Funktionen in verschiedenen Include-Dateien:

ALLOC.H

Funktionen		
brk	calloc	coreleft
farcalloc	farcoreleft	farfree
farheapcheck	farheapcheckfree	farheapchecknode
farheapfillfree	farheapwalk	farmalloc
farrealloc	**free***	heapcheck
heapcheckfree	heapchecknode	heapwalk
malloc*	realloc	sbrk

BIOS.H

Funktionen			
bioscom*	**biosdisk**	biosequip	**bioskey**
biosmemory	biosprint	biostime	

CONIO.H

Funktionen			
cgets	**clreol***	**clrscr***	**cprintf**
cputs	cscanf	delline*	**getch***
getche	getpass	gettext	**gotoxy***
gettextinfo	highvideo	insline*	**kbhit***
lowvideo	movetext	normvideo	putch
puttext	_setcursortype	**textattr***	textmode
textcolor*	**textbackground***	ungetch	wherex
wherey	window		

CTYPE.H

Funktionen				
isalnum	**isalpha**	**isascii**	**iscntrl**	**isdigit**
isgraph	**islower**	**isprint**	**ispunct**	**isspace**
isupper	**isxdigit**	toascii	**tolower**	_tolower
toupper	_toupper			

DIR.H

Funktionen

chdir	findfirst	findnext	fnmerge	fnsplit
getcurdi	getcwd	getdisk	mkdir	mktemp
rmdir	searchpath	setdisk		

DOS.H

Funktionen

absread	abswrite	allocmem	bdos
bdosptr	country	ctrlbrk	**delay***
disable*	dosexterr	dostounix	_emit_
enable*	FP_OFF	FP_SEG	freemem
geninterrupt*	getcbrk	getdate	getdfree
getdta	getfat	getfatd	getftime
getpsp	gettime*	**getvect**	getverify
harderr	hardresume	hardretn	inp
inport*	**inportb***	**int86**	int86x
intdos	intdosx	intr	keep
MK_FP	nosound	outp	**outport***
outportb*	parsfnm	**peek**	**peekb**
poke	**pokeb**	randbrd	randbwr
segread	setblock	setcbrk	setdate
setdta	settime	**setvect***	setverify
sleep*	**sound***	unixtodos	unlink

STRING.H

Funktionen

memccpy	memchr	memcmp	memcpy	memicmp
memmove	memset	movedata	movmem	setmem
stpcpy	**strcat***	**strchr**	**strcmp***	strcmpi
strcpy*	strcspn	strdup	_strerror	strerror
stricmp*	**strlen***	**strlwr**	**strncat***	**strncmp**
strncmpi	**strncpy***	strnicmp	strnset	strpbrk
strrchr	**strrev**	strset	strspn	**strstr**
strtok	**strupr**			

MATH.H

Funktionen

abs*	acos*	asin*	atan*	atan2	atof*
cabs*	ceil	cos*	cosh*	**exp***	fabs*
floor*	fmod	frexp	hypot	labs	ldexp
log*	**log10***	matherr	modf	poly	**pow***
pow10*	sin*	sinh*	sqrt*	tan*	tanh*

STDIO.H (muss immer eingebunden werden !!!)

Funktionen

clearerr	fclose*	fcloseall	
fdopen	feof*	ferror	
fflush*	fgetc	fgetchar	
fgetpos	fgets	fileno	
flushall*	fopen*	fprintf*	
fputc	fputchar	fputs	
fread	freopen	fscanf*	
fseek*	fsetpos	ftell	
fwrite	getc	getchar*	
gets*	getw	perror	
printf*	putc	putchar	
puts*	putw	remove	
rename	rewind*	scanf*	
setbuf	setvbuf	sprintf*	
sscanf*	strerror	_strerror	
tmpfile	tmpnam	ungetc	
unlink	vfprintf	vfscanf	
vprintf	vscanf	vsprintf	vsscanf

TIME.H

Funktionen

asctime	clock*	ctime
difftime*	gmtime	localtime
stime	time*	tzset

"Wichtige" C-Befehle

Dateioperationen:

fopen:
Öffnet ein File und gibt einen File-Handle darauf zurück

```
FILE *fopen(const char *filename, const char *mode);
```

fclose:
Schließt einenDatei

```
int fclose(FILE *stream); Prototyp in stdio.h
```

Das Ergebnis bei fehlerfreier Ausführung ist 0, im Fehlerfall wird EOF zurückgeliefert.

feof:
Liefert != 0, wenn Dateiende erreicht ist .

```
int feof(FILE *stream);
```

rewind:
Positioniert den Dateizeiger auf den Dateianfang.

```
void rewind(FILE *stream);
```

fseek:
Positioniert den Dateizeiger innerhalb einer Datei.

```
int fseek(FILE *stream, long offset, int whence);
```

Prototyp in stdio.h . offset legt die neue Position relativ zu der Position fest, die in whence angegeben wird. Mögliche Werte für whence sind:

```
SEEK_SET (== 0)  Dateianfang
SEEK_CUR (== 1)  Position des Dateizeigers
SEEK_END (== 2)  Dateiende
```

fprintf:
Ausgabe auf einen File-Handle

```
int fprintf(FILE *stream, const char *format[, argument,...]);
```

fwrite:
Ausgabe kompexer Daten (Strukuren) auf einen File-Handle

```
size_t fwrite(const void *ptr, size_t size, size_t n, FILE*stream);
```

*ptr ist ein Zeiger auf die Daten, size ist die Größe der Daten, n ist die Anzahl der zu schreibenden Elemente.

fscanf:
Einlesen aus einem File-Handle

```
int fscanf(FILE *stream, const char *format[, address,...]);
```

fread:
Einlesen kompexer Daten (Strukuren) von einem File-Handle

```
size_t fread(void *ptr, size_t size, size_t n, FILE *stream);
```

*ptr ist ein Zeiger auf die Variable, die die gelesenen Daten enhalten soll, size ist die Größe der Daten, n ist die Anzahl der zu lesendenden Elemente.

Eingabe:

fflush:
Löscht Puffer des Streams. Ohne Parameter wird Tastaturpuffer gelöscht.
int fflush(FILE *stream); Prototyp in stdio.h

getchar:
Liest ein Zeichen von STDIN.
int getchar(void);

gets:
Liest einen String von STDIN bis Zeilenumbruch (Enter) erfolgt.
char *gets(char *string);

scanf:
liest Zeichen von STDIN in Variable ein bis Leerzeichen auftritt. Dann wird weiter eingelesen in die nächste Variable bis wieder ein Leerzeichen auftritt usw.
int scanf(const char *format [, ...]);

getch:
Liest ein Zeichen von der Tastatur, ohne "Echo" auf den Schirm.
int getch(void);
int getche(void);

Ausgabe:

printf:
Ausgabe auf STDOUT
int printf(const char *format [, argument, ...]);

puts:
Ausgabe eines Strings zu stdout.
int puts(const char *s);

clreol:
Löscht im Textmodus alle Zeichen von der momentanen Position des Cursors bis zum Zeilen- bzw. Fensterende.
void clreol(void);

clrscr:
Löscht Bildschirm
void clrscr(void);

gotoxy:
Positioniert den Text-Cursor (relativ zu einem Textfenster).
void gotoxy(int x, int y);

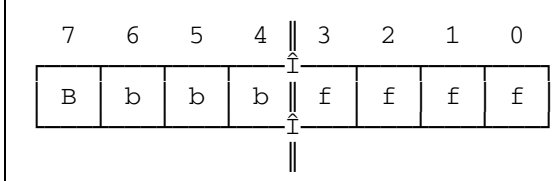
kbhit:
Prüft, ob eine Taste gedrückt wurde.
int kbhit(void);

Ausgabe von Text mit Attributen:

textattr:
Setzt das Attribut für Textausgaben. Ausgabe dann mit cprintf().

newattr

Der Parameter newattr hat folgende Struktur:



Bedeutung der 8-Bits von newattr:
- ffff = 4-Bit Vordergrundfarbe (0 bis 15)
- bbb = 3-Bit Hintergrundfarbe (0 bis 7)
- B = blinken-EIN/AUS-Bit

void textattr(int newattr);

textcolor:
Legt die Vordergrundfarbe für Textausgaben fest. Ausgabe dann mit cprintf().
void textcolor(int newcolor);

textbackground:
Legt die Hintergrundfarbe für Textausgaben fest. Ausgabe dann mit cprintf().
void textbackground(int newcolor);

COLORS (Textmodus)

Konstante	Wert	Hinter-grund?	Vorder-grund?
BLACK	0	Ja	Ja
BLUE	1	Ja	Ja
GREEN	2	Ja	Ja
CYAN	3	Ja	Ja
RED	4	Ja	Ja
MAGENTA	5	Ja	Ja
BROWN	6	Ja	Ja
LIGHTGRAY	7	Ja	Ja
DARKGRAY	8	Nein	Ja
LIGHTBLUE	9	Nein	Ja
LIGHTGREEN	10	Nein	Ja
LIGHTCYAN	11	Nein	Ja
LIGHTRED	12	Nein	Ja
LIGHTMAGENTA	13	Nein	Ja
YELLOW	14	Nein	Ja
WHITE	15	Nein	Ja
BLINK	128	Nein	***

cprintf:
Ausgabe mit vorher festgelegten Attributen
int cprintf (const char *format [, argument, ...]);

Mathematik-Befehle:

abs:
Bildet dem Absolutwert
abs(int x);

pow :
Berechnet die y. Potenz von x, also x^y.
double pow(double x, double y);

sqrt:
Liefert die Quadratwurzel des Arguments.
double sqrt(double x);

floor:
Rundet ab.
double floor(double x);

log:
Berechnet den natürlichen Logarithmus des Arguments.
double log(double x);

log10:
Berechnet den Logarithmus des Arguments zur Basis 10.
double log10(double x);

itoa:
Konvertiert ein Integer in einen String.
char *itoa(int value, char *string, int radix);
radix definiert die Basis, bezüglich der value konvertiert wird. radix muß größer als 1 und kleiner als 37 sein.

max:
Bildet das Maximum
max(a,b) liefert a für a >= b, ansonsten b

min:
Bildet das Minimum
min(a,b) liefert a für a <= b, ansonsten b

Stringbefehle:

strcpy:
Kopiert den Inhalt von src in den String dest.

```
char *strcpy(char *dest, const char *src);
```

strncpy:
Kopiert maxlen Zeichne von src in den String dest. ist src nicht so lange wie maxlen, wird mit „0“-Zeichen aufgefüllt. Ist src länger, wird der String abgeschnitten.

```
char *strncpy(char *dest, const char *src, size_t maxlen);
```

strlen:
Ermittelt die Länge eines Strings. Das abschließende NULL-Zeichen wird nicht mitgezählt.

```
size_t strlen(const char *s);
```

strcat:
Hängt den Inhalt von src an den String dest an.

```
char *strcat(char *dest, const char *src);
```

strncat:
Hängt maximal maxlen Zeichen von src an dest an.

```
char *strncat(char *dest, const char *src, size_t maxlen);
```

strcmp:
Vergleicht zwei Strings miteinander.

```
int strcmp(const char *s1, const char *s2);
```

Ergebnis :

```
< 0 für s1  <  s2
== 0 für s1 == s2
> 0 für s1  >  s2
```

strchr:
Durchsucht String s auf das erste Vorkommen eines angegebenen Zeichens c.
Gibt den Zeiger auf diese Stelle zurück

```
char *strchr(const char *s, int c);
```

strstr:
Durchsucht String s1 nach dem Teilstring s2.
Es wird ein Zeiger auf den Anfang des Teilstrings in s1 zurückgegeben

```
char *strstr(const char *s1, const char *s2);
```

Fortsetzung Stringbefehle:

Überprüfen der Zeichenart:
Alle Befehle liefern bei erfolgreicher Ausführung einen Wert ungleich 0.
Eine erfolgreiche Ausführung ist dabei wie folgt definiert:

int isalpha(int c);	c ist ein Buchstabe (A bis Z oder a bis z)
int isascii(int c);	c ist ein ASCII-Zeich, weil das niederwertige Byte von c im Bereich von 0 bis 127 (0x00--0x7F) liegt
int isdigit(int c);	c ist eine Ziffer (0 bis 9)
int isxdigit(int c);	c ist eine Hex-Ziffer (0 bis 9, A bis F, a bis f)
int islower(int c);	c ist ein Kleinbuchstabe (a bis z)
int isupper(int c);	c ist ein Großbuchstabe (A bis Z)
int ispunct(int c);	c ist ein Interpunktionszeichen (iscntrl oder isspace)
int isprint(int c);	c ist ein druckbares Zeichen (0x20 bis 0x7E)
int isgraph(int c);	c ist ein druckbares Zeichen, Leerzeichen nicht erlaubt
int iscntrl(int c);	c ist das Zeichen DEL oder ein anderes normales Steuerzeichen (0x7F oder 0x00 bis 0x1F)
int isspace(int c);	c ist ein Leerzeichen, Tab, Waagenr cklauf, Zeilenvorschub, vertikal Tab oder Seitenvorschub (0x09 bis 0x0D, 0x20)

Strings konvertieren:

strrev:
Invertiert den String
```
char *strrev(char *s);
```

atof:
Konvertiert einen String in eine Fließkommazahl.
```
double atof(const char *s);
```

atoi:
Konvertiert einen String in ein Integer.
```
int atoi(const char *s);
```

toupper:
Konvertiert einen Kleinbuchstaben in einen Großbuchstaben. Großbuchstaben bleiben unverändet.
```
int toupper(int ch);
```

strupr:
Konvertiert die Kleinbuchstaben im String s in Großbuchstaben.
```
char *strupr(char *s);
```

tolower:
Konvertiert einen Großbuchstaben in einen Kleinbuchstaben. Kleinbuchstaben bleiben unverändet.
```
int tolower(int ch);
```

strlwr:
Konvertiert die Großbuchstaben im String s in Kleinbuchstaben.
```
char *strlwr(char *s);
```

Direkter Speicherzugriff: (z.B. Auf Bildschirmspeicher ab Adr. B800:0000)

peek:
Liest einen Int-Wert (2 Byte) aus der Adresse, die mit den vorzeichenlosen Int-Variablen segment (für Segment-Adresse) und offset (für Offest-Adresse) definiert ist.

```
int peek(unsigned segment, unsigned offset);
```

peekb:
Liest einen Char-Wert (1 Byte) aus der Adresse, die mit den vorzeichenlosen Int-Variablen segment (für Segment-Adresse) und offset (für Offest-Adresse) definiert ist.

```
char peekb(unsigned segment, unsigned offset);
```

poke:
Schreibt den Int-Wert value (2 Byte) in die Adresse, die mit den vorzeichenlosen Int-Variablen segment (für Segment-Adresse) und offset (für Offest-Adresse) definiert ist.

```
void poke(unsigned segment, unsigned offset, int value);
```

pokeb:
Schreibt den Char-Wert value (1 Byte) in die Adresse, die mit den vorzeichenlosen Int-Variablen segment (für Segment-Adresse) und offset (für Offest-Adresse) definiert ist.

```
void pokeb(unsigned segment, unsigned offset, char value);
```

Für Bildschirmzugriff:
Jedes angezeigt Zeichen auf dem Bildschirm benötigt 2 Byte Speicherplatz im Bildschirmspeicher der bei der Adresse B800:0000 beginnt. Daraus folgt die Segmentadresse von B800 und die Offsetadresse beginnt bei 0000

Im 1. Byte steht der ASCII-Wert des angezeigten Zeichens.
Im 2. Byte steht das Attribut des Zeichens. (Attribute siehe Seite 6 in diesem Script)

Direkter Hardwarezugriff (Portzugriff):

inport:
Liest einen 16-Bit-Wert von einer I/O-Adresse.

```
int inport(int portid);
```

inportb:
Liest einen 8-Bit-Wert von einer I/O-Adresse.

```
unsigned char inportb(int portid);
```

outport:
Schreibt einen int (2 Bytes) zu einer I/O-Adresse.

```
void outport(int portid, int value);
```

outportb:
Schreibt ein einzelnes Byte zu einer I/O-Adresse.

```
void outportb(int portid, unsigned char value);
```

Gängige Port-Adressen (portid):
(Seite 61 C-Script mit weiteren Adr.)

Port-Adr (hex)	Baustein
3F8 – 3FF	1. Serielle Schnittstelle
2F8 – 2FF	2. Serielle Schnittstelle
378 – 37A	1. Parallel-Schnittstelle
278 – 27A	2. Parallel-Schnittstelle
1F0 – 1F8	Primärer HD-Controller
170 – 178	Sekundärer HD-Controller
060 – 06F	Tastatur-Controller
070 – 07F	Echtzeituhr
200 – 207	Joystick
3D0 – 3F7	Disketten-Controller

Befehle zur Interupt-Behandlung:

disable:
Unterdrückt alle Hardware-Interrupts außer NMI bis zum nächsten Aufruf von enable.

```
void disable(void);
```

Durch setzen des Bit 7 in Adresse 0x70 kann auch der NMI unterdrückt werden.

enable:
Erlaubt die Unterbrechung des Programms durch Hardware-Interrupts.

```
void enable(void);
```

geninterrupt:
Erzeugt einen Software-Interrupt für den Prozessor.

```
void geninterrupt(int intr_num);
```

getvect:
Liest einen Interrupt-Vektor des Systems.

```
void interrupt (*getvect(int interruptnr))();
```

setvect:
Setzt einen Interrupt-Vektor des Systems.

```
void setvect(int interruptno, void interrupt (*isr) ( ));
```

int86:
Löst einen Software-Interupt aus.

```
int int86(int intno, union REGS *inregs, union REGS *outregs);
```

In intno steht die Nummer des Software-Interupts der ausgelöst werden soll.

Beispiel:

```
union REGS vorher, nachher;      // Zwei Unions definieren, die die Registerinhalte des
                                 // Prozessors vor der Interupt-Ausführung (vorher) und
                                 // nach der Interupt-Ausführung (nachher) enthalten

vorher.h.al = funktionsnummer;   // Im AL-Register der union vorher wird die Funktions-
                                 // nummer innerhalb des Interupts übergeben
                                 // z.B. 9 für Stringsausgabe bei Interupt 21h

.....  // andere benötigte Register entsprechend belegen

int86(0x21, &vorher, &nachher);  // Auslösen des Software-Interupts

.....  // in der union nachher ist nun der Registerinhalt nach der Interupt-Ausführung
       // enthalten
```

```
union REGS {
  struct  WORDREGS  x;
  struct  BYTEREGS  h;
};
```

```
struct  BYTEREGS  {
  unsigned char  al, ah, bl, bh;
  unsigned char  cl, ch, dl, dh;
};
```

```
struct  WORDREGS  {
  unsigned int  ax, bx, cx, dx;
  unsigned int  si, di, cflag, flags;
};
```

Allgemeine Befehle:

exit:
Beendet das laufende Programm.
void exit(int status);

delay:
Wartet die angegebene Anzahl von Millisekunden.
void delay(unsigned milliseconds);

sleep:
Hält die Ausführung des Programms für einen bestimmten Zeitraum (in Sekunden) an.
void sleep(unsigned seconds);

gettime:
Liest die Uhrzeit des Systems.
void gettime(struct time *timep);

```
struct time {
  unsigned char  ti_min;   /* Minuten               */
  unsigned char  ti_hour;  /* Stunden               */
  unsigned char  ti_hund;  /* Hundertstelsekunden   */
  unsigned char  ti_sec;   /* Sekunden              */
};
```

sound:
Aktiviert den eingebauten Lautsprecher des Computers.
void sound(unsigned frequency);

malloc:
Dynamische Belegung von Speicherplatz. malloc liefert einen Zeiger auf den neu belegten Speicherbereich bzw. den Wert NULL, wenn nicht genügend Platz auf dem Heap (Stack) vorhanden ist oder size den Wert 0 hat. size gibt die Größe des zu reservierenden Speichers in Bytes an.
void *malloc(size_t size);

free:
Gibt einen mit malloc oder calloc dynamisch reservierten Speicherbereich wieder frei.
void free(void *block);

bioskey:
Fragt den Tastaturpuffer ab.
int bioskey(int cmd);
Bei cmd=0 wird der Wert des aktuellen Tastendrucks geliefert. Ist das Lower-Byte gleich 0 wurde eine Steuertaste oder Sondertaste gedrückt.
Mit cmd=1 kann man prüfen, ob ein Zeichen anliegt. Wenn kein Zeichen anliegt, wird 0 zurückgegeben.
Mit cmd=2 kann man den Zustand einiger Sondertaste überprüfen. Dabei wird durch die jeweilige Sondertaste ein Bit im Rückgabewert gesetzt:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Taste	Einfg EIN	Caps-Lock EIN	Num-Lock EIN	Srcoll-Lock EIN	Alt gedrückt	STRG gedrückt	Shift Links gedrückt	Shift rechts

Sonstige Befehle:

biosdisk

BIOS-Diskette/Festplattenunterstützung

int biosdisk(int cmd, int drive, int head, int track, int sector, int nsects, void *buffer);

Parameter der Funktion:

cmd = Nummer der auszuführende Operation:

- 0 : erzwingt einen Reset des Laufwerk-Controllers
- 1 : liefert den Status der letzten Laufwerksoperation
- 2 : liest einen oder mehrere Sektoren in den Speicher
- 3 : schreibt einen oder mehrere Sektoren aus dem Speicher
- 4 : vergleicht einen oder mehrere Sektoren
- 5 : formatiert eine Spur
- 6 : formatiert eine Spur und trägt Kennzeichenbits für defekte Sektoren ein
- 7 : formatiert ab einer bestimmten Spur
- 8 : liefert die aktuellen Laufwerksparameter in den erstent 4 Bytes von buffer
- 9 : initialisiert die Parametertabellen für Festplatten
- 10 : long read - liest 512 Bytes, plus 4 zusätzliche Prüfbytes pro Sektor
- 11 : long write - schreibt 512 Bytes, plus 4 zusätzliche Prüfbytes pro Sektor
- 12 : Setzt den Schreib-/Lesekopf auf eine bestimmte Spur
- 13 : alternate reset
- 14 : liest den Sektorpuffer
- 15 : schreibt den Sektorpuffer
- 16 : prüft, ob das angegebene Laufwerk bereit ist
- 17 : setzt den Schreib-/Lesekopf auf Spur 0 zurück
- 18 : Test des Controller-RAM's
- 19 : Test des Laufwerks
- 20 : test der internen Controller-Funktionen

drive = Nummer des Laufwerkes:

- 0x00 : erstes Floppy-Laufwerke (normalerweise A:)
- 0x01 : zweites Floppy-Laufwerk (normalerweise B:)
- 0x80 : erstes Festplattenlaufwerk (normalerweise C:)
- 0x81 : zweites Festplattenlaufwerk
- 0x82 : drittes Festplattenlaufwerk

head = Nummer des Kopfes

track = Nummer des Track

sector = Nummer des Sektors

nsects = Anzahl der zu lesenden Sektoren

*buffer = Zeiger auf die Puffer-Variable. Die Daten werden in einen Puffer mit 512 Bytes pro Sektor geschrieben (und daraus gelesen)

Bei Festplatten wird das physikalische Laufwerk angegeben, nicht die Partition. Falls nötig, muß das Anwenderprogramm die Informationen der Partitions-Tabelle selbst auswerten.

Hinweis:

biosdisk verwendet den Interrupt 0x13 to für direkten Zugriff auf Disketten/Festplatten. biosdisk arbeitet unterhalb der Ebene der Datei- und Sektorenaufteilung. Dadurch können Dateiinhalte und Verzeichnisse zerstört werden.

Rückgabewerte von biosdisk:

Fehlerfreier Ausführung:
High-Byte=0 ; Low-Byte enthält die Anzahl der bearbeiteten Sektoren (gelesen, beschrieben, verglichen, etc.);

Fehlerfall:
Im Fehlerfall enthält das High-Byte einen der folgenden Werte:

Wert	Beschreibung
0x01	ungültiger Befehl
0x02	Addresskennzeichnung nicht gefunden
0x03	Schreibversuch auf schreibgeschützte Diskette
0x04	Sektor nicht gefunden
0x05	Reset konnte nicht ausgeführt werden (Festplatte)
0x06	Diskettenwechsel seit letzter Operation
0x07	Parametertabelle ungültig
0x08	DMA-Fehler
0x09	Überschreitung der 64K-Grenze bei DMA-Operation
0x0A	defekten Sektor gefunden
0x0B	defekte Spur gefunden
0x0C	ungültige Spur
0x10	CRC/ECC-Fehler bei Lesevorgang
0x11	Datenfehler durch CRC/ECC korrigiert (Siehe Hinweis:)
0x20	Controller-Fehler
0x40	fehlerhafte Suchoperation
0x80	Laufwerk nicht bereit
0xAA	Festplatte nicht bereit
0xBB	undefinierbarer Fehler aufgetaucht (nur Festplatte)
0xCC	Schreibfehler aufgetaucht
0xE0	Statusfehler
0xFF	Sense operation failed

Hinweis:
0x11 ist kein Fehler, da die Daten richtig sind. Der Wert wird geliefert, um einer Anwendung die Möglichkeit zu geben diese Tatsache auszuwerten.