und drei Zeichen von der Stelle eingefügt, auf die HL zeigt.

Der resultierende Dateiname kann nicht über 31 Zeichen lang sein.

3 Fehlererkennung

Register A enthält 2FH.
Wenn der Dateiname kein Kennwort, dafür aber eine Laufwerksbezeichnung enthielt, so ist das ZERO-Flag gesetzt.
Wenn der Dateiname schon eine Typbezeichnung enthielt, so ist das SUBTRACT-Flag und das ZERO-Flag gesetzt.
Die Gültigkeit des neuen Dateinamens wird hier nicht überprüft.

multipliziert A*HL=AHL

```
*****************************************************
*                                                   *
* I:   AF: A= erster Multiplikator                  *
*      BC: ./.                                      *
*      DE: ./.                                      *
*      HL: zweiter Multiplikator                    *
*                                                   *
*---------------------------------------------------*
*                                                   *
* O:   AF: A=höchstwertiges Byte des Ergebnis       *
*      BC: ./.                                      *
*      DE: ./.                                      *
*      HL: niederwertige Bytes des Ergebnis         *
*                                                   *
*****************************************************
```

Multipliziert den Inhalt des AKKU mit dem Inhalt von HL.
Das Ergebnis steht in HL, der Übertrag im AKKU.

Dividiert HL/A

```
*************************************************
*                                               *
* I:    AF: A= Divisor                          *
*       BC: ./.                                 *
*       DE: ./.                                 *
*       HL: Divident                            *
*                                               *
*-----------------------------------------------*
*                                               *
* O:    AF: A=Rest                              *
*       BC: B=0  C=Divisor                      *
*       DE: ./.                                 *
*       HL: Ergebnis                            *
*                                               *
*************************************************
```

Dividiert den Inhalt von HL durch den Inhalt des AKKU.
Das Ergebnis steht in HL, der Rest im AKKU.

Erzeugt Hexadezimaldarstellung von DE

```
*************************************************
*                                               *
* I:   AF: n/a                                  *
*      BC: ./.                                  *
*      DE: enthält Integerzahl                  *
*      HL: zeigt auf Platz für Text             *
*                                               *
*-----------------------------------------------*
*                                               *
* O:   AF: geändert                             *
*      BC: n/a                                  *
*      DE: ./.                                  *
*      HL: zeigt hinter Text                    *
*                                               *
*************************************************
```

Diese Routine erlaubt die Umwandlung eines 2-Byte Wertes in hexadezimale Darstellung.

1 Einsprungbedingungen

Register DE enthält den darzustellenden Wert.
Register HL zeigt auf einen freien Platz im Speicher, in den die entsprechende Ascii-Darstellung abgelegt werden kann. Ab HL müssen 4 Speicherstellen zur Zeichenaufnahme bereitstehen.

2 Ablauf der Routine

Zuerst wird der Wert des D-Registers umgewandelt, dann der Wert des E-Registers.

3 Anmerkung

Diese Routine wird auch vom Systemmonitor zur hexadezimalen Darstellung genutzt.

BESCHREIBUNG DER RESTART BEFEHLE

Die Restart (RST) Befehle sind Subroutinen in der Z-80 Maschinensprache (vergleichbar dem CALL). Im Vorteil zum CALL, der zum Aufruf 3 Byte benötigt, braucht der Restart nur 1. Folgende Restart's sind im Betriebssystem als Unterroutinen belegt.

RST 00 - springt auf die Speicherstelle 0, entspricht dem Kaltstart

RST 08 - Syntaxcheck: Die durch HL addressierte Speicherstelle wird mit dem dem RST 08 folgendem Byte verglichen. Bei Gleichheit wird der RST 10 angesprungen, bei Ungleichheit der Syntaxerror.

RST 10 - ladet die durch HL+1 addressierte Speicherstelle in den Akku. Blanks und Linefeeds werden überlesen. Bei Ziffern ist das Carry-Flag gesetzt, bei ":" oder 00 das Zero-Flag.

RST 18 - vergleicht das HL mit dem DE-Register.
HL größer DE Zero-Flag=0; Carry-Flag=0
HL gleich DE Zero-Flag=1; Carry-Flag=0
HL kleiner DE Zero-Flag=0; Carry-Flag=1

RST 20 - testet den Typ des X-Registerinhaltes der in der Speicherstelle 40AF Hex abgelegt ist.
Integer: C=1, P=1, S=1
Single : C=1
Double : P=1
String : Z=1, C=1, P=1
(Z=Zero-, C=Carry-, P=Parity-, S=Sign-Flag)

RST 28 - wird benutzt um ein Systemmodul zu laden. Der Ladecode steht im Akku. Er wird außerdem bei Drücken der Break-Taste angesprungen

RST 30 - wird allgemein benutzt, um ein DEBUG Programm aufzurufen. Die Rückkehr erfolgt bei Beendigung des Debug-Programmes

RST 38 - wenn die CPU in den Interruptmode 1 (IM 1) geschaltet ist, wird alle 25 Millisekunden diese Routine ausgeführt.