Brückmann
Englisch
Gerits

ATARI® ST Intern

EIN DATA BECKER BUCH

Brückmann
Englisch
Gerits

ATARI® ST Intern

EIN DATA BECKER BUCH

ISBN 3-89011-119-X

Copyright (C) 1985 DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Wichtiger Hinweis!

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technische Angaben und Programme in diesem Buch wurden von den Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler sind die Autoren jederzeit dankbar.

Inhaltsverzeichnis

	Einführung	1
1	**Die ICs**	**3**
1.1	Der 68000-Prozessor	3
1.2	Die ATARI-Customer-Chips	18
1.3	Der Floppy-Disk-Controller WD 1772	27
1.4	Der MFP 68901	38
1.5	Die ACIAs 6850	55
1.6	Der programmierbare Soundgenerator YM-2149	64
1.7	Die Belegung des I/O-Bereichs des ATARI ST	73
2	**Die Schnittstellen**	**87**
2.1	Die Tastatur	87
2.1.1	Die Maus	92
2.1.2	Keyboard-Kommandos	95
2.2	Die Videobuchse	108
2.3	Der Centronics-Anschluß	111
2.4	Die RS232-Schnittstelle	114
2.5	Die MIDI-Anschlüsse	117
2.6	Der Modulschacht	120
2.7	Die Floppy-Disk-Schnittstelle	121
2.8	Die DMA-Schnittstelle	
3	**Das Betriebssystem des ATARI ST**	**127**
3.1	Das GEMDOS	128
3.2	Das BIOS	172
3.3	Das XBIOS	185
3.4	Die Grafik	227
3.5	Die Exceptions	265
3.5.1	Die Interruptstruktur des ATARI ST	268
3.6	Der VT52-Emulator	275
3.7	Die Systemvariablen	281
3.8	Der Befehlssatz des 68000-Prozessors	290
3.9	Das BIOS-Listing	307
	Anhang	**463**
	Die Systemfonts	463

Einführung

Wohl selten hat ein Computer vor und während seiner Markteinführung soviel Getöse verursacht wie der ATARI 520ST. Bereits vor der Hannovermesse 1985 begann es, und es ist weit entfernt davon, wieder zu verklingen.
Die Gründe hierfür sind vielfältig. Einmal war das angekündigte Preis/Leistung-Verhältnis als sensationell zu bezeichnen, zum anderen hat es derzeit, wo es das nackte Gerät bereits zu kaufen gibt, Atari ungeheuer schwer, die wirklich über jeden Zweifel erhabene Leistungsfähigkeit auch mit entsprechender Software (wozu derzeit auch noch das Betriebssystem selbst zählt) zu versorgen.
Man könnte meinen, je exklusiver die Leistungsmerkmale, umso einfacher die Programmerstellung. Das genaue Gegenteil ist offensichtlich der Fall. Sicher läßt sich mit dem vorhandenen C-Compiler auch Software von Leuten erstellen, die auf dem 68000 nicht unbedingt zu Hause sind, aber was herauskommt sind dann solche Kuriositäten wie ein BASIC von gut 150K (gegenwärtiger Stand), womit dann glücklich noch ca. 19K für den Anwender frei sind, da ja das Betriebssystem auch im Ram untergebracht ist. So etwas bietet jeder Homecomputer.
Das wird sich sicherlich noch alles ändern, denn die Schwierigkeit besteht darin, daß Atari wohl die hervorragende Hardware geschaffen hat, das Betriebssystem (GEMDOS) jedoch von Digital Research, welcher Umstand an sich noch nicht das Dilemma erklärt. Schwierig ist es deswegen, weil man auf keine 'Konfektionsware' zurückgreifen kann, wie es bei den zahlreichen CP/M- und MSDOS-Varianten der Fall ist, sondern praktisch ein Betriebssystem aus dem Boden gestampft werden muß, welches den hohen Ansprüchen an Leistungsfähigkeit und Benutzerfreundlichkeit genügt.

Nun gut, das sind Dinge, die jetzt mittlerweile so gut wie gelöst sind. Bis zum Jahresende wird es sicher einen Berg guter Anwendungsprogramme geben, aber sie haben uns, wie Sie sicher verstehen werden, die Arbeit an diesem Buch nicht gerade erleichtert. Dank guter Kontakte zu Atari bekamen wir

aber die notwendigen Unterlagen, zwar recht bunt gewürfelt, aber brauchbar, wofür wir an dieser Stelle danken möchten.
Mehr noch aber war die ganz persönliche Auseinandersetzung mit dem Gerät für jeden von uns notwendig, um Ihnen dieses Buch präsentieren zu können.

Nun, nachdem alles 'gelaufen' ist, möchten wir das Gerät nicht mehr missen, denn nach dem Erkunden seiner Eigenheiten wollen wir doch endlich auch ein wenig damit spielen.
Ihnen wünschen wir, daß Sie nach (oder bei) der Lektüre gleich mit dem Spiel (über einen Assembler sollten Sie verfügen) beginnen können.

Düsseldorf, den 30. September 1985

Ihre Autoren

1 DIE ICs DES ATARI ST

1.1 Der Prozessor 68000

Das Herz des gesamten ATARI ST-Systems ist der Mikroprozessor MC68000. Dieser 16-Bit-Prozessor zählt zur absoluten Spitzenklasse. Sowohl Programmierer als auch Hardware-Designer finden in ihm einen gleichermaßen einfach wie komfortabel zu handhabenden Prozessor. Obwohl die Entwicklung des 68000 bereits im Jahre 1977 bei der Firma MOTOROLA begann, und erste Prototypen schon 1979 verfügbar waren, setzt sich der 68000 doch erst in letzter Zeit erfolgreich gegen seinen schärfsten Konkurrenten, den 8086/8088 von INTEL durch. Maßgeblich für den Erfolg der INTEL-Prozessoren ist sicher die Tatsache, daß sie in den IBM-PCs und den diversen Nachbauten eingesetzt sind. Die Zahl der (preiswerten) mit einem 68000 ausgerüsteten Computer war jedoch bis zum Erscheinen der ATARI ST-Serie gleich Null. Jetzt aber sind diese leistungsstarken 16-Biter in eine Preisklasse gerutscht, die bislang vornehmlich 8-Bit-Rechnern vorbehalten war.

Was aber sind nun die hervorstechensten Merkmale des MC68000? In Schlagworten ausgedrückt läßt sich die folgende (sicher unvollständige) Tabelle aufstellen:

- *16 Datenbits*
- *24 Adressbits (16 Megabyte Adressbereich!)*
- *Alle Signale direkt ohne Multiplexer verfügbar*
- *Problemloser Betrieb 'alter' 8-Bit-Peripherie*
- *leistungsfähige Maschinenbefehle*
- *einfache und leicht zu lernende Assembler-Syntax*
- *14 verschiedene Adressierungsarten*
- *17 Register mit einer Breite von 32 Bit*

Die genannten und weitere, vorläufig unerwähnte Leistungsmerkmale machen den 68000 zu einem hervorragenden Mikroprozessor für Personal- und Homecomputer. Nachdem auch die Preise für Halbleiterspeicher immer weiter sinken, die

Kapazität der verfügbaren Speicher jedoch ständig steigt, bekommen die 8-Bit-Rechner mit 64 KByte Speicher zunehmend Konkurrenz durch den 68000. Bei diesen neuen Rechnern sind Speichergrößen üblich, die noch vor wenigen Jahren der mittleren Datentechnik vorbehalten waren.

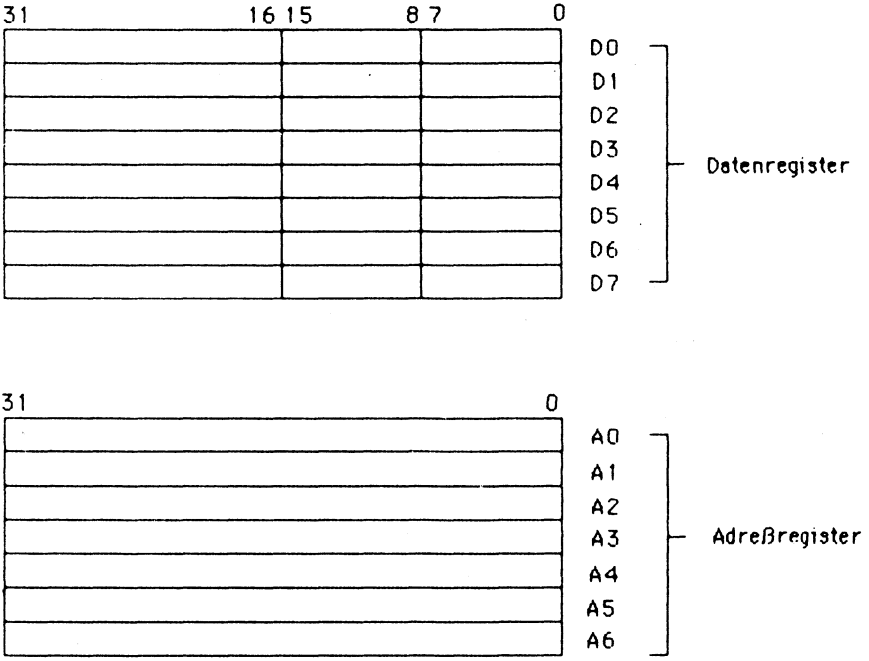
1.1.1 Die Register des 68000

Betrachten wir zunächst die Register des 68000. Die Darstellung Bild 1.1.1 zeigt nicht nur die 17 vorhandenen 32-Bit-Register, sondern auch denn Programm-Counter und das Status-Register.

Die acht vollständig identischen Daten-Register können zur Aufnahme und Berechnung von Daten, wie auch von Adressen dienen. Sie stellen gewissermaßen die von den 8-Bit-Prozessoren bekannten Akkumulatoren dar. Man verfügt also insgesamt über 8 Akkus, eine Erweiterung, die besonders Maschinenprogrammierer zu schätzen lernen werden. Der Zugriff auf die Daten in den Registern ist sehr flexibel. So können Daten mit 1-, 8-, 16- und 32-Bit-Größen verarbeitet werden. Auch eine (allerdings auf BCD-Zahlen beschränkte) 4-Bit-Verarbeitung ist möglich. Bei der Verarbeitung von Daten auf Bit-Ebene kann jedes der 32 Bits der Register gezielt beeinflußt werden. Bei Verarbeitung von 8- und 16-Bit-Daten kann allerdings nur auf die unteren 8 bzw. 16 Bit der Datenregister direkt zugegriffen werden.

Bei den Adressregistern kann nicht ganz so flexibel auf die Daten zugegriffen werden, wie dieses bei den Datenregisten möglich ist. Allerdings werden sie in der Hauptsache zur Adressierung und nicht zur Berechnung von Daten genutzt. Eine Bearbeitung der Daten ist nur mit Wort- (16 Bit) und Langwort-Operationen (32 Bit) möglich.
Die Adressregister müssen in zwei Gruppen unterschieden werden. Allgemein nutzbar sind die Adressregister A0 bis A6. Die Register A7 und A7' erfüllen einen speziellen Zweck. Diese Register werden vom Prozessor als Stackpointer

68000 Registersatz



benutzt. Die Ursache für das Vorhandensein zweier Stackpointer ist durch den Umstand begründet, daß der 68000 in den zwei Betriebsarten USER MODE und SUPERVISOR MODE operieren kann. Je nach Betriebsart wird der User Stackpointer oder der Supervisor Stackpointer als Register A7 angesprochen. Beachten Sie, daß tatsächlich zwei unterschiedliche Register 'unter' A7 arbeiten, jedoch nur das der jeweiligen Betriebsart zugeordnete als Registerinhalt verfügbar ist. Die Unterschiede zwischen den beiden Betriebsarten werden etwas später geklärt.

Auch der Program Counter ist als 32-Bit-Register ausgeführt. Damit wäre theoretisch ein Adressbereich von über 4 Gigabyte möglich. Beim 68000 werden jedoch die Adressbits A24 bis A31 nicht benutzt. Dadurch ist der durchgängig anzusprechende Adressbereich des 68000 auf 16 Megabyte 'beschränkt'.
Das Status-Register des 68000 umfaßt 16 Bits. Allerdings werden von diesen nur 10 Bits genutzt. Das Status-Register ist in zwei Hälften geteilt. Die unteren 8 Bits (eigentlich nur die Bits 0 bis 4) stellen das sogenannte User Byte dar. In diesen Bits, oft als Flags bezeichnet, werden Ergebnisse von arithmetischen und Vergleichsoperationen angezeigt und können zu ergebnisabhängigen Programmverzweigungen genutzt werden. Die Erklärung der einzelnen Bits im User-Byte würde an dieser Stelle zu weit führen. Wir wollen uns mit einer kurzen Aufstellung begnügen:

Bit 0 = Carry-Flag
Bit 1 = Overflow-Fag
Bit 2 = Zero-Flag
Bit 3 = Negativ-Flag
Bit 4 = eXtend-Flag

Das System-Byte des Status-Registers belegt die Bits 8 bis 10, 13 und 15. Die restlichen im Status-Register nicht belegten Bits ergeben beim Lesen der Daten immer Null.
Das Bit 15 des Status-Registers wird als Trace-Bit bezeichnet. Dieses Bit erlaubt einen rein softwaregesteuerten Single Step (Einzelschritt) durch ein beliebiges

Programm.
Das Bit 13 ist das Supervisor-Bit. Ist das S-Bit gesetzt, so befindet sich der 68000 im Supervisor Mode. Diese Betriebsart ist üblicherweise dem Betriebssystem vorbehalten. Nur in dieser Betriebsart kann der Prozessor alle Befehle ausführen. Im User Mode, in dem üblicherweise die Anwenderprogramme laufen, sind die sogenannten privilegierten Befehle nicht nutzbar. Durch einen speziellen Hardware-Aufbau kann der Zugriff auf Speicherbereiche (z.B. wichtige Systemvariablen, I/O-Bereich) im User Mode untersagt werden. Auch kann das System-Byte des Status-Registers nur im Supervisor Mode manipuliert werden. Damit kann auf einfache Weise vom Supervisor in den User Modus gewechselt werden. Umgekehrt ist diese Möglichkeit natürlich nicht gegeben, um das Konzept der strikten Trennung zwischen den beiden Bereichen nicht zu unterlaufen.
Die Bits 8 bis 10 werden als Interrupt-Maske bezeichnet und stehen in direktem Zusammenhang mit den Pins IPL0 bis IPL2.

Die Möglichkeiten, mit dem 68000 Interrupts zu verarbeiten, sind sehr leistungsstark. Insgesamt existieren 7 verschiedene Interrupt-Prioritäten. Die oberste Priorität besitzt der 'Nicht Maskierbare Interrupt', NMI. Dieser Interrupt wird erkannt, wenn alle drei IPL-Anschlüsse gleichzeitig Low werden. Sind dagegen alle drei IPL-Anschlüsse auf High-Pegel, so wird dieses als kein Interrupt, also normaler Betrieb interpretiert. Die verbleibenden 6 Möglichkeiten können mit Hilfe der drei Bits im System-Byte des Status-Registers maskiert werden. Der 68000 verfügt also über insgesamt 6 maskierbare Interruptebenen. Ist in der Interruptmaske z.B. das Bit I2 gesetzt, I0 und I1 dagegen gelöscht, so werden nur Interrupts der Ebenen 7, 6 und 5 (Kombinationen 000, 001 und 010) vom Prozessor erkannt und bearbeitet. Alle anderen Bitkombinationen an den Anschlüssen IPL0 bis IPL2 werden vom Prozessor ignoriert.

1.1.2 Die Exceptions des 68000

Haben wir bisher von Interrupts gesprochen, so nur aus dem Grunde, da dieser Begriff von anderen Prozessoren her üblich ist. Dabei sind die durch äußere Signale ausgelösten Interrupts nach MOTOROLA-Nomenklatur eine spezielle Art von Exception übersetzt etwa Ausnahme. Grundsätzlich gibt es beim Betrieb des 68000 nicht nur die Unterscheidung zwischen User- und Supervisormodus, sondern auch die Unterscheidung zwischen normalem Betrieb und Ausnahmebehandlung. Eine solche Ausnahme bilden die schon bekannten Interrupts. Weitere Ursachen für Exceptions sind unter anderem das Auftreten undefinierter Opcodes, Wort- oder Langwortzugriff auf ungerade Adressen, aber auch das Ausführen spezieller Opcodes, die gewissermaßen Software-Interrupts darstellen. Um das Bearbeiten der vielen verschiedenen Exceptions einfach zu gestalten, sind beim 68000 die ersten 1024 Bytes des Speichers ($000000 bis $0003FF) reserviert. In diesem Bereich liegt die Exception-Tabelle. Jeder Exception wird ein aus vier Bytes bestehendes Langwort zugeordnet. Tritt eine Exception auf, so holt der 68000 selbsttätig den Inhalt der zugehörigen Speicherplätze und setzt die Programmausführung an der enthaltenen Adresse fort. Einzige Ausnahme ist der ebenfalls den Exceptions zugeordnete Reset, dem insgesamt 8 Bytes, also zwei Langworte zugeordnet sind. Im ersten Langwort ist üblicher Weise der Initialwert des Supervisor Stackpointers enthalten. Das zweite Langwort enthält die Adresse der eigentlichen Reset-Routine. Der Aufbau und die Belegung der Exception-Tabelle ist im Kapitel 3.5 ausgeführt.

1.1.3 Die Anschluß-Belegung des 68000

Die Anschlüsse des 68000 können zu acht Gruppen zusammengefaßt werden. Die acht Gruppen finden Sie in der Grafik 1.1.3.

In der ersten Gruppe sind Daten- und Adressbus zusammengefaßt. Der Datenbus besetht aus den Anschlüssen **D0** bis **D15**, der Adressbus aus den Anschlüssen **A1** bis **A23**. Ein Adressbit **A0** ist beim 68000 nicht vorhanden. Daraus ließe sich schließen, daß der Speicher nur Wortweise (ein Wort = 2 Byte = 16 Bit) und nicht Byteweise (Byte = 8 Bit) angesprochen werden kann. Tatsächlich ist aber auch beim 68000 ein Byte-Zugriff sowohl auf die ungeraden wie auch die geraden Adressen möglich. Die hierfür erforderlichen Signale werden etwas später gezeigt.
Wichtig ist in diesem Zusammenhang, daß beim Wort-Zugriffen auf den Speicher das Byte von der ungeraden Adresse als Lowbyte und das Byte von der geraden Adresse als Highbyte betrachtet wird. Wort-Zugriffe auf den Speicher dürfen nur von geraden Adressen erfolgen dürfen. Das bedeutet, daß Opcodes (immer Worte oder ein vielfaches davon) ebenfalls immer auf geraden Adressen stehen müssen.
Sowohl Daten- wie auch Adressbus können zu bestimmten Zeiten in den sogenannten 'Tri-State'-Zustand, einen neben High und Low existierenden dritten Zustand übergehen, in dem die Pins hochohmig und damit auf dem Bus inaktiv sind. Diese Möglichkeit ist besonders im Zusammenhang mit DMA (DMA siehe unten) wichtig.

Die zweite Gruppe der Anschlüsse umfaßt die Signale zur asynchronen Bussteuerung. In dieser Gruppe finden sich fünf Signale, welche im Folgenden einzeln behandelt werden sollen.

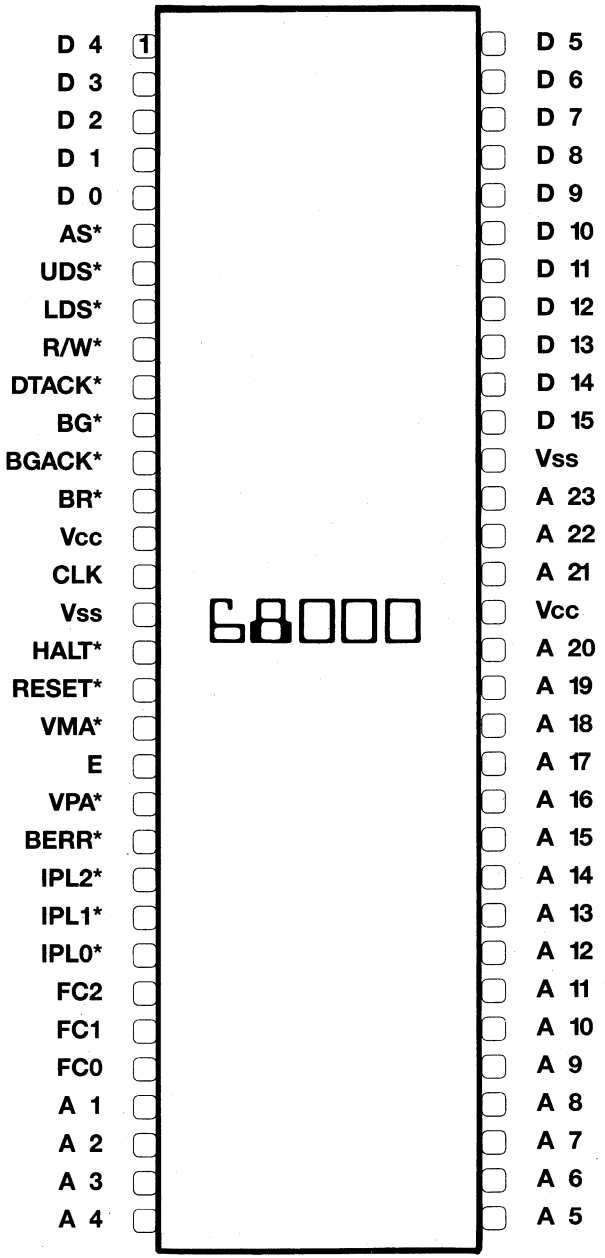
R/W (READ/WRITE)
Da ist zunächst das von allen Prozessoren bekannte R/W-Signal. Mit Hilfe dieses Signals wird nach 'außen' an Speicher und Peripherie signalisiert, ob der Prozessor Daten

von der momentanen Adresse lesen oder auf diese Adresse schreiben will. (R/W ist TRI-State)

AS (ADRESS STROBE)
Jeder Prozessor verfügt über ein Signal, welches mitteilt, daß die an den Datenleitungen anliegenden Adressen 'gültig' sind. Dieses Signal wird beim 68000 ADRESS STROBE (low aktiv), kurz AS genannt.

UDS,LDS (UPPER DATA STROBE, LOWER DATA STROBE)
Könnte der MC68000 (oft auch als 68k bezeichnet) nur auf Speicherworte zugreifen, so wären diese Signale nicht nötig. Da aber auch ein Byte-Zugriff auf das Low-Byte und das High-Byte eines Wortes möglich ist, muß der Prozessor eine Möglichkeit haben, die beiden Bytes zu unterscheiden. Diese Unterscheidung wird im System an Hand von UDS (Upper Data Strobe) und LDS (Lower Data Strobe) vorgenommen. Greift der Prozessor auf ein Wort zu, so werden LDS und UDS gleichermaßen aktiviert, also Low. Beim Zugriff auf ein Byte auf einer ungeraden Adresse (odd adress) jedoch wird nur der Lower Data Strobe aktiviert. Entsprechend wird UPPER DATA STROBE beim Zugriff auf Bytes an geraden Adressen (even adress) Low.
Durch diesen Mechanismus erübrigt sich das im Adressbus fehlende Bit A0. Je nach Zugriff (word, byte even, byte odd) muß zwischen drei verschiedenen Zuständen unterschieden werden, eine Aufgabe, die nur ein Signal (A0) nicht lösen könnte.
Sowohl LDS wie UDS sind TRI-State-Ausgänge.

DTACK
Mit Ausnahme der Signale LDS und UDS sind die bisherigen Signale auch für einen 8-Bit-Prozessor nötig. Anders jedoch verhält es sich mit dem Signal DTACK. Bei jedem Lese- und Schreibzugriff muß DTACK Low werden, um anzuzeigen, daß die Daten von Prozessor gelesen werden können bzw. von Speicher oder Peripherie übernommen worden sind. Wird dieses Signal nicht innerhalb eines Buszyklus Low, so werden automatisch Adress- und Datenleitungen quasi eingefroren, bis das


D 4
D 3
D 2
D 1
D 0
AS*
UDS*
LDS*
R/W*
DTACK*
BG*
BGACK*
BR*
Vcc
CLK
Vss
HALT*
RESET*
VMA*
E
VPA*
BERR*
IPL2*
IPL1*
IPL0*
FC2
FC1
FC0
A 1
A 2
A 3
A 4
D 5
D 6
D 7
D 8
D 9
D 10
D 11
D 12
D 13
D 14
D 15
Vss
A 23
A 22
A 21
Vcc
A 20
A 19
A 18
A 17
A 16
A 15
A 14
A 13
A 12
A 11
A 10
A 9
A 8
A 7
A 6
A 5

DTACK-Signal Low wird. Man spricht auch von Warte-Zyklen, die der Prozessor einlegt. Auf diese einfache Art und Weise kann der Prozessor auch mit langsamen Speichern oder Peripherie-Bausteinen zusammenarbeiten. Durch die Warte-Zyklen wird zwar die effektive Arbeitsgeschwindigkeit des Prozessors gedrosselt, bei geschickter Schaltungsauslegung jedoch wird das DTACK-Signal nur in den Adressbereichen verzögert, in denen eine Wartezeit nötig ist. Beim ATARI-ST werden für den Zugriff auf den Speicher keine Warte-Zyklen eingebaut, der Prozessor 'fährt immer mit Volldampf'.

Die Gruppe der Anschlüsse zur synchronen Bussteuerung besteht aus den drei Signalen **VMA, VPA** und **E**. Ein Computer besteht nicht nur aus Prozessor und Speicher, sondern muß auch über Schnittstellen zum Anschluß von Tastatur, Bildschirm, Drucker etc. verfügen. Diese Schnittstellen werden in den meisten Fällen über spezielle ICs realisiert. Nun war bei der Entwicklung des 68000 bereits eine breite Palette verschiedener Schnittstellenbausteine vorhanden. Um den Hardware-Entwicklern den Einsatz dieser Schnittstellen-Bausteine zu erleichtern, wurden die Signale des synchronen Busses eingeführt.

Das Signal **E**, oft auch 02 (phi 2) genannt, stellt den Bezugstakt für die Peripherie dar. Anwender von Prozessoren der Baureihen 68xx und 65xx kennen dieses Signal als Systemtakt. Da die Peripherie-Bausteine oft nur eine maximale Arbeitsfrequenz von 1 oder 2 MHZ haben, der 68000 jedoch in der Standardausführung mit 8MHz arbeitet, wird zur Erzeugung des E-Signals der Arbeitstakt des Prozessors durch 10 geteilt. Damit ergibt sich die Frequenz des Signals E im ST zu 800 kHz. Der E-Ausgang ist immer aktiv. Er verfügt nicht über die Möglichkeit, in den TRI-STATE-Zustand überzugehen.

Das Signal **VPA** (Valid Peripheral Adress) ist die Rückmeldung an den 68000, daß auf der anliegenden Adresse ein über den synchronen Bus zu betreibendes Bauteil handelt. Ohne dieses Signal würde der Datentransfer in der üblichen Weise über

den asynchronen Bus ablaufen. Auch im Zusammenhang mit der Interrupt-Erzeugung spielt das Signal VPA eine Rolle, wie wir später sehen werden.

VMA (Valid Memory Adress) schließlich wird im Anschluß an die VPA-Generierung erzeugt um das CHIP-Select-Signal für den synchronen Bus in eine für die Peripherie-Bausteine verträgliche Form zu bringen.

Eine weitere Gruppe von Signalen des 68000 erlaubt einen einfachen DMA-Betrieb in 68000-Systemen. Mit DMA wird der direkte Zugriff (Direkt Memory Access) spezieller Bausteine, den sogenannten DMA-Controllern, auf den Speicher des Computers bezeichnet. DMA stellt die schnellste Methode zum Datentransfer innerhalb eines Computersystems dar.
Um DMA ausführen zu können, muß der Prozessor in einen möglichst inaktiven Zustand übergehen. Zuvor jedoch muß dem Prozessor erst einmal signalisiert werden, daß er sich (sobald möglich) quasi tot stellt. Zu diesem Zweck wird das Signal **BR** (Bus Request, Busanforderung) vom DMA-Controller auf Low gelegt. Sobald der 68000 das Signal BR erkennt, wird er den laufenden Schreib- oder Lesezyklus beenden und danach das Signal **BG** (Bus Grant, Buszuteilung) aktivieren. Jetzt muß der DMA-anfordernde Baustein warten, bis die Signale AS, DTACK und (möglicherweise) **BGACK** inaktiv werden. Sobald das geschehen ist, wird das Signal BGACK (Bus Grant Acknowledge, Buszuteilungserkennung) vom anfordernden Baustein aktiviert und damit der Bus übernommen. Am Prozessor sind jetzt alle wesentlichen Signale hochohmig, insbesondere Daten-, Adress- und Steuerbus werden vom Prozessor nicht weiter beeinflußt. Somit kann der DMA-Controller die gewünschten Adressen auf den Bus legen und Daten schreiben oder lesen. Ist die Aktion des DMA-Chips beendet, so nimmt er das BGACK-Signal in den inaktiven Zustand und der Prozessor übernimmt wieder den Bus.

Die fünfte Signal-Gruppe des 68000 umfaßt die Signale zur Interrupt-Erzeugung. Das im 68000 gewählte Interrupt-Konzept besticht durch seine außergewöhnlichen Leistungsmerkmale.

199 Interrupt-Vektoren stehen dem Anwender insgesamt zur Verfügung. Diese Interrupt-Vektoren werden unterteilt in 7 Autovektoren und 192 Non-Autovektoren. Außerdem stehen 7 verschiedene Interruptebenen (Prioritätsebenen) zur Verfügung.
Ausgelöst werden Interrupts durch Signale an den drei Anschlüssen **IPL0** bis **IPL2**. Mit diesen drei Anschlüssen sind acht verschiedene Kombinationen möglich. Die anliegende Kombination legt die Priorität des Interrupt fest. Dabei hat die Kombination, bei der IPL0, IPL1 und IPL2 auf High liegen, die geringste Priorität, sie signalisiert den Zustand 'kein Interrupt'. Sind dagegen alle drei Anschlüsse Low, so handelt es sich bei dem Interrupt um einen nicht maskierbaren Interrupt der immer ausgeführt wird. Bei allen dazwischen möglichen Kombinationen ist es von speziellen Bits im Status-Register des 68000 und damit von der Programmkontrolle abhängig, ob ein anliegender Interrupt zugelassen ist, oder nicht.

Was aber bedeuten die Begriffe Autovektor und Non-Autovektor?
Wird bei Anliegen eines Interrupts an den Pins IPL0 bis IPL2 das Signal VPA aktiv (low), so wird der an den IPL-Pins anliegende Code direkt in eine Vektor-Nummer umgerechnet. Jeder der sieben verschiedenen Interruptcodes an den IPL-Anschlüssen verfügt über einen eigenen Vektor. Der Begriff Autovektor ergibt sich durch die Tatsache, daß die Vektornummer automatisch mit dem anliegenden Interrupt-Code an IPL0 - IPL2 feststeht.
Wird jedoch im Falle eines Interrupts an Stelle des Signals VPA das Signal DTACK Low, so handelt es sich um einen Non-Autovektor. In diesem Fall wird die Vektornummer vom Interrupt auslösenden Baustein nach Erzeugung des Signals DTACK auf die unteren 8 Bit des Datenbusses gelegt. Üblicher-, aber nicht notwendigerweise wird die Vektornummer im Bereich der Anwender-Interruptvektoren von $40 bis $FF liegen.

Die sechste Gruppe von Anschlüssen besteht aus den drei 'Function Code'-Ausgängen **FC0** bis **FC2**. Bei diesen Anschlüssen handelt es sich um Status-Anzeigen des Prozessors. Mit Hilfe dieser Anschlüsse könnte der 68000 bis auf vier mal 16 Megabyte = 64 Megabyte ausgebaut werden. Dieser Ausbau erfordert allerdings eine MMU, Memory Management Unit. Die MMU des ST ist für diesen Speicherausbau nicht vorgesehen. Allerdings wird von der MMU erkannt, ob ein Zugriff auf den Speicher im User- oder Supervisor-Mode vorgenommen wird. Diese Information wird ausgewertet, da einige Speicherbereiche nur im Supervisor-Mode zugänglich sind. Auch die Interrupt-Bestätigung wird durch die Information der FC-Anschlüsse vorgenommen. Die möglichen Kombinationen und Funktionen zeigt die folgende Tabelle.

FC2	FC1	FC0	Status
0	0	0	unbestimmt
0	0	1	User-Modus Datenzugriff
0	1	0	User-Modus Programm
0	1	1	unbestimmt
1	0	0	unbestimmt
1	0	1	Supervisor Datenzugriff
1	1	0	Supervisor Programm
1	1	1	Interruptbestätigung

In der letzten Signalgruppe sind Signale zur Systemsteuerung enthalten. Zu dieser Gruppe zählen die Eingänge **CLK** und **BERR** sowie die bidirektionalen Anschlüsse **RESET** und **HALT**.

An den Eingang **CLK** wird die Arbeitsfrequenz des Prozessors angelegt. Der 68000 wird in verschiedenen Geschwindigkeitsstufen ausgeliefert. Je nach Spezifikation darf die maximale Betriebsfrequenz 4, 6, 8, 10 oder sogar 12,5 MHz betragen. Im ST ist die 8-MHz-Version eingebaut. Die minimale Arbeitsfrequnez des 68000 beträgt etwa 2 MHz. Da die interen Register dynamisch sind, darf die Arbeitsfrequenz nicht zu niedrig werden, da sonst die Information in den Registern nicht rechtzeitig aufgefrischt

wird und verloren geht.

Der RESET-Anschluß wird benötigt, um beim Einschalten des Systems definierte Zustände zu erzielen. Laut Datenblatt des 68000 sind zwei unterschiedliche Reset-Zustände zu unterscheiden. Beim Power-On, also beim Einschalten des Systems müssen RESET und HALT-Anschluß für mindestens 100 Millisekunden auf Low gehalten werden, um ein korrektes Initialisieren zu erreichen. Für jede weitere Initialisierung reicht danach ein mindestens 4 Clock-Takte langer Low-Impuls am 68k.
Im Detail wird beim RESET folgendes vorgenommen:
Das Systembyte des Statusregisters wird mit dem Wert $27 geladenden. Dadurch wird der Prozessor in den Supervisor-Status gebracht, das Trace-Flag im Statusregister ist gelöscht und der Interruptlevel wird auf 7 (niedrigste Priorität, alle Ebenen zugelassen) eingestellt. Zusätzlich werden Supervisor-Stackpointer und Program Counter mit dem Inhalt der ersten 8 Bytes des Speichers geladen, wobei der Wert des Program Counters durch den Anfang der Reset-Routine bestimmt wird.
Da jedoch der RESET-Anschluß bidirektional ist, kann auch der Prozessor den RESET-Pin unter Programmkontrolle für eine bestimmte Zeit auf Low legen. Für diesen Zweck existiert der Befehl RESET, der den Anschluß für 124 Takte auf Low legt. Damit ist dem Prozessor die Möglichkeit gegeben, seine Peripherie-ICs zu jedem beliebigen Zeitpunkt zu initialisieren, ohne selbst initialisiert zu werden. In der RESET-Zeit führt der 68000 intern NOPs aus und ist nicht unterbrechbar.

Der HALT-Anschluß wird, wie bereits erwähnt, in Verbindung mit dem RESET-Anschluß benötigt, um nach dem Einschalten eine ordnungsgemäße Initialisierung des Prozessors zu erreichen. Allerdings hat dieser Anschluß noch weitere Funktionen. Wird an den Anschluß ein Low gelegt, während der RESET-Pin High bleibt, so wird der Prozessor in den Halt-Zustand gebracht. In diesem Zustand werden bis auf die DMA-Anschlüsse alle Ausgänge des 68000 in den

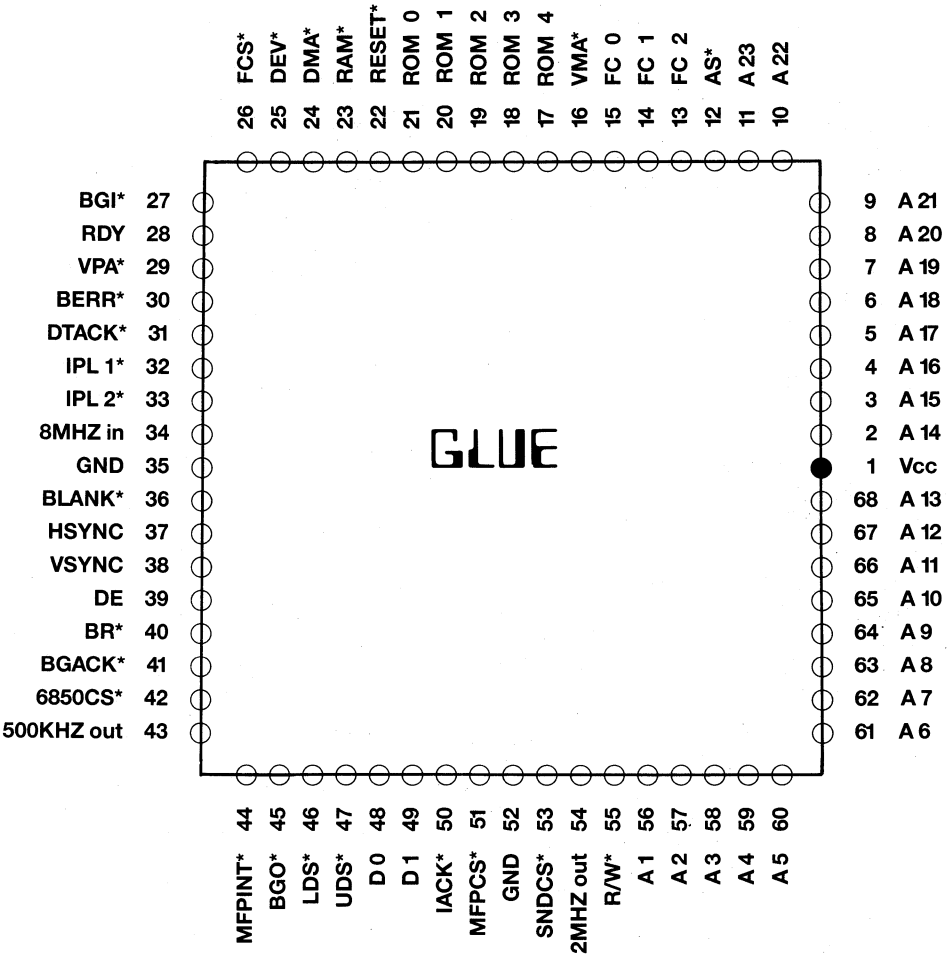
Tri-State-Zustand versetzt. Der Halt-Zustand endet, wenn HALT wieder High wird. In der Praxis kann mit diesem Signal eine einfache Einzelschrittsteuerung aufgebaut werden.
Da der HALT-Pin bidirektional ist, kann er auch als Ausgang arbeiten. Legt der Prozessor selbst diesen Anschluß auf Low, so signalisiert er damit, daß ein besonders schwerwiegender Systemfehler aufgetreten ist. Solche schweren Fehler sind z.B. doppelte Bus- und Adressfehler.

Mit einem Low an dem **BERR**-Anschluß wird eine Exception-Behandlung eingeleitet, die im Grunde wie ein externer Interrupt abläuft.
In einem ordnungsgemäß funktionierenden System wird jeder Zugriff über den asynchronen Bus mit dem DTACK-Signal quittiert. Wenn jedoch das DTACK-Signal aus irgendwelchen Gründen ausbleibt, so kann mit einer geeigneten Hardware ein BERR erzeugt werden, um den Prozessor davon zu unterrichten, daß ein Fehler aufgetreten ist. Ein weiterer Verwendungszweck kann in Verbindung mit einer MMU das Überprüfen der Zugriffsberechtigung auf bestimmte Speicherbereiche sein. Die Art des Zugriffs wird ja an den FC-Pins signalisiert. Wird im User-Mode auf geschützte Speicherbereiche zugegriffen, so wird in der Regel ein BERR erzeugt werden.
Wird BERR zusammen mit dem HALT-Anschluß Low, so versucht der Prozessor automatisch, den abgebrochenen Befehl zu wiederholen. Ist auch der zweite 'Anlauf' nicht erfolgreich, so spricht man von einem doppelten Busfehler, der den Prozessor in den Halt-Zustand versetzt.

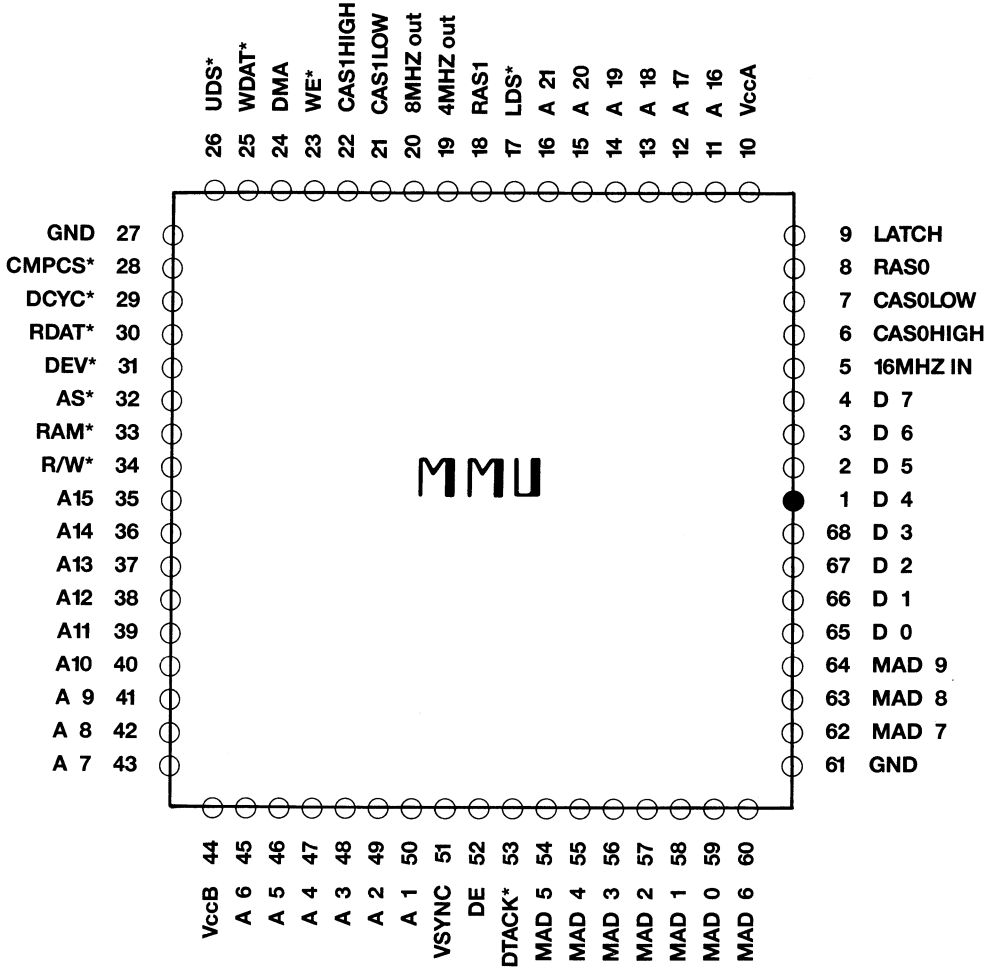
1.2 Die ATARI-Custom Chips

Insgesamt vier von ATARI speziell entwickelte ICs versehen im ST ihren Dienst. Diese Chips mit den Namen GLUE, MMU, DMA und SHIFTER spielen eine enorme Rolle für die ausgesprochene Preiswürdigkeit des Systems. In diesen Chips sind viele hundert Gatterfunktionen integriert. Die ersten Prototypen des ST bestanden aus fünf ca. 50 mal 30 cm großen, fast vollständig mit TTL-ICs gestückten Leiterplatten. Erst durch die Integration dieser vielen TTL-Gatter in vier ICs wurde die Serienproduktion des ST möglich und rentabel. Jetzt gibt es nur noch acht (in Buchstaben ACHT!!) TTL-ICs im ATARI ST. Da die ICs speziell für den ATARI ST entwickelt wurden, ist natürlich kein Datenblatt von ATARI erhältlich. Aber auch ohne Datenblätter können wir Ihnen eine Menge Information über die Funktion der Chips mitteilen.
Da die Chips nach einem funktionierenden Prototypen entwickelt wurden, ergeben sich einige interressante Fakten. So sind die ICs praktisch nur im Verbund arbeitsfähig. Der DMA-Chip z.B. kann nicht allein in einem beliebigen anderen Computersystem als DMA-Controller eingesetzt werden, da ihm eine wesentliche Funktion fehlt. Er besitzt keinen eigenen Adresszähler und kann somit auch nicht selbsttätig den Speicher adressieren. Diese Funktion wird von der MMU übernommen. Genauso ist es dem SHIFTER ergangen. Dieses für die Erzeugung des Video-Bildes und der Farben verantwortliche IC hat ebenfalls keine Möglichkeit das Video-RAM zu adressieren. Auch hier springt die MMU hilfreich ein, indem sie die Adressierung übernimmt.

Letztendlich kann es dem Systemprogrammierer ziemlich egal sein, in welchem IC sich welches Register befindet. Wesentlich ist nur, daß die Adressen der Register bekannt sind und man weiß, was mit ihnen zu steuern ist. Darum wollen wir in diesem Kapitel nur kurz auf die Anschlüsse der einzelnen ICs eingehen. Sie sollen ja nicht wie ein 'Ochs vorm Berg' sitzen und sich fragen, wie um alles in der Welt das Ganze funktioniert.

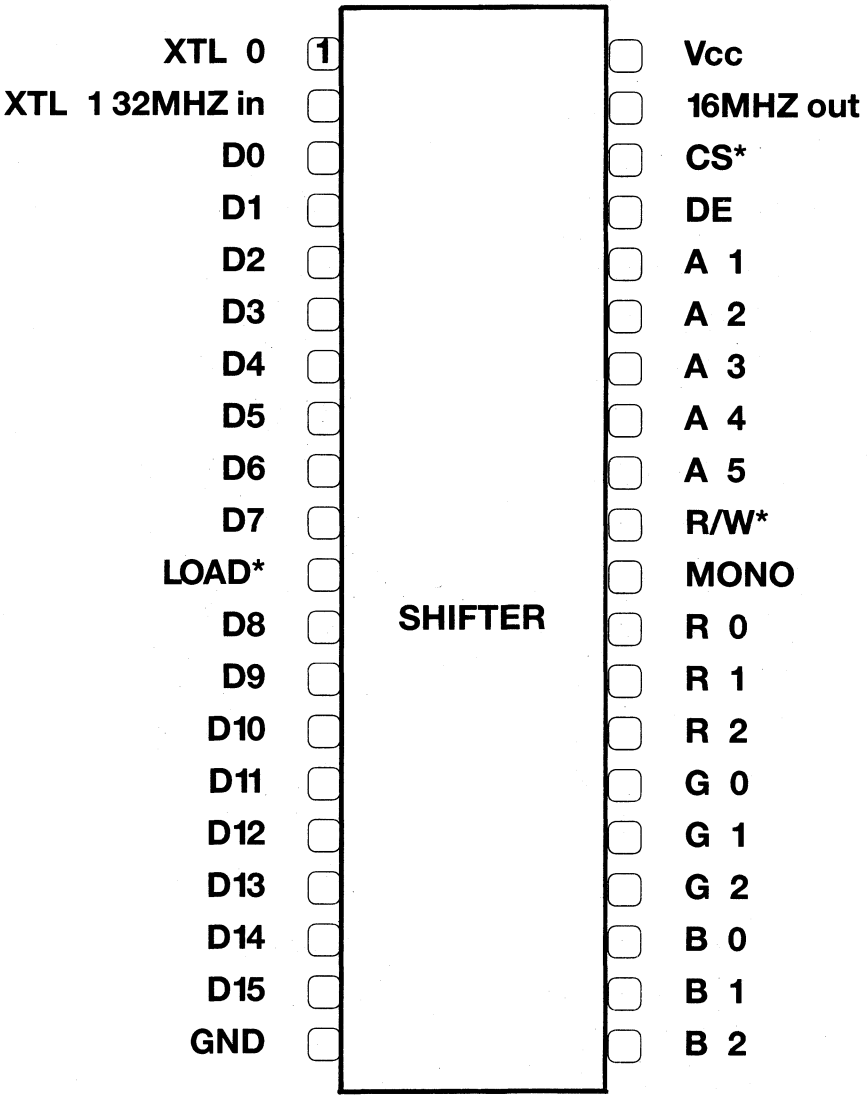

GLUE
26 FCS*
25 DEV*
24 DMA*
23 RAM*
22 RESET*
21 ROM 0
20 ROM 1
19 ROM 2
18 ROM 3
17 ROM 4
16 VMA*
15 FC 0
14 FC 1
13 FC 2
12 AS*
11 A 23
10 A 22
BGI* 27
RDY 28
VPA* 29
BERR* 30
DTACK* 31
IPL 1* 32
IPL 2* 33
8MHZ in 34
GND 35
BLANK* 36
HSYNC 37
VSYNC 38
DE 39
BR* 40
BGACK* 41
6850CS* 42
500KHZ out 43
9 A 21
8 A 20
7 A 19
6 A 18
5 A 17
4 A 16
3 A 15
2 A 14
1 Vcc
68 A 13
67 A 12
66 A 11
65 A 10
64 A 9
63 A 8
62 A 7
61 A 6
44 MFPINT*
45 BGO*
46 LDS*
47 UDS*
48 D 0
49 D 1
50 IACK*
51 MFPCS*
52 GND
53 SNDCS*
54 2MHZ out
55 R/W*
56 A 1
57 A 2
58 A 3
59 A 4
60 A 5

Das sicher wichtigste IC der 'Vierer-Bande' ist der GLUE. Die Bezeichnung deutet schon auf die Funktion: Glue bedeutet etwa Leim oder Kleister. Dies IC mit seinen insgesamt 68 Anschlüssen hält das gesamte System zusammen, indem es die annähernd vollständige Decodierung der Adressbereiche für das RAM und die Peripherie-ICs vornimmt. Die Decodierung ist auch der Grund, weshalb dieses IC als einziges im gesamten System den vollständigen Adressbus zur Verfügung hat.
Weiterhin werden die DMA-Handshake-Signale BR, BG und BGACK vom GLUE erzeugt bzw. ausgewertet. Die Zeitpunkte der DMA-Anforderung werden dem GLUE durch Signale vom DMA-Controller diktiert. Interessanter Weise verfügt der GLUE neben dem BG (Bus Grant) Eingang auch noch über einen BGO, Bus Grant Out. Das läßt erkennen, daß die ICs schon in Hinblick auf eine eventuelle spätere Erweiterung konzipiert wurden.
Auch die Interrupt-Signale werden vom GLUE erzeugt. Im ST werden nur die Anschlüsse IPL1 und IPL2 verwendet. Ohne zusätzliche Hardware kann also kein NMI (Interruptlevel 7) ausgelöst werden. Die Anschlüsse MFPINT und IACK sind ebenfalls für die Interrupt-Steuerung zuständig.
Die Function Code-Anschlüsse sind an den GLUE geführt, da er auch die Bereichsüberprüfung und Zugriffsberechtigung bei Speicherzugriffen vornimmt. Dazu muß natürlich das BERR-Signal vom GLUE im Fehlerfall erzeugt werden. Aber auch DTACK und VMA/VPA können vom GLUE erzeugt werden. VPA ist besonders in Hinblick auf die Chip Select-Signale für die Peripherie-ICs nötig.
Als Takt-Frequenz erhält der GLUE eine Frequenz von 8 MHz. Durch Teilerketten werden die Frequenzen 2 MHz (Betriebsfrequenz des Soundchip) und 500 kHz (Taktfrequenz für Tastatur- und Midi-ACIA) erzeugt.
Auch die Signale HSYNC, VSYNC, BLANK und DE (Display Enable) für den Betrieb des Monitors werden durch Teilerketten im GLUE erzeugt. Allerdings kann die Erzeugung der Synchron-Takte im GLUE ausgeschaltet werden. In diesem Fall können externe Sync-Signale am Monitor-Anschluß eingespeist werden. Damit kann das Bild des ST mit dem Bild einer Video-Kamera synchronisiert werden.


MMU
26 UDS*
25 WDAT*
24 DMA
23 WE*
22 CAS1HIGH
21 CAS1LOW
20 8MHZ out
19 4MHZ out
18 RAS1
17 LDS*
16 A 21
15 A 20
14 A 19
13 A 18
12 A 17
11 A 16
10 VccA
GND 27
CMPCS* 28
DCYC* 29
RDAT* 30
DEV* 31
AS* 32
RAM* 33
R/W* 34
A15 35
A14 36
A13 37
A12 38
A11 39
A10 40
A 9 41
A 8 42
A 7 43
9 LATCH
8 RAS0
7 CAS0LOW
6 CAS0HIGH
5 16MHZ IN
4 D 7
3 D 6
2 D 5
1 D 4
68 D 3
67 D 2
66 D 1
65 D 0
64 MAD 9
63 MAD 8
62 MAD 7
61 GND
44 VccB
45 A 6
46 A 5
47 A 4
48 A 3
49 A 2
50 A 1
51 VSYNC
52 DE
53 DTACK*
54 MAD 5
55 MAD 4
56 MAD 3
57 MAD 2
58 MAD 1
59 MAD 0
60 MAD 6

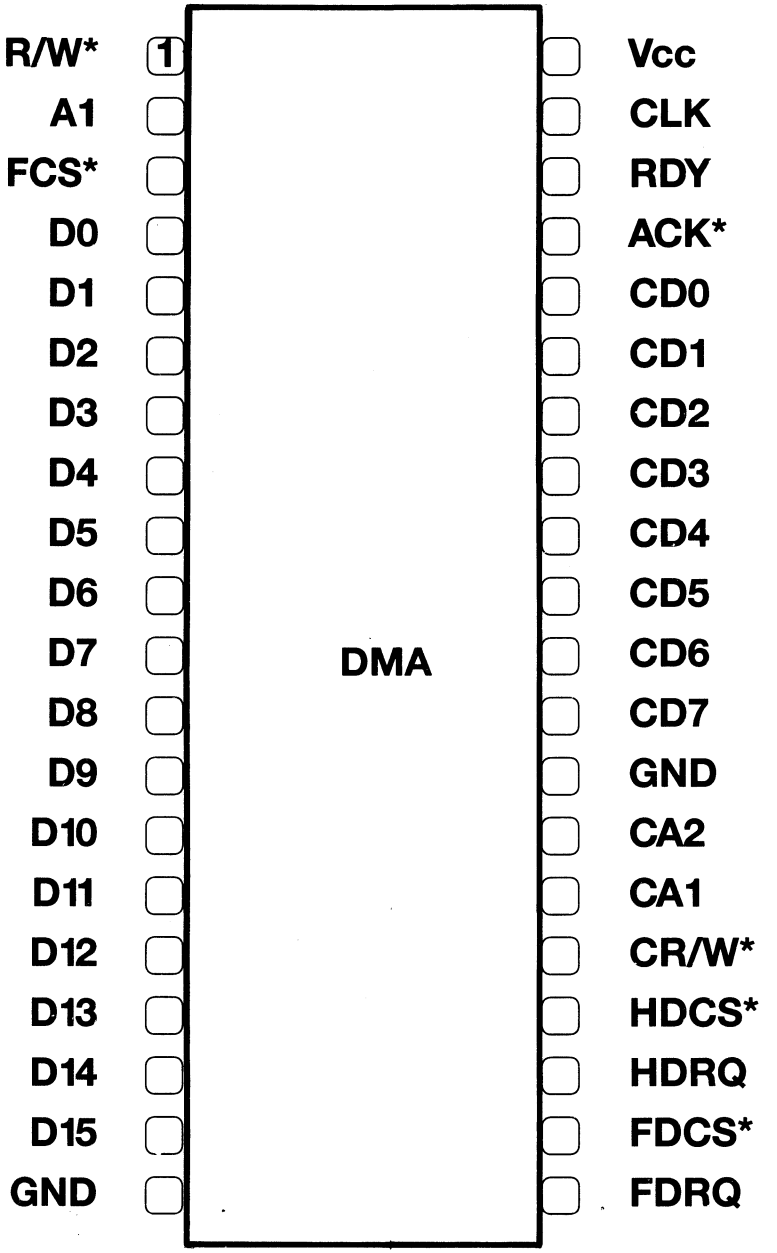
Auch die MMU verfügt über insgesamt 68 Anschlüsse. Dieses IC übernimmt hauptsächlich drei wesentliche Aufgaben. Die wichtigste Aufgabe ist die Kopplung des gemultiplexten Adressbusses der dynamischen RAMs an den Bus des Prozessors. Dazu sind die Adressleitungen A1 bis A21 an die MMU gelegt. Das ergibt einen Adressbereich von insgesamt 4 Megabyte. Tatsächlich ist sowohl Hardware wie auch Betriebssystem des ST für solche Speichergrößen ausgelegt. Zum Betrieb der dynamischen RAMS werden die Signale RAS0, RAS1, CAS0L, CAS0H, CAS1L und CAS1H sowie der gemultiplexte Adressbus aus der MMU herausgeführt. Natürlich sind auch die Signale DTACK, R/W, AS, LDS und UDS an die MMU geführt beziehungsweise aus ihr herausgeführt.
Eine weitere wesentliche Aufgabe der MMU wurde bereits erwähnt. Sie unterstützt den SHIFTER beim Erzeugen der Video-Signale, indem sie die Bildinformation im RAM des ST adressiert und dem SHIFTER übergibt. Zu diesem Zweck ist eine Zählerkette in der MMU integriert, die mit einem Startwert geladen wird und kontinuierlich alle 500 Nanosekunden ein Wort im Speicher adressiert und die enthaltene Information dem SHIFTER mit dem Signal DCYC übergibt. Der Startwert des Video-Zählers und damit die Lage des Bildschirmspeichers kann beliebig in 256-Byte-Schritten verschoben werden.
Eine weitere in der MMU integrierte Zählerkette ist, wie bereits kurz erwähnt, für die Adressierung des Speichers bei DMA zuständig. Dieser Zähler wird zu Beginn einer DMA-Operation (Disk oder Harddisk) mit der Adresse geladen, in den oder aus dem die Daten transportiert werden. Mit jedem übertragenen Wort wird der Zähler automatisch erhöht.

Der SHIFTER hat die Aufgabe, aus der im Video-RAM gespeicherten Information Impulse für den Monitor zu erzeugen. Wird der ST in der Auflösungsstufe 640 x 200 oder 320 x 200 betrieben, dann werden auch die Farbinformationen vom SHIFTER erzeugt.
Die Information aus dem RAM gelangt mit dem Signal LOAD über den Datenbus in den SHIFTER. Bei einer Auflösung von 640 x 400 Punkten wird das erzeugte Video-Signal am MONO-Anschluß


XTL 0
1
XTL 1 32MHZ in
D0
D1
D2
D3
D4
D5
D6
D7
LOAD*
D8
D9
D10
D11
D12
D13
D14
D15
GND
SHIFTER
Vcc
16MHZ out
CS*
DE
A 1
A 2
A 3
A 4
A 5
R/W*
MONO
R 0
R 1
R 2
G 0
G 1
G 2
B 0
B 1
B 2

ausgegeben. Da in dieser Auflösung keine Farben möglich sind, sind die jeweils dreifach vorhandenen RGB-Ausgänge inaktiv. In den beiden anderen Auflösungen ist dagegen der MONO-Ausgang inaktiv, jetzt wird die Bildinformation aus den Signalen an den RGB-Ausgängen gebildet.
Die drei zu einer Farbe gehörenden Ausgänge arbeiten zusammen mit externen Widerständen als Digital/Analog-Wandler. Je nach eingestellter Farbe ergeben sich an den Ausgängen unterschiedliche Pegel, die vom Monitor in unterschiedliche Farben umgesetzt werden.
Über die mit dem Adressbus verbundenen Anschlüsse R1 bis R5 werden die sogenannten Palettenregister angesprochen. In den Palettenregistern sind die Werte der Farben enthalten, die den einzelnen Bitmustern zugeordnet sind. Insgesamt sind 16 Paletten-Register entsprechend den maximal 16 gleichzeitig darstellbaren Farben im 320 x 200-Modus vorhanden. Da jedoch alle 16 Farben aus den drei Grundfarben Rot, Grün und Blau in jeweils 8 Stufen erzeugt werden können, beträgt die Gesamtzahl der darstellbaren Farben 512.

Der DMA-Controller ist wie der SHIFTER in einem handelsüblichen 40-poligen Gehäuse untergebracht. Seine Aufgaben sind die Ansteuerung des Floppy-Controllers, der Harddisk und eventueller weiterer Peripherie (z.B. Netzwerk).
Die Geschwindigkeit des Datentransfers beim Betrieb von Floppy-Laufwerken könnte noch ohne Probleme vom Prozessor übernommen werden. Bei Harddisks jedoch ist die Geschwindigkeit des Datentransfers so hoch, daß selbst einem mit 8 MHz getakteten 68000 die 'Puste' ausgeht. Diese hohen Geschwindigkeiten sind nur noch mittels DMA erzielbar.
Da der DMA-Controller Daten transportiert, ist er mit dem Datenbus des Prozessors verbunden. Intern verfügt er über zwei Register, von denen eines quasi als bidirektionaler Buffer der Daten zum DMA-Port arbeitet. Die genaue Verwendung der Register werden wir später besprechen. Interessant ist jedoch die Tatsache, daß der 16-Bit Datenbus des Prozessors auf einen 8-Bit-Datenbus auf der Floppy- und Harddisk-Seite reduziert wird. Beim Datentransfer werden automatisch je nach Richtung zwei Bytes zu einem Wort


R/W*
A1
FCS*
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15
GND
1
DMA
Vcc
CLK
RDY
ACK*
CD0
CD1
CD2
CD3
CD4
CD5
CD6
CD7
GND
CA2
CA1
CR/W*
HDCS*
HDRQ
FDCS*
FDRQ

zusammengefügt oder ein Wort in zwei Bytes zerlegt und übertragen.

Zur Ansteuerung des Floppy-Controllers dienen die Signale CA1 und CA2, CR/W und FDCS sowie das vom Floppy-Controller gelieferte Signal FDRQ. CA1 und CA2 stellen Register-Auswahlsignale dar, mit denen die Register des FDC (Floppy Disk Controller) selektiert werden können. CR/W bestimmt die Richtung der Datenübertragung von und zum FDC, aber auch zu weiterer am DMA-Port angeschlossener Peripherie.
Das Signal RDY dient zur Kommunikation mit dem GLUE (DMA-Anforderung) und der MMU (Adresszähler). Mit diesem Signal zeigt der DMA an, daß ein Wort übertragen werden soll.

Wie man sieht, ist ein separater Betrieb der einzelnen ICs annähernd unmöglich. Alle Funktionen sind so ineinander verzahnt, das sich bei fast allen Operationen die einzelnen ICs ergänzen.

1.3 Der Floppy Disk Controller WD 1772

Obwohl der 1772 von WESTERN DIGITAL über nur 28 Pins verfügt, enthält der Chip doch einen vollständigen Floppy Disk Controller (FDC) mit eingebauten Möglichkeiten, die so manchen 40-poligen FDC in den Schatten stellen. Dieses IC ist zu den bekannteren Controllern der Serien 179x/279x weitgehend softwarekompatibel. Leser, die mit Controllern dieser Serien Erfahrung haben, werden die Ähnlichkeiten schnell bemerken.

Im Telegrammstil lassen sich die folgenden Möglichkeiten des 1772 aufzählen:

- *Einfache 5 Volt-Versorgung*
- *Eingebauter Datenseparator*
- *Eingebaute Schreib-Vorkompensationslogik*
- *Single und Double Density*
- *Eingebaute Motorkontrolle*
- *DMA-fähig*

Obwohl dem Anwender in der Wahl der benutzten Diskettenformate, also z.B. Sektorlänge, Anzahl der Sektoren pro Track und Anzahl der Tracks pro Diskette vom Controller fast keine Einschränkungen gemacht werden, ist das verwendete Format wegen der festgelegten Datenübertragung doch spezifiziert. So können z.B. Apple- oder Commodore-Disketten mit diesem Controller nicht verarbeitet werden.
Bevor wir jedoch auf die Möglichkeiten des FDC eingehen, wollen wir einen Blick auf die 28 Anschlüsse des ICs werfen.

1.3.1 Die Anschlüsse des 1772

Die Anschlüsse des FDC 1772 lassen sich grob in drei Gruppen zusammenfassen. Die erste Gruppe umfasst die Versorgungsanschlüsse.

Vcc :
Stromversorgung + 5 Volt

GND :
Masseanschluss

MR :
Master Reset. Mit einem Low an diesem Anschluß wird der FDC in einen definierten Grundzustand versetzt.

Das Prozessor-Interface des FDC bildet die zweite Gruppe von Anschlüssen. Über diese Pins läuft (in der Hauptsache) der Datentransfer zwischen FDC und Prozessor.

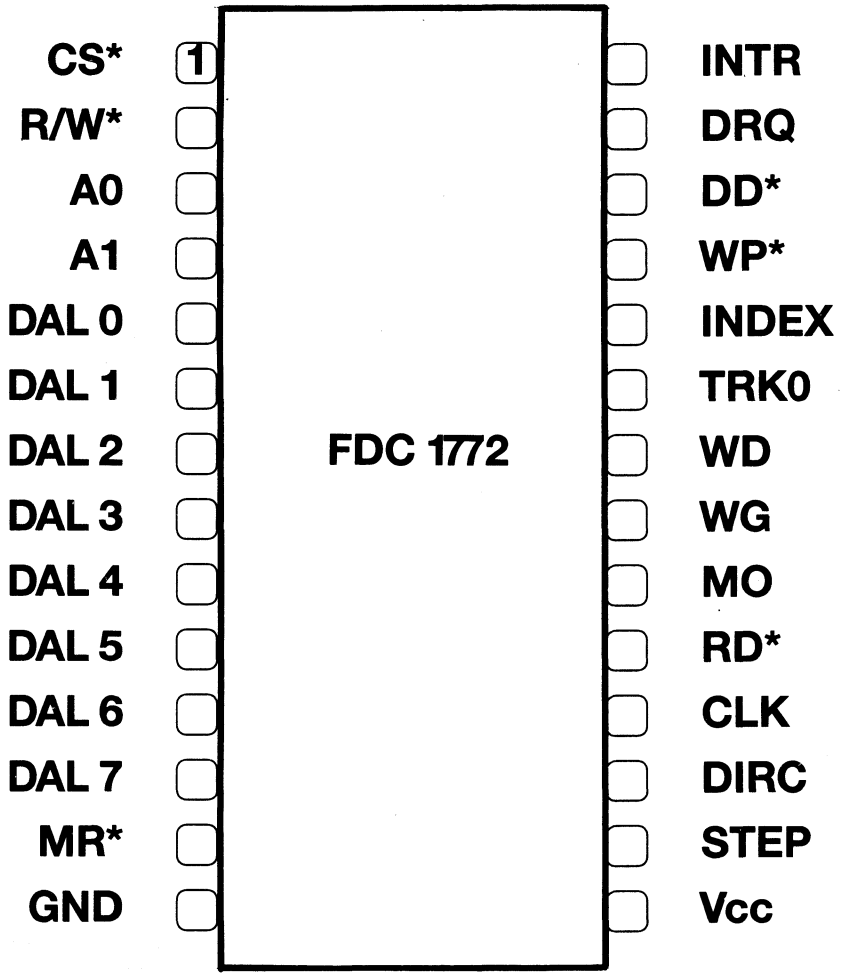
D0-D7 :
Über diesen 8 Bit breiten bidirektionalen Bus werden Daten, Kommandos und Statusinformationen zwischen FDC und Computersystem ausgetauscht.

CS :
Auf die Register des FDC kann nur zugegriffen werden, wenn der CS-Anschluß Low ist.

R/W :
Read/Write. Der Pegel an diesem Anschluß bestimmt die Datenrichtung. Bei einem High können Daten aus dem FDC gelesen, bei einem Low in den FDC geschrieben werden.

A0,A1 :
Mit diesen Bits wird (in Verbindung mit R/W) eingestellt, auf welches Register des 1772 zugegriffen wird. Insgesamt enthält der 1772 fünf Register, die teilweise gelesen und beschrieben werden können. Andere Register lassen sich nur beschreiben oder auslesen. Die folgende Tabelle zeigt die vom Hersteller getroffene Zuordnung:

A1	A0	R/W=1	R/W=0
0	0	Status Reg.	Command Reg.
0	1	Track Reg.	Track Reg.
1	0	Sector Reg.	Sector Reg.
1	1	Data Reg.	Data Reg.


CS*
R/W*
A0
A1
DAL 0
DAL 1
DAL 2
DAL 3
DAL 4
DAL 5
DAL 6
DAL 7
MR*
GND
1
FDC 1772
INTR
DRQ
DD*
WP*
INDEX
TRK0
WD
WG
MO
RD*
CLK
DIRC
STEP
Vcc

DRQ :

Data Request. Dieser Ausgang zeigt mit einem High-Pegel an, daß entweder beim Lesen das Datenregister voll ist, also ausgelesen werden muß, oder daß beim Schreiben das Datenregister leer ist, also neu versorgt werden muß. Mit Hilfe dieses Anschlusses läßt sich ein DMA-Betrieb des FDC erzielen.

INTRQ :

Interrupt Request. Dieser Ausgang wird High, wenn ein Kommando beendet wurde. Zurückgesetzt wird der Ausgang mit dem Lesen des Status-Registers.

CLK :

Clock. Das Clock-Signal zählt nur bedingt zum Prozessor-Bus. An diesem Eingang muß eine Frequenz von 8 MHz anliegen, über die das interne Timing des FDC bestimmt wird.

Die dritte Gruppe von Signalen umfaßt das Floppy-Interface. Über diese Anschlüsse ist eine fast vollständige Steuerung der Laufwerke möglich.

STEP :

Am Step-Ausgang liefert der 1772 einen Impuls für jeden Schritt des Kopfmotors.

DIRC :

Direction. Mit diesem Anschluß wird entschieden, in welche Richtung der Kopf zu bewegen ist. Bei einem High an diesem Pin bewegt sich der Kopf zur Mitte der Diskette.

RD :

Read Data. An RD wird die von der Diskette gelieferte Information angelegt. Diese Information enthält gleichermaßen Takt- wie Datenimpulse, die im internen Datenseparator getrennt und aufbereitet werden.

MO :

Motor On. Über diesen Anschluß werden die Motoren der Laufwerke gesteuert. Die Steuerung geschieht in der Form, das bei Lese-, Schreib- und Such-Operationen die Motoren automatisch gestartet werden.

WG :

Write Gate. WG wird Low, bevor auf die Diskette geschrieben wird. Ohne dieses Signal verriegeln die Laufwerke die Schreiblogik, so daß ein Beschreiben der Diskette unmöglich wird.

WD :

Write Data. Dieser Ausgang liefert den seriellen Datenstrom der zu schreibenden Daten- und Taktimpulse.

TR00 :

Track 00. Mit Hilfe dieses Eingangs kann der Controller ermitteln, ob der Schreib-Lesekopf des selektierten Laufwerks auf der Spur Null platziert ist. In diesem Fall wird TR00 vom Laufwerk auf Low gelegt.

IP :

Index Pulse. Das Index-Loch in der Diskette markiert den physikalischen Beginn eines jeden Tracks. Beim Formatieren der Diskette wartet der FDC zunächst auf den Anfang der Spur, bevor er mit dem Formatieren beginnt.

WPRT :

Write Protect. Bei den im ATARI benutzten 3 1/2"-Disketten wird die Diskette mit Hilfe eines kleinen Schiebers vor unerwünschtem Beschreiben geschützt. Ob die Diskette schreibgeschützt ist wird dem FDC an diesem Eingang mitgeteilt.

DDEN :

Double Density Enable. Dieses Signal paßt nur bedingt in die Gruppe der Floppy-Steuer-Signale. Mit dem Pegel

an diesem Anschluß wird zwischen den Aufzeichnungsformaten Single Density und Double Density umgeschaltet.

1.3.2 Die Register des WD 1772

Bei der Beschreibung der Anschlüsse A1 und A0 wurden die internen Register bereits kurz genannt. Jetzt soll eine genauere Beschreibung der Funktion dieser Register folgen.

CR (Command Register)

In dieses 8-Bit-Register werden die vom FDC auszuführenden Kommandos geschrieben. Allerdings dürfen Kommandos nur dann in das CR geschrieben werden, wenn kein anderes Kommando in Arbeit ist. Obwohl der FDC 'nur' 11 verschiedene Kommandos kennt, ist eine solche Anzahl verschiedenen Möglichkeiten vorhanden, daß die Kommandos im Detail etwas später behandelt werden.
Das CR kann nur beschrieben werden.

STR (Status Register)

Das Status Register gibt Aufschluß über verschiedene Zustände des FDC. Dabei wird die Information in einzelnen Bits codiert. Da die Bedeutungen der Bits vom derzeitigen Kommando abhängen, werden wir sie bei der Kommando-Beschreibung einfügen.
Das Status Register kann nur gelesen werden.

TR (Track Register)

Im TR ist die momentane Position des Schreib/Lese-Kopfes enthalten. Mit jeder Bewegung des Kopfes wird das TR entsprechend der Bewegungsrichtung um eins erhöht oder verringert. Bei einigen Kommandos wird der Inhalt des TR mit den von der Diskette gelesenen Informationen verglichen. Das Ergebnis beeinflußt dann das Status Register.
TR kann sowohl gelesen als auch beschrieben werden.

SR (Sector Register)

Das SR enthält bei Lese- und Schreiboperationen die Nummer des gewünschten Sektors. Wie auch TR kann SR gelesen und beschrieben werden.

DR (Data Register)

Über das Data Register werden die auf die Diskette zu schreibenden Daten an den FDC übergeben. Beim Lesen dagegen kann die Disketteninformation aus dem DR gelesen werden.

1.3.3 Die Programmierung des FDC

Die Programmierung des FDC stellt für den Systemprogrammierer keine große Schwierigkeit dar. Im ST wird die direkte (und sicher in den meißten Fällen überflüssige) Programmierung des FDC durch den Anschluß an den DMA-Baustein teilweise etwas erschwert, teilweise aber auch drastisch vereinfacht.
Die 11 möglichen Kommandos des FDC lassen sich in vier Gruppen oder Typen unterteilen.

Type	Kommando
1	Restore, Spur Null suchen
1	Seek, Spur suchen
1	Step, einen Track in die bisherige Richtung
1	Step In, Kopf einen Track nach innen (zur Disk-Mitte)
1	Step Out, Kopf einen Track nach außen (zum Disk-Rand)
2	Read Sector, Sektor lesen
2	Write Sector, Sektor schreiben
3	Read Adress, ID lesen
3	Read Track, ganzen Track lesen
3	Write Track, ganzen Track schreiben (formatieren)
4	Force Intrrupt, Interrupt auslösen

Die Typ 1 Kommandos

Die Kommandos dieser Gruppe sind für die Positionierung des Schreib/Lesekopfes zuständig. Die Bitmuster dieser fünf Befehle finden Sie in der folgenden Aufstellung:

	Bit							
	7	6	5	4	3	2	1	0
Restore	0	0	0	0	H	V	R1	R0
Seek	0	0	0	1	H	V	R1	R0
Step	0	0	1	U	H	V	R1	R0
Step In	0	1	0	U	H	V	R1	R0
Step Out	0	1	1	U	H	V	R1	R0

Alle fünf Kommandos haben mehrere variable Bits. So wird in den Bits R0 und R1 die Zeit zwischen zwei Stepimpulsen angegeben. Die möglichen Kombinationen sind:

R1	R0	Steprate
0	0	2 Millisekunden
0	1	3 Millisekunden
1	0	5 Millisekunden
1	1	6 Millisekunden

Diese Bits müssen entsprechend den Erfordernissen des Laufwerks im Kommandobyte gesetzt werden.
Das V-Bit in den Kommandos der Gruppe 1 ist das sogenannte Verify-Flag. Wird dieses Flag im Kommandowort gesetzt, so führt der FDC nach einer Kopfbewegung einen automatischen Verify durch, d.h. er vergleicht die im Trackregister gespeicherte Tracknummer mit der im ersten zu lesenden ID-Feld gespeicherten Tracknummer. Stellt der FDC Übereinstimmung fest, so wird das Kommando ohne Fehler beendet.
Das H-Bit bestimmt, ob die sogenannte Spin-Up-Sequenz eingehalten wird. Da in Systemen mit dem 1772 die Laufwerksmotoren direkt vom FDC gesteuert werden, muß nach dem Einschalten eine gewisse Zeit gewartet werden, bis der Motor seine Nennumdrehungszahl von 300 UPM erreicht hat. Ist

im Kommandowort das H-Bit gelöscht, so überprüft der FDC vor Aktivierung des Kommandos den Zustand des Motor-On-Pins. Ist der Motor nicht eingeschaltet, so wird dieser Anschluß auf High gelegt (Motor On) und der FDC wartet insgesamt 6 Umdrehungen (Index Impulse), bevor das Kommando gestartet wird. Läuft der Motor jedoch, so wird die Wartezeit nicht eingelegt, da in diesem Fall die Nenndrehzahl wahrscheinlich erreicht ist.
Bei den drei verschiedenen Step-Kommandos hat das Bit 4 im Kommandowort die Bezeichnung U-Bit. Bei gesetztem U-Bit wird jeder Step, also jeder Wechsel der Kopfposition im Trackregister mitgeführt. Ist das U-Bit gelöscht, so unterbleibt das Update.

Die Typ 2 Kommandos

Die Kommandos dieser Gruppe umfassen das Lesen und Schreiben von Sektoren. Auch bei diesen Kommandos haben einzelne Bits unterschiedliche Bedeutungen:

	Bit							
	7	6	5	4	3	2	1	0
Read Sector	1	0	0	M	H	E	0	0
Write Sector	1	0	1	M	H	E	P	A0

Das H-Bit (Bit 3) in den beiden Befehlen ist das schon bekannte Spin-Up-Bit. Ist das E-Bit gesetzt, so wird vom FDC eine Wartezeit von 30 Millisekunden eingelegt, bevor das Kommando gestartet wird. Diese Wartezeit wird bei einigen Laufwerken benötigt, um ein Schwingen der Kopfmechanik nach einem Spurwechsel ausklingen zu lassen. Ist das E-Bit auf Null, so wird mit dem Kommando unmittelbar begonnen.
Das M-Bit bestimmt, ob ein oder mehrere Sektoren nacheinander gelesen werden sollen. Ist das Bit Null, so wird nur jeweils ein Sektor gelesen oder geschrieben. Multi-Sektor-Lesen mit gesetztem M-Bit läuft so ab, daß der FDC nach jedem Sektor automatisch den Inhalt des Sektorregisters erhöht und den neuen Sektor liest.

Beim Sektorlesen müssen die Bits 0 und 1 gelöscht sein. Beim Schreiben jedoch haben sie eine spezielle Bedeutung. Das als A0-Bit bezeichnete Bit 0 bestimmt, ob ein gelöschtes oder normales Data Adress Mark geschrieben wird. Da die weitaus meisten Betriebssysteme diese Möglichkeit nicht nutzen, wird in der Regel ein normales Data Adress Mark geschrieben.
Das P-Bit (Bit 1) bestimmt, ob die Prekompensation beim Schreiben der Daten ein- oder ausgeschaltet ist. Die Prekompensation wird üblicherweise eingesetzt, um auf den inneren Tracks einer Diskette eine größere Datensicherheit zu erlangen.

Die Typ 3 Kommandos

In dieser Gruppe sind die Kommandos Read Adress, Read Track und Write Track zusammengefaßt.
Read Adress liefert dem Programmierer Informationen aus dem nächsten zu lesenden ID-Feld auf der Diskette. Das ID-Feld beschreibt in vier Bytes die Daten Track, Sektor, Diskettenseite und Sektorlänge eines jeden Sektors.
Read Track liefert alle Bytes, die beim Formatieren auf die Diskette geschrieben wurden, also auch die Daten, die sonst üblicher Weise nicht zugänglich sind, da sie zwischen den Sektoren stehen.
Write Track schließlich formatiert eine Spur zur späteren Datenspeicherung. Die benötigten Bitmuster für die Kommandos lauten:

	Bit							
	7	6	5	4	3	2	1	0
Read Adress	1	1	0	0	H	E	0	0
Read Track	1	1	1	0	H	E	0	0
Write Track	1	1	1	1	H	E	P	0

H- und E-Bit geben wie auch bei den Kommandos der Gruppe 2 an, ob die Spin-Up-Sequenz und die Head-Settle-Time von 30 Millisekunden gewartet werden muß. Das P-Bit beim Formatieren hat die selbe Funktion wie beim Schreiben von Sektoren, bestimmt also über die Precompensation beim

Formatieren.

Das Typ 4 Kommando

Diese Gruppe besteht aus nur einem Befehl, dem Force Interrupt. Dieser Befehl kann als einziger zu jedem Zeitpunkt, also auch während der Bearbeitung eines anderen Kommandos dem FDC übermittelt werden. Sobald der FDC dieses Kommando in einer der möglichen Variationen erhält, wird der laufende Befehl beendet. Das Bitmuster des Force Interrupt lautet:

	Bit							
	7	6	5	4	3	2	1	0
Force Intr.	1	1	0	1	I3	I2	I1	I0

Die Bits I0 bis I3 geben an, welche Bedingungen den Interrupt auslösen sollen. I0 und I1 haben jedoch beim 1772 keine Bedeutung und sollten Low sein. Ist I2 gesetzt, so wird mit jedem Indeximpuls ein Interrupt erzeugt. Mit dieser Option könnte softwaremäßig die Drehzahl der Diskette ermittelt werden. Wird I3 gesetzt, so wird unmittelbar ein Interrupt erzeugt und das laufende Kommando beendet. Sind alle I-Bits Null, so wird der laufende Befehl ohne Interrupt beendet.

1.4 Der MFP 68901

MFP ist die Abkürzung für Multi Function Peripheral. Sie werden zustimmen, daß der Name für dieses IC nicht zu hoch gegriffen ist, wenn Sie mit den Möglichkeiten dieses ICs vertraut sind. In kurzen Stichworten können die folgenden Leistungsmerkmale genannt werden:

- *8-Bit-Parallelport*
- *Datenrichtung jedes Portbits separat programmierbar*
- *Verwendung der Portbits als Interrupteingang möglich*
- *16 mögliche Interruptquellen*
- *Vier universelle Timer*
- *serielle Schnittstelle eingebaut*

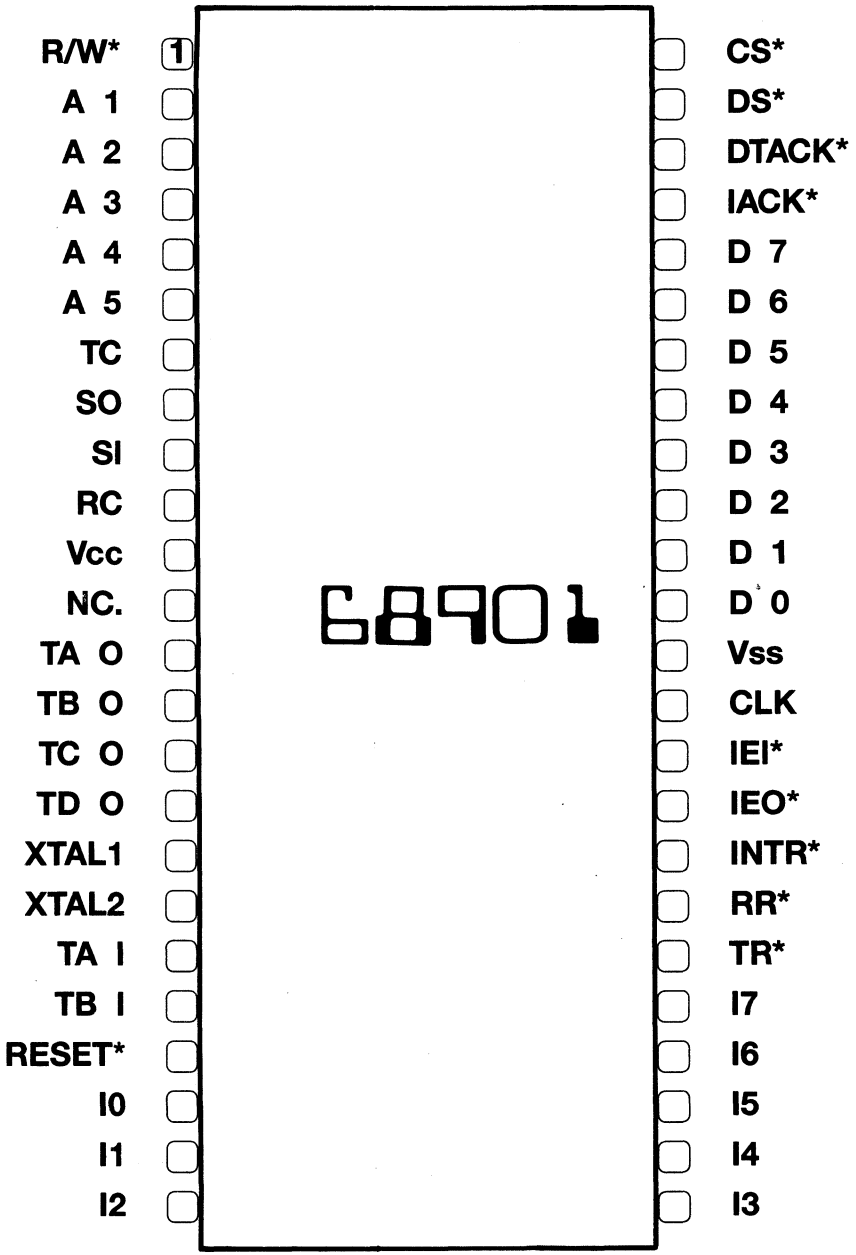
Um alle Möglichkeiten des Chips unterzubringen, wurde ein 48-poliges Gehäuse notwendig. Wir wollen zunächst kurz auf die Bedeutung der verschiedenen Pins eingehen, bevor der interne Registeraufbau besprochen wird.

1.4.1 Die Anschlüsse des 68901

Die Anschlüsse des MFP lassen sich in Funktionsgruppen zusammenfassen.

Die erste Funktionsgruppe besteht aus den Versorgungsanschlüssen.

GND, Vcc, CLK :
Über Vcc und GND wird der MFP mit der Stromversorgung verbunden. CLK ist der Clock-Eingang des MFP. Dieses Clock-Signal muß nicht notwendiger Weise mit dem Systemtakt des Prozessors übereinstimmen. Im ATARI ST wird der MFP mit einer Frequenz von 4 MHz versorgt.


R/W* 1
A 1
A 2
A 3
A 4
A 5
TC
SO
SI
RC
Vcc
NC.
TA O
TB O
TC O
TD O
XTAL1
XTAL2
TA I
TB I
RESET*
I0
I1
I2
68901
CS*
DS*
DTACK*
IACK*
D 7
D 6
D 5
D 4
D 3
D 2
D 1
D 0
Vss
CLK
IEI*
IEO*
INTR*
RR*
TR*
I7
I6
I5
I4
I3

Der Anschluß an den Datenbus des Prozessors wird mit den Pins D0 bis D7, CS, DS, DTACK, RS1 bis RS5 und RESET vorgenommen.

D0-D7 :
Diese bidirektionalen Anschlüsse des 68901 werden normalerweise mit den unteren 8 Datenbits des 68000 verbunden. Zwar wäre auch ein Anschluß an den Datenleitungen D8 bis D15 möglich. Dann aber wäre eine wesentliche Möglichkeit, die Erzeugung von Non-Auto-Interrupts, unmöglich. Bei dieser Interrupt-Methode wird der Interrupt-Vektor auf die unteren acht Datenbits des Busses gelegt.

CS (Chip Select):
CS wird benötigt, um die internen Register des MFP ansprechen zu können. CS ist Low aktiv.

DS (Data Strobe):
Der Anschluß DS wird entweder mit dem Prozessorsignal LDS oder UDS verbunden. Welches Signal verwendet wird, hängt davon ab, ob der MFP auf der oberen oder unteren Hälfte des Datenbusses betrieben wird.

DTACK (Data Transfer Acknowledge):
Dieses Signal zeigt den Abschluß des Buszyklus (lesen oder schreiben) des Prozessors an.

RS1-RS5 (Register Select):
Die RS-Anschlüsse werden normalerweise mit den unteren fünf Adressleitungen des Prozessors verbunden und dienen der Auswahl der 24 internen Register.

RESET :
Wenn der RESET-Anschluß für mindestens 2 Mikrosekunden Low ist, wird der MFP in einen definierten Grundzustand versetzt. RESET wird beim Einschalten des Systems, aber auch bei der Ausführung des RESET-Befehls des Prozessors erzeugt.

Eine weitere wichtige Gruppe von Signalen stellen die Interrupt-Anschlüsse dar. Zu dieser Gruppe sind die Pins IRQ, IACK, IEI und IEO zu zählen.

IRQ (Interrupt ReQuest)
IRQ wird dann Low, wenn ein Ereignis im MFP einen Interrupt ausgelöst hat. Mit diesem Anschluß wird dem Prozessor der Interrupt signalisiert.

IACK (Interrupt ACKnowledge)
Im Falle eines Interrupts (Low an IRQ und IEI) legt der MFP bei Low an IACK und DS einem Interrupt-Vektor auf die Datenleitungen. Jede der 16 möglichen Interruptquellen kann einen eigenen Vektor liefern, so daß die Behandlung des Interrupts deutlich vereinfacht und beschleunigt wird.

IEI, IEO (Interrupt Enable In, Out)
Diese beiden Anschlüsse erlauben die Verkettung von mehreren MFPs. Dabei wird die Priorität der einzelnen MFPs durch die Position in der Kette festgelegt. Bei zwei MFPs wird IEI des höher priorisierten MFP auf Masse gelegt. IEO wird mit dem IEI-Eingang des zweiten 68901 verbunden. Der IEO-Ausgang des zweiten MFP bleibt unbeschaltet. Im Falle eines Interrupts wird der Prozessor das Signal IACK erzeugen. Dieses Signal gelangt an beide MFPs. Wenn der erste MFP der Kette für den Interrupt verantwortlich ist, so wird er seinen IEO-Ausgang auf High legen. Bis zum Beenden der Interruptroutine kann keine neue Unterbrechung durch den zweiten oder weitere MFPs erzeugt werden.

Die nächste zu betrachtende Gruppe sind die acht I/O-Anschlüsse.

IO0-7 (Input Output)
Diese Anschlüsse können zum einen als ganz normale Ein/Ausgabe-Leitungen benutzt werden. Die Datenrichtung

jedes Portbits ist in einem Datenrichtungsregister separat einstellbar. Zusätzlich jedoch kann jedes Portbit als Interrupt-Eingang programmiert werden. Mit einem speziellen Register kann für jedes Bit getrennt die Flanke des auslösenden Signals eingestellt werden.

Eine weitere Gruppe von Anschlüssen stellen die Timer-Pins dar. Auch die Timer Clock-Anschlüsse werden unter dieser Gruppe aufgeführt.

XTAL1,2 (Timer Clock Crystal)
An diesen Anschlüssen kann ein Quarz angeschlossen werden, der die Arbeitsfrequenz der vier Timer liefert.

TAI,TBI (Timer Input)
Die Timer A und B können anders als die Timer C und D nicht nur als reine Zähler der an XTAL1 und 2 angelegten Timerfrequenz benutzt werden, sondern können auch als Ereigniszähler und zur Impulsbreitenmessung eingesetzt werden. In diesen beiden Fällen muß ein externes Signal an die entsprechenden Timer-Eingänge angelegt werden.

TAO,TBO,TCO,TDO (Timer Output)
Jeder Timer verfügt über einen Ausgang, dessen Pegel wechselt, wenn der Timer seinen Zählerstand von 01 nach 00 wechselt. Unabhängig von der eingestellten Betriebsart arbeiten die Zähler grundsätzlich als Abwärtszähler. Ihr Inhalt wird mit jedem Impuls um Eins verringert.

Die vorletzte Gruppe von Signalen beinhaltet die Anschlüsse der universellen seriellen Schnittstelle. Die im MFP eingebaute Vollduplex fähige Schnittstelle kann wahlweise asynchron oder synchron und mit unterschiedlichen Sende- und Empfangsbaudraten arbeiten.

SI (Serial Input)
An den SI-Eingang wird ein zu empfangener serieller Bitstrom angelegt.

SO (Serial Output)
Entsprechend SI ist an diesem Ausgang der Bitstrom der zu sendenden Daten verfügbar.

RC (Receiver Clock)
Die Übertragungsgeschwindigkeit der Empfangsdaten wird durch die Frequenz an diesem Eingang bestimmt. Als Quelle dieses Signals kann z.B. einer der eingebauten vier Timer genutzt werden.

TC (Transmitter Clock)
Entsprechend zu RC bestimmt die Frequenz dieses Signals die Baudrate der zu sendenden Daten.

Die letzte Gruppe von Signalen ist im ATARI nicht verwendet. Diese Signale werden benötigt, wenn die serielle Schnittstelle mittels DMA betrieben wird.

RR (Receiver Ready)
Dieser Anschluß gibt Aufschluß über den Zustand des Empfänger-Datenregisters. Ist ein Zeichen vollständig empfangen, so wird dieser Anschluß auf Masse gelegt. Dadurch wird dem DMA-Controller mitgeteilt, daß ein Zeichen vom MFP abgeholt werden muß.

TR (Transmitter Ready)
Dieser Anschluß übernimmt eine ähnliche Funktion für den Sendeteil der seriellen Schnittstelle. Mit einem Low an diesem Pin wird dem DMA-Controller mitgeteilt, daß ein neues Zeichen in den MFP übertragen werden muß.

1.4.2 Die Register des MFP

Wie bereits erwähnt enthält der 68901 insgesamt 24 verschiedene Register. Die hohe Anzahl und die logische Anordnung der Register erleichtert die Programmierung des MFP sehr.

Reg 1 GPIP, General Purpose I/O Interrupt Port
Dieses Register ist das Datenregister des 8-Bit-Ports. Mittels dieser Adresse werden die Daten an die Portbits ausgegeben bzw. eingelesen.

Reg 2 AER, Active Edge Register
Wenn Portbits als Interrupteingang benutzt werden, so kann mit diesem Register festgelegt werden, ob derInterrupt bei einem Low-High- oder High-Low-Wechsel ausgelöst wird. Dabei bedeutet eine Null in diesem Register einen Interrupt bei High-Low-Wechsel, eine Eins dagegen Interrupt bei Low-High-Wechsel.

Reg 3 DDR, Data Direction Register
Wie bereits erwähnt kann die Datenrichtung eines jeden Portbits individuell eingestellt werden. Wird in ein DDR-Bit eine Null geschrieben, so arbeitet der entsprechende Anschluß als Eingang, eine Eins in dem DDR-Bit programmiert das Portbit als Ausgang.
Portbits werden durch AER- und DDR-Bit mit der selben Position beeinflußt.

Reg 4,5 IERA, IERB, Interrupt Enable Register
Jede Interrupt-Quelle des MFP läßt sich separat ein- und ausschalten. Da insgesamt 16 verschiedene Interrupt-Quellen oder -Kanäle vorhanden sind, mußten zwei 8-Bit-Register verwendet werden. Wird eine Eins in ein Bit von IERA oder IERB geschrieben, so ist der entsprechende Kanal enabled, also eingeschaltet. Eine Null dagegen disabled den Kanal. Tritt an einem ausgeschalteten Kanal das den Interrupt auslösende Ereignis ein, so wird es vom MFP vollständig ignoriert.

Die folgende Tabelle zeigt, welches Bit welchem Interruptereignis zugeordnet ist:

```
IERA
Bit 7 : IO-Port Bit 7, höchste Priorität
Bit 6 : IO-Port Bit 6
Bit 5 : Timer A
Bit 4 : Empfänger Buffer voll
Bit 3 : Empfänger Fehler
Bit 2 : Sender Buffer leer
Bit 1 : Sender Fehler
Bit 0 : Timer B

IERB
Bit 7 : IO-Port Bit 5
Bit 6 : IO-Port Bit 4
Bit 5 : Timer C
Bit 4 : Timer D
Bit 3 : IO-Port Bit 3
Bit 2 : IO-Port Bit 2
Bit 1 : IO-Port Bit 1
Bit 0 : IO-Port Bit 0, niedrigste Priorität
```

Diese Zuordnung ist auch für die im Folgenden beschriebenen IP-, IM- und IS-Register gültig.

Reg 6, 7 IPRA, IPRB, Interrupt Pending Register

Sobald ein Interrupt eines eingeschalteten Kanals auftritt, wird das dem Kanal zugeordnete Bit im Interrupt Pending Register auf Eins gesetzt. Wird im System von der Möglichkeit der Vektor-Erzeugung Gebrauch gemacht, so wird das Bit automatisch wieder gelöscht, wenn der MFP die Vektornummer auf den Datenbus legt. Wird von dieser Möglichkeit jedoch kein Gebrauch gemacht, so muß das IPR softwaremäßig gelöscht werden. Um ein Bit im IPR zu löschen, wird ein Byte an den MFP geschickt, in dem alle bis auf das zu löschende Bit gesetzt sind.

Die Zuordnung der Bits in den IPR entspricht der unter

IER (Register 4 und 5) beschriebenen Zuordnung.

Reg 8, 9 ISRA, ISRB, Interrupt In-Service Register

Die Funktion dieser Register ist etwas komplex und hängt von Bit 3 des Registers 12 ab. Dieses als S-Bit bezeichnete Bit bestimmt, ob der 68901 im sogenannten 'Software End-of-Interrupt'-Modus (SEI) oder im 'Automatic End-of-Interrupt'-Modus (AEI) arbeitet. Im AEI-Modus wird das Interrupt Pending Bit (IPR) gelöscht, wenn der Prozessor während eines IACK-Zyklus die Vektornummer vom MFP holt. Auch das zugehörige Interrupt In-Service Bit wird zu diesem Zeitpunkt gelöscht. Danach können neu eintreffende Ereignisse einen Interrupt auslösen, auch wenn die Interrupt-Routine des ersten Interrupts noch nicht beendet ist.

Im SEI-Modus dagegen wird das korrespondierende ISR-Bit gesetzt, wenn die Vektornummer des Interrupts von Prozessor angefordert wird. Am Ende der Interruptroutine muß das Bit gelöscht werden, in dem ein Byte an den MFP geschickt wird, in dem alle bis auf das zu löschende Bit gesetzt sind. Solange das Interrupt In-Service Bit gesetzt ist, sind alle Interrupts niedrigerer Priorität im MFP ausmaskiert. Da das Pending-Bit des aktiven Kanals mit Übergabe der Vektornummer gelöscht wurde, kann sogar ein gleichartiger Interrupt ein zweites Mal auftreten. Allerdings werden dieser und eventuelle niedrigere Interrupts erst bearbeitet (lösen einen IRQ aus), wenn das gesetzte In-Service Bit gelöscht wird.

Reg 10,11 IMRA, IMRB, Interrupt Mask Register

Mit Hilfe dieser Register können einzelne, in den IER eingeschaltete Interruptquellen ausmaskiert werden. Das bedeutet, daß die Interruptereignisse zwar intern erkannt und im IPR signalisiert werden, die IRQ-Leitung jedoch auf High-Pegel bleibt.

Reg 12 VR, Vector Register

Im Falle eines Interrupts kann der 68901 auf Anforderung des Prozessors im Interrupt Acknowledge Cycle eine der Interruptquelle entsprechende Vektornummer generieren. Jeder der 16 Interrupt Kanäle verfügt über seinen eigenen Vektor. Die 16 Kanäle werden entsprechend der Priorität in den unteren vier Bits der Vektornummer codiert, die oberen vier Bits des Vektors werden aus dem Vektor Register kopiert. Dazu müssen diese Bits im VR entsprechend gesetzt werden.
Das Bit 3 des VR ist das schon erwähnte S-Bit. Ist dieses Bit gesetzt (wie im ST), so arbeitet der MFP im 'Software End-of-Interrupt'-Modus, bei gelöschtem Bit ist der Modus 'Automatic End-of-Interrupt'

Reg 13,14 TACR, TBCR, Timer A/B Control Register

Bevor wir auf die Programmierung dieser Register eingehen, müssen wir einige Worte über die Möglichkeiten der Timer verlieren.
Die Timer A und B verfügen über identische Möglichkeiten. Jeder Timer besteht aus einem Datenregister, einem programmierbaren Vorteiler und einem 8-Bit-Abwärtszähler. Der Inhalt des Zählers wird bei jedem vom Vorteiler gelieferten Impuls um Eins verringert. Steht der Zähler auf dem Wert 01, so wechselt beim nächsten Impuls der dem entsprechenden Timer zugeordnete Ausgang seinen Pegel. Gleichzeitig wird der Timer mit dem Wert des Timer-Datenregisters geladen. Ist bei diesem Kanal das IER-Bit gesetzt, so wird ein Interrupt ausgelöst.
Als Quelle des Zählertakts wird üblicherweise die an XTAL1, 2 anliegende Quarzfrequenz verwendet. Diese Betriebsart wird als Delay-Mode (Verzögerungsmodus) bezeichnet und steht auch bei den Timern C und D zur Verfügung.
Bei den Timern A und B jedoch können auch externe Impulse an den Timereingängen TAI und TBI gezählt werden. In diesem Fall spricht man vom Event Count-Mode. Beim Zählen externer Impulse wird der

Vorteiler ausgeschaltet, um jeden anliegenden Impuls zählen zu können. Die maximale Zählfrequenz an den Timereingängen darf ein Viertel der Betriebsfrequenz des MFP, im ST also 1 MHz, nicht überschreiten.
Eine weitere Besonderheit dieser Betriebsart ist die Tatsache, daß die Timer-Eingänge den Interrupts der I/O-Anschlüsse I3 und I4 zugeordnet sind. Je nach Programmierung der entsprechenden Bits im Active Edge Register kann durch einen Pegelsprung an den Timereingängen ein Interrupt ausgelöst werden. TAI ist dabei I4, TBI dem Anschluß I3 zugeordnet. I3 und I4 können jedoch weiterhin als I/O-Anschlüsse ohne Interrupt-Möglichkeit genutzt werden.
Timer A und B kennen noch eine dritte Betriebsart, die Pulsweitenmessung. Diese Betriebsart ähnelt dem Delay-Mode mit dem Unterschied, daß mittels der Timereingänge TAI und TBI die Timer ein- und ausgeschaltet werden können. Auch bei dieser Betriebsart werden die AER-Bits von I3 und I4 benutzt, um festzulegen, ob die Timer bei einem High oder einem Low an den Timereingängen arbeiten dürfen. Ist z.B. AER-Bit 4 gesetzt, so arbeitet der Zähler, wenn TAI High ist. Wechselt TAI von High nach Low, so wird ein Interrupt erzeugt, vorausgesetzt, der Interrupt des I4-Kanals ist zugelassen.
Doch kommen wir nun zur Programmierung von TACR und TBCR. In beiden Registern werden nur fünf der acht möglichen Bits genutzt. Die Bits 0 bis 3 bestimmen die Betriebsart des entsprechenden Timers nach folgender Tabelle:

```
Bit 3 2 1 0   Funktion

    0 0 0 0   Timer Stop, keine Funktion ausführen
    0 0 0 1   Delay Modus, Vorteiler teilt durch 4
    0 0 1 0   Delay Modus, Vorteiler teilt durch 10
    0 0 1 1   Delay Modus, Vorteiler teilt durch 16
    0 1 0 0   Delay Modus, Vorteiler teilt durch 50
    0 1 0 1   Delay Modus, Vorteiler teilt durch 64
```

```
0 1 1 0   Delay Modus, Vorteiler teilt durch 100
0 1 1 1   Delay Modus, Vorteiler teilt durch 200
1 0 0 0   Event Count Mode
1 0 0 1   Pulsweiten-Mode, Vorteiler teilt durch 4
1 0 1 0   Pulsweiten-Mode, Vorteiler teilt durch 10
1 0 1 1   Pulsweiten-Mode, Vorteiler teilt durch 16
1 1 0 0   Pulsweiten-Mode, Vorteiler teilt durch 50
1 1 0 1   Pulsweiten-Mode, Vorteiler teilt durch 64
1 1 1 0   Pulsweiten-Mode, Vorteiler teilt durch 100
1 1 1 1   Pulsweiten-Mode, Vorteiler teilt durch 200
```

Die Bits 4 der Timer Control Register haben eine besondere Funktion. Mit diesen Bits können die Timer Ausgänge der zugehörigen Timer zu jedem Zeitpunkt auf Low gelegt werden. Im Anschluß an die Operation kann der Ausgang jedoch unmittelbar wieder High werden, wenn gerade zu diesem Zeitpunkt der Timer abläuft.

Reg 15 TCDCR Timers C and D Control Register

Die Timer C und D verfügen nur über den Delay-Mode. Deshalb reicht ein Byte zur Steuerung der beiden Timer. Die Steuerinformation wird in den unteren drei Bits der Nibbles (Vier-Bit-Hälften) programmiert. Die Bits 0 bis 2 sind dem Timer D zugeordnet, Timer C wird mit den Bits 4 bis 6 beeinflußt.

```
Bit 2 1 0   Funktion Timer D
Bit 6 5 4   Funktion Timer C

  0 0 0   Timer Stop, keine Funktion ausführen
  0 0 1   Delay Modus, Vorteiler teilt durch 4
  0 1 0   Delay Modus, Vorteiler teilt durch 10
  0 1 1   Delay Modus, Vorteiler teilt durch 16
  1 0 0   Delay Modus, Vorteiler teilt durch 50
  1 0 1   Delay Modus, Vorteiler teilt durch 64
  1 1 0   Delay Modus, Vorteiler teilt durch 100
  1 1 1   Delay Modus, Vorteiler teilt durch 200
```

Die Bits 3 und 7 in diesem Register haben keine

Funktion.

Reg 16 - 19 TADR, TBDR, TCDR, TDDR, Timer Data Register
Die vier Timer-Datenregister werden mit dem Wert programmiert, der in den Zähler geladen wird, wenn er nach Erreichen des Zählerstandes 01 einen Impuls bekommt. Von diesem Wert wird dann kontinuierlich abwärts gezählt.

Reg 20 SCR, Syncronous Character Register
Bei der synchronen Datenübertragung wird in dieses Register ein Wert geschrieben, der beim Empfang den Beginn der Übertragung kennzeichnet. Ist der Synchron-Mode ausgewählt, so werden alle empfangenen Zeichen mit dem im SCR gespeicherten Wert verglichen. Erst bei Übereinstimmung werden die nachfolgenden Zeichen in den Empfänger-Buffer übertragen.

Reg 21 UCR, USART Control Register
USART ist die Abkürzung für Universal Synchronous/Asynchronous Receiver/Transmitter. Mit dem UCR werden bis auf die Baudrate alle Betriebsparameter der Schnittstelle eingestellt. Wie auch bei den Timern werden verschiedene Parameter in einzelnen Bits codiert. Die Bedeutung ist in der folgenden Tabelle aufgeschlüsselt:

```
Bit 0     : nicht benutzt
Bit 1     : 0 = Parity Odd
            1 = Parity Even

Bit 2     : 0 = keine Parity (Bit 1 wird ignoriert)
            1 = Parity entsprechend Bit 1

Bit 3,4   : diese Bits bestimmen Anzahl von Start-
            und Stopbits und das gewünschte Format.
```

Bit 4	3	Start	Stop	Format
0	0	0	0	Synchron
0	1	1	1	Asynchron
1	0	1	1,5	Asynchron
1	1	1	2	Asynchron

Bit 5,6 : Diese Bits geben die Anzahl der zu übertragenden Datenbits an.

Bit 6	5	Wortlänge
0	0	8 Bits
0	1	7 Bits
1	0	6 Bits
1	1	5 Bits

Bit 7 : 0 = Frequenz an TC und RC wird direkt als Übertragungsfrequenz benutzt (nur für synchrone Übertragung zu verwenden)
1 = Frequenz an TC und RC wird intern durch 16 geteilt.

Reg 22 RSR, Receiver Status Register

Das RSR gibt Aufschluß über alle den Empfänger betreffenden Zustände. Wieder sind verschiedene Zustände in einzelnen Bits codiert.

Bit 0 Receiver Enable Bit.
Wird dieses Bit gelöscht, so wird sofort der Empfangsteil ausgeschaltet. Alle Flags im RSR werden automatisch gelöscht. Bei gesetztem Bit arbeitet der Empfänger normal.

Bit 1 Syncronous Strip Enable.
Mit diesem Bit kann bei synchronem Datentransfer bestimmt werden, ob ein mit dem im SCR gespeichertes identisches Zeichen in den Receiver Buffer übertragen wird oder nicht.

Bit 2 Match/Character In Progress.
Im synchronen Übertragungsformat signalisiert dieses Bit, daß ein mit dem SCR-Byte identisches Zeichen empfangen wurde. In der asynchronen Betriebsart wird dieses Bit gesetzt, sobald das Startbit erkannt wurde. Mit Erkennen des Stopbits wird das Flag automatisch wieder gelöscht.

Bit 3 Found - Search/Break Detected.
Auch Bit 3 des RSR hat je nach Format unterschiedliche Aufgaben. Im synchronen Übertragungsformat wird das Bit gesetzt, wenn ein empfangenes Zeichen mit dem im SCR gespeicherten übereinstimmt. Diese Bedingung kann über den Receiver Error Kanal einen Interrupt auslösen. Im asynchronen Betrieb wird das Bit gesetzt, wenn ein BREAK empfangen wurde. Die Break-Bedingung wird erfüllt, wenn nach dem Startbit nur Nullen empfangen werden. Zur Unterscheidung des BREAK von einer 'echten' Null muß der Pegel der Empfängerleitung auch zur Zeit des Stop-Bits Low bleiben.

Bit 4 Frame Error.
Ein Frame Error tritt auf, wenn ein Byte empfangen wurde, das nicht Null ist, dessen Stopbit jedoch Null ist.

Bit 5 Parity Error.
Der Zustand dieses Bits gibt Aufschluß darüber, ob das letzte empfangene Zeichen den Anforderungen an die Parität gerecht wurde. Ist die Paritätsprüfung beim Empfang ausgeschaltet, so wird das PE-Bit nicht beeinflußt.

Bit 6 Overrun Error.
Dieses Bit wird gesetzt, wenn ein vollständiges Zeichen im Empfangsschieberegister zusammengesetzt worden ist, das zuvor empfangene Zeichen jedoch noch nicht aus dem Empfänger Buffer ausgelesen

wurde. Dieser Fehler kann einen Interrupt auslösen.

Bit 7 Buffer Full.
Dieses Bit wird gesetzt, wenn ein vollständiges Zeichen in den Empfänger Buffer aus dem Schieberegister übertragen wird. Sobald der Prozessor das Byte ausliest, wird das Bit gelöscht.

Reg 23 TSR, Transmitter Status Register
Wie das RSR Informationen über den Empfänger liefert, so ist das TSR für den Sender zuständig.

Bit 0 Transmitter Enable.
Wird dies Bit gelöscht, so ist der Sendeteil vollständig ausgeschaltet. Gleichzeitig wird das End-Bit gelöscht und das UE-Bit gesetzt (UE- und End-Bit siehe unten). Der Ausgang des Empfängers wird entsprechend H- und L-Bit eingestellt.

Bit 1,2 High- und Low-Bit.
Mit diesen Bits kann der Programmierer beeinflussen, welchen Zustand der Ausgang des ausgeschalteten Transmitters hat. Sind beide Bits gelöscht, so ist der Ausgang hochohmig. Ist nur das L-Bit gesetzt, so ist der Ausgang Low. Ist nur das High-Bit gesetzt, so ist der Ausgang High. Sind jedoch beide Bits gesetzt, so ist der sogenannte Loop-Back-Modus eingeschaltet. In diesem Zustand wird der Ausgang des Transmitters intern mit dem Eingang des Empfängers verbunden. Der Ausgang selbst liegt dann auf High-Pegel.

Bit 3 Break.
Bei der synchronen Datenübertragung hat das Break-Bit keine Funktion. In der asynchronen Betriebsart jedoch wird, sobald das Bit gesetzt ist, die Break-Bedingung ausgesendet.

Bit 4 End of Transmittion.
Wird der Sendeteil während der laufenden Übertragung eines Zeichens ausgeschaltet, so wird das End-Bit gesetzt, sobald das Zeichen vollständig gesendet wurde. Wird jedoch zum Zeitpunkt des Abschaltens kein Zeichen übertragen, so wird das End-Bit unmittelbar gesetzt.

Bit 5 Auto Turnaround.
Ist dieses Bit gesetzt, so wird der Empfänger automatisch eingeschaltet, wenn der Sender ausgeschaltet und ein eventuelles letztes Zeichen vollständig gesendet wurde.

Bit 6 Underrun Error.
Dieses Bit wird gesetzt, wenn das im Sendeschieberegister enthaltene Zeichen vollständig gesendet wurde, bevor ein neues Zeichen in den Sendebuffer geschrieben wurde.

Bit 7 Buffer Empty.
Dieses Bit wird gesetzt, wenn ein Zeichen aus dem Sendebuffer in das Schieberegister übertragen wurde. Das Bit wird automatisch gelöscht, wenn neue Daten in den Sendebuffer geschrieben werden.

Reg24 UDR, USART Data Register
Über dieses Register werden die Sende- und Empfangsdaten übertragen. Beim Hineinschreiben gelangen die Daten in den Sendebuffer, beim Lesen des Registers erhält man den Inhalt des Empfängerbuffers.

1.5 Die ACIAs 6850

ACIA ist die Abkürzung für 'Asyncronous Communications Interface Adapter'. Dieses 24-polige IC beinhaltet alle zum Betrieb einer seriellen Schnittstelle benötigten Stufen. Auch Fehlererkennung und Datenformatierung werden von diesem Baustein übernommen. Ursprünglich für 6800-Computersysteme entworfen, kann dies IC aber auch in 65xx- und 680xx-Systemen recht einfach eingesetzt werden. Im ATARI ST übernehmen gleich zwei dieser Bausteine wichtige Aufgaben. Ein 6850 wird für die Kommunikation mit der Tastatatur eingesetzt. Die Tastatur des ST verfügt über einen eigenen Prozessor, der die Tastatur-, Joystick- und Maus-Abfrage sowie das Führen der Uhrzeit übernimmt. Die Daten der Tastatur werden über eine serielle Schnittstelle in das eigentliche ST-System mit dem 68000 übertragen. Die zweite ACIA ist für den Betrieb der MIDI-Schnittstelle zuständig.
Bei der Tastatur-ACIA ist eine Änderung der Betriebsparameter nicht empfehlenswert. Zu schnell könnte man die Verbindung zwischen ST und Tastatur stören. Mit der MIDI-ACIA jedoch kann sicher noch die eine oder andere sinnvolle Applikation geschaffen werden. Schließlich steht ja nirgends geschrieben, das die MID-Anschlüsse nicht auch für andere Zwecke gebraucht werden können. Eine Idee, die so bei Betrachtung der Schnittstelle aufkommt, ist die Vernetzung mehrerer STs zu einem Rechner-Verbund über die MIDI-Schnittstelle.

1.5.1 Die Anschlüsse des 6850

Lesern, die sich bisher nicht mit dem Prinzip der seriellen Datenübertragung beschäftigt haben, soll an dieser Stelle die Bedeutung der einzelnen Anschlüsse des 6850 erläutert werden.

Vss

Dieser Anschluß ist der Masseanschluß der Stromversorgung des ICs.

RX Data Receive DATA

An diesen Anschluß wird der zu empfangene Bitstrom angelegt. Dabei muß zuerst (nach dem Startbit) das niederwertigste Datenbit ankommen.

RX CLK Receive Clock

Das Signal an diesem Anschluß bestimmt die Baudrate, also die Übertragungsgeschwindigkeit beim Empfang von Daten. Mit dem Signal dieses Anschlusses werden die hereinkommenden Daten synchonisiert. Die Frequenz von RX CLK richtet sich nach der gewünschten Übertragungsgeschwindigkeit und nach dem intern programmierten Teilerverhältnis.

TX CLK Transmitter Clock

Wie auch bei RX CLK bestimmt die Frequenz dieses Signals die Übertragungsgeschwindigkeit, hier jedoch beim Senden.

RTS Request To Send

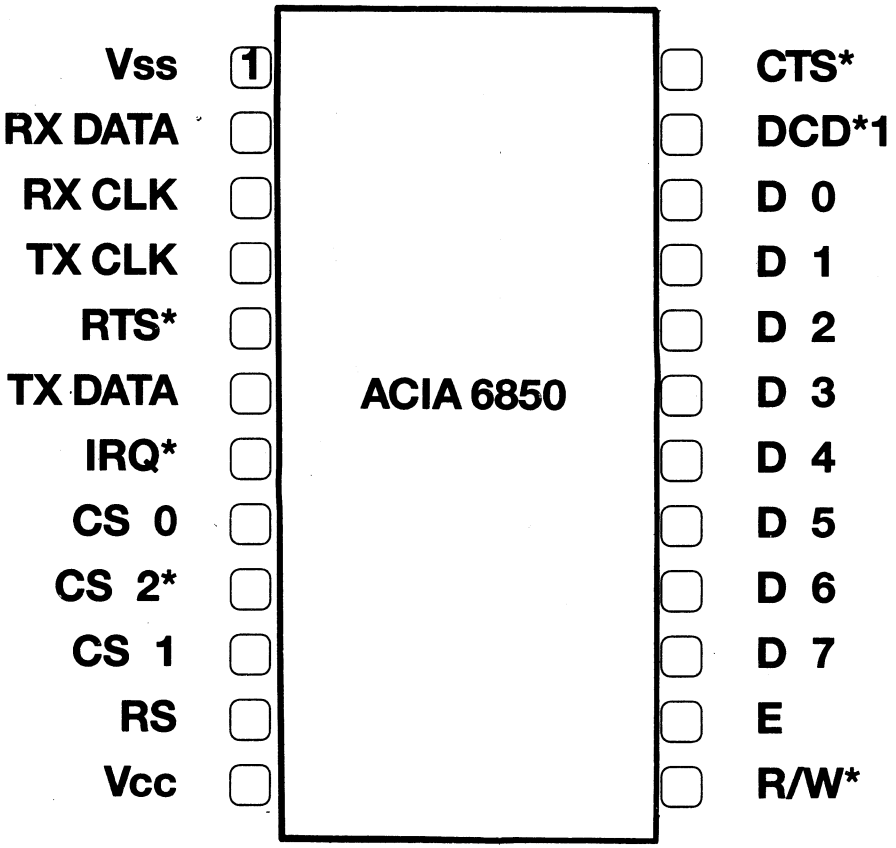
Dieser Ausgang des 6850 läßt sich gezielt vom Prozessor auf Low oder High setzen. Häufig wird dieser Anschluß zur Steuerung des Datentransfers benutzt. Mit einem Low an diesem Ausgang wird z.B. einem Modem signalisiert, daß der Rechner Daten senden will.

TX Data Transmitter Data

Über diesen Anschluß werden zu übertragende Daten bitweise, also seriell aus dem Computer 'herausgeschoben'.

IRQ Interrupt ReQuest

Bei verschiedenen Zuständen der ACIA wird IRQ Low, um dem Prozessor zu signalisieren, daß seine Aufmerksamkeit benötigt wird. Mögliche Zustände sind vollständiger Empfang oder vollständiges Senden eines Zeichens.


Vss
RX DATA
RX CLK
TX CLK
RTS*
TX DATA
IRQ*
CS 0
CS 2*
CS 1
RS
Vcc
ACIA 6850
CTS*
DCD*1
D 0
D 1
D 2
D 3
D 4
D 5
D 6
D 7
E
R/W*

CS 0,1,2 Chip Select

Diese drei Anschlüsse werden zur Selection des der ACIA in einem Computersystem benötigt. Durch die relativ hohe Anzahl von drei CS-Signalen vereinfacht sich in kleineren Systemen oft die zur Adressdecodierung benötigte Hardware.

RS Register Select

Über diesen Anschluß wird in Verbindung mit dem Signal R/W das gewünschte interne Register angesprochen. Die Auswahl und Programmierung der Register wird weiter unten beschrieben.

Vcc

Dieser Anschluß ist der für alle ICs obligatorische Stromversorgungsanschluß. Die Betriebsspannung beträgt die üblichen 5 Volt Gleichspannung.

R/W Read/Write

Das Signal am R/W-Eingang beschreibt die vom Prozessor gewünschte Datenrichtung bei der Kommunikation mit der ACIA. Bei einem High-Signal liest der Prozessor Daten, bei Low-Pegel schreibt er Daten in den 6850.

E Enable

Das E-Signal bestimmt den Zeitpunkt des Lesens oder Schreibens der Daten. Da alle Schreib-/Lesevorgänge mit diesem Signal synchron ablaufen müssen, spricht man bei 68xxx vom synchronen Busanschluß.

D0 - D7

Die Datenanschlüsse des 6850 werden mit dem Datenbus des Prozessors verbunden. Bis auf die Zeiten des Zugriffs auf die ACIA sind diese bidirektionalen Anschlüsse hochohmig.

DCD Data Carrier Detect

DCD stellt ein Signal zur Modemsteuerung dar. Über diesen Anschluß teilt das Modem dem Rechner mit, daß

Daten empfangen werden sollen. Liegt der Eingang DCD auf High, so ist ein Empfang von seriellen Daten unmöglich.

CTS Clear To Send
Das Signal CTS antwortet dem Rechner auf das Signal RTS. Das Senden von Daten ist nur möglich, wenn dieser Anschluß auf Low-Pegel liegt.

1.5.2 Die Register des 6850

Insgesamt enthält der 6850 vier verschiedene Register. Zwei Register können nur gelesen, die anderen beiden nur beschrieben werden. Die Unterscheidung der Register erfolgt mit den beiden Signalen R/W und RS nach folgender Tabelle:

R/W	RS	Register	Zugriff
0	0	Control Register	schreiben
0	1	Senderegister	schreiben
1	0	Status Register	lesen
1	1	Empfangsregister	lesen

Die Sende- und Empfangsregister (auch RX- und TX-Buffer genannt) sind für den Transfer der Daten zuständig. Wenn der Empfang von Daten möglich ist, so gelangen die seriell ankommenden Bits in ein Schieberegister. Sobald die spezifizierte Anzahl Bits angekommen ist, wird der Inhalt des Schieberegisters in den TX Buffer übertragen. Ähnlich läuft der Vorgang beim Senden ab. Hier wird das zu übertragende Byte zunächst in das Senderegister geschrieben, aus dem es automatisch in das Sende-Schieberegister übertragen wird.

Das Control-Register

Das Control-Register mit seinen acht Bits steuert die internen Vorgänge. Um die Steuerung der diversen Funktionen mit einem Byte zu lösen ist, wurde einzelnen Bits eine

bestimmte Bedeutung zugeordnet.

CR 0,1

Diese Bits bestimmen, durch welchen Faktor die Transmitter- und Receiver-Clock-Frequenz geteilt wird. Zusätzlich ist mit diesen Bits eine Master Reset-Funktion verbunden. Da der 6850 über keinen separaten Reset-Anschluß verfügt, muß vor dem ersten Gebrauch der ACIA der Master Reset programmtechnisch durchgeführt werden. Folgende Möglichkeiten bieten diese Bits:

CR1	CR0	
0	0	RXCLK/TXCLK ohne Teilung
0	1	RXCLK/TXCLK durch 16 (für Midi)
1	0	RXCLK/TXCLK durch 64 (für Tastatur)
1	1	Master Reset

CR 2,3,4

Diese sogenannten Word Select Bits geben an, ob 7 oder 8 Datenbits und ein oder zwei Stopbits übertragen werden. Auch die Art der Paritätsprüfung wird in diesen Bits angegeben.

CR4	CR3	CR2	
0	0	0	7 Datenbits, 2 Stopbits, Parity even
0	0	1	7 Datenbits, 2 Stopbits, Parity odd
0	1	0	7 Datenbits, 1 Stopbit, Parity even
0	1	1	7 Datenbits, 1 Stopbit, Parity odd
1	0	0	8 Datenbits, 2 Stopbit, keine Parity
1	0	1	8 Datenbits, 1 Stopbit, keine Parity
1	1	0	8 Datenbits, 1 Stopbit, Parity even
1	1	1	8 Datenbits, 1 Stopbit, Parity odd

CR 6,5

Diese Transmitter Control Bits bestimmen den Pegel am RTS-Ausgang und erlauben oder verbieten einen Interrupt durch die ACIA, wenn das Sende-Register geleert ist. Zusätzlich kann das Senden eines sogenannten

BREAK-Signals über den seriellen Ausgang veranlaßt werden. Das BREAK-Signal besteht aus eine langen Folge von Null-Bits.

CR6	CR5	
0	0	RTS Low, Transmitter IRQ disabled
0	1	RTS Low, Transmitter IRQ enabled
1	0	RTS High, Transmitter IRQ disabled
1	1	RTS Low, Transmitter IRQ disabled, BREAK senden

CR 7

Das Receiver Interrupt Enable Bit bestimmt, ob durch den Empfänger Interrupts ausgelöst werden können. Ursache für einen Interrupt kann ein Wechsel des Pegels an der DCD-Leitung von Low nach High sein. Auch wenn der Empfänger-Datenbuffer voll ist, kann ein IRQ ausgelöst werden. Außerdem kann ein Interrupt im Falle eines OVERRUN ausgelöst werden. Dieser Zustand tritt ein, wenn ein empfan genes Zeichen nicht rechtzeitig vom Prozessor gelesen wurde, und ein vollständiges neues Zeichen von der ACIA in das interne Receiver Datenregister übertragen wurde.

CR7	
0	Interrupts disabled (ausgeschaltet)
1	Interrupts enabled (eingeschaltet)

Das Statusregister

Das Statusregister der ACIA gibt Aufschluß über bestimmte Zustände des Bausteins. Auch bei diesem Byte sind die Informationen in einzelen Bits codiert.

SR 0

Bei einem High dieses Bits ist das RX Datenregister voll. Um einem OVERRUN zuvor zu kommen, muß das Byte gelesen werden, bevor ein neues Zeichen vollständig empfangen wird.

SR 1

Dieses Bit reflektiert den Zustand des TX Datenbuffers. Ist das Register leer, so ist das Bit gesetzt.

SR 2

Mit einem Low-High-Wechsel am Pin DCD wird SR2 gesetzt. Ist der Receiver Interrupt zugelassen, so wird ein IRQ ausgelöst. Gelöscht wird das Bit, wenn zuerst das Statusregister, danach das Receiver Register gelesen wird. Damit wird auch der IRQ aufgehoben. Bleibt nach dem Lesen der beiden Register SR2 weiterhin High, so ist dies ein Zeichen dafür, das auch das Signal am DCD-Pin noch High ist. Sobald DCD auf Low geht, wird dann auch SR2 Low werden.

SR 3

Dieser Anschluß zeigt direkt den Zustand des Pins CTS an. Dies Signal kann nicht durch Master Reset oder Programmierung der ACIA beeinflußt werden.

SR 4

SR4 zeigt sogenannte Frame Fehler. Frame- oder Rahmenfehler entstehen, wenn die Empfängerschaltung kein Stopbit erkennen kann. Es wird mit jedem neuen Zeichen entsprechend gesetzt.

SR 5

Dieses Bit zeigt die schon erwähnte Overrun-Bedingung an. Zurückgesetzt wird SR5 mit dem Lesen des RX Buffers.

SR 6

An Hand dieses Bits kann bei eingeschalteter Paritätsprüfung erkannt werden, ob die Parität des empfangenen Zeichens stimmt. Im Fehlerfall ist das Bit gesetzt.

SR 7

Bit 7 des Status Registers signalisiert den Zustand des

IRQ-Pins. Dies Bit ermöglicht das Zusammenschalten von mehreren IRQ-Anschlüssen an einen Interrupt-Eingang. Im Falle eines Interrupts kann programmtechnisch ermittelt werden, welches IC den Interrupt ausgelöst hat.

Die ACIAs im ATARI ST

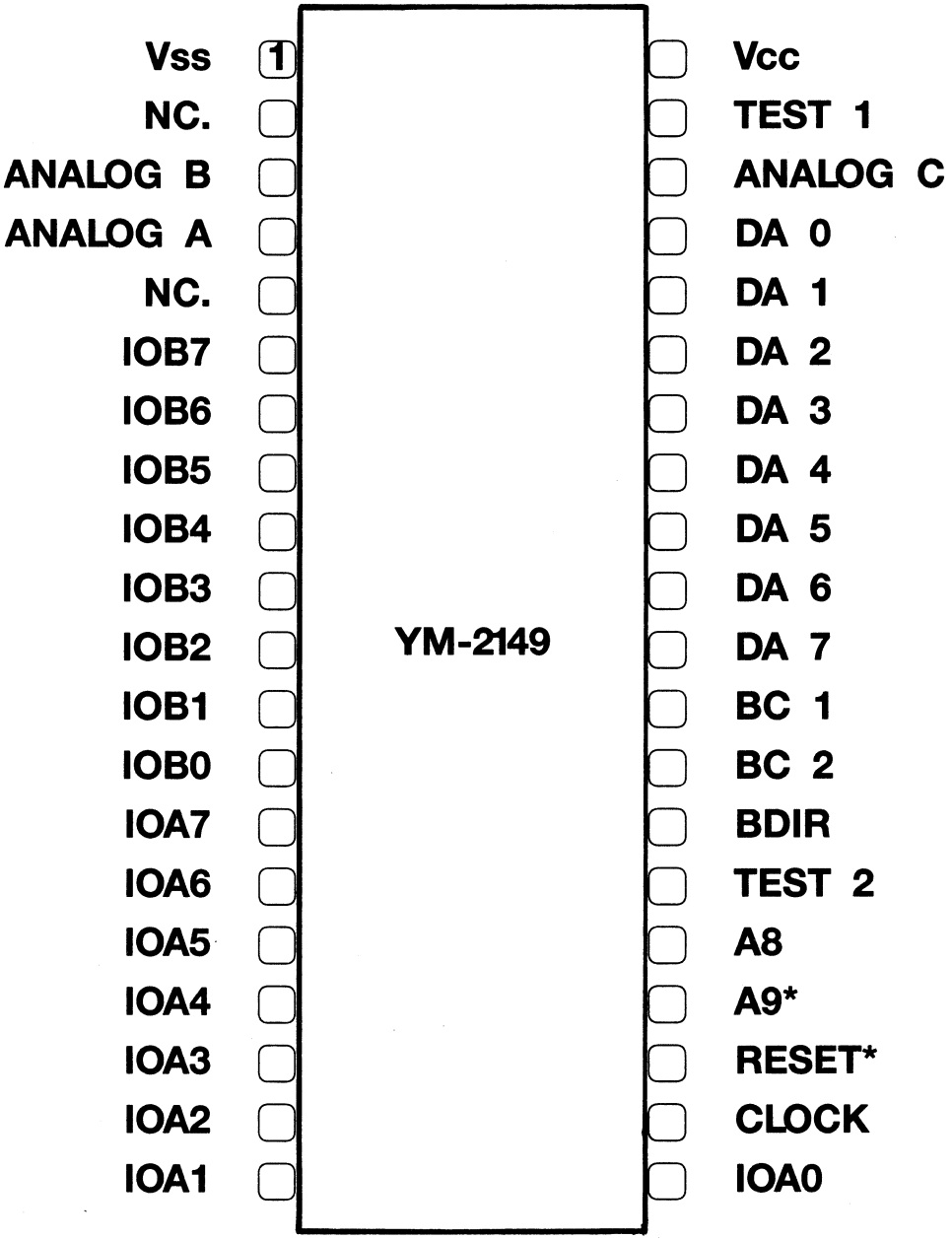
Viele Möglichkeiten der ACIAs werden im ST nicht benötigt. Insbesondere sind die Anschlüsse CTS, DCD und RTS nicht belegt. Für den vorgesehenen Verwendungszweck ist das aber auch nicht erforderlich.
Die Tastatur-ACIA belegt die Adressen $FFFC00 und $FFFC02. Die eingestellten Parameter sind 8 Datenbits, 1 Stopbit, keine Parity, 7812.5 Baud (500kHz/64).
Bis auf die Baudrate sind die Parameter der MIDI-Schnittstelle identisch. Sie beträgt 31250 Baud (500kHz/16). Dieser Wert ist durch die MIDI-Spezifikationen festgelegt.

1.6 Der Sound Generator YM-2149

Der YM-2149 von YAMAHA, ein mit dem recht verbreiteten General Instruments IC AY-3-8910 kompatibler Sound Generator (PSG) ist ein Soundchip der Spitzenklasse. Er wurde entwikkelt für Telespiele, um diese mit besonders realistischem Sound zu versehen, nachdem die ersten Telespiele nur recht monotone Geräusche produzieren konnten.
Um möglichst universell einsetzbar zu sein, wurde der PSG mit einer Vielzahl von Möglichkeiten zur Klangbeeinflussung versehen. Zusätzlich sagte man sich bei der Entwicklung dieses ICs wohl, daß in fast allen Einsatzgebieten auch irgendwelche Tasten, Joysticks oder Schalter abgefragt werden müssen. So gab man diesem PSG auch noch zwei bidirektionale 8-Bit-Parallelports mit.
Die Leistungsdaten dieses ICs im Überblick lauten:

- *drei unabhängig programmierbare Ton-Oszillatoren*
- *ein programmierbarer Rausch-Generator*
- *vollständig software-gesteuerte Analogausgänge*
- *programmierbarer Mischer für Ton und Rauschen*
- *15 logarithmisch gestufte Lautstärkestufen*
- *Programmierbare Hüllkurven*
- *zwei bidirektionale 8-Bit-Datenports*
- *TTL-Kompatibel*
- *Einfache 5 Volt Betriebsspannung*

Insgesamt verfügt der YM-2149 über 16 Register. Über diese Register können alle Klangmöglichkeiten des Chips programmiert werden.
Die Schaltung des PSG kann in einzelne Funktionsblöcke unterteilt werden.
Da ist zunächst der Block der Tongeneratoren. Die Tongeneratoren werden mit einem Taktsignal versorgt, das aus dem durch 16 geteilten Clock-Signal gewonnen wird. Die Tongeneratoren sind zuständig für die grundsätzliche Erzeugung der drei rechteckförmigen Tonfrequenzen.
Der Rauschgenerator erzeugt ein frequenzmoduliertes Rechtecksignal, dessen Pulsbreite von einem Pseudo-Rauschgenerator beeinflußt wird.


Vss 1
NC.
ANALOG B
ANALOG A
NC.
IOB7
IOB6
IOB5
IOB4
IOB3
IOB2
IOB1
IOB0
IOA7
IOA6
IOA5
IOA4
IOA3
IOA2
IOA1
YM-2149
Vcc
TEST 1
ANALOG C
DA 0
DA 1
DA 2
DA 3
DA 4
DA 5
DA 6
DA 7
BC 1
BC 2
BDIR
TEST 2
A8
A9*
RESET*
CLOCK
IOA0

Die Mischer koppeln die Ausgangssignale der drei Generatoren mit dem Rauschsignal. Die Kopplung kann für jeden Kanal getrennt programmiert werden.
Der Funktionsblock der Amplitudenkontrolle bietet dem Anwender zwei Möglichkeiten. Zum einen kann die Ausgangsamplitude (die Lautstärke) der drei Kanäle über die Programmierung des entsprechenden Lautstärke-Registers beeinflußt werden.
Alternativ besteht die Möglichkeit, sie vom PSG variabel zu beeinflussen. Dann wird der Ausgang des Hüllkurven-Registers genutzt, um die Lautstärke zu beeinflussen. Da die Hüllkurve (auch als Envelope, Umschlag oder Umhüllung bezeichnet) mit vier getrennten Parametern programmierbar ist, bestehen vielfältige Möglichkeiten der Tonbeeinflussung.
Der Funktionsblock der D/A-Wandler ist zuständig für die Erzeugung der Lautstärke der Ausgangssignale. Da die Lautstärke- und Envelope-Informationen als digitale Werte vorliegen, werden sie im D/A-Wandler umgesetzt.
Der letzte Funktionsblock hat mit der Tonerzeugung nichts zu tun. In diesem Block sind die zwei I/O-Ports untergebracht.

1.6.1 Die Anschlüsse des Sound Chip

Da die Bezeichnungen der Anschlüsse des PSG nicht unbedingt selbsterklärend sind, hier die detaillierte Beschreibung der Funktion der Pins:

Vss :
Dies ist der Masse-Anschluß des PSG.

NC. :
Dieser Anschluß wird nicht belegt.

ANALOG B :
Dies ist der Ausgang des Kanals B. Hier können die von Kanal B erzeugten Töne abgenommen werden. Die maximale Ausgangsspannung ist 1 Vss.

ANALOG A :
Funktion wie Pin 3 für den Kanal A.

NC. :
Dieser Anschluß wird nicht belegt.

IOB7 - 0 :
Die IOB-Anschlüsse stellen einen der zwei 8-Bit-Ports des PSG dar. Je nach Programmierung arbeiten die Anschlüsse als Ausgang oder Eingang. Dabei kann nur die Betriebsart für den ganzen Port eingestellt werden. Ein gemischter Betrieb (gleichzeitig Bits als Eingang, andere Bits als Ausgang) ist nicht möglich, wohl aber können die Datenrichtungen der beiden Ports getrennt voneinander bestimmt werden.

IOA7 - 0 :
Funktion wie IOB, nur für den Port A

CLOCK :
Von der Frequenz dieses Signals werden alle Tonfrequenzen durch Teilung abgeleitet. Die Frequenz dieses Signals sollte zwischen 1 und 2 MHz liegen.

RESET :
Durch einen Low-Pegel an diesem Anschluß werden alle internen Register zurückgesetzt. Ohne Reset stehen nach dem Einschalten zufällige Werte in den verschiedenen Registern, die Folge wäre ein (wahrscheinlich) sehr unmusikalisches 'Geräusch'.

A9 :
Dieser Anschluß kann als eine Art Chip Select-Signal verstanden werden. Um Register des PSG ansprechen zu können, muß dieses Signal Low sein.

A8 :
Funktion wie A9, nur ist dieser Anschluß High-aktiv.

TEST2 :

Test2 wird nur von der Herstellerfirma verwendet und muß im Betrieb unbeschaltet bleiben.

BDIR & BC1,2 :

Der Anschluß BDIR-Signal (Bus DIRection) und die Anschlüsse BC1 und BC2 (Bus Control) steuern den Registerzugriff auf den PSG. Auf den ersten Blick mag die in der Tabelle gezeigte Zuordnung etwas seltsam erscheinen. Da dies IC jedoch ursprünglich als Baustein zum Prozessor 1610, einem speziellen 16-Bit-Prozessor von General Instruments, entwikkelt wurde, hat man beim Entwurf auf die speziellen Eigenschaften und Steueranschlüsse dieses Prozessors Rücksicht genommen.

BDIR	BC2	BC1	Funktion des PSG
0	0	0	INACTIVE
0	0	1	LATCH ADRESS
0	1	0	INACTIVE
0	1	1	READ FROM PSG
1	0	0	LATCH ADRESS
1	0	1	INACTIVE
1	1	0	WRITE TO PSG
1	1	1	LATCH ADRESS

Innerhalb dieser Tabelle sind nur vier von acht Kombinationen wirklich sinnvoll. Darum wird häufig der Anschluß BC2 auf +5 Volt gelegt. Die verbleibende Tabelle wird nur noch von den Signalen BDIR und BC1 bestimmt und sieht folgendermaßen aus:

BDIR	BC1	Funktion
0	0	INAKTIV, PSG-Datenbus hochohmig
0	1	READ, PSG-Register lesen
1	0	WRITE, PSG-Register beschrieben
1	1	LATCH, Registernummer schreiben

DA0 - 7 :

Diese Anschlüsse des Sound Chips werden mit dem Datenbus des Prozessors verbunden. Die Bezeichnung DA deutet an, daß sowohl Daten als auch (Register-)Adressen über diese Anschlüsse geführt werden.

ANALOG C :

Funktion wie Pin 3 für den Kanal C.

TEST1 :

Wie auch Test1 wird dieser Anschluß nur von der Herstellerfirma verwendet und muß im Betrieb unbeschaltet bleiben.

Vcc :

An diesem Anschluß wird die Betriebsspannung von +5 Volt angelegt.

1.6.2 Die Funktion der einzelnen Register des 2149

Da jetzt geklärt ist, wie über die Anschlüsse BDIR und BC1 die Register grundsätzlich angesprochen werden können, wollen wir sehen, was für Funktionen die Register ausführen. Dabei ist die Registernummer, die in der folgenden Aufstellung verwendet wird, gleich mit der Nummer, die im Adress-Register eingetragen werden muß, um das gewünschte Register anzusprechen.
Interessant ist noch die Tatsache, daß das Adressregister seinen Inhalt bis zur nächsten Programmierung behält. Man kann also ohne Probleme mehrmals nacheinander auf ein Datenregister zugreifen, ohne jedes Mal das Adressregister neu laden zu müssen.
Doch jetzt zur Registerbeschreibung.

Reg 0,1 :

Diese Register bestimmen die Periodendauer und damit die Frequenz des Tonsignals an ANALOG A. Allerdings

sind nicht alle 16 Bit benutzt. Verwendung finden alle 8 Bit des Register 0 und die vier unteren Bit des Registers 1. Dabei kann die Frequenz mit dem Register 0 fein, mit dem Reg. 1 in groben Stufen beeinflußt werden. Je kleiner der 12-Bit-Wert dieser Register wird, desto höher wird der Ton.

Reg 2,3 :
Funktion wie Reg 0,1, aber Kanal B.

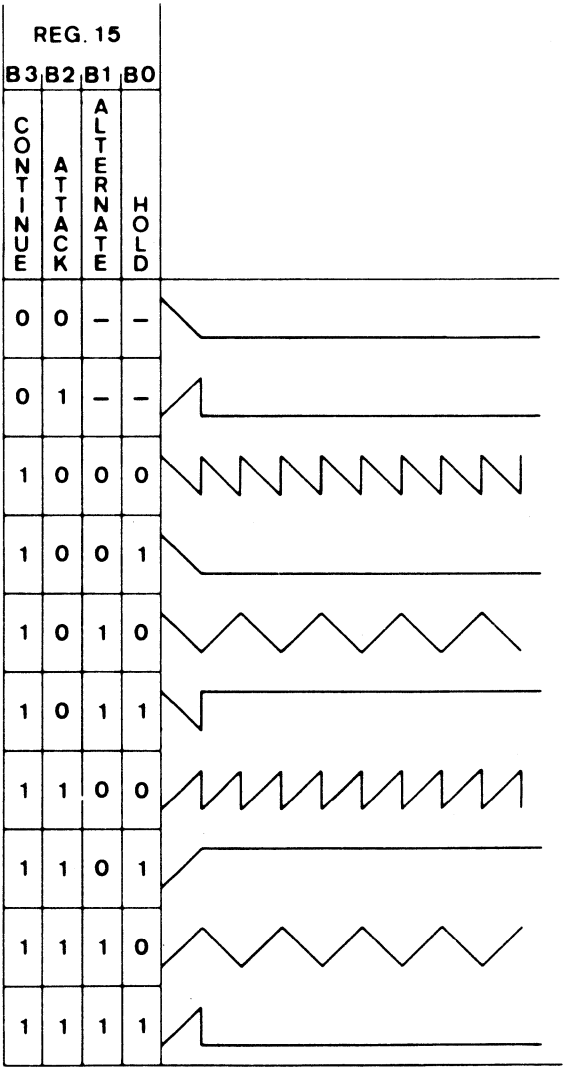
Reg 4,5 :
Funktion wie Reg 0,1, aber Kanal C.

Reg 6 :
Dieses Register beeinflußt mit seinen unteren 5 Bit den Rausch-Generator. Auch hier gilt: je kleiner der Wert im Register, desto höher die RauschFrequenz.

Reg 7 :
In diesem Multi-Funktionsregister kontrollieren einzelne Bits unterschiedliche Aufgaben.

```
Bit 0:Ton von Kanal A ein-/ausschalten    0=ein / 1=aus
Bit 1:Ton von Kanal B ein-/ausschalten    0=ein / 1=aus
Bit 2:Ton von Kanal C ein-/ausschalten    0=ein / 1=aus
Bit 3:Rauschen zu Kanal A zu-/abschalten  0=zu  / 1=ab
Bit 4:Rauschen zu Kanal B zu-/abschalten  0=zu  / 1=ab
Bit 5:Rauschen zu Kanal C zu-/abschalten  0=zu  / 1=ab
Bit 6:Port A als Ein-/Ausgang             0=in  / 1=out
Bit 7:Port B als Ein-/Ausgang             0=in  / 1=out
```

Reg 8 :
Dieses Register bestimmt die Lautstärke des Signals an Kanal A. Zur Lautstärkeeinstellung werden die vier unteren Bits verwendet.
Das Bit 4 hat eine besondere Bedeutung. Ist es gesetzt, dann wird die Lautstärke durch das Hüllkurven-Register bestimmt, der Inhalt der Bits 0 bis 3 wird dann ignoriert.



REG. 15				
B3	B2	B1	B0	
CONTINUE	ATTACK	ALTERNATE	HOLD	
0	0	–	–	
0	1	–	–	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

1.8.2.1 Hüllkurven des PSG

Reg 9 :
Wie Reg 8 für Kanal B.

Reg 10 :
Wie Reg 8 für Kanal C.

Reg 11,12 :
Alle 16 Bit dieser beiden Register beeinflussen die Periodendauer der Hüllkurve. Der Inhalt des Reg. 11 wird als Low-Byte betrachtet, d.h. er beeinflußt die Periodendauer in feinen Schritten, das Reg. 12 ist das High-Byte des Hüllkurven-Generators.

Reg 13 :
Die Bits 0 bis 3 dieses Registers bestimmen die Kurvenform des Hüllkurven-Generators. Die Zuordnung in Worten verständlich wiedergeben zu wollen, ist fast unmöglich. Die erzeugbaren Hüllkurven sind darum in der Grafik 1.6.2 gezeigt.

Reg 14,15 :
Diese beiden Register sind den beiden 8 Bit-Ports zugeordnet. Port A belegt das Register 14, Port B das Register 15. Ist ein Port als Ausgang programmiert (Bits 6 und 7 im Reg. 7), so kann über diese Register der auszugebende Wert programmiert werden. Bei einem Betrieb als Eingang kann der an den Anschlüssen anliegende Wert über diese Register ermittelt werden.

1.7 Die Belegung des I/O-Bereichs des ST

Die gesamte Ein/Ausgabe, also alle Peripherie-ICs und auch die Register der speziellen ATARI-Chips belegen den 32 KByte großen Adressbereich von $FF8000 bis $FFFFFF. Da der gesamte I/O-Bereich so reichlich bemessen ist, ergeben sich natürlich große Lücken, die einzelne zusammengehörende Registergruppen von anderen Registern trennen.

Die folgende Tabelle gibt als Übersicht Aufschluß über die einzelnen zu unterscheidenden Bereiche. Bedenken Sie aber bitte, daß der I/O-Bereich nur im Supervisor-Modus zugänglich ist. Ein Zugriff im User-Mode führt grundsätzlich zu einem Bus-Fehler.

```
$FF8000  Speicherkonfiguration
$FF8200  Register für Video-Anzeige
$FF8400  reserviert
$FF8600  DMA/Disk-Controller
$FF8800  Sound Chip
$FFFA00  MFP 68901
$FFFC00  ACIAs für Tastatur und MIDI
```

Die Adressangaben zeigen nur den Beginn der jeweiligen Bereiche an und geben noch keinen Aufschluß über die tatsächlich vorhandenen Register. Darum sollen die einzelnen Bereiche etwas detailierter untersucht werden.

$FF8000 Speicherkonfiguration

In diesem Bereich existiert ein einziges 8-Bit-Begister auf der Adresse $FF8001. In ihm wird in denunteren vier Bits die Speicherkonfiguration eingestellt. Bevor wir jedoch auf die möglichen Einstellungen eingehen, kurz noch ein paar Worte zu den von der Hardware des ST vorgegebenen Möglichkeiten.
Das MMU-IC wurde von den ATARI-Entwicklern auf maximale Vielseitigkeit konstruiert. So lassen sich im ST je nach gewünschtem Speicherausbau drei verschiedene Speichertypen

verwenden. Es sind dies die Typen 64KBit, 256KBit und die derzeit noch nicht in Massenstückzahlen erhältlichen 1Megabit-Chips. Da alle diese ICs Bit-orientiert sind, werden für den Speicherausbau jeweils 16 Speicher-ICs benötigt. Die Bezeichnung für 16 solcher ICs (unabhängig von der Speicherkapazität) ist Bank. Damit ergeben sich die möglichen Ausbaustufen zu 128 KByte, 512 KByte oder gar 2 Megabyte.
Allerdings kann die MMU bei Bedarf auch zwei getrennte Banks ansteuern. Die für die zweite Bank benötigten RAS- und CAS-Signale sind an der MMU vorhanden. Da die Bestückungen in den Banks unterschiedlich sein dürfen, ergeben sich viele Kombinationsmöglichkeiten, die in der folgenden Tabelle aufgezeigt werden.
Die wesentliche Aufgabe des Konfigurationsregisters ist es, die vielen möglichen Speicherausbauten jeweils zu einem durchgehend adressierbaren Speicherbereich zu verbinden. Ohne diese Fähigkeit der MMU würde bei Bestückung mit zwei Banks zu je 512 KByte eine große Lücke zwischen den beiden Speicherbereichen entstehen.

$FF8001	Bit	Speicher Konfiguration	
	3-0	Bank 0	Bank 1
	0000	128K	128K
	0001	128K	512K
	0010	128K	2 M
	0011	reserviert	
	0100	512K	128K
	0101	512K	512K
	0100	512K	2 M, im 520 ST üblich
	0100	reserviert	
	1000	2 M	128K
	1001	2 M	512K
	1010	2 M	2 M
	1011	reserviert	
	11xx	reserviert	

Das Speicherkonfigurationsregister kann gelesen und beschrieben werden.

$FF8200 Register für Video-Anzeige

In den Registern dieses Adressbereichs werden die Lage des Video-RAMs, die eingestellte Auflösung des Bildschirms und die Werte der Farbpalette gespeichert.

```
$FF8201  8-Bit     Lage des Bildschirmspeichers Highbyte
$FF8203  8-Bit     Lage des Bildschirmspeichers Midbyte
```

Diese zwei sowohl zu beschreibenden wie auch zu lesenden Register bestimmen den Anfang des 32K-Video-RAMs im gesamten Speicher. Um das Video-RAM an jeder beliebigen Adresse beginnen zu lassen, müßte ein weiteres Register für das Lowbyte vorhanden sein. Ein solches Register fehlt jedoch, als Wert wird immer 0 angenommen. Dadurch kann das Video-RAM nur in Stufen von 256 Bytes in seiner Lage verschoben werden. Nach dem Einschalten wird das Video-RAM an die obere Speichergrenze gelegt. Im 520 ST ergibt das eine Bildschirm-Basisadresse von $78000.

```
$FF8205  8-Bit     Video Adresszähler Highbyte
$FF8207  8-Bit     Video Adresszähler Midbyte
$FF8209  8-Bit     Video Adresszähler Lowbyte
```

Diese drei Register können nur ausgelesen werden. Sie geben Auskunft über die derzeit dargestellte Videoadresse. Im ST wird der Videospeicher vollkommen linear abgetastet. Alle zwei Mikrosekunden wird der Inhalt dieser drei Register um 2 erhöht.

```
$FF820A  Bit       Synchronisationsmodus
         1 0
         : :---    0=interne, 1=externe Synchronisation
         :-----    0=60 Hertz, 1=50 Hertz Bildfrequenz
```

Die unteren zwei Bits dieses Registers bestimmen den

sogenannten Sychronisationsmodus. Die restlichen Bits werden nicht genutzt. Ist das Bit 0 gesetzt, so werden die intern erzeugten HSync- und VSync-Impulse ausgeschaltet. Dann kann die Synchronisation des Bildes von außen über die Monitor-Buchse zugeführt werden. Diese Option schafft ganz neue Möglichkeiten besonders im Bereich der Videofilmtechnik. Da sich auf einfache Weise die Bilder einer Video-Kamera und des ST synchronisieren lassen, kann der ST zum Einblenden von Schrift oder Grafik in einen Film eingesetzt werden.
Bit 1 des Sync-Mode-Registers bestimmt die Bildfrequenz. Allerdings ist dieses Bit nur in den beiden 'niedrigen' Auflösungstufen wirksam. In der hochauflösenden Betriebsart arbeitet der ST immer mit ca. 70 Hertz Bildfrequenz, die vom ATARI-Monitor vorgeschrieben werden, um die hohe Auflösung auch flimmer- und verzerrungsfrei darzustellen.
Das Sync-Mode-Register kann sowohl gelesen wie auch beschrieben werden.

```
$FF8240  16-Bit    Farbpalettenregister 0
$FF8242  16-Bit    Farbpalettenregister 1
   :        :              :
   :        :      Farbpalettenregister 2 bis 13
   :        :              :
$FF825C  16-Bit    Farbpalettenregister 14
$FF825E  16-Bit    Farbpalettenregister 15
```

Insgesamt verfügt der ST über 512 darstellbare Farben, kann aber nur maximal 16 unterschiedliche Farben zu einer Zeit auf dem Bildschirm darstellen. Diese Beschränkung kann so erklärt werden, daß man sagt, der Anwender habe 16 Farbstifte zur Verfügung, und jeder Farbstift könne eine von 512 Farben haben. Die Farbpalettenregister stellen nun die 16 Stifte dar. Die Codierung der Farben geschieht bitweise nach einem festen Schema. Jedes der 16 Register enthält 9 die Farbe beeinflussende Bits, die wie folgt organisiert sind:

```
FEDCBA9876543210
-----xxx-xxx-xxx
```

Die durch ein x gekennzeichneten Bits sind in den Registern beeinflußbar, die mit einem - markierten tellen in dem Registerwort haben keine Auswirkung. Die Bits 0 bis 2 bestimmen den Blau-Anteil der gewünschten Farbe, Bit 4 bis 6 sind für den Grün-Anteil und Bit 8 bis A für den Rot-Anteil zuständig. Je höher der Wert in den drei Bits ist (0 bis 7 sind ja jeweils möglich), um so intensiver wird die resultierende Farbe.
In der mittleren Auflösungsstufe sind bei einer Grafik-Auflösung von 640 x 200 Punkten nur vier verschiedene Farben gleichzeitig darstellbar. Dann werden die Palettenregister 4 bis 15 ignoriert.
Wenn man von maximal 16 darstellbaren Farben spricht, so ist das zwar grundsätzlich richtig, der Inhalt des Palettenregisters 0 jedoch bestimmt in allen Auflösungsstufen die Farbe des Hintergrundes.
Bei der höchsten Auflösungsstufe mit 640 x 400 Punkten kann nur eine 'Farbe' gewählt werden. In diesem Fall bestimmt das Bit 0 des Palettenregisters 0 die Hintergrundfarbe. Ist das Bit gelöscht, so erscheint die Schrift schwarz auf hellem Hintergrund, bei gesetztem Bit 0 wird das Bild invertiert, d.h. es erscheinen helle Buchstaben auf dunklem Hintergrund.
Die Farb-Register können sowohl gelesen wie auch beschrieben werden.

$FF8260	Bit	Auflösung
	1 0	
	0 0	320 mal 200 Punkte, vier Bildebenen
	0 1	640 mal 200 Punkte, zwei Bildebenen
	1 0	640 mal 400 Punkte, eine Bildebene
	1 1	reserviert

In diesem Register wird die Hardware auf die gewünschte Grafik-Auflösung eingestellt.
Das Auflösungsregister kann sowohl gelesen wie auch beschrieben werden.

$FF8600 DMA/Disk-Controller

Der DMA/Disk-Controller-Bereich beinhaltet alle Register die in irgend einer Weise mit DMA und Diskettenbetrieb zu tun haben. Dazu gehören Zugriff auf die Register des FDC, Status-Informationen über den Erfolg einer DMA-Operation und DMA-Basis-Adressregister.

```
$FF8600          reserviert
$FF8602          reserviert

$FF8604  16-Bit  FDC-Access/Sektor Count
```

Über die unteren 8 Bits dieser Adresse kann der Prozessor üblicher Weise auf die Register des FDC zugreifen. Die oberen 8 Bits enthalten keine Information und sind immer 1. Welches Register des FDC über $FF8604 angesprochen wird, hängt von der Information im DMA Mode Control-Register $FF8606 ab. Der Zugriff auf den FDC erfoglt also immer indirekt.
Das Sektor Count-Register unter der Adresse $FF8604 kann angesprochen werden, wenn das entsprechende Bit im DMA Mode Control-Register $FF8606 gesetzt wird.
Der Inhalt dieser Adresse kann gleichermaßen beschrieben wie gelesen werden. Der Zugriff muß ein Wort-Zugriff sein.

```
$FF8606  16-Bit  DMA Mode/Status
```

Wird dieses Register gelesen, so erhält man den DMA Status in den unteren drei Bit des Registers. Die Bits bedeuten im einzelnen:

```
Bit 0    0=kein Fehler, 1=Fehler bei DMA
Bit 1    0=Sektor Count ist Null, 1=Sektor Count <> Null
Bit 2    Zustand des Signals DATA REQUEST des FDC.
```

Schreibzugriffe auf diese Adresse beeinflussen das DMA-Mode-Register. Auch hier sind einzelnen Bits bestimmte Aufgaben zugeordnet:

```
Bit 0    nicht verwendet
Bit 1    0=Pin A0 ist Low
         1=Pin A0 ist High
Bit 2    0=Pin A1 ist Low
         1=Pin A1 ist High
Bit 3    0=Zugriff auf FDC
         1=Zugriff auf HDC
Bit 4    0=Zugriff auf FDC-Register
         1=Zugriff auf Sector Count Register
Bit 5    0, reseviert
Bit 6    0=DMA einschalten
         1=kein DMA
Bit 7    0=Zugriff auf Hard Disk Controller (HDC)
         1=Zugriff auf FDC
Bit 8    0=FDC/HDC-Register lesen
         1=FDC/HDC-Register schreiben

$FF8609  8-Bit    DMA Basis und Zähler High-Byte
$FF860B  8-Bit    DMA Basis und Zähler Mid-Byte
$FF860D  8-Bit    DMA Basis und Zähler Low-Byte
```

Bei DMA-Transfers muß der Hardware mitgeteilt werden, ab welcher Speicheradresse die Daten übertragen werden sollen. Diese Basis-Adresse wird in den zuvor genannten Registern eingetragen. So muß beim Lesen von der Diskette die Adresse eingetragen, wohin die Daten geladen werden sollen. Mit jedem übertragenen Wort wird der Inhalt der Register entsprechend erhöht.
Die Initialisierung der drei Register muß mit dem Low-Byte beginnen. Danach sind (in dieser Reihenfolge) das Mid- und High-Byte zu initialisieren.

$FF8800 Sound Chip

Der im ST eingesetzte Soundchip YM-2149 verfügt über 16 interne Register, die allerdings nicht direkt adressiert werden können. Statt dessen muß zuerst die Nummer des gewünschten Registers in das Select Register geladen werden.

Danach kann das ausgewählte Register solange beschrieben oder gelesen werden, bis eine neue Registernummer in den PSG geschrieben wird. Damit reduziert sich die Anzahl der benötigten Register von 16 auf 2.

```
$FF8800  8-Bit      Read Data/Register Select
```

Beim Lesen dieser Adresse erhält man die Daten aus dem zuletzt angewählten Register. Das wird normalerweise das Register des Port A sein, da der ST bei jedem Timer-Interrupt (alle 5 Millisekunden) die Diskettenlaufwerke selektiert, um einen Diskettenwechsel festzustellen. Diese Änderung kann übrigens an Hand des Write-Protect-Signals ermittelt werden. Wird jedoch ein anderes Register ausgewählt, so kann auf dessen Inhalt zugegriffen werden. Man muß allerdings darauf gefasst sein, daß zwischen Schreiben der Registernummer und Lesen des Inhalts ein Interrupt auftreten kann, so daß letztendlich trotzdem der Port A gelesen wird. Darum sollte der Interrupt für die Dauer der gewünschten Operation ausgeschaltet sein. In diesem Zusammenhang soll noch die Verwendung 16 Port-Bits geklärt werden. Port B wird als Datenport für den Drucker, Port A für diverse Steuerungsaufgaben benutzt.

```
Port A
Bit 0    Seitenauswahl-Signal für Doppelkopf-Floppys
Bit 1    Drive Select-Signal Floppy 0
Bit 2    Drive Select-Signal Floppy 1
Bit 3    RS 232 RTS-Ausgang
Bit 4    RS 232 DTR-Ausgang
Bit 5    Centronics Strobe
Bit 6    frei verwendbarer Ausgang (Monitor-Buchse)
Bit 7    reserviert
```

Wird auf die Adresse $FF8800 geschrieben, so wird das Select-Register des PSG angesprochen. Die Information in den unteren vier Bit wird dann als Registernummer ausgewertet. Die benötigte 4-Bit-Nummer entnehmen Sie bitte der Beschreibung des PSG.

```
$FF8802  8-Bit     Write Data
```

Über diese Adresse werden die Daten in das zuvor gewählte Register des PSG geschrieben. Ein Lesen dieser Adresse ergibt nur den Wert $FF, da in diesem Fall an BDIR und BC1 eine Null anliegt. Das aber ist eine Eingangskombination, die den PSG nicht selectiert.
Die etwas ungewöhnliche Anordnung der Register ist bei näherer Betrachtung äußerst gelungen. Soll ein Register gelesen werden, so kann ohne Neuberechnung der Adresse das Datum aus dem PSG nach Übermittlung der Registernummer gelesen werden. Das Schreiben von Registernummer und Daten kann mit einem einzigen MOVEP-Befehl geschehen.

$FFFA00 MFP 68901

Der MFP mit seinen 24 Registern belegt die ungeraden Adressen von $FFFA01 bis $FFFA2F. Die Reihenfolge der Register im Speicher entspricht der Reihenfolge in der Beschreibung des MFP.

```
$FFFA01  8-Bit     Paralellport
$FFFA03  8-Bit     Flankenregister
$FFFA05  8-Bit     Datenrichtung
$FFFA07  8-Bit     Interrupt Enable A
$FFFA09  8-Bit     Interrupt Enable B
$FFFA0B  8-Bit     Interrupt Pending A
$FFFA0D  8-Bit     Interrupt Pending B
$FFFA0F  8-Bit     Interrupt In-Service A
$FFFA11  8-Bit     Interrupt In-Service B
$FFFA13  8-Bit     Interrupt Mask A
$FFFA15  8-Bit     Interrupt Mask B
$FFFA17  8-Bit     Vektor Register
$FFFA19  8-Bit     Timer A Control
$FFFA1B  8-Bit     Timer B Control
$FFFA1D  8-Bit     Timer C und D Control
$FFFA1F  8-Bit     Timer A Data
```

```
$FFFA21  8-Bit     Timer B Data
$FFFA23  8-Bit     Timer C Data
$FFFA25  8-Bit     Timer D Data
$FFFA27  8-Bit     Sync Character
$FFFA29  8-Bit     USART Control
$FFFA2B  8-Bit     Receiver Status
$FFFA2D  8-Bit     Transmitter Status
$FFFA2F  8-Bit     USART Data
```

Die Bedeutung und Programmierung der einzelnen Register können Sie dem Kapitel über den MFP entnehmen. Wir wollen an dieser Stelle nur beschreiben, wie die Anschlüsse des I/O-Ports und die Timeranschlüsse belegt sind.

```
I/O-Port
Bit 0    Centronics Busy
Bit 1    RS 232 Data Carrier Detect-Eingang
Bit 2    RS 232 Clear To Send-Eingang
Bit 3    reserviert
Bit 4    Keyboard und Midi-Interrupt
Bit 5    FDC und HDC-Interrupt
Bit 6    RS 232 Ring Indicator
Bit 7    Monocrome Monitor Detect
```

Die Timer A und B des MFP verfügen über je einen Eingang, mit dem externe Impulse gezählt werden können oder die als Steuereingang in der Timer-Betriebsart Pulsweitenmessung eingesetzt werden können. Timer A wird im ST nicht verwendet. Aus diesem Grund ist auch der entsprechende Eingang verfügbar. Da aber kein User-Port zur Verfügung steht, wurde der Timereingang mit auf den Busy-Anschluß der Centronics-Schnittstelle gelegt. Damit steht der Eingang dem Anwender zur Verfügung.
Timer B wird im ST zum Zählen der Bildschirmzeilen verwendet. Dazu ist das Signal DE, Display Enable an den Zählereingang B gelegt.
Die Ausgänge der Timer A bis C werden im ST nicht benutzt. Der Timer D aber liefert an seinem Ausgang die Taktsignale für die im MFP eingebaute serielle Schnittstelle. Dazu ist

der Timerausgang mit den Eingängen für Receiver- und Transmitterclock verbunden.

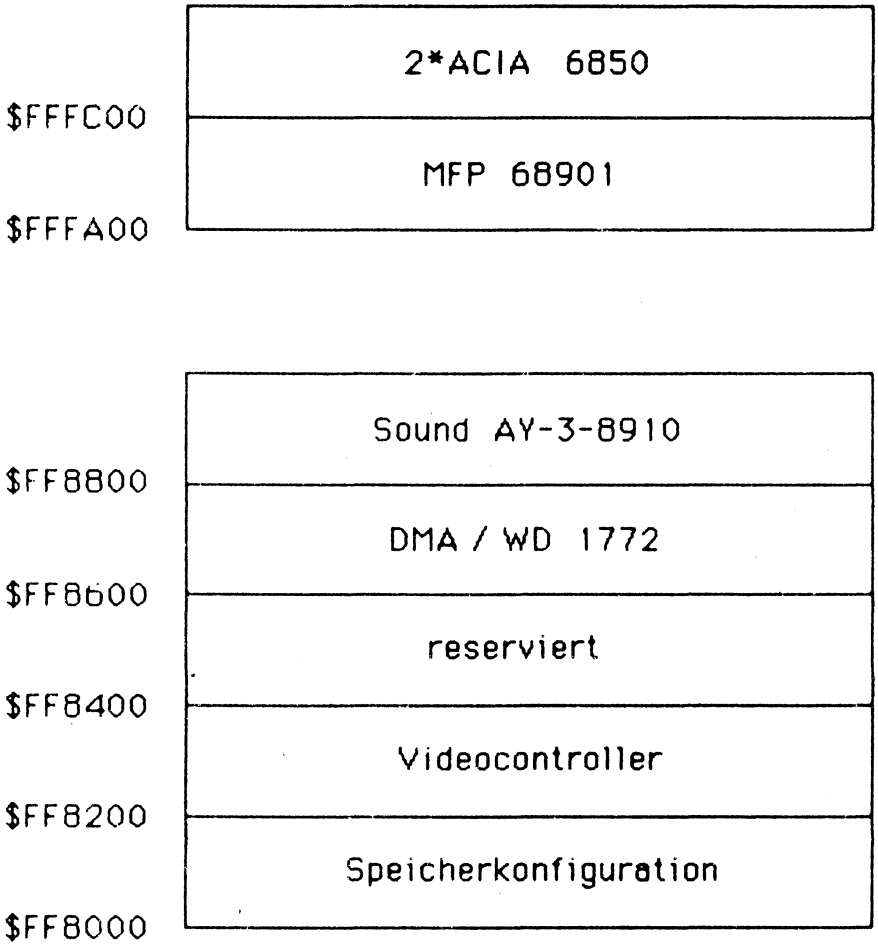
$FFFC00 Keyboard und MIDI-ACIAs

Die zwei zur Kommunikation mit der Tastatur und mit MIDI ausgerüsteten Musikinstrumenten eingesetzten ACIAs verfügen nur über je zwei Adressen. Sie sind in einem Adressblock zusammengefasst.

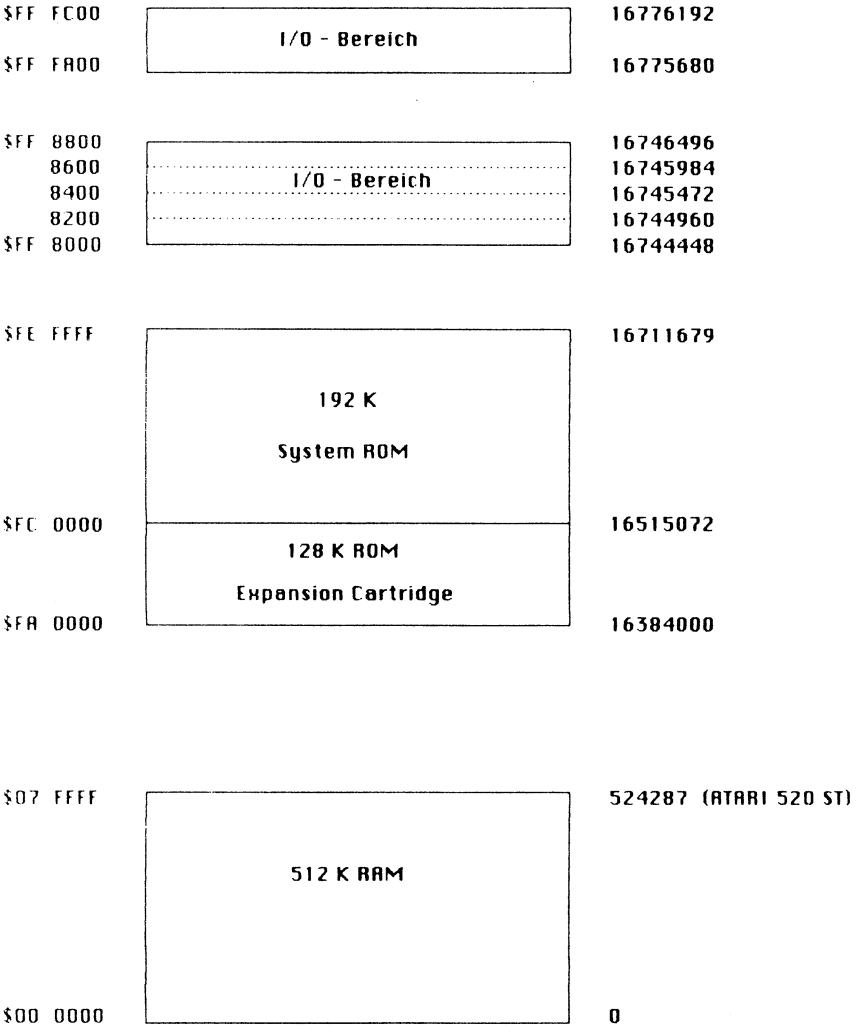
```
$FFFC00  8-Bit    Keyboard ACIA Control
$FFFC02  8-Bit    Keyboard ACIA DATA

$FFFC04  8-Bit    MIDI ACIA Control
$FFFC06  8-Bit    MIDI ACIA DATA
```

Die Aufteilung des I/O-Bereichs



Memory - Map des ATARI ST



2 DIE SCHNITTSTELLEN

2.1 Die Tastatur

Glauben Sie, es sei müßig, über eine so triviale Sache wie die Tastatur ausführlich zu berichten, da Tastaturen ja doch alle gleich funktionieren?
Weit gefehlt! Eigentlich müßte die Überschrift lauten: 'Keyboard Subsystem' oder so ähnlich. Die Tastatur wird nämlich von einem eigenen leibhaftigen Prozessor gesteuert. Welche Konsequenzen das für Sie als Assembler-Programmierer hat, werden Sie gleich sehen.
Als Prozessor dient ein Single Chip Computer aus der Familie 6800, nämlich der 6301. Single Chip bedeutet, daß alles, was zum Betrieb benötigt wird, sich in einem einzigen IC befindet. In der Tat ist auf der Tastatur-Platine auch nur der 6301 mit ein paar passiven Bauelementen zu sehen.
So verfügt der 6301 auf dem Chip über Rom, Ram, etliche I/O-Leitungen und sogar über eine serielle Schnittstelle. Über diese wird übrigens der Verkehr mit der Hauptplatine abgewickelt.
Der Vorteil einer derartigen Anordnung liegt auf der Hand: Der Hauptrechner braucht sich nämlich nicht mehr mit der zeitraubenden Abfrage der Tastaturmatrix abzuplagen. Stattdessen kann er sich voll der Abarbeitung Ihrer Programme widmen. Der Tastaturprozessor meldet sich selbsttätig, falls ein Ereignis eintreten sollte, von dem das Betriebssystem in Kenntnis zu setzen ist.
Dem 6301 obliegt aber nicht nur die relativ langweilige Bewachung der Tastatur. Er handelt auch souverän die wesentlich komplizierteren Aktionen ab, die im Zusammenhang mit der Maus notwendig sind. So erhält der Hauptrechner bei evtl. Bewegungen des Tierchens die neuen X- und Y-Koordinaten gleich mundgerecht serviert. Überflüssig zu erwähnen, daß auch alle Vorgänge, die möglicherweise angeschlossene Joysticks verursachen, dem Boß gepetzt werden.

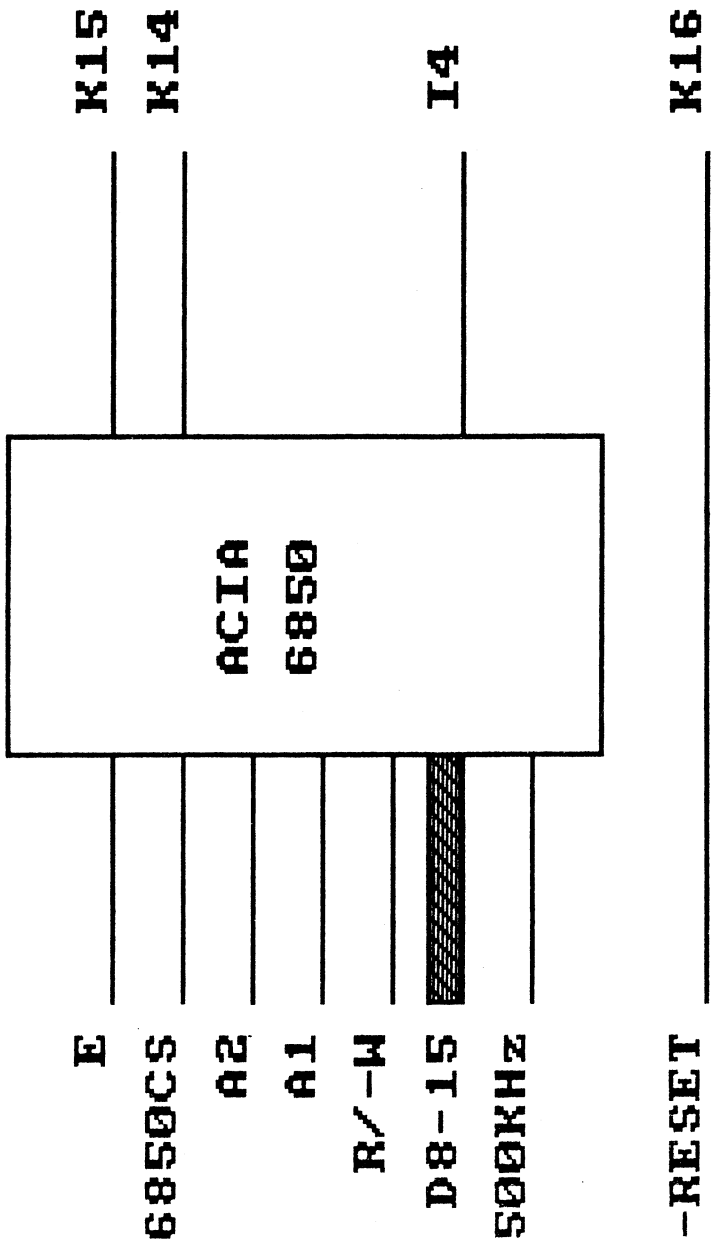


Abb. 2.1-1

Außerdem ist noch eine Echtzeituhr enthalten, die in Schritten von 1sec zählt.
Auf Abb. 2.1-1 finden Sie eine Übersicht über die Schnittstelle zum 68000. Wie Sie sehen, wird der Hauptrechner so wenig wie möglich belästigt. Die ACIA 6850 sorgt dafür, daß er nur gestört wird, wenn tatsächlich ein Byte vollständig von der Tastatur empfangen wurde. Die ACIA ist übrigens unter den Adressen $FFFC00 (Controlregister) und $FFFC02 (Datenregister) zu erreichen. Die einzige Verbindung zur Tastatur findet über die Leitungen K14 und K15 statt. K bezeichnet übrigens die Steckverbindung, über die die Tastatur mit der Hauptplatine verbunden ist.
Die Meldung, daß die ACIA ein Byte empfangen hat, wird über die Leitung I4 zunächst an den MFP 68901 gemeldet, der dann beim 68000 einen Interrupt auslöst. Die Taktfrequenz von 500KHz stammt vom GLUE. Daraus resultiert die 'krumme' Übertragungsgeschwindigkeit von 7812.5Bd.
Falls es Sie wundert, daß auch Daten *zum* Tastaturprozessor übertragen werden können, finden Sie des Rätsels Lösung in 2.1.2.

Das Blockschaltbild der Tastaturplatine finden Sie auf Abb. 2.1-2. Die Funktion ist ebenso einfach, wie das Bild übersichtlich ist. Kein Wunder, denn je umfangreicher die Software, desto einfacher die Hardware. So verfügt der Prozessor denn auch über stattliche 4K Rom. Das Ram ist mit 128 Byte vergleichsweise klein, aber es dient ja auch nur als Puffer und der Aufnahme einiger Pointer und Zähler.
Die mit K bezeichneten Leitungen sind wieder der Steckverbindung zur Hauptplatine zugeordnet. Mit wenigen Ausnahmen werden sie dort gleich zu den Anschlüssen für Maus und Joystick durchgeschleift. K16 ist die Reset-Leitung vom 68000. K15 führt die Sendedaten vom 6850, K14 die Sendedaten vom 6301.
Für die Abfrage der Tastaturmatrix sind die I/O-Ports 1(0-7), 3(1-7) und 4(0-7) zuständig. Über die Ports 3 und 4 wird zyklisch an je eine Leitung Low gelegt. Danach wird jedesmal der Zustand von Port 1 abgefragt. Ist eine Taste gedrückt, dringt der Low-Pegel zum Port 1 durch.

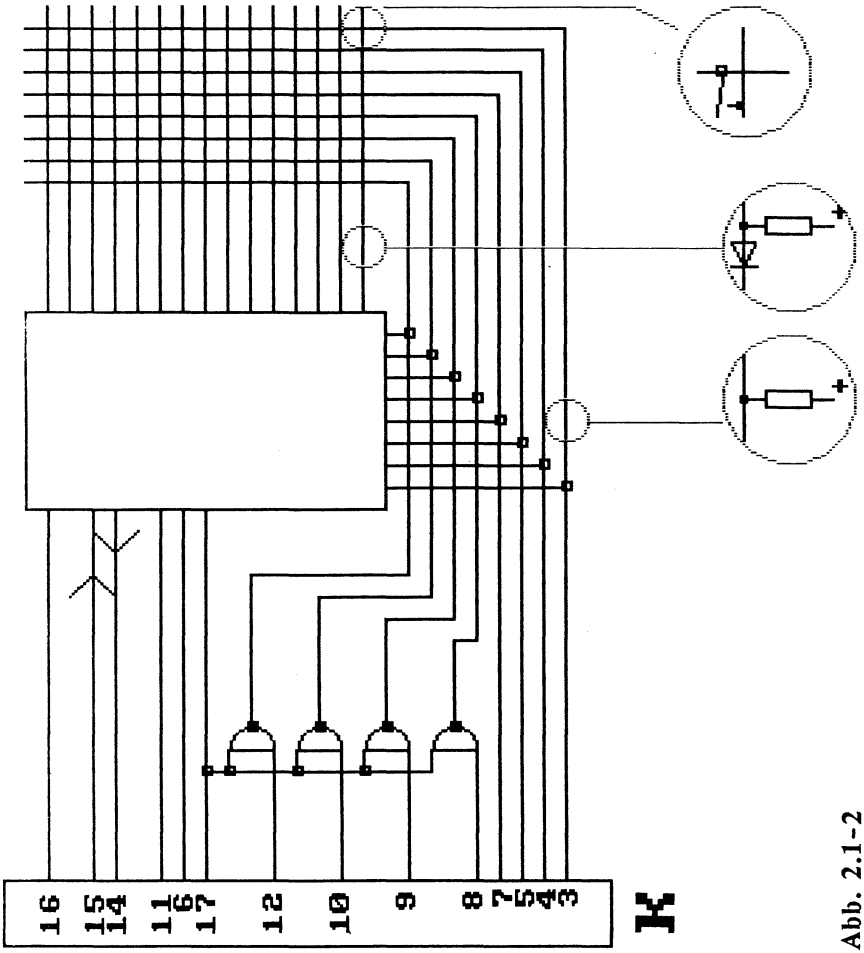


Abb. 2.1-2

Aus der Kombination der Werte an Port 3/4 und 1 läßt sich jede Taste identifizieren.
Ist keine der Leitungen von Port 3/4 auf Low gelegt und dennoch ein Bit von Port 1 =0, so ist am Außenstecker 1 ein Joystick aktiv. Die Daten von Außenstecker 0, an den ja die Maus oder auch ein Joystick angeschlossen werden kann, dringen nicht zufällig durch, da sie erst mit Port 2 (Bit 0) durch die NAND-Gatter geschleust werden müssen. Die Drucktasten der Maus, bzw. der Joysticks landen an Port 2 (1 u. 2).
Die Zuordnungen der K-Leitungen zu den Signalnamen der Außenstecker finden Sie im nächsten Kapitel.

Zum Schluß noch ein kleines Kuriosum: Der Prozessor 6301 ist zwar völlig eigenständig, jedoch läßt er sich auch so konfigurieren, daß er mit einem externen Rom arbeitet. Dabei werden etliche Portleitungen zu Adressleitungen umfunktioniert. Welche Konfiguration (von acht möglichen) der Prozessor annimmt, wird dadurch bestimmt, welche logischen Pegel während des Reset-Zyklus von außen an Port 2 (0-2) gelegt werden. Alle drei Leitungen High bringt den Prozessor in den Modus 7, eben den, der für die ihm hier zugedachte Aufgabe richtig ist. Nun hängen aber die Bits 1 und 2 an den Tasten der Maus. Läßt man diese während des Einschaltens in Ruhe, ist alles in Ordnung. Halten Sie jedoch beim Einschalten beide gedrückt, geht der Prozessor in den Modus 1 und macht eine prächtige Bauchlandung, da die übrige Hardware für diese Betriebsart nicht vorgesehen ist. Sie merken das daran, daß sich der Maus-Cursor auf dem Bildschirm nicht bewegt, was Sie auch unternehmen mögen. Da hilft nur der Druck auf das Reset-Knöpfchen.

2.1.1 DIE MAUS

Dieses kleine Gerätchen hat es in sich. Der Aufbau ist recht einfach, aber wirkungsvoll. Es besteht im wesentlichen aus vier Lichtschranken, zwei Taktscheiben und dem Antriebsmechanismus.
Die Aufgabe der Maus besteht darin, dem Computer Informationen über ihre Bewegungen zukommen zu lassen. Diese Information besteht aus den Komponenten: Richtung auf der X-Achse, Richtung auf der Y-Achse und zurückgelegter Weg auf jeder Achse.
Um das zu bewerkstelligen, treibt der von außen sichtbare Gummiball im Innern zwei Taktscheiben an, deren Antriebsachsen im Winkel von 90 Grad zueinander stehen. Je nachdem, in welche Richtung nun die Maus bewegt wird, dreht sich die eine oder andere Achse mehr oder weniger, vor oder zurück.
Die absolute Bewegung auf jeder Achse festzustellen, ist überhaupt kein Problem. Von den Taktscheiben werden nämlich zugehörige Lichtschranken unterbrochen, bzw. freigegeben. Man braucht nur noch die Impulse jeder Scheibe zu zählen, um über den zurückgelegten Weg auf jeder Achse informiert zu sein.
Schwieriger wird es, wenn auch noch die Bewegungsrichtung verlangt ist. Dazu haben jedoch die Konstrukteure zu einem genialen Trick gegriffen: Auf jeder Taktscheibe sitzt nämlich nicht nur eine, sondern zwei Lichtschranken, die sich auf dem Rad gegenüberstehen. Sie sind so angeordnet, daß sie durch die Scheibe nicht genau zur gleichen Zeit verdunkelt werden, sondern kurz nacheinander. Vielleicht wird dieser Umstand auf der Abb. 2.1.1-1 nicht so deutlich; deshalb erklären wir es hier. Die Richtung läßt sich nun dadurch feststellen, indem man beobachtet, welche der beiden Lichtschranken zuerst unterbrochen wird. Deshalb sind auch die Impulse beider Lichtschranken nach außen geführt, insgesamt also vier. Entsprechend ihrer Bedeutung tragen sie denn auch die Namen XA, XB, YA und YB.
Die beiden Kontakte, die Sie auf dem Bild sehen, sollen die Drucktasten darstellen.
Der große Kasten auf dem Bild ist ein vierfacher Operations-

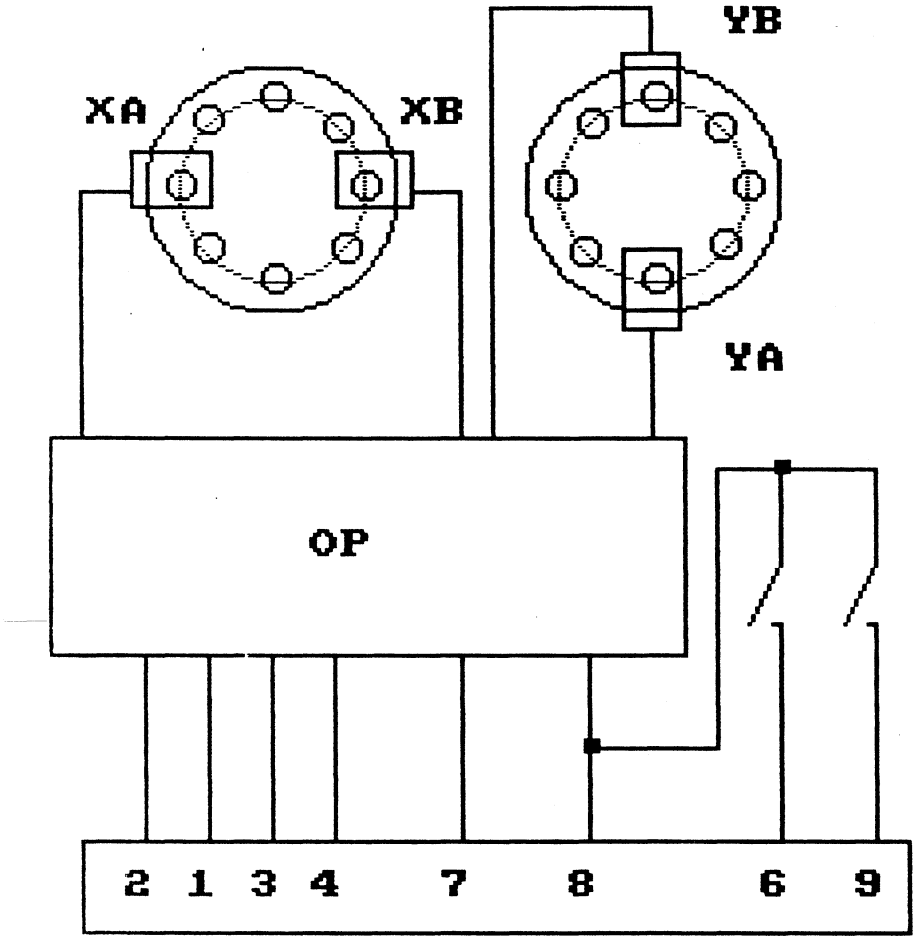


Abb. 2.1.1-1

verstärker, der die recht buckeligen Lichtschrankenimpulse in schöne Rechtecksignale formt.

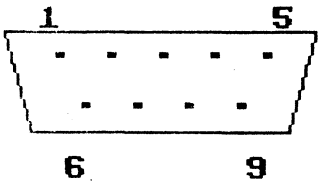
Unten sehen Sie die Belegung der Control-Ports am Computer so abgebildet, wie Sie sie sehen, wenn Sie von außen darauf schauen. Hier die Bedeutung, hinter dem Schrägstrich die Bezeichnung, wenn ein Joystick angeschlossen ist, in Klammern die Pin-Nummer des Tastatur-Steckverbinders:

Port 0

1	XB/UP	(K12)
2	XA/DOWN	(K10)
3	YA/LEFT	(K9)
4	YB/RIGHT	(K8)
6	LEFT BUTTON/FIRE	(K11)
7	+5V	(K13)
8	GND	(K1)
9	RIGHT BUTTON	(K6)

Port 1

1	UP	(K7)
2	DOWN	(K5)
3	LEFT	(K4)
4	RIGHT	(K3)
5	Port 0 enable	(K17)
6	FIRE	(K6)
7	+5V	(K13)
8	GND	(K1)



2.1.2 KEYBOARD-KOMMANDOS

Der Tastaturprozessor 'versteht' etliche Kommandos, die sich darauf beziehen, wie z.B. die Maus zu behandeln ist. Man kann die Uhrzeit setzen, den internen Speicher auslesen usw. Ein Beispiel für die Anwendung finden Sie im Assemblerlisting hinter Kommando $21.
Die 'normale' Tätigkeit des Prozessors besteht ja darin, die Tastatur zu überwachen und jeden Tastendruck zu melden. Das geschieht in der Form, daß die Tastennummer beim Drücken der Taste ausgegeben wird. Beim Loslassen der Taste wird ebenfalls die Tastennummer gesendet, allerdings mit gesetztem Bit 7. Daraus folgt, daß es keine größere Tastennummer als 127 geben darf. *Die Zuordnung der Tastennummern zu den Tasten finden Sie am Ende dieses Kapitels in Abb. 2.1.2-1.* In Wirklichkeit können diese Nummern aber nur bis 117 gehen, da nämlich Werte ab $F6 für andere Zwecke reserviert sind. Es muß nämlich eine Möglichkeit geben, dem Hauptrechner auch andere Informationen als nur Tastennummern zu übergeben, z.B. die Uhrzeit oder die aktuelle Position der Maus. Da dies nicht in einem einzigen Byte abgehandelt werden kann, sondern nur in einem sog. Paket, kennzeichnen die Bytes ab $F6 den Start eines solchen Pakets. Welches Kennbyte (sog. Header) welchem Paket voransteht, ist bei den einzelnen Kommandos beschrieben.

Ein Befehl an den Tastaturprozessor besteht aus dem Befehlscode (ein Byte) und evtl. notwendigen Parametern. Die folgende Beschreibung ist nach Befehlsbytes sortiert.

$07
Die Reaktion auf das Drücken einer der beiden Maus-Tasten wird eingestellt.
Es wird ein Parameterbyte mit folgendem Aufbau verlangt:

```
Bit 0  =1: Beim Drücken einer Maus-Taste wird  die  absolute
       Position gemeldet. Bit 2 muß =0 sein.
Bit 1  =1: Beim Loslassen einer Maus-Taste wird die absolute
       Position gemeldet. Bit 2 muß =0 sein.
Bit 2  =1: Die  Maus-Tasten  werden   wie   normale   Tasten
```

```
        behandelt. Die linke Taste wird als Tastennummer  $74
        gemeldet, die rechte als $75.
Die Bits 3-7 müssen immer =0 sein.
```

$08

Es soll künftig die relative Maus-Position gemeldet werden. Dieser Befehl veranlaßt den Tastaturprozessor, automatisch die relative Position (die Distanz zur vorigen Position) zu melden, wenn eine Bewegung der Maus erfolgt ist. Eine Bewegung gilt dann als gegeben, wenn die Anzahl der Taktscheibenimpulse eine einstellbare Schwelle erreicht hat, die Maus also um ein bestimmtes Minimum bewegt wurde. Siehe auch $0B.

Ein relatives Maus-Paket sieht folgendermaßen aus:

```
1 Byte  Header im Bereich $F8-$FB. Die beiden untersten Bits
        des Headers geben den Zustand der beiden Maus-Tasten
        wieder.
1 Byte  Relative X-Position (vorzeichenbehaftet!)
1 Byte  Relative Y-Position (vorzeichenbehaftet!)
```

Wenn sich zwischen zwei Paketen die relative Position einer Achse so stark verändert hat, daß die Distanz in einem Byte nicht mehr darstellbar wäre, wird automatisch ein weiteres Paket erzeugt, welches den Rest überträgt.

$09

Es soll künftig die absolute Maus-Position gemeldet werden. Mit diesem Befehl werden auch die Koordinatenmaxima eingestellt, die höchstens zu melden sind. Gleichzeitig werden die internen Koordinatenzeiger auf Null gesetzt. Folgende Parameter sind verlangt:

```
1 Wort  Maximale X-Koordinate
1 Wort  Maximale Y-Koordinate
```

Mausbewegungen unter den Nullpunkt oder über die Maxima werden nicht gemeldet.

$0A
Mit diesem Befehl ist es möglich, anstelle der Koordinaten die Tastennummern der Cursor-Tasten zu erhalten. Dem Betriebssystem erscheint dann eine Mausbewegung derart, als ob die entsprechenden Cursor-Tasten gedrückt worden wären. An Parametern sind notwendig:

1 Byte Anzahl der Taktimpulse (X), nach denen jeweils einmal die Tastennummer für Cursor links (bzw. rechts) abgesetzt wird.

1 Byte Anzahl der Taktimpulse (Y), nach denen jeweils einmal die Tastennummer für Cursor auf (oder ab) abgesetzt wird.

$0B
Hiermit wird die Triggerschwelle gesetzt, oberhalb der Bewegungen erstmals gemeldet werden. Es verstreicht also eine bestimmte Anzahl Taktimpulse, bevor ein Paket abgesetzt wird. Das funktioniert nur in der relativen Betriebsart. Folgende sind die Parameter:

1 Byte Schwelle in X-Richtung

1 Byte Schwelle in Y-Richtung

$0C
Maus skalieren. Hiermit wird bestimmt, alle wieviel Taktimpulse die Koordinatenzähler um 1 verändert werden. Dieser Befehl ist nur im absoluten Modus gültig. Folgende Parameter werden verlangt:

1 Byte X-Skalierung

1 Byte Y-Skalierung

$0D
Absolute Mausposition abfragen. Parameter werden keine verlangt, jedoch wird ein Paket folgenden Formats gesendet:

1 Byte Header = $F7

1 Byte Tastenstatus

Bit 0 =1: Rechte Taste ist seit der letzten Abfrage gedrückt.

Bit 1 =1: Rechte Taste ist seit der letzten Abfrage

```
              nicht gedrückt.
        Bit 2 =1: Linke Taste ist seit der letzten Abfrage
              gedrückt.
        Bit 3 =1: Linke Taste ist seit der letzten Abfrage
              nicht gedrückt

        Aus dieser eigenartigen Zuordnung kann man
        schließen, daß eine Taste ihren Zustand seit der
        letzten Abfrage (evtl. mehrfach) verändert hat, wenn
        die beiden zugehörigen Bits =0 sind.
1 Wort  Absolute X-Koordinate
1 Wort  Absolute Y-Koordinate
```

$0E
Interne Koordinatenzähler setzen. Es werden folgende Parameter verlangt:

```
1 Byte  =0 als Füllbyte
1 Wort  X-Koordinate
1 Wort  Y-Koordinate
```

$0F
Der Ursprung für die Y-Koordinaten ist unten (dem Anwender am nächsten).

$10
Der Ursprung für die Y-Koordinaten ist oben.

$11
Die Datenübertragung zum Hauptrechner wird wieder aufgenommen (siehe $13).
Auch jeder andere Befehl (außer $13) bringt den Transfer wieder in Gang.

$12
Maus abschalten. Jeder Maus-Modus-Befehl ($08, $09, $0A) schaltet die Maus wieder ein. Ist die Maus im Modus $0A, hat dieser Befehl keine Auswirkung.

$13
Datenübertragung zum Hauptrechner anhalten.
ACHTUNG: *Soweit es der kleine Puffer des 6301 erlaubt, werden Mausbewegungen oder Tastendrücke noch gespeichert. Aktionen über die Pufferkapazität hinaus gehen verloren.*

$14
Jede Joystick-Bewegung wird automatisch gemeldet. Die gesendeten Pakete haben folgendes Format:

1 Byte Header = $FE oder $FF für Joystick 0/1.
1 Byte Bits 0-3 für die Stellung (jede Richtung ein Bit), Bit 7 für den Knopf.

$15
Beendet den automatischen Meldemodus für den Joystick. Bei Bedarf muß ein Paket mit $16 angefordert werden.

$16
Joystick-Abfrage. Nach diesem Befehl sendet die Tastatur ein Paket wie oben beschrieben.

$17
Joystick-Dauermeldung. Ein Parameter wird verlangt.

1 Byte Zeit zwischen zwei Meldungen in 1/100sec.

Ab jetzt werden ständig (so lange kein anderer Modus eigestellt wird) Pakete folgenden Formats gesendet:

1 Byte Bit 0 für den Knopf von Joystick 1, Bit 1 für den von Joystick 0.
1 Byte Bits 0-3 für die Stellung von Joystick 1, Bits 4-7 für die Stellung des anderen Joystick.

ACHTUNG: *Der Abtastabstand sollte nicht kürzer gewählt werden, als der Übertragungskanal benötigt, die beiden Bytes des Paketes abzusetzen.*

$18
Feuerknopf-Dauermeldung. Es wird ständig der Zustand des Knopfes von Joystick 1(!) abgefragt und das Ergebnis, zu einem Byte zusammengepackt, gemeldet. Das heißt, daß in einem Meldebyte acht Abfragen enthalten sind, wobei Bit 7 als zeitlich am ehesten eingetreten zu betrachten ist. Der Tastaturrechner ermittelt die Zeit, die zwischen der Abholung der Bytes durch den Hauptrechner liegt. Diese Zeit wird in acht gleiche Intervalle eingeteilt, in denen der Knopf abgefragt wird. Deshalb sollte die Abholung so gleichmäßig wie möglich erfolgen.
Dieser Modus bleibt bis zur Übermittlung eines anderen Befehls erhalten.

$19
Cursortasten-Simulationsmodus für Joystick 0(!). Die jeweilige Stellung des Joystick wird dem Hauptrechner so gemeldet, als ob die entsprechenden Cursortasten (mehrfach) gedrückt worden wären. Um nicht bei den folgenden Parametern nicht immer nahezu dasselbe erklären zu müssen, hier das Wesentliche: Alle Zeiten werden in 1/10sec verstanden. R gibt die Zeit an, bis zu deren Erreichen Cursorklicks in Abständen von T gesendet werden. Danach beträgt der Abstand V. Wenn R =0, ist nur V für den Abstand zuständig. Der Mechanismus tritt natürlich nur dann in Aktion, wenn der Joystick länger als T, bzw. R in derselben Stellung ist.

```
1 Byte  RX
1 Byte  RY
1 Byte  TX
1 Byte  TY
1 Byte  VX
1 Byte  VY
```

$1A
Joysticks abschalten. Jeder andere Joystick-Befehl setzt die Abfrage wieder in Gang.

$1B

Uhrzeit setzen. Mit diesem Befehl wird die interne Echtzeituhr des Tastaturprozessors gestellt. Die zu übergebenden Werte sind sämtlich als gepackte BCD-Zahl zu verstehen, d.h. eine Ziffer (0-9) pro Halbbyte, was eine zweistellige Dezimalzahl pro Byte ergibt. Folgende Parameter sind notwendig:

```
1 Byte  Jahreszahl zweistellig (85, 86 usw.).
1 Byte  Monat zweistellig (12, 01 usw.).
1 Byte  Tag zweistellig (31, 01, 02 usw.).
1 Byte  Stunde zweistellig.
1 Byte  Minute zweistellig.
1 Byte  Sekunde zweistellig.
```

Jedes Halbbyte, welches keine gültige BCD-Ziffer enthält (z.B. F), wird ignoriert. Auf diese Weise ist es möglich, auch nur Teile von Datum oder Uhrzeit zu ändern.

$1C

Uhrzeit abfragen. Aufgrund dieses Befehls liefert die Tastatur ein Paket zurück, das im Aufbau dem oben beschriebenen Format entspricht. Lediglich ein Header ist natürlich noch vorangesetzt. Er hat den Wert $FC.

$20

Speicher laden. Der interne Speicher des Tastaturprozessors (natürlich ist nur der Ram im Bereich $80 bis $FF sinnvoll) kann mit diesem Befehl beschrieben werden. Uns ist unklar, welchen Sinn das ergeben soll, denn nach den bisherigen Untersuchungen (wir haben das Betriebssystem des 6301 bereits disassembliert) ist kein freier Raum verfügbar, den man nach Belieben manipulieren könnte. Vielleicht lassen sich aber auf diesem Wege bestimmte Parameter (nach genauer Kenntnis des Aufbaues) beeinflussen, die auf 'legalem' Wege nicht zugänglich sind. Hier nun die Parameter:

```
1 Wort  Gewünschte Startadresse
1 Byte  Anzahl der Bytes (max. 128)
Datenbytes (entsprechend der Anzahl)
```

Der Abstand, in dem die Datenbytes übergeben werden, muß

kleiner als 20msec sein.

$21
Speicher lesen. Dieser Befehl bildet das Gegenstück zu $20. An Parametern werden verlangt:

1 Wort Adresse, ab der gelesen werden soll.

Es wird ein Paket folgenden Formates zurückgesendet:

1 Byte Header 1 =$F6. Es handelt sich hierbei um den sog. Status-Header, der allen Paketen vorangestellt wird, die irgendwelche Betriebszustände des Tastaturprozessors beeinhalten. Zu den allgemeinen Status-Meldungen kommen wir jedoch noch.

1 Byte Header 2 =$20 als Kennzeichen, daß dieses Status-Paket den Speicherinhalt transportiert.

6 Bytes Speicherinhalt, beginnend mit der im Befehl angegebenen Adresse.

Zur Auflockerung ein kleines Programm, mit dem wir das Rom des 6301 ausgelesen und auf Drucker ausgegeben haben. Hieran sehen Sie auch, wie Sie überhaupt an die Status-Pakete der Tastatur herankommen. Vom Betriebssystem des 68000 werden diese nämlich normalerweise weggeworfen. Über die verwendeten GEMDOS- und XBIOS-Aufrufe informieren Sie sich bitte im Kapitel 3.1.

```
 1                          prt     equ     0
 2                          chout   equ     3
 3                          gemdos  equ     1
 4                          bios    equ     13
 5                          xbios   equ     14
 6                          stvec   equ     12
 7                          rdm     equ     $21
 8                          wrkbd   equ     25
 9                          kbdvec  equ     34
10                          term    equ     0
11                          start:
12 00000000 3F3C0022                move.w  #kbdvec,-(a7)
```

```
13 00000004 4E4E                        trap    #xbios
14 00000006 548F                        addq.l  #2,a7
15 00000008 41F900000000                lea     0,a0
16 0000000E 43F9000000D6                lea     keyin,a1
17 00000014 23C000000104                move.l  d0,savea
18 0000001A 23F0000C00000100            move.l  stvec(a0,d0),save
19 00000022 2189000C                    move.l  a1,stvec(a0,d0)
20 00000026 383CF000                    move.w  #$f000,d4
21                              loop:
22 0000002A 33C40000010A                move.w  d4,tbuf+1
23 00000030 61000084                    bsr     keyout
24                              wait:
25 00000034 0C390000000000F8            cmpi.b  #0,rbuf
26 0000003C 67F6                        beq     wait
27 0000003E 3C3C0006                    move.w  #6,d6
28 00000042 610A                        bsr     bufout
29 00000044 5C44                        addq.w  #6,d4
30 00000046 0C44FFFF                    cmpi.w  #$ffff,d4
31 0000004A 6DDE                        blt     loop
32 0000004C 6052                        bra     exit
33                              bufout:
34 0000004E 49F9000000F9                lea     rbuf+1,a4
35                              bytout:
36 00000054 101C                        move.b  (a4)+,d0
37 00000056 6106                        bsr     hexout
38 00000058 5306                        subq.b  #1,d6
39 0000005A 66F8                        bne     bytout
40 0000005C 4E75                        rts
41                              hexout:
42 0000005E 3240                        movea.w d0,a1
43 00000060 E808                        lsr.b   #4,d0
44 00000062 02800000000F                andi.l  #15,d0
45 00000068 47F9000000E8                lea     table,a3
46 0000006E 14330000                    move.b  0(a3,d0),d2
47 00000072 E14A                        lsl.w   #8,d2
48 00000074 3009                        move.w  a1,d0
49 00000076 02800000000F                andi.l  #15,d0
50 0000007C 14330000                    move.b  0(a3,d0),d2
51 00000080 3002                        move.w  d2,d0
```

```
52 00000082 3F02                        move.w  d2,-(a7)
53 00000084 E048                        lsr.w   #8,d0
54 00000086 6108                        bsr     chrout
55 00000088 301F                        move.w  (a7)+,d0
56 0000008A 6104                        bsr     chrout
57 0000008C 103C0020                    move.b  #" ",d0
58                              chrout:
59 00000090 3F00                        move.w  d0,-(a7)
60 00000092 3F3C0000                    move.w  #prt,-(a7)
61 00000096 3F3C0003                    move.w  #chout,-(a7)
62 0000009A 4E4D                        trap    #bios
63 0000009C 5C8F                        addq.l  #6,a7
64 0000009E 4E75                        rts
65                              exit:
66 000000A0 307900000104                movea   savea,a0
67 000000A6 203900000100                move.l  save,d0
68 000000AC 2140000C                    move.l  d0,stvec(a0)
69 000000B0 3F3C0000                    move.w  #term,-(a7)
70 000000B4 4E41                        trap    #gemdos
71                              keyout:
72 000000B6 13FC0000000000F8            move.b  #0,rbuf
73 000000BE 487900000109                pea     tbuf
74 000000C4 3F3C0002                    move.w  #2,-(a7)
75 000000C8 3F3C0019                    move.w  #wrkbd,-(a7)
76 000000CC 4E4E                        trap    #xbios
77 000000CE DFFC00000008                adda.l  #8,a7
78 000000D4 4E75                        rts
79                              keyin:
80 000000D6 103C0008                    move.b  #8,d0
81 000000DA 43F9000000F8                lea     rbuf,a1
82                              repin:
83 000000E0 12D8                        move.b  (a0)+,(a1)+
84 000000E2 5300                        subq.b  #1,d0
85 000000E4 66FA                        bne     repin
86 000000E6 4E75                        rts
87                              table:
88 000000E8 3031323334353637            dc.b    "0123456789ABCDEF"
88 000000F0 3839414243444546
89 000000F8                     rbuf    ds.b    8
```

```
90 00000100                    save    ds.l    1
91 00000104                    savea   ds.l    1
92 00000108                    dummy   ds.b    1
93 00000109 21                 tbuf    dc.b    rdm
94 0000010A                            ds.b    2
95 0000010C                            .end
```

$22
Routine ausführen. Mit diesem Befehl können Sie ein Unterprogramm im 6301 ausführen. Natürlich müssen Sie genau wissen, was es macht und wo es steht, sofern Sie es nicht selbst mittels $20 in den Ram transferiert haben (wenn Sie dort überhaupt noch ein freies Plätzchen gefunden haben). Als Parameter wird nur

```
1 Wort  Startadresse
```

verlangt.

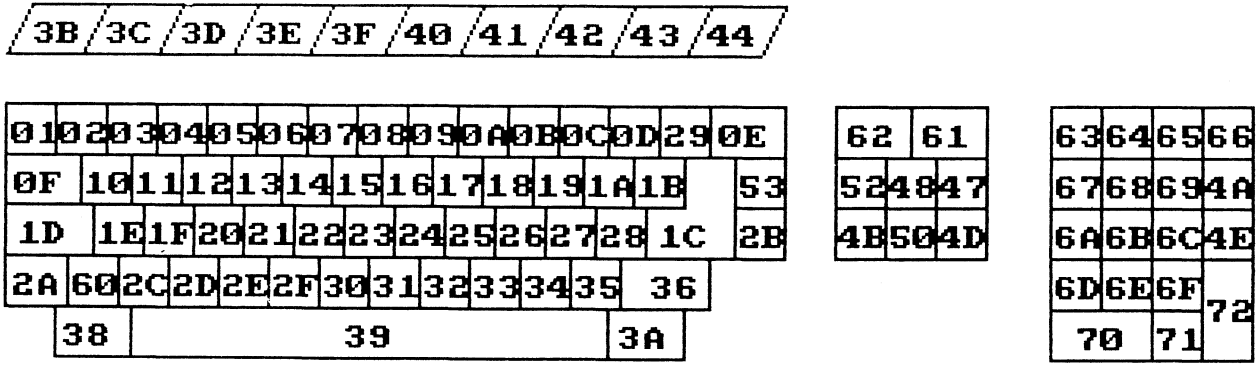
Status-Meldungen
Sie können jederzeit die Betriebsparameter der Tastatur abfragen, indem Sie einfach zu dem Befehlsbyte, mit dem Sie die Betriebsart (deren Parameter Sie jetzt wissen möchten) einstellen würden, $80 aufaddieren. Sie erhalten dann ein Status-Paket zurück (Header =$F6), dessen Format exakt den Parametern entspricht, die auch zur Einstellung der Betriebsart notwendig gewesen wären. Puh....
Ein Beispiel macht das sicher deutlicher:
Sie wollen wissen, wie die Maus skaliert ist. Also senden Sie als Kommando den Wert $8C (da ja $0C die Skalierung setzt). Sie erhalten folgendes zurück:

```
1 Byte  Status-Header =$F6
1 Byte  X-Skalierung
1 Byte  Y-Skalierung
```

also dasselbe Format, das beim Befehl $0C notwendig gewesen wäre. Bei Befehlen, die keine Parameter verlangen, erhalten Sie den auslösenden Befehl als solchen zurück. Sie möchten z.B. wissen, in welcher Betriebsart sich der Joystick befindet ($14 oder $15). Sie schicken also den Wert $94 (oder $95, das spielt keine Rolle). Als Status-Paket erhalten Sie, außer dem Header, entweder $14 oder $15

zurück, je nach dem, in welcher Betriebsart sich der Joystick-Handler befand.
Erlaubte Status-Abfragen sind: $87, $88, $89, $8A, $8B, $8C, $8F, $90, $92, $94, $95, $99 und $9A.

Zum Schluß noch ein Tip für Freaks, denen die Funktionen der Tastatur zu mager sind und noch mehr 'Intelligenz' hineinpacken wollen:
Den Prozessor 6301 gibt es auch in einer sog. Piggy-Pack-Version und heißt dann 63P01 (Hitachi). Diese unterscheidet sich dadurch, daß sie kein Rom eingebaut hat, sondern auf ihrer Oberseite eine Fassung für Eproms vom Typ 2732 oder 2764 (8K!) trägt. So könnten Sie Ihre eigenen Vorstellungen realisieren und z.B. die beiden Joystick-Anschlüsse zu einer Universal-4-Bit-I/O umfunktionieren, wozu Sie dann auch den Befehlsvorrat erweitern, um die neuen Funktionen auch vom XBIOS zu erreichen.


3B 3C 3D 3E 3F 40 41 42 43 44
01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 29 0E
0F 10 11 12 13 14 15 16 17 18 19 1A 1B 53
1D 1E 1F 20 21 22 23 24 25 26 27 28 1C 2B
2A 60 2C 2D 2E 2F 30 31 32 33 34 35 36
38 39 3A
62 61
52 48 47
4B 50 4D
63 64 65 66
67 68 69 4A
6A 6B 6C 4E
6D 6E 6F 72
70 71

2.2 Die Video-Buchse

Ohne sie geht nichts. Sie würden im wahrsten Sinne des Wortes im Dunkeln tappen. Auffallend sind die vielen Pins der Verbindung. Sicher werden zum Anschluß eines RGB-Monitors mehr Leitungen benötigt als für einen monochromen Bildschirm, aber sieben täten es auch. So hat es denn mit den übrigen Leitungen auch seine besondere Bewandtnis. Auf Abb. 2.2-1 finden Sie ein Blockschaltbild, aus dem Sie sehen können, wie der Video-Anschluß ins System eingebunden ist. Die Numerierung der Pins ist in der Abbildung auf der nächsten Seite so angegeben, wie Sie sie sehen, wenn Sie von außen in die Buchse hineinschauen. Hier die Pinbelegung:

1 AUDIO OUT. Dieser Anschluß stammt von dem Verstärker, der hinter die Ausgänge des Sound-Chip geschaltet ist. Hier läßt sich notfalls auch ein hochohmiger (600 Ohm) Kopfhörer anschließen, falls Sie keinen Original-Monitor verwenden.

2 Nicht belegt.

3 GPO. General Purpose Output. Dies ist ein Bit zu Ihrer freien Verfügung. Die Leitung führt TTL-Pegel und stammt von I/O-Port A Bit 6 des Sound-Chip.

4 MONOCHROME DETECT. Ist diese Leitung, die zum I7-Eingang des MFP 68901 führt, =low, geht der Rechner in den hochauflösenden Monochrome-Modus. Wechselt während des Betriebes der Zustand der Leitung, wird ein Kaltstart ausgelöst.

5 AUDIO IN führt zum Eingang des unter 1 beschriebenen Verstärkers und wird dort mit dem Ausgang des Sound-Chip gemischt.

6 GREEN ist der analoge Grün-Ausgang des Video-Shifters.

7 RED. Rot-Ausgang.

8 GROUND. Masseanschluß.

9 HORIZONTAL SYNC ist für den horizontale Strahlrücklauf des Monitors verantwortlich.

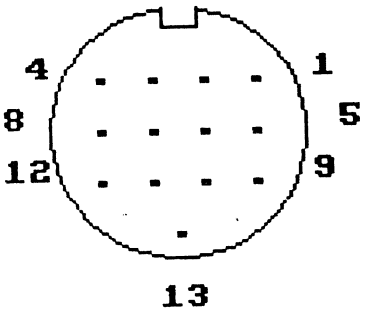
10 BLUE ist der analoge Blau-Ausgang des Video-Shifters.

11 MONOCHROME versorgt einen einfarbigen Monitor mit dem Helligkeitssignal.

12 VERTICAL SYNC sorgt für den Strahlrücklauf am Bildende.

13 GROUND.

Ein Tip für Bastler:
Ein passender Stecker ist im Handel nicht erhältlich. Wollen Sie einen Stecker zum Anschluß systemfremder Monitore herstellen, nehmen Sie einfach ein Stück Lochrasterplatine, in die sie Stifte einlöten, denn glücklicherweise sind die Pins im 1/10"-Raster angeordnet. Pin 13 fällt zwar aus der Reihe, aber er wird ohnehin nicht benötigt, da die Masse auch an Pin 8 zur Verfügung steht.



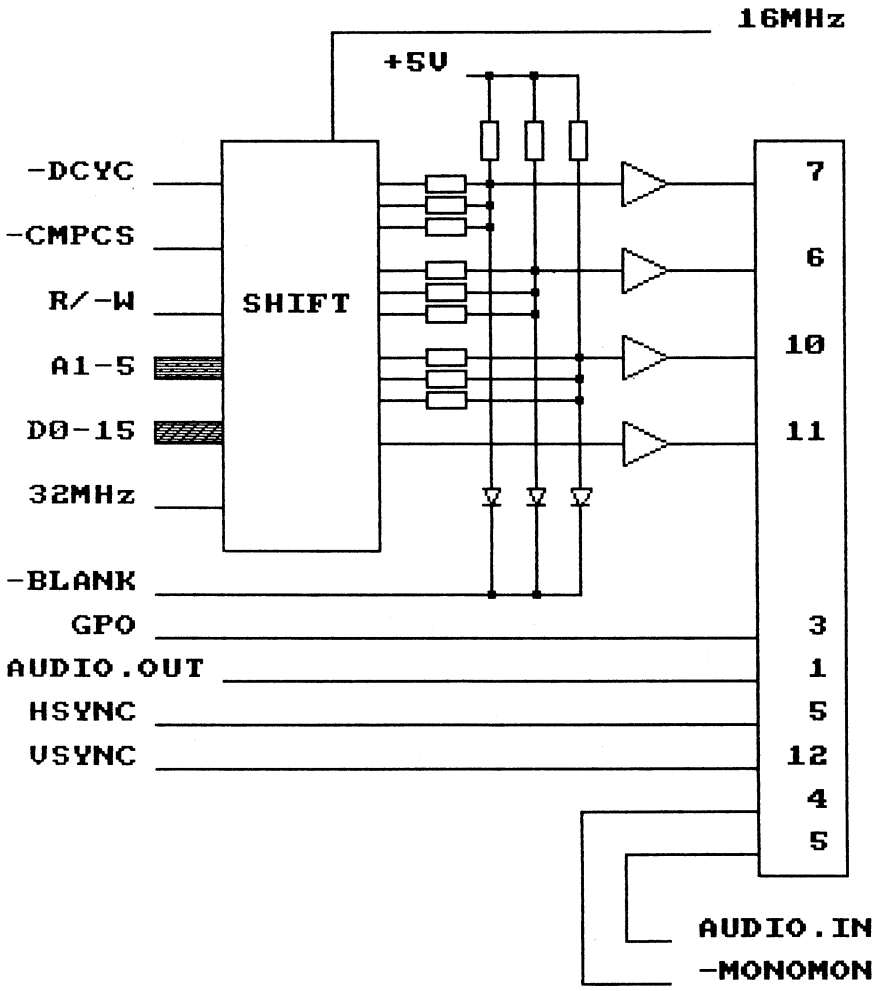

16MHz
+5V
-DCYC
-CMPCS
R/-W
A1-5
D0-15
32MHz
SHIFT
-BLANK
GPO
AUDIO.OUT
HSYNC
VSYNC
7
6
10
11
3
1
5
12
4
5
AUDIO.IN
-MONOMON

Abb. 2.2-1

2.3 Der Centronics-Anschluß

Auch wenn er eher nach V24 aussieht: An diese Buchse kann ein gewöhnlicher Centronics-Parallel-Drucker angeschlossen werden, vorausgesetzt, man verfügt über ein passendes Kabel. Wie Sie auf Abb. 2.3-1 sehen können, ist die Einbindung in das System etwas ungewöhnlich. Man hat nämlich zur Bedienung der Datenleitungen und des Strobe den Universal-Port des Sound-Chip herangezogen. So finden Sie diesen auch auf dem Bildchen, wobei Sie die übrigen Leitungen, die in diesem Abschnitt nicht beschrieben werden, nicht stören sollen. Sie gehören zur Floppy- und V24-Schnittstelle und sind dort behandelt.
Hier nun die Pinbeschreibung:

1 -STROBE zeigt dem angeschlossenen Gerät mit einem Low-Impuls die Gültigkeit des an den Datenleitungen anliegenden Byte an.

2-9 DATEN

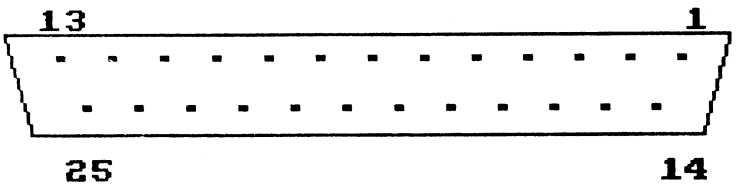
11 BUSY wird vom Drucker immer dann auf high gelegt, wenn er nicht in der Lage ist, weitere Daten zu empfangen. Das kann verschiedene Ursachen haben. Meistens ist nur der Puffer voll oder das Gerät ist offline.

Die Pins 18-25 führen GROUND.

Alle übrigen Pins sind nicht belegt.

Ein Tip zur Herstellung eines passenden Kabels:
Besorgen Sie sich sog. Flachkabel-Quetschverbinder. Sie benötigen einen vom Typ D25-Sub, einen Cinch 36-pol (3M, AMP) und die entsprechende Länge Flachbandleitung 25-polig. Sie quetschen nun die Stecker so an das Kabel (das geht im Schraubstock), daß die Pins 1 beider Seiten (in der Regel markiert) auf der gleichen Leitung liegen. Die übrigen Verbindungen stimmen dann automatisch. Auf der Druckerseite bleiben dabei natürlich etliche Pins frei.

ACHTUNG:
Bei 'normalen' ASCII-Druckern werden Sie vermutlich die Umlaute nicht darstellen können. Abhilfe schafft ein IBM-kompatibles Gerät oder die Änderung der Tastaturtabelle im Betriebssystem.



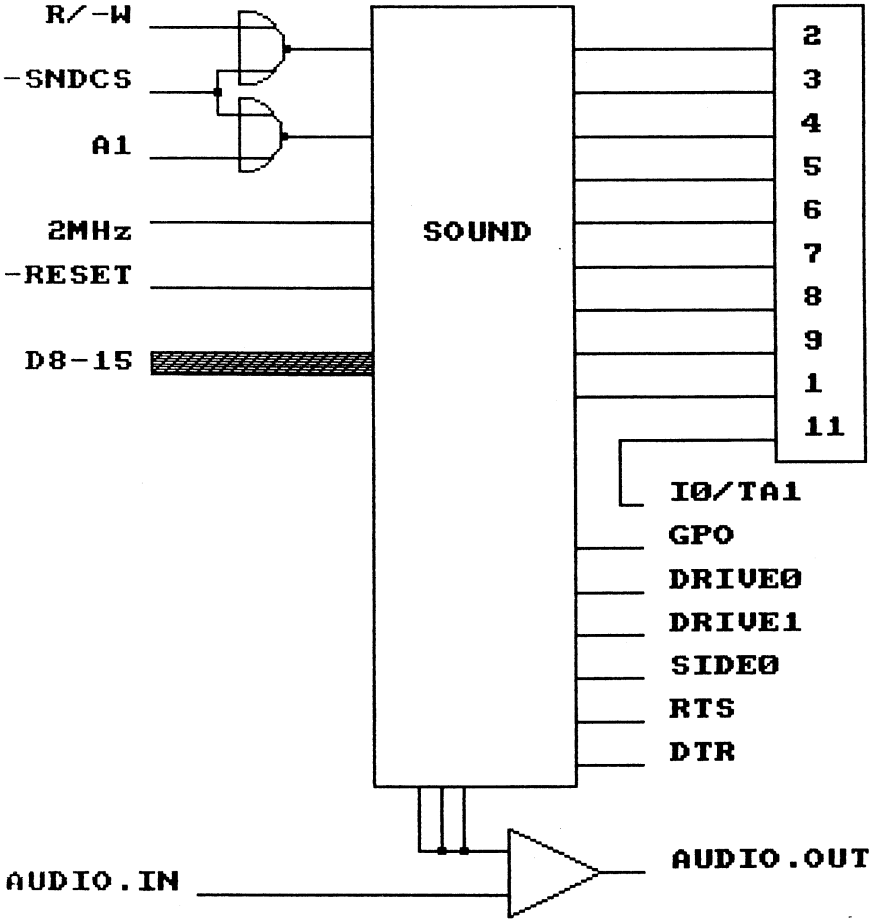

R/-W
-SNDCS
A1
2MHz
-RESET
D8-15
SOUND
2
3
4
5
6
7
8
9
1
11
I0/TA1
GPO
DRIVE0
DRIVE1
SIDE0
RTS
DTR
AUDIO.IN
AUDIO.OUT

Abb. 2.3-1

2.4 Die RS232-Schnittstelle

Diese Schnittstelle dient gewöhnlich der Kommunikation mit anderen Rechnern, auch über Modem (Akustikkoppler). Seltener wird hier ein Drucker angeschlossen. Beachten Sie hierzu die Beschreibung von Pin 5!

Die Abbildung 2.4-1 zeigt die Einbindung ins Gesamtsystem. Normalerweise haben Sie mit der expliziten Handhabung dieser Schnittstelle nichts zu tun. Man läßt es vom Betriebssystem erledigen. Dennoch fällt auf, daß hier nicht, wie sonst üblich, die Steuerung der Schnittstelle einem speziellen IC (UART) überlassen wird, sondern die Leitungen mehr oder weniger 'zu Fuß' bedient werden. Sicher werden die Daten nicht 'persönlich' bitweise verwurstet. Dazu dient das Schieberegister des MFP. Die Handshake-Leitungen jedoch kommen aus den verschiedensten Ecken. Beachten Sie das bei der nun folgenden Pinbeschreibung.

1 GEHÄUSEMASSE (Abschirmung) wird selten benutzt.

2 TxD Sendedaten

3 RxD Empfangsdaten

4 RTS Sendeanforderung stammt von I/O-Port A Bit 3 des Sound-Chip und wird immer dann high, wenn der Rechner in der Lage ist, ein Byte zu empfangen. Bei Atari wird das Signal nach Empfang eines Byte zunächst low gesetzt, bis das eben empfangene Byte verarbeitet ist.

5 CTS Empfangsbereitschaft eines angeschlossenen Gerätes wird an dem Interrupt-Eingang I2 des MFP abgefragt. Dieses Signal wird im Betriebssystem des Rechners (derzeit noch) falsch behandelt. So ist nur der Anschluß von Geräten möglich, die nach jedem empfangenen Byte (wie der 520ST mit RTS) mit der Leitung 'klappern'. Sie geht nämlich auf den Eingang I2 des MFP und wird dort

unglücklicherweise nur auf die Signalflanke untersucht. Mit dem Anschluß eines Druckers werden Sie deshalb kein Glück haben, da dieser gewöhnlich das Signal CTS so lange high hält, wie sein Puffer noch nicht voll ist. Es entsteht also keine Signalflanke nach jedem Byte, was zur Folge hat, daß nur das erste Byte eines Textes übertragen wird, danach nichts mehr.

7 GND Signalmasse

8 DCD Trägersignal erkannt. Diese Leitung, die zum Interrupt-Eingang I1 des MFP geht, wird normalerweise von einem Modem bedient, welches so dem Rechner anzeigt, daß eine Verbindung mit einem anderen Teilnehmer zustande gekommen ist.

20 DTR Gerät bereit. Mit dieser Leitung wird einem angeschlossenen Gerät signalisiert, daß der Rechner eingeschaltet ist und die Schnittstelle bei Bedarf auch bedient wird. Sie stammt von I/O-Port A Bit 4 des Sound-Chip.

22 RI (eigentlich: Klingelanzeiger) ist eine recht wichtige Unterbrechung an I6 des MFP und wird von einem Modem dazu benutzt, dem Rechner mitzuteilen, daß ein Teilnehmer eine Verbindung wünscht, also 'angerufen' hat.

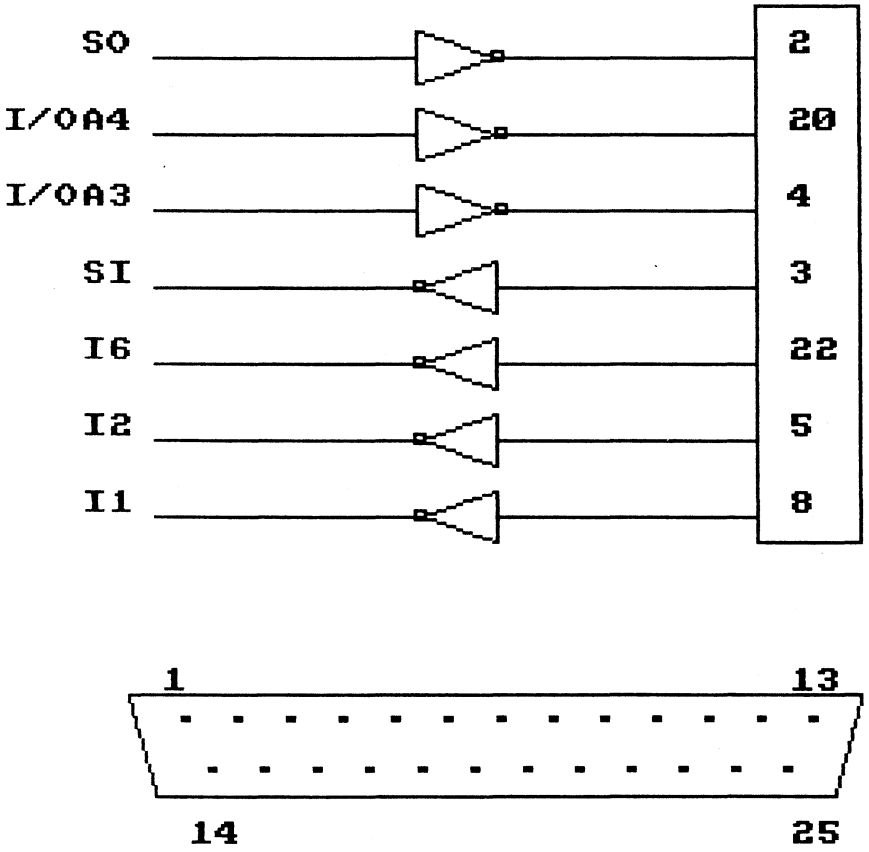


Abb. 2.4-1

2.5 Die MIDI-Anschlüsse

Der Begriff MIDI dürfte nur wenigen unter Ihnen geläufig sein. Er ist eine Abkürzung und steht für Musical Instrument Digital Interface, also, salopp ausgedrückt, eine Schnittstelle für Musikinstrumente.
Nun ist Ihnen sicher klar, daß man nicht etwa einfach eine Flöte dort anschließen kann. Deshalb ein wenig zur Historie. Es waren, wie so oft, wieder einmal die Anwender, diesmal Musikprofis (genauer: Keyboarder, also Künstler, deren Instrument ein Synthesizer ist), die lautstark zur Einigkeit unter den Herstellern aufriefen. Sie empfanden es nämlich als Unding, bei Live-Auftritten ihre oftmals umfangreiche Ausrüstung mit einem wahren Drahtverhau zu verbinden, der auch sehr empfänglich für Störungen im falschen Moment war. Die Idee war, mehrere Synthesizer von einer Klaviatur aus zu bedienen. Nun war bis vor kurzem ein solches Gerät ein Ausbund an Analogtechnik, also mit etlichen Reglern versehen. Auch die Klangerzeugung war (und ist teilweise noch) analog, weshalb sich die Hersteller wenigstens einigten, daß einer Steuerspannungsdifferenz von 1V ein Tonhöhenunterschied von einer Oktave entsprechen sollte. So konnte man mehrere Geräte zwar 'fernspielen', aber noch nicht fernbedienen.
Das änderte sich erst grundlegend, als man zur digitalen Klangerzeugung überging. Hier brauchten nun nicht mehr in dem Umfange wie bisher Knöpfchen gedreht werden, sondern die Knöpfe wurden gedrückt, womit die Grundlage für eine digitale Bedienbarkeit geschaffen war.
Einige Hersteller steckten nun erneut die Köpfe zusammen und heckten ein digitales Interface aus, welches mit seinen Grundbefehlen zwar allgemeingültig ist, aber dennoch zusätzliche Features eines bestimmten Gerätes unterstützt.
Rein physikalisch basiert das Interface auf der Teletype-Basis, also Stromschleifenprinzip, welches sich durch eine hohe Störsicherheit auszeichnet, nur wesentlich schneller. So beträgt denn die Übertragungsrate 31250 Baud, also ebenso viele Bits/sec. Das Datenformat ist mit einem Startbit, acht Datenbits und einem Stopbit festgelegt.

So kann denn zur Steuerung auch ein IC dienen, welches sonst für RS232-Zwecke herangezogen wird. Die Einbindung ins System sehen Sie auf Abbildung 2.5-1.
Logisch ist MIDI ein Vielkanal-System, d.h. es können von einem Master 16 Geräte bedient werden oder ein Gerät mit 16 Stimmen, vereinfacht ausgedrückt. Diese Geräte 'hängen' alle an derselben Leitung (Busprinzip). Zur Identifikation, welches Gerät oder welche Stimme gemeint ist, wird jedem Datenpaket die Kanalnummer vorangestellt. Das Gerät, welches diese Nummer als seine eigene erkennt, wird die gewünschte Aktion ausführen.

Nun mögen Sie sich fragen, was denn ein solches Interface in einem Computer zu suchen hat. Ganz einfach: Ein Computer ist aufgrund seiner hohen Geschwindigkeit und Speicherkapazität hervorragend geeignet, ein ganzes Arsenal von Synthesizern mit Einstellungen oder gar fertigen Melodien (Sequenzer) zu versorgen oder solche von einem Keyboard 'mitzuschneiden' und zu speichern.
Zu diesem Zweck hat der 520ST die Schnittstellen MIDI-IN und MIDI-OUT. Sogar vom XBIOS werden diese Schnittstellen unterstützt, so daß Sie sich mit dem 'Kleinkram' nicht herumprügeln müssen.

Der Schleifenstrom geht über die Pins 4 und 5, und zwar bei MIDI-OUT bei 4 hinaus (+) und bei 5 herein, wenn ein Gerät angeschlossen ist.
Bei MIDI-IN ist es umgekehrt, denn der Strom fließt bei 4 herein und bei 5 wieder hinaus. Dabei durchläuft er einen sog. Optokoppler, mit dem der Rechner galvanisch vom Sender getrennt wird.
Die Empfangsdaten werden nach MIDI-OUT wieder durchgeschleift (Pins 1 und 3), womit auch die Funktion MIDI-THRU, wenn auch nicht ganz normgerecht, implementiert wäre.

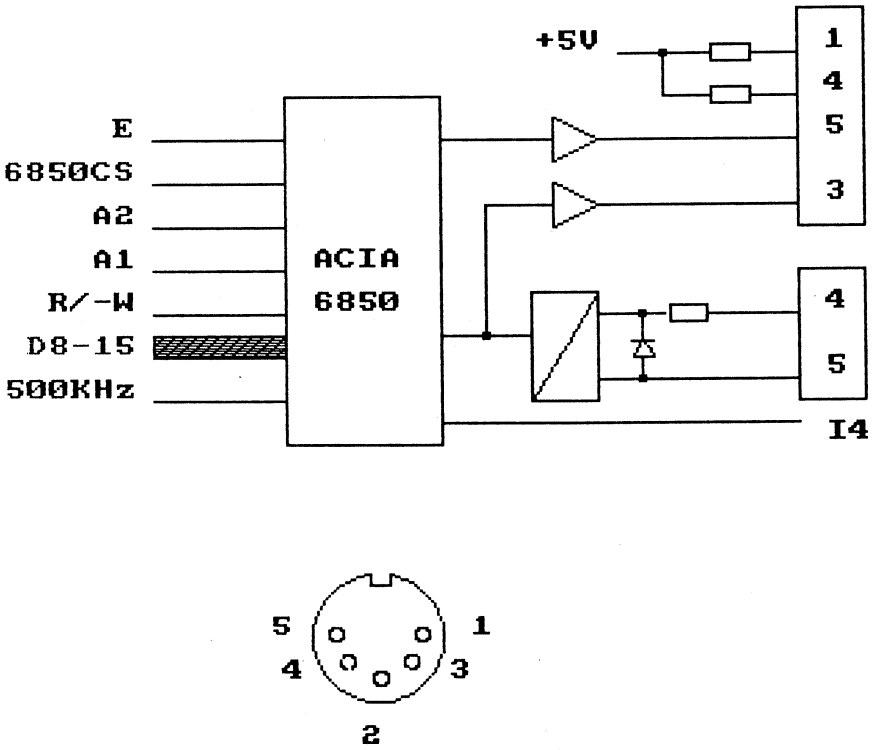


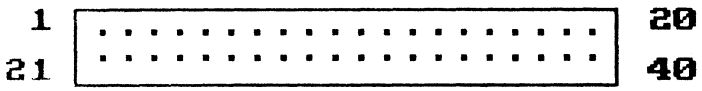
Abb. 2.5-1

2.6 Der Modulschacht

Der Modulschacht dient *ausschließlich* der Aufnahme sog. Rom-Cartridges. Es lassen sich auf diesem Wege maximal 128K (Bytes) im Adressraum $FA0000 bis $FBFFFF ansprechen.
Warum wir die Ausschließlichkeit des Lesezugriffs so sehr betonen hat folgenden Grund:
Wir hätten es für praktisch gehalten, eine Cartridge mit Ram auszurüsten und in diese dann nach dem Systemstart ständig benötigte Programme hineinzuladen, die auch nach einem Reset noch erhalten gewesen wären. Um das zu probieren, haben wir das Signal R/-W nach außen gelegt. Die Erfahrung lehrte jedoch, daß ein Schreibzugriff auf die o.g. Adressen einen Bus-Error erzeugt. Dafür sorgt nämlich der GLUE. Wie Sie sehen, hat man bei Atari nichts dem Zufall überlassen.

+5V	1	21	A8
+5V	2	22	A14
D14	3	23	A7
D15	4	24	A9
D12	5	25	A6
D13	6	26	A10
D10	7	27	A5
D11	8	28	A12
D8	9	29	A11
D9	10	30	A4
D6	11	31	-ROM3
D7	12	32	A3
D4	13	33	-ROM4
D5	14	34	A2
D2	15	35	-UDS
D3	16	36	A1
D0	17	37	-LDS
D1	18	38	GND
A13	19	39	GND
A15	20	40	GND

LAGE:



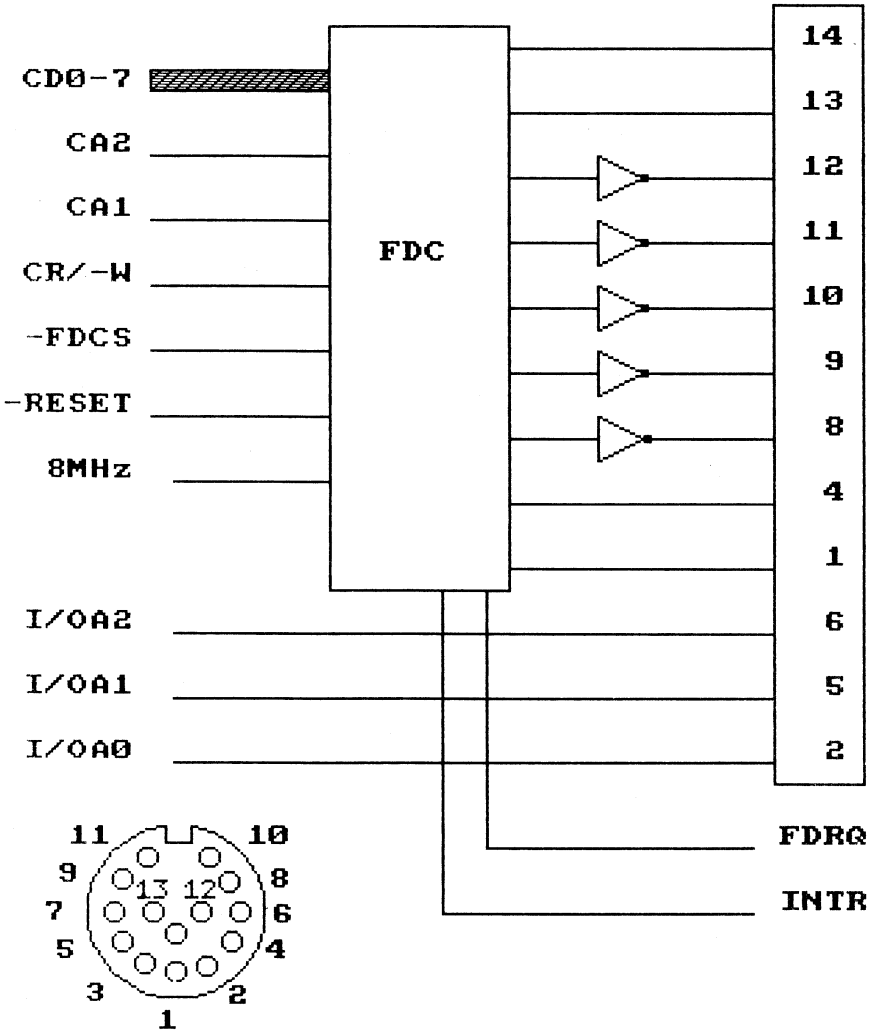
2.7 Die Floppy Disk-Schnittstelle

Zunächst einmal fällt die Schnittstelle für die Floppy-Laufwerke durch die ungewöhnliche Steckverbindung auf. Der 14-polige DIN-Stecker sieht aus wie ein etwas zu groß geratener sogenannter Diodenstecker. An ihm sind alle für den Betrieb von zwei Laufwerken erforderlichen Signale vorhanden.

Die meisten Signale sind Ihnen bereits von der Beschreibung des Floppy Controllers 1772 bekannt, da neun der vorhandenen Anschlüsse direkt oder über einen Buffer mit dem Controller verbunden sind. Nur die Auswahl-Signale Drive Select 1 und Drive Select 2 sowie das Seitenauswahlsignal Side 0 Select werden nicht vom Floppy Controller sondern vom Port A des Sound Chip geliefert.

Belegung des Floppy-Steckers:

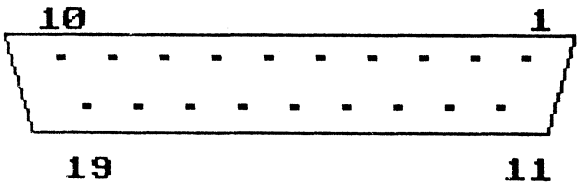
1 READ DATA	8 MOTOR ON
2 SIDE 0 SELECT	9 DIRECTION IN
3 GND	10 STEP
4 INDEX	11 WRITE DATA
5 DRIVE 0 SELECT	12 WRITE GATE
6 DRIVE 1 SELECT	13 TRACK 00
7 GND	14 WRITE PROTECT

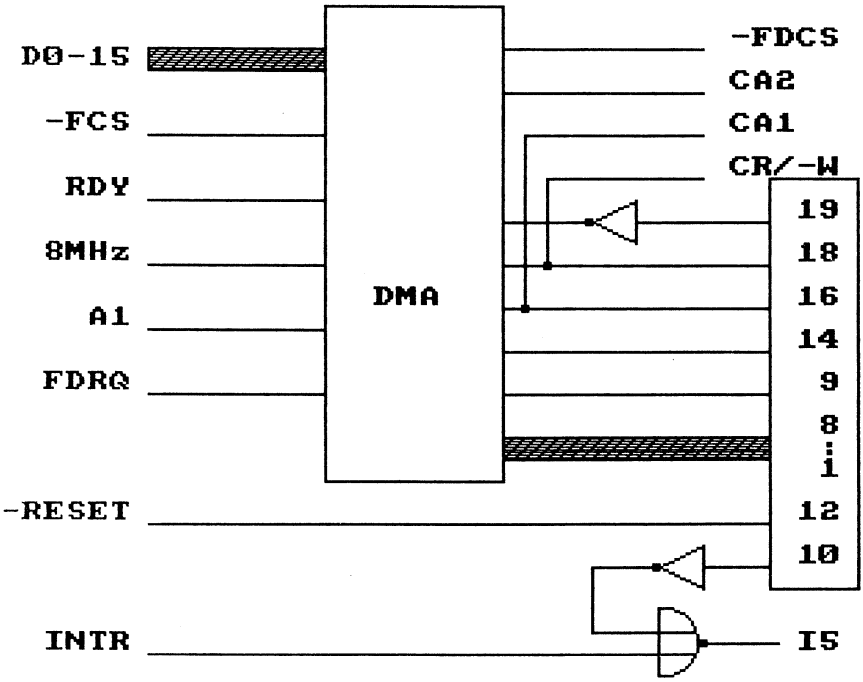

CD0-7
CA2
CA1
CR/-W
-FDCS
-RESET
8MHz
FDC
I/OA2
I/OA1
I/OA0
14
13
12
11
10
9
8
4
1
6
5
2
FDRQ
INTR
11
10
9
8
7
6
5
4
3
2
1
13
12

2.8 Die DMA-Schnittstelle

An diese 19-polige Subminiatur-D-Buchse können bis zu 8 externe Geräte angeschlossen werden. Als Geräte kommen Harddisks, Netzwerke, aber auch Co-Prozessoren in Frage. Die Kommunikation zwischen den externen Geräten und dem ST läuft mit Geschwindigkeiten von bis zu 1 Million Bytes pro Sekunde ab. Leider konnten bis zum Zeitpunkt der Drucklegung keine Experimente mit DMA-Geräten durchgeführt werden. Aus diesem Grund müssen die folgenden Angaben nicht mit hundertprozentiger Sicherheit richtig sein. Allerdings können aus der Anschaltung des FDC an den DMA-Baustein einige Schlüsse gezogen werden.
Die RESET-Leitung an Pin 12 erlaubt das Zurücksetzen der Geräte vom ATARI aus. Wird dieser Anschluß Low, wie dies beim Einschalten des ATARI oder bei der Ausführung des RESET-Befehls geschieht, so werden externe Geräte in einen definierten Einschaltzustand versetzt, ohne das alle Geräte einzeln aus- und wieder eingeschaltet werden müssen.
Da die externen Geräte in der Mehrzahl über ein Controller-IC verfügen werden, muß auch das Signal CS, Chip Select Pin 9, zur Verfügung stehen. In Vrbindung damit ist auch das Signal A1 zu sehen, daß dann wichtig ist, wenn der Controller über mehr als nur ein Register verfügt. Mit diesem Signal kann immerhin zwischen zwei Registern unterschieden werden.
Der Datentransfer läuft über den bidirektionalen Datenbus an den Anschlüssen 1 bis 8.
Die R/W-Leitung am DMA-Bus bestimmt die Richtung des Datentransfers. Der DMA-Chip kann wahlweise Daten auf den Bus schreiben (R/W ist High) oder Daten vom Bus lesen (R/W ist Low). Allerdings können Daten nur auf Anforderung des externen Gerätes gelesen oder geschrieben werden. Die Freigabe eines Transfers wird mit dem Signal DRQ (Pin 19) signalisiert.
Das ACK-Signal an Pin 14 scheint eine rein hardware-mäßige Bestätigung des DRQ-Signals zu sein. Die tatsächliche Bedeutung konnte jedoch nicht überprüft werden.
Das letzte Signal am DMA-Port ist der INT-Eingang. Ein Low

an diesem Anschluß kann einen Interrupt auslösen. Die Harddisk z.B. signalisiert durch ein Low das Ende des Kommandos. Der Interrupt belegt den selben Interrup-Eingang am MFP wie auch der Floppy-Controller. Es ist dies der Eingang IO 5. Dadurch können Floppy und Harddisk nicht gemeinsam Daten übertragen. Dazu wäre jedoch der DMA-Chip auch garnicht in der Lage, da er nur über einen DMA-Kanal verfügt.
Allerdings wird der Interrupt dieses Eingangs im MFP intern disabled, da sowohl die Floppy- wie auch die Harddisk-Routinen das Port-Bit in einer Schleife prüfen, um das Ende des Kommandos festzustellen. Damit wurde wohl die Implementation des Time-Out vereinfacht, der immer dann ausgelöst wird, wenn Floppy oder Harddisk nicht innerhalb einer bestimmten Zeit auf das Kommando reagiert haben.




D0-15
-FCS
RDY
8MHz
A1
FDRQ
-RESET
INTR
DMA
-FDCS
CA2
CA1
CR/-W
19
18
16
14
9
8
1
12
10
I5

3.0 DAS BETRIEBSSYSTEM DES ATARI ST

GEMDOS - was ist das? Ist das auch im ATARI ST? Da soll doch als Betriebssystem TOS drin sein. Oder CP/M68K. Oder was?

Diese Frage ist mit wenigen Worten beantwortet. Das Betriebssystem des ST wird nach dem Chef der Firma ATARI als TOS - Tramiel Operating System - bezeichnet. Dieses TOS hat, im Gegensatz zu früheren Informationen nichts mit CP/M 68K von DIGITAL RESEARCH zu tun. Zwar wurde zu Beginn der Entwicklung des ST das CP/M 68K auf dem ATARI implementiert, diese Entwicklung jedoch später wieder eingestampft, weil CP/M 68K nicht gerade ein Ausbund an Arbeitsgeschwindigkeit darstellt. Selbst ein 68000 mit DMA und 8 MHz würde durch das Betriebssystem regelrecht gebremst.

Anfang 1985 begann bei DIGITAL RESEARCH die Entwicklung eines neuen Betriebssystems für 68000-Rechner, die mit der Benutzeroberfläche GEM ausgerüstet werden sollten. Dieses Betriebssystem bekam den Namen GEMDOS. Und genau dieses GEMDOS stellt den hardwareunabhängigen Teil des TOS dar. Wie auch das mittlerweile recht betagte CP/M besteht das TOS aus aus einem hardwareabhängigen und einem hardwareunabhängigen Teil. Der hardwareabhängige Teil ist das BIOS und das X-BIOS, der hardwareunabhängige Teil wird GEMDOS genannt. Im GEMDOS sind eine Vielzahl von Funktionen eingebaut, über die der Programmierer die wesentlichen Ein/Ausgabe-Funktionen des Rechners steuern kann. So sind Funktionen zur Tastatureingabe, der Ausgabe von Texten auf Bildschirm oder Drucker und dem Betrieb der verschiedenen Schnittstellen vorhanden. In einer weiteren ganz wichtigen Gruppe sind die Funktionen zum File-Handling, also zur logischen Datei- und Diskettenverwaltung enthalten.

3.1 Das GEMDOS

Wenn man einmal die unter GEMDOS vorhandenen Funktionen betrachtet, so wird man eventuell feststellen, daß das Ganze garnicht einmal so neu ist. Alle Funktionen des GEMDOS sind Funktionen des Betriebssystems MS-DOS sehr ähnlich. Auch und gerade die verwendeten Funktionsnummern entsprechen denen des MS-DOS. Allerdings sind nicht alle MS-DOS-Funktionen im GEMDOS implementiert. Besonders im Bereich der Dateiverwaltung wurden im GEMDOS nur die UNIX-kompatiblen Funktionen eingebaut. Die 'alten' blockorientierten Zugriffsmöglichkeiten, die im MS-DOS wohl aus Kompatibilitätsgründen zu CP/M vorhanden waren, fehlen unter GEMDOS. Aber auc spezielle, in der Hardware der MS-DOS-Rechner (8088 Prozessor) begründete Funktionen fehlen hier natürlich.

Ein weiterer wesentlicher Unterschied zu MS-DOS ist die Tatsache, daß bei GEMDOS-Aufrufen wie auch bei BIOS und XBIOS die Funktionsnummer, also die Nummer der gewünschten GEMDOS-Routine und eventuelle Parameter auf dem Stack und nicht in Registern übergeben werden. Für diese Art der Parameterübergabe bietet sich der 68000 geradezu an. Nach dem Aufbau der Parameterliste wird mittels Trap #1 das GEMDOS aufgerufen und die Funktion ausgeführt. Nach dem Aufruf muß der Programmierer den Stack selber wieder in Ordnung bringen, also den durch die Parameter belegten Speicher räumen.
Der grundsätzliche Aufruf der GEMDOS-Funktionen unterscheidet sich von den Aufrufen des BIOS und XBIOS also nur durch die Trapnummer.

Zu allen GEMDOS-Aufrufen muß beachtet werden, daß die Register D0 und A0 in allen Fällen verändert sind. Im Register D0 wird ein eventueller Rückgabewert oder eine Fehlernummer übergeben, A0 zeigt nach dem Aufruf (meistens) auf die Stack-Adresse in der die Funktionsnummer steht. Eventuell noch benötigte Registerwerte in D0 oder A0 müssen Sie also vor einem Aufruf des GEMDOS in Deckung bringen.

$00 TERM

Ein Aufruf des GEMDOS mit der Funktionsnummer 0 beendet das laufende Programm und kehrt an das Programm zurück, von dem es gestartet wurde. Bei Applicationen, also Programmen die vom Desktop aus gestartet wurden, wird die Programmkontrolle wieder dem Desktop übergeben. Wurde das Programm von einem anderen Programm aus aufgerufen, so wird in das aufrufende Programm zurückgesprungen. Dieser Punkt ist vor allem für das Nachladen von Programmteilen wichtig.

```
CLR.W  -(SP)
TRAP   #1
```

$01 CONIN

Mit CONIN kann ein einzelnes Zeichen von der Tastatur geholt werden. Dabei wird solange gewartet, bis ein Zeichen verfügbar ist. Das Ergebnis, also das von der Tastatur gelesene Zeichen wird im Register D0 übergeben. Im Low Byte des Low Word steht der ASCII-Code der gedrückten Taste, im Low Byte des High Word des Registers wird der von der Tastatur gelieferte Scan-Code übergeben. Das ist wichtig bei der Abfrage der Tasten, denen kein ASCII-Code zugeordnet. Das trifft z.B. auf die 10 Funktionstasten oder die Tasten des Cursorblocks zu. Diese Tasten liefern bei Betätigung den ASCII-Wert Null.
Auch kann bei Bedarf an Hand des Scan-Codes unterschieden werden, ob Tasten des Ziffernblocks oder der Haupttastatur gedrückt wurden, da diese Tasten zwar identische ASCII-Codes, jedoch unterschiedliche Scan-Codes liefern.

Scan- und ASCII-Codes der Tasten finden Sie im Anhang.

```
.
MOVE.W #1,-(SP)     * Funktionsnummer auf den Stack
TRAP   #1           * GEMDOS aufrufen
ADDQ.L #2,SP        * Stack korrigieren
.                   * Ergenis in D0
```

$02 CONOUT

CONOUT stellt die einfachste (primitivste) Zeichenausgabe des GEMDOS dar. Mit dieser Funktion kann nur ein Zeichen auf dem Bildschirm ausgegeben werden. Dazu wird das Zeichen als erstes Wort auf den Stack gelegt. Der ASCII-Wert des auszugebenden Zeichens muß im Low Byte des Wortes stehen, das High Byte sollte Null sein.
Mit CONOUT auszugebende Zeichen werden über die Device-Nummer 2, normale Consolausgabe, ausgegeben. Damit werden Steuerzeichen wie Control-Character und Escape-Sequenzen ordnungsgemäß interpretiert.

```
MOVE.W #'A',-(SP)  * Ein A soll ausgegeben werden
MOVE.W #2,-(SP)    * Conout
TRAP   #1          * GEMDOS aufrufen, Zeichen ausgegen
ADDQ.L #4,SP       * Stack korrigieren
```

$03 AUXILIARY INPUT

Unter der Bezeichnung Axiliary Port verbirgt sich die RS232-Schnittstelle des ATARI. Mit der Funktion Cauxin kann von der Schnittstelle in Zeichen geholt werden. Die Funktion

kehrt zurück, wenn ein Zeichen vollständig empfangen wurde. Das Zeichen befindet sich nach der Rückkehr in den unteren acht Bit des Registers D0.

$04 AUXILIARY OUTPUT

Analog der Eingabe von Zeichen über die serielle Schnittstelle kann mit dieser Funktion ein Zeichen gesendet werden. Auch bei dieser Funktion sollte der Programmierer die oberen acht Bit des Wortes löschen und den Code des zu sendenden Zeichens in den unteren acht Bit übergeben.

$05 PRINTER OUTPUT

Printer Output ist die einfachste Methode, einen an der Centronics-Schnittstelle angeschlossenen Drucker zu betreiben. Mit jedem Aufruf wird dem Drucker ein Zeichen übergeben.
Wichtig bei der Funktion Printer Output ist der Return-Wert in D0. Wurde das Zeichen aun den Drucker gesendet, so erhält man den Wert -1 ($FFFFFFFF) in D0 zurück. Ist jedoch der Drucker innerhalb von ca. 30 Sekunden nicht in der Lage, das Zeichen anzunehmen (nicht eingeschaltet, OFF LINE, kein Papier usw.), so meldet das GEMDOS dem Programm einen Time Out. In D0 ist dann eine Null enthalten.

```
MOVE.W #'A',-(SP)  * Ein A soll ausgegeben werden
MOVE.W #5,-(SP)    * Funktionsnummer
TRAP   #1          * GEMDOS aufrufen, Zeichen ausgeben
ADDQ.L #4,SP       * Stack korrigieren
TST.W  D0          * Flags beeinflussen
BEQ    printerfehler
```

$06 RAWCONIO

RAWCONIO ist eine etwas ungewöhnliche Mischung aus Tastatureingabe und Bildschirmausgabe und erhält einen Parameter auf dem Stack.
Bei einem Funktionswert $FF wird die Tastatur überprüft. Wenn ein Zeichen vorhanden ist, so wird es wie schon bei CONIN besprochen mit ASCII- und SCAN-Code in D0 übergeben. Ist jedoch kein Tastenwert vorhanden, so wird der Wert Null sowohl als ASCII- wie auch SCAN-Code in D0 übergeben. Der Aufruf RAWCONIO mit Parameter $FF ist somit der BASIC-Funktion INKEY vergleichbar.
Wird ein von $FF unterschiedlicher Wert an die Funktion übergeben, so wird der Wert als auszugebendes Zeichen interpretiert und an der momentanen Cursor-Position ausgegeben. Auch diese Ausgabe interpretiert die Control-Zeichen und Escape-Sequenzen richtig.

```
ANFANG:

MOVE.W #$FF,-(SP)  * Funktionswert Tastatur prüfen
MOVE.W #6,-(SP)    * Funktionsnummer
TRAP   #1          * GEMDOS aufrufen, Tastatur prüfen
ADDQ.L #4,SP       * Stack korrigieren
TST.W  D0          * Zeichen angekommen?
BEQ    ANFANG      * noch nicht!
CMP.B  #3,D0       * ^C frei gewähltes Ende-Kennzeichen
BEQ    ENDE
MOVE   D0,-(SP)    * Zeichen für die Ausgabe auf den Stack
MOVE   #6,-(SP)    * Funktionsnummer
TRAP   #1          * GEMDOS aufrufen, Tastatur prüfen
ADDQ.L #4,SP       * Stack korrigieren
BRA    ANFANG      * neues Zeichen holen
```

$07 DIRECT CON IN WITHOUT ECHO

Die Funktion $07 unterscheidet sich von $01 nur dadurch, daß das von der Tastatur erhaltene Zeichen nicht auf dem Bildschirm ausgegeben wird. Es wird also genau wie bei CONIN auf eine Taste gewartet.

$08 CON IN WITHOUT ECHO

Die Funktion $08 unterscheidet sich nicht von der zuvor genannten Funktion $07. Beide Funktionsaufrufe bewirken exakt das Selbe. Die Ursache für dieses scheinbar unsinnige Verhalten liegt in der schon erwähnten Kompatibilität mit MS-DOS begründet. Unter MS-DOS unterscheiden sich die beiden Funktionen dadurch, daß mit $08 bestimmte, beim ATARI nicht vorhandene Tasten richtig ausgewertet werden, während diese Auswertung bei der Funktion $07 unterbleibt.

$09 PRINT LINE

Funktionen zur Ausgabe einzelner Zeichen auf dem Bildschirm haben Sie bereits mit CONOUT und RAWCONIO kennengelernt. PRINT LINE stellt Ihnen eine komfortablere Methode der Textausgabe zur Verfügung. Mit dieser Funktion kann eine ganze Zeichenkette an der momentanen Cursorposition ausgegeben werden. Dazu wird die Adresse der Zeichenkette als Parameter auf dem Stack übergeben. Die Zeichenkette selbst wird durch ein Null-Byte begrenzt. Auch bei dieser Funktion können Escape-Sequenzen und Control-Character übermittelt werden.

Nach dem Aufruf erhalten Sie in D0 die Anzahl der Zeichen, die ausgegeben wurden. Die Länge der Zeichenkette ist nicht begrenzt.

```
MOVE.L   #text,-(SP) * Adresse der Zeichenkette auf den Stack
 MOVE     #$09,-(SP)  * Funktionsnummer PRT LINE
 TRAP     #1          * GEMDOS aufrufen
 ADDQ.L   #6,SP       * Stack wieder 'aufräumen'
.
.
string0  .dc.b  'dies ist der auszugebende Text',$0d,$0a,0
```

$0A READLINE

READLINE stellt eine recht komfortable Funktion zur Eingabe von Zeichen über die Tastatur dar. Im Gegensatz zu den 'einfacheren' zeichenorientierten Eingabefunktionen kann mit READLINE eine ganze Eingabezeile von der Tastatur geholt werden. Die eingegebenen Zeichen werden gleichzeitig auf dem Bildschirm dargestellt.
Als Parameter wird der Funktion die Adresse eines Eingabe-Buffers übergeben. Der Wert des ersten Bytes des Eingabe-Buffers bestimmt die maximale Länge der Eingabezeile, muß also vor dem Aufruf initialisiert werden. Das zweite Byte des Buffers enthält nach dem Beenden der Eingabe die Anzahl der eingegebenen Zeichen. Die Zeichen selbst stehen im Buffer ab dem dritten Byte.
Die von READLINE benutzte Routine zur Tastatureingabe ist von den zeichenorientierten Console-Eingaben gänzlich unterschiedlich. Escape-Sequenzen werden bei der Ausgabe nicht interpretiert. Nur Control-Zeichen wie Control-H (Backspace) und Control-I (TAB) werden erkannt und entsprechend behandelt. Insgesamt sind die folgenden, teilweise sonst nicht wirksamen Control-Zeichen möglich:

^C beendet Eingabe UND Programm (!)
^H Backspace ein Zeichen
^I TAB, Tabulatorsprung

^J Line Feed, Eingabe beenden
^M CR, Eingabe beenden
^R eingegebene Zeile wird in neuer Zeile ausgegeben
^U Zeile nicht werten, in neuer Zeile beginnen
^X Zeile löschen, Cursor an Zeilenanfang.

Wenn man sicher ist, nicht die Control-Funktionen zu verwenden, so kann man READLINE recht einfach handhaben. Besonders die Funktion ^H, das Löschen eingegebener Zeichens ist nützlich. Trotzdem sollte man bei umfangreicheren Programmen eine eigene Eingaberoutine schreiben, da ^C sehr 'gefährlich' ist. Nicht nur am unmittelbaren Zeilenanfang (wie bei CP/M), sondern immer während der Eingabe wird mit ^C das Programm sofort beendet. Diese Möglichkeit ist nicht ausschaltbar.

Werden mehr Zeichen eingegeben, als bei der Initialisierung im ersten Byte des Buffers angegeben wurden, so wird automatisch die Eingabe abgebrochen. Wird die Eingabe mit ENTER, ^J oder ^M abgeschlossen, so steht das Abschlußzeichen nicht im Buffer (im Gegensatz zur offiziellen ATARI-Dokumentation).
Nach der Eingabe enthält D0 die Anzahl der eingegebenen Zeichen ohne ENTER, die ja auch im Buffer+1 gefunden werden kann.

$0B CONSTAT

Alle Tastenbetätigungen werden zunächst in einem Buffer des Betriebssystems zwischengespeichert. Dieser Speicher ist insgesamt 64 Bytes groß. Die zwischengespeicherten Tastenwerte werden erst beim Aufruf einer GEMDOS-Eingaberoutine aus dem Buffer an die Routine übergeben.
Mit CONSTAT kann geprüft werden, ob im Tastatur-Buffer Zeichen gespeichert sind. Nach dem Aufruf erhält man in D0 den Wert Null oder $FFFF. Eine Null in D0 signalisiert, daß keine Zeichen verfügbar sind.

$0E SETDRV

Mit der Funktion SETDRV kann das aktuelle Laufwerk bestimmt werden. Dazu wird dem Aufruf ein Parameter (16 Bit) übergeben, in dem die Laufwerksbezeichnung enthalten ist. Das Laufwerk A wird mit der Nummer 0, das Laufwerk B der Nummer 1 angesprochen.
Nach dem Aufruf erhält man in D0 die Nummer des vor dem Aufruf aktiven Laufwerkes.

$10 CONOUT STAT

CONOUT STAT liefert nach dem Aufruf in D0 den Status der Console. Bei Rückgabe des Wertes $FFFF kann ein Zeichen auf dem Bildschirm dargestellt werden. Ist jedoch der Rückgabewert Null, so ist derzeit keine Zeichenausgabe auf dem Bildschirm möglich. Alle Versuche, einen Not-Ready-Status der Console zu erzeugen waren übrigens erfolglos. Die einzige vorstellbare Möglichkeit für den Not-Ready-Status wäre, wenn die Ausgabe der einzelnen Bit-Muster eines Zeichens selbst durch einen Interrupt unterbrochen würde und in der Interrupt-Routine eine neue Zeichenausgabe vorgenommen werden soll. In diesem Fall würde natürlich die Darstellung reichlich durcheinander geraten. Allerdings konnte dieser Fall nicht erzeugt werden.

$11 PRTOUT STAT

Diese Funktion liefert den Status, also den Zustand der Centronics-Schnittstelle. Ist kein Drucker angeschlossen (nicht eingeschaltet, OFF LINE), so erhält man nach dem Aufruf in D0 den Wert Null als Kennzeichen 'Drucker nicht verfügbar'. Ist der Drucker jedoch empfangsbereit, so erhält man in D0 den Wert $FFFF zurück.

$12 AUXIN STAT

Durch den Aufruf von AUXIN STAT erfährt man, ob ein Zeichen vom Empfänger der seriellen Schnittstelle verfügbar ist ($FFFF) oder nicht (Null). Wie bei allen anderen Funktionen wird der Wert in D0 zurückgegeben.

$13 AUXOUT STAT

AUXOUT STAT gibt Auskunft über den Zustand des Senders der seriellen Schnittstelle. Ein Wert von $FFFF signaliert, daß die serielle Schnittstelle ein Zeichen senden kann, Null dagegen signalisiert, das derzeit kein Zeichen zum Transmitter geschickt werden sollte.

$19 CURRENT DISK

Für manche Anwendungen ist es erforderlich zu wissen, welches Laufwerk derzeit angemeldet ist. Mit der Funktion $19 kann das aktuelle Laufwerk einfach ermittelt werden. Nach dem Aufruf erhält man die Nummer des Laufwerks in D0 zurück. Die Bedeutung der Nummern ist identisch mit den Laufwerksnummern bei $0E, Set Drive, also 0 = A, 1 = B u.s.w.

$1A SET DISK TRANSFER ADDRESS

Die Disk Transfer Adresse ist die Adresse eines 44-Byte-Buffers, der für verschiedene Diskettenoperationen (besonders Directory-Operationen) benötigt wird. Beispiele für die Verwendung der DTA finden Sie bei den GEMDOS-Funktionen SEARCH FIRST und SEARCH NEXT.

```
MOVE.L #DTADRESS,-(SP) * Adresse des 44-Byte DTA-Buffers
MOVE.W #$1A,-(SP)      * Funktionsnummer SET DTA
TRAP   #1              * DTA setzen
ADDQ.L #6,SP           * Stack aufräumen
```

$20 SUPER

Diese Funktion ist besonders für die Programmierer interessant, die während eines im User-Modus laufenden Programms auf die nur im Supervisor-Modus zugänglichen Peripherie-Bausteine oder die Systemvariablen zugreifen wollen. Nach Aufruf dieser Funktion aus dem User-Modus heraus befindet sich der MC68000 im Supervisor-Mode. Im Gegensatz zur XBIOS-Routine zum Einschalten des Supervisor-Modus können nach erfolgtem SUPER-Aufruf weitere GEMDOS-, BIOS- und XBIOS-Aufrufe erfolgen.
Zunächst wollen wir den Fall betrachten, daß sich ein Programm im User-Mode befindet und die SUPER-Funktion mit einem Wert von Null auf dem Stack aufruft. In diesem Fall befindet sich das Programm nach dem Aufruf im Supervisor-Mode. Der Supervisor-Stackpointer wird auf den derzeitigen Wert des User-Stacks gebracht und in D0 erhält man den ursprünglichen Wert des Supervisor-Stacks zurück. Diesen Wert sollte man sich im Programm merken, um später wieder in den User-Modus zu gelangen.
Wird jedoch als Parameter beim erstmaligen Aufruf ein von Null unterschiedlicher Wert der SUPER-Funktion übergeben, so wird dieser Wert als gewünschter neuer Supervisor-Stack interpretiert. Auch in diesem Fall erhalten Sie in D0 den alten Supervisor-Stackpointer zurück, dessen Wert das Programm auf jeden Fall speichern sollte.
Bevor ein Programm beendet wird, sollte nämlich der User-Modus wieder eingeschaltet werden. Dieser erneute Wechsel der Betriebsart benötigt die beim ersten Aufruf

erhaltene Adresse, um den Supervisor-Stack auf seinen ursprünglichen Wert zu setzen.

Die Superfunktion unterscheidet sich in einem ganz wesentlichen Punkt von den bisherigen und allen anderen GEMDOS-Funktionen. Dieser Aufruf kann unter Umständen auch die Inhalte von A1 und D1 verändern. Wenn Sie also wichtige Werte in diesen Registern halten, so müssen Sie die Werte vor einem Aufruf der SUPER-Funktion irgendwo ablegen.

```
                        * der 68000 befindet sich im User-Modus
CLR.L  -(SP)            * User Stack soll Supervisor Stack werden
MOVE.W #$20,-(SP)       * SUPER aufrufen
TRAP   #1               * nach dem Trap ist Supervisor-Modus aktiv
ADD.L  #6,SP            * D0 = alter Supervisor-Stack
MOVE.L D0,_SAVE_SSP     * Wert merken
.
.
.
.  hier kann im Supervisor-Mode gearbeitet werden
.
.
.
MOVE.L _SAVE_SSP,-(SP) * alten Supervisor-Stackpointer
MOVE.W #$20,-(SP)       * SUPER aufrufen
TRAP   #1               * jetzt sind wir wieder im User Mode
ADD.L  #6,SP
.
.
.
```

$2A GET DATE

Sicher haben Sie bereits einmal mit dem Kontrollfeld experimentiert. Neben verschiedenen anderen Funktionen ist im Kontrollfeld eine Uhr mit Uhrzeit und Datum enthalten. Für einige Anwendungen kann es nützlich sein, das eingestellte Datum zur Verfügung zu haben. Das Datum läßt sich problemlos mit der Funktion GET DATE ermitteln. Dieser Funktionsaufruf benötigt keine Parameter und stellt nach dem Aufruf das Datum im Low-Word des Registers D0 zur Verfügung. Allerdings ist es recht aufwendig codiert, so daß das Ergebnis in D0 noch aufbereitet werden muß, um das richtige Datum zu erhalten.
In den unteren 5 Bits wird der Tag im Bereich von 1 bis 31 codiert. Die Bits 5 bis 8 enthalten den Monat im Wertebereich von 1 bis 12 und das Jahr ist in den Bits 9 bis 15 enthalten. Der Wertebereich in diesen 'Jahresbits' geht von 0 bis 119. Der Wert dieser Bits muß zum Wert 1980 addiert werden, um die tatsächlich eingestellte Jahreszahl zu bekommen. Das Datum 12.12.1992 z.B. ergäbe $198C in D0. Binär läßt sich das als %0001100.1100.01100 darstellen. Die Längen der drei Felder haben wir durch einen Punkt markiert.

$2B SET DATE

Auch aus Anwendungsprogrammen können Uhrzeit und Datum gestellt werden. Das ist besonders für solche Programme interessant, die Datum und/oder Uhrzeit verwenden. Ein Beispiel wäre die Rechnungsbearbeitung, in der das aktuelle Datum in den Rechnungsausdruck eingefügt wird. Solche Programme können dann selbstätig den Benutzer zur Eingabe des Datums auffordern. Damit kann vermieden werden, daß der Anwender zu Beginn der Arbeit das Setzen von Datum und Uhrzeit im Kontrollfeld vergißt und mit dem nach dem Einschalten vorhandenen Wert arbeiten.
Das Datum muß der Funktion SET DATE im selben Format übergeben werden, wie es von GET DATE geliefert wird, also Bit 0-4 = Tag, Bit 5-8 = Monat, Bit 9-15 =Jahr-1980.

```
MOVE.W #%101101011001,-(SP)  * Datum soll 25.10.1985 sein
MOVE.W #$2A,-(SP)      * Funktionsnummer von SET DATE
TRAP   #1              * Datum setzen
ADDQ.L #6,SP           * Stack reparieren
```

$2C GET TIME

Die Funktion GET TIME liefert das aktuelle (sprich eingestellte) Datum der vom GEMDOS mitgeführten Uhr. Ähnlich wie beim Datum wird die Uhrzeit, die in D0 nach dem Aufruf zur Verfügung steht, in einzelnen Bits des Registers nach einem speziellen Muster codiert. In den Bits 0 bis 4 werden die Sekunden dargestellt. Da aber mit 5 Bit nur der Bereich von 0 bis 31 dargestellt werden kann, läuft die interne Uhr in 2-Sekunden-Schritten. Um das richtige Sekunden-Ergebnis zu erhalten, muß also der Inhalt dieser untersten 5 Bits mit zwei multipliziert werden. In den Bits 5 bis 10 wird die Anzahl der Minuten festgehalten, während die restlichen Bits 11 bis 15 Auskunft darüber geben, welche Stunde (im 24-Stunden-Format) dem ATARI geschlagen hat.

$2D SET TIME

Auch das Setzen der Uhrzeit ist unter GEMDOS einfach möglich. Die Funktion SET TIME erwartet dazu auf dem Stack einen 16-Bit-Wert (Wort), in dem die Zeit in der selben Form codiert ist, wie auch GET TIME die Uhrzeit liefert.

```
MOVE.W #%1000101010111101,-(SP)  * Uhrzeit 17:21:58
MOVE.W #$2D,-(SP)        * Funktionsnummer von SET TIME
TRAP   #1                * Datum setzen
ADDQ.L #6,SP             * Stack reparieren
```

$2F GET DTA

Die Funktion $2F stellt das Gegenstück zur Funktion $1A, SET DTA, dar. Ein Aufruf dieser Funktion ergibt die Adresse des derzeitigen Disk Transfer Buffers in D0. Eine genauere Erläuterung dieses Buffers finden Sie weiter unten bei den Funktionen SEARCH FIRST und SEARCH NEXT.

$30 GET VERSION NUMBER

Ein Aufruf dieser Funktion übergibt in D0 die Versionsnummer des GEMDOS. In der derzeitig auf Diskette gelieferten Version des GEMDOS wird eine entsprechende Anfrage immer mit $0D00 beantwortet, das entspricht der Version 13.00. Laut offizieller ATARI-Dokumentation sollte zwar ein Wert von $0100 für diese erste Version gemeldet werden, möglicherweise jedoch soll der Wert signalisieren, daß es sich bei der vorhandenen GEMDOS-Version um die $D = Diskettenversion handelt.

$31 KEEP PROCESS

Diese Funktion ist vergleichbar mit der GEMDOS-Funktion TERM $00. Auch nach Aufruf dieser Funktion wird das Programm beendet. Allerdings unterscheidet sich $31 in einigen wesentlichen Punkten von $00.
Nach Abarbeiten des TRAP #1 wird wie bei TERM die Kontrolle an das Programm zurück gegeben, welches das soeben beendete Programm gestartet hat. Im Unterschied zu TERM jedoch kann dem Aufrufer eine Abschluß-Bedingung mitgeteilt werden. Liefert TERM grundsätzlich den Abschlußwert Null (kein Fehler) so kann bei $31 zwischen Null und Eins als Abschlußwert gewählt werden. Ein von Null unterschiedlicher Wert signalisiert 'Fehler bei der Programmabarbeitung'.
Ein weiterer wesentlicher Punkt liegt in der Speicherverwaltung des GEMDOS begründet. Wird ein Programm gestartet, so wird ihm der gesamte verfügbare Speicherplatz zur Verfügung gestellt. Wird das Programm mit TERM beendet, so wird der Speicherplatz wieder freigegeben und GEMDOS zur Verfügung gestellt. Der gesamte freigegebene Speicherbereich wird zudem gelöscht sprich mit Nullen gefüllt. Das Programm ist tatsächlich physikalisch aus dem Speicher verschwunden. Mit der Funktion $31 jedoch kann ein beliebiger Speicherbereich ab der Start-Adresse des Programms geschützt werden. Dieser Speicherbereich wird nach dem Beenden des Programms nicht freigegeben und auch nicht gelöscht. Das Programm könnte wieder gestartet werden ohne es neu zu laden.
KEEP PROCESS wird mit zwei Parametern aufgerufen. Das Beispielprogramm zeigt die Parameterübergabe

```
MOVE.W #0,-(SP)        * Fehlercode kein Fehler, sonst 1
MOVE.L #$1000,-(SP)    * $1000 Bytes ab Programmstart schützen
MOVE.W #$31,-(SP)      * Funktionsnummer, Programm endet
TRAP   #1              *                                jetzt
```

$36 GET DISK FREE SPACE

Für diskettenorientierte Programme kann es sehr wichtig sein, die Größe des freien Platzes auf der Diskette zu ermitteln. Damit hat man die Möglichkeit, den Anwender zum rechtzeitigen Diskettenwechsel aufzufordern. 'Disk Full'-Fehlermeldungen oder gar Datenverluste können dann sicher verhindert werden können.
Genau diese Information liefert die Funktion $36. Dazu muß die Nummer des gewünschten Laufwerks und die Adresse eines 16 Byte großen Buffers an die Funktion übergeben werden. Wird als Laufwerknummer der Wert 0 übergeben, so wird die Information vom aktiven Laufwerk geholt, bei einer 1 von Drive A, bei 2 von Drive B.
Die im Buffer übergebene Information teilt sich in 4 Langworte auf.
Das erste Langwort enthält die Anzahl noch freier sogenannter Allocation Units. Jede Datei, und sei sie auch nur acht Bytes groß, benötigt mindestens eine solche Alloction Unit.
Das zweite Langwort gibt Aufschluß über die Anzahl der auf der Diskette vorhandenen Allocation Units, unabhängig, ob sie bereits belegt oder noch frei sind. Bei der 'kleinen' einseitigen Diskette beträgt dieser Wert $15C oder 351, die doppelseitigen Disketten haben $2C7 = 711 Allocation Units.
Das dritte Langwort enthält die Größe eines Diskettensektors in Byte. Das sind bei ATARI bislang immer 512 Bytes ($200 Bytes).
Im letzten Langwort schließlich wird die Anzahl der zu einer Allocation Unit gehörenden physikalischen Sektoren vermerkt. Hier ist normalerweise eine 2 eingetragen. Zwei Sektoren bilden also eine Allocation Unit.
Aus den so gewonnenen Informationen kann nun ohne Schwierigkeiten die Anzahl der noch verfügbaren Bytes berechnet werden.

```
MOVE.W #0,-(SP)         * Information vom aktive Drive
MOVE.L #BUFFER,-(SP)    * Adresse des 16-Byte Buffers
MOVE   #$36,-(SP)       * Funktionsnummer
TRAP   #1
ADDQ.L #8,SP            * Stack aufräumen
.
.
.
.bss
BUFFER:
freal: .ds.l   1        * Free Allocation Units
total: .ds.l   1        * Total Allocation Units
bps:   .ds.l   1        * Bytes/physical Sector
pspal: .ds.l   1        * phys. Sectors/Alloc. Unit
```

$39 MKDIR

Vom Desktop aus kann ein Unterdirectory mit dem Punkt 'NEUER ORDNER' angelegt werden. Mit dem Aufruf $39 kann ein solches Unterdirectory auch von Anwenderprogrammen heraus angelegt werden. Mit Hilfe solcher Unterdirectorys oder Ordner kann man bei ein wenig Planung die Diskette recht übersichtlich organisierten.
Um einen neuen Ordner zu schaffen, wird der Funktion $39 die Adresse des Ordner-Namens, auch als Pfadname (engl. Pathname) bezeichnet, übergeben. Dieser Name darf aus 8 Zeichen und drei Zeichen Extension bestehen. Damit gilt für den Pfadnamen die gleiche Einschränkung wie für Filenamen. Er muß beim Aufruf von MKDIR mit einem Null-Byte abgeschlossen sein.
Nach dem Aufruf wird in D0 mitgeteilt, ob die Operation erfolgreich abgeschlossen wurde. Ist in D0 eine Null enthalten, so war der Aufruf erfolgreich. Fehler werden durch eine negative Nummer in D0 gekennzeichnet. Eine

Übersicht über alle im Zusammenhang mit GEMDOS-Funktionen auftretenden Fehlermeldungen finden Sie zum Schluß dieses Kapitels.

```
MOVE.L #pathname        * Adresse des Pfadnamens
MOVE   #$39,-(SP)       * Funktionsnummer
TRAP   #1
ADDQ.L #6,SP            * Stack reparieren
TST.W  D0               * Fehler aufgetreten?
BNE    fehler           * es scheint so
.
.
.
pathname:
       .dc.b   'privat.dat',0
```

$3A RMDIR

Ein mit MKDIR erzeugtes Subdirectory kann mit $3A auch wieder gelöscht werden. Wie zuvor wird dazu der mit Null abgeschlossene Pfadname an RMDIR übergeben. Und auch die Fehlermeldungen sind entsprechend MKDIR Null bei Erfolg oder ein negativer Wert bei Fehlern. Eine wichtige Fehlermeldung soll an dieser Stelle schon einmal erwähnt werden. Es ist dies die Meldung -36 ($FFFFFFCA). Diese Fehlermeldung erhalten Sie, wenn das zu löschende Subdirectory Files enthält.
Man kann mit RMDIR also nur leere Subdirectorys löschen. Im Falle der beschriebenen Fehlermeldung muß man dann zunächst mit UNLINK ($41) alle Files im Subdirectory löschen und danach einen neuen Anlauf nehmen.

$3B CHDIR

Das unter GEMDOS verfügbare System der Subdirectorys ist in genau der selben Form auch unter UNIX verfügbar. Nun ist dieses System zwar auch auf Rechnern mit Diskettenstationen gewinnbringend einzusetzen, seinen Vorteil spielt es jedoch erst dann aus, wenn im System ein großer Massenspeicher wie z.B. eine Harddisk mit mehreren Megabyte Speicherkapazität vorhanden ist. Nach einiger Zeit würde wahrscheinlich die meiste Zeit mit dem Suchen von Files im Directory zugebracht werden.
Zur besseren Organisation der Daten können in Subdirectorys weitere Subdirectorys untergebracht werden. Dadurch kann es nötig werden, mehrere Subdirectorys anzugeben, bis man das Directory hat, in dem das gewünschte File gespeichert ist. Ein Beispiel könnte lauten:

```
/hugos.dat/cfiles.s/csorts.s/cqsort.s
```

Übersetzt würde das etwa lauten:
Lade die Datei cqsort.s aus dem Subdirectory csorts.s. Dieses Subdirectory csorts.s befindet sich im Subdirectory cfiles.s welches seinerseits im Subdirectory hugos.dat zu finden ist. Würde man den ganzen Ausdruck als Filenamen angeben, so könnte tatsächlich das gewünschte File geladen werden (vorausgesetzt das File und alle Subdirectorys sind vorhanden). Will man jedoch auf das über den selben Pfad (verstehen Sie jetzt den Begriff Pfadname?) zugängliche File cqsort1.s zugreifen, so müßte der komplette Pfad neu eingegeben werden. Man kann aber auch das durch den Pfad spezifizierte Subdirectory zum aktuellen Directory machen, indem die Funktion CHDIR mit Angabe des gewünschten Pfads aufruft. Danach kann auf alle Files im ausgewählten Subdirectory nur über den Filenamen zugegriffen werden. Der Pfad ist durch die Funktion festgelegt.

```
MOVE.L #path,-(SP)      * Adresse des Pfads
MOVE.W #$3B,-(SP)       * Funktionsnummer
TRAP   #1
ADDQ.L #6,SP            * Stack reparieren
TST.W  D0               * Fehler aufgetreten?
BNE    fehler           * es scheint so
.
.
.
path:
       .dc.b   '/hugos.dat/privat.dat/',0
```

$3C CREATE

In allen Betriebssystemen läuft der Zugriff auf Dateien über die Sequenz Eröffnen der Datei, Zugriff auf die Daten, also lesen oder schreiben von Daten und abschließendem Schließen der Datei. Diese 'Dreieinigkeit' existiert auch unter GEMDOS, allerdings gibt es eine Ausnahme. Unter CP/M z.B., kann auch eine nicht vorhandene Datei eröffnet werden. Mit dem Eröffnen wird dann die bisher nicht vorhandene Datei eingerichtet. Unter GEMDOS jedoch muß eine Datei zuerst einmal kreiert, also geschaffen werden. Dazu dient der Aufruf $3C, CREATE. Diesem GEMDOS-Aufruf werden zwei Parameter übergeben, die Adresse des gewünschten Filenamens und ein Attribut-Wort.
Wird als Attribut-Wort Null übergeben, so wird eine ganz normale Datei eingerichtet, die sowohl beschrieben wie auch gelesen werden kann. Wird als Attribut der Wert 1 übergeben, so kan die Datei nach dem Schließen nur noch gelesen werden. Sie ist softwaremäßig schreibgeschützt (was natürlich nicht verhindern kann, daß die Datei nach einem Formatieren der Diskette verschwunden ist).
Weitere mögliche Attribute sind $02, $04 und $08. Das

Attribut $02 erzeugt eine 'versteckte' Datei und das Attribut $04 eine 'versteckte' Systemdatei. Das Attribut $08 schließlich erzeugt eine Datei mit einem 'Volume-Label'. Das Volume-Label ist der (optionale) Name, der einer Diskette beim Formatieren gegeben werden kann. Aus den maximal 11 Bytes des Namens und der Extension wird dann der Diskettenname erzeugt. Dateien mit einem der letzten drei Attribute sollten von der normalen Directory-Suche ausgenommen sein. Im ST erscheinen sie aber doch im Directory.
Wenn die Funktion CREATE beendet ist, so wird im Register D0 ein File-Descriptor, auch File-Handle genannt zurückgegeben. Über diese File-Handle (ein numerischer Wert zwischen 6 und 45) gehen alle weiteren Zugriffe auf die Datei. Beim Schreiben, Lesen und schließen von Dateien muß die Handle angegeben werden. Insgesamt können also maximal $28 = 40 Dateien gleichzeitig geöffnet werden.
Wenn beim CREATE-Aufruf bereits eine Datei mit diesem Namen existiert, so wird diese auf Null-Länge abgeschnitten. Das ist gleichbedeutetnd mit der Sequenz Löschen der alten Datei und Erstellen einer neuen, gleichnamigen Datei, geht aber sehr viel schneller.
Erhalten Sie nach dem CREATE-Aufruf eine Handle-Nummer in D0 zurück, so muß die Datei nicht noch zusätzlich mit $3D OPEN eröffnet werden.

```
MOVE.W #$0,-(SP)          * Datei soll R/W-Status erhalten
MOVE.L #filename,-(SP)    * Adresse des Filenamens auf den Stack
MOVE.W #$3C,-(SP)         * Funktionsnummer
TRAP   #1                 * GEMDOS aufrufen
ADDQ.L #8,SP              * Stack aufräumen
TST    D0                 * Fehler aufgetreten?
BMI    error              * scheint so
MOVE   D0,handle          * File-Handle für spätere Zugriffe merken.
.
.
.
```

```
filename:                    * Null-Byte nicht vergessen !
        .dc.b   'myfile.dat',0
.
handle:
        .ds.w   1
```

$3D OPEN

Mit CREATE kann man nur neue Dateien erstellen oder bestehende auf Null-Länge kürzen. Bestehende Dateien müssen aber auch weiter verarbeitet werden können. Dazu müssen solche Dateien mit der OPEN-Funktion eröffnet werden.
Der erste Parameter der OPEN-Funktion ist das Modus-Wort. Bei einer Null im Modus-Wort kann die eröffnete Datei nur gelesen, bei einer 1 nur beschrieben werden. Mit einem Wert von 2 kann die Datei gleichermaßen gelesen wie beschrieben werden. Als zweiter Parameter wird in der schon bekannten Weise die Adresse des durch ein Null-Byte abgeschlossenen Dateinamens übergeben.
Als Ergebnis der OPEN-Funktion wird in D0 die Handle-Nummer übergeben, wenn die Datei vorhanden und die gewünschte Zugriffsart möglich ist. Sonst erhält man in D0 eine Fehlernummer. Auf die Aufstellung der möglichen Fehlernummern am Ende des Kapitels soll noch einmal hingewiesen werden.

```
MOVE.W #$2,-(SP)        * Datei soll gelesen und beschieben werden
MOVE.L #filename        * Adresse des Filenamens auf den Stack
MOVE.W #$3D,-(SP)       * Funktionsnummer
TRAP   #1               * GEMDOS aufrufen
ADDQ.L #8,SP            * Stack aufräumen
TST.W  D0               * Fehler aufgetreten?
BMI    error            * scheint so
MOVE   D0,handle        * File-Handle für spätere Zugriffe merken.
```

```
.
.
.
filename:               * Null-Byte nicht vergessen !
        .dc.b   'myfile.dat',0
.
handle:
        .ds.w   1
```

$3E CLOSE

Jede eröffnete Datei sollte geschlossen werden, wenn innerhalb des Programms nicht mehr auf die Datei zugegriffen werden muß, oder wenn das Programm selbst beendet wird. Besonders zum Schreiben eröffnete Dateien müssen unbedingt vor Programmende geschlossen werden, da es sonst zu einem unübersehbaren Datenverlust kommen kann.
Geschlossen werden Dateien mit dem Aufruf CLOSE, dem die Handle-Nummer als Parameter übergeben wird. Als Return-Wert erhält man den Wert Null, wenn die Datei korrekt geschlossen worden ist.

```
MOVE.W handle,-(SP)     * hier haben wir die Handle-Nummer abgelegt
MOVE.W #$3E,-(SP)       * Funktionsnummer
TRAP   #1               * GEMDOS aufrufen
ADDQ.L #4,SP            * Stack aufräumen
TST.W  D0               * Fehler aufgetreten?
BMI    error            * scheint so
.
.
.
handle:
        .ds.w   1
```

$3F READ

Das Öffnen und Schließen von Dateien ist natürlich nur die halbe Miete, es soll ja auch mit Daten gearbeitet werden. So sollen z.B. Briefe, Lohn- und Gehaltslisten, Meßwerte und Artikelnummern gespeichert und später wieder gelesen werden können. Das Lesen solcher Daten kann in sehr eleganter Weise mit der Funktion READ geschehen. READ bekommt drei Parameter übermittelt: zuerst die Adresse eines Buffers, in den die Daten gelesen werden sollen, dann die Anzahl der aus der Datei zu lesenden Bytes und zum Schluß die Handle-Nummer der Datei. Diese Nummer haben Sie (hoffentlich) beim zuvor erfolgten OPEN erhalten.
Bei der Funktion CREATE haben wir bereits die möglichen Handle-Nummern erwähnt. Unerwähnt geblieben ist jedoch, warum als erste freie Handle-Nummer die Sechs vergeben wird. Ursache ist der Umstand, das auch sogenannte Devices, in unserem Falle die Tastatur, der Bildschirm, der Drucker und die serielle Schnittstelle bei READ- und WRITE-Operationen über eine Handle-Nummer angesprochen werden können. Die ermittelten Zuordnungen lauten:

0 = Console-Eingabe
1 = Console-Ausgabe
2 = RS 232
3 = Drucker

Auch die Nummern 4 und 5 bewirken Console-Ein- und Ausgaben. Allerdings liefert das System bei Benutzung dieser Handle-Nummern mitunter den Fehler 'Invalid Handle Number'. Die richtige Programmierung und der genaue Verwendungszweck dieser beiden Nummern ist nicht bekannt.

Als Rückgabewert erhalten Sie in D0 entweder eine Fehlernummer (hoffentlich nicht) oder die Anzahl der fehlerfrei gelesenen Bytes. Eine Meldung über das Erreichen des Dateiendes erhält man nicht. Dies ist aber auch nicht nötig, da die Größe der Datei in Bytes im Directory-Eintrag enthalten ist (siehe SEARCH FIRST/SEARCH NEXT). Wird über

das logische Datei-Ende hinausgelesen, so wird keine Meldung ausgegeben. Spätestens am Ende der letzten von der Datei belegten Allocation Unit wird jedoch das Lesen abgebrochen. Die Anzahl der gelesenen Bytes ist in diesem Fall immer eine durch $400 teilbare Zahl.

```
MOVE.L #buffer,-(SP)    * Adresse des Datenbuffers
MOVE.L #$100,-(SP)      * 256 Bytes lesen
MOVE.W handle,-(SP)     * Platz für die Handle-Nummer
MOVE.W #$3E,-(SP)       * Funktionsnummer
TRAP   #1
ADD.L  #12,SP
TST.L  D0               * ist ein Fehler aufgetreten?
BMI    error            * scheint so
.
.
handle:
       .ds.w   1        * Platz für die Handle-Nummer
.
buffer:
       .ds.b   $100     * reicht in unserem Beispiel
```

$40 WRITE

Genau so einfach wie das Lesen von Dateien ist das Schreiben. Auch die übergebenen Parameter sind mit den beim Lesen benötigten Parametern identisch. Als Handle können von OPEN- und CREATE-Aufrufen gelieferte File-Descriptoren, aber auch die bei READ genannten Device-Nummern benutzt werden. Durch einfache Änderung der Handle-Nummer kann so die Ausgabe eines Programms wahlweise auf den Bildschirm, den Drucker oder in eine Datei erfolgen.

$41 UNLINK

Mit UNLINK können nicht mehr benötigte Dateien gelöscht werden. Dazu muß dem Aufruf die Adresse des Filenamens oder bei Bedarf auch des vollständigen Pfadnamens übergeben werden. Enthält das Register D0 nach dem Aufruf eine Null, so ist die Datei gelöscht worden. Sonst erhält man eine Fehlernummer in D0 zurück.

```
MOVE.L #pathname,-(SP) * Adresse des Datenbuffers
MOVE.W #$41,-(SP)      * Funktionsnummer
TRAP   #1
ADD.L  #6,SP
TST.W  D0              * ist ein Fehler aufgetreten?
BMI    error           * scheint so
.
.
.
pathname:
       .dc.b   '/rolli/privat/pakman.prg',0
```

$42 LSEEK

Bisher haben wir nur rein sequentielle Datenzugriffe kennengelernt. Wir können eine beliebige Datei von Anfang an durchlesen, bis eine gewünschte Information erreicht ist. Mit jedem Lesen wird ein interner Dateipointer mitgeführt, der jeweils auf das nächste zu lesende Byte zeigt. Diesen Pointer können wir nur durch Lesen aus der Datei kontinuierlich in Richtung Dateiende bewegen. Ein paar Bytes zurück oder vorwärts, also richtig wahlfrei können wir den Zeiger nicht verstellen. Das ist jedoch für viele Anwendungen erforderlich.
LSEEK bietet nun eine außerordentlich komfortable

Möglichkeit, den Dateipointer auf jedes beliebiges Byte innerhalb der Datei zu stellen und ab dieser Stelle zu lesen oder zu schreiben. Diese UNIX-kompatible Möglichkeit des GEMDOS ist um vieles komfortabler als die z.B. von CP/M bekannten Möglichkeiten zur relativen Dateiverwaltung.
Insgesamt werden der LSEEK-Funktion drei Parameter übergeben. Der erste Parameter gibt die Anzahl der Bytes an, um die der Pointer bewegt werden soll. Ein weiterer Parameter ist die Handle-Nummer der Datei. der letzte Parameter ist ein Modus-Wort, welches beschreibt, wie der Dateipointer zu bewegen ist. Eine Null als Modus bewegt den Zeiger an den Anfang und von da die angegebene Zahl Bytes zum Ende der Datei. Als Anzahl dürfen nur positive Werte gewählt werden. Bei einem Modus-Wert von 1 wird der Zeiger um den gewünschten positiven oder auch negativen Betrag ab der aktuellen Position bewegt und bei einer 2 gilt die Entfernungsangabe ab dem Ende der Datei. Bei einer 2 als Moduswert sind nur negative Werte zugelassen.
Nach dem Aufruf erhält man in D0 die absolute Position des Zeigers ab Dateianfang oder eine Fehlermeldung.

```
MOVE.W #1,-(SP)        * relativ vom derzeitigen Filepointer
MOVE.W handle,-(SP)    * File-Handle
MOVE.l #$-20,-(SP)     * 32 Bytes zurück
MOVE.W #$42,-(SP)      * Funktionsnummer
TRAP   #1
ADD.L  #10,SP
TST.W  D0              * ist ein Fehler aufgetreten?
BMI    error           * scheint so
.
.
handle:
       .ds.w   1       * Platz für die Handle-Nummer
```

$43 CHANGE MODE (CHMOD)

Mit der Funktion CREATE kann man einer Datei ein bestimmtes Attribut zuordnen. Dieses Attribut kann nur mit der Funktion CHANGE MODE ermittelt und gegebenenfalls geändert werden. Allerdings muß dazu der Name der Datei bekannt sein, da die Adresse des Namens oder des kompletten Pfadnamens als Parameter an CHMOD übergeben werden muß. Ein weiteres Parameter-Wort gibt an, ob die Datei-Attribute ermittelt oder gesetzt werden sollen. Außerdem muß ein Wort übergeben werden, in dem die neu zu setzenden Attribute enthalten sind. Beim Ermitteln der Attribute einer Datei wird dieses Wort nicht benötigt, sollte aber als Dummy-Wert der Routine übergeben werden. Die möglichen Datei-Attribute haben wir zum Teil bereits bei der Funktion CREATE gezeigt, sie sollen trotzdem noch einmal in tabellarischer Form aufgeführt werden.

$00 = normaler Datei-Status, lesen/schreiben möglich
$01 = File ist READ-ONLY.
$02 = 'verborgenes' File
$04 = System-File
$08 = File ist Volume-Label, enthält den Diskettennamen
$10 = File ist ein Subdirectory
$20 = Datei wurde beschrieben und korrekt geschlossen.

Die Attribute $10 und $20 können beim Kreieren einer Datei nicht angegeben werden. Das Attribut $20 wird vom Betriebssystem vergeben, während für das Erstellen von Subdirectorys vorteilhaft die GEMDOS-Funktion MKDIR verwendet wird. Die MKDIR-Funktion erzeugt nicht nur den Directory-Eintrag mit dem entsprechenden Attribut, sondern richtet das Subdirectory physikalisch auf der Diskette ein.

Grundsätzlich erhält man nach dem Aufruf in D0 den aktuellen Attributwert zurück, nach dem Setzen des Attributs also auch den neuen Wert, oder aber, wie bei allen anderen Funktionsaufrufen eine negative Fehlermeldung.

1.Beispiel

```
MOVE.W #1,-(SP)         * Datei soll READ-ONLY-Attribut erhalten
MOVE.W #1,-(SP)         * Kennzeichen Attribut setzen.
MOVE.L #pathname,-(SP) * den Pfadnamen benötigen wir auch
MOVE.W #$43,-(SP)       * Funktionsnummer
TRAP   #1
ADD.L  #10,SP
TST.W  D0               * ist ein Fehler aufgetreten?
BMI    error            * scheint so
.
.
.
pathname:               * Null-Byte am Ende nicht vergessen!
       .dc.b   'killme.not',0
```

2.Beispiel:

```
MOVE.W #0,-(SP)         * Dummy-Wert, wird nicht zwingend benötigt
MOVE.W #0,-(SP)         * Kennzeichen Attribut ermitteln
MOVE.L #pathname,-(SP) * und den Pfadnamen
MOVE.W #$43,-(SP)       * Funktionsnummer
TRAP   #1
ADD.L  #10,SP
TST.W  D0               * ist ein Fehler aufgetreten?
BMI    error            * scheint so
.
.
.
pathname:               * Null-Byte am Ende nicht vergessen!
       .dc.b   'was-bin.ich',0
```

$45 DUP

Wie bereits bei den Funktionen READ und WRITE erwähnt stehen dem Programmierer auch die Devices Console, Lineprinter und RS232 wie Dateien zur Verfügung. Dadurch lassen sich Ein- und Ausgaben von Dateien auch auf diese Devives umleiten. Mit der DUP-Funktion einer der Devices eine File-Handlenummer zugewiesen werden. Nach dem Aufruf erhält man die nächste freie Handlenummer zurück.

$46 FORCE

Die FORCE-Funktion erlaubt eine noch weitergehende Beeinflussung der Handle-Nummern. Wird in einem Programm die Console-Ein/Ausgabe ausschließlich über die READ- und WRITE-Funktionen mit den Handlenummern 0 und 1 vorgenommen, so kann mit diesem Aufruf die Ein- bzw. Ausgabe umgeleitet werden. Bildschirmausgaben werden in eine Datei geschrieben, Eingaben werden nicht von der Tastatur sondern aus einer zuvor eröffneten Datei geholt.

$47 GETDIR

Mit der Funktion $3B kann ein beliebiges Subdirectory zum aktuellen Directory ernammt werden. Alle Dateizugriffe ohne Pfadnamen laufen dann nur im eingestellten Subdirectory ab. Allerdings kann es unter bestimmten Voraussetzungen wesentlich sein, den derzeitigen Pfadnamen zum aktuellen Subdirectory zu ermitteln. Diese Arbeit erledigt der Funktionsaufruf GETDIR, $47. Als Parameter benötigt dieser Aufruf die Bezeichnung des gewünschten Laufwerks (0=aktuelles Drive, 1=Drive A, 2=Drive B u.s.w.) und einen Pointer auf einen Buffer, der 64 Bytes groß sein muß. In diesem Buffer wird der komplette Pfadname zum aktuellen Directory abgelegt. Der Pfadname wird durch ein Nul-Byte abgeschlossen. Wird die Funktion aufgerufen, wenn das Haupt-Directory aktiv ist, so wird kein Pfadname übergeben. In diesem Fall enthält gleich das erste Byte des Buffers Null. Nach dem Aufruf muß D0 den Wert Null enthalten. Bei negativen Werten ist ein Fehler aufgetreten, z.B. dann, wenn eine falsche Laufwerksnummer angegeben wurde.

```
MOVE.W #0,-(SP)         * Pfadname vom aktuellen Drive holen
MOVE.L #buffer,-(SP)    * Adresse des 64-Byte-Buffers
MOVE.W #$47,-(SP)       * Funktionsnummer
TRAP   #1
ADDQ.L #8,SP
TST.L  D0               * Fehler?
BNE    error            * D0 <> Null bei Fehler
.
.
.
buffer:
       .ds.b   64       * Buffer für Pathname
```

$48 MALLOC

Die MALLOC-Funktion und die zwei folgenden Aufrufe MFREE und SETBLOCK befassen sich mit der Speicherorganisation des GEMDOS. Wie bereits bei der Funktion $31, KEEP PROCESS erwähnt, wird einem Programm nach dem Laden der gesamte verfügbare Speicherbereich zugeordnet. Das ist in vielen Fällen unkritisch, da nur ein einziges Programm laufen soll. Es gibt aber gerade unter GEM Anwendungsfälle, in denen eine solche Organisation nicht sinnvoll ist. So kann ja durchaus aus einem Programm heraus ein Accessory wie z.B. der VT-52-Terminal Emulator aufgerufen werden. Ein solches Programm benötigt aber unter Umständen auch Speicherplatz, der aber dann nicht mehr zur Verfügung steht. Auch kann bei vollständig belegtem Speicher kein weiteres Programm-Modul nachgeladen werden. Aus diesem Grunde sollte jedes Programm immer nur den Platz reservieren, den es tatsächlich als Programm- und Datenbereich benötigt. Der nicht benötigte Speicher kann dann an GEMDOS zurückgegeben werden.
Sollte ein Programm jedoch nach der Freigabe des nicht benötigten Speichers doch einmal weiteren Speicher benötigen, so kann es den Speicher vom GEMDOS über die Funktion MALLOC (Memory Allocate) anfordern. Dazu wird mit der Funktion MALLOC die Anzahl der benötigten Bytes mitgeteilt. Als Return-Wert in D0 erhält man die Startadresse des durch den Aufruf reservierten Speicherbereichs oder eine Fehlermeldung, wenn versucht wird, mehr Speicher zu resevieren als überhaupt noch frei ist.
Wird als Anzahl der zu allocierenden Bytes -1L übergeben, so erhält man die Anzahl der noch freien Bytes in D0 zurück.

1. Beispiel

```
MOVE.L #-1,-(SP)        * Anzahl freier Bytes ermitteln
MOVE.W #$48,-(SP)       * Funktionsnummer
TRAP   #1
ADDQ.L #6,SP            * Anzahl freier Bytes in D0
.
.
```

2. Beispiel:

```
MOVE.L #$1000,-(SP)     * Hex 1000 Bytes dem Programm zuteilen
MOVE.W #$48,-(SP)       * Funktionsnummer
TRAP   #1
ADDQ.L #6,SP
TST.W  D0               * Fehler oder Adresse des Speichers?
BMI    error            * negative Long-Werte = Fehler!
MOVE.L D0,mstart        * sonst Start-Adresse des reservierten Bereichs
.
.
mstart:
       .ds.l   1
```

$49 MFREE

Ein mit MALLOC reservierter Speicher kann jederzeit mit MFREE wieder freigegeben werden. Dazu wird dem GEMDOS die Adresse übermittelt, ab der der Speicher freigegeben werden soll. Der Wert wird üblicherweise die von MALLOC gelieferte Adresse sein.
Bei einem Rückgabewert von Null in D0 wurde der Speicher ohne Fehler vom GEMDOS freigegeben, negative Werte signalisieren in der üblichen Weise Fehler.

```
MOVE.L #mstart,-(SP)    * Adresse eines zuvor allocierten Bereichs
MOVE.W #$49,-(SP)       * Funktionsnummer
TRAP   #1
ADDQ.L #6,SP            * Anzahl freier Bytes in D0
TST.L  D0               * Fehler?
BNE    error            * D0 <> 0 ist Fehler!
.
.
mstart:
       .ds.l   1
```

$4A SETBLOCK

Mit der SETBLOCK-Funktion kann im Gegensatz zur MALLOC-Funktion ein genau bestimmter Speicherbereich reserviert werden. Der darüber liegende Speicher wird an GEMDOS zurückgegeben, auch wenn er zuvor reserviert war. Mit dieser Funktion kann z.B. ein Programm seinen tatsächlichen Speicherbedarf reservieren und den verbleibenden Bereich freistellen.
Als Parameter werden der Funktion die Anfangsadresse und die Länge des zu resevierenden Bereichs übergeben. Der mit diesen Parametern spezifizierte Bereich wird dann vom GEMDOS reserviert und erst mit Beenden des Programms oder nach Aufruf der MFREE-Funktion wieder freigegeben.
Üblicherweise beginnen Programme mit der folgenden oder einer ähnlichen Befehlssequenz. Nach dem Aufruf muß in D0 eine Null stehen, sonst ist ein Fehler aufgetreten.

```
MOVE.L A7,A5          * Stackpointer nach A5 retten
MOVE.L #USTCK,A7      * Stack für das Programm einrichten
MOVE.L 4(A5),A5       * A5 zeigt jetzt auf den Base-Page-Anfang,
                      * genau $100 Bytes unter dem Programmanfang
MOVE.L $C(A5),D0      * $C(A5) enthält Länge des Programm-Bereichs
ADD.L  $14(A5),D0     * $14(A5) enthält Länge des
                      * initialisierten Datenbereichs
ADD.L  $1C(A5),D0     * $1C(A5) enthält Länge des
                      * uninitialisierten Datenbereichs
ADD.L  #$100,D0       * $100 Bytes Base-Page reservieren
MOVE.L D0,-(SP)       * D0 enthält die Länge des zu reserv. Bereichs
MOVE.L A5,-(SP)       * A5 enthält Beginn des zu reserv. Speichers
MOVE.W #0,-(SP)       * ein Wort ohne Bedeutung, trotzdem nötig!
MOVE.W #$4A,-(SP)     * Funktionsnummer
TRAP   #1
ADD.L  #12,SP         * die übliche Aufräumarbeit auf dem Stack
TST.L  D0             * Fehler aufgetreten?
BNE    error          * besser abbrechen, das ging schief
.                     * hier gehts weiter, wenn alles geklappt hat
.
```

$4B EXEC

Die EXEC-Funktion erlaubt das Laden und Nachladen von Programmen. Wahlweise kann das geladene Programm automatisch gestartet werden. Auf dem Stack werden neben der Funktionsnummer die Adressen von drei Strings und ein Moduswort erwartet.
Die erste Adresse ist ein Pointer auf einen sogenannten 'Environment'-String, einen String also, der die 'Umgebung' beschreibt. Soll die Umgebung nicht festgelegt werden, so genügt die Adresse eines 'Null'-Strings, also die Adresse eines Null-Bytes.
Der zweite Pointer kann eine Kommandozeile für das

aufzurufende Programm enthalten. Eine Kommandozeile ist vergleichbar mit der Zeile, die Sie einem Programm vor dem Start eingeben können, wenn Sie im Desktop unter 'Optionen, Anwendung anmelden' den Punkt 'TOS übernimmt Parameter' angewählt haben.
Der dritte Pointer zeigt auf den File- oder Pathname der Datei. Alle drei Strings müssen mit einem Nullbyte abgeschlossen werden, oder aus nur diesem Null-Byte bestehen.
Das Moduswort kann wahlweise Null oder Drei sein. Der Standardwert Null startet das geladene Programm automatisch, eine Drei läd das Programm, ohne automatisch mit der Ausführung zu beginnen. In diesem letzteren Fall erhält man nach dem Laden in D0 entweder die Adresse der Base Page oder eine Fehlermeldung zurück.

```
MOVE.L #env,-(SP)       * Environment
MOVE.L #com,-(SP)       * Kommandozeile
MOVE.L #fil,-(SP)       * Filename
MOVE.W #0,-(SP)         * Laden und Starten bitte
MOVE.W #$4B,-(SP)       * Funktionsnummer
TRAP   #1
ADD.L  #14,SP           * hier kommen wir erst nach dem Ende
.                       * des nachgeladenen Programms an
.
.
fil:                    * Sortierroutine nachladen
       .dc.b   'qsort.prg',0
com:                    * die Datei aufsteigend sortieren
       .dc.b   'up daten.asc',0
env:                    * kein Environment
       .dc.b   0
.
.
```

$4C TERM

TERM $4C stellt nach TERM $00 und TERM $31 die dritte Methode dar, ein Programm zu beenden. TERM $4C stellt den vom Programm belegten Speicher automatisch wieder GEMDOS zur Verfügung. Anders aber als TERM $00 kann mit dieser Funktion dem Aufrufer ein von Null unterschiedlicher und selbst definierbarer Rückgabewert übermittelt werden. Damit hat man die Möglichlichkeit, dem aufrufenden Programm eine wenn auch kurze Nachricht mitzuteilen.

```
MOVE.W #37,-(SP)      * jede nach Bedarf jeder 2-Byte-Wert
MOVE.W #$4C,-(SP)     * Programm jetzt
TRAP   #1             *                beenden
.                     * hier kommen wir nie hin
```

$4E SFIRST

Mit der SFIRST-Funktion kann überprüft werden, ob eine Datei mit dem angegebenen Namen im Directory vorhanden ist. Wird eine gleichnamige Datei gefunden, so erhält man den Filenamen, das File-Attribut, Datum und Uhrzeit der Dateierstellung und die Größe der Datei in Bytes zurück. Die gelieferte Information wird von GEMDOS im DTA-Buffer abgelegt, dessen Adresse ja mit der SETDTA-Funktion bestimmt werden kann.
Besonders angenehm ist die Tatsache, daß der Filename nicht unbedingt vollständig eingegeben werden muß. Einzelne Zeichen im Filenamen können gegen ein Fragezeichen '?' ausgetauscht werden, aber auch ganze Buchstabengruppen können durch einen '*' ersetzt werden. In der extremen Form würde ein Filename auf die Zeichenkette '*.*' reduziert. In diesem Fall würde die erste Datei im Directory den

Anforderungen genügen und der Filename mit den weiteren Informationen im DTA-Buffer erscheinen.
Außer dem Filenamen muß dem SFIRST-Aufruf noch ein Such-Attribut mitgegeben werden. Die möglichen Parameter des Such-Attributs entsprechen den Attributen, die bei der CHMOD-Funktion einer Datei gegeben werden können:

```
$00 = Normaler Zugriff, lesen/schreiben
$01 = Normaler Zugriff, schreibgeschützt
$02 = Verborgener Eintrag (wird im ST-Desktop ignoriert)
$04 = Verborgener System-Eintrag (wie $02 ignoriert)
$08 = Volume Label, Disketten-Name
$10 = Subdirectory
$20 = Datei wurde beschrieben und geschlossen
```

Beim Suchen nach Dateien gelten die folgenden Regeln:
Ist Attribut-Wort Null, so werden nur normale Dateien erkannt. System-Dateien oder Subdirectorys werden nicht erkannt.
System-Files, verborgene Dateien und Subdirectorys werden gefunden, wenn die entsprechenden Attribut-Bits gesetzt sind. Volume-Labels werden aber dann weiterhin nicht erkannt.
Um das Volume-Label zu erhalten, muß diese Option ausdrücklich im Attribut gesetzt werden. Alle anderen Dateien werden dann nicht durchsucht.
Nach dem Aufruf erhält man in D0 den Wert Null, wenn ein entsprechendes File gefunden wurde. In diesem Fall wird der 44 Bytes große DTA-Buffer wie folgt aufgebaut:

```
Byte  0 - 20  für GEMDOS reserviert.
Byte 21       File Attribut
Byte 22 - 23  Uhrzeit der Dateierstellung
Byte 24 - 25  Datum der Dateierstellung
Byte 26 - 29  Dateigröße in Bytes (Lowbyte, Highbyte)
Byte 30 - 43  Name und Extension der gefundenen Datei
```

Wurde jedoch keine dem angegebenen Suchstring entsprechende Datei gefunden, so erhält man die Fehlermeldung -33, File

Not Found.

```
MOVE.L #dta,-(SP)       * DTA-Buffer einrichten
MOVE.W #1A,-(SP)        * Funktionsnummer SETDTA
TRAP   #1
ADDQ.L #6,SP
MOVE.W #attrib,-(SP)    * Attribut-Wert
MOVE.L #filnam,-(SP)    * Filename, nach dem gesucht wird
MOVE.W #$4E,-(SP)       * Funktionsnummer
TRAP   #1
ADDQ.L #8,SP
TST    D0               * Datei gefunden?
BNE    notfound         * scheinbar nicht
.
.
attrib:
       .dc.b   0        * nur normale Files suchen
filnam:
       .dc.b   '*.*',0 * die erste mögliche Datei suchen
.
.
dta:
       .ds.b   44       * Platz für den DTA-Buffer
```

$4F SNEXT

Mit der SNEXT-Funktion (Search Next) kann geprüft werden, ob sich möglicherweiseandere Dateien auf der Diskette befinden, die mit dem angegebenen Filenamen übereinstimmen. Dazu muß nur die Funktionsnummer übergeben werden, Parameter werden von SNEXT nicht benötigt. Allerdings sind alle Parameter bereits vom SFIRST-Aufruf richtig gesetzt.
Wird der Suchstring wie im vorigen Beispiel sehr global gehalten, so können mit SFIRST und SNEXT alle Dateien einer

Diskette nacheinander bestimmt und gegebenenfalls angezeigt werden. Damit hat man in Programmen die Möglichkeit, mit nur wenig Aufwand ein Directory anzuzeigen. Es wird dazu wiederholt die SNEXT-Funktion aufgerufen und nach jedem Aufruf der Inhalt von D0 geprüft. Ergibt sich hier ein von Null unterschiedlicher Wert, so ist entweder ein Fehler aufgetreten, oder es sich alle Directory-Einträge durchsucht.

$56 RENAME

In irgend einer Form ist eine RENAME-Funktion in wohl jedem etwas komfortableren diskettenorientierten Betriebssystem enthalten, da das Umbenennen von Dateien recht häufig nötig ist. Unter GEMDOS reicht zum Umbenennen der Aufruf der Funktion RENAME, der zwei Pointer auf File- oder Pfadnamen übergeben werden. Der erste Pointer zeigt auf den neuen Namen, bei Bedarf auch mit Angabe eines Pfadnamens der Datei, der zweite Pointer zeigt auf den bisherigen Namen. Neben den zwei Pointern wird noch ein 2-Byte-Parameter benötigt. Die Bedeutung des zusätzlich benötigten Wort-Parameters konnte von uns nicht in Erfahrung gebracht werden. Unterschiedliche Werte haben keine (erkennbare) Wirkung.
Als Rückgabewert erhält man in D0 entweder Null, dann ist der Name korrekt geändert worden, oder man erhält einen Fehlercode zurück.

```
MOVE.L #newnam,-(SP)    * neuer Dateiname
MOVE.L #oldnam,-(SP)    * umzubenennende Datei
MOVE.W #0,-(SP)         * Dummy?
MOVE.W #$56,-(SP)       * Funktionsnummer
TRAP   #1
ADD.L  #12,SP
TST.L  D0               * auf Fehler prüfen
.
```

```
.
oldnam:                 * Nullbyte am Ende nicht vergessen!
       .dc.b    'oldfile.dat',0
newnam:
       .dc.b    'newname.dat',0
.
.
```

$57 GSDTOF

Wenn im Desktop das Directory nicht als Icons sondern als Text dargestellt wird, so sieht man neben der Dateigröße in Bytes auch das Datum der Dateierstellung und die Uhrzeit. Diese beiden letzten Angaben können mit der Funktion $57 wahlweise gesetzt oder ermittelt werden. Dazu ist es allerdings erforderlich, das die Datei bereits mit OPEN oder CREATE eröffnet wurde. Die bei der Eröffnung erhaltene Handle-Nummer muß als Parameter übergeben werden. Weitere Parameter sind ein Wort, das als Flag signalisiert, ob Datum und Zeit (Null) gesetzt oder ermittelt (Eins) werden sollen sowie ein Pointer auf einen 4 Byte großen Buffer, der entweder die Ergebnis-Daten erhält, oder der vor dem Aufruf mit den benötigten Daten versorgt wird.
Dieser Datum-Buffer enthält in den beiden ersten Bytes die Zeit, das Datum steht in den letzten Bytes. Das Format der Daten ist identisch mit denen der Funktionen zum Setzen/Ermitteln von Datum und Zeit.

Beispiel 1:

```
MOVE.W #1,-(SP)         * Datum und Zeit ermitteln
MOVE.W #handle,-(SP)    * Datei muß zuerst eröffnet werden
MOVE.L #$buff,-(SP)     * 4 Bytes zum Speichern
MOVE.W #$57,-(SP)       * Funktionsnummer
TRAP   #1
```

```
ADD.L  #10,SP
.
.
handle:
       .ds.b   2
buff:
       .ds.b   4
.
.
```

Beispiel 2:

```
MOVE.W #0,-(SP)         * Datum und Zeit setzen
MOVE.W #handle,-(SP)    * Datei muß zuerst eröffnet werden
MOVE.L #$buff,-(SP)     * 4 Bytes zum Speichern
MOVE.W #$57,-(SP)       * Funktionsnummer
TRAP   #1
ADD.L  #10,SP
.
.
handle:
       .ds.b   2
buff:
       .ds.b   4
.
.
```

Die Fehlerkodes des GEMDOS und ihre Bedeutung

Die GEMDOS-Funktionen liefern als Ergebnis einen Wert, der Aufschluß darüber gibt, ob während der Ausführung der Funktion ein Fehler aufgetreten ist. Ein Wert von Null bedeutet kein Fehler; negative Werte haben folgende Bedeutung:

-32 ungültige Funktionsnummer
-33 Datei nicht gefunden
-34 Pfadname nicht gefunden
-35 zuviele offene Dateien (keine Handles mehr übrig)
-36 Zugriff nicht möglich
-37 ungültige Handle-Nummer
-39 nicht genügend Speicher
-40 ungültige Speicherblockadresse
-46 ungültige Laufwerkbezeichnung
-49 keine weiteren Dateien

Zusätzlich zu diesen Fehlermeldungen können noch Fehlermeldungen des BIOS auftreten, die Nummer von -1 bis -31 haben und im Kapitel 3.3 beschrieben sind.

3.2 Die BIOS-Funktionen des ATARI ST

Die Schnittstelle zwischen dem GEMDOS und der Hardware des Rechners ist das BIOS (Basic Input Output System). Das BIOS kümmert sich, wie der Name schon sagt, um grundlegende Ein/Ausgabe-Funktionen. Dazu gehören die Bildschirmausgabe, Tastatureingabe, Druckerausgabe sowie die RS232-Schnittstelle und nicht zu vergessen die Ein/Ausgabe von und auf Diskette.

Die BIOS-Funktionen stehen auch Anwenderprogrammen zur Verfügung. Zum Aufruf dient ein TRAP-Befehl des 68000-Prozessors. Evtl. Daten werden dabei auf dem Stack übergeben, Ergebnisse der Funktion werden im Register D0 zurückgeliefert. Für den Maschinenspracheprogrammierer ist noch interessant, daß bei einem BIOS-Aufruf die Registerinhalte D0-D2 sowie A0-A2 benutzt werden; die übrigen Registerinhalte bleiben erhalten.

Noch einfacher gestaltet sich der Aufruf von BIOS-Funktionen, wenn Sie in C programmieren. Hier können Sie einfach Funktionsaufrufe mit den entsprechenden Parameterlisten benutzen. Die Funktionsaufrufe sind als Makros in einem include-File enthalten, das die Einbindung ins Betriebssystem übernimmt. Bei den Beispielen werden jeweils die Definition der Funktion und ihrer Parameter in C gezeigt. Für Assemblerprogrammierer ist die Anwendung im einem Beispiel beschrieben.

Für die BIOS-Funktionen ist der TRAP #13 reserviert.

0 getmpb

```
C: void getmpb(zeiger)
   long zeiger;
```

Assembler:

```
move.l  zeiger,-(sp)
move.w  #0,-(sp)
trap    #13
addq.l  #6,sp
```

Diese Funktion füllt einen Block von 12 Bytes Länge, dessen Adresse in 'zeiger' enthalten ist, mit dem sogenannten Memory Parameter Block. Dieser Block enthält selbst wieder drei Zeiger:

```
long    mfl          Memory free list
long    mal          Memory allocated list
long    rover        roving pointer
```

Die Strukturen, auf die jeder Zeiger zeigt, sind folgendermaßen aufgebaut:

```
long    link         Zeiger auf nächsten Block
long    start        Startadresse des Blocks
long    lenght       Länge des Blocks in Bytes
long    own          Process Descriptor
```

Beispiel

```
move.l  #puffer,-(sp)   Puffer für MPB
move.w  #0,-(sp)        getmpb
trap    #13
addq.l  #6,sp
```

Wir erhalten die Werte $48E, 0 und $48E. Ab Adresse $48E stehen folgende Daten:

```
link    0               kein weiterer Block
start   $3B900          Startadresse des freien Speichers
lenght  $3C700          Länge des freien Speichers
own     0               kein Process Descriptor
```

1 bconstat

```
C: int bconstat(dev)
   int dev;
```

Assembler:

```
move.w  dev,-(sp)
move.w  #1,-(sp)
trap    #13
addq.l  #4,sp
```

Diese Funktion liefert den Status eines Eingabegeräts, der folgendermaßen definiert ist:

```
Status  0       kein Zeichen bereit
Status  -1      (mindestens) ein Zeichen bereit
```

Der Parameter 'dev' gibt das Eingabegerät an:

```
dev     Eingabegerät
 0      PRT:, Centronics-Schnittstelle
 1      AUX:, RS232-Schnittstelle
 2      CON:, Tastatur und Bildschirm
 3      MIDI, MIDI-Schnittstelle
 4      IKBD, Tastatur-Port
```

Die folgende Tabelle gibt die erlaubten Zugriffe auf diese Geräte an:

Operation	PRT:	AUX:	CON:	MIDI	IKBD
Eingabe-Status	nein	ja	ja	ja	nein
Eingabe	ja	ja	ja	ja	nein
Ausgabe-Status	ja	ja	ja	ja	ja
Ausgabe	ja	ja	ja	ja	ja

Beispiel:

```
wait move.w  #1,-(sp)          RS232
     move.w  #1,-(sp)          bconstat
     trap    #13
     addq.l  #4,sp
     tst     d0                Zeichen vorhanden ?
     beq     wait              nein, warten
```

Das Beispielprogramm wartet so lange, bis ein Zeichen von der RS232 Schnittstelle bereitsteht.

2 conin

```
C: long conin(dev)
   int  dev;
```

Assembler:

```
   move.w  dev,-(sp)
   move.w  #2,-(sp)
   trap    #13
   addq.l  #4,sp
```

Diese Funktion holt ein Zeichen von einem Eingabegerät ab. Der Parameter 'dev' hat dabei die gleiche Bedeutung wie bei der vorherigen Funktion. Die Funktion kehrt erst dann zurück, wenn ein Zeichen bereitsteht.

Das erhaltene Zeichen steht im untersten Byte des Ergebnisses. War das Eingabegerät die Tastatur ('con', 2), so wird zusätzlich noch der Tasten-Scancode im unteren Byte

des oberen Wortes des Ergebnisses geliefert (siehe Beschreibung des Tastatur-Prozessors).

Beispiel:

```
move.w  #2,-(sp)        con
move.w  #2,-(sp)        bconin
trap    #13
addq.l  #4,sp
```

3 bconout

```
C: void bconout(dev, c)
   int  dev, c;
```

Assembler:

```
move.w  c,-(sp)
move.w  dev,-(sp)
move.w  #3,-(sp)
trap    #13
addq.l  #6,sp
```

Diese Funktion dient zur Ausgabe eines Zeichen 'c' an das Gerät 'dev' (Bedeutung wie bei den obigen Funktionen). Die Funktion kehrt erst dann zurück, wenn das Zeichen ausgegeben wurde.

Beispiel:

```
move.w  #'A',-sp
move.w  #0,-(sp)        PRT:
move.w  #3,-(sp)        bconout
trap    #13
addq.l  #6,sp
```

Das Beispiel gibt den Buchstaben 'A' an den Drucker aus.

4 rwabs

```
C: long rwabs(rwflag, puffer, anzahl, recnr, dev)
   long puffer;
   int  rwflag, anzahl, recnr, dev;
```

Assembler:

```
   move.w  dev,-(sp)
   move.w  recnr,-(sp)
   move.w  anzahl,-(sp)
   move.l  puffer,-(sp)
   move.w  rwflag,-(sp)
   move.w  #4,-(sp)
   trap    #13
   add.l   #14,sp
```

Diese Funktion dient zum Schreiben und Lesen von Sektoren auf Diskette. Die Parameter haben dabei folgende Bedeutung:

rwflag	Bedeutung
0	Sektoren lesen
1	Sektoren schreiben
2	Sektoren lesen, Diskettenwechsel ignorieren
3	Sektoren schreiben, Diskettenwechsel ignorieren

'puffer' ist dabei die Adresse eines Puffers, in den Daten von Diskette gelesen bzw. vom dem die Daten auf Diskette geschrieben werden. Der Puffer sollte an einer geraden Adresse beginnen, da sonst die Übertragung sehr langsam verläuft.

'anzahl' gibt an, wie viele Sektoren bei einem Aufruf gelesen oder geschrieben werden sollen. 'recno' gibt an, bei welchem logischen Sektor dabei begonnen wird.

'dev' schließlich bestimmt, welches Floppylaufwerk benutzt wird:

```
dev     Laufwerk
 0      Drive A
 1      Drive B
 2      Harddisk
```

Die Funktion liefert als Ergebnis einen Fehlercode. Lautet dieser Wert Null, so ist die Operation fehlerfrei verlaufen. Bei einem Fehler steht der negative Fehlercode dort. Die Bedeutung können Sie bei der Funktion 'floprd' des XBIOS nachlesen.

Beispiel:

```
   move.w  #0,-(sp)          Laufwerk A
   move.w  #10,-(sp)         Beginn bei logischem Sektor 10
   move.w  #2,-(sp)          2 Sektoren lesen
   move.l  #puffer,-(sp)     Pufferadresse
   move.w  #0,-(sp)          Sektoren lesen
   move.w  #4,-(sp)          rwabs
   trap    #13
   add.l   #14,sp
   ...
puffer     ds.b    2*512
```

5 setexec

```
C: long setexec(nummer, vector)
   int  nummer;
   long vector;
```

Assembler:

```
   move.l  vector,-(sp)
   move.w  nummer,-(sp)
   move.w  #5,-(sp)
   trap    #13
   addq.l  #8,sp
```

Die Funktion setexec erlaubt es, einen Exception-Vektor des 68000-Prozessors zu verändern. Dazu müssen die Nummer des Vektors in 'nummer', die Adresse der zugehörigen Routine in 'vector' enthalten sein. Die Funktion liefert als Ergebnis den alten Vektor zurück. Will man den Vektor nur lesen, so übergibt man als neuen Wert -1. Mit dieser Funktion lassen sich die 256 Vektoren des Prozessors sowie 8 Vektoren für GEM setzen, die die Nummern $100 bis $107 haben (Adressen $400 bis $41C).

Beispiel:

```
   move.l  #buserror,-(sp)
   move.w  #2,-(sp)
   move.w  #5,-(sp)
   trap    #13
   addq.l  #8,sp
   ...
buserror  ...
```

6 tickcal

```
C: long tickcal()
```

Assembler:

```
   move.w  #6,-(sp)
   trap    #13
   addq.l  #2,sp
```

Diese Funktion liefert als Ergebnis die Anzahl der Millisekunden zwischen zwei System-Timer-Aufrufen.

Beispiel:

```
   move.w  #6,-(sp)
   trap    #13
   addq.l  #2,sp
```

Ergebnis: 20 ms

7 getbpb

```
C: long getbpb(dev)
   int  dev;
```

Assembler:

```
move.w  dev,-(sp)
move.w  #7,-(sp)
trap    #13
addq.l  #4,sp
```

Diese Funktion liefert einen Zeiger auf den BIOS-Parameter-Block des Drives 'dev' (0 = Laufwerk A, 1 = Laufwerk B).

Der BPB ist folgendermaßen aufgebaut:

```
int  recsiz    Sektorgröße in Bytes
int  clsiz     Clustergröße in Sektoren
int  clsizb    Clustergröße in Bytes
int  rdlen     Directorylänge in Sektoren
int  fsiz      FAT-Größe in Sektoren
int  fatrec    Sektornummer des zweiten FAT
int  datrec    Sektornummer des ersten Daten-Clusters
int  numcl     Anzahl der Daten-Cluster auf der Diskette
int  bflags    diverse Flags
```

Die Funktion liefert für Laufwerk A die Adresse $3E3E, für Laufwerk B die Adresse $3E5E. Eine Adresse von Null kennzeichnet einen Fehler.

Beispiel:

```
move.w  #0,-(sp)        Laufwerk A
move.w  #7,-(sp)        getbpb
trap    #13
```

```
addq.l  #4,sp
```

Hier die Daten des BPB für 80 Track single sided und double sided Laufwerke:

Parameter	80 Track SS	80 Track DS
recsiz	512	512
clsiz	2	2
clsizb	1024	1024
rdlen	7	7
fsiz	5	5
fatrec	6	6
datrec	18	18
numcl	351	711

8 bcostat

```
C: long bcostat(dev)
   int  dev;
```

Assembler:

```
move.w  dev,-(sp)
move.w  #8,-(sp)
trap    #13
addq.l  #4,sp
```

Diese Funktion prüft, ob das mit 'dev' angegebene Ausgabegerät bereit ist, das nächste Zeichen auszugeben. 'dev' kann die Werte annehmen, die in Funktion Eins beschrieben wurden. Das Ergebnis dieser Funktion lautet entweder -1, falls das Ausgabegerät bereit ist, oder Null, falls gewartet werden muß.

Beispiel:

```
move.w  #0,-(sp)          Drucker bereit ?
move.w  #8,-(sp)          bcostat
```

```
trap    #13
addq.l  #4,sp
```

9 mediach

```
C: long mediach(dev)
   int  dev;
```

Assembler:

```
move.w  dev,-(sp)
move.w  #9,-(sp)
trap    #13
addq.l  #4,sp
```

Mit dieser Funktion läßt sich ermitteln, ob zwischenzeitlich eine Diskette gewechselt wurde. Dazu muß dem Parameter 'dev' die Laufwerksnummer übergeben werden (0 = Laufwerk A, 1 = Laufwerk B). Als Ergebnis können drei Werte auftreten:

```
0     Diskette wurde definitiv nicht gewechselt
1     Diskette kann gewechselt worden sein
2     Diskette wurde definitiv gewechselt
```

Beispiel:

```
move.w  #1,-(sp)        Laufwerk B
move.w  #9,-(sp)        mediach
trap    #13
addq.l  #4,sp
```

10 drvmap

```
C: long drvmap()
```

Assembler:

```
move.w  #10,-(sp)
trap    #13
addq.l  #2,sp
```

Diese Funktion liefert einen Bitvektor, der die angeschlossenen Laufwerke enthält. Die Bit-Nummer n ist dabei gesetzt, wenn Laufwerk n verfügbar ist (0 bedeutet wieder A usw.). Auch wenn nur ein Laufwerk angeschlossen, wird %11 geliefert, da von zwei logischen Laufwerken ausgegangen wird.

Beispiel:

```
move.w  #10,-(sp)       drvmap
trap    #13
addq.l  #2,sp
```

11 kbshift

```
C: long kbshift(modus)
   int  modus;
```

Assembler:

```
move.w  modus,-(sp)
move.w  #11,-(sp)
trap    #13
addq.l  #4,sp
```

Mit dieser Funktion können Sie den Status der Sondertasten der Tastatur erfahren oder verändern. Ist 'modus' -1, so erhalten Sie den Status, ein positiver Wert wird als Status übernommen. Der Status ist ein Bit-Vektor, der folgendermaßen aufgebaut ist:

Bit	Bedeutung
0	rechte Shift-Taste
1	linke Shift-Taste

2	Control-Taste
3	ALT-Taste
4	Caps Lock eingeschaltet
5	rechte Maus-Taste (CLR/HOME)
6	linke Maus-Taste (INSERT)
7	unbenutzt

Beispiel:

```
move.w  #-1,-(sp)       Shift-Status lesen
move.w  #11,-(sp)       kbshift
trap    #13
addq.l  #4,sp
```

3.3 Die erweiterten BIOS-Funktionen des ATARI ST

Um die speziellen Hardware-Möglichkeiten des ATARI ST zu unterstützen, gibt es neben den normalen BIOS-Funktionen noch die erweiterten BIOS-Funktionen, die über einen TRAP# 14 aufgerufen werden. Der Aufruf der Funktionen kann wie bei den normalen BIOS-Funktionen sowohl von Assembler als auch von C aus erfolgen. Beim Aufruf von C aus ist wieder ein kleiner TRAP-Handler in Maschinensprache erforderlich, der so aussehen kann:

```
trap14:
        move.l  (sp)+,retsave  Rücksprungadresse merken
        trap    #14            XBIOS aufrufen
        move.l  retsave,-(sp)  Rücksprungadresse zurückholen
        rts

        .bss
retsave ds.l    1              Platz für Rücksprungadresse
```

Zum C-Compiler gehören noch Makro-Definitionen, die den Aufruf der erweiterten BIOS-Funktionen (eXtended BIOS, XBIOS) über Namen erlaubt. Bei der Makroexpansion werden aus dem Namen wieder die entsprechende Funktionsnummer und der TRAP-Aufruf erzeugt.

Beim Arbeiten mit Assembler wird einfach die Funktionsnummer des XBIOS auf dem Stack übergeben. Das XBIOS hat 40 verschiedene Funktionen, deren Bedeutung und Anwendung auf den folgenden Seiten beschrieben wird.

0 initmous

```
C: void initmous(typ, parameter, vector)
   int  typ;
   long parameter, vector;
```

Assembler:

```
   move.l  vector,-(sp)
   move.l  parameter,-(sp)
   move.w  typ,-(sp)
   move.w  #0,-(sp)
   trap    #14
   add.l   #12,sp
```

Mit Hilfe dieser XBIOS-Funktion werden die Routinen zur Mouse-Bearbeitung initialisiert. 'vector' ist dabei die Adresse einer Routine, die bei einem Mouse-Report vom Tastaturprozessor angesprungen wird. 'typ' wählt unter folgenden Alternativen aus:

typ	
0	Disable mouse
1	Enable mouse, relativer Modus
2	Enable mouse, absoluter Modus
3	unbenutzt
4	enable mouse, Tastatur-Modus

Damit können Sie also auswählen, ob Mausbewegungen gemeldet werden und auf welche Weise dies geschieht.

'parameter' zeigt auf einen Parameterblock, der folgendermaßen aufgebaut ist:

```
char topmode
char buttons
char xparam
char yparam
```

'topmode' entscheidet über die Anordnung des Koordinatensystems. 0 bedeutet Y=0 liegt in der unteren Ecke, 1 bedeutet daß die Y-Achse von oben aus zählt. 'buttons' ist ein Parameter für den Befehl 'set mouse buttons' des Tastatur-Prozessors (siehe Beschreibung des IKBD, intelligent keyboard). 'xparam' und 'yparam' sind Skalierungsfaktoren für die Mouse-Bewegung. Wenn Sie 'typ' zu 2 gewählt haben, also mit absolut mouse arbeiten, so schließen sich an den Parameterblock noch weitere vier Parameter an:

```
int  xmax
int  ymax
int  xstart
int  ystart
```

Dabei handelt es sich um die X- und Y-Koordinaten des Maximalwerts, den die Mouse-Position annehmen kann sowie den Startwert, auf den die Mouse gesetzt wird.

Beispiel:

```
   move.l  #vector,-(sp)       Adresse der Mouse-Routine
   move.l  #parameter,-(sp)    Adresse des Parameterblocks
   move.w  #1,-(sp)            enable relative mouse
   move.w  #0,-(sp)            init mouse
   trap    #14
   add.l   #12,sp
   ...
parameter  dc.b  ......
   ...
vector     ...                 Mouse Interrupt-Routine
```

1 ssbrk

```
C: long ssbrk(anzahl)
   int  anzahl;
```

Assembler:

```
   move.w  anzahl,-(sp)
   move.w  #1,-(sp)
   trap    #14
   addq.l  #4,sp
```

Diese Funktion reserviert Speicherplatz. Die Anzahl der Bytes muß dabei in 'anzahl' übergeben werden. Der Speicherplatz wird am oberen Ende des Speichers bereitgestellt. Die Funktion liefert als Ergebnis die Adresse des reservierten Speicherbereichs. Diese Funktion muß vor der Initialisierung des Betriebssystems aufgerufen werden, d.h. sie muß noch vom Boot-ROM ausgeführt werden, ehe das Betriebssystem geladen wird.

Beispiel:

```
   move.w  #$400,-(sp)          1 KByte reservieren
   move.w  #1,-(sp)             ssbrk
   trap    #14
   addq.l  #4,sp
```

2 physbase

```
C:  long physbase()
```

Assembler:

```
    move   #2,-(sp)
    trap   #14
    addq.l #2,sp
```

Diese Funktion liefert die Basis des physikalischen Bildschirm-RAMs. Unter der physikalischen Bildschirmadresse versteht man den Speicherbereich, der vom Videoshifter dargestellt wird. Das Ergebnis ist ein Langwort.

Beispiel:
$78000, Basisadresse des Bildschirms bei 512 K RAM

3 logbase

```
C:  long logbase()
```

Assembler:

```
move   #3,-(sp)
trap   #14
addq.l #2,sp
```

Die logische Bildschirmbasis ist die Adresse, die bei allen Ausgabefunktionen als Bildschirmbasis benutzt wird. Sind physikalische und logische Bildschirmbasis verschieden, so wird ein Bildschirm dargestellt während in einem anderen RAM-Bereich bereits ein anderes Bild aufgebaut wird, welches dann später angezeigt wird. Das Ergebnis dieses Funktionsaufrufs ist wieder ein Langwort.

Beispiel:
$78000, Basisadresse des Bildschirms bei 512 K RAM

4 getrez

```
C:  int getrez()
```

Assembler:

```
move.w #4,-(sp)
trap   #14
addq.l #2,sp
```

Dieser Funktionsaufruf liefert die Bildschirmauflösung zurück:

```
0 :=  Low Resolution, 320*200 Punkte, 16 Farben
1 :=  Medium Resolution, 640*200 Punkte, 4 Farben
2 :=  High Resolution, 640*400 Punkte, monochrom
```

Beispiel:
2, monochrom

5 setscreen

```
C:  void setscreen(logadr, physadr, rez)
    long logadr, physadr;
    int  rez;
```

Assembler:

```
move.w  res,-(sp)
move.l  physadr,-(sp)
move.l  logadr,-(sp)
move.w  #5,-(sp)
trap    #14
add.l   #10,sp
```

Diese Funktion dient zum Ändern der Bildschirmparameter, die mit den vorherigen drei Funktion gelesen werden konnten. Soll ein Parameter nicht gesetzt werden, so muß ein negativer Wert übergeben werden. Die Parameter werden in der nächsten VBL-Routine gesetzt, damit keine Störungen auf dem Bildschirm auftreten.

Beispiel:
Die physikalische und die logische Bildschirmadresse soll auf $70000 gesetzt, die Auflösung soll beibehalten werden.

```
move.w  #-1,-(sp)          Auflösung erhalten
move.l  #$70000,-(sp)      physikalische Basis
move.l  #$70000,-(sp)      logische Basis
```

```
    move.w  #5,-(sp)          setscreen
    trap    #14
    add.l   #10,sp
```

6 setpalette

```
C: void setpalette(paletteptr)
   long paletteptr;
```

Assembler:

```
    move.l  palettptr,-(sp)
    move.w  #6,-(sp)
    trap    #14
    addq.l  #6,sp
```

Mit dieser Funktion kann eine neue Farbpalette geladen werden. Dazu muß paletteptr ein Zeiger auf eine Tabelle mit 16 Farben (je ein Wort) sein. Die Adresse der Tabelle muß gerade sein. Die Farben werden zu Beginn des nächsten VBL geladen.

Beispiel:

```
    move.l  #palette,-(sp)     Adresse der neuen Farbpalette
    move.w  #6,-(sp)           setpalette
    trap    #14
    addq.l  #6,sp
    ....
palette   dc.w   $777,$700,$070,$007,$111,$222,$333,$444,
                 $555,$000,$001,$010,$100,$200,$020,$002,
                 $123,$456
```

7 setcolor

```
C: int setcolor(colornum, color)
   int colornum, color
```

Assembler:

```
    move.w  color,-(sp)
```

```
move.w  colornum,-(sp)
move.w  #7,-(sp)
trap    #14
addq.l  #6,sp
```

Diese Funktion erlaubt es, nur eine Farbe zu verändern. Dazu müssen die Farbnummer (0-15) und die zugehörige Farbe (0 - $777) angegeben werden. Wenn als Farbe -1 angegeben wird, wird die Farbe nicht gesetzt, und Sie erhalten nur den alten Farbwert zurück.

Beispiel:

```
move.w  #$777,-(sp)         Farbe weiß
move.w  #1,-(sp)            als Farbnummer 1
move.w  #7,-(sp)
trap    #14
addq.l  #6,sp
```

8 floprd

```
C: int floprd(puffer, filler,  dev,  sector,  track,  seite,
count)
   long  puffer, filler;
   int   dev, sector, track, seite, count;
```

Assembler:

```
move.w   count,-(sp)
move.w   seite,-(sp)
move.w   track,-(sp)
move.w   sector,-(sp)
move.w   dev,-(sp)
clr.l    -(sp)
move.l   puffer,-(sp)
move.w   #8,-(sp)
trap     #14
add.l    #20,sp
```

Mit dieser XBIOS-Funktion lassen sich ein oder mehrere Sektoren von Diskette lesen. Die Parameter haben dabei folgende Bedeutung:

count: Gibt an, wieviel Sektoren nacheinander gelesen werden. Dabei sind Werte zwischen eins und neun (Anzahl der Sektoren pro Track) möglich.

seite: Wählt die Diskettenseite aus, null bei einseitigen Laufwerken und null oder eins bei doppelseitigen Laufwerken.

track: Damit wird die Tracknummer bestimmt (0 -79 bei 80-Spur-Laufwerken oder 0-39 bei 40-Spur-Laufwerken).

sector: Die Sektornummer des ersten zu lesenden Sektors (0 - 9).
dev: Bestimmung der Laufwerksnummer, 0 für Laufwerk A und 1 für Laufwerk B.

filler: unbenutztes Langwort.

puffer: Puffer, in den die Diskettendaten geschrieben werden sollen. Der Puffer muß an einer Wortgrenze beginnen und groß genug für die zu lesenden Daten sein (512 Bytes mal Anzahl der Sektoren).

Die Funktion liefert als Ergebnis einen Fehlercode, der folgende Bedeutung hat:

```
 0  Ok, kein Fehler
-1  allgemeiner Fehler
-2  Drive not ready
-3  unbekannter Befehl
-4  CRC Error
-5  Bad request, ungültiger Befehl
-6  Seek Error, Track nicht gefunden
-7  Unknown media (ungültiger Bootsektor)
-8  Sektor nicht gefunden
-9  (Kein Papier)
```

```
-10  Schreibfehler
-11  Lesefehler
-12  allgemeiner Fehler
-13  Diskette schreibgeschützt
-14  Diskette wurde gewechselt
-15  Unbekanntes Gerät
-16  Bad Sectors (beim Verifizieren)
-17  Diskette einlegen (bei einem angeschlossenen Laufwerk)
```

Beispiel:

```
    move.w   #1,-(sp)            einen Sektor lesen
    move.w   #0,-(sp)            Seite Null
    move.w   #0,-(sp)            Track Null
    move.w   #1,-(sp)            Sektor Eins
    move.w   #1,-(sp)            Laufwerk B
    clr.l    -(sp)
    move.l   #puffer,-(sp)
    move.w   #8,-(sp)            floprd
    trap     #14
    add.l    #20,sp
    tst      d0                  Fehler aufgetreten ?
    bmi      error               ja
    ...
puffer  ds.b 512                 Puffer für einen Sektor
```

9 flopwr

```
C: int floprd(puffer, filler,  dev,  sector,  track,  seite,
count)
   long  puffer, filler;
   int   dev, sector, track, seite, count;
```

Assembler:

```
    move.w   count,-(sp)
    move.w   seite,-(sp)
    move.w   track,-(sp)
    move.w   sector,-(sp)
```

```
    move.w   dev,-(sp)
    clr.l    -(sp)
    move.l   puffer,-(sp)
    move.w   #9,-(sp)
    trap     #14
    add.l    #20,sp
```

Mit dieser XBIOS-Funktion lassen sich ein oder mehrere Sektoren auf Diskette schreiben. Die Parameter haben dabei die gleiche Bedeutung wie bei der Funktion 8 floprd. Die Funktion liefert als Ergebnis wieder einen Fehlercode, der die gleich Bedeutung wie beim Sektorlesen hat.

Beispiel:

```
    move.w   #3,-(sp)          drei Sektoren schreiben
    move.w   #0,-(sp)          Seite Null
    move.w   #7,-(sp)          Track sieben
    move.w   #1,-(sp)          Sektor Eins
    move.w   #0,-(sp)          Laufwerk A
    clr.l    -(sp)
    move.l   #puffer,-(sp)     Adresse des Puffers
    move.w   #9,-(sp)          flopwr
    trap     #14
    add.l    #20,sp
    tst      d0                Fehler aufgetreten ?
    bmi      error             ja
    ...
puffer  ds.b 3*512             Puffer für drei Sektoren
```

10 flopfmt

```
C: int flopfmt (puffer, filler, dev, spt, track, seite,
                interleave, magic, virgin)
   long puffer, filler, magic;
   int  dev, spt, track, seite, interleave, virgin;
```

Assembler:

```
move.w   virgin,-(sp)
move.l   magic,-(sp)
move.w   interleave,-(sp)
move.w   seite,-(sp)
move.w   track,-(sp)
move.w   spt,-(sp)
move.w   dev,-(sp)
clr.l    -(sp)
move.l   puffer,-(sp)
move.w   #10,-(sp)
trap     #14
add.l    #26,sp
```

Diese Routine dient zum Formatieren eines Tracks auf der Diskette. Die Parameter haben dabei folgende Bedeutung:

virgin: Mit diesem Wert werden die Sektoren beim Formatieren beschrieben. Als Standardwert hierzu empfiehlt sich $E5E5. Das Hi-Nibble jedes Bytes darf dabei nicht den Wert $F enthalten.

magic: Als magic muß die Konstante $87654321 benutzt werden, sonst wird die Formatierung abgebrochen.

interleave: Bestimmt, in welcher Reihenfolge die Sektoren auf Diskette geschrieben werden, normalerweise eins.

seite: Damit wird die Diskettenseite ausgewählt (0 oder 1).

track: Die Nummer des Tracks, der formatiert werden soll (0 - 79).

spt: Sektoren pro Track, normalerweise 9.

dev: Das Laufwerk, 0 für A und 1 für B.

filler: unbenutztes Langwort

puffer: Puffer für die kompletten Trackdaten; bei 9 Sektoren pro Track muß der Puffer mindestens 8 KByte groß sein.

Die Funktion gibt als Ergebnis einen Fehlercode zurück. Der Wert -16, bad sectors, bedeutet, daß bei einigen Sektoren die Daten nicht korrekt zurückgelesen werden konnten. In diesem Fall steht im Puffer eine Liste von schlechten Sektoren (Wort-Daten, durch Null abgeschlossen). Sie können daraufhin nochmal formatieren oder die Sektoren als schlecht markieren.

Beispiel:

```
   move.w   #$E5E5,-(sp)          Initialdaten
   move.l   #$87654321,-(sp)      magic
   move.w   #1,-(sp)              interleave
   move.w   #0,-(sp)              Seite 0
   move.w   #79,-(sp)             Track 79
   move.w   #9,-(sp)              9 Sektoren pro Track
   move.w   #0,-(sp)              Laufwerk A
   clr.l    -(sp)
   move.l   #puffer,-(sp)
   move.w   #10,-(sp)             flopfmt
   trap     #14
   add.l    #26,sp
   tst      d0
   bmi      error

puffer     ds.b   $2000          8K Puffer
```

11 unbenutzt

12 midiws

```
C: void midiws (count,ptr)
   int  count;
   long ptr;
```

Assembler:

```
move.l  ptr,-(sp)
move.w  count,-(sp)
move.w  #12,-(sp)
trap    #14
addq.l  #8,sp
```

Mittels dieser Funktion ist es möglich, einen String an die MIDI-Schnittstelle auszugeben (MIDI OUT). ptr muß dabei auf den String zeigen, count muß die Anzahl der Zeichen minus 1 enthalten, die gesendet werden.

Beispiel:

```
   move.l  #string,-(sp)         Adresse des Strings
   move.w  #stringend-string-1,-(sp)    Länge
   move.w  #12,-(sp)             midiws
   trap    #14
   addq.l  #8,sp

   ....

string     dc.b  'MIDI-Daten'
stringend  equ   *
```

13 mfpint

```
C: void mfpint (nummer, vector)
   int  nummer;
   long vector;
```

Assembler:

```
   move.l  vector,-(sp)
   move.w  nummer,-(sp)
   move.w  #13,-(sp)
   trap    #14
   addq.l  #8,sp
```

Damit kann eine Interruptroutine im MFP initialisiert werden. In 'nummer' muß dabei die Nummer des MFP-Interrupts stehen (0-15), während 'vector' die Adresse der zugehörigen Interruptroutine enhält. Der alte Interrupt-Vektor wird dabei überschrieben.

Beispiel:

```
   move.l  #busy,-(sp)       Busy-Interruptroutine
   move.w  #0,-(sp)          Vektor-Nummer 0
   move.w  #13,-(sp)         mfpint
   trap    #14
   addq.l  #8,sp
   ....
busy:
```

14 iorec

```
C: long iorec(dev)
   int  dev;
```

Assembler:

```
   move.w  dev,-(sp)
   move.w  #14,-(sp)
   trap    #14
   addq.l  #4,sp
```

Mit dieser Funktion wird ein Zeiger auf einen Puffer-Datensatz für ein Eingabegerät geholt. Folgende Eingabegeräte können angegeben werden:

dev	Eingabegerät
0	RS232
1	Tastatur
2	MIDI

Der Pufferdatensatz für ein Eingabegerät hat folgenden Aufbau:

```
long    ibuf      Zeiger auf Eingabepuffer
int     ibufsize  Größe des Eingabepuffers
int     ibufhd    Head Index
int     ibuftl    Tail Index
int     ibuflow   Low water mark
int     ibufhi    High water mark
```

Bei dem Eingabepuffer handelt es sich um einen zirkulären Puffer, der Head Index gibt die nächste Schreibposition an (der Puffer wird von einer Interruptroutine gefüllt) und der Tail Index gibt an, von wo aus dem Puffer gelesen werden kann. Sind Head und Tail Index gleich, so ist der Puffer leer. Low und High water mark werden im Zusammenhang mit der Kommunikationskontrolle bei RS232 benutzt (XON/XOFF oder RTS/CTS). Ist der Eingabepuffer bis zur High water mark gefüllt, so wird dem Sender über XOFF bzw. CTS mitgeteilt, daß der Rechner keine weiteren Daten mehr empfangen kann. Werden die empfangenen Daten dann vom Rechner weiterverarbeitet, so daß der Pufferinhalt unter die Low water mark sinkt, so wird die Übertragung wieder freigegeben.

Für die RS232-Ausgabe existiert ein gleich aufgebauter Puffer-Datensatz, der sich direkt an den Eingabe-Datensatz anschließt.

Beispiel:

```
move.w  #1,-(sp)        Pufferdatensatz für Tastatur
move.w  #14,-(sp)       iorec
trap    #14
addq.l  #4,sp
....
```

Ergebnis: $9F2

Die folgende Tabelle enthält die Daten für alle Geräte.

	RS232 Eingabe	RS232 Ausgabe	Tastatur	MIDI
Adresse	$9D0	($9DE)	$9F2	$A00
Pufferadresse	$6D0	$7D0	$8D0	$950
Pufferlänge	$100	$100	$80	$80
Head Index	0	0	0	0
Tail Index	0	0	0	0
Low water mark	$40	$40	$20	$20
High water mark	$C0	$C0	$20	$20

Head und Tail Index sind natürlich vom augenblicklichen Betriebszustannd abhängig. High und Low water mark sind jeweils auf 3/4 bzw. 1/4 der Puffergröße gesetzt. Sie sind nur bei eingschaltetem XON/XOFF bzw. RTS/CTS bei RS232 von Bedeutung.

15 rsconf

```
C: void rsconf(baud, ctrl, ucr, rsr, tsr, scr)
   int  baud, ctrl, ucr, rsr, tsr, scr;
```

Assembler:

```
move.w  scr,-(sp)
move.w  tsr,-(sp)
move.w  rsr,-(sp)
move.w  ucr,-(sp)
move.w  ctrl,-(sp)
move.w  baud,-(sp)
move.w  #15,-(sp)
trap    #14
add.l   #14,sp
```

Diese XBIOS-Funktion dient zum Konfigurieren der RS232 Schnittstelle. Die Parameter haben dabei folgende Bedeutung:

```
scr:  Synchronuous Character Register im MFP
tsr:  Transmitter Status Register im MFP
rsr:  Receiver Status Register im MFP
```

```
ucr:  USART Control Register im MFP
ctrl: Kommunikationsparameter
baud: Baudrate
```

Zur Bedeutung der MFP-Register schlagen Sie bitte im Kapitel über den MFP 68901 nach. Ist einer der Parameter -1, so wird der vorherige Wert beibehalten. Mit dem ctrl-Parameter können Sie den Handshake-Modus wählen:

ctrl	Bedeutung
0	kein Handshake, default nach dem Einschalten
1	XON/XOFF
2	RTS/CTS
3	XON/XOFF und RTS/CTS (nicht sinnvoll)

Der baud-Parameter enthält eine Kennziffer für die Baudrate.

baud	Baudrate
0	19200
1	9600
2	4800
3	3600
4	2400
5	2000
6	1800
7	1200
8	600
9	300
10	200
11	150
12	134
13	110
14	75
15	50

Beispiel:

```
move.w  #-1,-(sp)
move.w  #-1,-(sp)          MFP-Register nicht verändern
```

```
move.w  #-1,-(sp)
move.w  #-1,-(sp)
move.w  #1,-(sp)          XON/XOFF
move.w  #9,-(sp)          300 Baud
move.w  #15,-(sp)         rsconf
trap    #14
add.l   #14,sp
```

16 keytbl

```
C: long keytbl (unshift, shift, capslock)
   long unshift, shift, capslock;
```

Assembler:

```
move.l  capslock,-(sp)
move.l  shift,-(sp)
move.l  unshift,-(sp)
move.w  #16,-(sp)
trap    #14
addq.l  #14,sp
```

Mit dieser Funktion ist es möglich, eine neue Tastaturbelegung zu erzeugen. Dazu müssen Sie die Adressen der neuen Tabellen übergeben, die die Tastencodes bei normalen Tasten (ohne Shift), bei gedrückter Shift-Taste und bei Capitels Lock enthalten. Die Funktion liefert als Ergebnis die Adresse der Vektortabelle, in der die drei Tastaturtabellenzeiger stehen. Soll eine Tabelle unverändert bleiben, so muß als Adresse -1 übergeben werden. Ein Tastaturtabelle muß 128 Bytes lang sein. Sie wird über den Tasten-Scancode adressiert und liefert als Ergebnis den ASCII-Code der jeweiligen Taste.

Beispiel:

```
move.l  #-1,-(sp)         Caps Lock nicht ändern
move.l  #shift,-(sp)      Shift-Tabelle
```

```
   move.l  #unshift,-(sp)    Tabelle ohne Shift
   move.w  #16,-(sp)
   trap    #14
   addq.l  #14,sp

   ....
shift:  ...
unshift ...
```

17 random

```
C: long random()
```

Assembler:

```
   move.w  #17,-(sp)
   trap    #14
   addq.l  #2,sp
```

Diese Funktion liefert eine 24-Bit-Zufallszahl. Die Bit 24-31 sind dabei Null. Bei jedem Aufruf erhalten Sie ein anderes Ergebnis. Nach dem Einschalten des Rechners wird jedesmal ein anderer Startwert erzeugt.

Beispiel:

```
   move.w  #17,-(sp)        random
   trap    #14
   addq.l  #2,sp
```

18 protobt

```
C: void protobt(puffer, serialnr, disktyp, execflag)
   long puffer, serialnr;
   int  disktyp, execflag;
```

Assembler:

```
move.w  execflag,-(sp)
move.w  disktyp,-(sp)
move.l  serialnr,-(sp)
move.l  puffer,-(sp)
move.w  #18,-(sp)
trap    #14
add.l   #14,sp
```

Diese Funktion dient dazu, einen Bootsektor zur erzeugen. Ein Bootsektor steht auf Track 0, Sektor 1 auf Seite 0 einer Diskette und gibt dem DOS Auskunft über den Diskettentyp. Ist der Bootsektor ausführbar, kann damit das Betriebssystem geladen werden. Mit dieser Funktion können Sie einen neuen Bootsektor erzeugen, etwa für ein anderes Diskettenformat, oder einen vorhandenen Bootsektor verändern. Die Parameter haben folgende Bedeutung:

execflag: Damit wird entschieden, ob der Bootsektor ausführbar ist:

```
0   nicht ausführbar
1   ausführbar
-1  Bootsektor bleibt so wie er war
```

Der Disktyp kann folgende Werte annehmen:

```
0   40 Track, single sided  (180 K)
1   40 Track, double sided  (360 K)
2   80 Track, single sided  (360 K)
3   80 Track, double sided  (720 K)
-1  Disktyp bleibt unverändert
```

Serialnr ist eine 24-Bit-Seriennummer, die in den Bootsektor geschrieben wird. Ist die Seriennummer größer als 24 Bits ($01000000), so wird eine Zufalls-Seriennummer erzeugt (mit obiger Funktion). Ein Wert von -1 bedeutet, daß die Seriennummer nicht verändert wird.

Puffer ist die Adresse eines 512-Byte-Puffers, der den Bootsektor enthält bzw. in dem der Bootsektor erzeugt wird.

Ein Bootsektor hat folgenden Aufbau:

Adresse		40 Track SS	40 Track DS	80 Track SS	80 Track DS
0 -1	Branch-Befehl auf Bootprogramm sofern ausführbar				
2 -7	'Loader'				
8 -10	24-Bit-Seriennummer				
11-12	BPS	512	512	512	512
13	SPC	1	2	2	2
14-15	RES	1	1	1	1
16	FAT	2	2	2	2
17-18	DIR	64	112	112	112
19-20	SEC	360	720	720	1440
21	MEDIA	252	253	248	249
22-23	SPF	2	2	5	5
24-25	SPT	9	9	9	9
26-27	SIDE	1	2	1	2
28-29	HID	0	0	0	0
510-511	CHECKSUM				

Die Abkürzungen haben dabei folgende Bedeutung:

BPS: Bytes pro Sektor. Die Sektorgröße ist bei allen Formaten einheitlich 512 Bytes

SPC: Sektoren pro Cluster. Die Anzahl der Sektoren, die vom DOS zu einem Block zusammengefaßt werden, 2 Sektoren gleich 1 KByte.

RES: Anzahl der reservierten Sektoren am Beginn der Diskette einschließlich dem Bootsektor.

FAT: Die Anzahl der File Allocation Tables auf der Diskette.

DIR: Die maximale Anzahl der Directory-Einträge.

SEC: Die Gesamtzahl der Sektoren auf der Diskette.

MEDIA: Media Descriptor Byte, wird vom ST-BIOS nicht benutzt.

SPF: Anzahl der Sektoren in jedem FAT.

SPT: Anzahl Sektoren pro Track.

SIDE: Anzahl der Seiten auf der Diskette.

HID: Anzahl der versteckten Sektoren auf der Diskette.

Der Bootsektor ist kompatibel zu MS-DOS 2.x. Deshalb sind alle 16-Bit-Worte im 8086-Format abgespeichert (erst Low-Byte, dann Hi-Byte).

Ist die Checksumme des gesamten Bootsektors $1234, so ist der Sektor ausführbar. In diesem Falle steht ab Adresse 30 das Bootprogramm.

Beispiel:

```
move.w  #-1,-(sp)        Ausführbarkeit nicht ändern
move.w  #3,-(sp)         80 Tracks, DS
move.l  #-1,-(sp)        Seriennummer nicht ändern
move.l  #puffer,-(sp)
move.w  #18,-(sp)        protobt
trap    #14
add.l   #14,sp

puffer ds.b 512
```

Mit diesem Beispielprogramm kann z.B. ein vorhandener Bootsektor für 80 Tracks, double sided, angepaßt werden.

19 flopver

```
C: int flopver(puffer, filler, dev,  sector,  track,  seite,
count)
   long  puffer, filler;
   int   dev, sector, track, seite, count;
```

Assembler:

```
   move.w  count,-(sp)
   move.w  seite,-(sp)
   move.w  track,-(sp)
   move.w  sector,-(sp)
   move.w  dev,-(sp)
   clr.l   -(sp)
   move.l  puffer,-(sp)
   move.w  #19,-(sp)
   trap    #14
   add.l   #16,sp
```

Diese Funktion dient zum Verifizieren eines oder mehrerer Sektoren auf der Diskette. Dabei werden die Sektoren von Diskette gelesen und mit dem Pufferinhalt im Speicher verglichen. Die Parameter haben die gleiche Bedeutung wie bei den Funktionen zum Lesen und Schreiben von Sektoren. Stimmen Sektor- und Pufferinhalt überein, so wird als Ergebnis der Funktion Null zurückgegeben. Bei einem Fehler steht die negative Fehlernummer in D0. Die Bedeutung dieses Fehlercodes steht bei der Funktion 'Sektor lesen'. Im Fehlerfalle steht im Puffer eine Liste der fehlerhaften Sektoren (16-Bit-Werte), die durch ein Nullwort abgeschlossen ist. Benutzt man zum Schreiben von Sektoren die BIOS-Funktion 5 rwabs und ist die Variable fverify ($444) gesetzt, so werden die Sektoren nach dem Schreiben automatisch verifiziert.

Beispiel:

```
move.w  #1,-(sp)          ein Sektor
move.w  #0,-(sp)          Seite Null
move.w  #39,-(sp)         Track 39
move.w  #1,-(sp)          Sektor 1
move.w  #0,-(sp)          Laufwerk A
clr.l   -(sp)
move.l  #puffer,-(sp)     Pufferadresse
move.w  #19,-(sp)         flopver
trap    #14
add.l   #16,sp
tst     d0                Fehler ?
bmi     error
```

20 scrdmp

```
C: void scrdmp()
```

Assembler:

```
move.w  #20,-(sp)
trap    #14
addq.l  #2,sp
```

Mit dieser Funktion wird eine Bildschirm-Hardcopy auf einen angeschlossenen Drucker ausgegeben. Dabei werden die voreingestellten Druckerparameter ('Drucker Anpassung') benutzt. Diese Funktion können Sie jederzeit auch durch gleichzeitiges Drücken der Tasten ALT und HELP auslösen oder vom Desktop aus durch 'Bildschirm Drucken'.

Beispiel:

```
move.w  #20,-(sp)     Hardcopy
trap    #14           XBIOS aufrufen
addq.l  #2,sp
```

21 cursconf

```
C: int cursconf(funktion, rate)
   int funktion, rate;
```

Assembler:

```
move.w  rate,-(sp)
move.w  funktion,-(sp)
move.w  #21,-(sp)
trap    #14
addq.l  #6,sp
```

Diese XBIOS-Funktion dient zur Bestimmung der Cursor-Funktion. 'funktion' kann dabei Werte von 0-5 haben, die folgende Bedeutung haben:

funktion	Bedeutung
0	Cursor abschalten
1	Cursor einschalten
2	Cursor blinkend
3	Cursor nicht blinkend
4	Cursor Blinkrate setzen
5	Cursor Blinkrate holen

Sie können mit dieser Funktion also bestimmen, ob der Cursor sichtbar ist, ob er feststehend ist oder blinkt. Ein Ergebnis liefert diese XBIOS-Funktion nur, wenn Sie die alte Blinkrate holen. Die Einheit der Blinkfrequenz ist abhängig von der Bildfrequenz: Bei einem Monochrommonitor beträgt Sie 70 Hz; ein Farbmonitor wird mit 50 Hz betrieben. Mit Funktionsnummer 5 können Sie eine neue Blinkrate einstellen. Den Parameter 'rate' brauchen Sie nur dann benutzen, wenn Sie eine neue Blinkrate übergeben wollen.

Beispiel:

```
move.w  #20,-(sp)           20/70 Sekunden
move.w  #4,-(sp)            Blinkrate setzen
```

```
move.w  #21,-(sp)            cursconf
trap    #14
addq.l  #6,sp
```

22 settime

```
C: void settime(time)
   long time;
```

Assembler:

```
move.l  time,-(sp)
move.w  #22,-(sp)
trap    #14
addq.l  #6,sp
```

Mit dieser Funktion können Sie Uhrzeit und Datum neu setzen. Die Zeit wird dabei im unteren Wort von time übergeben, das Datum im oberen Wort. Zeit und Datum sind folgendermaßen kodiert:

```
Bit 0-4    Sekunden in Zweierschritten
Bit 5-10   Minuten
Bit 11-15  Stunden

Bit 16-20  Tag 1-31
Bit 21-24  Monat 1-12
Bit 25-31  Jahr 0-119 (minus Offset 1980)
```

Beispiel:

```
move.l  #%10110011000001000000000000000,-(sp)
move.w  #22,-(sp)       settime
trap    #14
addq.l  #6,sp
```

Dieser Aufruf setzt das Datum auf den 16. September 1985 und die Uhrzeit auf 8 Uhr.

23 gettime

```
C: long gettime()
```

Assembler:

```
move.w  #23,-(sp)
trap    #14
addq.l  #2,sp
```

Diese Funktion liefert das aktuelle Datum und die Uhrzeit im oben beschriebenen Format.

Beispiel:

```
move.w  #23,-(sp)     gettime
trap    #14
addq.l  #2,sp
move.l  d0,time       Zeit und Datum merken
```

24 bioskeys

```
C: void bioskeys()
```

Assembler:

```
move.w  #24,-(sp)
trap    #14
addq.l  #2,sp
```

Wenn Sie mit der XBIOS-Funktion 16, keytbl, eine neue Tastaturbelegung gewählt haben, stellt diese Funktion wieder die Standardtastaturbelegung des BIOS her. Sie können diese Funktion z.B. vor dem Verlassen eines eigenen Programms aufrufen, das die Tastaturbelegung verändert hat.

Beispiel:

```
   move.w  #24,-(sp)      bioskeys
   trap    #14
   addq.l  #2,sp
```

25 ikbdws

```
C: void ikbdws(anzahl, zeiger)
   int  anzahl;
   long zeiger;
```

Assembler:

```
   move.l  zeiger,-(sp)
   move.w  anzahl,-(sp)
   move.w  #25,-(sp)
   trap    #14
   addq.l  #8,sp
```

Zum Übermitteln von Befehlen an den Tastaturprozessor (intelligent keyboard) dient diese XBIOS-Funktion. 'zeiger' ist dabei die Adresse eines Strings, der gesendet werden soll, 'anzahl' ist die Länge des Strings minus 1.

Beispiel:

```
   move.l  #string,-(sp)             Adresse des Strings
   move.w  #strend-string-1,-(sp)    Länge minus 1
   move.w  #25,-(sp)                 ikbdws
   trap    #14
   addq.l  #8,sp
   ...
string     dc.b   $80,1
strend     equ    *
```

26 jdisint

```
C: void jdisint(nummer)
   int  nummer;
```

Assembler:

```
   move.w  nummer,-(sp)
   move.w  #26,-(sp)
   trap    #14
   addq.l  #4,sp
```

Diese Funktion ermöglicht es, einen Interrupt des MFP 68901 selektiv zu sperren. Als Parameter wird dabei die Interruptnummer des MFP übergeben (0 - 15). Die Bedeutung der einzelnen Interrupts ist im Kapitel "Interrupt" beschrieben.

Beispiel:

```
   move.w  #10,-(sp)    RS232 Transmitter Interrupt sperren
   move.w  #26,-(sp)    disable Interrupt
   trap    #14
   addq.l  #4,sp
```

27 jenabint

```
C: void jenabint(nummer)
   int  nummer;
```

Assembler:

```
   move.w  nummer,-(sp)
   move.w  #27,-(sp)
   trap    #14
   addq.l  #4,sp
```

Soll ein Interrupt des MFP wieder freigegeben werden, so können Sie dazu diese Funktion benutzen. Übergeben wird dabei wieder die Nummer des Interrupts von 0 - 15.

Beispiel:

```
move.w  #12,-(sp)    RS232 Receiver Interrupt freigeben
move.w  #27,-(sp)    enable Interrupt
trap    #14
addq.l  #4,sp
```

28 giaccess

```
C: char giaccess(daten, register)
   char daten;
   int  register;
```

Assembler:

```
move.w  #register,-(sp)
move.w  #daten,-(sp)
move.w  #28,-(sp)
trap    #14
addq.l  #6,sp
```

Diese Funktion erlaubt den Zugriff auf die Register des GI-Soundchips. 'register' muß dabei die Registernummer des Soundchips enthalten (0-15). Die Bedeutung der einzelnen Register entnehmen Sie bitte der Hardware-Beschreibung des Soundchips. Ob das angegebene Register gelesen werden soll oder ob in das Register geschrieben wird, wird durch Bit 7 der Registernummer entschieden:

```
Bit 7   0:  lesen
        1:  schreiben
```

Beim Schreiben wird in 'daten' der zu schreibende 8-Bit-Wert übergeben; beim Lesen liefert die Funktion den Inhalt des

entsprechenden Registers.

Beispiel:

```
move.w   #$80+3,-(sp)      Register 3 schreiben
move.w   #$50,-(sp)        zu schreibender Wert
move.w   #28,-(sp)         giaccess
trap     #14
addq.l   #6,sp
```

29 offgibit

```
C: void offgibit(bitnummer)
   int  bitnummer;
```

Assembler:

```
move.w  #bitnummer,-(sp)
move.w  #29,-(sp)
trap    #14
addq.l  #4,sp
```

Mit diesem Funktionsaufruf läßt sich selektiv ein Bit im Port A des Soundchips setzen. Port A ist ein 8-Bit Ausgabe-Port, in dem die einzelnen Bits folgende Bedeutung haben:

```
Bit 0:   Floppy Seite 0/ Seite 1 selektieren
Bit 1:   Floppy Laufwerk A selektieren
Bit 2:   Floppy Laufwerk B selektieren
Bit 3:   RS232 RTS (Request to Send)
Bit 4:   RS232 DTR (Data Terminal Ready)
Bit 5:   Centronics Strobe
Bit 6:   General Purpose Output
Bit 7:   unbenutzt
```

Beispiel:

```
move.w  #4,-(sp)        DTR-Bit
move.w  #29,-(sp)       ongibit
trap    #14
addq.l  #4,sp
```

30 ongibit

```
C: void ongibit(bitnummer)
   int  bitnummer;
```

Assembler:

```
move.w  #bitnummer,-(sp)
move.w  #30,-(sp)
trap    #14
addq.l  #4,sp
```

Das Gegenstück zu der obigen Funktion bildet diese. Damit ist es möglich, ein Bit im Port A des Soundchips zu löschen.

Beispiel:

```
move.w  #4,-(sp)        DTR-Bit
move.w  #30,-(sp)       ongibit
trap    #14
addq.l  #4,sp
```

31 xbtimer

```
C: void xbtimer(timer, control, data, vector)
   int  timer, control, data;
   long vector;
```

Assembler:

```
move.l  vector,-(sp)
move.w  data,-(sp)
```

```
move.w  control,-(sp)
move.w  timer,-(sp)
move.w  #31,-(sp)
trap    #14
add.l   #12,sp
```

Mit dieser Funktion können Sie einen Timer im MFP 68901 starten und ihm eine Interruptroutine zuordnen. 'timer' ist die Nummer des Timers im MFP:

```
Timer A :  0
Timer B :  1
Timer C :  2
Timer D :  3
```

'data' und 'control' sind die Werte, die in das entsprechende Kontroll- und Datenregister des Timers übertragen werden. Beachten Sie dazu die Hardware-Beschreibung des MFP 68901. 'vector' ist die Adresse der zugehörigen Interruptroutine, die nach Ablauf des Timers angesprungen wird. Die vier Timer des MFP werden teilweise schon vom Betriebssystem benutzt:

```
Timer A:  Für den Endbenutzer reserviert
Timer B:  Horizontal Blank Zähler
Timer C:  200 Hz System Timer
Timer D:  RS232 Baudrate-Generator
          (der zugehörige Interruptvektor ist frei)
```

Beispiel:

```
move.l  #vector,-(sp)      Interruptroutine
move.w  data,-(sp)         Daten-
move.w  control,-(sp)      und Kontroll-Register
move.w  #0,-(sp)           Timer A
move.w  #31,-(sp)          xbtimer
trap    #14
add.l   #12,sp
```

32 dosound

```
C: void dosound(zeiger)
   long zeiger;
```

Assembler:

```
   move.l  zeiger,-(sp)
   move.w  #32,-(sp)
   trap    #14
   addq.l  #6,sp
```

Diese Funktion erlaubt eine automatische komfortable Abarbeitung eines Sounds. Dazu muß 'zeiger' auf einen String aus Soundbefehlen zeigen. Folgende Befehle können benutzt werden:

Befehle $00 - $0F
Diese Befehle werden als Registernummer des Soundchips interpretiert. Ihnen folgt jeweils ein Byte, das in das entsprechende Register geladen wird.

Befehl $80
Diesem Befehl folgt ein Argument, das in ein temporäres Register geladen wird.

Befehl $81
Diesem Befehl müssen drei Argumente folgen. Das erste Argument ist die Nummer des Registers des Soundchips, in das der Inhalt des temporären Registers geladen wird. Das zweite Argument ist ein Zweierkomplementwert, der zu dem temporären Register addiert wird. Das dritte Argument enthält ein Endekriterium. Das Ende ist erreicht, wenn der Inhalt des temporären Registers gleich dem Endekriterium ist.

Befehle $82 - $FF
Diesen Befehlen folgt jeweils ein weiteres Argument. Lautet dieses Argument null, so wird die Soundverarbeitung abgebrochen. Ansonsten gibt dieses Argument an, wieviele

Timerticks (20ms, 50 Hz) bis zur nächsten Verarbeitung des Sounds vergehen.

Beispiel:

```
   move.l  #zeiger,-(sp)     Zeiger auf Sound-Befehle
   move.w  #32,-(sp)         dosound
   trap    #14
   addq.l  #6,sp
   ....
zeiger     dc.b  0,10,1,50,...
```

33 setprt

```
C: void setprt(config)
   int  config;
```

Assembler:

```
   move.w  config,-(sp)
   move.w  #33,-(sp)
   trap    #14
   addq.l  #4,sp
```

Mittels dieser Funktion kann die eingestellte Druckerkonfiguration gelesen oder verändert werden. Enhält 'config' den Wert -1, so wird der augenblickliche Wert geliefert, ansonsten wird der Wert als neue Druckerkonfiguration übernommen.

Die Druckerkonfiguration ist ein Bitvektor mit folgender Bedeutung:

Bitnummer	0	1
0	Matrixdrucker	Typenraddrucker
1	Farbdrucker	Monochromdrucker
2	ATARI-Drucker	Epson-Drucker
3	Test-Modus	Qualitäts-Modus

```
   4      Centronics-Port   RS232-Port
   5      Endlospapier      Einzelblatt
 6 - 14          reserviert
  15             immer 0
```

Beispiel:

```
move.w  #%000100,-(sp)    Epson-Drucker
move.w  #33,-(sp)         setprt
trap    #14
addq.l  #4,sp
```

34 kbdvbase

```
C: long kbdvbase()
```

Assembler:

```
move.w  #34,-(sp)
trap    #14
addq.l  #2,sp
```

Diese XBIOS-Funktion liefert einen Zeiger auf eine Vektortabelle, die die Adressen von Routinen enthält, die Daten vom Tastaturprozessor verarbeiten. Die Tabelle ist folgendermaßen aufgebaut:

```
long   midivec        MIDI-Eingabe
long   vkbderr        Tastatur Error
long   vmiderr        MIDI Error
long   statvec        IKBD Status
long   mousevec       Mouse-Routinen
long   clockvec       Uhrzeit-Routine
long   joyvec         Joystick-Routinen
```

'midivec' zeigt auf eine Routine, die die empfangenen Daten vom MIDI-Eingang (Byte in D0) in den MIDI-Puffer schreibt.

'vkbderr' und 'vmiderr' werden aufgerufen, wenn ein Überlauf von der Tastatur- oder MIDI-ACIA gemeldet wird.

Die restlichen vier Routinen 'statvec', 'mousevec', 'clockvec' und 'joyvec' verarbeiten die entsprechenden Datenpakete, die von der Tastatur-ACIA kommen. Diesen Routinen wird ein Zeiger auf das empfangene Paket in A0 und auf dem Stack übergeben. Der Mouse-Vektor wird von GEM benutzt. Wenn Sie eigene Routinen benutzen wollen, so müssen sie mit RTS abgeschlossen werden und dürfen nicht länger als eine Millisekunde Verarbeitungszeit benötigen.

Beispiel:

```
move.w  #34,-(sp)      kbdvbase
trap    #14
addq.l  #2,sp
```

Als Ergebnis erhalten wir $A0E. Das Vektorfeld enthält folgende Werte:

```
midivec     $79C6
vkbderr     $759C
vmiderr     $759C
statvec     $7034
mousevec    $15296
clockvec    $6A46
joyvec      $7034
```

35 kbrate

```
C: int kbrate(verzoeg, wiederhol)
   int verzoeg, wiederhol;
```

Assembler:

```
move.w  wiederhol,-(sp)
move.w  verzoeg,-(sp)
```

```
move.w  #35,-(sp)
trap    #14
addq.l  #6,sp
```

Mit dieser Funktion kann der Tastatur-Repeat kontrolliert werden. Der Parameter 'verzoeg' gibt die Verzögerungszeit nach dem Drücken einer Taste an, nach der die Taste automatisch wiederholt wird. Der Parameter 'wiederhol' bestimmt die Zeitspanne, nach der der Tastendruck jeweils wiederholt wird. Diese Werte können Sie vom Desktop aus mittels des Kontrollfeldes mit den beiden Schiebereglern ändern. Die Zeiten basieren auf den 50-Hz-Systemtakt. Wird für einen der Parameter -1 angegeben, so wird der entsprechende Wert nicht gesetzt. Die Funktion liefert die vorherigen Werte als Ergebnis; Bit 0-7 enthalten den 'wiederhol'-Wert und Bit 8-15 den Wert von 'verzoeg'.

Beispiel:

```
move.w  #-1,-(sp)       alte Werte lesen
move.w  #-1,-(sp)
move.w  #35,-(sp)       kbrate
trap    #14
addq.l  #6,sp
```

Ergebnis: D0 = $0B03

36 prtblk

```
C: void prtblk(parameter)
   long parameter;
```

Assembler:

```
move.l  parameter,-(sp)
move.w  #36,-(sp)
trap    #14
addq.l  #6,sp
```

Diese Funktion ähnelt der Funktion scrdmp (20) und wird von dieser benutzt. Die Funktion erwartet jedoch eine Parameterliste, deren Adresse übergeben wird. Diese Liste ist folgendermaßen aufgebaut:

```
long    blkprt    Adresse des Bildschirm-RAMs
int     offset
int     width     Bildschirmbreite
int     height    Bildschirmhöhe
int     left
int     right
int     scrres    Bildschirmauflösung (0, 1 oder 2)
int     dstres    Druckerauflösung (0 oder 1)
long    colpal    Adresse der Farbpalette
int     type      Druckertyp (0-3)
int     port      Druckerport (0=Centronics, 1=RS232)
long    masks     Zeiger auf Halbtonmaske
```

Beispiel:

```
   move.l  #parameter,-(sp)      Adresse des Parameterblocks
   move.w  #36,-(sp)             prtblk
   trap    #14
   addq.l  #6,sp
   ....
parameter  dc.l  ...
```

37 wvbl

```
C: void wvbl()
```

Assembler:

```
   move.w  #36,-(sp)
   trap    #14
   addq.l  #2,sp
```

Diese Funktion wartet auf den nächsten Bildrücklauf. Man kann sie z.B. benutzen, um Grafikausgaben mit dem Strahlrücklauf zu synchronisieren.

Beispiel:

```
   move.w  #36,-(sp)      wvbl abwarten
   trap    #14
   addq.l  #2,sp
```

38 supexec

```
C: void supexec(adresse)
   long adresse;
```

Assembler:

```
   move.l  adresse,-(sp)
   move.w  #38,-(sp)
   trap    #14
   addq.l  #6,sp
```

Soll eine Routine im Supervisor-Modus des 68000-Prozessors ausgeführt werden, so können Sie dies mit dieser Funktion erreichen. Dazu übergibt man einfach die Adresse der Routine an diese Funktion.

Beispiel:

```
   move.l  #adresse,-(sp)
   move.w  #38,-(sp)
   trap    #14
   addq.l  #6,sp
   ...
adresse    move.l $400,d0
   ...
```

39 puntaes

```
C: void puntaes()
```

Assembler:

```
move.w  #39,-(sp)
trap    #14
addq.l  #2,sp
```

Mit dieser Funktion kann das AES abgeschaltet werden, sofern es sich nicht im ROM befindet.

Beispiel:

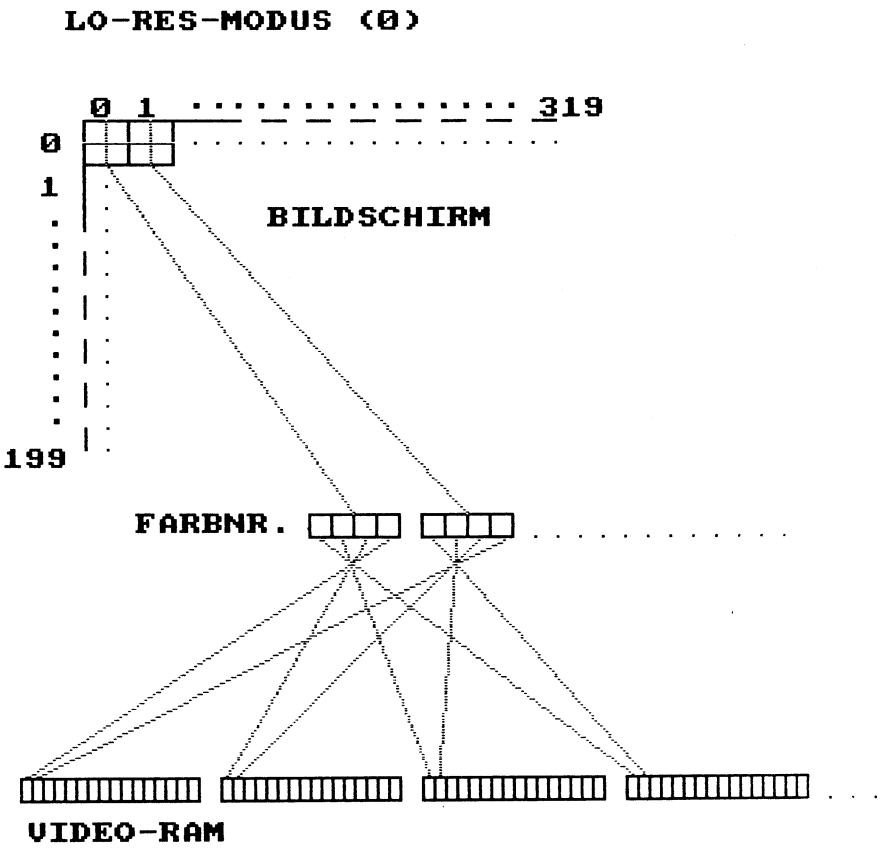
```
move.w  #39,-(sp)          AES abschalten
trap    #14
addq.l  #2,sp
```

3.4 Die Grafik des ATARI ST

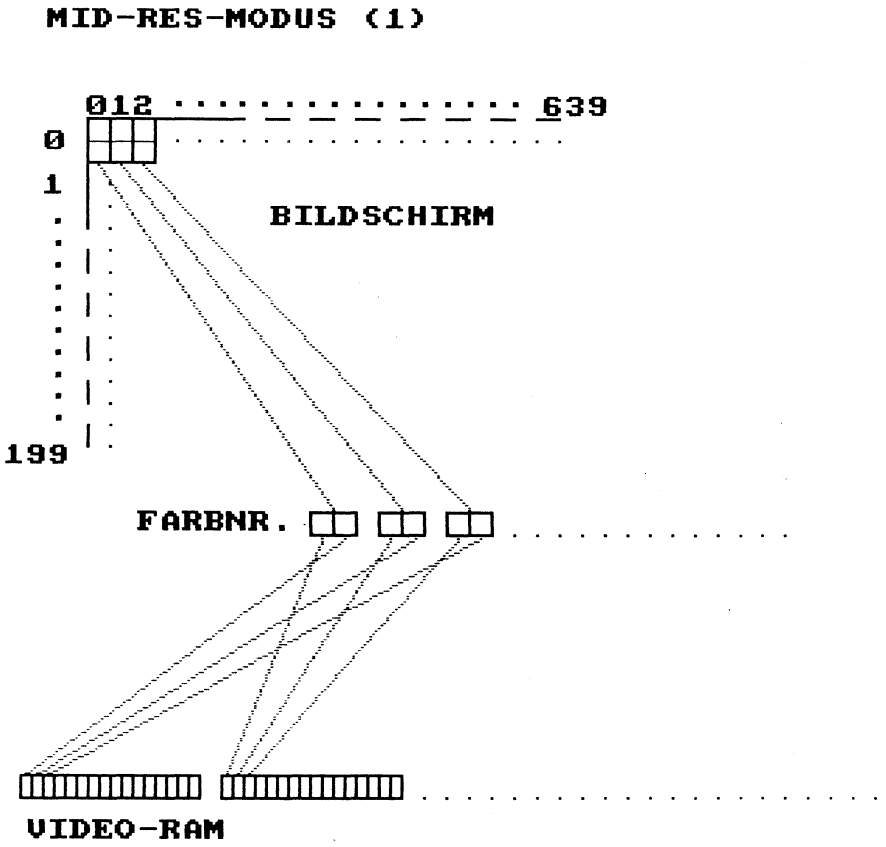
Neben der hohen Rechengeschwindigkeit und dem zur Verfügung stehenden großen Speicher ist sicher die Grafik die auf den ersten Blick faszinierendste Eigenschaft des ST. Mit dem normalerweise gelieferten Monochrom-Monitor und der Auflösung von 640x400 Punkten schlägt er in seiner Preis/Leitungsklasse alles bisher dagewesene. Aber auch bei der Farbauflösung kann sich der ATARI mit 16 Farben bei 320x200 Bildpunkten durchaus sehen lassen.
Wie die Grafik organisiert ist, und wie man auch ohne Benutzung des für den Anfänger recht komplexen Grafik-Pakets GEM zu ansehnlichen und schnellen Grafiken kommen kann, wollen wir in diesem Kapitel klären. Der ATARI stellt nämlich dem Programmierer (Assembler, aber auch C) sehr hilfreiche Routinen zur Verfügung, mit deren Hilfe die Grafik-Programmierung zwar nicht gerade zum Kinderspiel wird, die einem aber doch eine ganze Menge Programmierarbeit abnehmen können. Leider sind einige dieser Funktionen so umfangreich, daß eine detaillierte Beschreibung den Rahmen dieses Buches bei weitem sprengen würde. Wir haben uns darum auf die einfacheren, nicht jedoch weniger interessanten Funktionen beschränken müssen.
Diese Grafik-Routinen werden in sehr eleganter Weise aufgerufen. Die Software-Entwickler haben sich die Tatsache zu Nutze gemacht, das es im 68000 zwei Gruppen von Opcodes gibt, die vom 68000 nicht 'verstanden' werden und einen Trap, also einen Software-Interrupt auslösen, wenn sie in einem Programm auftreten. Es sind dies die beiden Gruppen von Opcodes, die mit $Axxx und $Fxxx beginnen. Im ATARI wird der Trap der $Axxx-Opcodes genutzt, um die Grafik-Routinen anzusprechen. Dazu überprüft der Trap-Handler, also das durch den Trap aufgerufene Programm, welchen Wert das unterste Byte des 'Befehls' hat. Hier sind Werte zwischen Null und $E zugelassen. Damit ergeben sich insgesamt 14 Grafikroutinen, die zunächst einmal übersichtsmäßig vorgestellt werden sollen. Später werden wir dann auf die wesentlichen Befehle im Detail zu sprechen kommen.

$A000 Adressen der benötigten Variablenbereiche ermitteln
$A001 Punkt auf dem Bildschirm setzen
$A002 Farbe eines Bildschirmpunktes ermitteln
$A003 Linie auf dem Bildschirm ziehen
$A004 horizontale Linie ziehen (sehr schnell!)
$A005 Rechteck mit Farbe füllen
$A006 Vieleck Zeile nach Zeile füllen
$A007 Bit Block Transfer
$A008 Text Block Transfer
$A009 Maus-Cursor einschalten
$A00A Maus-Cursor ausschalten
$A00B Maus-Cursor-Form ändern
$A00C Sprite löschen
$A00D Sprite einschalten
$A00E Raster Form copieren

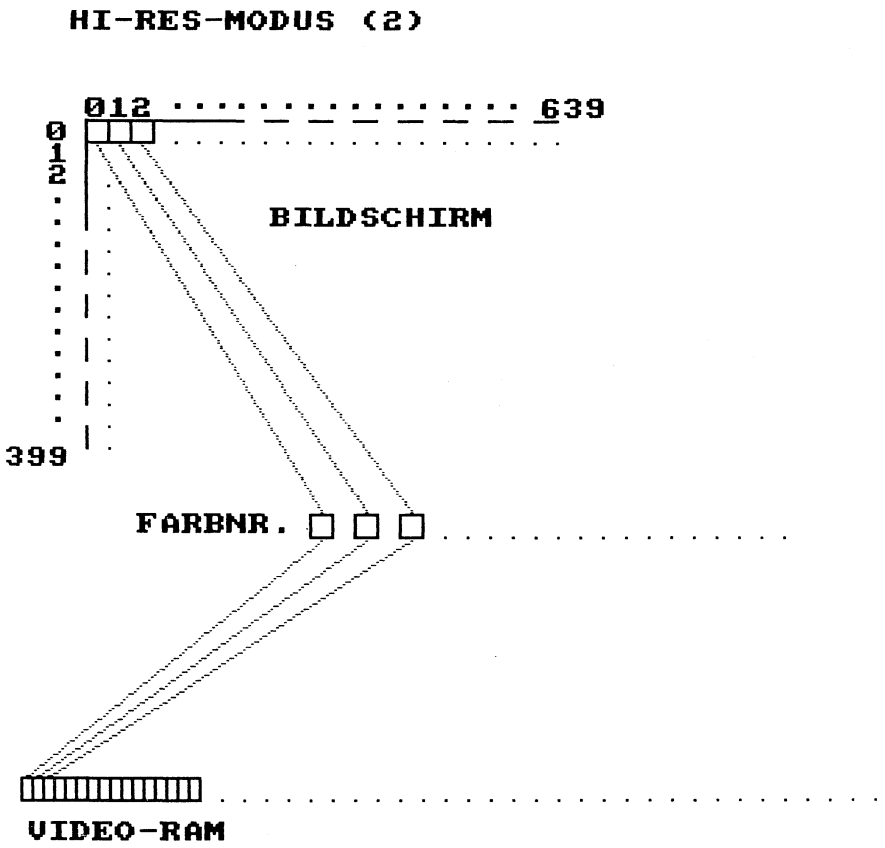
Diese Routinen stellen quasi das Grundgerüst, den hardwareabhängigen Teil des GEM dar. Alle GEM-Grafik- und Textausgaben werden über die Routinen der $Axxx-Opcodes ausgeführt. Besonders für Spiele lassen sich die Line A-Opcodes vorteilhaft einsetzen. Hier werden Windows ja nur in den seltensten Fällen benötigt. Ein weiterer wesentlicher Punkt ist die Geschwindigkeit der Line A-Befehle. Die direkte Nutzung der Grafik-Routinen ist deutlich schneller, als wenn die Ausgaben über das GEM vorgenommen würden. Bevor wir jetzt die einzelnen Befehle detailiert beschreiben, betrachten wir kurz den Aufbau der Grafik in den verschiedenen Grafik-Modi des ST.
Unmittelbar nach dem Einschalten des ST wird ein Bereich von 32 KByte von der oberen Speichergrenze als Video-RAM initialisiert. Damit ergibt sich im normalen Betrieb eine Adresslage von $78000 bis $7FFFF als Bildschirm-Ram. Dieses Video-RAM kann wie ein Fenster in den ST betrachtet werden. Beginnen wir mit dem einfachsten Modus, dem 640x400-Modus. In diesem Fall bilden je 80 Bytes oder besser gesagt 40 Worte (Doppelbytes) eine Bildschirmzeile. Das Wort mit der niedrigsten Adresse wird am linken Bildschirm-Rand dargestellt, die weiteren Worte werden kontinuierlich von links nach rechts dargestellt. Innerhalb eines Worts liegt


LO-RES-MODUS (0)
0 1 319
0
1
199
BILDSCHIRM
FARBNR.
VIDEO-RAM

das oberste, höchstwertige Bit links, das niederwertigste Bit ganz rechts.
Mit diesen Daten kann nun jeder Punkt des Bildschirms recht einfach angesteuert oder ausgelesen werden. Um zum Beispiel den ersten Bildschirm-Punkt zu setzen, muß in die Speicherstelle $78000 der Wert $8000 geschrieben werden. Allerdings muß an dieser Stelle eine kleine Einschränkung gemacht werden. Die Lage des Bildschirm-RAMs läßt sich im ST recht einfach verschieben. Dann stimmt die absolute Adresse $78000 natürlich nicht mehr. Aus diesem Grund ist es in den meisten Fällen günstiger, den Punkt über die Line 'A'-Funktion $A001 zu setzen. Diese Routine geht von einem X-Y-Koordinatensystem mit Ursprung in der linken oberen Ecke aus und bestimmt selber die Lage des Video-RAMs, um den Punkt an der richtigen Stelle zu setzen.
In diesem Auslösungsmodus wird also jeder Bildpunkt durch ein Bit representiert. Ist das Bit gesetzt, so erscheint der Punkt dunkel oder auch hell, wenn im Color-Palette-Register 0 der inverse Darstellungsmodus gewählt ist. Der Bildschirm besteht aus nur einer Bit-Ebene, Plane genannt. Unterschiedliche Farben jedoch lassen sich mit nur einer Bit-Ebene (einer Plane) nicht darstellen. Darum wird die Auflösung in den Farbmodi mit steigender Anzahl gleichzeitig darstellbarer Farben geringer.
In der Auflösungsstufe 640x200 Punkte sind immerhin schon 4 Farben möglich. In dieser Darstellungsart bilden immer zwei aufeinander folgende Speicherworte eine zusammengehörende Einheit. Die Farbe eines Bildpunkts wird bestimmt durch den Wert der beiden gleichwertigen Bits in den beiden Worten. Sind beide Bits Null, so ergibt sich die Hintergrundfarbe. Für die Pixel-Darstellung gehören also zwei aufeinander folgende Worte zusammen. Für die Farben jedoch gehören alle ungeraden Worte zu einer Plane. Die zweite Plane wird aus den geraden Worten gebildet. In diesem Modus verfügt nam also über zwei Planes (Farb-Bitebenen).
Richtig bunt wird die Sache dann aber im Modus mit 'nur' 320x200 Punkten. In dieser Betriebsart bilden dann sogar 4 aufeinander folgende Speicher-Worte eine Einheit, die über die Farbe der 16 zugehörigen Pixel bestimmen. Um bei dem


MID-RES-MODUS (1)
012
639
0
1
199
BILDSCHIRM
FARBNR.
VIDEO-RAM

zuvor bereits einmal benutzen Beispiel zu bleiben: um den Punkt in der linken oberen Ecke des Bildschirms zu setzen, müssen die vier Worte $78000, $78002, $78004 und $78006 in ihren obersten Bits manipuliert werden. Je nach Bitmuster in den Worten ergibt sich dann die gewünschte Farbe. Das erfordert natürlich schon einiges an Rechenaufwand, um einen Punkt Modus-unabhänig in der jeweils gewünschten Farbe zu setzen. All diese Arbeit wird einem jedoch von der $A001-Routine abgenommen. Diese Routine setzt alle zusammen gehörenden Bits der jeweils eingestellten Auflösung für die gewünschte Farbe. In diesem Modus sind natürlich auch vier Planes vorhanden. Die erste Plane wird, um bei unserem Beispiel zu bleiben, von den Worten an den Adressen $7F000, $7F008, $7F010 ..., die weiteren Planes von den entsprechend anderen Adressen gebildet.
Ein weiterer Punkt, den es noch zu klären gibt, betrifft die Fonts oder Zeichensätze. Da der ST keinen Text-Modus sondern nur den Grafik-Modus kennt, muß auch die Text-Ausgabe in hochauflösender Grafik geschehen. Bisher sind im ST drei verschiedene Fonts eingebaut. Man kann aber jederzeit weitere Fonts von Diskette nachladen. Jeder Font hat einen Header, also einen Vorspann, der wichtige Informationen über die darstellbaren Zeichen enthält. Da die wesentlichen Daten im Font-Header abgelegt sind, ist man bei der Darstellung ungewöhnlich wenig eingeschränkt. Die Zeichen können annähernd beliebig hoch und breit sein. Das Zeitalter der 8x8-Matrix bei Character-Ausgabe ist vorbei. Sogar echte Proportionalschrift ist auf dem Bildschirm (!) möglich.
Die drei eingebauten Fonts jedoch nutzen nur relativ wenige der vielfältigen Möglichkeiten, die das GEM bei der Zeichengenerierung erlauben. Alle drei Schriften sind fixe Schriften, haben also eine fest definierte Größe in Pixeln und einen definierten Abstand. Die kleinste Schrift hat eine Matrix von 6x6 Bildpunkten. Bei einer Auflösung von 640x400 Punkten könnten so 106 Zeichen in 66 Zeilen untergebracht werden. Diese Schrift ist im ST nur bei der Ausgabe unter GEM, nicht jedoch zur Ausgabe unter TOS einsetzbar und wird z.B. bei der Ausgabe des Directory in Icon-Form eingesetzt. Die nächstgrößere Schrift setzt sich aus 8x8 Punkten


HI-RES-MODUS (2)
0 1 2 639
0
1
2
399
BILDSCHIRM
FARBNR.
VIDEO-RAM

zusammen. Diese Schrift wird verwendet, wenn ein Farbmonitor am ST angeschlossen ist, während die normale Schwarz-Weiß-Darstellung mit dem dritten, größten Font vorgenommen. Dieser Font benutzt eine Matrix von 8x16 Punkten. Der genaue Aufbau des Font-Headers finden Sie beim Befehl $A008, der eine sehr vielseitige Textausgabe darstellt, die weit über das hinausgeht, was mit den Routinen des BIOS und des GEMDOS möglich ist.

Als letztes müssen noch einige im Folgenden häufig auftretende Begriffe geklärt werden, deren Bedeutung nicht so unbedingt klar sein können. Es sind dies die Bezeichnungen CONTRL-Array, INTIN-Array, INTOUT-Array, PTSIN-Array und PTSOUT-Array. Diese Arrays werden in der Hauptsache von GEM benutzt, um den einzelnen GEM-Funktionen Parameter zu übergeben oder auch Ergebnisse dieser Funktionen zu speichern. Aber auch die Funktionen der Line-A benutzen Teile dieser Arrays zur Übergabe von Parametern. Die Arrays sind im Speicher als Datenbereiche definiert, wobei jedes Element im Array aus 2 Bytes besteht.
Das CONTRL-Array enthält für GEM-Funktionen im ersten Element (CONTRL (0)) immer die Nummer der gewünschten Funktion. Dieser Parameter wird jedoch von den Line A's nicht benutzt. CONTRL (1) enthält bei jedem Aufruf des GEM die Anzahl der für die Funktion erforderlichen XY-Koordinaten. Diese Koordinaten müssen vor dem Aufruf im PTSIN-Array abgelegt werden. Das Element CONTRL (2) wird vor dem Aufruf nicht versorgt. Es enthält nach dem Aufruf die Anzahl der XY-Koordinaten im PTSOUT-Array. CONTRL (3) gibt vor dem Aufruf einer GEM-Funktion an, wieviele Parameter der Funktion im INTIN-Array übergeben werden, während CONTRL (4) die Anzahl der Parameter im INTOUT-Array nach dem Aufruf bereit hält. Die weiteren Parameter des CONTRL-Arrays sind für den Anwender der Line A nicht relevant.
Leider können nicht alle Parameter für die A-Opcodes in den zuvor beschriebenen Arrays werden. Darum gibt es noch einen Speicherbereich, der als Variablenbereich für (fast) alle Grafik-Ausgaben genutzt wird. Die Funktion und Anwendung dieser über 50 Variablen finden Sie am Schluß dieses

Kapitels komplett aufgelistet. Wichtige Variablen werden aber auch bei den Funktionen an den Stellen erläutert, an denen sie benötigt werden.
Übrigens müssen Sie damit rechnen, daß die Register D0 bis D2 und A0 bis A2 von den Aufrufen der Funktionen geändert werden. Teilweise erhalten Sie Ergebnisse in den Registern zurück. Wichtige Werte in den genannten Registern sollten also besser vor einem Aufruf gerettet werden.

$A000 Initialisierung

Initialisierung ist für diese Funktion eigentlich der falsche Ausdruck. Nach dem Aufruf erhält der Programmierer in den Registern D0 und A0 bis A2 die Adressen wichtiger Datenbereiche zurück. Irgendwelche Eingabe-Parameter benötigt diese Funktion nicht.
In D0 und A0 wird dem Programm die Startadresse der Line A-Variablen mitgeteilt. A1 zeigt nach dem Aufruf auf eine Tabelle mit drei Adressen. Diese drei Adressen sind die Start-Adressen der drei System-Font Header. Das Register A2 schließlich zeigt auf eine Tabelle mit den Start-Adressen der 15 Line A-Routinen.
Dieser Opcode zerstört (mindestens) die Inhalte der Register D0 bis D2 und A0 bis A2. Wichtige Werte sollten also besser vor dem Aufruf gerettet werden.

$A001 PUT PIXEL

Dieser Opcode setzt einen Punkt auf den durch die Koordinaten in PTSIN (0) und PTSIN (1) angegebenen Punkt. Die Farbe wird in INTIN (0) übergeben. PTSIN (0) enthält die X-Koordinate, PTSIN (1) die Y-Koordinate.
Das verwendete Koordinatensystem hat seinen Ursprung in der linken oberen Ecke. Der mögliche Bereich der X- und Y-Koordinaten richtet sich natürlich nach den Möglichkeiten des eingestellten Modus. Allerdings werden Überschreitungen im X-Bereich nicht als Fehler behandelt. Statt dessen wird

einfach bei einer Überschreitung die Y-Koordinate um den entsprechenden Betrag erhöht. Bei Überschreitungen des Y-Bereichs erfolgt keine Ausgabe.
Die Farbe in INTIN (0) ist vom verwendeten Modus abhängig. Beim Betrieb des Monochrom-Monitors wird nur das Bit Null des Werts in INTIN (0) ausgewertet.

$A002 GET PIXEL

Mit diesem Opcode kann die Farbe eines Pixels bestimmt werden. Wie bei $A001 wird die XY-Koordinate in PTSIN (0) und PTSIN (1) angegeben, den Farbwert des Pixels erhält man im Register D0 zurück.

$A003 LINE

Mit dem Line-Opcode kann eine Linie zwischen den Punkten mit den Koordinaten x1,y1 und x2,y2 gezogen werden. Die Parameter werden bei dieser Funktion nicht über die Parameter-Arrays übergeben, sondern müssen vor dem Aufruf in die Line A-Variablen übertragen werden. Die benutzen Variablen sind:

```
_X1      = x1-Koordinate
_Y1      = y1-Koordinate
_X2      = x2-Koordinate
_Y2      = y2-Koordinate
_FG_BP_1 = Plane 1 (alle drei Modi)
_FG_BP_2 = Plane 2 (640x200, 320x200)
_FG_BP_3 = Plane 3 (nur 320x200)
_FG_BP_4 = Plane 4 (nur 320x200)
_LN_MASK = Bitmuster der Linie
           z.B. $FFFF = durchgezogen
                $CCCC = durchbrochen
_WRT_MOD = bestimmt den Schreibmodus.
_LSTLIN  = diese Variable sollte auf -1 ($FFFF)
           gesetzt werden.
```

Ein Punkt, den es bei manchen Anwendungen zu beachten gilt, ist die Tatsache, daß beim Ziehen der Linie das höchste Bit des Linienbitmusters immer an den linken Bildschirmrand gesetzt wird. Die Linie wird also immer von links nach rechts und von oben nach unten, nicht jedoch von x1,y1 nach x2,y2 gezogen.
Bereichsüberschreitungen werden wie bei PUT PIXEL behandelt. Soll also eine Line von 0,0 nach 650,50 gezogen werden, so wird tatsächlich eine Linie von 0,0 bis 639,48 gezogen. Der 'Rest' ergibt eine zusätzliche Linie von 0,49 bis 10,50.
Für das Linienziehen stehen insgesamt 4 verschiedene Schreibmodi mit den Werten 0 bis 3 zur Verfügung. Beim Schreibmodus Null wird das ursprünglich vorhandene Bitmuster 'unter' der Linie gelöscht und das durch _LN_MASK bestimmte Bitmuster eingetragen (Replace Mode). Im Transparent Mode (_WRT_MOD = 1) werden Hintergrund, also altes Bitmuster und neues Linienmuster verodert, also nur zusätzlich Punkte gesetzt. Im XOR Mode (_WRT_MOD = 2) werden Hintergrund und Linienmuster Exclusiv-verODERt. Der letzte Modus (_WRT_MOD =3) schließlich ist der sogenannte 'Inverse Transparent Mode'. Wie auch beim Transparent Mode handelt es sich um eine Oderverknüpfung von Vordergrund und Hintergrunddaten, wobei jedoch die Vordergrunddaten, also das durch _LN_MASK festgelegte Bitmuster vor der Veroderung invertiert wird.

$A004 HORIZONTAL LINE

Diese Funktion zieht eine Linie von x1,y1 nach x2,y1. Das Ziehen einer horizontae Linie ist bedeutend schneller, als wenn eine Linie diagonal gezeichnet werden muß. Im Grunde werden auch diagonale Linien über diese Funktion erzeugt, indem die Linie in mehrere horizontale Einzelstücke zerlegt wird. Die Parameter werden direkt in die benötigten Variablen eingetragen.

```
_X1     = x1-Koordinate
_Y1     = y1-Koordinate
_X2     = x2-Koordinate
```

```
_FG_BP_1 = Plane 1 (alle drei Modi)
_FG_BP_2 = Plane 2 (640x200, 320x200)
_FG_BP_3 = Plane 3 (nur 320x200)
_FG_BP_4 = Plane 4 (nur 320x200)
_WRT_MOD = bestimmt den Schreibmodus.
_patptr  = Pointer auf das zu verwendende Linienmuster
_patmsk  = 'Maske' für das Linienmuster
```

Die in _WRT_MOD zulässigen Werte liegen auch bei diesem Aufruf zwischen 0 und 3. Der Inhalt der Variable _patptr ist die Adresse, an der das gewünschte Linienmuster oder besser Füllmuster zu finden ist. Die H-Line-Funktion eignet sich nämlich hervorragend zum Erzeugen beliebig gefüllter Flächen. Eine wesentliche Rolle spielt dabei die Variable _patmsk. Von ihrem Wert ist die Anzahl der 16-Bit-Werte an der Adresse in _patptr abhängig. Enthält z.B. _patmsk den Wert 5, so sollten ab der Adresse in _patptr sechs 16-Bit-Werte als Linienmuster im Speicher stehen. Soll nun eine horizontale Linie mit der Y-Koordinate Null gezogen werden, so wird das erste Bit-Muster als Linienmuster genommen. Für eine Linie mit der Y-Koordinate 1 wird das zweite Wort als Muster genommen u.s.w. Das Muster für eine Linie mit der Y-Koordinate 6 wird wieder vom ersten Wert in der Bit-Tabelle bestimmt. Auf diese Weise können sehr komplexe Füllmuster ohne großen Aufwand erzeugt werden.

$A005 FILLED RECTANGLE

Der Opcode $A005 stellt eine Erweiterung oder besser spezielle Nutzung des Opcodes $A004 dar. Mit dieser Funktion können beliebig gefüllte Rechtecke erzeugt werden. Als wesentliche Parameter werden die linke obere und die rechte untere Ecke als Koordinaten angegeben.

```
_X1      = x1-Koordinate, links oben
_Y1      = y1-Koordinate
_X2      = x2-Koordinate, rechts unten
_Y2      = y2-Koordinate
```

```
_FG_BP_1  = Plane 1 (alle drei Modi)
_FG_BP_2  = Plane 2 (640x200, 320x200)
_FG_BP_3  = Plane 3 (nur 320x200)
_FG_BP_4  = Plane 4 (nur 320x200)
_WRT_MOD  = bestimmt den Schreibmodus.
_patptr   = Pointer auf das zu verwendende Füllmuster
_patmsk   = 'Maske' für das Füllmuster
_CLIP     = Clipping Flag
_XMN_CLIP = x Minimum für Clipping
_XMX_CLIP = x Maximum für Clipping
_YMN_CLIP = y Minimum für Clipping
_YMX_CLIP = y Maximum für Clipping
```

Bis auf die 'Clip'-Variablen sind die verwendeten Variablen bereits bei den vorigen Aufrufen erläutert. Was aber hat es mit dem Clipping auf sich? Nun, Clippings sind Ausschnitte. Das Clipping erzeugt also Ausschnitte aus dem gesamten Bild. Wird das Clipping Flag auf Eins (oder jeden Wert ungleich Null) gesetzt, dann wird das Rechteck, das mit $A005 gezeichnet wird, nur in dem durch die Clip-Bereichsvariablen angegebenen Ausschnitt dargestellt. Ein Beispiel mag dieses Verhalten ein wenig erläutern:
Es werden als Koordinaten die Werte 100,100 und 200,200 angegeben. Das Clip-Flag sei 1 und die Clip-Bereichsvariablen enthalten einmal die Werte 150,150 für XMN_CLIP und YMN_CLIP sowie 300,300 in XMX_CLIP und YMX_CLIP. Als Füllmuster soll nur einfach der Wert $FFFF für alle Linen gelten. Mit diesen Werten wird das Rechteck die Koordinaten 150,150 als linke obere Ecke und 200,200 als rechte untere Ecke haben. Der 'fehlende' Bereich ist durch die Clip-Angaben nicht gezeichnet worden. Ein Löschen des Clip-Flags jedoch zeichent das Rechteck in der ursprünglich gewünschten Größe.

$A006 FILLED POLYGON

Auch $A006 ist eine Erweiterung des Aufrufs $A004. Mit diesem Opcode können beliebig geformte Flächen mit einem Muster gefüllt werden. Allerdings wird mit dem dem Aufruf

nicht die gesamte Fäche gefüllt, sondern nur eine Rasterzeile, also eine einen Punkt breite horizontale Linie gezeichnet. Dadurch ergeben sich natürlich wesentlich größere Beeinflussungsmöglichkeiten auf das Füllmuster.
Die benötigten Variablen sind:

```
PTSIN      = Array mit den XY-Koordinaten
CONTRL (1) = Anzahl der Koordinatenpaare
_Y1        = Y-Koordinate der zu füllenden Zeile
_FG_BP_1   = Bitwert für Plane 1
_FG_BP_2   = Bitwert für Plane 2
_FG_BP_3   = Bitwert für Plane 3
_FG_BP_4   = Bitwert für Plane 4
_WRT_MOD   = Schreibmodus (0-3)
_patptr    = Pointer auf die Füllmuster
_patmsk    = Anzahl der Füllmuster-Worte
_CLIP      = Clipping Flag
_XMN_CLIP  = x Minimum für Clipping
_XMX_CLIP  = x Maximum für Clipping
_YMN_CLIP  = y Minimum für Clipping
_YMX_CLIP  = y Maximum für Clipping
```

Im Grunde haben Sie hier alle Parameter genau so zu setzen, wie dies auch für einen Aufruf von $A005 erforderlich ist. Nur die drei ersten Koordinaten sind unterschiedlich. Im PTSIN-Array werden die XY-Koordinaten gespeichert. Wichtig ist, daß Sie die Startkoordinaten auch noch einmal als letzte Koordinaten angeben. Um also ein Dreieck zu definieren, müssen Sie z.B. die Koordinaten (320,100), (120,300), (520,300) und (320,100) eingeben. Die Anzahl der effektive Koordinatenpaare, in unserem Beispiel also drei, müssen Sie in CONTRL (1), dem zweiten Element des Arrays hinterlegen. Mit einem Aufruf der $A006-Funktion müssen Sie noch die Y-Koordinate der zu ziehenden Linie angeben. Natürlich können Sie nacheinander alle Y-Koordinaten von 0 bis 399 (0 bis 199 in den Frabmodi) füllen lassen. Schneller ist es jedoch, sich aus den XY-Werten den kleinsten und größten verwendeten Y-Wert zu suchen, und nur für diesen Bereich die Funktion aufzurufen.

$A007 BITBLT

Der Bit Block Transfer wird vom Text Block Transfer, $A008, und von Copy Raster Form, $A00E, genutzt. Als Parameter muß im Register A6 ein Pointer auf eine Parameter-Tabelle übergeben werden. Leider konnte der Aufbau dieser Parameter-Tabelle nicht definitiv erforscht werden. Unsere Versuche ergaben zu etwa 70% klassische Systemabstürze. Aus diesem Grund kann zu dieser Funktion leider nicht viel gesagt werden.

$A008 TEXTBLT

Mit der Text BLock Transfer-Funktion kann ein beliebiges Zeichen aus einem beliebigen Textfont an jeder Grafik-Position ausgegeben werden. Zusätzlich kann das Zeichen in seiner Form verändert werden. So kann das Zeichen z.B. kursiv, fett, 'outlined', vergrößert oder gedreht dargestellt werden. Diese Dinge lassen mit den 'normalen' Zeichenausgaben über das Bios oder GEMDOS nicht erzielen. Dafür müssen aber auch eine Vielzahl von Parametern gesetzt und kontroliert werden. Es muß schon ein etwas umfangreicheres Programm geschrieben werden, um Texte mit dieser Funktion auszugeben. Wenn die zusätzlichen Möglichkeiten nicht unbedingt erforderlich sind, kann eigentlich nur von der Benutzung dieser Funktion abgeraten werden. Aber entscheiden Sie doch bitte selbst.
Bevor wir jedoch konkret werden und ein Zeichen auf den Bildschirm zaubern, müssen wir uns zunächst ein wenig mit der Organisation der Fonts beschäftigen. Besonders der Fontheader muß unter die Lupe genommen werden, da mit den darin enthaltenen Daten der Font in allen Einzelheiten beschrieben wird.

Grundsätzlich besteht ein Font aus den vier Datenbereichen Fontheader, Fontdaten, Character Offset Tabelle und

Horizontal Offset Tabelle. Im Fontheader sind allgemeine Daten über den Font wie z.B. der Name des Fonts, seine Größe, die Anzahl der enthaltenen Zeichen und verschiedene andere Aspekte enthalten. Diese Information belegt insgesamt 88 Bytes. Die Fontdaten enthalten die Bitmuster der vorhandenen darstellbaren Zeichen. Diese Daten sind äußerst platzsparend organisiert.
Um die Organisation besser beschreiben zu können, denken wir uns einfach einen Font mit nur zwei Zeichen, z.B. einem 'A' und einem 'B'. Diese Zeichen sollen in einer 9x9-Matrix dargestellt werden. Die Fontdaten stehen jetzt so im Speicher, daß das Bitmuster der obersten Scanline des 'A' an einer Wort-Grenze beginnend gespeichert ist.
Da unser Font 9 Pixel = 9 Bit breit ist, wird ein Byte vollständig, das folgende Byte aber nur mit dem obersten Bit benutzt. Jetzt müßten eigentlich 7 Bits verschenkt werden, wenn die oberste Scanline des 'B' auch wieder auf einer Wort-Grenze beginnen soll. Dem ist aber nicht so, statt dessen beginnt die erste Scanline des 'B' im Bit 6 des zweiten Bytes der Fontdaten. Nur die Daten der zweiten und weiteren Scanlines des 'A' beginnen immer auf einer Wort-Grenze. Auf diese Weise wird im Font fast kein Bit verschenkt. Nur die Anfänge der Scanlines des ersten Zeichens beginnt tatsächlich auf einer Wort-Grenze, alle anderen Scanlines können auf jeder Bit-Position beginnen.
Durch diese platzsparende Speicherung muß die Position jedes Zeichens innerhalb des Font berechnet werden. Die Berechnung der Scanline-Positionen wird möglich durch die Character Offset Tabelle. In dieser Tabelle ist für jedes darstellbare Zeichen ein Eintrag enthalten, der die Bitposition der ersten Scanline des Zeichens relativ zum Anfang der Fontdaten enthält. Für unser Beispiel enthält eine solche Tabelle die Einträge $0000, $0009 und $0012. Durch die Einführung dieser Tabelle ist es möglich, echte Proportionalschrift auf dem Bildschirm zu erzeugen, da die Breite des auszugebenden Zeichens errechnet werden kann. Man subtrahiert dazu den Eintrag des auszugebenden Zeichens vom Eintrag des nächsten Zeichens. Um auch die Breite des letzten Zeichens bestimmen zu können, ist der letzte Eintrag

vorhanden, den aber kein Zeichen mehr zugeordnet ist.
Neben der Character Offfset Tabelle gibt es noch die Horizontal Offset Tabelle. Diese Tabelle wird jedoch von den meisten Fonts nicht genutzt. Auch die im ST vorhandenen Fonts nutzen die Möglichkeiten dieser Tabelle nicht. Wäre die Tabelle vorhanden, so enthielte sie für jedes Zeichen einen positiven oder negativen Offsetwert, um den das Zeichen bei der Ausgabe nach links oder rechts verschoben würde.

Zum Ende der Beschreibung des Fontaufbaus folgt die Bedeutungen der Variablen im Fontheader.

```
Byte 0-1   : Font Identifier. Eine Nummer, die den Font beschreibt.
             1 = Systemfont.
Byte 2-3   : Fontgröße in Punkt (Punkt ist ein Maß aus der Satztechnik).
Byte 4-35  : Der Name des Fonts als ASCII-Zeichenkette.
Byte 36-37 : Der niedrigste ASCII-Wert der darstellbaren Zeichen.
Byte 38-39 : Der höchste ASCII-Wert der darstellbaren Zeichen.
Byte 40-49 : relative Abstände von Top, Ascent, Half, Decent und Bottom Line
             zur Base Line.
Byte 50-51 : Breite des breitesten Zeichens im Font.
Byte 52-53 : Breite der breitesten Character-Zelle. Die Zelle ist immer
             mindestens ein Pixel breiter, als das eigentliche Zeichen,
             damit zwei nebeneinander stehende Zeichen voneinander getrennt
             sind.
Byte 54-55 : Linker Offset.
Byte 56-57 : Rechter Offset. Die beiden Offset-Werte werden nur beim
             Kursiv-Stellen von Zeichen (Skewing) verwendet.
Byte 58-59 : Thickening. Soll ein Zeichen fetter dargestellt werden, so
             sollte der Wert in dieser Variablen Benutzt werden.
Byte 60-61 : Underline. Enthält die Höhe des Unterstreichungstrichs in Pixeln.
Byte 62-63 : Lightening Mask. 'Light' dargestellte Zeichen finden Sie
             im Desktop, wenn in einem Pull-Down-Menu eine Option nicht
             verfügbar ist. Diese hellgrau erscheinenden Zeichen entstehen
             durch Ausmaskieren von Bits mit der Lightening Mask. Üblich
             ist der Wert $5555.
Byte 64-65 : Skewing Mask. Wie zuvor, nur für die Kursiv-Stellung von Zeichen.
```

Byte 66-67 : Flag. Bit 0 wird gesetzt, wenn der Font der System-Font ist.
Bit 1 muß gesetzt werden, wenn die Horizontal Offset Tabelle vorhanden ist.
Bit 2 ist das sogenannte Byte-Swap-Flag. Ist das gesetzt, so stehen die Bytes im 68000-Format (Lowbyte-Highbyte) im Speicher. Ein gelöschtes Byte-Swap-Flag signalisiert, das die Daten im INTEL-Format, also umgekehrt im Speicher stehen. Mit diesem Bit können Fonts vom IBM mit GEM auch auf dem ST und umgekehrt verwendet werden.
Bit 3 ist gesetzt, wenn die Breite aller Zeichen im Font gleich ist.

Byte 68-71 : Pointer auf die Horizontal Offset Tabelle oder Null.

Byte 72-75 : Pointer auf die Character Offset Tabelle.

Byte 76-79 : Pointer auf die Font-Daten.

Byte 80-81 : Form Width. In dieser Variable ist die Summe der Breiten aller Zeichen enthalten. Der Wert gibt somit die Länge der Scanlines aller Zeichen und damit den Anfang der nächsten Line an.

Byte 82-83 : Form Height. In dieser Variable ist die Anzahl der Scanlines dieses Fonts enthalten.

Byte 84-87 : Enthält einen Pointer auf den nächsten Font.

Nach so viel Vorrede sollen jetzt die Parameter genannt werden, die beim $A008-Opcode zu beachten bzw. zu versorgen sind.

```
_WRT_MODE    = Schreibmodus
_TEXT_FG     = Text Vordergrund-Farbe
_TEXT_BG     = Text Hintergrund-Farbe
_FBASE       = Pointer auf den Anfang der Font-Daten
_FWIDTH      = Breite des Fonts
_SOURCEX     = X-Koordinate des Zeichens im Font
_SOURCEY     = Y-Koordinate des Zeichens im Font
_DESTX       = X-Koordinate des Zeichens auf dem Schirm
_DESTY       = Y-Koordinate des Zeichens auf dem Schirm
_DELX        = Breite des Zeichens in Pixeln
_DELY        = Höhe des Zeichens in Pixeln
_STYLE       = Bitweise codierte Flags für spez. Effekte
```

```
_LITEMASK     = Bitmuster wird beim 'lightning' verwendet
_SKEWMASK     = Bitmuster wird beim 'skewing' verwendet
_WEIGHT       = Faktor bei Zeichenverbreiterung
_R_OFF         = rechter Offset des Zeichens beim
'Skewing'
_L_OFF        = linker Offset des Zeichens beim 'Skewing'
_SCALE              =       Flag       für        Scaling
(Vergrößerung/Verkleinerung)
_XACC_DDA     = Akkumulator für Scaling
_DDA_INC       =   Faktor    der    Vergrößerung    oder
Verkleinerung
_T_SCLSTS     = 'Scaling' Richtungsflag
_CHUP         = Zeichenrotationsvector
_MONO_STATUS = Flag fixe Schrift (Monospaced)
_scrtchp      = Pointer auf Arbeitsbuffer für Effekte
_scrpt2       = Offset Scaling-Buffer in _scrtchp
```

Weiterhin werden die fünf Clip-Variablen ausgewertet.

Wie man sieht, sind es eine enorme Vielzahl von Variablen, die bei der Ausgabe von Grafiktext ausgewertet werden. Hier können wir nur auf die wesentlichen (und von uns erforschten) Variablen eingehen. Die nicht besprochenen Variablen mögen Ihnen als fruchtbares Experimentierfeld dienen.

Der Schreibmodus erlaubt die Ausgabe von Zeichen in den bisher bekannten vier Modi Replace, Or, Xor und Inverse Or. Zusätzlich sind 16 weitere Modi vorhanden, deren Auswirkungen jedoch nicht erforscht wurden. In Zusammenhang mit den ersten vier Schreibarten wird die Variable _TEXT_FG benutzt. Sie bildet die zum Darstellen verwendete Vordergrundfarbe. Die Hintergrundfarbe _TEXT_BG spielt nur bei den 16 zusätzlichen Modi eine Rolle. Somit ist klar, daß die zusätzlichen Modi nur in Verbindung mit einem Farbbildschirm relevant sind.
Die Variablen _FBASE und _FWIDTH werden entsprechend dem gewünschten Font gesetzt. Den Anfang der Fontdaten können Sie dem Header des gewünschten Fonts entnehmen (Byte 76-79

im Header). _FWIDTH muß mit dem Inhalt der Bytes 80 und 81 des Header beschrieben werden.
_SOURCEX bestimmt, welches Zeichen Sie ausgeben. Hier ist der ASCII-Wert des gewünschten Zeichens einzutragen. _SOURCEY ist üblicherweise Null, da ja das Zeichen von der obersten bis zu untersten Scanline abgebildet werden soll.
_DELX als Breite des Zeichens kann berechnet werden, indem der Eintrag der Character Offset Table des gewünschten Zeichens vom nächsten Eintrag abgezogen wird. Das Ergebnis ist die Breite des Zeichens in Pixeln. _DELY muß mit dem Wert der Bytes 82-83 des Headers geladen werden.
Etwas besonderes hat es mit der _STYLE-Variable auf sich. Hier kann angegeben werden, ob Zeichen original oder verändert dargestellt werden. Die möglichen Veränderungen sind hervorheben (thicken, Bit 0), schattieren (lighten, Bit 1), kursivstellen (Bit 2), und Outlined (Bit 4). Die jeweilige Veränderung wird durch Setzen der Bits eingeschaltet.
Eine weitere Veränderung ist das Scaling. Mit Scaling kann ein Zeichen in seiner Größe verändert werden. Leider klappt beim ST scheinbar nur das Vergrößern der Zeichen, Verkleinerungen konnten nicht erzielt werden.
Ist das Scaling-Flag gelöscht (Null), so wird das Zeichen in seiner Originalgröße dargestellt. Ob vergrößert oder verkleinert werden soll, bestimmt das _T_SCLSTS-Flag. Hier muß für Vergrößerung ein Wert ungleich Null eingetragen werden. _DDA_INC soll den Wert der Vergrößerung oder Verkleinerung enthalten. Allerdings konnte nur bei einem Wert von $FFFF eine Vergrößerung festgestellt werden.
Eine weitere interessante Variable ist _CHUP. Mit Hilfe dieser Variable können die Zeichen auf dem Bildschirm gedreht werden. Dazu muß der Winkel im Bereich von 0 bis 360 Grad in 1/10 Grad angegeben werden. Aber auch bei dieser Funktion muß eine Einschränkung gemacht werden. Brauchbare Ergebnisse werden nur bei Drehungen um jeweils 90 Grad gemacht. Die anzugebenden Werte sind also $0000 für Normal-Lage, $0384 für 90 Grad Drehung, $0704 (Schrift steht auf dem Kopf) und $0A8C für 270 Grad.
Soll mit den Effekten gearbeitet werden, so muß in _scrchp

ein Pointer auf einen Buffer übergeben werden, in dem TEXTBLT Zwischenwerte speichern kann. Die genaue Größe dieses Buffers ist leider nicht bekannt, allerdings hat sich ein Buffer von 1K immer als ausreichend erwiesen. Bei Vergrößerung muß ein weiterer Buffer (_scrpt2) angegeben werden. Als Parameter wird ein Offset angegeben, der sich auf den Anfang des _scrtchp-Buffers bezieht. Ein Wert von $40 hat sich hier als ausreichend erwiesen.

$A009 SHOW MOUSE

Ein Aufruf dieses Opcodes schaltet die Darstellung des Maus-Cursors ein. Der Cursor folgt der Maus, wenn diese bewegt wird. Damit kann die Maus auch in Programmen genutzt werden, wenn auf die Benutzeroberfläche GEM verzichtet wird. Besonders für Spiele ist diese Möglichkeit reizvoll.
Benötigte Parameter werden im INTIN- und CONTRL-Array übergeben. CONTRL (1) sollte vor dem Aufruf gelöscht werden und CONTRL (3) wird auf Eins gesetzt. INTIN (0) hat eine spezielle Bedeutung. Die Routine zur Verwaltung des Maus-Cursors zählt die Anzahl der Aufrufe zum Entfernen und Einschalten des Cursors. Wurde zwei mal der Cursor ausgeschaltet, so muß auch zweimal der Aufruf zum Einschalten erfolgen, bevor er tatsächlich auf dem Bildschirm erscheint. Dieses Verhalten läßt sich aber beeinflussen, indem in INTIN (0) gelöscht wird. Mit diesem Parameter wird der Cursor unabhängig von der Anzahl vorangegangener HIDE CURSOR-Aufrufe sofort gesetzt. Ist der Wert in INTIN (0) jedoch ungleich Null, so muß die tatsächlich benötigte Anzahl von $A009-Aufrufen erfolgen, um den Cursor sichtbar zu machen.

$A00A HIDE CURSOR

Diese Funktion stellt die schon erwähnte Möglichkeit zum Entfernen des Cursors dar. Wird diese Funktion wiederholt aufgerufen, so wird die Anzahl vom Betriebsystem mitgeführt

und bestimmt die Anzahl der benötigten Aufrufe SHOW CURSOR, bis der Cursor tatsächlich erscheint.

$A00B TRANSFORM MOUSE

Ist der Pfeil als Mauscursor für Spiele ungeeignet? Machen Sie sich doch einfach ihren eigenen Cursor. Wie wär's mit einem kleinen Auto, das an Stelle des Pfeils über den Bildschirm flitzt.? Mit dem Opcode $A00B wird Ihrer Fantasie, jedenfalls was die Cursor-Form angeht, freien Lauf gelassen.
Die Parameter müssen im INTIN-Array übergeben werden. Insgesamt werden 34 Worte benötigt. Die folgende Tabelle gibt Aufschluß über Verwendung und mögliche Werte:

INTIN (3) Maskenfarbindex, normalerweise 0
INTIN (4) Datenfarbindex, normalerweise 1
INTIN (5) bis (20) enthalten 16 Worte Cursormaske
INTIN (21) bis (36) enthalten 16 Worte Cursordaten

Die Form des Cursors wird bestimmt durch die Cursordaten. Jede 1 in den Daten erzeugt einen Punkt auf dem Bildschirm. Wenn jedoch der Cursor über einen Buchstaben oder ein Muster gelegt wird, so kann die Grenze zwischen Hintergrund und Cursor nicht ausgemacht werden. An diesem Punkt tritt die Maske in Aktion. Jedes in der Maske gesetzte Bit löscht den Hintergrund an der angegebenen Stelle. Auf diese Weise läßt sich ein heller Rahmen um die Cursor-Form ziehen. Betrachten Sie einmal den normalen Pfeil-Cursor, wenn er einen Buchstaben oder ein Muster überdeckt, um die Auswirkungen der Maske zu sehen.

$A00C UNDRAW SPRITE

Dieser Opcode ist in Verbindung mit $A00D, Draw Sprite, zu sehen. Tatsächlich kennt der ST keine Hardware-Sprites in der Form, wie sie z.B. vom COMMODORE C64 bekannt sind. Die

Sprites des ST werden rein softwaremäßig organisiert. Jeder Sprite ist 16x16 Pixel groß. Ein Beispiel für ein ST-Sprite ist z.B. der Maus-Cursor. Er wird über diese Funktion erzeugt.
Um ein zuvor gezeichnetes Sprite zu löschen, muß im Register A2 die Adresse eines Buffers angegeben werden, in dem beim Zeichnen des Sprite der Hintergrund gerettet wird. Der Opcode überträgt nur den Inhalt des Hintergrund-Buffers an die richtige Stelle des Bildschirms. Der Buffer selbst muß für jede Plane 64 Bytes groß sein. Zusätzlich werden 10 Bytes unabhängig von der Anzahl der Planes benutzt. Bei monochromer Darstellung ist der Buffer somit 74 Bytes, in der 320x200 Pixel-Auflösung (vier Planes). 4x64+10=266 Bytes groß.

$A00D DRAW SPRITE

Mit dieser Funktion wird das gewünschte Sprite auf den Bildschirm gezeichnet. Dazu müssen Parameter in den Registern D0, D1, A0 und A2 übergeben werden.
D0 und D1 enthalten die X- und Y-Koordinate der gewünschten Position des Sprite auf dem Bildschirm, kurz Hot Spot genannt. A0 ist ein Pointer auf den sogenannten Sprite Definition Block und A2 enthält die Adresse des Sprite Save Buffers, in dem der Hintergrund für das Löschen des Sprite übertragen wird.
Der Sprite Definition Block muß folgenden Aufbau haben:

```
Wort 1 : X-Offset zum Hot-Spot
Wort 2 : Y-Offset zum Hot-Spot
Wort 3 : Format-Flag 0=VDI-Format, 1=XOR-Format
Wort 4 : Hintergrund-Farbe (bg, Background)
Wort 5 : Vordergrund-Farbe (fg, Foreground)
```

Daran schließen sich 32 Worte an, die das Muster des Sprite enthalten. Dabei muß das Muster in der folgenden Reihenfolge im Speicher stehen:

```
Wort 6 : Hintergrund-Muster oberste Zeile
Wort 7 : Vordergrund-Muster oberste Zeile
Wort 8 : Hintergrund-Muster zweite Zeile
Wort 9 : Vordergrund-Muster zweite Zeile
u.s.w.
```

Die Information im Format-Flag hat die folgende Bedeutung:

VDI-Format

fg	bg	Ergebnis
0	0	Der Hintergrund erscheint
0	1	Die in Wort 4 hinterlegte Farbe erscheint
1	0	Die in Wort 5 hinterlegte Farbe erscheint
1	1	Die in Wort 5 hinterlegte Farbe erscheint

XOR-Format

fg	bg	Ergebnis
0	0	Der Hintergrund erscheint
0	1	Die in Wort 4 hinterlegte Farbe erscheint
1	0	Pixel auf dem Screen wird mit fb-Bit XORed
1	1	Die in Wort 5 hinterlegte Farbe erscheint

$A00E COPY RASTER FORM

Mit dem $A00E-Opcode können beliebige Bereiche des Bildschirms copiert werden. Dabei kann nicht nur innerhalb des Bildschirms copiert werden, sondern auch aus dem Bildschirm in das freie RAM und was noch wichtiger ist, aus dem RAM in den Bildschirm copiert werden. Aber auch komplette Bildschirmseiten lassen sich mit dem Copy-Raster-Opcode sehr schnell übertragen. Der Name Raster Form drückt jedoch auch eine Einschränkung der Fuktion aus. Jede zu übertragende Rasterform muß auf einer Wortgrenze beginnen und aus einem vielfachen von Worten bestehen.
Die Parameter sind recht zahlreich und werden in den CONTRL-, PTSIN- und INTIN-Arrays übergeben. Zusätzlich müssen zum Copy Raster zwei 'Memory Form Definition'-Blöcke im Speicher stehen. Beginnen wir mit den MFD-Blöcken. Da ein

Kopiervorgang immer eine Quelle und ein Ziel hat, beschreibt der eine Block den Quell-Speicherbereich, der zweite beschreibt das Ziel. Jeder Block besteht aus 10 Worten. In den ersten beiden Worten ist die Adresse des Speichers enthalten, der durch den Block beschrieben wird. Das dritte Wort gibt die Breite der Form in Pixeln an. Wort 4 bestimmt die Höhe der Form in Pixeln. Das fünfte Wort beschreibt die Breite der Form in Worten. Wort 6 sollte auf 1 gesetzt werden und Wort 7 gibt an, aus wievielen Planes die Form besteht. Die restlichen Worte sollten auf Null gesetzt werden, da sie für spätere Erweiterungen reserviert sind.

Die 'Line A'-Variablen im Überblick

Nach der Initialisierung $A000 erhält man in D0 und A0 die Adresse eines Variablenbereichs zurück, in dem über 50 Line A-Variablen enthalten sind. Wesentliche Variablen wurden bereits bei den verschiedenen Aufrufen erklärt, nicht jedoch die Lage der Variablen innerhalb des Variablenblocks. Diese Aufstellung soll jetzt erfolgen. Bei der Benennung der Variablen haben wir uns an die Namen gehalten, die in der offiziellen ATARI-Dokumentation genannt werden.
Offset ist der Wert, der angegeben werden muß, wenn auf die Variable Registerrelativ zugegriffen wird.
Mit einem Fragezeichen versehene Variablen konnten nicht definitiv geklärt werde.

Offs.	Name	Größe	Funktion
0	v_planes	Word	Anzahl der Planes
2	v_lin_wr	Word	Bytes pro Scanline
4	CONTRL	Long	Pointer auf das CONTRL-Array
8	INTIN	Long	Pointer auf das INTIN-Array
12	PTSIN	Long	Pointer auf das PTSIN-Array
16	INTOUT	Long	Pointer auf das INTOUT-Array
20	PTSOUT	Long	Pointer auf das PTSOUT-Array
24	_FG_BP_1	Word	Plane 0 Farbwert
26	_FG_BP_2	Word	Plane 1 Farbwert

28	_FG_BP_3	Word	Plane 2 Farbwert
30	_FG_BP_4	Word	Plane 3 Farbwert
32	_LSTLIN	Word	sollte -1 ($FFFF) sein (?)
34	_LN_MASK	Word	bei $A003 Linienmuster
36	_WRT_MODE	Word	Schreibmodus (0=Schreibmodus 1=Transparent 2=Xor-Mode 3=Invers-Transparent Mode)
38	_X1	Word	X1-Koordinate
40	_Y1	Word	Y1-Koordinate
42	_X2	Word	X2-Koordinate
44	_Y2	Word	Y2-Koordinate
46	_patptr	Long	Pointer auf das Füllmuster (siehe $A004)
50	_patmsk	Word	Füllmuster-'Maske' (siehe $A004)
52	_multifill	Word	0=Füllmuster ist nur für eine Plane 1=Füllmuster ist für Multi-Plane ausgelegt
54	_CLIP	Word	0=kein Clipping eingeschaltet (siehe $A005) ungleich 0=Clipping
56	_XMN_CLIP	Word	und
58	_YMN_CLIP	Word	definieren linke obere Ecke des sichtbaren Bereichs bei Clipping
60	_XMX_CLIP	Word	und
62	_YMX_CLIP	Word	definieren rechte untere Ecke des sichtbaren Bereichs bei Clipping
64	_XACC_DDA	Word	sollte vor jedem Aufruf von TXTBLT auf $8000 gesetzt werden (?)
66	_DDA_INC	Word	Vergrößerungs/Verkleinerungsfaktor $FFFF für Vergrößerung, Verkleinerung funktioniert nicht (?)
68	_T_SCLSTS	Word	0=Verkleinerung (?) 1=Vergrößerung
70	_MONO_STATUS	Word	1=kein Proportionalfont 0=Proportionalschrift oder Breite der Zeichen ändert sich bei Fettschrift oder kursivstellen
72	_SOURCEX	Word	X-Koordinate des Zeichens im Font
74	_SOURCEY	Word	Y-Koordinate des Zeichens im Font (0)

Bemerkung:

SOURCEX ist der Wert des Zeichens aus der Horizontal Offset Tabelle (HOT) und läßt sich nach folgender Formel berechnen:
SOURCEX = HOT-Element (ASCII-Wert minus FIRST ADE)
Die Variable FIRST ADE ist in den Bytes 36,37 des Font Header enthalten (siehe Beispiel)

76	_DESTX	Word	X-Position Zeichen auf den Bildschirm
78	_DESTY	Word	Y-Position Zeichen auf den Bildschirm
80	_DELX	Word	Breite des Zeichens
82	_DELY	Word	Höhe des Zeichens

Bemerkung:
DELX kann berechnet werden nach folgender Formel:
DELX = SOURCEX+1 minus SOURCEX
(siehe $A008)
DELY ist der Wert FORM Height aus den Bytes 82,83 des Font Header

84	_FBASE	Long	Pointer auf Start der Fontdaten
88	_FWIDTH	Word	Breite der Fontform
90	_STYLE	Word	Flags spezielle Effekte (siehe $A008)
92	_LITEMASK	Word	Maske für das Schattieren
94	_SKEWMASK	Word	Maske für Kursiv-Schrift
96	_WEIGHT	Word	Anzahl der Bits, um daß das Zeichen verbreitert werden soll
98	_R_OFF	Word	Offset für Kursiv-Schrift
100	_L_OFF	Word	Offset für Kursiv-Schrift

Bemerkung:
Die vorgenannten fünf Variablen sollten mit den entsprechenden Werten aus dem Font-Header versorgt werden.

102	_SCALE	Word	0=kein Scaling 1=Scaling (vergrößern/verkleinern)
104	_CHUP	Word	Winkel für Zeichenrotation 0=normale Zeichendarstellung $384=90 Grad gedreht $708=180 Grad gedreht

			$A8C=270 Grad gedreht
106	_TEXT_FG	Word	Vordergrundfarbe für Textdarstellung
108	_scrtchp	Long	Adresse des für Erzeugung der spez. Text-Effekte benötigten Buffers
112	_scrpt2	Word	Offset des Vergrößerungsbuffers im _scrtchp-Buffer ($40)
114	_TEXT_BG	Word	Hintergrundfarbe für Textdarstellung
116	_COPYTRAN	Word	(?)

Beispiele für die Nutzung der Line A-Opcodes

Um Ihnen die ersten Versuche mit den Line A-Opcodes ein wenig zu erleichtern, sollen jetzt noch ein paar Beispiele folgen, die Ihnen als Starthife dienen können. Im ersten Beispiel wird mit $A001 ein Punkt auf dem Bildschirm gesetzt und anschließend die Farbe des Punktes mit $A002 festgestellt.

```
*************************************************************************
*
*
*               Demo der Funktionen $A000,$A001 und $A002
*
*               rbr     28.09.85
*
*************************************************************************



intin           equ     8
ptsin           equ     12

init            equ     $a000
setpix          equ     $a001
getpix          equ     $a002


start:
                .dc.w   init            * $A000 aufrufen
                move.l  intin(a0),a3    * Adresse des INTIN-Arrays
                move.l  ptsin(a0),a4    * Adresse des PTSIN-Arrays

                move    #300,(a4)       * X-Koordinate
                move    #100,2(a4)      * Y-Koordinate

                move    #1,(a3)         * Farbwert, Pixel setzen
*                                       0 löscht Pixel
```

```
                .dc.w   setpix          * Pixel setzen

                move    #300,(a4)       * X-Koordinate
                move    #100,2(a4)      * Y-Koordinate

                .dc.w   getpix          * Farbwert holen

*               d0 enthält jetzt den Farbwert
```

Als Farbwert ist bei Monochrom-Monitoren nur Null oder Eins sinnvoll. Bei der Darstellung in einer der Farbmodi jedoch kann hier auch ein anderer Wert eingetragen werden.

Das nächste Beispiel zeigt, wie mit der Funktion FILLED POLYGON ein Dreieck auf dem Bildschirm gezeichnet wird.

```
*********************************************************************
*
*
*               a006  -  filled polygon
*
*               rbr     28.09.85
*
*********************************************************************

contrl          equ     4
ptsin           equ     12

fg_bp1          equ     24
fg_bp2          equ     26
fg_bp3          equ     28
fg_bp4          equ     30
wrt_mod         equ     36

y1              equ     40

patptr          equ     46
patmsk          equ     50
multifill       equ     52
```

```
clip            equ     54
xmn_clip        equ     56
ymn_clip        equ     58
xmx_clip        equ     60
ymx_clip        equ     62


init            equ     $a000
polygon         equ     $a006


                .dc.w   init            * Adresse des Variablenblocks
*                                         nach A0 holen

        move.w  #1,fg_bp1(a0)           * Farbe festlegen
        clr.w   fg_bp2(a0)              * Nur Monochrom
        clr.w   fg_bp3(a0)
        clr.w   fg_bp4(a0)

        move.w  #2,wrt_mod(a0)          * Replace-Mode

        move.l  #fill,patptr(a0)        * Pointer auf das Füllmuster
        move.w  #4,patmsk(a0)           * Vier Füllmuster
        clr.w   multifill(a0)           * nur eine Plane (Monochrom)
        clr.w   clip(a0)                * Kein Clipping

        move.l  contrl(a0),a6           * Adresse des CONTRL-Arrays nach A6
        addq.l  #2,a6                   * A6 > CONTRL(1)
        move.w  #3,(a6)                 * drei XY-Paare in PTSIN

        move.l  ptsin(a0),a6            * Adresse PSTIN-Array nach A6
        move.l  #tab,a5                 * Tabelle der Koordinaten
        move.w  #8,d3                   * acht Koordinaten übertragen
loop    move.w  (a5)+,(a6)+
        dbra    d3,loop

        move.w  #100,d3                 * erste Scanline
loop1   move.w  d3,y1(a0)               * nach Y1
```

```
        move.l  a0,-(sp)            * Adresse Variablenblock retten

        dc.w    polygon             * Scanline füllen, A0 zerstört

        move.l  (sp)+,a0            * A0 restaurieren
        addq.w  #1,d3               * nächste Scanline berechnen
        cmp.w   #301,d3             * letzte Scanline gezeichnet?
        bne     loop1               * nein, nächste Scanline
        rts                         * all done, we made it

fill:
        dc.w    %1100110011001100
        dc.w    %0110011001100110
        dc.w    %0011001100110011
        dc.w    %1001100110011001

tab:
        dc.w    320,100
        dc.w    120,300
        dc.w    520,300
        dc.w    320,100
```

Das nächste Beispiel zeigt, wie die Cursorform manipuliert werden kann und wie der Cursor anschließend eingeschaltet wird.

```
*****************************************************************
*
*
*       show mouse - transform  mouse
*
*
*****************************************************************
intin           equ     8

init_a          equ     $a000
show_mouse      equ     $a009
transmouse      equ     $a00b

start:
        .dc.w   init_a                  * Adresse INTIN nach A5

        move.l  intin(a0),a5
        move    #0,6(a5)                * INTIN (3) = Maskenfarbwert
        move    #1,8(a5)                * INTIN (4) = Datenfarbwert

        add.l   #10,a5                  * a5 > INTIN (5)

        lea     maus,a4                 * Daten für neuen Cursor
        move    #16,d0                  * 32 Words = 16 Longs

loop:
        move.l  (a4)+,(a5)+             * ins INTIN-Array übertragen
        dbra    d0,loop

        .dc.w   transmouse              * und die Form setzen

        .dc.w   init_a

        move.l  intin(a0),a0
        clr.w   (a0)                    * Anzahl Hide Cursor-Aufrufe ignorieren
```

```
        .dc.w   show_mouse              * jetzt noch den Cursor an

        rts                             * alles getan, was nötig ist
maus:
maske:
        .dc.w   %0000000110000000
        .dc.w   %0000011111100000
        .dc.w   %0001111111111000
        .dc.w   %0111111111111110
        .dc.w   %1111111111111111
        .dc.w   %1111001111001111
        .dc.w   %1111001111001111
        .dc.w   %1111001111001111
        .dc.w   %0000001111000000
        .dc.w   %0000001111000000
        .dc.w   %0000001111000000
        .dc.w   %0000001111000000
        .dc.w   %0000001111000000
        .dc.w   %0000001111000000
        .dc.w   %0000000000000000
daten:
        .dc.w   %0000000000000000
        .dc.w   %0000000110000000
        .dc.w   %0000011001100000
        .dc.w   %0001100000011000
        .dc.w   %0110000110000110
        .dc.w   %0110000110000110
        .dc.w   %0000000110000000
        .dc.w   %0000000110000000
        .dc.w   %0000000110000000
        .dc.w   %0000000110000000
        .dc.w   %0000000110000000
        .dc.w   %0000000110000000
        .dc.w   %0000000110000000
        .dc.w   %0000000000000000
        .dc.w   %0000000000000000
```

Ein letztes, etwas umfangreiches Beispiel soll in groben Zügen die Nutzung des Text-Bit-Block $A008 demonstrieren. Diese Funktion ist jedoch so umfangreich, das im Beispiel nur wenige Möglichkeiten genutzt sind. Betrachten Sie dieses, aber auch die vorigen drei Beispiele nur als Aufforderung zu eigenen Experimenten. Es lohnt sich wirklich, die $A00X-Opcodes zu erforschen.
Das Programm erzeugt ein vergrößertes 'A' in Outlined-Schriftart aus dem 16x8-Systemfont. Andere Effekte können von Ihnen nach Bedarf dazugefügt werden.

```
*****************************************************************
*
*       txtblt aus system font ATARI ST
*       rb      16/08/85
*               20/08/85        scaling
*
*****************************************************************

*****************************************************************
*
*       Equates aus dem Variablenblock
*
*****************************************************************

wrt_mode        equ     36

clip            equ     54
xmn_clip        equ     56
ymn_clip        equ     58
xmx_clip        equ     60
ymx_clip        equ     62

xacc_dda        equ     64
dda_inc         equ     66
t_sclsts        equ     68
mono_status     equ     70

sourcex         equ     72
```

```
sourcey         equ     74

destx           equ     76
desty           equ     78

delx            equ     80
dely            equ     82

fbase           equ     84
fwidth          equ     88

style           equ     90
litemask        equ     92
skewmask        equ     94
weight          equ     96
r_off           equ     98
l_off           equ     100
scale           equ     102
chup            equ     104
text_fg         equ     106
scrtchp         equ     108
scrpt2          equ     112
text_bg         equ     114

*********************************************************************
*
*       Equates der benutzten Opcodes
*
*********************************************************************

init_a          equ     $a000
textblt         equ     $a008

*********************************************************************
*
*       Equates aus dem Fontheader
*
*********************************************************************
```

```
first_ade       equ     36
last_ade        equ     38
off_table       equ     72
fontdat         equ     76
formwidth       equ     80
formheight      equ     82


*****************************************************************
*
*       ein zeichen ausgeben
*
*****************************************************************

start   move    #200,-(sp)              Y-position
        move    #580,-(sp)              X-position
        move    #2,-(sp)                Font-nummer
        move    #$41,-(sp)              Character = 'A'
        bsr     printchar               Alle Daten auf den Stack
        addq.l  #8,sp
        rts


printchar:

        dc.w    init_a                  a000 aufrufen

        move    6(sp),d1                fontnummer
        lsl     #2,d1                   mal 2
        move.l  (a1,d1),a2              adresse des fontheaders nach a2
        move    4(sp),d0                char nach d0
        move    #0,wrt_mode(a0)         overwrite
        move    #$f,text_fg(a0)
        move    #0,text_bg(a0)
        move.l  fontdat(a2),a5          anfang bitmuster der char's
        move.l  a5,fbase(a0)
        move    formwidth(a2),a5        pixel/char horizontal
        move    a5,fwidth(a0)
        sub     first_ade(a2),d0        erstes char im set
```

```
        move.l  off_table(a2),a1
        lsl     #1,d0                   worttabelle
        move    (a1,d0),a5              anfang bitmuster des gew. char
        move    a5,sourcex(a0)
        move    #0,sourcey(a0)
        move    8(sp),destx(a0)         x-position
        move    10(sp),desty(a0)        y-position
        addq    #2,d0                   naechster character
        move    (a1,d0),d0
        sub     sourcex(a0),d0
        move    d0,delx(a0)
        move    formheight(a2),a5
        move    a5,dely(a0)
        move    #1,scale(a0)            scaling einschalten
        move    #$FFFF,dda_inc(a0)      einzig wirksamer wert
        move    #1,t_sclsts(a0)         groesser ja, kleiner nicht?
        move    %10000,style(a0)        outlined
        move    #0,r_off(a0)
        move    #0,l_off(a0)
        move    #900,chup(a0)           char rotation um 90 grad
        move    #1,mono_status(a0)      monospaced flag
        move    #$8000,xacc_dda(a0)
        move    #0,clip(a0)             clip flag
        move.l  #buffer,scrtchp(a0)     buffer fuer special effects
        move    #$40,scrpt2(a0)         pointer in buffer fuer scaling

        dc.w    textblt

        rts
.bss
buffer:  .ds.b   $500                   wer weiss wie gross???
```

3.5 Die Exception-Vektoren

Die ersten 1024 Byte im Adreßbereich des 68000-Prozessors sind für Exception-Vektoren reserviert. Dort sind die Adressen von Routinen hinterlegt, die bei besonderen Umständen ein Ausnahmebehandlung einleiten.

Eine Bedingung, die zu einer Ausnahmebehandlung führt, kann entweder vom Prozessor selbst kommen oder von den an ihn angeschlossenen Peripheriebausteinen und Steuereinheiten. Zu den von außen kommenden Unterbrechungen gehören zum einen die Interrupts, die im nächsten Kapitel beschrieben sind. Des weiteren kann von außen ein sogenannter Busfehler erzeugt werden.

Ein Busfehler kann durch vielerlei Umstände erzeugt werden. Zum einen werden damit bestimmte Speicherbereiche vor unberechtigtem Zugriff geschützt. Wie Sie vielleicht bereits wissen, kann der 68000 ein zwei Betriebsarten arbeiten. In der ersten Ebene, dem Supervisor-Modus, wird das Betriebssystem betrieben. Der User-Modus ist für Anwenderprogramme bestimmt. Damit ein Anwenderprogramm nicht unkontrolliert auf wichtige Systemvariablen sowie auf die Peripheriebausteine zugreifen kann, wird bei einem Zugriff im User-Modus ein Busfehler erzeugt. Tritt ein solcher Fehler auf, so bricht der Prozessor die Abarbeitung des Befehls ab, rettet Programmzähler und Statusregister auf den Stack und verzweigt in eine Routine, deren Adresse er sich aus den untersten 1024 Byte des Speichers holt. Im Falle des Busfehlers steht die Adresse ab Speicherstelle 8 (ein Langwort). Was passiert nun in dieser Routine?

Zuerst wird die Vektornummer der Unterbrechung ermittelt, im Falle des Busfehlers ist dies eine 2. Nun werden auf dem Bildschirm 2 Explosionswolken ('mushrooms') dargestellt. Der Anwender kann aus der Anzahl der Bilder auf die Vektornummer schließen. Anschließend wird wieder zum GEM Desktop zurückgekehrt.

Die Tabelle enthält sämtliche Ausnahmevektoren.

Vektornummer	Adresse	Bedeutung
0	$000	Stackpointer nach Reset
1	$004	Programmzähler nach Reset
2	$008	Busfehler
3	$00C	Adreßfehler
4	$010	Illegaler Befehl
5	$014	Division durch Null
6	$018	CHK-Befehl
7	$01C	TRAPV-Befehl
8	$020	Privileg-Verletzung
9	$024	Trace
10	$028	Line A Emulator
11	$02C	Line F Emulator
12-14	$030-$038	reserviert
15	$03C	nichtinitialisierter Interrupt
16-23	$040-$05C	reserviert
24	$060	spurios Interrupt
25	$064	Level 1 Interrupt
26	$068	Level 2 Interrupt
27	$06C	Level 3 Interrupt
28	$070	Level 4 Interrupt
29	$074	Level 5 Interrupt
30	$078	Level 6 Interrupt
31	$07C	Level 7 Interrupt
32	$080	TRAP #0 Befehl
33	$084	TRAP #1 Befehl
34	$088	TRAP #2 Befehl
35	$08C	TRAP #3 Befehl
36	$090	TRAP #4 Befehl
37	$094	TRAP #5 Befehl
38	$098	TRAP #6 Befehl
39	$09C	TRAP #7 Befehl
40	$0A0	TRAP #8 Befehl
41	$0A4	TRAP #9 Befehl
42	$0A8	TRAP #10 Befehl
43	$0AC	TRAP #11 Befehl
44	$0B0	TRAP #12 Befehl

```
  45          $0B4    TRAP #13 Befehl
  46          $0B8    TRAP #14 Befehl
  47          $0BC    TRAP #15 Befehl
 48-63     $0C0-$0FC  reserviert
 64-255    $100-$3FC  User Interrupt-Vektoren
```

Beim ATARI ST werden folgende Vektoren benutzt:

```
Line A Emulator      $EB9A
Level 2 Interrupt    $543C
Level 4 Interrupt    $5452
TRAP #1 GEMDOS       $965E
TRAP #2 GEM          $2A338
TRAP #13 BIOS        $556C
TRAP #14 XBIOS       $5566
```

Der Vektor für Division durch Null zeigt auf 'rte' und kehrt direkt ins unterbrochene Programm zurück. Die Vektoren 64-79 sind für die Interrupts des MFP 68901 reserviert. Alle anderen Vektoren zeigen auf Adresse $5838, die die Vektornummer ausgibt und das Programm beendet, wie es beim Busfehler beschrieben wurde.

Sämtliche nicht benutzten Vektoren können Sie für eigene Zwecke nutzen, z.B. den Line F Emulator oder die 12 nicht benutzten Traps.

3.5.1 Die Interrupt-Struktur des ATARI ST

Im ATARI ST werden die Interrupt-Möglichkeiten, die der MC68000-Prozessor bietet, voll ausgenutzt. Wie Sie vielleicht aus der Hardwarebeschreibung des Prozessors schon entnommen haben, hat der Prozessor sieben Interrupt-Ebenen mit unterschiedlicher Priorität. Welche Ebenen effektiv einen Interrupt auslösen können, bestimmt die Interrupt-Maske im Systembyte des Statusregisters. Ein Interrupt kann nur von der Ebene ausgelöst werden, die höher als die augenblickliche Maske im Statusregister liegt.

Ein Interrupt einer bestimmten Priorität wird dem Prozessor durch die drei Interrupt Priority Level Eingänge mitgeteilt. Dabei ergibt sich folgende Zuordnung:

Ebene	IPL 2	1	0
7 (NMI)	0	0	0
6	0	0	1
5	0	1	0
4	0	1	1
3	1	0	0
2	1	0	1
1	1	1	0
0	1	1	1

Sind alle drei Leitungen auf 1 (Interrupt-Ebene 0), so liegt kein Interrupt an. Die Interrupt-Ebene 7 ist der sogenannte NMI (nicht maskierbarer Interrupt), der auch bei dann ausgeführt wird, wenn die Interrupt-Maske im Statusregister auf sieben ist. Welchem Interrupt nun welcher Interrupt-Vektor zugeordnet wird (d.h. die Adresse der Routine, in der der Interrupt bearbeitet wird), hängt von dem Peripheriebaustein ab, der den Interrupt auslöst. Bei den sogenannten Auto-Vektoren leitet der Prozessor selbst aus der Interrupt-Ebene die Interrupt-Nummmer ab. Dabei wird folgende Tabelle benutzt:

Ebene	Vektornummer	Vektoradresse
IPL 1	25	$64
IPL 2	26	$68
IPL 3	27	$6C
IPL 4	28	$70
IPL 5	29	$74
IPL 6	30	$78
IPL 7	31	$7C

Beim ATARI ST sind nur die Leitungen IPL 1 und IPL 2 benutzt; die Leitung IPL 0 ist permanent auf 1-Pegel, so daß nur die Ebenen 2, 4 und 6 zur Verfügung stehen. Dabei ergibt sich folgende Zuordnung:

```
IPL 2      HBL,  horizontal blank, Zeilenrücklauf
IPL 4      VBL,  vertical blank, Bildrücklauf
IPL 6      MFP 68901
```

Der HBL-Interrupt wird bei jedem Zeilenrücklauf vom Videoteil generiert. Abhängig vom angeschlossenen Monitor (Monochrom oder Farbe) wird er alle 50 bis 64 us generiert. Er tritt somit sehr häufig auf und ist normalerweise durch eine Interrupt-Maske von drei nicht erlaubt. Die Standard HBL-Routine hat deshalb nur die Aufgabe, die Interrupt-Maske, falls sie auf Null ist und den HBL-Interrupt ermöglicht, auf drei zu erhöhen, so daß nachfolgende HBLs keinen Interrupt mehr auslösen können. Eine Anwendung könnte der HBL-Interrupt für spezielle Bildschirmeffekte erfahren. Mit Hilfe dieses Interrupts weiß man ja genau, welche Zeile des Bildes gerade dargestellt wurde. Von viel größerer Bedeutung ist jedoch der VBL-Interrupt, der bei jedem Bildrücklauf ausgelöst wird. Abhängig vom angeschlossenen Monitor sind dies 50-, 60- oder 70mal in der Sekunde.

Die Vertical Blank Interruptroutine erledigt eine ganze Menge von Aufgaben, die periodisch ausgeführt werden müssen oder die Bildschirmdarstellung betreffen. Beim Eintritt in die Routine wird zuerst der Bildzähler (frame counter) 'frclock' $466 erhöht. Als nächstes wird getestet, ob der

VBL-Interrupt softwaremäßig gesperrt ist. Dies ist der Fall wenn 'vblsem' $452 (Vertical Blank Semaphor) null oder negativ ist. In diesem Fall wird die Abarbeitung der Routine sofort abgebrochen und wieder in das unterbrochene Programm gesprungen. Ansonsten werden alle Register auf den Stack gerettet und der Zähler 'vbclock' $462, der die ausgeführten VBL-Routinen zählt, erhöht. Als nächstes wird geprüft, ob in der Zwischenzeit ein anderer Monitor angeschlossen wurde. Falls von Monochrom- auf Farbmonitor gewechselt wurde, wird der Videoshifter entsprechend umprogrammiert. Dies ist erforderlich, weil durch die hohe Bildfrequenz von 70 Hz des Monochrommonitors ein angeschlossener Farbmonitor beschädigt werden könnte. Die Routine zum Cursor-Blinken wird als nächstes aufgerufen. Auch wenn Sie über die entsprechenden BIOS-Funktionen eine neue Farbpalette laden oder die Bildschirmadresse ändern wollen, geschieht dies hier in der VBL-Routine. Da zu diesem Zeitpunkt nichts dargestellt wird, kann eine Änderung hier ohne Störungen vorgenommen werden. Falls 'colorptr' $45A ungleich Null ist, wird er als Zeiger auf eine neue Farbpalette interpretiert, und diese in den Videoshifter geladen. Anschließend wird der Zeiger wieder gelöscht. Wenn 'screenpt' gesetzt ist, so wird dieser Wert als neue Basisadresse des Bildschirms übernommen. Damit sind die bildschirmspezifischen Teile abgehandelt.

Nun wird die Floppy-VBL-Routine aufgerufen, die mit Hilfe des Write Protect Status' feststellt, ob eine Diskette gewechselt wurde. Die weitere Aufgabe dieser Routine besteht darin, die Floppies zu deselektieren, nachdem der Disk-Controller den Laufwerksmotor abgeschaltet hat.

Jetzt kommt die Stelle, die für den Programmierer am interessantesten ist, die Abarbeitung der VBL-Queue. Es gibt nämlich die Möglichkeit, dem Betriebssystem mitzuteilen, eigene Routinen innerhalb des VBL-Interrupts auszuführen. Wieviel Routinen dies maximal sind, steht in 'nvbls' $454. Standardmäßig ist dieser Wert auf 8 initialisiert, er kann jedoch bei Bedarf erhöht werden. In Adresse 'vblqueue' $456 steht ein Zeiger auf ein Vektorfeld, das die (8) Adressen

der eigenen VBL-Routinen enthält. Innerhalb der VBL-Routine wird nun jede Adresse getestet und - falls die Adresse nicht Null ist - die angesprochende Routine ausgeführt.

Wollen Sie eine eigene VBL-Routine installieren, so prüfen Sie die 8 Einträge, bis Sie einen finden, der Null enthält. An diese Adresse können Sie einen Zeiger auf Ihre Routine schreiben, die von jetzt an in jedem VBL-Interrupt mit ausgeführt wird. Sollten schon alle 8 Einträge besetzt sein, so können Sie alle Einträge in einen freien Speicherbereich kopieren, die Adresse Ihrer Routine daran anhängen und anschließend 'vblqueue' auf das neue Vektorfeld zeigen lassen. Natürlich dürfen Sie nicht vergessen, 'vbls', die Anzahl der Routinen, entsprechend zu erhöhen. Ihre Routine darf alle Register mit Ausnahme des USP verändern.

Ehe die VBL-Routine beendet wird, wird noch das 'dmpflg' $4EE geprüft. Hat diese Speicherstelle den Wert Null, dann wird eine Hardcopy des Bildschirms ausgegeben. Das Flag wird in der Tastatur-Interruptroutine gesetzt, wenn die Tasten ALT und HELP gleichzeitig gedrückt werden. Anschließend werden die Registerinhalte wieder zurückgeholt, vblsem wird freigegeben und es wird in die unterbrochene Routine zurückverzweigt.

Unter der Interrupt-Ebene sechs stand in unserer obigen Tabelle der MFP 68901 aufgeführt. Dieser Baustein ist in der Lage, selbsttätig Interrupt-Vektoren zu erzeugen. Man spricht hierbei im Gegensatz zu den oben benutzen Auto-Vektoren von Nonauto-Vektoren, weil der Prozessor den Vektor nicht selbst generiert. Der MFP 68901 arbeitet im ATARI ST als Interruptcontroller. Er verwaltet die Interrupt-Anforderungen sämtlicher Peripheriebausteine einschließlich seiner eigenen.

Der MFP kann sechzehn Interrupts verwalten, die gegeneinander priorisiert sind, ähnlich wie die sieben Ebenen des Prozessors. Gegenüber dem 68000 erscheinen alle Interrupts des MFP 68901 auf der Ebene 6, sind also höher priorisiert

als die HBL- und VBL-Interrupts. Die folgende Tabelle enthält die Zuordnungen innerhalb des MFP.

Ebene	Belegung
15	Monochrom Monitor Detect
14	RS232 Ring Indicator
13	System Clock Timer A
12	RS232 Empfängerpuffer voll
11	RS232 Empfängerfehler
10	RS232 Sendepuffer leer
9	RS232 Sendefehler
8	Zeilenrücklaufzähler, Timer B
7	Floppy-Controller und DMA
6	Tastatur und MIDI-ACIAs
5	Timer C
4	RS232 Baudrate-Generator, Timer D
3	unbenutzt
2	RS232 CTS
1	RS232 DCD
0	Centronics Busy

Von diesen möglichen Interrupt-Quellen sind jedoch nicht alle freigegeben; einige Signale werden durch Polling bearbeitet. Im folgenden sind die Interrupts beschrieben, die vom Betriebssystem benutzt werden.

Level 2, RS232 CTS, Adresse $73C0

Dieser Interrupt wird jedesmal ausgelöst, wenn über die CTS-Leitung der RS232-Schnittstelle mitgeteilt wird, daß ein angeschlossener Empfänger weitere Daten empfangen kann. Die Routine sendet dann das nächste Zeichen aus dem RS232-Sendepuffer.

Level 5, Timer C, Adresse $7C5C

Dieser Timer läuft mit 200 Hz. In der Interruptroutine wird zuerst der 200 Hz-Zähler $4BA erhöht. Die nächsten Aktionen werden nur bei jedem 4. Aufruf der Interruptroutine

durchgeführt, also nur alle 20 ms (50 Hz). Zuerst wird eine Routine aufgerufen, die die Sound-Verarbeitung übernimmt. Eine weitere Aufgabe dieses Interrupts ist die Tastenwiederholung (Repeat) bei gedrückter Taste und eingeschaltetem Repeat. Zum Abschluß wird noch die evt timer Routine des GEM aufgerufen, die über den Vektor $400 angesprungen wird.

Level 6, Tastatur und MIDI, Adresse $752A

Auf dieser Interrupt-Ebene des MFP sind zwei Peripheriebausteine angeschlossen, die beiden ACIAs, die Daten von der Tastatur und von der MIDI-Schnittstelle empfangen. Um zu entscheiden, welcher der beiden Bausteine einen Interrupt angefordert hat, werden die Interrupt-Request-Bits im Statusregister der ACIAs getestet und das empfangene Byte wird falls erforderlich abgeholt. Wenn es von der Tastatur kommt, wird der Scancode über die Tastaturtabelle in den ASCII-Code umgewandelt und in den Empfangspuffer geschrieben, was bei MIDI-Daten sofort geschieht. Von der Tastatur-ACIA kommen ebenfalls Maus- und Joystick-Daten, die auch entsprechend aufbereitet werden.

Level 9, RS232 Sendefehler, Adresse $7426

Falls beim Senden von RS232-Daten ein Fehler auftritt, wird diese Interruptroutine aktiviert. Hier wird das Transmitter-Statusregister gelesen und der Status im RS232 Parameterblock vermerkt.

Level 10, RS232 Sendepuffer leer, Adresse $7374

Jedesmal wenn der MFP ein Datenbyte komplett über die RS232 Schnittstelle ausgegeben hat, löst er diesen Interrupt aus. Er ist dann bereit, das nächste Byte zu senden. Falls noch Daten im Sendepuffer stehen, wird das nächste Byte ins Senderegister geschrieben, das jetzt entsprechend der gewählten Baudrate bitweise herausgeschoben werden kann.

Level 11, RS232 Empfängerfehler, Adresse $7408

Falls beim Empfang von RS232 Daten ein Fehler auftritt, wird diese Interruptroutine aktiviert. Dabei kann es sich z.B. um einen Parity-Fehler oder einen Überlauf handeln. Die Routine löscht nur das Empfängerstatus-Register und kehrt dann zurück.

Level 12 RS232 Empfängerpuffer voll, Adresse $72C0

Wenn der MFP ein komplettes Byte empfangen hat, tritt dieser Interrupt auf. Hier kann das Zeichen abgeholt und in den Empfängerpuffer geschrieben werden(falls dort noch Platz ist). In dieser Routine wird auch ein evtl. eingestellter Handshake-Modus berücksichtigt (Senden von XON/XOFF bzw. RTS/CTS).

Die anderen Interruptmöglichkeiten des MFP werden nicht benutzt, können jedoch für eigene Routinen eingesetzt werden. Z.B. kann der Interrupt Level 0, Centronics Strobe für eine gepufferte Druckerausgabe benutzt werden.

3.6 Der VT52-Emulator des ATARI ST

Bei der Textausgabe auf den Bildschirm haben Sie beim ATARI ST zwei Möglichkeiten. Zum einen können Sie sich der GEMDOS Funktionen mittels TRAP#1 oder einem direkten BIOS-Aufruf mit TRAP#13 bedienen. Die andere Möglichkeit besteht in der Nutzung der VDI-Funktionen.

Bei beiden Varianten haben Sie spezielle Möglichkeiten zur Bildschirmsteuerung. Betrachten wir zunächst die Ausgabe über die normalen DOS- bzw. BIOS-Aufrufe. Hier wurde bei der Bildschirmausgabe ein Terminal vom Typ VT52 emuliert, das eine ganze Anzahl von Steuerfunktionen zur Verfügung stellt. Diese Steuerfunktionen werden durch ein spezielles Sonderzeichen eingeleitet, den Escape-Code. Escape oder abgekürzt ESC hat den ASCII-Code 27. Dem ESC-Code folgen noch ein Buchstabe, der die Funktion bestimmt, sowie bei Bedarf einige weitere Parameter. In der folgenden Aufstellung sind sämtliche Steuercodes und ihre Bedeutung aufgelistet.

ESC A Cursor hoch
Damit wird der Cursor um eine Zeile nach oben bewegt. Befand sich der Cursor schon in der obersten Zeile, so passiert nichts.

ESC B Cursor runter
Diese ESC-Sequenz positioniert den Cursor um eine Zeile nach unten. Befindet sich der Cursor schon in der untersten Zeile, so geschieht wiederum nichts.

ESC C Cursor rechts
Um eine Spalte nach rechts wird der Cursor durch diese Sequenz bewegt.

ESC D Cursor links
Damit wird der Cursor um eine Position nach links bewegt. Diese Funktion ist identisch mit dem Kontrollcode Backspace (BS, ASCII-Code 8). Befindet sich der Cursor schon in ersten Spalte, geschieht

nichts.

ESC E Clear Home

Diese Steuersequenz löscht den kompletten Bildschirm und positioniert den Cursor in die linke obere Bildschirmecke (Home-Position).

ESC H Cusor Home

Damit können Sie den Cursor in die linke obere Bildschirmecke bringen, ohne jedoch den Bildschirminhalt zu löschen.

ESC I Cursor nach oben

Diese Sequenz bewegt den Cursor um eine Zeile nach oben. Im Gegensatz zu ESC A wird jedoch dann, wenn der Cursor sich schon in der obersten Zeile befand, eine Leerzeile eingefügt und der Rest des Bildschirms entsprechend um eine Zeile nach unten gescrollt. Die Spaltenposition des Cursors bleibt dabei unverändert.

ESC J Rest des Bildschirms löschen

Mittels dieser Funktion wird der Rest des Bildschirms ab der augenblicklichen Cursorposition einschließlich gelöscht. Der Cursor bleibt dabei an derselben Stelle.

ESC K Rest der Zeile löschen

Diese ESC-Sequenz löscht den Rest der Zeile, in der sich der Cursor befindet. Gelöscht wird ab Cursorposition einschließlich; die Cursorposition selbst wird nicht verändert.

ESC L Zeile einfügen

Damit ist es möglich, an der augenblicklichen Cursorposition eine Leerzeile einzufügen. Der Rest des Bildschirms wird nach unten geschoben; die unterste Zeile geht dabei verloren. Nach dem Einfügen steht der Cursor am Beginn der neuen Zeile.

ESC M Zeile löschen

Diese Funktion löscht die Zeile, in der sich der Cursor befindet und rückt den Rest des Bildschirms nach oben. Die unterste Bildschirmzeile wird dadurch frei. Der Cursor befindet sich nach dem Löschen in der ersten Spalte der nachgerückten Zeile.

ESC Y Cursor positionieren

Dies ist mit die wichtigste Funktion. Sie erlaubt es, den Cursor beliebig auf dem Bildschirm zu positionieren. Dazu braucht die Funktion noch die Cursorzeile und -spalte als Parameter, die in dieser Reihenfolge mit einem Offset von 32 erwartet werden. Wollen Sie den Cursor in Zeile 7, Spalte 40 setzen, so müssen Sie die Sequenz ESC Y CHR$(32+7) CHR$(32+40) ausgeben. Zeilen und Spalten werden dabei von Null an gezählt; bei einem 25 * 80 Bildschirm sind die Zeilen von 0 bis 24 und die Spalten von 0 bis 79 nummeriert.

Die weiteren ESC-Sequenzen des VT52-Terminals beginnen mit einem Kleinbuchstaben.

ESC b Schriftfarbe wählen

Mit dieser Funktion können Sie die Schriftfarbe für die weitere Ausgabe wählen. Bei einem Monochrommonitor haben Sie nur die Wahl zwischen 0 = weiß und 1 = schwarz. Bei Farbdarstellung können Sie je nach Modus aus 4 oder 16 Farben auswählen. Bei dem Parameter werden nur die untersten vier Bits ausgewertet (mod 16). Sie können für die Farbe eins sowohl die Ziffer '1', als auch den Buchstaben 'A' oder 'a' verwenden.

ESC c Hintergrundfarbe wählen

Diese Funktion dient analog zur Wahl der Hintergrundfarbe. Wählen Sie für Schrift- und Hintergrund die gleiche Farbe, so können Sie die nachfolgende Textausgabe natürlich nicht mehr sehen.

ESC d Bildschirm bis Cursorposition löschen
Durch diese Sequenz wird der Bildschirm vom Anfang bis zur augenblicklichen Cursorposition einschließlich gelöscht. Die Position des Cursors wird dabei nicht verändert.

ESC e Cursor einschalten
Durch diese Escape-Sequenz wird der Cursor sichtbar, z.B. kann der Cursor eingeschaltet werden, wenn vom Benutzer ein Eingabe erwartet wird.

ESC f Cursor ausschalten
Hierdurch wird der Cursor wieder abgeschaltet.

ESC j Cursorposition speichern
Wollen Sie die augenblickliche Position des Cursors speichern, so bedienen Sie sich dazu dieser Sequenz.

ESC k Cursor auf gespeicherte Position setzen
Das Gegenstück zu der obigen Funktion ist diese. Damit wird der Cursor wieder auf die Position gesetzt, die vorher mit ESC j gespeichert wurde.

ESC l Zeile löschen
Damit wird die Zeile gelöscht, in der sich der Cursor befindet. Die übrigen Zeilen bleiben davon unbeeinträchtigt. Nach dem Löschen befindet sich der Cursor in der ersten Spalte der gelöschten Zeile.

ESC o Zeilenanfang löschen
Hiermit wird der Anfang der Cursorzeile bis zur Cursorposition einschließlich gelöscht. Die Position des Cursors bleibt unverändert.

ESC p Reverse ein
Durch diese Sequenz wird auf invertierte Ausgabe geschaltet. Dabei werden für die nachfolgende Ausgabe Schrift- und Hintergrundfarbe vertauscht. Bei einem Monochrommonitor erhalten Sie weiße Schrift auf

schwarzem Grund.

ESC q Reverse aus
Zum Zurückschalten auf Normaldarstellung dient diese Sequenz.

ESC v Automatischer Überlauf ein
Nach Ausführen dieser Sequenz wird bei einer versuchten Ausgabe über das Zeilenende hinaus automatisch eine neue Zeile begonnen.

ESC w Automatischer Überlauf aus
Damit wird die obige Sequenz wieder aufgehoben. Beim Versuch, über das Zeilenende zu schreiben, werden alle nachfolgenden Zeichen in die letzte Spalte dieser Zeile geschrieben.

Ähnliche Funktionen stehen Ihnen auch unter VDI zur Verfügung. Die VDI Escape-Funktionen (Opcode 5) dienen diesem Zweck. Durch Auswahl einer Funktionsnummer wird die entsprechende Bildschirmfunktion ausgewählt. Beachten Sie jedoch, daß unter VDI Zeilen- und Spaltenzählung nicht bei Null, sondern bei eins beginnen.

Unter VDI gibt es auch eine Funktion, die einen String an einer bestimmen Bildschirmkoordinate ausgibt. Dabei können Sie bei Bedarf noch zusätzlich die ESC-Funktionen der VT52 Emulation benutzen.

Die Ausgabe von 'nichtdruckbaren' Steuerzeichen

Die drei Systemfonts des ATARI ST haben auch Zeichen für die ASCII-Codes Null bis 31 vorgesehen, die normalerweise als Steuercodes interpretiert werden. Beim ATARI ST haben nur die Codes 7 (BEL, Tonausgabe), 8 (BS Backspace), 9 (TAB, Tabulator) sowie 10, 11 und 12 (LF, Linefeed, VT Vertical Tabulator, und FF Form Feed bewirken alle einen Zeilenvorschub) sowie 13 (CR, Carriage Return) eine Wirkung sowie natürlich ESC. Die übrigen Codes bewirken nichts. Wie

kommt man nun an die Zeichen mit ASCII-Codes unter 32 ?

Dazu ist in der BIOS-Funktion 3 'conout' noch eine weitere Gerätenummer vorgesehen. Normalerweise dient die Nummer 2 'con' zur Ausgabe auf den Bildschirm. Wählt man jedoch die Nummer 5, so werden sämtliche Codes von 0 bis 255 als druckbare Zeichen ausgegeben, Steuercodes werden nicht mehr berücksichtigt.

Im Anhang finden Sie die drei Systemfonts des ATARI ST abgebildet.

3.7 Die Systemvariablen des ATARI ST

Der ATARI ST benutzt eine Reihe von Systemvariablen, deren Bedeutung und Adressen sich auch in künftigen Versionen des Betriebssystem nicht ändern werden. Sollten Sie andere Variablen, z.B. aus den BIOS-Listing benutzen, die hier nicht aufgeführt sind, so sollten Sie sich immer im Klaren darüber sein, daß diese bei einer neuen Version des Betriebssystems eine andere Bedeutung haben können. Die Systemvariablen sind im unteren RAM-Bereich direkt oberhalb der 68000-ExecptionVektoren untergebracht, also ab Adresse $400 bzw. 1024. Auf den Adreßbereich von 0 bis $800 (2048) kann nur im Supervisormodus zugegriffen werden. Ein Zugriff im Usermodus des 68000 führt zu einem Bus-Error.

In der folgenden Auflistung wird jeweils der Originalname von ATARI benutzt. Neben der Adresse der jeweiligen Variablen wird ein typischer Inhalt angegeben sowie die Bedeutung näher beschrieben.

```
Adresse Länge Name      Bespielinhalt

$400  L   etv_timer          $F526
```

Dies ist der Event-Timer-Vektor des GEM. Damit werden periodische Aufgaben des GEM erledigt.

```
$404  L   etv_critic         $5562
```

Critical Error handler. Unter GEM zeigt dieser Vektor auf Adresse $2A156. Dort wird der Versuch unternommen, Diskettenfehler zu korrigieren, z.B. wird bei Single-Floppy-Systemen aufgefordert, eine andere Diskette einzulegen.

```
$408  L   etv_term           $5328
```

Dies ist der GEM-Vektor zur Beendigung eines Programms.

```
$40C  5L  etv_xtra
```

Hier ist Platz für 5 weitere GEM-Vektoren, die z.Z. jedoch noch nicht genutzt werden.

```
$420  L   memvalid          $752019F3
```

Enhält die Speicherzelle den angegebenen Wert, so ist die Konfiguration des Memeory Controllers gültig.

```
$424  W   memctrl           $0400
```

Hierbei handelt es sich um eine Kopie des Konfigurationswertes im Memory Controller. Der angegebene Wert gilt für eine 512 K Maschine.

```
$426  L   resvalid          $31415926
```

Steht hier der angegebene Wert, so wird bei einem Reset über der Resetvektor in Adresse $42A gesprungen.

```
$42A  L   resvector         $FC0008
```

Siehe oben.

```
$42E  L   phystop           $80000
```

Dies ist das physikalische Ende des RAM-Speicher; $80000 bei einer 512 K Maschine.

```
$432  L   _membot           $3B900
```

Hier beginnt der Benutzerspeicher (TPA, transient program area).

```
$436  L   _memtop           $78000
```

Dies ist das obere Ende des Benutzerspeichers.

```
$43A  L   memval2          $237698AA
```

Dieser 'magische' Wert erklärt zusammen mit 'memvalid' die Speicherkonfiguration für gültig.

```
$43E  W   flock            0
```

Enhält diese Variable einen Wert ungleich Null, so findet gerade ein Floppyzugriff statt und die VBL-Floppy-Routine ist gesperrt.

```
$440  W   seekrate         3
```

Damit wird die Seekrate (Zeit, um den Schreib/Lese-Kopf zum nächsten Track zu bewegen) bestimmt nach folgender Tabelle:

Seekrate	Zeit
0	6 ms
1	12 ms
2	2 ms
3	3 ms

```
$442  W   _timer_ms        $14, 20 ms
```

Der Zeitabstand zwischen zwei Timeraufrufen, 20 ms entspricht 50 Hz.

```
$444  W   _fverify         $FF
```

Enthält diese Speicherstelle einen Wert ungleich Null, so wird nach jedem Floppy-Schreibzugriff ein Verify durchgeführt.

```
$446  W   _bootdev         0
```

Enthält die Gerätenummer des Laufwerks, von dem das Betriebssystem geladen wurde.

```
$448  W   palmode          0
```

Enthält diese Variable einen Wert ungleich Null, so befindet sich das System im PAL-Modus (50 Hz); bei Null bedeutet dies NTSC-Modus (60 Hz).

```
$44A  W   defshiftmod       0
```

Wenn der ATARI von Monochrombetrieb auf Farbe umschaltet, so holt er sich von hier die neue Auflösung (0 = low resolution, 1 = medium resolution).

```
$44C  W   sshiftmd          $200
```

Hier steht eine Kopie der Registerinhalts für die Bildschirmauflösung.

```
        0     320 * 200, low resolution
        1     640 * 200, medium resolution
        2     640 * 400, high resolution
```

```
$44E  L   _v_bas_ad         $78000
```

Diese Variable enthält einen Zeiger auf das Video-RAM (logical screenbase). Die Bildschirmadresse muß immer an einer 256-Byte-Grenze beginnen.

```
$452  W   vblsem            1
```

Ist diese Variable auf Null, so wird die Vertical Blank Routine nicht ausgeführt.

```
$454  W   nvbls             8
```

Anzahl der Vertical Blank Routinen.

```
$456  L   _vblqueue         $4CE
```

Zeiger auf eine Liste von 'nvbls' Routinen, die während des VBL ausgeführt werden.

```
$45A  L   colorptr          0
```

Ist dieser Wert ungleich Null, so wird er als Zeiger auf eine Farbpalette interpretiert, die beim nächsten VBL geladen wird.

```
$45E  L   screenpt          0
```

Dies ist ein Zeiger auf den Beginn des Video-RAMs, der beim nächsten VBL gesetzt wird. (Null falls keine neue Adresse gesetzt werden soll.)

```
$462  L   _vbclock          $2D26A
```

Zähler für die Anzahl der VBL-Interrupts

```
$466  L   _frclock          $2D267
```

Anzahl der ausgeführten VBL-Routinen (nicht durch vblsem gesperrt).

```
$46A  L   hdv_init          $5AE8
```

Vektor zur Harddisk-Initialisierung.

```
$46E  L   swv_vec           $501E
```

Vektor für Änderung der Bildschirmauflösung. Über diesen Vektor wird gesprungen, wenn die Bildschirmauflösung verändert wird (Default ist Reset).

```
$472  L   hdv_bpb           $5B6E
```

Vektor zum Holen des BIOS-Parameter-Blocks für eine Harddisk.

```
$476  L   hdv_rw            $5D88
```

Schreib/Lese-Routine für eine Harddisk.

```
$47A  L   hdv_boot          $60B2
```

Vektor auf eine Routine zum Booten von Harddisk.

```
$47E  L   hdv_mediach       $5D1E
```

Media Change Routine für Harddisk

```
$482  W   _comload          0
```

Wenn diese Variable vom Bootprogramm ungleich Null gesetzt wird, so wird nach dem Laden des Betriebssystems versucht, ein Programm namens 'COMMAND.PRG' zu laden (z.B. eine Anwendung anstelle des Desktop).

```
$484  B   conterm           6
```

Attribut-Vektor für Consol-Ausgabe.

```
    Bit   Bedeutung
     0    Tastatur-Klick ein/aus
     1    Tastatur-Repeat ein/aus
     2    Ton nach CTRL G ein/aus

     3    Bei der BIOS-Funktion 'conin' wird in Bit 24-31
          'kbshift' zurückgeliefert.
```

```
$485  B   unbenutzt, reserviert
```

```
$486  L   trp14ret          0
```

Rücksprungadresse bei TRAP #14 Aufruf

```
$48A  L   criticret         0
```

Return-Adresse des Critical-Error-Handlers.

```
$48E 4L   themd             0
```

Memory Descriptor, wird durch die BIOS-Funktion 'getmpb' ausgefüllt.

```
$49E 2W   ____md          0
```

Platz für weitere Memory Descriptoren.

```
$4A2  L   savptr          $5CE
```

Zeiger auf ein Save-Bereich für die Prozessor-Register nach einem BIOS-Aufruf.

```
$4A6  W   _nflops         2
```

Anzahl der angeschlossenen Floppies.

```
$4A8  L   con_state       $8AEE
```

Vektor für Bildschirmausgabe; wird z.B. bei ESC-Funktionen auf entsprechende Routine gerichtet.

```
$4AC  W   save_row        0
```

Zwischenspeicher für Cursorzeile beim Cursorpositionieren mit ESC Y.

```
$4AE  L   sav_context     0
```

Zeiger auf einen Zwischenspeicher bei einer Execption-Behandlung.

```
$4B2 2L   _bufl           $4F9E, $4FB2
```

Zeiger auf zwei Buffer-List-Header des GEMDOS. Der esrte Header ist für Datensektoren zuständig, der zweite für FAT (file allocation table) und Directory. Jeder Buffer Control Block (BCB) ist folgendermaßen aufgebaut:

```
   long  BCB    $4F8A, Zeiger auf nächsten BCB
   int   drive  -1,    Drivenummer oder -1
   int   typ    2      Puffertyp
   int   rec    $41C   Record-Nummer in diesem Puffer
   int   dirty  0      dirty-Flag (Pufferinhalt verändert)
   long  DMD    $2854  Zeiger auf Drive Media Descriptor
   long  buffer $4292  Zeiger auf den Puffer selbst
```

```
$4BA  L   _hz_200          $71280
```

Zähler für 200 Hz Systemtakt.

```
$4BC 4B   the_env          0
```

Default Environment String, vier Nullbytes.

```
$4C2  L   _drvbits         3
```

32-Bit-Vektor für angeschlossenen Drives. Bit 0 steht für Drive A, Bit 1 für Drive B usw.

```
$4C6  L   _dskbufp         $12BC
```

Zeiger auf einen 1024-Byte Disk-Puffer. Der Puffer wird auf für GSX-Grafik-Operationen benutzt und sollte nicht von Interruptroutinen benutzt werden.

```
$4CA  L   _autopath        0
$4CE 8L   _vbl_list        $15398,0,0...
```

Liste der 8 Standard-VBL-Routinen.

```
$4EE  W   _dumpflg         $FFFF
```

Dieses Flag wird durch gleichzeitiges Drücken von ALT und HELP um eins erhöht. Ein Wert von eins löst eine Bildschirm-Hardcopy auf den Drucker aus. Durch nochmaliges Drücken von ALT HELP wird eine Hardcopy wieder abgebrochen.

```
$4F0  W   _prtabt          0
```

Printer-Abort-Flag wegen Time-Out.

```
$4F2  L   _sysbase          $5000
```

Zeiger auf Beginn des Betriebssystems.

```
$4F6  L   _shell_p          0
```

```
$4FA  L   end_os            $3B900
```

Zeiger auf Endes des Betriebssystems im RAM, Beginn der TPA.

```
$4FE  L   exec_os           $1EB00
```

Zeiger auf Beginn des AES. Wird normalerweise nach der Initialisierung des BIOS angesprungen.

3.8 Der Befehlssatz des 68000

Sollten Sie bereits mit der Maschinensprache irgendeines 8 Bit-Prozessors vertraut sein: Vergessen Sie jetzt bitte alles, was Sie darüber wissen. Das wird Ihr Verständnis der folgenden Ausführungen wesentlich erleichtern!
Der Prozessor 68000 unterscheidet sich grundlegend in Befehlsaufbau und Architektur von den bisher bekannten (auch 8086!) Prozessoren. Der wesentliche Unterschied liegt nicht etwa in der eher unbedeutenden Tatsache, daß die standardmäßige Verarbeitungsbreite nunmehr über 16 Bit und nicht über 8 Bit geht (was manchmal auch hinderlich ist und zu Programmierfehlern verleitet), sondern darin, daß die internen Register, von einigen Ausnahmen abgesehen, nicht einem bestimmten Zweck zugeordnet sind, sondern eher als Mehrzweckregister anzusehen sind, mit denen fast alles machbar ist.
War bisher der Akku immer Ziel arithmetischer Operationen, so fehlt dieser beim 68000 völlig. Es gibt acht Datenregister (D0-D7) mit einer Breite von 32 Bit, von denen in der Regel mindestens eines an Rechenoperationen beteiligt ist und acht Adressregister (A0-A7) mit ebenfalls 32 Bit, die gewöhnlich zur Bildung komplexer Adressen herangezogen werden. Fest zugeordnet ist das Register A7, welches als Stackpointer dient. Dieser ist gleich zweimal vorhanden, nämlich einmal als User-Stackpointer (USP) und einmal als System-Stackpointer (SSP). Die Unterscheidung wird deswegen gemacht, weil es auch zwei Betriebsarten gibt, nämlich den User-Status und den System-Status.
Diese beiden unterscheiden sich nicht nur dadurch, daß sie unterschiedliche Stackpointer benutzen, sondern im User-Status sind auch bestimmte Befehle nicht ausführbar. Das sind die sog. privilegierten Befehle (siehe auch Befehlsbeschreibung), mit deren Hilfe der frisch-fröhliche Programmierer das System ganz schön durcheinanderbringen kann. Deshalb erzeugt der Versuch einer Anwendung sofort eine Exception-Behandlung. Eine Exception ist übrigens auch die einzige Möglichkeit, vom User-Status in den System-Status zu gelangen.

Dann gibt es noch das Status-Byte (16 Bit), dessen obere Hälfte als System-Byte bezeichnet wird, weil hier z.B. die Interrupt-Maske usw. untergebracht sind, also Dinge, die den 'normalen' Anwender nichts angehen, weshalb auch ein Schreibzugriff auf dieses Byte zu den privilegierten Befehlen zählt. Das untere Byte, das sog. User-Byte, enthält die Flags, die abhängig vom Ergebnis von Operationen gesetzt oder gelöscht werden, also Carry-Flag, Zero-Flag usw. In der Regel bezieht sich der Programmierer indirekt auf diese Flags, indem z.B. die Ausführung eines bedingten Sprunges vom Zustand der Flags abhängig gemacht wird.

Zwei Dinge sind noch zu erwähnen: Mehrbyte-Werte (Adressen oder Operanden) werden im Speicher nicht, wie bei den 8-Bittern, in der Reihenfolge Low-Byte High-Byte abgelegt, sondern umgedreht. 4-Byte-Ausdrücke (Langwort) werden also im Speicher (natürlich auch in den Registern) mit dem höchstwertigen Byte zuerst abgelegt.
Das zweite ist der Umstand, daß nicht belegte Operations-Codes nicht zu einem 'Absturz' des Prozessors führen, sondern eine spezielle Exception verursachen, deren ordnungsgemäße Abhandlung natürlich im Betriebssystem verankert sein muß.

3.8.1 ADRESSIERUNGSARTEN

Dies ist das wohl interessanteste Thema beim 68000, weil die enorme Leistungsfähigkeit erst durch die vielfältigen Adressierungsmöglichkeiten richtig zum Tragen kommt.
Soweit es sich bei den effektiven Adressen (das sind Adressen, die, aus u.U. mehreren Komponenten zusammengesetzt, letztendlich den Operanden bestimmen) um Speicheradressen handelt, sind diese grundsätzlich 32 Bit breit, auch wenn eine oder mehrere im Befehl spezifizierte Komponenten kürzer sind. Diese werden immer mit dem Vorzeichen auf die volle Breite gespreizt.
Der Reiz der Adressierung besteht darin, daß bei fast allen Befehlen (natürlich gibt es Einschränkungen) sowohl Quell- als auch Zieloperand mit einer der Adressierungsarten spezifiziert werden kann. Das heißt, daß auch Speicheroperationen nicht zwangsläufig über ein Register führen müssen, also auch Speicher-Speicher-Operationen möglich sind.
In der Assembler-Schreibweise ist zuerst der Quelloperand angegeben, danach (hinter dem Komma) der Zieloperand.

Register direct
Der Operand befindet sich in einem Register. Man unterscheidet zwischen *Data Register Direct* und *Address Register Direct.*
Im ersten Fall darf der Operand bit-, byte-, wort- oder langwortorientiert sein, im zweiten Fall ist Wort oder Langwort vorgeschrieben, falls das Adressregister Ziel der Operation ist.
Beispiel: ADD.B D0,D1 oder ADDA.W D0,A2

Absolute Data Addressing
Der Operand befindet sich im Adressraum des Speichers. Das kann natürlich auch ein Peripheriebaustein sein (siehe MOVEP). Die Adresse wird absolut angegeben.
Diese kann sowohl eine Breite von einem Langwort haben, womit der gesamte Adressraum überstrichen wird, als auch nur ein Wort breit sein. In diesem Fall wird sie vor Ausführung

der Operation um das Vorzeichen (das höchstwertige Bit) auf Langwort erweitert. So wird z.B. aus dem Wort $7FFF das Langwort $00007FFF, aus $FFFF jedoch $FFFFFFFF. Es können also mit der Kurzform nur die untersten und die obersten 32K des Adressraumes angesprochen werden. Das ist im 520ST derart genutzt, daß das Betriebssystem oft mit dieser Adressierungsart auskommt, weil wichtige Systemvariable unten im Speicher stehen und alle Peripheriebausteine ganz oben dekodiert sind.
Beispiel: MOVE.L $7FFF,$01234567
Befehle, in denen beide Operanden mit einem Langwort adressiert werden, zählen mit 10 Byte zu den Längsten.

Program Counter Relative Addressing
Diese Adressierungsart erlaubt es, sogar Konstanten in einem voll verschieblichen Programm zu adressieren, da Basis der Adressberechnung der augenblickliche Stand des Befehlszählers ist.
Es gibt zwei Varianten. Einmal wird ein 16 Bit Offset zum Befehlszähler vorzeichenrichtig hinzuaddiert, zum zweiten wird zusätzlich noch der Inhalt eines Registers (falls dieses nur in Wortbreite angegeben ist, um das Vorzeichen erweitert), allerdings darf hier der Offset nur 8 Bit breit sein.
Beispiel: MOVE.B $1234(PC),$12(PC,D0.W)

Register Indirect Addressing
Hiervon gibt es mehrere Varianten, die einzeln besprochen werden sollen.

Register Indirect
Hierbei befindet sich die Operandenadresse in einem Adressregister.
Beispiel: CLR.L (A0)

Postincrement Register Indirect
Die Adressierung des Operanden findet zunächst wie oben statt, jedoch wird anschließend der Inhalt des angegebenen Adressregisters um die Operandenlänge erhöht, also bei der Angabe xxx.B um 1, bei xxx.L um 4.
Beispiel: BSET.B #0,(A0)+ oder BCLR.L #23,(A1)+

Predecrement Register Indirect
Hier wird vor der Adressierung des Operanden das Adressregister um die Operandenlänge erniedrigt.
Beispiel: CMPI.W #$0123,-(A3)

Register Indirect with Offset
Zum Inhalt des Adressregisters wird ein 16 Bit Offset hinzuaddiert.
Beispiel: EOR.L D0,$1234(A4)

Indexed Register Indirect with Offset
Wie oben, jedoch wird zusätzlich noch der Inhalt eines weiteren (Adress- oder Daten-) Registers vorzeichenrichtig aufaddiert. Der Offset darf hierbei nur eine Breite von 8 Bit haben.
Beispiel: MOVE.W $12(A5,A6.L),D1

Immediate Addressing
Hierbei ist der Operand als solcher im Anhang des Befehles enthalten. Selbstverständlich kann ein derart bestimmter Operand nur als Quelle dienen. Die Immediate-Operanden können in der Regel alle gängigen Breiten annehmen.
Beispiel: ADDI.W #$1234,D5
In der Variante *QUICK* darf die Konstante nur 3 Bit breit sein, also Werte von 0-7 annehmen. Eine Ausnahme ist der MOVE-Befehl, wo die Konstante 8 Bit haben darf, wobei als Ziel aber nur ein Datenregister erlaubt ist.
Beispiel: ADDQ.L #1,A0 oder MOVEQ #123,D1

Implied Register
Diese nur der Vollständigkeit halber erwähnte Adressierungsart besteht darin, daß eine Operandenadresse bereits im Befehl selbst festgelegt ist. Die Operanden befinden sich also entweder im Befehlszähler, im Status-Register oder einem der beiden Stack-Pointer.
Beispiel: MOVE SR,D6

Zu den Offsets wäre noch zu bemerken, daß diese als Zweierkomplement-Zahl angesehen werden. Ihr höchstwertiges Bit bildet also das Vorzeichen. So ist mit einem 8 Bit-Wert ein Offset von +127/-128 möglich, mit 16 Bit ca. +/-32K.

3.8.2 DIE BEFEHLE

Bei der folgenden Befehlsbeschreibung sind die einzelnen Bitmuster nicht aufgeführt, da das in diesem Zusammenhang zu weit führen würde. So soll sie auch nicht der Bitfieselei zur Erstellung von Maschinenprogrammen auf 'altgermanisch' dienen (was ohnehin ziemlich aussichtslos wäre), sondern vielmehr der sicheren Interpretation der in diesem Buch enthaltenen Listings.
Weiterführende Informationen entnehmen Sie bitte speziellen Büchern, z.B. 'Das große Prozessorbuch zum 68000' (DATA BECKER) oder 'HD68000 USERS MANUAL' (HITACHI).
So sind denn auch die Befehle nur in ihrer Grundform erläutert, die möglichen Varianten jedoch nur namentlich aufgeführt. Wie die einzelnen Variationen aussehen können, wollen wir hier kurz erklären.

Die Varianten sind durch einen Nachsatz hinter der Operation gekennzeichnet. Dabei bedeuten:
A ist Kennzeichen dafür, daß das Ziel der Operation ein Adressregister ist. Wortoperanden werden grundsätzlich mit dem Vorzeichenbit auf Langwort erweitert.
I kennzeichnet einen Immediate-Operanden als Quelle der Operation. I-Operanden dürfen in der Regel alle Breiten annehmen.
Q bedeutet 'Quick' und stellt eine Sonderform der Immediate-Adressierung dar. Ein derartiger Operand ist gewöhnlich 3 Bit breit, was einem Wertebereich von 0 bis 7 entspricht. Dieser eingeengte Bereich hat den Vorteil, daß der Operand noch im OP-Code Platz hat. Da ein spezieller Befehl zum Inkrementieren eines Registers nicht vorhanden ist, tut hier z.B. ADDQ.L #1,A0 gute Dienste. Eine Ausnahme bildet MOVEQ. Hier darf der Operand einen Wert von 0-255 aufweisen.
X schließlich kennzeichnet arithmetische Operationen, die sich des X-Flags bedienen. Dieses Flag hat eine besondere Bedeutung. Bei arithmetischen Befehlen wird es gleich dem Carry-Flag gesetzt, welches ja bekanntlich einen Übertrag kennzeichnet. Das Carry-Flag wird jedoch auch von Daten-

transfer-Operationen verändert, das X-Flag jedoch nicht, so daß dieses für weitere Berechnungen zur Verfügung steht. Das ist insbesondere für Rechenvorgänge mit höherer Genauigkeit als die standardmäßigen 32 Bit praktisch, wo Zwischenergebnisse zunächst gerettet werden müssen, wobei jedoch das Carry-Flag seinen Zustand ändern kann.

Nahezu alle Befehle weisen hinter ihrem Operationsteil einen Nachsatz der Art .B, .W oder .L auf. Dieser gibt die Verarbeitungsbreite der Operation an. Obwohl ja z.B. ein Datenregister eine Breite von 32 Bit = 4 Byte = 1 Langwort hat, wird also der Befehl CLR.B D0 nur das niedrigstwertige Byte dieses Registers löschen. Bei Registern spezifiziert .W übrigens das niederwertige Wort. Das höherwertige Wort ist nicht explizit ansprechbar. Befindet sich der Operand im Speicher, so ist wichtig zu wissen, daß .W- und .L-Operanden unbedingt auf einer geraden Adresse beginnen müssen. Dasselbe gilt auch für den OP-Code als solchen, der ja auch immer ein Wort umfaßt.
Ist das Ziel der Operation ein Adressregister, sind nur Operanden vom Typ .W und .L erlaubt, wobei die ersteren mit dem Vorzeichen auf Langwort erweitert werden.
In einigen Listings werden Sie Befehle der Form
MOVE.L #27,D0 finden. Das ist auf die Schreibfaulheit des Programmierers zurückzuführen. Er verläßt sich nämlich darauf, daß der Assembler aus den #27 schon #$0000001B machen wird.

Nun zu den einzelnen Befehlen:

ABCD Add Decimal with Extend
Über ein Datenformat haben wir noch nicht gesprochen: Über das BCD-Format. Das bedeutet nichts weiter als 'Binary Coded Decimal' und kennzeichnet eine Ziffer im Bereich 0-9. Da diese Information nur 4 Bit benötigt, kann man in einem Byte folglich eine zweistellige Dezimalzahl unterbringen. Der Befehl ABCD kann also zwei solcher Zahlen addieren. Die Verarbeitungsbreite ist immer 8 Bit.

ADD Add Binary
Hiermit werden einfach zwei Operanden addiert.
Varianten sind *ADDA*, *ADDQ*, *ADDI* und *ADDX*.

AND Logical And
Zwei Operanden werden entsprechend der AND-Funktion logisch verknüpft.
Variante: *ANDI*

ASL Arithmetik Shift Left
Der Operand wird um die angegebenen Stellen nach links geschoben, wobei das jeweils höchstwertige Bit in das C- und X-Flag kopiert wird. Von rechts wird jeweils eine 0 nachgezogen. Wird ein Datenregister geschoben, ist die Verarbeitungsbreite beliebig. Die Angabe, wie oft geschoben werden soll, ist entweder als I-Operand (3 Bit) angegeben oder in einem weiteren Datenregister hinterlegt. Wird eine Speicherstelle geschoben, ist die Verarbeitungsbreite immer Wort. Ein Zähler wird dann nicht angegeben; er ist immer =1.

ASR Arithmetik Shift Right
Der Operand wird nach rechts geschoben, wobei das jeweils niedrigstwertige Bit nach C und X kopiert wird. Von links wird mit dem höchstwertigen Bit (Vorzeichen) aufgefüllt. Was Verarbeitungsbreite und Zähler betrifft, gilt das gleiche, wie bei ASL.

Bcc Branch Conditionally
Die bedingten Sprünge sind eine Sache für sich. Das Sprungziel ist immer eine relative Adresse, die entweder die Größe eines Byte oder eines Wortes (vorzeichenbehaftet!) hat. Entsprechend überstreicht der Sprung einen Bereich von entweder +127/-128 Byte oder +32K-1/-32K. Bezugspunkt ist die Adresse des folgenden Befehls.
Ob dieser Sprung auch tatsächlich ausgeführt wird, ist von der verlangten Bedingung abhängig, die anhand der Flags verifiziert wird. Hier die Varianten und ihre Bedingungen. Ein Minuszeichen vor einem Flag bedeutet, daß dieses zur

Erfüllung der Bedingung nicht gesetzt sein darf. Logische Verknüpfungen sind mit '*' für AND und '/' für OR gekennzeichnet.

BRA	Branch Always	Bedingungslos
BCC	Branch Carry Clear	-C
BCS	Branch Carry Set	C
BEQ	Branch Equal	Z
BGE	Branch Greater or Equal	N*V/-N*-V
BGT	Branch Greater Than	N*V*-Z/-N*-V*-Z
BHI	Branch Higher	-C*-Z
BLE	Branch Less or Equal	Z/N*-V/-N*V
BLS	Branch Lower or Same	C/Z
BLT	Branch Less Than	N*-V/-N*V
BMI	Branch Minus	N
BNE	Branch Not Equal	-Z
BPL	Branch Plus	-N
BVC	Branch Overflow Clear	-V
BVS	Branch Overflow Set	V

BCHG Bit Test and Change
Das angegebene Bit des Operanden wird invertiert. Der ursprüngliche Zustand kann anhand des Z-Flags festgestellt werden. Der Operand befindet sich entweder im Speicher (Breite=.B) oder in einem Datenregister (Breite=.L). Die Bitnummer wird entweder als I-Operand angegeben, oder sie befindet sich einem Datenregister.

BCLR Bit Test and Clear
Das angegebene Bit wird gelöscht. Alles weitere wie bei BCHG.

BSET Bit Test and Set
Das angegebene Bit wird gesetzt. Randbedingungen wie bei BCHG.

BSR Branch to Subroutine
Dies ist ein unbedingter relativer Sprung in ein Unterprogramm. Sprungweiten wie bei Bcc.

BTST Bit Test
Das Bit wird nur auf seinen Zustand geprüft. Alles weitere wie bei BCHG.

CHK Check Register against Boundaries
Ein Datenregister wird überprüft, ob sein Inhalt kleiner als 0 oder größer als der angegebene Operand ist. Sollte das der Fall sein, führt der Prozessor eine Exception-Behandlung durch. Das Programm wird an der Adresse fortgesetzt, die im Speicher bei $18 (Vektor 6) hinterlegt ist. Andernfalls erfolgt keine besondere Aktion. Die Verarbeitungsbreite ist nur Wort.

CLR Clear Operand
Der angegebene Operand wird gelöscht.

CMP Compare
Der erste Operand wird vom zweiten subtrahiert, ohne jedoch den zweiten Operanden zu verändern. Lediglich die Flags werden entsprechend gesetzt.
Varianten: *CMPA* und *CMPI*
Bei der Variante *CMPM* werden beide Operanden durch die Adressierungsart (Ax)+ angesprochen.

DBcc Test Condition, Decrement and Branch
Ein Datenregister wird heruntergezählt und die Flags auf die angegebene Bedingung überprüft. Ein Sprung wird ausgeführt, wenn entweder die Bedingung erfüllt ist oder das Register -1 ist. Sprungbedingungen und Sprungweiten wie unter Bcc.

DIVS Divide Signed
Der zweite Operand wird vorzeichenrichtig durch den ersten Operanden dividiert. Danach enthält der zweite Operand im unteren Wort das ganzzahlige Ergebnis, im oberen Wort einen evtl. Rest, welcher dasselbe Vorzeichen wie der Quotient aufweist. Als Datenbreite ist für den ersten Operanden .W festgelegt, für den zweiten .L.

DIVU Divide Unsigned
Operation wie oben, jedoch bleibt das Vorzeichen unbeachtet.

EOR Exclusive Or
Die beiden Operanden werden nach den Regeln des EXOR verknüpft.
Variante: *EORI*

EXG Exchange Registers
Die beiden angegebenen Register werden miteinander vertauscht.

EXT Sign Extend
Der Operand wird mit seinem Bit 7 (im Falle .W) oder Bit 15 (.L) auf die angegebene Verarbeitungsbreite aufgefüllt.

JMP Jump
Unbedingter Sprung zu der angegebenen Adresse. Der Unterschied zu BRA besteht darin, daß hier die Adresse nicht relativ, sondern als tatsächliches Sprungziel angesehen wird.

JSR Jump to Subroutine
Sprung zu einem Unterprogramm. Der Unterschied zu BSR ist wie oben beschrieben.

LEA Load Effective Address
Dieser oft mißverstandene Befehl lädt ein Adressregister nicht etwa mit dem Inhalt der angegebenen Operandenadresse, wie bei den anderen Befehlen üblich, sondern *mit der Adresse als solcher*!

LINK Link Stack
Dieser Befehl legt zunächst das angegebene Adressregister auf dem Stack ab. Der Inhalt des Stackpointer (A7) wird dann in dieses Register übertragen und der angegebene Offset auf den Stackpointer aufaddiert.
Mit diesem praktischen Befehl können Datenbereiche für ein Unterprogramm reserviert werden, ohne daß im Programm selbst ein Raum geschaffen werden muß, was bei Programmen, die im Rom ablaufen, auch unmöglich wäre. Der C-Compiler benutzt diese Möglichkeit fleißig für lokale Variable.

LSL Logical Shift Left
Wirkungsweise und Einschränkungen wie ASL.

LSR Logical Shift Right
Wirkungsweise und Einschränkungen wie ASR, jedoch wird hier von links nicht das Vorzeichen nachgezogen, sondern eine 0,

MOVE
Der erste Operand wird zum zweiten Operanden transferiert.
Varianten: *MOVEA, MOVEQ*

MOVEM Move Multiple Registers
Hierbei kann ein Operand aus einer Registerliste bestehen. Auf diese Weise können z.B. mit einem einzigen Befehl sämtliche Register auf den Stack gelegt werden.
Beispiel: MOVEM.L A0-A6/D0-D7,-(A7)

MOVEP Move Peripheral Data
Diese Spezialität wurde eigens zur Handhabung von Peripheriebausteinen geschaffen. Diese verfügen in der Regel nur über einen 8 Bit breiten Datenbus und sind somit an den 68000 entweder über die unteren oder die oberen 8 Bit dessen Datenbus angeschlossen. Soll nun ein Wort oder Langwort transferiert werden, werden die Bytes einzeln über entweder das untere oder obere Byte des Datenbus übergeben, abhängig davon, ob die angegebene Adresse gerade oder ungerade ist. Die Adresse wird dabei immer um zwei erhöht, so daß der

Transfer immer auf der Hälfte des Datenbus fortgesetzt wird, auf der er begonnen wurde. Entsprechend dem Zweck dieses Befehls ist ein Operand immer ein Datenregister, der andere immer vom Typ Register Indirect with Offset.

MULS Multiply Signed
Vorzeichenrichtige Multiplikation der beiden Operanden.

MULU Multiply Unsigned
Multiplikation der beiden Operanden ohne Berücksichtigung des Vorzeichens.

NBCD Negate Decimal with Extend
Ein BCD-Operand wird der Operation 0-Operand-X unterzogen.

NEG Negate Binary
Der Operand wird der Behandlung 0-Operand unterzogen.
Variante: *NEGX*

NOP No Operation
Wie der Name schon sagt, macht dieser Befehl gar nichts.

NOT One's Complement
Der Operand wird invertiert.

OR Logical Or
Die beiden Operanden werden entsprechend der Vorschrift für OR miteinander verknüpft.

PEA Push Effective Address
Die Adresse selbst, nicht deren Inhalt, wird auf den Stack gelegt.

RESET Reset External Devices
Die Reset-Leitung des 68000 ist bidirektional. Hierüber kann also nicht nur der Prozessor von außen zurückgesetzt werden, sondern er kann auch seinerseits mit diesem Befehl alle angeschlossenen Bausteine rücksetzen.
Dies ist ein privilegierter Befehl!

ROL Rotate Left
Der Operand wird nach links geschoben, wobei das herausfallende Bit wieder rechts hereingeholt wird und auch das Carry-Flag beeinflußt. Verarbeitungsbreiten und Schiebezähler wie bei ASL.

ROR Rotate Right
Wie oben, nur von links nach rechts.

ROXL Rotate Left with Extend
Wie ROL, jedoch landet das herausgeschobene Bit zunächst im X-Flag, welches statt seiner hereingeholt wird.

ROXR Rotate Right with Extend
Wie oben, jedoch umgekehrte Schieberichtung.

RTE Return from Exception
Rücksprung aus einer Exception-Routine an die Stelle, wo die Exception aufgetreten war.
Dies ist ein privilegierter Befehl!

RTS Return from Subroutine
Rücksprung aus einem Unterprogramm an die Stelle, an der es aufgerufen wurde.

RTR Return and Restore
Wie oben, jedoch wird vor dem Rücksprung noch das CC-Register (das ist das mit den Flags) vom Stack geholt (auf dem es vorher auch abgelegt worden sein *muß*, weil der Rücksprung sonst in die Prärie geht).

SBCD Subtract Decimal with Extend
Der erste Operand wird vom zweiten subtrahiert. Bezüglich des Datenformats sehen Sie bitte bei ABCD nach.

Scc Set Conditionally
Der Operand (nur .B) wird =$ff gesetzt, wenn die Bedingung erfüllt ist. Andernfalls wird er gelöscht. Bezüglich der möglichen Bedingungen orientieren Sie sich bitte bei Bcc.

STOP
Die Verarbeitung des Prozessors wird angehalten und kann nur durch einen externen Interrupt wieder zum Leben erweckt werden.
Dies ist ein privilegierter Befehl!

SUB Subtract Binary
Der erste Operand wird vom zweiten subtrahiert.

SWAP Swap Register Halves
Die beiden Hälften eines Datenregisters werden miteinander vertauscht.

TAS Test and Set Operand
Der Operand (nur .B) wird auf Vorzeichen und 0 geprüft (Beeinflussung von C- und N-Flag). Danach wird sein Bit 7 auf 1 gesetzt.

TRAP
Von diesem Befehl macht der Anwendungsprogrammierer Gebrauch, wenn er Funktionen des Betriebssystems aufrufen will. Faktisch wird durch diesen Befehl eine Exception ausgelöst, die darin besteht, das Programm bei der durch die angegebene Vektornummer bestimmten Adresse fortzusetzen. Zur Anwendung sehen Sie bitte auch die Kapitel über BIOS und XBIOS.

TRAPV Trap on Overflow
Falls das V-Flag gesetzt ist, wird mit diesem Befehl eine Exception ausgelöst, die darin besteht, daß das Programm an der bei Vektor 7 ($1C) hinterlegten Adresse fortgesetzt wird.

TST Test
Aktion wie TAS, jedoch wird der Operand nicht verändert.

UNLK Unlink
Dieser Befehl bildet das Gegenstück zu LINK. Der Stackpointer (A7) wird mit dem angegebenen Adressregister geladen und dieses mit dem letzten Stackeintrag versorgt. Auf diese Weise wird der mit LINK reservierte Bereich wieder freigegeben.

Nachtrag zu den Condition-Codes: Die bei Bcc aufgeführten Bedingungen sind nicht vollständig, da sie an dieser Stelle keinen Sinn machen. Jedoch gibt es bei den Befehlen DBcc und Scc noch die Varianten T (DBT, ST) und F (DBF, SF). T steht für True und bedeutet, daß die Bedingung auf jeden Fall erfüllt ist. F steht für False und ist das Gegenteil: Die Bedingung ist nie erfüllt.

```
************************************************
005000 601C                     bra     $501E
005002 0100                     dc.b    1,0
005004 0000501E                 dc.l    $501E
005008 00005000                 dc.l    $5000
00500C 00019C00                 dc.l    $19C00
005010 0000501E                 dc.l    $501E
005014 00019BF4                 dc.l    $19BF4
005018 06201985                 dc.l    $06201985
00501C 0000                     dc.w    0
00501E 46FC2700                 move.w  #$2700,SR
005022 23F90000501E0000042A     move.l  $501E,$42A
00502C 23FC3141592600000426     move.l  #$31415926,$426
005036 41F9FFFF8800             lea     $FFFF8800,A0
00503C 10BC0007                 move.b  #7,(A0)
005040 117C00C00002             move.b  #$C0,2(A0)
005046 10BC000E                 move.b  #$E,(A0)
00504A 117C00070002             move.b  #7,2(A0)
005050 083A0000FFC8             btst    #0,$501A(PC)
005056 6708                     beq     $5060
005058 13FC0002FFFF820A         move.b  #2,$FFFF820A
005060 43F9FFFF8240             lea     $FFFF8240,A1
005066 303C000F                 move.w  #$F,D0
00506A 41FA03B0                 lea     $541C(PC),A0
00506E 32D8                     move.w  (A0)+,(A1)+
005070 51C8FFFC                 dbra    D0,$506E
005074 9BCD                     sub.l   A5,A5
005076 307C05FC                 move.w  #$5FC,A0
00507A 327C5000                 move.w  #$5000,A1
00507E 7000                     moveq.l #0,D0
005080 30C0                     move.w  D0,(A0)+
005082 B3C8                     cmp.l   A0,A1
005084 66FA                     bne     $5080
005086 206D042E                 move.l  $42E(A5),A0
00508A 91FC00008000             sub.l   #$8000,A0
005090 2B48044E                 move.l  A0,$44E(A5)
005094 13ED044FFFFF8201         move.b  $44F(A5),$FFFF8201
```

```
ATARI ST BIOS
zum Programmstart
Version 1
Reset-Adresse
Beginn der Betriebssystems
Ende des Betriebssystems
Reset-Adresse

Erstellungsdatum 20. 6. 1985

Supervisor-Modus, keine Interrupts
resvector laden
resmagic nach resvalid
Adresse des Soundchips
Port A und B
auf Ausgabe
Port A selektieren
Floppies deselektieren


syncmode auf 50 Hz Pal
Adresse color0
16 Farben
Adresse der Farbtabelle
Farbpalette laden
nächste Farbe
A5 löschen
Ende der Betriebssystem-Variablen
Beginn des Betriebssystems
D0 löschen
Speicherbereich löschen
Ende schon erreicht ?
nein
phystop, RAM-Ende
minus 32 K
als Videoadresse v bs ad
dbaseh, Videoadresse für Hardware
```

```
00509C 13ED0450FFFF8203     move.b  $450(A5),$FFFF8203   dbasel
0050A4 323C07FF             move.w  #$7FF,D1             ($7FF+1)*16 = 32 K Video-RAM
0050A8 20C0                 move.l  D0,(A0)+
0050AA 20C0                 move.l  D0,(A0)+
0050AC 20C0                 move.l  D0,(A0)+             Bildschirm löschen
0050AE 20C0                 move.l  D0,(A0)+
0050B0 51C9FFF6             dbra    D1,$50A8             nächste 16 Bytes
0050B4 207AFF5E             move.l  $5014(PC),A0
0050B8 0C9087654321         cmp.l   #$87654321,(A0)
0050BE 6704                 beq     $50C4
0050C0 41FAFF46             lea     $5008(PC),A0
0050C4 23E80004000004FA     move.l  4(A0),$4FA           end os
0050CC 23E80008000004FE     move.l  8(A0),$4FE           exec os
0050D4 2B7C00005AE8046A     move.l  #$5AE8,$46A(A5)      hdv init
0050DC 2B7C00005D880476     move.l  #$5D88,$476(A5)      hdv rw
0050E4 2B7C00005B6E0472     move.l  #$5B6E,$472(A5)      hdv bpb
0050EC 2B7C00005D1E047E     move.l  #$5D1E,$47E(A5)      hdv mediach
0050F4 2B7C000060B2047A     move.l  #$60B2,$47A(A5)      hdv boot
0050FC 2B6D044E0436         move.l  $44E(A5),$436(A5)    v bs ad als memory top
005102 2B6D04FA0432         move.l  $4FA(A5),$432(A5)    end os als memory bottom
005108 4FF900003E2A         lea     $3E2A,A7             Stackpointer initialisieren
00510E 3B7C00080454         move.w  #8,$454(A5)          nvbls, Anzahl der VBL-Routinen
005114 50ED0444             st      $444(A5)             fverify, Floppy Verify nach Schreiben
005118 3B7C00030440         move.w  #3,$440(A5)          seekrate für Floppy auf 3 ms
00511E 2B7C000012BC04C6     move.l  #$12BC,$4C6(A5)      dskbufp auf $12BC, Floppypuffer
005126 3B7CFFFF04EE         move.w  #$FFFF,$4EE(A5)      dumpflg für Hardcopy löschen
00512C 2B7C00000500004F2    move.l  #$5000,$4F2(A5)      sysbase, Beginn des Betriebssystems
005134 2B7C000005FC04A2     move.l  #$5FC,$4A2(A5)       savptr für TRAPs auf $5FC
00513C 2B7C00005328046E     move.l  #$5328,$46E(A5)      hdv dsb auf rts
005144 47FA03FC             lea     $5542(PC),A3         Zeiger auf rte
005148 49FA01DE             lea     $5328(PC),A4         Zeiger auf rts
00514C 0CB9FA52235F00FA0000 cmp.l   #$FA52235F,$FA0000   cartbase, Diagnostic-Cartrige eingesetzt?
005156 6726                 beq     $517E                ja, Vektoren nicht initialisieren
005158 43FA06DE             lea     $5838(PC),A1         terminate process
00515C D3FC02000000         add.l   #$2000000,A1         in Bits 24-31 steht Vektornummer
005162 41F900000008         lea     $8,A0                Beginn mit Bus Error
005168 303C003D             move.w  #$3D,D0              Anzahl der Vektoren
```

```
00516C 20C9                move.l  A1,(A0)+          Vektor setzen
00516E D3FC01000000        add.l   #$1000000,A1      Nummer in Bits 24-31 erhöhen
005174 51C8FFF6            dbra    D0,$516C          nächsten Vektor
005178 23CB00000014        move.l  A3,$14            'Division durch Null' auf rte
00517E 2B7C000054520070    move.l  #$5452,$70(A5)    Vertical Blank Interrupt, IPL4
005186 2B7C0000543C0068    move.l  #$543C,$68(A5)    Horizontal Blank Interrupt, IPL2
00518E 2B4B0088            move.l  A3,$88(A5)        TRAP #2 auf rte
005192 2B7C0000556C00B4    move.l  #$556C,$B4(A5)    TRAP #13 Vektor
00519A 2B7C000055600B8     move.l  #$5566,$B8(A5)    TRAP #14 Vektor
0051A2 2B7C0000EB9A0028    move.l  #$EB9A,$28(A5)    LINE A Vektor
0051AA 2B4C0400            move.l  A4,$400(A5)       etv timer auf rts
0051AE 2B7C000055620404    move.l  #$5562,$404(A5)   etv critic Vektor
0051B6 2B4C0408            move.l  A4,$408(A5)       etv term auf rts
0051BA 41ED04CE            lea     $4CE(A5),A0       vbl list
0051BE 2B480456            move.l  A0,$456(A5)       als Zeiger auf vblqueue
0051C2 303C0007            move.w  #7,D0             8 Vektoren
0051C6 4298                clr.l   (A0)+             Liste löschen
0051C8 51C8FFFC            dbra    D0,$51C6          nächsten Vektor
0051CC 61001D14            bsr     $6EE2             MFP initialisieren
0051D0 7002                moveq.l #2,D0             Bitnummer
0051D2 6100012A            bsr     $52FE             cartscan, Cartridge testen
0051D6 9BCD                sub.l   A5,A5             A5 löschen
0051D8 6100036A            bsr     $5544             wvbl, auf nächsten Bilrücklauf warten
0051DC 61000366            bsr     $5544             wvbl, auf nächsten Bilrücklauf warten
0051E0 103C0002            move.b  #2,D0             Default-Auflösung auf Monochrom
0051E4 08390007FFFFFA01    btst    #7,$FFFFFA01      MFP gpip monochrom detect testen
0051EC 670C                beq     $51FA             kein Monochrommonitor ?
0051EE 102D044A            move.b  $44A(A5),D0       defshiftmd holen
0051F2 B03C0002            cmp.b   #2,D0             Farbmodus ?
0051F6 6D02                blt     $51FA             ja
0051F8 4200                clr.b   D0                sonst Low Resolution wählen
0051FA 1B40044C            move.b  D0,$44C(A5)       sshiftmd merken
0051FE 13C0FFFF8260        move.b  D0,$FFFF8260      shiftmd, Shifter programmieren
005204 B03C0001            cmp.b   #1,D0             mittlere Auflösung ?
005208 660A                bne     $5214             nein
00520A 33F9FFFF825EFFFF8246 move.w  $FFFF825E,$FFFF8246 Farbe 15 (schwarz) nach Farbe 3 kopieren
005214 4EB90000F6C4        jsr     $F6C4             Bildschirmausgabe initialisieren
```

```
00521A 2B7C0000501E046E     move.l  #$501E,$46E(A5)     hdv dsb
005222 33FC000100000452     move.w  #1,$452             vblsem, VBL freigeben
00522A 4240                 clr.w   D0                  Bitnummer 0
00522C 610000D0             bsr     $52FE               cartscan, Cartridge testen
005230 46FC2300             move.w  #$2300,SR           IPL auf 3
005234 7001                 moveq.l #1,D0               Bitnummer 1
005236 610000C6             bsr     $52FE               cartscan, Cartridge testen
00523A 61004234             bsr     $9470               DOS initialisieren
00523E 610000B0             bsr     $52F0               dskboot
005242 4A7900000482         tst.w   $482                cmdload, shell von Diskette laden ?
005248 6718                 beq     $5262               nein
00524A 61003C64             bsr     $8EB0               Cursor einschalten
00524E 610006FE             bsr     $594E               auto exec
005252 487A0099             pea     $52ED(PC)           Nullname
005256 487A0095             pea     $52ED(PC)           Nullname
00525A 487A007E             pea     $52DA(PC)           'COMMAND.PRG'
00525E 4267                 clr.w   -(A7)
005260 605C                 bra     $52BE

005262 610006EA             bsr     $594E
005266 41FA0066             lea     $52CE(PC),A0
00526A 327C0502             move.w  #$502,A1
00526E 0C100023             cmp.b   #35,(A0)            '#' Drive-Kennung ?
005272 6602                 bne     $5276               nein
005274 2449                 move.l  A1,A2               Adresse der Drivebezeichnung
005276 12D8                 move.b  (A0)+,(A1)+         Drivenummer kopiren
005278 6AF4                 bpl     $526E               kompletten String kopieren
00527A 103900000446         move.b  $446,D0             bootdev
005280 D03C0041             add.b   #$41,D0             'A', Drivename berechnen
005284 1480                 move.b  D0,(A2)             in Filenamen einfügen
005286 487900000502         pea     $502                enviroment string
00528C 4879000052ED         pea     $52ED               Nullname
005292 487A0059             pea     $52ED(PC)           Nullname
005296 3F3C0005             move.w  #5,-(A7)
00529A 3F3C004B             move.w  #$4B,-(A7)          exec, Programm laden
00529E 4E41                 trap    #1                  GEMDOS Aufruf
0052A0 DEFC000E             add.w   #$E,A7              Stackpointer korrigieren
```

```
0052A4 2040                  move.l  D0,A0              Base Page Adresse merken
0052A6 2179000004FE0008      move.l  $4FE,8(A0)         exec os
0052AE 487900000502          pea     $502
0052B4 2F08                  move.l  A0,-(A7)
0052B6 487A0035              pea     $52ED(PC)
0052BA 3F3C0004              move.w  #4,-(A7)
0052BE 3F3C004B              move.w  #$4B,-(A7)         exec, Programm laden
0052C2 4E41                  trap    #1                 GEMDOS aufruf
0052C4 DEFC000E              add.w   #$E,A7             Stack korrigieren
0052C8 4EF90000501E          jmp     $501E              falls Rückkehr, dann Reset

0052CE 504154483D00          dc.b    'PATH=',0
0052D4 233A5C00FF            dc.b    '#:\',0,$FF
0052DA 434F4D4D414E442E5052  dc.b    'COMMAND.PRG',0
0052E4 4700
0052E6 47454D2E505247        dc.b    'GEM.PRG'

0052ED 000000                dc.b    0,0,0

*****************************************************  dskboot, von Diskette booten
0052F0 7003                  moveq.l #3,D0              Bitnummer 3
0052F2 610A                  bsr     $52FE              Cartridge testen
0052F4 20790000047A          move.l  $47A,A0            hdv boot, Bootvektor laden
0052FA 4E90                  jsr     (A0)               und Versuch starten
0052FC 4E75                  rts

*****************************************************  Cartridge testen
0052FE 41F900FA0000          lea     $FA0000,A0         cartbase
005304 0C98ABCDEF42          cmp.l   #$ABCDEF42,(A0)+   Anwender-Cartridge eingesetzt ?
00530A 661A                  bne     $5326              nein
00530C 01280004              btst    D0,4(A0)           Initialisierung ?
005310 670E                  beq     $5320              nein
005312 48E7FFFE              movem.l D0-D7/A0-A6,-(A7)  Register retten
005316 20680004              move.l  4(A0),A0           Adresse der Init-Routine
00531A 4E90                  jsr     (A0)               Initialisierung ausführen
00531C 4CDF7FFF              movem.l (A7)+,D0-D7/A0-A6  Register zurückholen
005320 4A90                  tst.l   (A0)               weiter Anwendung in Cartridge ?
```

```
005322 2050              move.l  (A0),A0          Link-Adresse holen
005324 66E6              bne     $530C            nächste Anwendung initialisieren
005326 4E75              rts

************************************************
005328 4E75              rts                      Default-Vektor

************************************************     Speichertest
00532A D1C1              add.l   D1,A0            Offset zu Basisadresse addieren
00532C 4240              clr.w   D0               Testmuster löschen
00532E 43E801F8          lea     $1F8(A0),A1      Endadresse des Tests
005332 B058              cmp.w   (A0)+,D0         Speicherinhalt testen
005334 6608              bne     $533E            ungleich, abbrechen
005336 D07CFA54          add.w   #$FA54,D0        nächstes Testmuster erzeugen
00533A B3C8              cmp.l   A0,A1            Endadresse ereeicht ?
00533C 66F4              bne     $5332            nein, weiter testen
00533E 4ED5              jmp     (A5)             zurück zur Aufrufadresse

************************************************
005340 9BCD              sub.l   A5,A5
005342 0CAD752019F30420  cmp.l   #1965038067,$420(A5)  memvalid
00534A 6608              bne     $5354
00534C 0CAD237698AA043A  cmp.l   #594974890,$43A(A5)   memval2
005354 4ED6              jmp     (A6)

************************************************
005356 00000000
00535A 00000000
00535E 00000000
005362 00000000

************************************************
005366 1839FFFF8260      move.b  $FFFF8260,D4     shiftmd, Videoauflösung holen
00536C C87C0003          and.w   #3,D4            Bits 0 und 1 isolieren
005370 D844              add.w   D4,D4            Wert für Worttabelle verdoppeln
005372 B1FC00008000      cmp.l   #$8000,A0
005378 6220              bhi     $539A
```

```
00537A B1FC00000000         cmp.l   #0,A0
005380 6604                 bne     $5386
005382 307B4086             move.w  $530A(PC,D4.w),A0
005386 4280                 clr.l   D0
005388 1039FFFF8201         move.b  $FFFF8201,D0          dbaseh
00538E E148                 lsl.w   #8,D0
005390 1039FFFF8203         move.b  $FFFF8203,D0          dbasel
005396 E188                 lsl.l   #8,D0
005398 D1C0                 add.l   D0,A0
00539A 7E0F                 moveq.l #15,D7
00539C 3C3B4078             move.w  $5416(PC,D4.w),D6
0053A0 3605                 move.w  D5,D3
0053A2 2448                 move.l  A0,A2
0053A4 D4FB406A             add.w   $5410(PC,D4.w),A2
0053A8 1011                 move.b  (A1),D0
0053AA 4BFA0004             lea     $53B0(PC),A5
0053AE 6042                 bra     $53F2

0053B0 3202                 move.w  D2,D1
0053B2 10290001             move.b  1(A1),D0
0053B6 4BFA0004             lea     $53BC(PC),A5
0053BA 6036                 bra     $53F2

0053BC 3011                 move.w  (A1),D0
0053BE 4EFB4002             jmp     $53C2(PC,D4.w)

0053C2 6004                 bra     $53C8             Low-Resolution

0053C4 600C                 bra     $53D2             Medium-Resolution

0053C6 6014                 bra     $53DC             High-Resolution

************************************************      Low-Resolution
0053C8 30C0                 move.w  D0,(A0)+
0053CA 30C0                 move.w  D0,(A0)+
0053CC 30C0                 move.w  D0,(A0)+
0053CE 30C0                 move.w  D0,(A0)+
```

```
0053D0 600E                bra     $53E0

*****************************************************     Medium-Resolution
0053D2 30C1                move.w  D1,(A0)+
0053D4 30C1                move.w  D1,(A0)+
0053D6 30C2                move.w  D2,(A0)+
0053D8 30C2                move.w  D2,(A0)+
0053DA 6004                bra     $53E0

*****************************************************     High-Resolution
0053DC 30C1                move.w  D1,(A0)+
0053DE 30C2                move.w  D2,(A0)+
0053E0 51CBFFC6            dbra    D3,$53A8
0053E4 204A                move.l  A2,A0
0053E6 51CEFFB8            dbra    D6,$53A0
0053EA 5449                addq.w  #2,A1
0053EC 51CFFFAE            dbra    D7,$539C
0053F0 4ED6                jmp     (A6)

0053F2 2643                move.l  D3,A3
0053F4 7400                moveq.l #0,D2
0053F6 7607                moveq.l #7,D3
0053F8 E310                roxl.b  #1,D0
0053FA 40E7                move.w  SR, -(A7)          Carry-Flag merken
0053FC E352                roxl.w  #1,D2
0053FE 46DF                move.w  (A7)+,SR           Carry-Flag zurückholen
005400 E352                roxl.w  #1,D2
005402 51CBFFF4            dbra    D3,$53F8
005406 260B                move.l  A3,D3
005408 4ED5                jmp     (A5)               zurück zum Aufruf

*****************************************************
00540A 3EC8                dc.w    16072
00540C 3EC8                dc.w    16072
00540E 3EA4                dc.w    16036
005410 00A0                dc.w    160
005412 00A0                dc.w    160
```

```
005414 0050                dc.w    80
005416 0000                dc.w    0
005418 0000                dc.w    0
00541A 0001                dc.w    0

*************************************************     Farbpalette
00541C 0777                dc.w    $777                weiß
00541E 0700                dc.w    $700                rot
005420 0070                dc.w    $070                grün
005422 0770                dc.w    $770                gelb
005424 0007                dc.w    $007                blau
005426 0707                dc.w    $707                magenta
005428 0077                dc.w    $077                blaugrün
00542A 0555                dc.w    $555                hellgrau
00542C 0333                dc.w    $333                grau
00542E 0733                dc.w    $733                hellrot
005430 0373                dc.w    $373                hellgrün
005432 0773                dc.w    $773                hellgelb
005434 0337                dc.w    $337                hellblau
005436 0737                dc.w    $737                hellmagenta
005438 0377                dc.w    $377                hellblaugrün
00543A 0000                dc.w    $000                schwarz

***********************************************     HBL-Interrupt
00543C 3F00                move.w  D0,-(A7)            D0 retten
00543E 302F0002            move.w  2(A7),D0            Status vom Stack holen
005442 C07C0700            and.w   #$700,D0            IPL-Maske isolieren
005446 6606                bne     $544E               ungleich null ?
005448 006F03000002        or.w    #$300,2(A7)         sonst IPL auf 3
00544E 301F                move.w  (A7)+,D0            D0 zurückholen
005450 4E73                rte

***********************************************     VBL-Interrupt
005452 52B900000466        addq.l  #1,$466             frclock erhöhen
005458 537900000452        subq.w  #1,$452             vblsem
00545E 6B0000DC            bmi     $553C               VBL-Interrupt disabled ?
005462 48E7FFFE            movem.l D0-D7/A0-A6,-(A7)   Register retten
```

```
005466 52B900000462     addq.l  #1,$462          vbclock erhöhen
00546C 9BCD             sub.l   A5,A5            A5 löschen
00546E 1039FFFF8260     move.b  $FFFF8260,D0     shiftmd laden
005474 C03C0003         and.b   #3,D0
005478 B03C0002         cmp.b   #2,D0            High Resolution ?
00547C 6C10             bge     $548E            ja
00547E 08390007FFFFFA01 btst    #7,$FFFFFA01     mfp gpip, monochrom detect
005486 662C             bne     $54B4            nein, Farbmonitor
005488 103C0002         move.b  #2,D0            High Resolution
00548C 6016             bra     $54A4

00548E 08390007FFFFFA01 btst    #7,$FFFFFA01     mfp gpip, monochrom detect
005496 671C             beq     $54B4            Monochrom-Monitor ?
005498 102D044A         move.b  $44A(A5),D0      defshiftmd, Farbauflösung holen
00549C B03C0002         cmp.b   #2,D0            Monochrom ?
0054A0 6D02             blt     $54A4            nein
0054A2 4200             clr.b   D0               sonst low resolution
0054A4 1B40044C         move.b  D0,$44C(A5)      sshiftmd merken
0054A8 13C0FFFF8260     move.b  D0,$FFFF8260     shiftmd, Shifter programmieren
0054AE 206D046E         move.l  $46E(A5),A0      Vektor für Auflösungsänderung
0054B2 4E90             jsr     (A0)             Routine ausführen
0054B4 61003B42         bsr     $8FF8            Cursor blinken
0054B8 9BCD             sub.l   A5,A5            A5 löschen
0054BA 4AAD045A         tst.l   $45A(A5)         colorptr, Farbpalette neu laden ?
0054BE 6718             beq     $54D8            nein
0054C0 206D045A         move.l  $45A(A5),A0      colorptr, Adresse der neuen Palette
0054C4 43F9FFFF8240     lea     $FFFF8240,A1     color0, Adresse im Videoshifter
0054CA 303C000F         move.w  #$F,D0           16 Farben
0054CE 32D8             move.w  (A0)+,(A1)+      kopieren
0054D0 51C8FFFC         dbra    D0,$54CE         nächste Farbe
0054D4 42AD045A         clr.l   $45A(A5)         colorptr wieder löschen
0054D8 4AAD045E         tst.l   $45E(A5)         screenpt, neue Videoadresse ?
0054DC 671A             beq     $54F8            nein
0054DE 2B6D045E044E     move.l  $45E(A5),$44E(A5) screenpt nach v bs ad kopieren
0054E4 202D044E         move.l  $44E(A5),D0      v bs ad
0054E8 E048             lsr.w   #8,D0            Bit 8-15
0054EA 13C0FFFF8203     move.b  D0,$FFFF8203     nach dbasel
```

```
0054F0 E048                lsr.w   #8,D0              Bit 16-24
0054F2 13C0FFFF8201        move.b  D0,$FFFF8201       nach dbaseh
0054F8 610011EA            bsr     $66E4              VBL-Routine für Floppies
0054FC 3E3900000454        move.w  $454,D7            nvbls, Anzahl der VBL-Routinen
005502 6720                beq     $5524              keine Vektoren ?
005504 5387                subq.l  #1,D7              in dbraZähler verwandeln
005506 207900000456        move.l  $456,A0            Adresse der vblqueue
00550C 2258                move.l  (A0)+,A1           Vektor holen
00550E B3FC00000000        cmp.l   #$0,A1             Null ?
005514 670A                beq     $5520              Routine nicht ausführen
005516 48E70180            movem.l D7/A0,-(A7)        Register retten
00551A 4E91                jsr     (A1)               Routine ausführen
00551C 4CDF0180            movem.l (A7)+,D7/A0        Register zurückholen
005520 51CFFFEA            dbra    D7,$550C           nächsten Vektor testen
005524 9BCD                sub.l   A5,A5              A5 löschen
005526 4A6D04EE            tst.w   $4EE(A5)           dumpflg, Hardcopy gewünscht ?
00552A 660C                bne     $5538              nein
00552C 61000532            bsr     $5A60              Hardcopy
005530 33FCFFFF000004EE    move.w  #$FFFF,$4EE        dumpflg wieder löschen
005538 4CDF7FFF            movem.l (A7)+,D0-D7/A0-A6  Register zurückholen
00553C 527900000452        addq.w  #1,$452            vblsem, VBL wieder freigeben
005542 4E73                rte

************************************************     wvbl, Warten auf nächsten V-Sync
005544 40E7                move.w  SR, -(A7)          Status retten
005546 027CF8FF            and.w   #$F8FF,SR          IPL 0, Interrupts erlauben
00554A 203900000466        move.l  $466,D0            frclock laden
005550 B0B900000466        cmp.l   $466,D0            frclock
005556 67F8                beq     $5550              noch gleich ?
005558 46DF                move.w  (A7)+,SR           Status zurückholen
00555A 4E75                rts

************************************************     critical error handler
00555C 2F3900000404        move.l  $404,-(A7)         etv critic
005562 70FF                moveq.l #-1,D0
005564 4E75                rts
```

```
*****************************************************   TRAP #14
005566 41FA0084            lea     $55EC(PC),A0       Adresse der TRAP #14 Routinen
00556A 6004                bra     $5570

*****************************************************   TRAP #13
00556C 41FA004C            lea     $55BA(PC),A0       Adresse der TRAP #13 Routinen
005570 2279000004A2        move.l  $4A2,A1            savptr, Zeiger auf save area
005576 301F                move.w  (A7)+,D0           Statusregister nach D0
005578 3300                move.w  D0,-(A1)           in save area retten
00557A 231F                move.l  (A7)+,-(A1)        PC retten
00557C 48E11F1F            movem.l D3-D7/A3-A7,-(A1)  und C-Register retten
005580 23C9000004A2        move.l  A1,$4A2            savptr updaten
005586 0800000D            btst    #13,D0             Aufruf aus Supervisor-Modus ?
00558A 6602                bne     $558E              ja
00558C 4E6F                move.l  USP,A7             sonst USP benutzen
00558E 301F                move.w  (A7)+,D0           Funktionsnummer vom Stack holen
005590 B058                cmp.w   (A0)+,D0           mit maximaler Nummer vergleichen
005592 6C10                bge     $55A4              zu groß, abbrechen
005594 E548                lsl.w   #2,D0              Nummer in Longzähler wandeln
005596 20300000            move.l  0(A0,D0.w),D0      Adresse der Routine holen
00559A 2040                move.l  D0,A0              nach A0
00559C 6A02                bpl     $55A0              Bit 31 gelöscht ?
00559E 2050                move.l  (A0),A0            sonst Adresse indirekt benutzen
0055A0 9BCD                sub.l   A5,A5              A5 löschen
0055A2 4E90                jsr     (A0)               Routine ausführen
0055A4 2279000004A2        move.l  $4A2,A1            savptr holen
0055AA 4CD9F8F8            movem.l (A1)+,D3-D7/A3-A7  Register zurückholen
0055AE 2F19                move.l  (A1)+,-(A7)        PC auf Stack
0055B0 3F19                move.w  (A1)+,-(A7)        Status auf Stack
0055B2 23C9000004A2        move.l  A1,$4A2            savptr updaten
0055B8 4E73                rte                        zurück zum Aufruf

*****************************************************   Adressen der TRAP#13 Routinen
0055BA 000C                dc.w    12                 Anzahl der Routinen
0055BC 0000572E            dc.l    $572E              0, getmpb
0055C0 00005694            dc.l    $5694              1, bconstat
0055C4 0000569A            dc.l    $569A              2, bconin
```

```
0055C8 000056A6            dc.l    $56A6               3, bconout
0055CC 80000476            dc.l    $476+$80000000      4, (indirekt) rwabs
0055D0 0000575A            dc.l    $575A               5, setexec
0055D4 00005772            dc.l    $5772               6, tickcal
0055D8 80000472            dc.l    $472+$80000000      7, (indirekt) getbpb
0055DC 000056A0            dc.l    $56A0               8, bcostat
0055E0 8000047E            dc.l    $47E+$80000000      9, (indirekt) mediach
0055E4 00005716            dc.l    $5716               10, drvmap
0055E8 0000571C            dc.l    $571C               11, shift

*************************************************      Adressen der TRAP#14 Routinen
0055EC 0028                dc.w    40                  Anzahl der Routinen
0055EE 00007AC0            dc.l    $7AC0               0, initmouse
0055F2 00005328            dc.l    $5328               1, rts
0055F6 0000577A            dc.l    $577A               2, physbase
0055FA 0000578E            dc.l    $578E               3, logbase
0055FE 00005794            dc.l    $5794               4, getrez
005602 000057A0            dc.l    $57A0               5, setscreen
005606 000057EE            dc.l    $57EE               6, setpalette
00560A 000057F6            dc.l    $57F6               7, setcolor
00560E 000062D2            dc.l    $62D2               8, floprd
005612 000063B0            dc.l    $63B0               9, flopwr
005616 00006468            dc.l    $6468               10, flopfmt
00561A 00005B64            dc.l    $5B64               11, getdsb
00561E 00006B74            dc.l    $6B74               12, midiws
005622 00007138            dc.l    $7138               13, mfpint
005626 00007440            dc.l    $7440               14, iorec
00562A 00007468            dc.l    $7468               15, rsconf
00562E 00007BC6            dc.l    $7BC6               16, keytrans
005632 00006062            dc.l    $6062               17, rand
005636 0000614A            dc.l    $614A               18, protobt
00563A 00006602            dc.l    $6602               19, flopver
00563E 00005A4E            dc.l    $5A4E               20, dumpit
005642 00009024            dc.l    $9024               21, cursconf
005646 00006AAA            dc.l    $6AAA               22, settime
00564A 00006A90            dc.l    $6A90               23, gettime
00564E 00007BF2            dc.l    $7BF2               24, bioskeys
```

```
005652 00006CE6              dc.l    $6CE6          25, ikbdws
005656 00007162              dc.l    $7162          26, jdisint
00565A 0000719C              dc.l    $719C          27, jenabint
00565E 00007A30              dc.l    $7A30          28, giaccess
005662 00007A9A              dc.l    $7A9A          29, offgibit
005666 00007A74              dc.l    $7A74          30, ongibit
00566A 00007B8A              dc.l    $7B8A          31, xbtimer
00566E 00007C0C              dc.l    $7C0C          32, dosound
005672 00007C20              dc.l    $7C20          33, setprt
005676 00007C54              dc.l    $7C54          34, ikbdvecs
00567A 00007C32              dc.l    $7C32          35, kbrate
00567E 00007D92              dc.l    $7D92          36, prtblk
005682 00005544              dc.l    $5544          37, wvbl
005686 0000568E              dc.l    $568E          38, supexec
00568A 0000581C              dc.l    $581C          39, puntaes

************************************************    supexec, Routine im Supervisormode
00568E 206F0004              move.l  4(A7),A0       Adresse vom Stack holen
005692 4ED0                  jmp     (A0)           Routine im Supervisor-Modus ausführen

************************************************    bconstat, Eingabestatus holen
005694 41FA0020              lea     $56B6(PC),A0   Status-Tabelle
005698 6010                  bra     $56AA

************************************************    bconin, Eingabe
00569A 41FA0032              lea     $56CE(PC),A0   Eingabe-Tabelle
00569E 600A                  bra     $56AA

************************************************    bcostat, Ausgabestatus holen
0056A0 41FA0044              lea     $56E6(PC),A0   Statustabelle
0056A4 6004                  bra     $56AA

************************************************    bconout, Ausgabe
0056A6 41FA0056              lea     $56FE(PC),A0
```

```
*****************************************************   conout
0056AA 302F0004            move.w  4(A7),D0           Gerätenummer holen
0056AE E548                lsl.w   #2,D0              mal 4
0056B0 20700000            move.l  0(A0,D0.w),A0      Adresse der Routine holen
0056B4 4ED0                jmp     (A0)               Routine ausführen

*****************************************************   Eingabestatus
0056B6 00005328            dc.l    $5328              rts
0056BA 00006C70            dc.l    $6C70              RS 232 Status
0056BE 00006CFA            dc.l    $6CFA              Consol Status
0056C2 00006B88            dc.l    $6B88              Midi Status
0056C6 00005328            dc.l    $5328              rts
0056CA 00005328            dc.l    $5328              rts

*****************************************************   Eingabe
0056CE 00006C3C            dc.l    $6C3C              Parallel Port
0056D2 00006C86            dc.l    $6C86              RS 232 Eingabe
0056D6 00006D10            dc.l    $6D10              Consol Eingabe
0056DA 00006BA4            dc.l    $6BA4              Midi Eingabe
0056DE 00005328            dc.l    $5328              rts
0056E2 00005328            dc.l    $5328              rts

*****************************************************   Ausgabestatus
0056E6 00006C5C            dc.l    $6C5C              Centronics Status
0056EA 00006C96            dc.l    $6C96              RS 232 Status
0056EE 00006D46            dc.l    $6D46              Consol Status
0056F2 00006CBA            dc.l    $6CBA              Keyboard Status
0056F6 00006B48            dc.l    $6B48              Midi Status
0056FA 00005328            dc.l    $5328              rts

*****************************************************   Ausgabe
0056FE 00006BD4            dc.l    $6BD4              Centronics Ausgabe
005702 00006CAE            dc.l    $6CAE              RS 232 Ausgabe
005706 00008ADE            dc.l    $8ADE              Consol Ausgabe
00570A 00006B5A            dc.l    $6B5A              Midi Ausgabe
00570E 00006CCC            dc.l    $6CCC              Keyboard Ausgabe
005712 00008AD2            dc.l    $8AD2              Ascii-Ausgabe
```

```
*****************************************************   drvmap, aktive Floppies holen
005716 202D04C2              move.l  $4C2(A5),D0      drvbits, Bitvektor holen
00571A 4E75                  rts

*****************************************************   shift, Tastaturstatus
00571C 7000                  moveq.l #0,D0
00571E 102D0A5D              move.b  $A5D(A5),D0      kshift, Shiftstatus holen
005722 322F0004              move.w  4(A7),D1         Parameter vom Stack holen
005726 6B04                  bmi     $572C            negativ, dann nicht setzen
005728 1B410A5D              move.b  D1,$A5D(A5)      als kshift übernehmen
00572C 4E75                  rts

*****************************************************   getmpb, Memory Parameter Block
00572E 206F0004              move.l  4(A7),A0         neuen MPB
005732 43ED048E              lea     $48E(A5),A1      themd, Memory descriptor
005736 2089                  move.l  A1,(A0)          mp mfl = Adresse des MD
005738 42A80004              clr.l   4(A0)            mp mal = Null
00573C 21490008              move.l  A1,8(A0)         mp rover = Adresse des MD
005740 4291                  clr.l   (A1)             m link löschen
005742 236D04320004          move.l  $432(A5),4(A1)   membot als mstart
005748 202D0436              move.l  $436(A5),D0      memtop
00574C 90AD0432              sub.l   $432(A5),D0      minus membot
005750 23400008              move.l  D0,8(A1)         Länge als m lenght
005754 42A9000C              clr.l   12(A1)           m own = Null
005758 4E75                  rts

*****************************************************   setexc, Exception Vektor setzen
00575A 302F0004              move.w  4(A7),D0         Vektor Nummer holen
00575E E548                  lsl.w   #2,D0            mal 4 gleich Adresse
005760 91C8                  sub.l   A0,A0            A0 löschen
005762 41F00000              lea     0(A0,D0.w),A0    Adresse des alten Vektors holen
005766 2010                  move.l  (A0),D0          alten Vektor nach D0
005768 222F0006              move.l  6(A7),D1         neuen Vektor
00576C 6B02                  bmi     $5770            negativ, dann nicht setzen
00576E 2081                  move.l  D1,(A0)          Vektor setzen
005770 4E75                  rts
```

```
*****************************************************   tickcal, Timerwert im Millisekunden
005772 4280                 clr.l   D0
005774 302D0442             move.w  $442(A5),D0         timr ms  holen
005778 4E75                 rts

*****************************************************   physbase, physikalische Videoadresse
00577A 7000                 moveq.l #0,D0
00577C 1039FFFF8201         move.b  $FFFF8201,D0        dbaseh
005782 E148                 lsl.w   #8,D0
005784 1039FFFF8203         move.b  $FFFF8203,D0        dbasel
00578A E188                 lsl.l   #8,D0               Videoadresse in D0
00578C 4E75                 rts

*****************************************************   logbase, logische Videoadresse
00578E 202D044E             move.l  $44E(A5),D0         v bs ad  holen
005792 4E75                 rts

*****************************************************   getrez, Videoauflösung holen
005794 7000                 moveq.l #0,D0
005796 102D8260             move.b  $FFFF8260(A5),D0    shiftmd laden
00579A C03C0003             and.b   #$3,D0              Bits 0 und 1 isolieren
00579E 4E75                 rts

*****************************************************   setscreen, Bildschirmadresse setzen
0057A0 4AAF0004             tst.l   4(A7)               logische Adresse
0057A4 6B06                 bmi     $57AC               negativ, nicht setzen
0057A6 2B6F0004044E         move.l  4(A7),$44E(A5)      v bs ad setzen
0057AC 4AAF0008             tst.l   8(A7)               pysikalische Adresse
0057B0 6B10                 bmi     $57C2               negativ, nicht setzen
0057B2 13EF0009FFFF8201     move.b  9(A7),$FFFF8201     dbaseh
0057BA 13EF000AFFFF8203     move.b  10(A7),$FFFF8203    dbasel
0057C2 4A6F000C             tst.w   12(A7)              Videoauflösung
0057C6 6B24                 bmi     $57EC               negativ, nicht setzen
0057C8 1B6F000D044C         move.b  13(A7),$44C(A5)     sshiftmd
0057CE 6100FD74             bsr     $5544               wvbl, VBL abwarten
0057D2 13ED044CFFFF8260     move.b  $44C(A5),$FFFF8260  sshiftmd nach shiftmd
0057DA 426D0452             clr.w   $452(A5)            vblsem, VBL disabled
```

```
0057DE 4EB90000F6C4          jsr     $F6C4              Bildschirmausgabe initialisieren
0057E4 33FC000100000452      move.w  #1,$452            vblsem, VBL wieder erlauben
0057EC 4E75                  rts

*****************************************************   setpalette, neue Farbpalette laden
0057EE 2B6F0004045A          move.l  4(A7),$45A(A5)     colorptr setzen (Ausführung in VBL)
0057F4 4E75                  rts

*****************************************************   setcolor, einzelne Farbe setzen
0057F6 322F0004              move.w  4(A7),D1           Farbnummer
0057FA D241                  add.w   D1,D1              mal 2
0057FC C27C001F              and.w   #$1F,D1            auf gültige Nummer beschränken
005800 41F9FFFF8240          lea     $FFFF8240,A0       Adresse color0
005806 30301000              move.w  0(A0,D1.w),D0      alte Farbe holen
00580A C07C0777              and.w   #$777,D0           nicht relevante Bits löschen
00580E 4A6F0006              tst.w   6(A7)              neuen Farbwert testen
005812 6B06                  bmi     $581A              negativ, nicht setzen
005814 31AF00061000          move.w  6(A7),0(A0,D1.w)   neue Farbe setzen
00581A 4E75                  rts

*****************************************************   puntaes, AES löschen und neu starten
00581C 207AF7F6              move.l  $5014(PC),A0       os magic
005820 0C9087654321          cmp.l   #$87654321,(A0)    schon da ?
005826 660E                  bne     $5836              nein, fertig
005828 B1F90000042E          cmp.l   $42E,A0            im ROM ?
00582E 6C06                  bge     $5836              ja, nichts zu machen
005830 4290                  clr.l   (A0)               magic löschen
005832 6000F7EA              bra     $501E              zu Reset

005836 4E75                  rts

*****************************************************   term, Abbruch des laufenden Programms
005838 6102                  bsr     $583C              PC auf Stack
00583A 4E71                  nop
00583C 23DF000003C4          move.l  (A7)+,$3C4         PC einschließlich Vektornummer retten
005842 48F9FFFF00000384      movem.l D0-D7/A0-A7,$384   Register retten
00584A 4E68                  move.l  USP,A0             USP
```

```
00584C 23C8000003C8          move.l  A0,$3C8             retten
005852 303C000F              move.w  #$F,D0              16 Worte
005856 41F9000003CC          lea     $3CC,A0             save area
00585C 224F                  move.l  A7,A1               SP nach A1 holen
00585E 30D9                  move.w  (A1)+,(A0)+         Wort vom Stack retten
005860 51C8FFFC              dbra    D0,$585E            nächstes Wort
005864 23FC1234567800000380  move.l  #$12345678,$380     magic für gerettete Register
00586E 7200                  moveq.l #0,D1
005870 1239000003C4          move.b  $3C4,D1             Vektornummer nach D1 holen
005876 5341                  subq.w  #1,D1               in dbra-Zähler wandeln
005878 6116                  bsr     $5890               ensprechende Anzahl 'mushrooms' ausgeben
00587A 23FC000005FC000004A2  move.l  #$5FC,$4A2          savptr für BIOS rücksetzen
005884 3F3C0001              move.w  #1,-(A7)            Return Code für Error
005888 42A7                  clr.l   -(A7)               terminate process
00588A 4E41                  trap    #1                  GEMDOS Aufruf
00588C 6000F790              bra     $501E               falls return, dann Reset

************************************************       'mushrooms' auf Bildschirm schreiben
005890 1E39FFFF8260          move.b  $FFFF8260,D7        shiftmd, Auflösung holen
005896 CE7C0003              and.w   #3,D7               signifikante Bits isolieren
00589A DE47                  add.w   D7,D7               mal 2 für Wortzugriff
00589C 4280                  clr.l   D0
00589E 1039FFFF8201          move.b  $FFFF8201,D0        dbaseh
0058A4 E148                  lsl.w   #8,D0
0058A6 1039FFFF8203          move.b  $FFFF8203,D0        dbasel
0058AC E188                  lsl.l   #8,D0               Bildschirmadresse
0058AE 2040                  move.l  D0,A0               nach A0
0058B0 D0FB702A              add.w   $58DC(PC,D7.w),A0   plus Offset für Bildschirmmitte
0058B4 43FA0038              lea     $58EE(PC),A1        Adresse des Bitmusters der 'mushrooms'
0058B8 3C3C000F              move.w  #$F,D6              16 Rasterzeilen
0058BC 3401                  move.w  D1,D2
0058BE 2448                  move.l  A0,A2               Zeiger auf Beginn der Zeile merken
0058C0 3A3B7020              move.w  $58E2(PC,D7.w),D5   Anzahl der Worte (Bildebenen)
0058C4                       move.w  (A1),(A0)+          eine Rasterzeile schreiben
0058C6 51CDFFFC              dbra    D5,$58C4            ein komplettes mushroom
0058CA 51CAFFF4              dbra    D2,$58C0            das nächste auf der selben Zeile
0058CE 5449                  addq.w  #2,A1               nächstes Wort des Bitmusters
```

```
0058D0 D4FB7016          add.w   $58E8(PC,D7.w),A2   nächste Zieladresse
0058D4 204A              move.l  A2,A0               Beginn der Zeile zurückholen
0058D6 51CEFFE4          dbra    D6,$58BC            nächste Rasterzeile
0058DA 4E75              rts

************************************************   Bildschirmmitte
0058DC 3E80              dc.w    100*160             low resolution
0058DE 3E80              dc.w    100*160             mid resolution
0058E0 3E80              dc.w    200*80              high resolution

************************************************   Anzahl der Bildebenen -1
0058E2 0003              dc.w    3                   low resolution
0058E4 0001              dc.w    1                   mid resolution
0058E6 0000              dc.w    0                   high resolution

************************************************   Zeilenlänge
0058E8 00A0              dc.w    160                 low resolution
0058EA 00A0              dc.w    160                 mid resolution
0058EC 0050              dc.w    80                  high resolution

************************************************   Bitmuster 'mushrooms'
0058EE 07C0              dc.w    %0000011111000000
0058F0 1FF0              dc.w    %0001111111110000
0058F2 3BF8              dc.w    %0011101111111000
0058F4 77F4              dc.w    %0111011111110100
0058F6 B7FA              dc.w    %1011011111111010
0058F8 BBFA              dc.w    %1011101111111010
0058FA DFF6              dc.w    %1101111111110110
0058FC 66FC              dc.w    %0110011011111100
0058FE 3288              dc.w    %0011001010001000
005900 0280              dc.w    %0000001010000000
005902 0440              dc.w    %0000010001000000
005904 0440              dc.w    %0000010001000000
005906 0540              dc.w    %0000010101000000
005908 0520              dc.w    %0000010100100000
00590A 0920              dc.w    %0000100100100000
00590C 0920              dc.w    %0000100100100000
```

```
00590E 1290              dc.w    %0001001010010000

*************************************************      fastcopy, Floppysektor kopieren
005910 206F0004          move.l  4(A7),A0               Quelladresse
005914 226F0008          move.l  8(A7),A1               Zieladresse
005918 303C003F          move.w  #$3F,D0                (63+1)*8 = 512 Bytes
00591C 12D8              move.b  (A0)+,(A1)+
00591E 12D8              move.b  (A0)+,(A1)+
005920 12D8              move.b  (A0)+,(A1)+
005922 12D8              move.b  (A0)+,(A1)+            jeweils 8 Bytes kopieren
005924 12D8              move.b  (A0)+,(A1)+
005926 12D8              move.b  (A0)+,(A1)+
005928 12D8              move.b  (A0)+,(A1)+
00592A 12D8              move.b  (A0)+,(A1)+
00592C 51C8FFEE          dbra    D0,$591C               nächste 8 Bytes
005930 4E75              rts

*************************************************      Harddisk Initialisierung
005932 2F390000046A      move.l  $46A,-(A7)             hdv init auf Stack
005938 4E75              rts                            Sprung auf Routine

*************************************************
00593A 5C4155544F5C      dc.b    '\AUTO\'
005940 2A2E50524700      dc.b    '*.PRG',0
005946 123456789ABCDEF0  dc.l    $12345678, $9ABCDEF0

*************************************************      auto, Starten und Ausführen eines Prgs
00594E 41FAFFEA          lea     $543 (PC),A0           Adresse Pfadnamen
005952 43FAFFEC          lea     $5940(PC),A1           Adresse Filenamen '*.PRG'
005956 23DF000005FC      move.l  (A7)+,$5FC             Rücksprungadresse
00595C 9BCD              sub.l   A5,A5                  A5 löschen
00595E 2B480600          move.l  A0,$600(A5)            Pfadnamen
005962 2B490604          move.l  A1,$604(A5)            Filenamen
005966 202D04C2          move.l  $4C2(A5),D0            drvbits, Vektor mit aktiven Drives
00596A 323900000446      move.w  $446,D1                bootdev
005970 0300              btst    D1,D0                  Drive aktiv ?
005972 6736              beq     $59AA                  nein, fertig
```

```
005974 41FAF977            lea     $52ED(PC),A0        Zeiger auf Nullnamen
005978 2F08                move.l  A0,-(A7)            Environment
00597A 2F08                move.l  A0,-(A7)            Command Tail
00597C 2F08                move.l  A0,-(A7)            Shellname
00597E 3F3C0005            move.w  #5,-(A7)            PSP erzeugen
005982 3F3C004B            move.w  #$4B,-(A7)          exec, Programm laden
005986 4E41                trap    #1                  GEMDOS Aufruf
005988 DEFC0010            add.w   #$10,A7             Stack korrigieren
00598C 2040                move.l  D0,A0               PSP
00598E 217C000059AC0008    move.l  #$59AC,8(A0)        PC für Auto Exec Programm
005996 2F0B                move.l  A3,-(A7)            Nullstring
005998 2F00                move.l  D0,-(A7)            PSP
00599A 2F0B                move.l  A3,-(A7)            Shellname
00599C 3F3C0004            move.w  #$4,-(A7)
0059A0 3F3C004B            move.w  #$4B,-(A7)          exec, Programm laden
0059A4 4E41                trap    #1                  GEMDOS Aufruf
0059A6 DEFC0010            add.w   #$10,A7             Stackpointer korrigieren
0059AA 4E75                rts

************************************************      Auto Exec Programm starten
0059AC 42A7                clr.l   -(A7)
0059AE 3F3C0020            move.w  #$20,-(A7)          super, im Supervisor-Modus gehen
0059B2 4E41                trap    #1                  GEMDOS Aufruf
0059B4 5C4F                addq.w  #6,A7               Stackpointer korrigieren
0059B6 2840                move.l  D0,A4               geretteter Stackpointer
0059B8 2A6F0004            move.l  4(A7),A5            Base Page Adresse
0059BC 4FED0100            lea     $100(A5),A7         Platz für Base page
0059C0 2F3C00000100        move.l  #$100,-(A7)         $100 Bytes
0059C6 2F0D                move.l  A5,-(A7)            Start des Speicherbereichs
0059C8 4267                clr.w   -(A7)
0059CA 3F3C004A            move.w  #$4A,-(A7)          setblock
0059CE 4E41                trap    #1                  GEMDOS Aufruf
0059D0 5C4F                addq.w  #6,A7               Stackpointer korrigieren
0059D2 4A40                tst.w   D0                  Fehler ?
0059D4 666A                bne     $5A40               ja
0059D6 3F3C0007            move.w  #7,-(A7)            R/O, Hidden und System Files
0059DA 2F3900000600        move.l  $600,-(A7)          Filename
```

```
0059E0 3F3C004E          move.w  #$4E,-(A7)      Search First
0059E4 7E08              moveq.l #8,D7           Bytes für Stackkorrektur
0059E6 487900000608      pea     $608            DMA-Adresse für DOS
0059EC 3F3C001A          move.w  #$1A,-(A7)      setdta
0059F0 4E41              trap    #1              GEMDOS-Aufruf
0059F2 5C4F              addq.w  #6,A7           Stackpointer korrigieren
0059F4 4E41              trap    #1              GEMDOS-Aufruf
0059F6 DEC7              add.w   D7,A7           Stackpointer korrigieren
0059F8 4A40              tst.w   D0              passendes File gefunden ?
0059FA 6644              bne     $5A40           nein
0059FC 207900000600      move.l  $600,A0         Pfadname
005A02 247900000604      move.l  $604,A2         Filename
005A08 43F900000634      lea     $634,A1         Auto-Name
005A0E 12D8              move.b  (A0)+,(A1)+     Pfadteil kopieren
005A10 B5C8              cmp.l   A0,A2
005A12 66FA              bne     $5A0E
005A14 41F900000626      lea     $626,A0         Name aus DMA-Puffer
005A1A 12D8              move.b  (A0)+,(A1)+     an Pfad anhängen
005A1C 66FC              bne     $5A1A
005A1E 487AF8CD          pea     $52ED(PC)       Nullname
005A22 487AF8C9          pea     $52ED(PC)       Nullname
005A26 487900000634      pea     $634            Filename
005A2C 4267              clr.w   -(A7)           Programm laden und starten
005A2E 3F3C004B          move.w  #$4B,-(A7)      exec, Programm laden
005A32 4E41              trap    #1              GEMDOS-Aufruf
005A34 DEFC0010          add.w   #$10,A7         Stackpointer korrigieren
005A38 7E02              moveq.l #2,D7           Bytes für Stackkorrektur
005A3A 3F3C004F          move.w  #$4F,-(A7)      Search Next
005A3E 60A6              bra     $59E6           nächstes File

005A40 4FF900003E2A      lea     $3E2A,A7        Stackpointer wieder auf Startwert
005A46 2F39000005FC      move.l  $5FC,-(A7)      Rücksprungadresse laden
005A4C 4E75              rts
```

```
************************************************      dumpit, Bildschirm-Hardcopy
005A4E 4279000004EE         clr.w   $4EE              dumpflg setzen
005A54 610A                 bsr     $5A60             Hardcopy
005A56 33FCFFFF000004EE     move.w  #-1,$4EE          dumpflg löschen
005A5E 4E75                 rts

************************************************      scrdmp, Hardcopy
005A60 9BCD                 sub.l   A5,A5             A5 löschen
005A62 2B6D044E0654         move.l  $44E(A5),$654(A5) v bs ad, Bildschirmadresse
005A68 426D0658             clr.w   $658(A5)          Offset auf Null
005A6C 4240                 clr.w   D0
005A6E 102D044C             move.b  $44C(A5),D0       sshiftmd, Bildschirmauflösung
005A72 3B400662             move.w  D0,$662(A5)       merken
005A76 D040                 add.w   D0,D0             mal 2
005A78 41FA005A             lea     $5AD4(PC),A0      Tabelle für Bildschirmauflösung
005A7C 3B700000065A         move.w  0(A0,D0.w),$65A(A5) Bildschirmbreite holen
005A82 3B700006065C         move.w  6(A0,D0.w),$65C(A5) Bildschirmhöhe holen
005A88 426D065E             clr.w   $65E(A5)          links
005A8C 426D0660             clr.w   $660(A5)          und rechts auf Null
005A90 2B7C00FF82400666     move.l  #$FF8240,$666(A5) Adresse der Farbpalette
005A98 426D066E             clr.w   $66E(A5)          Maskenzeiger löschen
005A9C 322D0A8C             move.w  $A8C(A5),D1       Druckerkonfiguration holen
005AA0 E649                 lsr.w   #3,D1             Test / Qualtitäts-Mode
005AA2 C27C0001             and.w   #1,D1             Bit isolieren
005AA6 3B410664             move.w  D1,$664(A5)       und merken
005AAA 322D0A8C             move.w  $A8C(A5),D1       Druckerkonfiguration holen
005AAE 3001                 move.w  D1,D0
005AB0 E848                 lsr.w   #4,D0             Parallel / Seriell
005AB2 C07C0001             and.w   #1,D0             Bit isolieren
005AB6 3B40066C             move.w  D0,$66C(A5)       und für Hardcopy merken
005ABA C27C0007             and.w   #7,D1             Druckertyp isolieren
005ABE 103B1020             move.b  $5AE0(PC,D1.w),D0 Zuordnung aus Tabelle holen
005AC2 33C00000066A         move.w  D0,$66A           und für Hardcopy merken
005AC8 486D0654             pea     $654(A5)          Adresse des Parameterblocks
005ACC 610022C4             bsr     $7D92             Hardcopy ausführen
005AD0 584F                 addq.w  #4,A7             Stackpointer korrigieren
005AD2 4E75                 rts
```

```
*******************************************
005AD4 014002800280          dc.w    320,640,640     Parametertabelle fur Hardcopy
005AD8 00C800C80190          dc.w    200,200,400     Bildschirmbreiten
                                                     Bildschirmhöhen

*******************************************
005AE0 00                    dc.b    0               Druckertypen (-1 = nicht implementiert)
005AE1 02                    dc.b    2               ATARI S/W Matrix
005AE2 01                    dc.b    1               ATARI S/W Typenrad
005AE3 FF                    dc.b    -1              ATARI Farb Matrix
005AE4 03                    dc.b    3               (ATARI Farb Typenrad ?)
005AE5 FF                    dc.b    -1              Epson S/W Matrix
005AE6 FF                    dc.b    -1              (Epson S/W Typenrad)
005AE7 FF                    dc.b    -1              (Epson Farb Matrix)
                                                     (Epson Farb Typenrad)

*******************************************
005AE8 4E56FFF0              link    A6,#-16         hdv init
005AEC 23FC0000012C000025F6 move.l  #300,$25F6      maxacctim auf 300*20 ms
005AF6 4240                  clr.w   D0
005AF8 33C0000004A6          move.w  D0,$4A6         nflops
005AFE 33C000004692          move.w  D0,$4692
005B04 3D40FFFE              move.w  D0,-2(A6)       mit Drive A beginnen
005B08 604E                  bra     $5B58
005B0A 207C00003E2A          move.l  #$3E2A,A0       Adresse des dsb
005B10 326EFFFE              move.w  -2(A6),A1       Drivenummer
005B14 D1C9                  add.l   A1,A0
005B16 4210                  clr.b   (A0)
005B18 4257                  clr.w   (A7)
005B1A 4267                  clr.w   -(A7)
005B1C 4267                  clr.w   -(A7)
005B1E 3F2EFFFE              move.w  -2(A6),-(A7)    Drive-Nummer
005B22 42A7                  clr.l   -(A7)
005B24 42A7                  clr.l   -(A7)
005B26 4EB90000628C          jsr     $628C           flopini
005B2C DFFC0000000E          add.l   #$E,A7          Stackpointer korrigieren
005B32 3F00                  move.w  D0,-(A7)        Fehlercode merken
005B34 306EFFFE              move.w  -2(A6),A0       Drive-Nummer
005B38 D1C8                  add.l   A0,A0
```

```
005B3A D1FC00004910          add.l   #$4910,A0
005B40 309F                  move.w  (A7)+,(A0)          Fehlercode
005B42 6610                  bne     $5B54               Drive nicht vorhanden ?
005B44 5279000004A6          addq.w  #1,$4A6             nflops erhöhen
005B4A 00B900000003000004C2  or.l    #3,$4C2             drvbits
005B54 526EFFFE              addq.w  #1,-2(A6)           Drive-Nummer erhöhen
005B58 0C6E0002FFFE          cmp.w   #2,-2(A6)           noch nicht 2
005B5E 6DAA                  blt     $5B0A               nächstes Drive initialisieren
005B60 4E5E                  unlk    A6
005B62 4E75                  rts

************************************************        getdsb
005B64 4E56FFFC              link    A6,#-4
005B68 4280                  clr.l   D0                  Null
005B6A 4E5E                  unlk    A6
005B6C 4E75                  rts

************************************************        getbpb, Get BIOS Parameterblock
005B6E 4E56FFF4              link    A6,#-12
005B72 48E7070C              movem.l D5-D7/A4-A5,-(A7)   Register retten
005B76 0C6E00020008          cmp.w   #2,8(A6)            Drive-Nummer
005B7C 6D06                  blt     $5B84               < 2, ok
005B7E 4280                  clr.l   D0                  sonst Null
005B80 60000192              bra     $5D14

005B84 302E0008              move.w  8(A6),D0            Drive-Nummer
005B88 EB40                  asl.w   #5,D0               mal 32
005B8A 48C0                  ext.l   D0
005B8C 2A40                  move.l  D0,A5
005B8E DBFC00003E3E          add.l   #$3E3E,A5           plus Basis-Adresse
005B94 284D                  move.l  A5,A4
005B96 3EBC0001              move.w  #1,(A7)             count, einen Sektor lesen
005B9A 4267                  clr.w   -(A7)               Seite 0
005B9C 4267                  clr.w   -(A7)               Track 0
005B9E 3F3C0001              move.w  #1,-(A7)            Sektor 1
005BA2 3F2E0008              move.w  8(A6),-(A7)         Drive-Nummer
005BA6 42A7                  clr.l   -(A7)               Filler
```

```
005BA8 2F3C000012BC      move.l  #$12BC,-(A7)      Sektor-Puffer
005BAE 4EB9000062D2      jsr     $62D2             Sektor lesen
005BB4 DFFC00000010      add.l   #$10,A7           Stackpointer korrigieren
005BBA 2D40FFF4          move.l  D0,-12(A6)        Fehlercode merken
005BBE 4AAEFFF4          tst.l   -12(A6)           und testen
005BC2 6C16              bge     $5BDA             ok ?
005BC4 3EAE0008          move.w  8(A6),(A7)        Drive-Nummer
005BC8 202EFFF4          move.l  -12(A6),D0        Fehlercode
005BCC 3F00              move.w  D0,-(A7)
005BCE 4EB90000555C      jsr     $555C             critical error handler
005BD4 548F              addq.l  #2,A7             Stackpointer korrigieren
005BD6 2D40FFF4          move.l  D0,-12(A6)        Fehlercode merken
005BDA 202EFFF4          move.l  -12(A6),D0
005BDE B0BC00010000      cmp.l   #$10000,D0
005BE4 67B0              beq     $5B96             Bootsektor nochmal lesen
005BE6 4AAEFFF4          tst.l   -12(A6)
005BEA 6C06              bge     $5BF2             ok ?
005BEC 4280              clr.l   D0
005BEE 60000124          bra     $5D14

005BF2 2EBC000012C7      move.l  #$12C7,(A7)       Puffer+11, Bytes per Sektor
005BF8 6100066C          bsr     $6266             u2i, 8086-Integer in 68000-Integer wandeln
005BFC 3E00              move.w  D0,D7             Bytes per Sektor merken
005BFE 670E              beq     $5C0E
005C00 1C39000012C9      move.b  $12C9,D6          Puffer+13, Sektoren pro Cluster
005C06 4886              ext.w   D6
005C08 CC7C00FF          and.w   #$FF,D6
005C0C 6606              bne     $5C14
005C0E 4280              clr.l   D0
005C10 60000102          bra     $5D14
005C14 3887              move.w  D7,(A4)           recsize
005C16 39460002          move.w  D6,2(A4)          clsiz
005C1A 2EBC000012D2      move.l  #$12D2,(A7)       Puffer+22, Sektoren per FAT
005C20 61000644          bsr     $6266             u2i, 8086-Integer in 68000-Integer wandeln
005C24 39400008          move.w  D0,8(A4)          fsiz
005C28 302C0008          move.w  8(A4),D0
005C2C 5240              addq.w  #1,D0             plus 1
```

```
005C2E 3940000A          move.w  D0,10(A4)       fatrec
005C32 3014              move.w  (A4),D0         recsize
005C34 C1EC0002          muls.w  2(A4),D0        mal clsiz
005C38 39400004          move.w  D0,4(A4)        ergibt clsizb
005C3C 2EBC000012CD      move.l  #$12CD,(A7)     Puffer+17, Anzahl Directory-Einträge
005C42 61000622          bsr     $6266           u2i, 8086-Integer in 68000-Integer wandeln
005C46 EB40              asl.w   #5,D0           mal 32
005C48 48C0              ext.l   D0
005C4A 81D4              divs.w  (A4),D0         durch recsiz
005C4C 39400006          move.w  D0,6(A4)        ergibt rdlen
005C50 302C000A          move.w  10(A4),D0       fatrec
005C54 D06C0006          add.w   6(A4),D0        plus rdlen
005C58 D06C0008          add.w   8(A4),D0        plus fsiz
005C5C 3940000C          move.w  D0,12(A4)       ergibt datrec
005C60 2EBC000012CF      move.l  #$12CF,(A7)     Puffer+19, Anzahl Sektoren
005C66 610005FE          bsr     $6266           u2i, 8086-Integer in 68000-Integer wandeln
005C6A 906C000C          sub.w   12(A4),D0       minus datrec
005C6E 48C0              ext.l   D0
005C70 81EC0002          divs.w  2(A4),D0        durch clsiz
005C74 3940000E          move.w  D0,14(A4)       ergibt numcl
005C78 2EBC000012D6      move.l  #$12D6,(A7)     Puffer+26, Anzahl Seiten
005C7E 610005E6          bsr     $6266           u2i, 8086-Integer in 68000-Integer wandeln
005C82 3B400014          move.w  D0,20(A5)       dnsides
005C86 2EBC000012D4      move.l  #$12D4,(A7)     Puffer+24, Sektoren pro Track
005C8C 610005D8          bsr     $6266           u2i, 8086-Integer in 68000-Integer wandeln
005C90 3B400018          move.w  D0,24(A5)       dspt
005C94 302D0014          move.w  20(A5),D0       dnsides
005C98 C1ED0018          muls.w  24(A5),D0       mal dspt
005C9C 3B400016          move.w  D0,22(A5)       ergibt dspc
005CA0 2EBC000012D8      move.l  #$12D8,(A7)     Puffer+28, Anzahl versteckter Sektoren
005CA6 610005BE          bsr     $6266           u2i, 8086-Integer in 68000-Integer wandeln
005CAA 3B40001A          move.w  D0,26(A5)       dhidden
005CAE 2EBC000012CF      move.l  #$12CF,(A7)     Puffer+19, Anzahl Sektoren auf Diskette
005CB4 610005B0          bsr     $6266           u2i, 8086-Integer in 68000-Integer wandeln
005CB8 48C0              ext.l   D0
005CBA 81ED0016          divs.w  22(A5),D0       durch dspc
005CBE 3B400012          move.w  D0,18(A5)       ergibt dntracks
```

```
005CC2 4247              clr.w   D7                Zähler auf Null
005CC4 6016              bra     $5CDC             Sprung ans Schleifenende
005CC6 204D              move.l  A5,A0             Pufferzeiger
005CC8 3247              move.w  D7,A1             Schleifenzähler
005CCA D1C9              add.l   A1,A0             plus BPB-Adresse
005CCC 3247              move.w  D7,A1             Schleifenzähler
005CCE D3FC000012BC      add.l   #$12BC,A1         plus Pufferadresse
005CD4 11690008001C      move.b  8(A1),28(A0)      Byte der Serialnummer kopieren
005CDA 5247              addq.w  #1,D7             nächstes Byte
005CDC BE7C0003          cmp.w   #3,D7             schon drei Bytes ?
005CE0 6DE4              blt     $5CC6             nein
005CE2 207C00000676      move.l  #$676,A0          cdev
005CE8 326E0008          move.w  8(A6),A1
005CEC D1C9              add.l   A1,A0
005CEE 227C00000674      move.l  #$674,A1          wpstatus
005CF4 346E0008          move.w  8(A6),A2
005CF8 D3CA              add.l   A2,A1
005CFA 1091              move.b  (A1),(A0)
005CFC 6704              beq     $5D02
005CFE 7001              moveq.l #1,D0             Diskettenstatus unsicher
005D00 6002              bra     $5D04
005D02 4240              clr.w   D0                Status sicher
005D04 227C00003E2A      move.l  #$3E2A,A1
005D0A 346E0008          move.w  8(A6),A2
005D0E D3CA              add.l   A2,A1
005D10 1280              move.b  D0,(A1)
005D12 200D              move.l  A5,D0             Adresse des BPB als Ergebnis
005D14 4A9F              tst.l   (A7)+
005D16 4CDF30C0          movem.l (A7)+,D6-D7/A4-A5 Register zurückholen
005D1A 4E5E              unlk    A6
005D1C 4E75              rts

*****************************************************  mediach
005D1E 4E560000          link    A6,#0
005D22 48E70304          movem.l D6-D7/A5,-(A7)    Register retten
005D26 0C6E00020008      cmp.w   #2,8(A6)          Drive-Nummer
005D2C 6D04              blt     $5D32             < 2?
```

```
005D2E 70F1              moveq.l #-15,D0          'unknown device'
005D30 604C              bra     $5D7E            Fehlerausprung
005D32 3E2E0008          move.w  8(A6),D7         Drive-Nummer
005D36 3A47              move.w  D7,A5
005D38 DBFC00003E2A      add.l   #$3E2A,A5
005D3E 0C150002          cmp.b   #2,(A5)
005D42 6604              bne     $5D48
005D44 7002              moveq.l #2,D0
005D46 6036              bra     $5D7E
005D48 207C00000676      move.l  #$676,A0         wplatch
005D4E 4A307000          tst.b   0(A0,D7.w)
005D52 6704              beq     $5D58
005D54 1ABC0001          move.b  #1,(A5)
005D58 2039000004BA      move.l  $4BA,D0          hz 200
005D5E 3247              move.w  D7,A1
005D60 D3C9              add.l   A1,A1
005D62 D3C9              add.l   A1,A1
005D64 D3FC00000678      add.l   #$678,A1         acctim
005D6A 2211              move.l  (A1),D1
005D6C 9081              sub.l   D1,D0
005D6E B0B9000025F6      cmp.l   $25F6,D0         maxacctim
005D74 6C04              bge     $5D7A
005D76 4240              clr.w   D0
005D78 6004              bra     $5D7E
005D7A 1015              move.b  (A5),D0
005D7C 4880              ext.w   D0
005D7E 4A9F              tst.l   (A7)+
005D80 4CDF2080          movem.l (A7)+,D7/A5      Register zurückholen
005D84 4E5E              unlk    A6
005D86 4E75              rts

************************************************   rwabs, Sektor(en) lesen/schreiben
005D88 4E56FFFC          link    A6,#-4
005D8C 48E70F04          movem.l D4-D7/A5,-(A7)   Register retten
005D90 0C6E00020012      cmp.w   #2,18(A6)        Drive-Nummer
005D96 6D06              blt     $5D9E            < 2 ?
005D98 70F1              moveq.l #-15,D0          nein, unknown device
```

```
005D9A 6000010E          bra     $5EAA           Fehleraussprung
005D9E 3C2E0012          move.w  18(A6),D6       Drivenummer merken
005DA2 0C6E00020008      cmp.w   #2,8(A6)        rwflag
005DA8 6C0000CE          bge     $5E78
005DAC 3006              move.w  D6,D0
005DAE EB40              asl.w   #5,D0
005DB0 48C0              ext.l   D0
005DB2 2A40              move.l  D0,A5
005DB4 DBFC00003E3E      add.l   #$3E3E,A5
005DBA 3E86              move.w  D6,(A7)
005DBC 6100FF60          bsr     $5D1E           media change testen
005DC0 3E00              move.w  D0,D7           Status merken
005DC2 BE7C0002          cmp.w   #2,D7           Diskette gewechselt ?
005DC6 660A              bne     $5DD2           nein
005DC8 70F2              moveq.l #-14,D0         media change error
005DCA 600000DE          bra     $5EAA           Fehlerausprung

005DCE 600000A8          bra     $5E78

005DD2 BE7C0001          cmp.w   #1,D7           Diskette evtl. gewechselt ?
005DD6 660000A0          bne     $5E78           nein
005DDA 3EBC0001          move.w  #1,(A7)         ein Sektor lesen (Bootsektor)
005DDE 4267              clr.w   -(A7)           Seite 0
005DE0 4267              clr.w   -(A7)           Track 0
005DE2 3F3C0001          move.w  #1,-(A7)        Sektor 1
005DE6 3F06              move.w  D6,-(A7)        Drive-Nummer
005DE8 42A7              clr.l   -(A7)           Filler
005DEA 2F3C000012BC      move.l  #$12BC,-(A7)    Sektorpuffer
005DF0 4EB9000062D2      jsr     $62D2           floprd
005DF6 DFFC00000010      add.l   #$10,A7         Stackpointer koorigieren
005DFC 2D40FFFC          move.l  D0,-4(A6)       Fehlernummer merken
005E00 4AAEFFFC          tst.l   -4(A6)          und testen
005E04 6C14              bge     $5E1A           ok ?
005E06 3E86              move.w  D6,(A7)
005E08 202EFFFC          move.l  -4(A6),D0       Fehler
005E0C 3F00              move.w  D0,-(A7)
005E0E 4EB90000555C      jsr     $555C           an critical error handler übergeben
```

```
005E14 548F              addq.l  #2,A7           Stackpointer korrigieren
005E16 2D40FFFC          move.l  D0,-4(A6)
005E1A 202EFFFC          move.l  -4(A6),D0
005E1E B0BC00010000      cmp.l   #$10000,D0
005E24 67B4              beq     $5DDA           nochmal lesen
005E26 4AAEFFFC          tst.l   -4(A6)          Fehlernummer
005E2A 6C08              bge     $5E34           ok ?
005E2C 202EFFFC          move.l  -4(A6),D0       Fehlernummer
005E30 60000078          bra     $5EAA           Fehleraussprung

005E34 4247              clr.w   D7              media change Status löschen
005E36 601C              bra     $5E54

005E38 207C000012BC      move.l  #$12BC,A0       Adresse des Sektorpuffers
005E3E 10307008          move.b  8(A0,D7.w),D0   Serialnummer
005E42 4880              ext.w   D0
005E44 1235701C          move.b  28(A5,D7.w),D1  mit vorherigem Wert
005E48 4881              ext.w   D1
005E4A B041              cmp.w   D1,D0           vergleichen
005E4C 6704              beq     $5E52           ok ?
005E4E 70F2              moveq.l #-14,D0         media change
005E50 6058              bra     $5EAA           Fehleraussprung

005E52 5247              addq.w  #1,D7           nächstes Byte der Serialnummer
005E54 BE7C0003          cmp.w   #3,D7           schon alle 3 Bytes getestet ?
005E58 6DDE              blt     $5E38           nein
005E5A 3046              move.w  D6,A0           Drive Nummer
005E5C D1FC00000676      add.l   #$676,A0        wplatch
005E62 3246              move.w  D6,A1           Drive Nummer
005E64 D3FC00000674      add.l   #$674,A1        wpstatus
005E6A 1091              move.b  (A1),(A0)
005E6C 660A              bne     $5E78
005E6E 3046              move.w  D6,A0
005E70 D1FC00003E2A      add.l   #$3E2A,A0
005E76 4210              clr.b   (A0)
005E78 4A79000004A6      tst.w   $4A6            nflops
005E7E 6604              bne     $5E84
```

```
005E80 70FE              moveq.l #-2,D0          drive not ready
005E82 6026              bra     $5EAA           Fehleraussprung

005E84 0C6E00010008      cmp.w   #1,8(A6)        Drive-Nummer
005E8A 6F04              ble     $5E90
005E8C 556E0008          subq.w  #2,8(A6)        Drive-Nummer
005E90 3EAE000E          move.w  14(A6),(A7)     count
005E94 3F06              move.w  D6,-(A7)        Drive-Nummer
005E96 3F2E0010          move.w  16(A6),-(A7)    recno
005E9A 2F2E000A          move.l  10(A6),-(A7)    buffer
005E9E 3F2E0008          move.w  8(A6),-(A7)     rwflag
005EA2 6110              bsr     $5EB4           floprw
005EA4 DFFC0000000A      add.l   #$A,A7          Stackpointer korrigieren
005EAA 4A9F              tst.l   (A7)+
005EAC 4CDF20E0          movem.l (A7)+,D5-D7/A5  Register zurückholen
005EB0 4E5E              unlk    A6
005EB2 4E75              rts

*************************************************       floprw, Sektor(en) lesen/schreiben
005EB4 4E56FFFA          link    A6,#-6
005EB8 48E73F04          movem.l D2-D7/A5,-(A7)  Register retten
005EBC 302E0010          move.w  16(A6),D0       Drivenummer
005EC0 EB40              asl.w   #5,D0           mal 32
005EC2 48C0              ext.l   D0
005EC4 2A40              move.l  D0,A5
005EC6 DBFC00003E3E      add.l   #$3E3E,A5       plus Basisadresse BPB
005ECC 082E0000000D      btst    #0,13(A6)       Pufferadresse ungerade ?
005ED2 6604              bne     $5ED8           ja
005ED4 4240              clr.w   D0              oddflag löschen
005ED6 6002              bra     $5EDA
005ED8 7001              moveq.l #1,D0           oddflag setzen
005EDA 3D40FFFE          move.w  D0,-2(A6)       und merken
005EDE 4A6D0016          tst.w   22(A5)          dspc gesetzt ?
005EE2 660A              bne     $5EEE           ja
005EE4 7009              moveq.l #9,D0           sonst 9 annehmen
005EE6 3B400016          move.w  D0,22(A5)       als dspt
005EEA 3B400018          move.w  D0,24(A5)       und dspc
```

```
005EEE 6000015E          bra     $604E

005EF2 4A6EFFFE          tst.w   -2(A6)          oddflag gesetzt ?
005EF6 6708              beq     $5F00           nein
005EF8 203C000012BC      move.l  #$12BC,D0       Sektor-Puffer
005EFE 6004              bra     $5F04

005F00 202E000A          move.l  10(A6),D0       Pufferadresse holen
005F04 2D40FFFA          move.l  D0,-6(A6)       und merken
005F08 3C2E000E          move.w  14(A6),D6       recno, logische Sektornummer
005F0C 48C6              ext.l   D6
005F0E 8DED0016          divs.w  22(A5),D6       durch dspc ergibt Tracknummer
ck5F12 382E000E          move.w  14(A6),D4       recno, logische Sektornummer
005F16 48C4              ext.l   D4
005F18 89ED0016          divs.w  22(A5),D4       durch dspc, Sektoren pro Track
005F1C 4844              swap    D4              Rest der Division als Sektornummer
005F1E B86D0018          cmp.w   24(A5),D4       mit dspt vergleichen
005F22 6C04              bge     $5F28           größer gleich ?
005F24 4245              clr.w   D5              Seite 0
005F26 6006              bra     $5F2E
005F28 7A01              moveq.l #1,D5           Seite 1
005F2A 986D0018          sub.w   24(A5),D4       dspt abziehen
005F2E 4A6EFFFE          tst.w   -2(A6)          oddflag gesetzt ?
005F32 6704              beq     $5F38           nein
005F34 7601              moveq.l #1,D3           Zähler auf 1 setzen
005F36 6018              bra     $5F50
005F38 302D0018          move.w  24(A5),D0       dspt
005F3C 9044              sub.w   D4,D0           minus Sektornummer
005F3E B06E0012          cmp.w   18(A6),D0       mit Anzahl Sektoren vergleichen
005F42 6C08              bge     $5F4C           größer gleich ?
005F44 362D0018          move.w  24(A5),D3       dspt
005F48 9644              sub.w   D4,D3           minus Sektornummer gleich Zähler
005F4A 6004              bra     $5F50
005F4C 362E0012          move.w  18(A6),D3       Anzahl Sektoren als Zähler
005F50 5244              addq.w  #1,D4           Sektornummer erhöhen (erste Nr. ist 1)
005F52 4A6E0008          tst.w   8(A6)           read-write-Flag testen
005F56 67000080          beq     $5FD8           read
```

```
005F5A 202EFFFA          move.l  -6(A6),D0       Pufferzeiger
005F5E B0AE000A          cmp.l   10(A6),D0       gleich angegebene Pufferadresse
005F62 6710              beq     $5F74           ja
005F64 2EAEFFFA          move.l  -6(A6),(A7)     Quelladresse
005F68 2F2E000A          move.l  10(A6),-(A7)    Zieladresse
005F6C 4EB900005910      jsr     $5910           fastcpy, einen Sektor kopieren
005F72 588F              addq.l  #4,A7           Stackpointer korrigieren
005F74 3E83              move.w  D3,(A7)         Anzahl Sektoren
005F76 3F05              move.w  D5,-(A7)        Seite
005F78 3F06              move.w  D6,-(A7)        Track
005F7A 3F04              move.w  D4,-(A7)        Sektor
005F7C 3F2E0010          move.w  16(A6),-(A7)    Drive
005F80 42A7              clr.l   -(A7)           Filler
005F82 2F2EFFFA          move.l  -6(A6),-(A7)    Sektor-Puffer
005F86 4EB9000063B0      jsr     $63B0           flopwr, Sektor lesen
005F8C DFFC00000010      add.l   #$10,A7         Stackpointer korrigieren
005F92 2E00              move.l  D0,D7           Fehler-Kode
005F94 4A87              tst.l   D7              ok ?
005F96 663E              bne     $5FD6           nein
005F98 4A7900000444      tst.w   $444            fverify, Verify verlangt ?
005F9E 6736              beq     $5FD6           nein
005FA0 3E83              move.w  D3,(A7)         Anzahl Sektoren
005FA2 3F05              move.w  D5,-(A7)        Seite
005FA4 3F06              move.w  D6,-(A7)        Track
005FA6 3F04              move.w  D4,-(A7)        Sektor
005FA8 3F2E0010          move.w  16(A6),-(A7)    Drive
005FAC 42A7              clr.l   -(A7)           Filler
005FAE 2F3C000012BC      move.l  #$12BC,-(A7)    Sektor-Puffer
005FB4 4EB900006602      jsr     $6602           flopver, Sektor verifizieren
005FBA DFFC00000010      add.l   #$10,A7         Stackpointer korrigieren
005FC0 2E00              move.l  D0,D7           Fehler-Kode
005FC2 4A87              tst.l   D7              ok ?
005FC4 6610              bne     $5FD6           nein
005FC6 2EBC000012BC      move.l  #$12BC,(A7)     Sektor-Puffer
005FCC 61000298          bsr     $6266           u2i, 8086-Integer in 68000-Integer wandeln
005FD0 4A40              tst.w   D0              Bad sector list
005FD2 6702              beq     $5FD6           Sektoren in Ordnung ?
```

```
005FD4 7EF0              moveq.l #-16,D7          bad sectors
005FD6 603A              bra     $6012

005FD8 3E83              move.w  D3,(A7)          Anzahl Sektoren
005FDA 3F05              move.w  D5,-(A7)         Seite
005FDC 3F06              move.w  D6,-(A7)         Track
005FDE 3F04              move.w  D4,-(A7)         Sektor
005FE0 3F2E0010          move.w  16(A6),-(A7)     Drive
005FE4 42A7              clr.l   -(A7)            Filler
005FE6 2F2EFFFA          move.l  -6(A6),-(A7)     Sektor-Puffer
005FEA 4EB9000062D2      jsr     $62D2            floprd, Sektor lesen
005FF0 DFFC00000010      add.l   #$10,A7          Stackpointer korrigieren
005FF6 2E00              move.l  D0,D7            Fehler-Kode merken
005FF8 202EFFFA          move.l  -6(A6),D0        benutzter Puffer
005FFC B0AE000A          cmp.l   10(A6),D0        gleich gewünschter Puffer ?
006000 6710              beq     $6012            ja
006002 2EAE000A          move.l  10(A6),(A7)      Quelladresse
006006 2F2EFFFA          move.l  -6(A6),-(A7)     Zieladresse
00600A 4EB900005910      jsr     $5910            fastcpy, Sektor kopieren
006010 588F              addq.l  #4,A7            Stackpointer korrigieren
006012 4A87              tst.l   D7               Fehlerkode testen
006014 6C12              bge     $6028            ok ?
006016 3EAE0010          move.w  16(A6),(A7)      Drivenummer
00601A 2007              move.l  D7,D0            Fehlerkode
00601C 3F00              move.w  D0,-(A7)         auf Stack
00601E 4EB90000555C      jsr     $555C            critical error handler
006024 548F              addq.l  #2,A7            Stackpointer korrigieren
006026 2E00              move.l  D0,D7            Fehlerkode holen
006028 BEBC00010000      cmp.l   #$10000,D7       nochmal versuchen ?
00602E 6700FF22          beq     $5F52            ja
006032 4A87              tst.l   D7               Fehlerkode
006034 6C04              bge     $603A            ok
006036 2007              move.l  D7,D0            Fehlerkode als Ergebnis
006038 601E              bra     $6058
```

```
00603A 3003              move.w  D3,D0           Sektor-Zähler
00603C 48C0              ext.l   D0
00603E 7209              moveq.l #9,D1           mal 512
006040 E3A0              asl.l   D1,D0
006042 D1AE000A          add.l   D0,10(A6)       Pufferadresse erhöhen
006046 D76E000E          add.w   D3,14(A6)       logische Sektornummer plus Zähler
00604A 976E0012          sub.w   D3,18(A6)       Anzahl zu verarbeitende Sektoren vermindern
00604E 4A6E0012          tst.w   18(A6)          noch Sektoren zu lesen/schreiven ?
006052 6600FE9E          bne     $5EF2           ja
006056 4280              clr.l   D0              ok
006058 4A9F              tst.l   (A7)+
00605A 4CDF20F8          movem.l (A7)+,D3-D7/A5  Register zurückholen
00605E 4E5E              unlk    A6
006060 4E75              rts

************************************************  random, Zufallszahl generieren
006062 4E56FFFC          link    A6,#-4
006066 4AB9000025FA      tst.l   $25FA           letzte Zufallszahl
00606C 6616              bne     $6084           nicht null ?
00606E 2039000004BA      move.l  $4BA,D0         200 Hz Zähler
006074 7210              moveq.l #16,D1
006076 E3A0              asl.l   D1,D0           Bit 0-15 nach 16-31
006078 80B9000004BA      or.l    $4BA,D0         plus 200 Hz Zähler
00607E 23C0000025FA      move.l  D0,$25FA        als Startwert benutzen
006084 2F3CBB40E62D      move.l  #$BB40E62D,-(A7) 3141592621
00608A 2F39000025FA      move.l  $25FA,-(A7)     letzter Zufallswert
006090 4EB9000094FA      jsr     $94FA           32 * 32 Bit-Multiplikation
006096 508F              addq.l  #8,A7           Stackpointer korrigieren
006098 5280              addq.l  #1,D0           Ergebnis plus 1
00609A 23C0000025FA      move.l  D0,$25FA        als neuen Startwert
0060A0 2039000025FA      move.l  $25FA,D0
0060A6 E080              asr.l   #8,D0           Bit 31-8 nach 23-0
0060A8 C0BC00FFFFFF      and.l   #$FFFFFF,D0     24-Bit-Zufallszahl als Ergebnis
0060AE 4E5E              unlk    A6
0060B0 4E75              rts
```

```
************************************************     bootload, Bootsektor laden
0060B2 4E560000            link    A6,#0
0060B6 48E70300            movem.l D6-D7,-(A7)      Register retten
0060BA 4EB900005932        jsr     $5932            hdv init
0060C0 4A79000004A6        tst.w   $4A6             nflops
0060C6 6704                beq     $60CC            kein Laufwerk angeschlossen ?
0060C8 7001                moveq.l #1,D0            'couldn't load'
0060CA 6002                bra     $60CE
0060CC 7002                moveq.l #2,D0            'no drive'
0060CE 3E00                move.w  D0,D7            Fehler merken
0060D0 4A79000004A6        tst.w   $4A6             nflops
0060D6 6744                beq     $611C
0060D8 0C79000200000446    cmp.w   #2,$446          bootdev
0060E0 6C3A                bge     $611C            keine Diskette ?
0060E2 3EBC0001            move.w  #1,(A7)          ein Sektor
0060E6 4267                clr.w   -(A7)            Seite 0
0060E8 4267                clr.w   -(A7)            Track 0
0060EA 3F3C0001            move.w  #1,-(A7)         Sektor 1
0060EE 3F3900000446        move.w  $446,-(A7)       bootdev als Drive-Nummer
0060F4 42A7                clr.l   -(A7)            Filler
0060F6 2F3C000012BC        move.l  #$12BC,-(A7)     Sektor-Puffer
0060FC 4EB9000062D2        jsr     $62D2            floprd, Sektor lesen
006102 DFFC00000010        add.l   #$10,A7          Stackpointer korrigieren
006108 4A80                tst.l   D0               Fehler ?
00610A 6604                bne     $6110            nein
00610C 4247                clr.w   D7
00610E 600C                bra     $611C
006110 4A3900000674        tst.b   $674             wpstatus
006116 6604                bne     $611C
006118 7003                moveq.l #3,D0            'unreadable'
00611A 6024                bra     $6140

00611C 4A47                tst.w   D7
00611E 6704                beq     $6124
006120 3007                move.w  D7,D0            alten Error-Code holen
006122 601C                bra     $6140
```

```
006124 3EBC0100            move.w  #$100,(A7)          $100 Worte
006128 2F3C000012BC        move.l  #$12BC,-(A7)        Sektor-Puffer
00612E 61000106            bsr     $6236               Checksumme berechnen
006132 588F                addq.l  #4,A7               Stackpointer korrigieren
006134 B07C1234            cmp.w   #$1234,D0           mit Checksumme für Bootsektor vergleichen
006138 6604                bne     $613E               ungleich ?
00613A 4240                clr.w   D0                  ok
00613C 6002                bra     $6140
00613E 7004                moveq.l #4,D0               'not valid boot sector'
006140 4A9F                tst.l   (A7)+
006142 4CDF0080            movem.l (A7)+,D7            Register zurückholen
006146 4E5E                unlk    A6
006148 4E75                rts

************************************************      proto bt, Bootsektor generieren
00614A 4E56FFFA            link    A6,#-6
00614E 48E70704            movem.l D5-D7/A5,-(A7)      Register retten
006152 4A6E0012            tst.w   18(A6)              execflg testen
006156 6C1E                bge     $6176               Ausführbarkeit beibehalten
006158 3EBC0100            move.w  #$100,(A7)          $100 Worte
00615C 2F2E0008            move.l  8(A6),-(A7)         Adresse des Sektorpuffers
006160 610000D4            bsr     $6236               Checksumme berechnen
006164 588F                addq.l  #4,A7               Stackpointer korrigieren
006166 B07C1234            cmp.w   #$1234,D0           gleich Checksumme für Bootsektor?
00616A 6704                beq     $6170               ja
00616C 4240                clr.w   D0                  nicht ausführbar
00616E 6002                bra     $6172
006170 7001                moveq.l #1,D0               ausführbar
006172 3D400012            move.w  D0,18(A6)           execflg auf 1 = Bootsektor ausführbar
006176 4AAE000C            tst.l   12(A6)              Serial-Nummer
00617A 6D3E                blt     $61BA               negativ, nicht ändern
00617C 202E000C            move.l  12(A6),D0           Serial-Nummer
006180 B0BC00FFFFFF        cmp.l   #$FFFFFF,D0         > $FFFFFF ?
006186 6F08                ble     $6190               nein
006188 6100FED8            bsr     $6062               rand, Zufallszahl generieren
00618C 2D40000C            move.l  D0,12(A6)           merken
006190 4247                clr.w   D7                  Zähler löschen
```

```
006192 6020              bra     $61B4

006194 202E000C          move.l  12(A6),D0       Zufallszahl
006198 C0BC000000FF      and.l   #$FF,D0         Bit 0-7
00619E 3247              move.w  D7,A1           Zeiger auf nächstes Byte im Puffer
0061A0 D3EE0008          add.l   8(A6),A1        plus Basisadresse
0061A4 13400008          move.b  D0,8(A1)        Byte der Serialnummer in Puffer schreiben
0061A8 202E000C          move.l  12(A6),D0       Zufallszahl
0061AC E080              asr.l   #8,D0           um 8 Bits nach rechts schieben
0061AE 2D40000C          move.l  D0,12(A6)       und neuen Wert merken
0061B2 5247              addq.w  #1,D7           Zähler erhöhen
0061B4 BE7C0003          cmp.w   #$3,D7          schon drei Bytes kopiert ?
0061B8 6DDA              blt     $6194           nein
0061BA 4A6E0010          tst.w   16(A6)          Diskettengröße
0061BE 6D28              blt     $61E8           negativ, nicht ändern
0061C0 3C2E0010          move.w  16(A6),D6       Diskettengröße
0061C4 CDFC0013          muls.w  #$13,D6         mal 19 gleich Zeiger auf Prototyp BPB
0061C8 4247              clr.w   D7              Zähler löschen
0061CA 6016              bra     $61E2
0061CC 3047              move.w  D7,A0           Zähler
0061CE D1EE0008          add.l   8(A6),A0        plus Adresse des Puffers
0061D2 3246              move.w  D6,A1
0061D4 D3FC00016D00      add.l   #$16D00,A1      Adresse des Prototyp BPB
0061DA 1151000B          move.b  (A1),11(A0)     BPB kopieren
0061DE 5246              addq.w  #1,D6
0061E0 5247              addq.w  #1,D7           Zähler erhöhen
0061E2 BE7C0013          cmp.w   #$13,D7         schon 19 ?
0061E6 6DE4              blt     $61CC           nein
0061E8 426EFFFA          clr.w   -6(A6)
0061EC 2D6E0008FFFC      move.l  8(A6),-4(A6)    Pufferadresse
0061F2 600E              bra     $6202
0061F4 206EFFFC          move.l  -4(A6),A0
0061F8 3010              move.w  (A0),D0         Wort aus Puffer holen
0061FA D16EFFFA          add.w   D0,-6(A6)       zur Checksummenbildung summieren
0061FE 54AEFFFC          addq.l  #2,-4(A6)       nächstes Wort
006202 202E0008          move.l  8(A6),D0        Pufferadresse
006206 D0BC000001FE      add.l   #$1FE,D0        plus $1FE
```

```
00620C B0AEFFFC          cmp.l   -4(A6),D0        schon letztes Wort
006210 62E2              bhi     $61F4
006212 303C1234          move.w  #$1234,D0       Checksumme für Bootsektor
006216 906EFFFA          sub.w   -6(A6),D0       von bisherigem Wert abziehen
00621A 226EFFFC          move.l  -4(A6),A1
00621E 3280              move.w  D0,(A1)         Checksumme in Puffer
006220 4A6E0012          tst.w   18(A6)          execflg
006224 6606              bne     $622C           Bootsektor ausführbar ?
006226 206EFFFC          move.l  -4(A6),A0
00622A 5250              addq.w  #1,(A0)         Checksumme erhöhen, nicht ausführbar
00622C 4A9F              tst.l   (A7)+
00622E 4CDF20C0          movem.l (A7)+,D6-D7/A5  Register zurückholen
006232 4E5E              unlk    A6
006234 4E75              rts

************************************************  Checksumme berechnen
006236 4E560000          link    A6,#0
00623A 48E70300          movem.l D6-D7,-(A7)     Register retten
00623E 4247              clr.w   D7              Summe löschen
006240 600C              bra     $624E
006242 206E0008          move.l  8(A6),A0        Adresse des Puffers
006246 3010              move.w  (A0),D0         Wort holen
006248 DE40              add.w   D0,D7           und summieren
00624A 54AE0008          addq.l  #2,8(A6)        Zeiger auf nächstes Wort
00624E 302E000C          move.w  12(A6),D0       Anzahl der Worte
006252 536E000C          subq.w  #1,12(A6)       minus 1
006256 4A40              tst.w   D0              schon alle Worte addiert ?
006258 66E8              bne     $6242           nein
00625A 3007              move.w  D7,D0           Ergebnis nach d0
00625C 4A9F              tst.l   (A7)+
00625E 4CDF0080          movem.l (A7)+,D7        Register zurückholen
006262 4E5E              unlk    A6
006264 4E75              rts
```

```
************************************************    u2i, 8086-Zahl in 68000-Zahl wandeln
006266 4E56FFFC             link    A6,#-4
00626A 206E0008             move.l  8(A6),A0        Adresse der Zahl
00626E 10280001             move.b  1(A0),D0        Hi-Byte holen
006272 4880                 ext.w   D0              auf Wort erweitern
006274 C07C00FF             and.w   #$FF,D0         Bits 0-7 isolieren
006278 E140                 asl.w   #8,D0           in Position 8-15 schieben
00627A 226E0008             move.l  8(A6),A1        Adresse der Zahl
00627E 1211                 move.b  (A1),D1         Lo-Byte holen
006280 4881                 ext.w   D1              auf Wort erweitern
006282 C27C00FF             and.w   #$FF,D1         Bit 0-7 isolieren
006286 8041                 or.w    D1,D0           mit Bit 8-15 verknüpfen
006288 4E5E                 unlk    A6
00628A 4E75                 rts

************************************************    flopini, Laufwerk initialisieren
00628C 43F9000006C8         lea     $6C8,A1         dsb0, Zeiger auf DSB Laufwerk A
006292 4A6F000C             tst.w   12(A7)          Laufwerk A ?
006296 6706                 beq     $629E           ja
006298 43F9000006CC         lea     $6CC,A1         dsb1, DSB von Laufwerk B nehmen
00629E 3379000004400002     move.w  $440,2(A1)      seekrate
0062A6 70FF                 moveq.l #-1,D0          Default Error Nummer
0062A8 42690000             clr.w   0(A1)           Tracknummer auf Null
0062AC 610004BA             bsr     $6768           floplock, Parameter setzen
0062B0 61000696             bsr     $6948           select Laufwerk und Seite
0062B4 337CFF000000         move.w  #$FF00,0(A1)
0062BA 61000618             bsr     $68D4           restore
0062BE 7E06                 moveq.l #6,D7
0062C0 610005A0             bsr     $6862
0062C4 6608                 bne     $62CE
0062C6 6100060C             bsr     $68D4           restore
0062CA 67000542             beq     $680E           flopok, kein Fehler
0062CE 60000530             bra     $6800           flopfail, Fehler
```

```
************************************************  floprd, Sektor(en) von Diskette lesen
0062D2 6100071E            bsr     $69F2          change, Test auf Diskettenwechsel
0062D6 70F5                moveq.l #-11,D0        Fehlernummer auf Readerror
0062D8 6100048E            bsr     $6768          floplock, Parameter setzen
0062DC 6100066A            bsr     $6948          select Laufwerk und Seite
0062E0 610005CC            bsr     $68AE          go2track, Track suchen
0062E4 66000090            bne     $6376          bei Fehler nochmal versuchen
0062E8 33FCFFFF000006A2    move.w  #-1,$6A2       currerr auf Default Error
0062F0 3CBC0090            move.w  #$90,(A6)      DMA-Status löschen
0062F4 3CBC0190            move.w  #$190,(A6)
0062F8 3CBC0090            move.w  #$90,(A6)      Datenrichtung auf READ
0062FC 33ED068CFFFF8604    move.w  $68C(A5),$FFFF8604  ccount nach dskctl, Sektorzähler
006304 3CBC0080            move.w  #$80,(A6)      Read Sector Befehl für 1772
006308 3E3C0090            move.w  #$90,D7        Read multiple
00630C 610006B6            bsr     $69C4          wdiskctl, D7 an 1772 übergeben
006310 2E3C00040000        move.l  #$40000,D7     Timeout Zähler initialisieren
006316 246D0692            move.l  $692(A5),A2    edma, Zieladresse für DMA
00631A 08390005FFFFFA01    btst    #5,$FFFFFA01   mfp gpip, 1772 fertig ?
006322 6734                beq     $6358          ja
006324 5387                subq.l  #1,D7          Timeout Zähler vermindern
006326 6724                beq     $634C          abgelaufen ?
006328 1B79FFFF8609069D    move.b  $FFFF8609,$69D(A5)  dmahigh
006330 1B79FFFF860B069E    move.b  $FFFF860B,$69E(A5)  dmamid
006338 1B79FFFF860D069F    move.b  $FFFF860D,$69F(A5)  dmalow
006340 B5ED069C            cmp.l   $69C(A5),A2    augenblickliche DMA-Adresse gleich edma?
006344 6ED4                bgt     $631A          nein, weiter warten
006346 610005E6            bsr     $692E          reset, Übertragung beenden
00634A 600C                bra     $6358

00634C 3B7CFFFE06A2        move.w  #-2,$6A2(A5)   currerr auf timeout
006352 610005DA            bsr     $692E          reset, Übertragung beenden
006356 601E                bra     $6376          nächsten Versuch starten
```

```
006358 3CBC0090            move.w  #$90,(A6)       DMA Statusregister selektieren
00635C 3016                move.w  (A6),D0         Status lesen
00635E 08000000            btst    #0,D0           DMA-Error ?
006362 6712                beq     $6376           ja, nochmal versuchen
006364 3CBC0080            move.w  #$80,(A6)       1772 Statusregister selektieren
006368 6100066E            bsr     $69D8           rdiskctl, Register lesen
00636C C03C0018            and.b   #$18,D0         RNF, Checksum und Lost Data isolieren
006370 6700049C            beq     $680E           flopok, kein Fehler
006374 6118                bsr     $638E           errbits, Fehlernummer bestimmen
006376 0C6D00010672        cmp.w   #1,$672(A5)     retrycnt auf zweitem Versuch ?
00637C 6604                bne     $6382           nein
00637E 610004FA            bsr     $687A           reseek, Home und reseek
006382 536D0672            subq.w  #1,$672(A5)     retrycnt, Versuchezähler vermindern
006386 6A00FF54            bpl     $62DC           noch ein Versuch ?
00638A 60000474            bra     $6800           nein, flopfail

************************************************   errbits, 1772-Status in Fehlernummer
00638E 72F3                moveq.l #-13,D1         write protect ?
006390 08000006            btst    #6,D0
006394 6614                bne     $63AA           ja
006396 72F8                moveq.l #-8,D1          record not found ?
006398 08000004            btst    #4,D0
00639C 660C                bne     $63AA           ja
00639E 72FC                moveq.l #-4,D1          CRC Error ?
0063A0 08000003            btst    #3,D0
0063A4 6704                beq     $63AA           ja
0063A6 322D06A0            move.w  $6A0(A5),D1     deferror, Default Error nehmen
0063AA 3B4106A2            move.w  D1,$6A2(A5)     als currerr
0063AE 4E75                rts

************************************************   flopwr, Sektor(en) auf Diskette schreiben
0063B0 61000640            bsr     $69F2           change, Test auf Diskettenwechsel
0063B4 70F6                moveq.l #-10,D0         Defaulterror auf Write Error
0063B6 610003B0            bsr     $6768           floplock, Parameter setzen
0063BA 302D0688            move.w  $688(A5),D0     csect, Sektornummer 1?
0063BE 5340                subq.w  #1,D0
0063C0 806D0686            or.w    $686(A5),D0     ctrack, Tracknummer 0?
```

```
0063C4 806D068A          or.w    $68A(A5),D0     cside, Seite 0?
0063C8 6606              bne     $63D0           nein, nicht Bootsektor
0063CA 7002              moveq.l #2,D0           media change
0063CC 6100065C          bsr     $6A2A           auf 'unsure' setzen
0063D0 61000576          bsr     $6948           select, Laufwerk und Seite selektieren
0063D4 610004D8          bsr     $68AE           go2track, Track suchen
0063D8 6600007E          bne     $6458           Fehler, nochmal versuchen
0063DC 3B7CFFFF06A2      move.w  #-1,$6A2(A5)    currerr auf Default error
0063E2 3CBC0190          move.w  #$190,(A6)      DMA-Stautus löschen
0063E6 3CBC0090          move.w  #$90,(A6)
0063EA 3CBC0190          move.w  #$190,(A6)      Datenrichtung auf WRITE
0063EE 3E3C0001          move.w  #1,D7           Sector-Count-Register
0063F2 610005D0          bsr     $69C4           wdiskctl, D7 an 1772
0063F6 3CBC0180          move.w  #$180,(A6)      1772 selektieren
0063FA 3E3C00A0          move.w  #$A0,D7         Write Sector
0063FE 610005C4          bsr     $69C4           wdiskctl, D7 an 1772
006402 2E3C00040000      move.l  #$40000,D7      Timeout Zähler
006408 08390005FFFFFA01  btst    #5,$FFFFFA01    mfp, gpip, 1772 fertig ?
006410 670A              beq     $641C           ja
006412 5387              subq.l  #1,D7           Timeout-Zähler erniedrigen
006414 66F2              bne     $6408           noch nicht abgelaufen ?
006416 61000516          bsr     $692E           reset, Übertragung abbrechen
00641A 6034              bra     $6450

00641C 3CBC0180          move.w  #$180,(A6)      1772 Statusregister selektieren
006420 610005B6          bsr     $69D8           rdiskctl, 1772-Register lesen
006424 6100FF68          bsr     $638E           errbits, Fehlernummer berechnen
006428 08000006          btst    #6,D0           write protect ?
00642C 660003D2          bne     $6800           flopfail, kein weiterer Versuch
006430 C03C005C          and.b   #$5C,D0         write protect, RNF, Checksum und Lost Data
006434 661A              bne     $6450           Fehler, nochmal versuchen
006436 526D0688          addq.w  #1,$688(A5)     csect, Sektornummer erhöhen
00643A 06AD00000200068E  add.l   #512,$68E(A5)   cdma, DMA-Adresse auf nächsten Sektor
006442 536D068C          subq.w  #1,$68C(A5)     ccount, Anzahl Sektoren vermindern
006446 670003C6          beq     $680E           flopok, alle Sektoren, dann fertig
00644A 61000524          bsr     $6970           select1, Sektornummer und DMA-Zeiger
00644E 608C              bra     $63DC           nächsten Sektor ohne Seek schreiben
```

```
006450 0C6D00010672        cmp.w   #1,$672(A5)          retrycnt, zweiter Versuch ?
006456 6604                bne     $645C                nein
006458 61000420            bsr     $687A                ressek, Home und Seek
00645C 536D0672            subq.w  #1,$672(A5)          retrycnt, Versuchezähler vermindern
006460 6A00FF6E            bpl     $63D0                noch ein Versuch ?
006464 6000039A            bra     $6800                flopfail, Fehler

*********************************************           flopfmt, Track formatieren
006468 0CAF876543210016    cmp.l   #$87654321,22(A7)    Magic Number ?
006470 6600038E            bne     $6800                nein, flopfail
006474 6100057C            bsr     $69F2                change, Test auf Diskettenwechsel
006478 70FF                moveq.l #-1,D0               Default Error Nummer
00647A 610002EC            bsr     $6768                floplock, Parameter setzen
00647E 610004C8            bsr     $6948                select, Laufwerk und Seite selektieren
006482 3B6F000E0696        move.w  14(A7),$696(A5)      spt, Sektoren pro Track
006488 3B6F00140698        move.w  20(A7),$698(A5)      interlv, Interleave Faktor
00648E 3B6F001A069A        move.w  26(A7),$69A(A5)      virgin, Sektordaten beim Formatieren
006494 7002                moveq.l #2,D0                'changed'
006496 61000592            bsr     $6A2A                Diskette gewechselt
00649A 610003C0            bsr     $685C                hseek, Track suchen
00649E 66000360            bne     $6800                flopfail, nicht gefunden
0064A2 336D06860000        move.w  $686(A5),0(A1)       ctrack, aktuellen Track in DSB schreiben
0064A8 3B7CFFFF06A2        move.w  #-1,$6A2(A5)         currerr auf Default Error
0064AE 6128                bsr     $64D8                Track formatieren
0064B0 6600034E            bne     $6800                flopfail, Fehler
0064B4 3B6D0696068C        move.w  $696(A5),$68C(A5)    spt Sektoren pro Track als ccount Zähler
0064BA 3B7C00010688        move.w  #1,$688(A5)          csect, Beginn mit Sektor 1
0064C0 6100015C            bsr     $661E                verify, Sektor verifizieren
0064C4 246D068E            move.l  $68E(A5),A2          cdma, Liste mit schlechten Sektoren
0064C8 4A52                tst.w   (A2)                 schlechter Sektor ?
0064CA 67000342            beq     $680E                nein
0064CE 3B7CFFF006A2        move.w  #-16,$6A2(A5)        currerr auf 'Bad Sector'
0064D4 6000032A            bra     $6800                flopfail, Fehler
```

```
*****************************************************  fmtrack, Track formatieren
0064D8 3B7CFFF606A0          move.w  #-10,$6A0(A5)    deferror, Default Error Nummer
0064DE 363C0001              move.w  #1,D3            Start mit Sektor 1
0064E2 246D068E              move.l  $68E(A5),A2      cdma, Puffer für Trackdaten
0064E6 323C003B              move.w  #$3B,D1          60 mal
0064EA 103C004E              move.b  #$4E,D0          $4E, Trackvorspann
0064EE 6100010A              bsr     $65FA            wmult, in Puffer schreiben
0064F2 3803                  move.w  D3,D4            Sektornummer merken
0064F4 323C000B              move.w  #$B,D1           12 mal
0064F8 4200                  clr.b   D0               0
0064FA 610000FE              bsr     $65FA            wmult, in Puffer schreiben
0064FE 323C0002              move.w  #2,D1            3 mal
006502 103C00F5              move.b  #$F5,D0          $F5
006506 610000F2              bsr     $65FA            wmult, in Puffer schreiben
00650A 14FC00FE              move.b  #$FE,(A2)+       $FE, Adreßmarke
00650E 14F900000687          move.b  $687,(A2)+       Track
006514 14F90000068B          move.b  $68B,(A2)+       Seite
00651A 14C4                  move.b  D4,(A2)+         Sektor
00651C 14FC0002              move.b  #2,(A2)+         Sektorgröße (512 Bytes)
006520 14FC00F7              move.b  #$F7,(A2)+       Checksumme schreiben
006524 323C0015              move.w  #$15,D1          22 mal
006528 103C004E              move.b  #$4E,D0          $4E
00652C 610000CC              bsr     $65FA            wmult, in Puffer schreiben
006530 323C000B              move.w  #$B,D1           12 mal
006534 4200                  clr.b   D0               0
006536 610000C2              bsr     $65FA            wmult, in Puffer schreiben
00653A 323C0002              move.w  #2,D1            3 mal
00653E 103C00F5              move.b  #$F5,D0          $F5
006542 610000B6              bsr     $65FA            wmult, in Puffer schreiben
006546 14FC00FB              move.b  #$FB,(A2)+       $FB, Datenblockmarke
00654A 323C00FF              move.w  #$FF,D1          256 mal
00654E 14ED069A              move.b  $69A(A5),(A2)+   virgin, Initialdaten in Puffer
006552 14ED069B              move.b  $69B(A5),(A2)+
006556 51C9FFF6              dbra    D1,$654E         nächstes Wort
00655A 14FC00F7              move.b  #$F7,(A2)+       Checksumme schreiben
00655E 323C0027              move.w  #$27,D1          40 mal
006562 103C004E              move.b  #$4E,D0          $4E
```

```
006566 61000092            bsr     $65FA              wmult, in Puffer schreiben
00656A D86D0698            add.w   $698(A5),D4        interlv addieren, nächsten Sektor
00656E B86D0696            cmp.w   $696(A5),D4        spt, größte Sektornummer ?
006572 6F80                ble     $64F4              nein, nächsten Sektor
006574 5243                addq.w  #1,D3              Startsektor plus eins
006576 B66D0698            cmp.w   $698(A5),D3        interlv
00657A 6F00FF76            ble     $64F2              nächsten Sektor
00657E 323C0578            move.w  #$578,D1           1401 mal (bis Trackende)
006582 103C004E            move.b  #$4E,D0            $4E
006586 6172                bsr     $65FA              wmult, in Puffer schreiben
006588 13ED0691FFFF860D    move.b  $691(A5),$FFFF860D dmalow
006590 13ED0690FFFF860B    move.b  $690(A5),$FFFF860B dmamid
006598 13ED068FFFFF8609    move.b  $68F(A5),$FFFF8609 dmahigh
0065A0 3CBC0190            move.w  #$190,(A6)         DMA-Status löschen
0065A4 3CBC0090            move.w  #$90,(A6)
0065A8 3CBC0190            move.w  #$190,(A6)         Datenrichtung auf WRITE
0065AC 3E3C001F            move.w  #$1F,D7            Sektorzähler auf 31
0065B0 61000412            bsr     $69C4              wdiskctl, D7 an 1772
0065B4 3CBC0180            move.w  #$180,(A6)         1772 selektieren
0065B8 3E3C00F0            move.w  #$F0,D7            Format Track Befehl
0065BC 61000406            bsr     $69C4              wdiskctl, D7 an 1772
0065C0 2E3C00040000        move.l  #$40000,D7         Timeout Zähler
0065C6 08390005FFFFFA01    btst    #5,$FFFFFA01       mfp gpip, 1772 fertig ?
0065CE 670C                beq     $65DC              ja
0065D0 5387                subq.l  #1,D7              Timeout Zähler vermindern
0065D2 66F2                bne     $65C6              noch nicht abgelaufen ?
0065D4 61000358            bsr     $692E              reset, abbrechen
0065D8 7E01                moveq.l #1,D7              Z-Bit löschen, Fehler
0065DA 4E75                rts

0065DC 3CBC0190            move.w  #$190,(A6)         DMA-Status selektieren
0065E0 3016                move.w  (A6),D0            Status lesen
0065E2 08000000            btst    #0,D0              DMA Error ?
0065E6 67F0                beq     $65D8              ja
0065E8 3CBC0180            move.w  #$180,(A6)         1772 Statusregister selektieren
0065EC 610003EA            bsr     $69D8              rdiskctl, Register lesen
0065F0 6100FD9C            bsr     $638E              errbits, Fehlernummer berechnen
```

```
0065F4 C03C0044          and.b   #$44,D0          Write Protect und Lost Data testen
0065F8 4E75              rts

0065FA 14C0              move.b  D0,(A2)+        Daten in Puffer schreiben
0065FC 51C9FFFC          dbra    D1,$65FA        nächstes Byte
006600 4E75              rts

*********************************************    flopver, Sektor(en) verifizieren
006602 610003EE          bsr     $69F2           change, Test auf Diskettenwechsel
006606 70F5              moveq.l #-11,D0         'read error', als Default error
006608 6100015E          bsr     $6768           floplock, Parameter setzen
00660C 6100033A          bsr     $6948           select
006610 6100029C          bsr     $68AE           go2track, Track suchen
006614 660001EA          bne     $6800           flopfail, Fehler
006618 6104              bsr     $661E           verify1, Sektoren verifizieren
00661A 600001F2          bra     $680E           flopok, fertig

*********************************************
00661E 3B7CFFF506A0      move.w  #-11,$6A0(A5)   deferror auf 'read error'
006624 246D068E          move.l  $68E(A5),A2     cdma, DMA-Puffer für bad sector list
006628 06AD00000200068E  add.l   #$200,$68E(A5)  cdma auf nächsten Sektor
006630 3B7C00020672      move.w  #2,$672(A5)     retrycnt, 2 Versuche
006636 3CBC0084          move.w  #$84,(A6)       Sektor-Register selektieren
00663A 3E2D0688          move.w  $688(A5),D7     csect, Sektornummer
00663E 61000384          bsr     $69C4           wdiskctl, an Diskcontroller
006642 13ED0691FFFF860D  move.b  $691(A5),$FFFF860D  dmalow
00664A 13ED0690FFFF860B  move.b  $690(A5),$FFFF860B  dmamid
006652 13ED068FFFFF8609  move.b  $68F(A5),$FFFF8609  dmahigh
00665A 3CBC0090          move.w  #$90,(A6)       DMA-Status löschen
00665E 3CBC0190          move.w  #$190,(A6)
006662 3CBC0090          move.w  #$90,(A6)       Datenrichtung auf READ
006666 3E3C0001          move.w  #1,D7           Sektor-Zähler auf 1
00666A 61000358          bsr     $69C4           wdiskctl
00666E 3CBC0080          move.w  #$80,(A6)       1772 Befehlsregister selektieren
006672 3E3C0080          move.w  #$80,D7         Read Sector Befehl
006676 6100034C          bsr     $69C4           wdiskctl
00667A 2E3C00040000      move.l  #$40000,D7      Timeout-Zähler
```

```
006680 08390005FFFFFA01     btst    #5,$FFFFFA01        mfp gpip, 1772 fertig ?
006688 670A                 beq     $6694               ja
00668A 5387                 subq.l  #1,D7               Timeout-Zähler dekrementieren
00668C 66F2                 bne     $6680               noch nicht abgelaufen ?
00668E 6100029E             bsr     $692E               reset 1772
006692 6036                 bra     $66CA               nächster Versuch

006694 3CBC0090             move.w  #$90,(A6)           DMA-Status-Register selektieren
006698 3016                 move.w  (A6),D0             Status lesen
00669A 08000000             btst    #0,D0               DMA error ?
00669E 672A                 beq     $66CA               ja, nochmal versuchen
0066A0 3CBC0080             move.w  #$80,(A6)           1772 Status-Register selektieren
0066A4 61000332             bsr     $69D8               rdiskctl, Status lesen
0066A8 6100FCE4             bsr     $638E               errbits, Fehlernummer berechnen
0066AC C03C001C             and.b   #$1C,D0             RNF, CRC und lost data prüfen
0066B0 6618                 bne     $66CA               Fehler, nächster Versuch
0066B2 526D0688             addq.w  #1,$688(A5)         csect, nächster Sektor
0066B6 536D068C             subq.w  #1,$68C(A5)         ccount, Sektorzähler dekrementieren
0066BA 6600FF74             bne     $6630               noch ein Sektor ?
0066BE 04AD00000200068E     sub.l   #$200,$68E(A5)      cdma, DMA-Zeiger zurücksetzen
0066C6 4252                 clr.w   (A2)                bad sector list mit Null abschließen
0066C8 4E75                 rts

0066CA 0C6D00010672         cmp.w   #1,$672(A5)         retrycnt, 2. Versuch ?
0066D0 6604                 bne     $66D6               nein
0066D2 610001A6             bsr     $687A               reseek
0066D6 536D0672             subq.w  #1,$672(A5)         retrycnt vermindern
0066DA 6A00FF66             bpl     $6642               noch ein Versuch ?
0066DE 34ED0688             move.w  $688(A5),(A2)+      csect, Sectornummer in bad sector list
0066E2 60CE                 bra     $66B2               nächsten Sektor

************************************************      flopvbl, Floppy Vertical Blank Handler
0066E4 9BCD                 sub.l   A5,A5               A5 löschen
0066E6 4DF9FFFF8606         lea     $FFFF8606,A6        Adresse der Floppyregister
0066EC 50ED0680             st      $680(A5)            motoron Flag setzen
0066F0 4A6D043E             tst.w   $43E(A5)            flock, Floppies in Arbeit ?
0066F4 6670                 bne     $6766               ja, nichts machen
```

```
0066F6 203900000466     move.l  $466,D0         frclock
0066FC 1200             move.b  D0,D1
0066FE C23C0007         and.b   #7,D1           mod 8 berechnen
006702 6638             bne     $673C           noch nicht 8. Interrupt ?
006704 3CBC0080         move.w  #$80,(A6)       1772 Statusregister selektieren
006708 E608             lsr.b   #3,D0           Bit 4 als Drivenummer benutzen
00670A C07C0001         and.w   #1,D0
00670E 41ED0674         lea     $674(A5),A0     wpstatus, Write Protect Status Tabelle
006712 D0C0             add.w   D0,A0           mit Drivenummer indizieren
006714 B079000004A6     cmp.w   $4A6,D0         nflops, Anzahl der Floppies
00671A 6602             bne     $671E
00671C 4240             clr.w   D0
00671E 5200             addq.b  #1,D0           driev select bit
006720 E308             lsl.b   #1,D0           in Position schieben
006722 0A000007         eor.b   #7,D0           invertieren für Hardware
006726 6100026C         bsr     $6994           Drive selektieren
00672A 3039FFFF8604     move.w  $FFFF8604,D0    dskctl, 1772 Status lesen
006730 08000006         btst    #6,D0           Write Protect Bit testen
006734 56D0             sne     (A0)            und merken
006736 1002             move.b  D2,D0           vorherigen Select-Status
006738 6100025A         bsr     $6994           wiederherstellen
00673C 302D0674         move.w  $674(A5),D0     wpstatus
006740 816D0676         or.w    D0,$676(A5)     in wplatch schreiben
006744 4A6D0682         tst.w   $682(A5)        deslflg, Floppies schon deselektiert ?
006748 6618             bne     $6762           ja
00674A 6100028C         bsr     $69D8           1772 Status-Register lesen
00674E 08000007         btst    #7,D0           Motor on Bit gesetzt ?
006752 6612             bne     $6766           ja, nicht deselektieren
006754 103C0007         move.b  #7,D0           beide Drives
006758 6100023A         bsr     $6994           deselektieren
00675C 3B7C00010682     move.w  #1,$682(A5)     deslflg setzen
006762 426D0680         clr.w   $680(A5)        motoron Flag löschen
006766 4E75             rts
```

```
************************************************  floplock
006768 48F978F8000006A4     movem.l D3-D7/A3-A6,$6A4  regsave
006770 9BCD                 sub.l   A5,A5             A5 löschen
006772 4DF9FFFF8606         lea     $FFFF8606,A6      Adresse der Floppy-Register
006778 50F900000680         st      $680              motoron Flag setzen
00677E 3B4006A0             move.w  D0,$6A0(A5)       deferror
006782 3B4006A2             move.w  D0,$6A2(A5)       currerr
006786 3B7C0001043E         move.w  #1,$43E(A5)       flock, Floppy-vbl-Routine sperren
00678C 2B6F0008068E         move.l  8(A7),$68E(A5)    cdma
006792 3B6F00100684         move.w  16(A7),$684(A5)   cdev
006798 3B6F00120688         move.w  18(A7),$688(A5)   csect
00679E 3B6F00140686         move.w  20(A7),$686(A5)   ctrack
0067A4 3B6F0016068A         move.w  22(A7),$68A(A5)   cside
0067AA 3B6F0018068C         move.w  24(A7),$68C(A5)   ccount
0067B0 3B7C00020672         move.w  #2,$672(A5)       retrycnt
0067B6 43ED06C8             lea     $6C8(A5),A1       dsb0
0067BA 4A6D0684             tst.w   $684(A5)          cdev
0067BE 6704                 beq     $67C4             Drive 0 ?
0067C0 43ED06CC             lea     $6CC(A5),A1       dsb1
0067C4 7E00                 moveq.l #0,D7
0067C6 3E2D068C             move.w  $68C(A5),D7       ccount, Anzahl Sektoren
0067CA E14F                 lsl.w   #8,D7             mal 512
0067CC E34F                 lsl.w   #1,D7
0067CE 206D068E             move.l  $68E(A5),A0       cdma, Start-DMA-Adresse
0067D2 D1C7                 add.l   D7,A0             plus Sektorenlänge
0067D4 2B480692             move.l  A0,$692(A5)       edma, ergibt End-DMA-Adresse
0067D8 4A690000             tst.w   0(A1)             dcurtrack, aktueller Track
0067DC 6A20                 bpl     $67FE             >= 0 ?
0067DE 61000168             bsr     $6948             nein, select
0067E2 42690000             clr.w   0(A1)             Track auf Null setzen
0067E6 610000EC             bsr     $68D4             restore, Kopf auf Track Null
0067EA 6712                 beq     $67FE             ok ?
0067EC 7E0A                 moveq.l #10,D7
0067EE 6172                 bsr     $6862             Track 10 suchen
0067F0 6606                 bne     $67F8             Fehler ?
0067F2 610000E0             bsr     $68D4             restore
0067F6 6706                 beq     $67FE             ok ?
```

```
0067F8 337CFF000000          move.w  #$FF00,0(A1)        recalibrate, Fehler
0067FE 4E75                  rts

************************************************        flopfail, Fehler in Floppy-Routine
e06800 7001                  moveq.l #1,D0               media change auf 'unsure'
006802 61000226              bsr     $6A2A               setzen
006806 302D06A2              move.w  $6A2(A5),D0         currerr
00680A 48C0                  ext.l   D0
00680C 6002                  bra     $6810

************************************************        flopok, fehlerfreie Floppy-Routine
00680E 4280                  clr.l   D0
006810 2F00                  move.l  D0,-(A7)            ok
006812 3CBC0086              move.w  #$86,(A6)           1772 selektieren
006816 3E290000              move.w  0(A1),D7            Tracknummer holen
00681A 610001A8              bsr     $69C4               wdiskctl, an Floppycontroller senden
00681E 3C3C0010              move.w  #$10,D6             seek-Befehl
006822 610000C6              bsr     $68EA               flopcms
006826 303900000684          move.w  $684,D0             cdev, Drive-Nummer
00682C E548                  lsl.w   #2,D0               als Index
00682E 41F900000678          lea     $678,A0             acctim
006834 21AD04BA0000          move.l  $4BA(A5),0(A0,D0.w) 200 Hz Zähler als letztes Zugriffszeit
00683A 0C7900010000004A6     cmp.w   #1,$4A6             nflops
006842 6606                  bne     $684A
006844 216D04BA0004          move.l  $4BA(A5),4(A0)      20 Hz Zähler für anderes Drive
00684A 201F                  move.l  (A7)+,D0            Fehlernummer zurückholen
00684C 4CF978F8000006A4      movem.l $6A4,D3-D7/A3-A6    regsave
006854 42790000043E          clr.w   $43E                flock löschen, vbl-Routine freigeben
00685A 4E75                  rts

************************************************        hseek, Kopf auf Track
00685C 3E3900000686          move.w  $686,D7             ctrack
006862 33FCFFFA000006A2      move.w  #-10,$6A2           currerr auf 'seek error'
00686A 3CBC0086              move.w  #$86,(A6)           Tracknummer übergeben
00686E 61000154              bsr     $69C4               wdsikctl
006872 3C3C0010              move.w  #$10,D6             Seek-Befehl
006876 60000072              bra     $68EA               flopcmds
```

```
*************************************************     ressek, Home und Seek
00687A 33FCFFFA000006A2      move.w  #-10,$6A2        currerr auf 'seek error'
006882 6150                  bsr     $68D4            restore
006884 664C                  bne     $68D2            Fehler ?
006886 42690000              clr.w   0(A1)            curent track auf Null
00688A 3CBC0082              move.w  #$82,(A6)        Trackregister selektieren
00688E 4247                  clr.w   D7               Track Null
006890 61000132              bsr     $69C4            wdiskctl
006894 3CBC0086              move.w  #$86,(A6)        Datenregister selektieren
006898 3E3C0005              move.w  #5,D7            Track 5
00689C 61000126              bsr     $69C4            wdiskctl
0068A0 3C3C0010              move.w  #$10,D6          Seek-Befehl
0068A4 6144                  bsr     $68EA            flopcmds
0068A6 662A                  bne     $68D2            Fehler
0068A8 337C00050000          move.w  #5,0(A1)         Tracknummer auf 5

*************************************************     go2track, Track suchen
0068AE 33FCFFFA000006A2      move.w  #-10,$6A2        currerr auf 'seek error'
0068B6 3CBC0086              move.w  #$86,(A6)        Datenregister selektieren
0068BA 3E2D0686              move.w  $686(A5),D7      ctrack, Tracknummer laden
0068BE 61000104              bsr     $69C4            wdiskctl
0068C2 7C14                  moveq.l #$14,D6          Seek mit Verify Befehl
0068C4 6124                  bsr     $68EA            flopcmds
0068C6 660A                  bne     $68D2            Fehler
0068C8 336D06860000          move.w  $686(A5),0(A1)   ctrack, Tracknummer merken
0068CE CE3C0018              and.b   #$18,D7          RNF, CRC lost data testen
0068D2 4E75                  rts

*************************************************     restore, Track Null suchen
0068D4 4246                  clr.w   D6               restore-Befehl
0068D6 6112                  bsr     $68EA            flopcmds
0068D8 660E                  bne     $68E8            Fehler
0068DA 08070002              btst    #2,D7            Track Null Bit testen
0068DE 0A3C0004              eor.b   #4,SR            Z-Flag invertieren
0068E2 6604                  bne     $68E8            nicht Track Null ?
0068E4 42690000              clr.w   0(A1)            current Track auf Null
0068E8 4E75                  rts
```

```
************************************************      flopcmds
0068EA 30290002              move.w  2(A1),D0        Seek rate
0068EE C03C0003              and.b   #$3,D0          Bit 0 und 1
0068F2 8C00                  or.b    D0,D6           in Befehlswort odern
0068F4 2E3C00040000          move.l  #$40000,D7      Timeout-Zähler
0068FA 3CBC0080              move.w  #$80,(A6)       1772 Register selektieren
0068FE 610000D8              bsr     $69D8           rdiskctl
006902 08000007              btst    #7,D0           motor on ?
006906 6606                  bne     $690E           ja
006908 2E3C00060000          move.l  #$60000,D7      sonst längeres Timeout
00690E 610000AA              bsr     $69BA           wdiskctl6, Befehl in d6 schreiben
006912 5387                  subq.l  #1,D7           Timeout-Zähler dekrementieren
006914 6712                  beq     $6928           abgelaufen ?
006916 08390005FFFFFA01      btst    #5,$FFFFFA01    mfp gpip, 1772 fertig ?
00691E 66F2                  bne     $6912           nein, warten
006920 610000AC              bsr     $69CE           rdiskctl7
006924 4246                  clr.w   D6              ok
006926 4E75                  rts

006928 6104                  bsr     $692E           reset 1772
00692A 7C01                  moveq.l #1,D6           Fehler
00692C 4E75                  rts

************************************************      reset1772, Floppycontroller rücksetzen
00692E 3CBC0080              move.w  #$80,(A6)       Befehslregister selektieren
006932 3E3C00D0              move.w  #$D0,D7         reset-Befehl
006936 6100008C              bsr     $69C4           wdiskctl
00693A 3E3C000F              move.w  #$F,D7          Verzögerungszähler
00693E 51CFFFFE              dbra    D7,$693E        abgelaufen ?
006942 6100008A              bsr     $69CE           rdiskctl, Status lesen
006946 4E75                  rts

************************************************      select, Drive und Seite selektieren
006948 426D0682              clr.w   $682(A5)        deslflg löschen
00694C 302D0684              move.w  $684(A5),D0     cdev, Drivenummer
006950 5200                  addq.b  #1,D0
006952 E308                  lsl.b   #1,D0           Bitnummer berechnen
```

```
006954 806D068A            or.w    $68A(A5),D0         cside, Seite in Bit 0
006958 0A000007            eor.b   #7,D0               Bits für Hardware invertieren
00695C C03C0007            and.b   #7,D0
006960 6132                bsr     $6994               Bits in Sound-Chip setzen
006962 3CBC0082            move.w  #$82,(A6)           Track-Register selektieren
006966 3E290000            move.w  0(A1),D7            Tracknummer holen
00696A 6158                bsr     $69C4               wdskctl
00696C 422D069C            clr.b   $69C(A5)            tmpdma, Bit 24-31 löschen
006970 3CBC0084            move.w  #$84,(A6)           Sektor-Register selektieren
006974 3E2D0688            move.w  $688(A5),D7         csect, Sektornummer holen
006978 614A                bsr     $69C4               wdiskctl
00697A 13ED0691FFFF860D    move.b  $691(A5),$FFFF860D  dmalow
006982 13ED0690FFFF860B    move.b  $690(A5),$FFFF860B  dmamid
00698A 13ED068FFFFF8609    move.b  $68F(A5),$FFFF8609  dmahigh
006992 4E75                rts

*****************************************************  setporta, Port A im Sound-Chip setzen
006994 40E7                move.w  SR, -(A7)           Status retten
006996 007C0700            or.w    #$700,SR            IPL 7
00699A 13FC000EFFFF8800    move.b  #$E,$FFFF8800       giselect
0069A2 1239FFFF8800        move.b  $FFFF8800,D1        Port A lesen
0069A8 1401                move.b  D1,D2               und nach D2
0069AA C23C00F8            and.b   #$F8,D1             Bits 0-2 löschen
0069AE 8200                or.b    D0,D1               neue Bits setzen
0069B0 13C1FFFF8802        move.b  D1,$FFFF8802        Ergebnis auf Port A schreiben
0069B6 46DF                move.w  (A7)+,SR            Status zurückholen
0069B8 4E75                rts

*****************************************************  wdiskct6
0069BA 6124                bsr     $69E0               Verzögerungsschleife für Diskcontroller
0069BC 33C6FFFF8604        move.w  D6,$FFFF8604        dskctl
0069C2 601C                bra     $69E0               Verzögerungsschleife für Diskcontroller
```

```
*****************************************************   wdiskctl
0069C4 611A                bsr     $69E0              Verzögerungsschleife für Diskcontroller
0069C6 33C7FFFF8604        move.w  D7,$FFFF8604       dskctl
0069CC 6012                bra     $69E0              Verzögerungsschleife für Diskcontroller

*****************************************************   rdiskct7
0069CE 6110                bsr     $69E0              Verzögerungsschleife für Diskcontroller
0069D0 3E39FFFF8604        move.w  $FFFF8604,D7       dskctl
0069D6 6008                bra     $69E0              Verzögerungsschleife für Diskcontroller

*****************************************************   rdiskctl
0069D8 6106                bsr     $69E0              Verzögerungsschleife für Diskcontroller
0069DA 3039FFFF8604        move.w  $FFFF8604,D0       dskctl
0069E0 40E7                move.w  SR, -(A7)          Status retten
0069E2 3F07                move.w  D7,-(A7)           D7 retten
0069E4 3E3C0020            move.w  #$20,D7            Zähler
0069E8 51CFFFFE            dbra    D7,$69E8           Verzögerungsschleife
0069EC 3E1F                move.w  (A7)+,D7           D7 zurück
0069EE 46DF                move.w  (A7)+,SR           Status zurück
0069F0 4E75                rts

*************************************************     change, Diskettenwechssl testen
0069F2 0C790001000004A6    cmp.w   #1,$4A6            nflops
0069FA 662C                bne     $6A28              keine oder 2 Floppies, fertig
0069FC 302F0010            move.w  16(A7),D0          Drive-Nummer
006A00 B07900004692        cmp.w   $4692,D0           gleich Disketten-Nummer ?
006A06 671C                beq     $6A24              ja
006A08 3F00                move.w  D0,-(A7)           Drive-Nummer
006A0A 3F3CFFEF            move.w  #-17,-(A7)         'insert disk'
006A0E 6100EB4C            bsr     $555C              critical error handler
006A12 584F                addq.w  #4,A7              Stackpointer korrigieren
006A14 33FCFFFF00000676    move.w  #$FFFF,$676        wplatch, Status für beide Drives unsicher
006A1C 33EF001000004692    move.w  16(A7),$4692       Diskettennummer merken
006A24 426F0010            clr.w   16(A7)             Drivenummer auf Null
006A28 4E75                rts
```

```
************************************************  setdmode, Drive-Change-Modus setzen
006A2A 41F900003E2A        lea     $3E2A,A0       Diskmode-Tabelle
006A30 1F00                move.b  D0,-(A7)       Modus merken
006A32 302D0684            move.w  $684(A5),D0    cdev, Drivenummer holen
006A36 119F0000            move.b  (A7)+,0(A0,D0.w)  Drive-Mode setzen
006A3A 4E75                rts

************************************************  dskf, Diskflags
006A3C AE                  dc.b    %10101110
006A3D D6                  dc.b    %11010110
006A3E 8C                  dc.b    %10001100
006A3F 17                  dc.b    %00010111
006A40 FB                  dc.b    %11111011
006A41 80                  dc.b    %10000000
006A42 6A                  dc.b    %01101010
006A43 2B                  dc.b    %00101011
006A44 A6                  dc.b    %10100110
006A45 00                  dc.b    0

************************************************  jdostime, IKBD-Format in DOS-Format
006A46 4BF900000000        lea     $0,A5          A5 löschen
006A4C 41ED0A43            lea     $A43(A5),A0    Zeiger auf Uhrzeit-Puffer
006A50 610000DE            bsr     $6B30          bcdbin
006A54 04000050            sub.b   #80,D0         Offset von 80 abziehen
006A58 1400                move.b  D0,D2          Jahr
006A5A E982                asl.l   #4,D2          in Position schieben
006A5C 610000D2            bsr     $6B30          bcdbin
006A60 D400                add.b   D0,D2          Monat addieren
006A62 EB82                asl.l   #5,D2          und in Position schieben
006A64 610000CA            bsr     $6B30          bcdbin
006A68 D400                add.b   D0,D2          Tag addieren
006A6A EB82                asl.l   #5,D2          und in Position schieben
006A6C 610000C2            bsr     $6B30          bcdbin
006A70 D400                add.b   D0,D2          Stunde addieren
006A72 ED82                asl.l   #6,D2          und in Position schieben
006A74 610000BA            bsr     $6B30          bcdbin
006A78 D400                add.b   D0,D2          Minute addieren
```

```
006A7A EB82                 asl.l   #5,D2           und in Position schieben
006A7C 610000B2             bsr     $6B30           bcdbin
006A80 E208                 lsr.b   #1,D0           2-Sekunden-Auflösung
006A82 D400                 add.b   D0,D2           und Sekunden addieren
006A84 2B420A4C             move.l  D2,$A4C(A5)     neue Zeit abspeichern
006A88 1B7C00000A8E         move.b  #$0,$A8E(A5)    Handshakeflag löschen
006A8E 4E75                 rts

************************************************  gettime, aktuelle Uhrzeit und Datum holen
006A90 1B7CFFFF0A8E         move.b  #-1,$A8E(A5)    Handshakeflag für Zeit anfordern
006A96 123C001C             move.b  #$1C,D1         get time of day Befehl
006A9A 61000234             bsr     $6CD0           senden
006A9E 4A2D0A8E             tst.b   $A8E(A5)        neue Zeit schon angekommen ?
006AA2 66FA                 bne     $6A9E           nein, warten
006AA4 202D0A4C             move.l  $A4C(A5),D0     Zeit nach d0 holen
006AA8 4E75                 rts

************************************************  settime, Uhrzeit und Datum setzen
006AAA 2B6F00040A50         move.l  4(A7),$A50(A5)  Zeit übergeben

************************************************  ikbdtime
006AB0 41F900000A5A         lea     $A5A,A0         Zeiger auf Ende des Time-Puffers
006AB6 242D0A50             move.l  $A50(A5),D2     umzuwandelnde Zeit holen
006ABA 1002                 move.b  D2,D0           nach d0
006ABC 0200001F             and.b   #31,D0          Bit 0-4 isolieren, Sekunden
006AC0 E300                 asl.b   #1,D0           2-Sekunden-Auflösung
006AC2 6154                 bsr     $6B18           konvertieren
006AC4 EA8A                 lsr.l   #5,D2           Minuten
006AC6 1002                 move.b  D2,D0
006AC8 0200003F             and.b   #63,D0          Bit 0-5 isolieren
006ACC 614A                 bsr     $6B18           konvertieren
006ACE EC8A                 lsr.l   #6,D2           Stunden
006AD0 1002                 move.b  D2,D0
006AD2 0200001F             and.b   #31,D0          Bit 0-4 isoliren
006AD6 6140                 bsr     $6B18           konvertieren
006AD8 EA8A                 lsr.l   #5,D2           Tag
006ADA 1002                 move.b  D2,D0
```

```
006ADC 0200001F          and.b   #31,D0          Bit 0-4 isolieren
006AE0 6136              bsr     $6B18           konvertieren
006AE2 EA8A              lsr.l   #5,D2           Monat
006AE4 1002              move.b  D2,D0
006AE6 0200000F          and.b   #15,D0          Bit 0-3 isolieren
006AEA 612C              bsr     $6B18           konvertieren
006AEC E88A              lsr.l   #4,D2           Jahr
006AEE 1002              move.b  D2,D0
006AF0 0200007F          and.b   #$7F,D0         Bit 0-7 isolieren
006AF4 6122              bsr     $6B18           konvertieren
006AF6 06100080          add.b   #$80,(A0)       Offset addieren
006AFA 123C001B          move.b  #$1B,D1         Set Time Of Day Befehl
006AFE 610001D0          bsr     $6CD0           an IKBD senden
006B02 7605              moveq.l #5,D3           Anzahl der Bytes zu senden
006B04 45F900000A54      lea     $A54,A2         Adresse des Parameterblocks
006B0A 610001E4          bsr     $6CF0           senden
006B0E 123C001C          move.b  #$1C,D1         Get Time Of Day Befehl
006B12 610001BC          bsr     $6CD0           an IKBD senden
006B16 4E75              rts

************************************************ binbcd, Byte nach BCD konverieren
006B18 7200              moveq.l #0,D1           Zehnerzähler
006B1A 760A              moveq.l #10,D3          10 laden
006B1C 9003              sub.b   D3,D0           10 abziehen
006B1E 6B04              bmi     $6B24
006B20 5201              addq.b  #1,D1           Zehnerzähler erhöhen
006B22 60F8              bra     $6B1C
006B24 0600000A          add.b   #10,D0          Einerstelle generieren
006B28 E901              asl.b   #4,D1           Zehner in oberes Nibble
006B2A D001              add.b   D1,D0           und Einer addieren
006B2C 1100              move.b  D0,-(A0)        Ergebnis in Puffer schreiben
006B2E 4E75              rts
```

```
************************************************      bcdbin, BCD nach binär konvertieren
006B30 7000               moveq.l #0,D0
006B32 1010               move.b  (A0),D0             BCD-Byte
006B34 E808               lsr.b   #4,D0               Zehnerstelle
006B36 E308               lsl.b   #1,D0
006B38 1200               move.b  D0,D1
006B3A E500               asl.b   #2,D0
006B3C D001               add.b   D1,D0
006B3E 1218               move.b  (A0)+,D1
006B40 0241000F           and.w   #15,D1              Einerstelle isolieren
006B44 D041               add.w   D1,D0               plus Zehnerstelle
006B46 4E75               rts

************************************************      midiost, MIDI Output Status
006B48 70FF               moveq.l #-1,D0              Default auf ok
006B4A 1439FFFFFC04       move.b  $FFFFFC04,D2        MIDI-ACIA Status lesen
006B50 08020001           btst    #1,D2               und testen
006B54 6602               bne     $6B58               ok
006B56 7000               moveq.l #0,D0               Status nicht ok
006B58 4E75               rts

************************************************      midiwc, Zeichen an MIDI ausgeben
006B5A 322F0006           move.w  6(A7),D1            Zeichen holen
006B5E 43F9FFFFFC04       lea     $FFFFFC04,A1        Zeiger auf MIDI-ACIA
006B64 14290000           move.b  0(A1),D2            MIDI-Status holen
006B68 08020001           btst    #1,D2               testen
006B6C 67F6               beq     $6B64               warten bis ok
006B6E 13410002           move.b  D1,2(A1)            Byte an ACIA ausgeben
006B72 4E75               rts

************************************************      midiws, String an MIDI senden
006B74 7600               moveq.l #0,D3
006B76 362F0004           move.w  4(A7),D3            Länge des Strings -1
006B7A 246F0006           move.l  6(A7),A2            Adresse des Strings
006B7E 121A               move.b  (A2)+,D1            Byte von String holen
006B80 61DC               bsr     $6B5E               an MIDI ausgeben
006B82 51CBFFFA           dbra    D3,$6B7E            nächstes Byte
```

```
006B86 4E75                rts

************************************************     midstat, MIDI Empfänger-Status
006B88 41ED0A00            lea     $A00(A5),A0      iorec für MIDI
006B8C 43F9FFFFFC04        lea     $FFFFFC04,A1     Zeiger auf MIDI-ACIA
006B92 70FF                moveq.l #-1,D0           ok als Default
006B94 45E80006            lea     6(A0),A2         Head Index
006B98 47E80008            lea     8(A0),A3         Tail Index
006B9C B54B                cmpm.w  (A3)+,(A2)+      nicht unterbrechbarer Test
006B9E 6602                bne     $6BA2            nicht gleich
006BA0 7000                moveq.l #0,D0            nicht ok
006BA2 4E75                rts

************************************************     midin, Zeichen von MIDI holen
006BA4 61E2                bsr     $6B88            midstat
006BA6 4A40                tst.w   D0               Zeichen bereit ?
006BA8 67FA                beq     $6BA4            nein
006BAA 40E7                move.w  SR, -(A7)        Status retten
006BAC 007C0700            or.w    #$700,SR         IPL 7, Interrupts sperren
006BB0 32280006            move.w  6(A0),D1         Head Index
006BB4 B2680008            cmp.w   8(A0),D1         mit Tail Index vergleichen
006BB8 6716                beq     $6BD0

006BBA 5241                addq.w  #1,D1            Head Index erhöhen
006BBC B2680004            cmp.w   4(A0),D1         größer Puffergröße ?
006BC0 6502                bcs     $6BC4            nein
006BC2 7200                moveq.l #0,D1            wieder am Pufferbeginn anfangen
006BC4 22680000            move.l  0(A0),A1         Zeiger auf MIDI-Puffer
006BC8 10311000            move.b  0(A1,D1.w),D0    Zeichen aus Puffer holen
006BCC 31410006            move.w  D1,6(A0)         neuen Head Index merken
006BD0 46DF                move.w  (A7)+,SR         Status zurückholen
006BD2 4E75                rts

************************************************     lstout, Zeichen an Centronics ausgeben
006BD4 242D04BA            move.l  $4BA(A5),D2      200 Hz Zähler
006BD8 94AD0A80            sub.l   $A80(A5),D2      minus Printer Timeout
006BDC 0C82000003E8        cmp.l   #1000,D2         länger als 5 Sekunden gewartet ?
006BE2 6518                bcs     $6BFC            ja, Time out
```

```
006BE4 242D04BA          move.l  $4BA(A5),D2         200 Hz-Zähler, Startzeit für dieses Zeichen
006BE8 6172              bsr     $6C5C               Buys-Status holen
006BEA 4A40              tst.w   D0                  und testen
006BEC 6618              bne     $6C06               Drucker bereit ?
006BEE 262D04BA          move.l  $4BA(A5),D3         200 Hz Zähler
006BF2 9682              sub.l   D2,D3               minus Startzeit
006BF4 0C8300001770      cmp.l   #6000,D3            30 Sekunden ?
006BFA 6DEC              blt     $6BE8               noch nicht erreicht
006BFC 7000              moveq.l #0,D0               Flag für Time out
006BFE 2B6D04BA0A80      move.l  $4BA(A5),$A80(A5)   200 Hz Zähler als letzte Timeout-Zeit
006C04 4E75              rts

006C06 40C3              move.w  SR, D3              Status retten
006C08 007C0700          or.w    #$700,SR            IPL 7, keine Interrupts
006C0C 7207              moveq.l #7,D1               Mixer
006C0E 61000E28          bsr     $7A38               Register selektieren
006C12 00000080          or.b    #$80,D0             Port B auf Ausgabe
006C16 7287              moveq.l #$87,D1             Write enable
006C18 61000E1E          bsr     $7A38
006C1C 46C3              move.w  D3,SR               Status zurückholen
006C1E 302F0006          move.w  6(A7),D0            auszugebendes Zeichen
006C22 728F              moveq.l #$8F,D1             auf Port B schreiben
006C24 61000E12          bsr     $7A38
006C28 610C              bsr     $6C36               Strobe Low
006C2A 6104              bsr     $6C30               Strobe High
006C2C 70FF              moveq.l #-1,D0              Flag für ok
006C2E 4E75              rts

**********************************************       Strobe High
006C30 7420              moveq.l #$20,D2             Bit 5
006C32 60000E46          bra     $7A7A               setzen

**********************************************       Strobe low
006C36 74DF              moveq.l #$DF,D2             Bit 5
006C38 60000E66          bra     $7AA0               löschen
```

```
**********************************************
006C3C 7207              moveq.l #7,D1          listin, Zeichen vom Parallelport holen
006C3E 61000DF8          bsr     $7A38          Mixer
006C42 0200007F          and.b   #$7F,D0        Register selektieren
006C46 7287              moveq.l #$87,D1        Port B auf Eingabe
006C48 61000DEE          bsr     $7A38
006C4C 61E2              bsr     $6C30          Strobe High = nicht busy
006C4E 610C              bsr     $6C5C          Parallelport Status holen
006C50 4A40              tst.w   D0
006C52 66FA              bne     $6C4E          warten bis Zeichen angekommen ist
006C54 61E0              bsr     $6C36          Strobe Low = busy
006C56 720F              moveq.l #15,D1         Port B selektieren
006C58 60000DDE          bra     $7A38          Byte vom Port lesen

**********************************************
006C5C 41F9FFFFFA01      lea     $FFFFFA01,A0   lstostat, Parallelport Status
006C62 70FF              moveq.l #-1,D0         Adresse des MFP
006C64 082800000000      btst    #0,0(A0)       Default auf ok
006C6A 6702              beq     $6C6E          Busy testen
006C6C 7000              moveq.l #0,D0          ok
006C6E 4E75              rts                    noch busy

**********************************************
006C70 41ED09D0          lea     $9D0(A5),A0    auxistat, RS232 Eingabestatus
006C74 70FF              moveq.l #-1,D0         iorec für RS232
006C76 45E80006          lea     6(A0),A2       Default auf ok
006C7A 47E80008          lea     8(A0),A3       Head Index
006C7E B54B              cmpm.w  (A3)+,(A2)+    Tail Index
006C80 6602              bne     $6C84          Puffer leer ?
006C82 7000              moveq.l #0,D0          nein
006C84 4E75              rts                    kein Zeichen da

**********************************************
006C86 61E8              bsr     $6C70          auxin, Zeichen von RS232 holen
006C88 4A40              tst.w   D0             auxin, Zeichen bereit ?
006C8A 67FA              beq     $6C86          nein, warten
006C8C 610005D8          bsr     $7266          rs232get, Zeichen holen
```

```
006C90 024000FF            and.w   #$FF,D0          Bit 0-7 isolieren
006C94 4E75                rts

************************************************  auxostat, RS232 Ausgabestatus
006C96 41ED09D0            lea     $9D0(A5),A0     iorec für RS232
006C9A 70FF                moveq.l #-1,D0          Default auf ok
006C9C 34280016            move.w  22(A0),D2       Tail Index
006CA0 6100087C            bsr     $751E           Test auf wrap arround
006CA4 B4680014            cmp.w   20(A0),D2       mit Head Index vergleichen
006CA8 6602                bne     $6CAC           ungleich
006CAA 7000                moveq.l #0,D0           kein Platz mehr im Puffer
006CAC 4E75                rts

************************************************  auxout, RS232 Ausgaberoutine
006CAE 322F0006            move.w  6(A7),D1        Datenbyte holen
006CB2 61000556            bsr     $720A           rs232put, und ausgeben
006CB6 65F6                bcs     $6CAE           nicht losgeworden, nochmal versuchen
006CB8 4E75                rts

************************************************  ikbdost, IKBD Ausgabestatus
006CBA 70FF                moveq.l #-1,D0          Default auf ok
006CBC 1439FFFFFC00        move.b  $FFFFFC00,D2    IKBD ACIA Status
006CC2 08020001            btst    #1,D2           testen
006CC6 6602                bne     $6CCA           ok
006CC8 7000                moveq.l #0,D0           nicht bereit
006CCA 4E75                rts

************************************************  ikbdwc, Zeichen an IKBD senden
006CCC 322F0006            move.w  6(A7),D1        Byte holen
006CD0 43F9FFFFFC00        lea     $FFFFFC00,A1    Adresse der Keyboard ACIA
006CD6 14290000            move.b  0(A1),D2        Status holen
006CDA 08020001            btst    #1,D2           bereit ?
006CDE 67F6                beq     $6CD6           nein, warten
006CE0 13410002            move.b  D1,2(A1)        Zeichen ausgeben
006CE4 4E75                rts
```

```
************************************************
006CE6 7600             moveq.l #0,D3
006CE8 362F0004         move.w  4(A7),D3
006CEC 246F0006         move.l  6(A7),A2
006CF0 121A             move.b  (A2)+,D1
006CF2 61DC             bsr     $6CD0
006CF4 51CBFFFA         dbra    D3,$6CF0
006CF8 4E75             rts

************************************************
006CFA 41ED09F2         lea     $9F2(A5),A0
006CFE 70FF             moveq.l #-1,D0
006D00 45E80006         lea     6(A0),A2
006D04 47E80008         lea     8(A0),A3
006D08 B54B             cmpm.w  (A3)+,(A2)+
006D0A 6602             bne     $6D0E
006D0C 7000             moveq.l #0,D0
006D0E 4E75             rts

************************************************
006D10 61E8             bsr     $6CFA
006D12 4A40             tst.w   D0
006D14 67FA             beq     $6D10
006D16 40E7             move.w  SR, -(A7)
006D18 007C0700         or.w    #$700,SR
006D1C 32280006         move.w  6(A0),D1
006D20 B2680008         cmp.w   8(A0),D1
006D24 671C             beq     $6D42
006D26 5441             addq.w  #2,D1
006D28 B2680004         cmp.w   4(A0),D1
006D2C 6502             bcs     $6D30
006D2E 7200             moveq.l #0,D1
006D30 22680000         move.l  0(A0),A1
006D34 7000             moveq.l #0,D0
006D36 30311000         move.w  0(A1,D1.w),D0
006D3A 31410006         move.w  D1,6(A0)
006D3E E188             lsl.l   #8,D0
```

```
ikbdws, String an Keyboard senden

Anzahl der Zeichen -1
Adresse des Strings
Byte von String holen
und ausgeben
nächstes Byte


constat, Tastatur Eingabestatus
iorec für Keyboard
Default auf ok
Head Index
Tail Index
Puffer leer ?
nein
kein Zeichen da


conin, Zeichen von Tastaur holen
constat, Taste gedrückt ?

nein, warten
Status retten
IPL 7, Interrupts sperren
Head Index
mit Tail Index vergleichen

Head Index + 2
größer gleich Puffergröße ?
nein
Pufferzeiger wieder auf Start
Zeiger auf Tastaturpuffer

Zeichen und Scancode holen
neuen Head Index merken
Scancode nach Bit 16-23
```

```
006D40 E048              lsr.w   #8,D0           ASCII-Code nach Bit 0-7
006D42 46DF              move.w  (A7)+,SR        Status zurückholen
006D44 4E75              rts

************************************************  conoutst, Consol Ausgabestatus
006D46 70FF              moveq.l #-1,D0          immer ok
006D48 4E75              rts

************************************************  ringbel, Ton nach CTRL G
006D4A 082D00020484      btst    #2,$484(A5)     Ton enabled ?
006D50 670E              beq     $6D60           nein
006D52 2B7C00007D5A0A86  move.l  #$7D5A,$A86(A5) Zeiger auf Soundtabelle für Bell
006D5A 1B7C00000A8A      move.b  #$0,$A8A(A5)    Soundtimer starten
006D60 4E75              rts

************************************************  Tastaturtabelle, ungeshiftet
006D62 001B313233343536  dc.b    $00,esc,'1','2','3','4','5','6'
006D6A 373839309E270809  dc.b    '7','8','9','0','ß','´',bs ,tab
006D72 71776572747A7569  dc.b    'q','w','e','r','t','z','u','i'
006D7A 6F70812B0D006173  dc.b    'o','p','ü','+',cr, $00,'a','s'
006D82 646667686A6B6C94  dc.b    'd','f','g','h','j','k','l','ö'
006D8A 8423007E79786376  dc.b    'ä','#',$00,'~','y','x','c','v'
006D92 626E6D2C2E2D0000  dc.b    'b','n','m',',','.','-',$00,$00
006D9A 0020000000000000  dc.b    $00,' ',$00,$00,$00,$00,$00,$00
006DA2 0000000000000000  dc.b    $00,$00,$00,$00,$00,$00,$00,$00
006DAA 00002D0000002B00  dc.b    $00,$00,'-',$00,$00,$00,'+',$00
006DB2 0000007F00000000  dc.b    $00,$00,$00,del,$00,$00,$00,$00
006DBA 0000000000000000  dc.b    $00,$00,$00,$00,$00,$00,$00,$00
006DC2 3C000028292F2A37  dc.b    '<',$00,$00,'8',')','/','*','7'
006DCA 3839343536313233  dc.b    '8','9','4','5','6','1','2','3'
006DD2 302E0D0000000000  dc.b    '0','.',cr, $00,$00,$00,$00,$00
006DDA 0000000000000000  dc.b    $00,$00,$00,$00,$00,$00,$00,$00
```

```
**************************************************    Tastaturtabelle, geshiftet
006DE2 001B2122DD242526     dc.b    $00,esc,'!','"','@','$','%','&'
006DEA 2F28293D3F600809     dc.b    '/','(',')','=','?','`',bs, tab
006DF2 51574552545A5549     dc.b    'Q','W','E','R','T','Z','U','I'
006DFA 4F509A2A0D004153     dc.b    'O','P','Ü','*',cr, $00,'A','S'
006E02 444647484A5B4C99     dc.b    'D','F','G','H','J','K','L','Ö'
006E0A 8E5E007C59584356     dc.b    'Ä','^',$00,'|','Y','X','C','V
006E12 424E4D3B3A5F0000     dc.b    'B','N','M',';',':','_',$00,$00
006E1A 0020000000000000     dc.b    $00,' ',$00,$00,$00,$00,$00,$00
006E22 0000000000000037     dc.b    $00,$00,$00,$00,$00,$00,$00,'7'
006E2A 38002D3400362B00     dc.b    '8',$00,'-','4',$00,'6','+',$00
006E32 3200307F00000000     dc.b    '2',$00,'0',del,$00,$00,$00,$00
006E3A 0000000000000000     dc.b    $00,$00,$00,$00,$00,$00,$00,$00
006E42 3E000028292F2A37     dc.b    '>',$00,$00,'(',')','/','*','7'
006E4A 3839343536313233     dc.b    '8','9','4','5','6','1','2','3'
006E52 302E0D0000000000     dc.b    '0','.',cr ,$00,$00,$00,$00,$00
006E5A 0000000000000000     dc.b    $00,$00,$00,$00,$00,$00,$00,$00

**************************************************    Tastaturtabelle, Caps Lock
006E62 001B313233343536     dc.b    $00,esc,'1','2','3','4','5','6'
006E6A 373839309E270809     dc.b    '7','8','9','0','ß','`',bs, tab
006E72 51574552545A5549     dc.b    'Q','W','E','R','T','Z','U','I'
006E7A 4F509A2B0D004153     dc.b    'O','P','Ü','+',cr, $00,'A','S'
006E82 444647484A4B4C99     dc.b    'D','F','G','H','J','K','L','Ö'
006E8A 8E23007E59684356     dc.b    'Ä','#',$00,'~','Y','X','C','V'
006E92 424E4D2C2E2D0000     dc.b    'B','N','M',',','.','-',$00,$00
006E9A 0020000000000000     dc.b    $00,' ',$00,$00,$00,$00,$00,$00
006E9E 0000000000000000     dc.b    $00,$00,$00,$00,$00,$00,$00,$00
006EA2 0000000000000000     dc.b    $00,$00,$00,$00,$00,$00,$00,$00
006EAA 00002D0000002B00     dc.b    $00,$00,'-',$00,$00,$00,'+',$00
006EB2 0000007F00000000     dc.b    $00,$00,$00,del,$00,$00,$00,$00
006EBA 0000000000000000     dc.b    $00,$00,$00,$00,$00,$00,$00,$00
006EC2 3C000028292F2A37     dc.b    '<',$00,$00,'(',')','/','*','7'
006ECA 3839343536313233     dc.b    '8','9','4','5','6','1','2','3'
006ED2 302E0D0000000000     dc.b    '0','.',cr ,$00,$00,$00,$00,$00
006EDA 0000000000000000     dc.b    $00,$00,$00,$00,$00,$00,$00,$00
```

```
************************************************
006EE2 41F9FFFFFA01          lea     $FFFFFA01,A0        Adresse des MFP
006EE8 7000                  moveq.l #0,D0               Register mit Null initialisieren
006EEA 01C80000              movep.l D0,0(A0)            gpip bis iera
006EEE 01C80008              movep.l D0,8(A0)            ierb bis isrb
006EF2 01C80010              movep.l D0,$10(A0)          isrb bis vr
006EF6 117C00480016          move.b  #$48,22(A0)         MFP Nonautovektornr auf $48, s-Bit setzen
006EFC 3B7C11110A84          move.w  #$1111,$A84(A5)
006F02 3B7C00140442          move.w  #20,$442(A5)        timer ms auf 20 Millisekunden
006F08 7002                  moveq.l #2,D0               Timer C selektieren
006F0A 7250                  moveq.l #80,D1              /64 für 200 Hz
006F0C 343C00C0              move.w  #$C0,D2             192
006F10 6100017E              bsr     $7090               Timer und Interruptvektor initialisieren
006F14 45F900007C5C          lea     $7C5C,A2            Timer C Interruptroutine
006F1A 7005                  moveq.l #5,D0               Timer C Interruptnummer
006F1C 61000228              bsr     $7146               initint, Interrupt installieren
006F20 7003                  moveq.l #3,D0               Timer D selektieren
006F22 7201                  moveq.l #1,D1               /4 für 9600 Baud
006F24 7402                  moveq.l #2,D2               9600 Baud
006F26 61000168              bsr     $7090               Timer und Interruptvektor initialisieren
006F2A 203C00980101          move.l  #$980101,D0         $00, $98, $01, $01
006F30 01C80026              movep.l D0,$26(A0)          nach scr, ucr, rsr, tsr
006F34 61000B3A              bsr     $7A70               DTR on
006F38 61000B2E              bsr     $7A68               RTS on
006F3C 41ED09D0              lea     $9D0(A5),A0         Zeiger auf iorec für RS232
006F40 43F90000705E          lea     $705E,A1            Startdaten für iorec
006F46 7021                  moveq.l #33,D0              34 Bytes
006F48 610000EC              bsr     $7036               ins RAM kopieren
006F4C 41ED0A00              lea     $A00(A5),A0         Zeiger auf iorec für MIDI
006F50 43F900007050          lea     $7050,A1            Startdaten für iorec
006F56 700D                  moveq.l #13,D0              14 Bytes
006F58 610000DC              bsr     $7036               ins RAM kopieren
006F5C 203C0000759C          move.l  #$759C,D0           Keyboard und MIDI Error-Vektor
006F62 2B400A12              move.l  D0,$A12(A5)         Zeiger auf Error-Routine Keyboard
006F66 2B400A16              move.l  D0,$A16(A5)         Zeiger auf Error-Routine MIDI
006F6A 2B7C000079C60A0E      move.l  #$79C6,$A0E(A5)     MIDI Interrupt Vektor
006F72 2B7C000075580A2A      move.l  #$7558,$A2A(A5)
```

```
006F7A 2B7C000075680A2E    move.l  #$7568,$A2E(A5)
006F82 13FC0003FFFFFC04    move.b  #3,$FFFFFC04        MIDI-ACIA Master-Reset
006F8A 13FC0095FFFFFC04    move.b  #$95,$FFFFFC04      /16, 8 Bit, 1 Stopbit, no Parity
006F92 1B7C00070484        move.b  #7,$484(A5)         Keyclick, Repeat und Bell enable
006F98 2B7C00006A460A22    move.l  #$6A46,$A22(A5)     clockvec
006FA0 203C00007034        move.l  #$7034,D0           joyvec & statvec
006FA6 2B400A1A            move.l  D0,$A1A(A5)
006FAA 2B400A1E            move.l  D0,$A1E(A5)
006FAE 2B400A26            move.l  D0,$A26(A5)
006FB2 7000                moveq.l #0,D0               Sound-Variablen löschen
006FB4 2B400A86            move.l  D0,$A86(A5)         Sound Pointer
006FB8 1B400A8A            move.b  D0,$A8A(A5)         Delay-Timer
006FBC 1B400A8B            move.b  D0,$A8B(A5)         Temp value
006FC0 2B400A80            move.l  D0,$A80(A5)         Printer Timeout
006FC4 6100FC6A            bsr     $6C30               Strobe auf High
006FC8 1B7C000F0A7E        move.b  #$F,$A7E(A5)
006FCE 1B7C00020A7F        move.b  #2,$A7F(A5)
006FD4 41ED09F2            lea     $9F2(A5),A0         Zeiger auf iorec für Keyboard
006FD8 43F900007042        lea     $7042,A1            Startdaten für iorec
006FDE 700D                moveq.l #13,D0              14 Bytes
006FE0 6154                bsr     $7036               ins RAM kopieren
006FE2 61000C0E            bsr     $7BF2               Zeiger auf BIOS-Tastaturtabelle
006FE6 13FC0003FFFFFC00    move.b  #3,$FFFFFC00        Reset Keyboard ACIA
006FEE 13FC0096FFFFFC00    move.b  #$96,$FFFFFC00      /64, 8 Bit, 1 Stopbit, no Parity
006FF6 267C00007080        move.l  #$7080,A3           Zeiger auf MFP Interuptvektoren
006FFC 7203                moveq.l #3,D1               4 Vektoren initialisieren
006FFE 2401                move.l  D1,D2
007000 2001                move.l  D1,D0               Interrupt Nummer
007002 06000009            add.b   #9,D0               plus Offset
007006 E582                asl.l   #2,D2
007008 24732000            move.l  0(A3,D2.w),A2       Vektor aus Tabelle holen
00700C 61000138            bsr     $7146               initint, Interrupt installieren
007010 51C9FFEC            dbra    D1,$6FFE            nächsten Vektor
007014 45ED752A            lea     $752A(A5),A2        MIDI und Keyboard Vektor
007018 7006                moveq.l #6,D0               Vektornummer 6
00701A 6100012A            bsr     $7146               initint, Interrupt installieren
00701E 45ED73C0            lea     $73C0(A5),A2        CTS Interruptroutine
```

```
007022 7002              moveq.l #2,D0          Vektornummer 2
007024 61000120          bsr     $7146          initint, Interrupt installieren
007028 247C0000703E      move.l  #$703E,A2      Zeiger auf Initdaten für IKBD
00702E 7603              moveq.l #3,D3          4 Bytes
007030 6100FCBE          bsr     $6CF0          String an IKBD senden
007034 4E75              rts

007036 10D9              move.b  (A1)+,(A0)+    Blockmove
007038 51C8FFFC          dbra    D0,$7036       nächstes Byte
00703C 4E75              rts

************************************************
00703E 8001121A          dc.b    $80,$01,$12,$1A  Reset Keyboard, disable Mouse und Joystick

************************************************  iorec für Keyboard
007042 000008D0          dc.l    $8D0           Keyboard Puffer
007046 0080              dc.w    128            Länge des Keyboard Puffers
007048 0000              dc.w    0              Head Index
00704A 0000              dc.w    0              Tail Index
00704C 0020              dc.w    32             Low water mark, 1/4 Pufferlänge
00704E 0060              dc.w    96             High water mark, 3/4 Pufferlänge

************************************************  iorec für MIDI
007050 00000950          dc.l    $950           MIDI Puffer
007054 0080              dc.w    128            Länge des MIDI-Puffers
007056 0000              dc.w    0              Head Index
007058 0000              dc.w    0              Tail Index
00705A 0020              dc.w    32             Low water mark, 1/4 Pufferlänge
00705C 0060              dc.w    96             High water mark, 3/4 Pufferlänge

************************************************  iorec für RS232
00705E 000006D0          dc.l    $6D0           RS232 Eingabepuffer
007062 0100              dc.w    256            Länge des Eingabepuffers
007064 0000              dc.w    0              Head Index
007066 0000              dc.w    0              Tail Index
007068 0040              dc.w    64             Low water mark, 1/4 Pufferlänge
00706A 00C0              dc.w    192            High water merk, 3/4 Pufferlänge
```

```
00706C 000007D0          dc.l    $7D0        RS232 Ausgabepuffer
007070 0100              dc.w    256         Länge des Ausgabepuffers
007072 0000              dc.w    0           Head Index
007074 0000              dc.w    0           Tail Index
007076 0040              dc.w    64          Low water mark, 1/4 Pufferlänge
007078 00C0              dc.w    192         High water merk, 3/4 Pufferlänge
00707A 00                dc.b    0           rsrbyte, Receiver Status
00707B 00                dc.b    0           tsrbyte, Transmitterstatus
00707C 00                dc.b    0           rxoff,
00707D 00                dc.b    0           txoff
00707E 01                dc.b    1           rsmode, XON/XOFF-Modus
00707F 00                dc.b    0           filler

************************************************

007080 00007426          dc.l    $7426       Interruptvektoren für MFP
007084 00007374          dc.l    $7374       #9,  Transmitter Error
007088 00007408          dc.l    $7408       #10, Transmitter Interrupt
00708C 000072C0          dc.l    $72C0       #11, Receiver Error
                                             #12, Receiver Interrupt

************************************************

007090 48E7F8F0          movem.l D0-D4/A0-A3,-(A7)   setimer, Timer im MFP initialisieren
007094 207CFFFFFA01      move.l  #$FFFFFA01,A0       Register retten
00709A 267C00007124      move.l  #$7124,A3           Adresse MFP
0070A0 247C00007128      move.l  #$7128,A2
0070A6 615A              bsr     $7102
0070A8 267C00007118      move.l  #$7118,A3
0070AE 247C00007128      move.l  #$7128,A2
0070B4 614C              bsr     $7102
0070B6 267C0000711C      move.l  #$711C,A3
0070BC 247C00007128      move.l  #$7128,A2
0070C2 613E              bsr     $7102
0070C4 267C00007120      move.l  #$7120,A3
0070CA 247C00007128      move.l  #$7128,A2
0070D0 6130              bsr     $7102
0070D2 267C0000712C      move.l  #$712C,A3
0070D8 247C00007130      move.l  #$7130,A2
0070DE 6122              bsr     $7102
```

```
0070E0 C749              exg     A3,A1
0070E2 47F900007134      lea     $7134,A3
0070E8 7600              moveq.l #0,D3
0070EA 16330000          move.b  0(A3,D0.w),D3
0070EE 11823000          move.b  D2,0(A0,D3.w)       Daten in MFP schreiben
0070F2 B4303000          cmp.b   0(A0,D3.w),D2       und vergleichen
0070F6 66F6              bne     $70EE               bis Übereinstimmung
0070F8 C749              exg     A3,A1
0070FA 8313              or.b    D1,(A3)
0070FC 4CDF0F1F          movem.l (A7)+,D0-D4/A0-A3   Register zurückholen
007100 4E75              rts

007102 6106              bsr     $710A
007104 1612              move.b  (A2),D3
007106 C713              and.b   D3,(A3)
007108 4E75              rts

00710A 7600              moveq.l #0,D3
00710C D6C0              add.w   D0,A3
00710E 1613              move.b  (A3),D3
007110 D688              add.l   A0,D3
007112 2643              move.l  D3,A3
007114 D4C0              add.w   D0,A2
007116 4E75              rts

************************************************
007118 06060808          dc.b    6,6,8,8
00711C 0A0A0C0C          dc.b    10,10,12,12
007120 0E0E1010          dc.b    14,14,16,16
007124 12121414          dc.b    18,18,20,20

007128 DFFEDFEF          dc.b    $DF,$FE,$DF,$EF
00712C 181A1C1C          dc.b    $18,$1A,$1C,$1C
007130 00008FF8          dc.b    0,0,$8F,$F8
007134 1E202224          dc.b    $1E,$20,$22,$24
```

```
*************************************************   mfpint, MFP-Interruptvektor setzen
007138 302F0004            move.w  4(A7),D0        Interruptnummer
00713C 246F0006            move.l  6(A7),A2        Interruptvektor
007140 02800000000F        and.l   #15,D0          Nummer 0-15, Langwort

*************************************************   initint, MFP Interruptvektor setzen
007146 48E7E0E0            movem.l D0-D2/A0-A2,-(A7) Register retten
00714A 6120                bsr     $716C           disable Interrupts
00714C 2400                move.l  D0,D2           Vektornummer
00714E E542                asl.w   #2,D2           als Index für langwort
007150 068200000100        add.l   #$100,D2        plus Basis der MFP-Vektoren
007156 2242                move.l  D2,A1           Vektoradresse
007158 228A                move.l  A2,(A1)         neuen Vektor setzen
00715A 614A                bsr     $71A6           enable Interrupts
00715C 4CDF0707            movem.l (A7)+,D0-D2/A0-A2 Register zurückholen
007160 4E75                rts

*************************************************   disint, MFP Interrupt disablen
007162 302F0004            move.w  4(A7),D0        Interruptnummer holen
007166 02800000000F        and.l   #15,D0          in Langwortindex wandlen
00716C 48E7C0C0            movem.l D0-D1/A0-A1,-(A7) Register retten
007170 41F9FFFFFA01        lea     $FFFFFA01,A0    Adresse des MFP
007176 43E80012            lea     18(A0),A1       Adresse imra
00717A 614A                bsr     $71C6           zu löschende Bitnummer berechnen
00717C 0391                bclr    D1,(A1)         und Bit löschen
00717E 43E80006            lea     6(A0),A1        Adresse iera
007182 6142                bsr     $71C6           zu löschende Bitnummer berechnen
007184 0391                bclr    D1,(A1)         und Bit löschen
007186 43E8000A            lea     10(A0),A1       Adresse ipra
00718A 613A                bsr     $71C6           zu löschende Bitnummer berechnen
00718C 0391                bclr    D1,(A1)         und Bit löschen
00718E 43E8000E            lea     14(A0),A1       Adresse isra
007192 6132                bsr     $71C6           zu löschende Bitnummer berechnen
007194 0391                bclr    D1,(A1)         und Bit löschen
007196 4CDF0303            movem.l (A7)+,D0-D1/A0-A1 Register zurückholen
00719A 4E75                rts
```

```
************************************************      jenabint, MFP Interrupt enablen
00719C 302F0004             move.w  4(A7),D0        Vektornummer
0071A0 02800000000F         and.l   #15,D0          in Langwortindex
0071A6 48E7C0C0             movem.l D0-D1/A0-A1,-(A7) Register retten
0071AA 41F9FFFFFA01         lea     $FFFFFA01,A0    Adresse MFP
0071B0 43E80006             lea     6(A0),A1        Adresse iera
0071B4 6110                 bsr     $71C6           zu setzende Bitnummer berechnen
0071B6 03D1                 bset    D1,(A1)         und Bit setzen
0071B8 43E80012             lea     18(A0),A1       Adresse imra
0071BC 6108                 bsr     $71C6           zu setzende Bitnummer berechnen
0071BE 03D1                 bset    D1,(A1)         und Bit setzen
0071C0 4CDF0303             movem.l (A7)+,D0-D1/A0-A1 Register zurückholen
0071C4 4E75                 rts

************************************************      bselect, Bit- und Registernummer berechnen
0071C6 1200                 move.b  D0,D1           Interruptnummer merken
0071C8 0C000008             cmp.b   #8,D0           größer 8 ?
0071CC 6D02                 blt     $71D0           nein
0071CE 5141                 subq.w  #8,D1           sonst Offset abziehen
0071D0 0C000008             cmp.b   #8,D0           größer 8 ?
0071D4 6C02                 bge     $71D8           ja
0071D6 5449                 addq.w  #2,A1           Zeiger vom a- auf b-Register
0071D8 4E75                 rts

************************************************      rs232ptr
0071DA 41F9000009D0         lea     $9D0,A0         Zeiger auf RS232 iorec
0071E0 43F9FFFFFA01         lea     $FFFFFA01,A1    Zeiger auf MFP
0071E6 4E75                 rts

************************************************      rs232ibuf
0071E8 34280008             move.w  8(A0),D2        Tail Index
0071EC 36280006             move.w  6(A0),D3        Head Index
0071F0 B443                 cmp.w   D3,D2           Head > Tail ?
0071F2 6204                 bhi     $71F8           nein
0071F4 D4680004             add.w   4(A0),D2        Puffergröße addieren
0071F8 9443                 sub.w   D3,D2           Tail - Head
0071FA 4E75                 rts
```

```
************************************************      rtschk
0071FC 082800010020          btst    #1,32(A0)        RTS/CTS-Mode ?
007202 6704                  beq     $7208            nein
007204 61000862              bsr     $7A68            rtson
007208 4E75                  rts

************************************************      rs232put, RS232 Ausgabe
00720A 40E7                  move.W  SR, -(A7)        Status retten
00720C 007C0700              or.w    #$700,SR         IPL 7, Interrupts sperren
007210 61C8                  bsr     $71DA            rs232ptr, Pufferzeiger holen
007212 082800000020          btst    #0,32(A0)        XON/XOFF-Mode ?
007218 6706                  beq     $7220            nein
00721A 4A28001F              tst.b   31(A0)           XON aktiv ?
00721E 6618                  bne     $7238            ja
007220 08290007002C          btst    #7,44(A1)        sendet MFP noch ?
007226 6710                  beq     $7238            ja
007228 34280014              move.W  20(A0),D2        Head Index
00722C B4680016              cmp.w   22(A0),D2        Tail Index
007230 6606                  bne     $7238            noch Zeichen im Puffer
007232 1341002E              move.b  D1,46(A1)        Byte an MFP übergeben
007236 601A                  bra     $7252

007238 34280016              move.W  22(A0),D2        Tail Index
00723C 610002E0              bsr     $751E            Test auf wrap around
007240 B4680014              cmp.w   20(A0),D2        mit Head Index vergleichen
007244 6716                  beq     $725C            gleich, Puffer voll
007246 2268000E              move.l  14(A0),A1        aktuelle Pufferadresse holen
00724A 13812000              move.b  D1,0(A1,D2.W)    Zeichen in Puffer schreiben
00724E 31420016              move.W  D2,22(A0)        neuen Tail Index merken
007252 61A8                  bsr     $71FC            rtschk, RTS setzen ?
007254 46DF                  move.W  (A7)+,SR         Status zurückholen
007256 023C00FE              and.b   #254,SR          Carry-Flag löschen
00725A 4E75                  rts
```

```
00725C 619E              bsr     $71FC           rtschk, RTS setzen ?
00725E 46DF              move.w  (A7)+,SR        Status zurückholen
007260 003C0001          or.b    #1,SR           Carry-Flag setzen, Zeichen nicht ausgegeben
007264 4E75              rts

************************************************ rs232get, RS232 Eingabe
007266 40E7              move.w  SR, -(A7)       Status retten
007268 007C0700          or.w    #$700,SR        IPL 7, Interrupts sperren
00726C 6100FF6C          bsr     $71DA           rs232ptr, RS232 Zeiger holen
007270 32280006          move.w  6(A0),D1        Head Index
007274 B2680008          cmp.w   8(A0),D1        Tail Index
007278 671A              beq     $7294           kein Zeichen im Puffer ?
00727A 61000296          bsr     $7512           Test auf wrap around
00727E 22680000          move.l  0(A0),A1        Pufferadresse holen
007282 7000              moveq.l #0,D0
007284 10311000          move.b  0(A1,D1.W),D0   Zeichen aus Puffer holen
007288 31410006          move.w  D1,6(A0)        neuen Head Index setzen
00728C 46DF              move.w  (A7)+,SR        Status zurück holen
00728E 023C00FE          and.b   #254,SR         Carry-Flag löschen, ok
007292 6006              bra     $729A

007294 46DF              move.w  (A7)+,SR        Status zurückholen
007296 003C0001          or.b    #1,SR           Carry-Flag setzen, kein Zeichen da
00729A 082800000020      btst    #0,32(A0)       XON/XOFF-Mode ?
0072A0 671C              beq     $72BE           nein
0072A2 4A28001E          tst.b   30(A0)          XON aktiv ?
0072A6 6716              beq     $72BE           nein
0072A8 6100FF3E          bsr     $71E8           benutzte Eingabepufferlänge holen
0072AC B468000A          cmp.w   10(A0),D2       gleich Low water mark ?
0072B0 660C              bne     $72BE           nein
0072B2 123C0011          move.b  #$11,D1         XON
0072B6 6100FF52          bsr     $720A           senden
0072BA 4228001E          clr.b   30(A0)          XON-Flag löschen
0072BE 4E75              rts
```

```
************************************************  rcvint, RS232 Receiver Interruptroutine
0072C0 48E7F0E0          movem.l D0-D3/A0-A2,-(A7) Register retten
0072C4 6100FF14          bsr     $71DA           rs232ptr, RS232 Zeiger holen
0072C8 1169002A001C      move.b  42(A1),28(A0)   Receiver Statusregister lesen
0072CE 08280007001C      btst    #7,28(A0)       Interrupt durch Receiver Buffer full ?
0072D4 67000092          beq     $7368           nein
0072D8 082800010020      btst    #1,32(A0)       RTS/CTS-Mode ?
0072DE 6704              beq     $72E4           nein
0072E0 61000782          bsr     $7A64           rtsoff
0072E4 1029002E          move.b  46(A1),D0       Daten aus Receiver Register lesen
0072E8 082800010020      btst    #1,32(A0)       RTS/CTS-Mode ?
0072EE 6624              bne     $7314           ja
0072F0 082800000020      btst    #0,32(A0)       XON/XOFF-Mode ?
0072F6 671C              beq     $7314           nein
0072F8 0C000011          cmp.b   #17,D0          XON empfangen ?
0072FC 6608              bne     $7306           nein
0072FE 117C0000001F      move.b  #$0,31(A0)      XOFF-Flag löschen
007304 6062              bra     $7368           Zeichen nicht in Puffer

007306 0C000013          cmp.b   #19,D0          XOFF empfangen ?
00730A 6608              bne     $7314           nein
00730C 117C00FF001F      move.b  #$FF,31(A0)     XOFF-Flag setzen
007312 6054              bra     $7368           Zeichen nicht in Puffer

007314 32280008          move.w  8(A0),D1        Tail Index
007318 610001F8          bsr     $7512           Test auf wrap around
00731C B2680006          cmp.w   6(A0),D1        Head gleich Tail ?
007320 6746              beq     $7368           ja
007322 24680000          move.l  0(A0),A2        Pufferadresse holen
007326 15801000          move.b  D0,0(A2,D1.w)   empfangenes Byte in Puffer
00732A 31410008          move.w  D1,8(A0)        neuen Tail Index merken
00732E 6100FEB8          bsr     $71E8           benutzte Eingabepufferlänge holen
007332 B468000C          cmp.w   12(A0),D2       gleich High water mark ?
007336 6624              bne     $735C           nein
007338 082800010020      btst    #1,32(A0)       RTS/CTS-Mode ?
00733E 6628              bne     $7368           ja
007340 082800000020      btst    #0,32(A0)       XON/XOFF-Mode ?
```

```
007346 6714              beq     $735C           nein
007348 4A28001E          tst.b   30(A0)          XOFF schon gesendet ?
00734C 660E              bne     $735C           ja
00734E 117C00FF001E      move.b  #$FF,30(A0)     Flag für XOFF setzen
007354 123C0013          move.b  #$13,D1         XOFF
007358 6100FEB0          bsr     $720A           senden
00735C 082800010020      btst    #1,32(A0)       RTS/CTS-Mode ?
007362 6704              beq     $7368           nein
007364 61000702          bsr     $7A68           rtson
007368 08A90004000E      bclr    #4,14(A1)       Interrupt Service-Bit löschen
00736E 4CDF070F          movem.l (A7)+,D0-D3/A0-A2 Register zurückholen
007372 4E73              rte

************************************************  txrint, Tramsmitter Buffer Empty
007374 48E720E0          movem.l D2/A0-A2,-(A7)  Register retten ?
007378 6100FE60          bsr     $71DA           rs232ptr, RS232 Zeiger holen
00737C 082800010020      btst    #1,32(A0)       RTS/CTS-Mode ?
007382 6630              bne     $73B4           ja, dann diesen Interrupt benutzen
007384 082800000020      btst    #0,32(A0)       XON/XOFF-Mode ?
00738A 6706              beq     $7392           nein
00738C 4A28001F          tst.b   31(A0)          XOFF aktiv ?
007390 6622              bne     $73B4           ja
007392 1169002C001D      move.b  44(A1),29(A0)   Transmitter Statusregister merken
007398 34280014          move.w  20(A0),D2       Head Index
00739C B4680016          cmp.w   22(A0),D2       mit Tail Index vergleichen
0073A0 6712              beq     $73B4           Sendepuffer leer
0073A2 6100017A          bsr     $751E           Test auf wrap arround
0073A6 2468000E          move.l  14(A0),A2       Zeiger auf Sendepuffer
0073AA 13722000002E      move.b  0(A2,D2.w),46(A1) Byte an MFP zu Senden übergeben
0073B0 31420014          move.w  D2,20(A0)       neuen Head Index merken
0073B4 08A90002000E      bclr    #2,14(A1)       Interrupt Service-Bit löschen
0073BA 4CDF0704          movem.l (A7)+,D2/A0-A2  Register zurückholen
0073BE 4E73              rte
```

```
**************************************************  ctsint, CTS Interruptroutine
0073C0 48E720E0          movem.l D2/A0-A2,-(A7)     Register retten
0073C4 6100FE14          bsr     $71DA              rs232ptr, RS232 Zeiger holen
0073C8 082800010020      btst    #1,32(A0)          RTS/CTS-Mode ?
0073CE 672A              beq     $73FA              nein, dann Interrupt ignorieren
0073D0 1169002C001D      move.b  44(A1),29(A0)      Transmitter Status merken
0073D6 08280007001D      btst    #7,29(A0)          Transmitter Buffer empty ?
0073DC 67F8              beq     $73D6              nein, warten (muß nach $73D0 springen!)
0073DE 34280014          move.w  20(A0),D2          Head Index
0073E2 B4680016          cmp.w   22(A0),D2          mit Tail Index vergleichen
0073E6 671E              beq     $7406              Sendepuffer leer
0073E8 61000134          bsr     $751E              Test auf wrap arround
0073EC 2468000E          move.l  14(A0),A2          Zeiger auf Sendepuffer
0073F0 13722000002E      move.b  0(A2,D2.w),46(A1)  Byte an MFP zum Senden übergeben
0073F6 31420014          move.w  D2,20(A0)          neuen Head Index merken
0073FA 08A900020010      bclr    #2,16(A1)          Interrupt Service-Bit löschen
007400 4CDF0704          movem.l (A7)+,D2/A0-A2     Register zurückholen
007404 4E73              rte

007406 60F2              bra     $73FA              Sendepuffer leer

**************************************************  rxerror, Fehler beim Emfangen
007408 48E780C0          movem.l D0/A0-A1,-(A7)     Register retten
00740C 6100FDCC          bsr     $71DA              rs232ptr, RS232 Zeiger holen
007410 1169002A001C      move.b  42(A1),28(A0)      Receiver Status merken
007416 1029002E          move.b  46(A1),D0          Datenregister lesen (löscht Status)
00741A 08A90003000E      bclr    #3,14(A1)          Interrupt Service-Bit löschen
007420 4CDF0301          movem.l (A7)+,D0/A0-A1     Register zurückholen
007424 4E73              rte

**************************************************  txerror, Fehler beim Senden
007426 48E700C0          movem.l A0-A1,-(A7)        Register retten
00742A 6100FDAE          bsr     $71DA              rs232ptr, RS323 Zeiger holen
00742E 1169002C001D      move.b  44(A1),29(A0)      Transmitter Status merken
007434 08A90001000E      bclr    #1,14(A1)          Interrupt Service-Bit löschen
00743A 4CDF0300          movem.l (A7)+/A0-A1        Register zurückholen
00743E 4E73              rte
```

```
*****************************************************   iorec, Zeiger auf Tabelle holen
007440 7200                 moveq.l #0,D1
007442 322F0004             move.w  4(A7),D1            Gerätenummer holen
007446 40E7                 move.w  SR, -(A7)           Status retten
007448 007C0700             or.w    #$700,SR            IPL 7, Interrupts sperren
00744C 45F90000745C         lea     $745C,A2            Basisadresse der Tabelle
007452 E581                 asl.l   #2,D1               mal 4 für Langwort
007454 20321800             move.l  0(A2,D1.l),D0       Zeiger auf iorec holen
007458 46DF                 move.w  (A7)+,SR            Status zurückholen
00745A 4E75                 rts

*****************************************************   iorec Tabelle
00745C 000009D0             dc.l    $9D0                RS232
007460 000009F2             dc.l    $9F2                IKBD
007464 00000A00             dc.l    $A00                MIDI

*****************************************************   rsconf, RS232 konfigurieren
007468 007C0700             or.w    #$700,SR            IPL 7, Interrupts sperren
00746C 6100FD6C             bsr     $71DA               rs232ptr, RS232 Zeiger holen
007470 0F490028             movep.l $28(A1),D7          ucr, rsr, tsr, scr retten
007474 7000                 moveq.l #0,D0
007476 1340002A             move.b  D0,42(A1)           Empfänger disablen
00747A 1340002C             move.b  D0,44(A1)           Sender disablen
00747E 4A6F0006             tst.w   6(A7)               Modus
007482 6B0A                 bmi     $748E               negativ, dann nicht setzen
007484 116F00070020         move.b  7(A7),32(A0)        rsmode neu setzen
00748A 7000                 moveq.l #0,D0
00748C 7400                 moveq.l #0,D2
00748E 4A6F0004             tst.w   4(A7)               Baudrate
007492 6B20                 bmi     $74B4               negatuv, dann nicht ändern
007494 322F0004             move.w  4(A7),D1            Baudrate Nummer holen
007498 45F9000074F2         lea     $74F2,A2            Tabelle für Baudrate Control
00749E 10321000             move.b  0(A2,D1.w),D0       Wert aus Tabelle holen
0074A2 45F900007502         lea     $7502,A2            Tabelle für Baudrate Data
0074A8 14321000             move.b  0(A2,D1.w),D2       Wert aus Tabelle holen
0074AC 2200                 move.l  D0,D1
0074AE 7003                 moveq.l #3,D0               Zeiger auf Timer D
```

```
0074B0 6100FBDE          bsr     $7090           Timer D für neue Baudrate setzen
0074B4 4A6F0008          tst.w   8(A7)           ucr setzen ?
0074B8 6B06              bmi     $74C0           nein
0074BA 136F00090028      move.b  9(A7),40(A1)    ucr setzen
0074C0 4A6F000A          tst.w   10(A7)          rsr setzen ?
0074C4 6B06              bmi     $74CC           nein
0074C6 136F000B002A      move.b  11(A7),42(A1)   rsr setzen
0074CC 4A6F000C          tst.w   12(A7)          tsr setzen ?
0074D0 6B06              bmi     $74D8           nein
0074D2 136F000D002C      move.b  13(A7),44(A1)   tsr setzen
0074D8 4A6F000E          tst.w   14(A7)          scr setzen ?
0074DC 6B06              bmi     $74E4           nein
0074DE 136F000F0026      move.b  15(A7),38(A1)   scr setzen
0074E4 7001              moveq.l #1,D0
0074E6 1340002A          move.b  D0,42(A1)       Empfänger enablen
0074EA 1340002C          move.b  D0,44(A1)       Sender enablen
0074EE 2007              move.l  D7,D0           alter Wert der Controlregister
0074F0 4E75              rts

************************************************          Timerwerte für RS232 Baudrate
0074F2 0101010101010101  dc.b    1,1,1,1,1,1,1,1        Hi-Byte
0074FA 0101010101010202  dc.b    1,1,1,1,1,2,2,2
007502 01020405080A0B10  dc.b    1,2,4,5,8,10,11,16     Lo-Byte
00750B 204060808FAF4060  dc.b    32,64,96,128,143,175,64,96

************************************************          wrapin, Test auf wrap arround
007512 5241              addq.w  #1,D1           Head Index + 1
007514 B2680004          cmp.w   4(A0),D1        gleich Puffergöße ?
007518 6502              bcs     $751C           nein
00751A 7200              moveq.l #0,D1           sonst wieder bei Null beginnen
00751C 4E75              rts

************************************************          wrapout, Test auf wrap arround
00751E 5242              addq.w  #1,D2           Tail Index + 1
007520 B4680012          cmp.w   18(A0),D2       gleich Puffergröße ?
007524 6502              bcs     $7528           nein
007526 7400              moveq.l #0,D2           sonst wieder bei Null beginnen
```

```
007528 4E75                rts

************************************************     midikey, Keyboard + MIDI Interrupt
00752A 48E7FFFE            movem.l D0-D7/A0-A6,-(A7)  Register retten
00752E 4BF900000000        lea     $0,A5              A5 löschen
007534 246D0A2A            move.l  $A2A(A5),A2        mbufrec, $7558
007538 4E92                jsr     (A2)               Interrupt von MIDI-ACIA ?
00753A 246D0A2E            move.l  $A2E(A5),A2        kbufrec, $7568
00753E 4E92                jsr     (A2)               Interrupt von Keyboard-ACIA ?
007540 08390004FFFFFA01    btst    #4,$FFFFFA01       steht noch ein Interrupt aus ?
007548 67EA                beq     $7534              ja
00754A 08B90006FFFFFA11    bclr    #6,$FFFFFA11       Interrupt Service-Bit löschen
007552 4CDF7FFF            movem.l (A7)+,D0-D7/A0-A6  Register zurückholen
007556 4E73                rte

************************************************     MIDI-Interrupt
007558 41ED0A00            lea     $A00(A5),A0        iorec für MIDI
00755C 43F9FFFFFC04        lea     $FFFFFC04,A1       Adresse der MIDI-ACIA
007562 246D0A16            move.l  $A16(A5),A2        $759C, MIDI-Error-Routine
007566 600E                bra     $7576

************************************************     Keyboard-Interrupt
007568 41ED09F2            lea     $9F2(A5),A0        iorec für Keyboard
00756C 43F9FFFFFC00        lea     $FFFFFC00,A1       Adresse der Keyboard-ACIA
007572 246D0A12            move.l  $A12(A5),A2        $759C, Keyboard-Error-Routine
007576 14290000            move.b  0(A1),D2           ACIA-Status holen
00757A 08020007            btst    #7,D2              Interrupt Request ?
00757E 671C                beq     $759C              nein
007580 08020000            btst    #0,D2              Receiver Buffer full ?
007584 670A                beq     $7590              nein
007586 48E720E0            movem.l D2/A0-A2,-(A7)     Register retten
00758A 6112                bsr     $759E              arcvint, Byte holen
00758C 4CDF0704            movem.l (A7)+,D2/A0-A2     Register zurückholen
007590 02020020            and.b   #$20,D2            schon getestetes Bit markieren
007594 6706                beq     $759C              kein Fehler
007596 10290002            move.b  2(A1),D0           Daten lesen, Status löschen
00759A 4ED2                jmp     (A2)               Error-Routine ausführen
```

```
00759C 4E75                rts

************************************************     arcvint, Byte von ACIA holen
00759E 10290002            move.b  2(A1),D0         Daten von ACIA holen
0075A2 B1FC000009F2        cmp.l   #$9F2,A0         Keyboard-ACIA ?
0075A8 66000416            bne     $79C0            nein, MIDI
0075AC 4A2D0A32            tst.b   $A32(A5)         Keyboard State
0075B0 6654                bne     $7606
0075B2 0C0000F6            cmp.b   #$F6,D0          Tastendruck ?
0075B6 650000F4            bcs     $76AC            ja
0075BA 040000F6            sub.b   #$F6,D0          Offset abziehen
0075BE 0280000000FF        and.l   #$FF,D0
0075C4 47F9000075F2        lea     $75F2,A3         Zeiger auf IKBD-Code-Tabelle
0075CA 1B7300000A32        move.b  0(A3,D0.w),$A32(A5)  IKBD merken
0075D0 47F9000075FC        lea     $75FC,A3         Zeiger auf IKBD-Längen Tabelle
0075D6 1B7300000A33        move.b  0(A3,D0.w),$A33(A5)  IKBD Index
0075DC 064000F6            add.w   #$F6,D0          Offset wieder addieren
0075E0 0C0000F8            cmp.b   #$F8,D0          Mouse Position Record ?
0075E4 6D0A                blt     $75F0            nein
0075E6 0C0000FB            cmp.b   #$FB,D0          Mouse Position Record ?
0075EA 6E04                bgt     $75F0            nein
0075EC 1B400A40            move.b  D0,$A40(A5)      Mouse Position merken
0075F0 4E75                rts

************************************************     IKBD-Parameter
0075F2 01020303030304050607                         Status-Codes
0075FC 07050202020206020101                         Längen

************************************************
007606 0C2D00060A32        cmp.b   #6,$A32(A5)      Joystick Record ?
00760C 64000084            bcc     $7692            ja
007610 45F900007656        lea     $7656,A2         Zeiger auf IKBD Parametertabelle
007616 7400                moveq.l #0,D2
007618 142D0A32            move.b  $A32(A5),D2      Kstate
00761C 5302                subq.b  #1,D2            1 - 5 => 0 - 4
00761E E342                asl.w   #1,D2            mal 2
007620 D42D0A32            add.b   $A32(A5),D2      + 1
```

```
007624 5302              subq.b  #1,D2              1 - 5 => 0 - 4
007626 E542              asl.w   #2,D2              mal 4
007628 20722000          move.l  0(A2,D2.w),A0      IKBD Record Zeiger
00762C 22722004          move.l  4(A2,D2.w),A1      IKBD Index Basis
007630 24722008          move.l  8(A2,D2.w),A2      IKBD Interruptroutine
007634 2452              move.l  (A2),A2            Interruptvektor holen
007636 7400              moveq.l #0,D2
007638 142D0A33          move.b  $A33(A5),D2        IKBD Index holen
00763C 93C2              sub.l   D2,A1              minus Basis
00763E 1280              move.b  D0,(A1)
007640 532D0A33          subq.b  #1,$A33(A5)        IKBD Index minus 1
007644 4A2D0A33          tst.b   $A33(A5)           Index testen
007648 660A              bne     $7654
00764A 2F08              move.l  A0,-(A7)           Recordzeiger übergeben
00764C 4E92              jsr     (A2)               Interruptroutine ausführen
00764E 584F              addq.w  #4,A7              Stackpointer korrigieren
007650 422D0A32          clr.b   $A32(A5)           IKBD State löschen
007654 4E75              rts

*********************************************        Parametertabelle für IKBD
007656 00000A34          dc.l    $A34
00765A 00000A3B          dc.l    $A3B
00765E 00000A1A          dc.l    $A1A
007662 00000A3B          dc.l    $A3B
007666 00000A40          dc.l    $A40
00766A 00000A1E          dc.l    $A1E
00766E 00000A40          dc.l    $A40
007672 00000A43          dc.l    $A43
007676 00000A1E          dc.l    $A1E
00767A 00000A43          dc.l    $A43
00767E 00000A49          dc.l    $A49
007682 00000A22          dc.l    $A22
007686 00000A49          dc.l    $A49
00768A 00000A4B          dc.l    $A4B
00768E 00000A26          dc.l    $A26
```

```
************************************************
007692 223C00000A4A          move.l  #$A4A,D1
007698 D22D0A32              add.b   $A32(A5),D1
00769C 5D01                  subq.b  #6,D1
00769E 2441                  move.l  D1,A2
0076A0 1480                  move.b  D0,(A2)
0076A2 246D0A26              move.l  $A26(A5),A2     Joystick Interruptroutine
0076A6 41ED0A49              lea     $A49(A5),A0     Adresse Joystick Daten
0076AA 609E                  bra     $764A

************************************************    empfangene Tastencodes verarbeiten
0076AC 122D0A5D              move.b  $A5D(A5),D1     Shiftstatus laden
0076B0 0C00002A              cmp.b   #$2A,D0         linke Shifttaste gedrückt ?
0076B4 6606                  bne     $76BC           nein
0076B6 08C10001              bset    #1,D1           Bit für linke Shiftaste setzen
0076BA 6074                  bra     $7730
0076BC 0C0000AA              cmp.b   #$AA,D0         linke Shiftaste losgelassen ?
0076C0 6606                  bne     $76C8           nein
0076C2 08810001              bclr    #1,D1           Bit für linke Shiftaste löschen
0076C6 6068                  bra     $7730
0076C8 0C000036              cmp.b   #$36,D0         rechte Shifttaste gedrückt ?
0076CC 6606                  bne     $76D4           nein
0076CE 08C10000              bset    #0,D1           Bit für rechte Shifttaste setzen
0076D2 605C                  bra     $7730
0076D4 0C0000B6              cmp.b   #$B6,D0         rechte Shiftaste losgelassen ?
0076D8 6606                  bne     $76E0           nein
0076DA 08810000              bclr    #0,D1           Bit für rechte Shifttaste löschen
0076DE 6050                  bra     $7730
0076E0 0C00001D              cmp.b   #$1D,D0         CTRL-Taste gedrückt ?
0076E4 6606                  bne     $76EC           nein
0076E6 08C10002              bset    #2,D1           Bit für CTRL-Taste setzen
0076EA 6044                  bra     $7730
0076EC 0C00009D              cmp.b   #$9D,D0         CTRL-Taste losgelassen ?
0076F0 6606                  bne     $76F8           nein
0076F2 08810002              bclr    #2,D1           Bit für CTRL-Taste löschen
0076F6 6038                  bra     $7730
0076F8 0C000038              cmp.b   #$38,D0         ALT-Taste gedrückt ?
```

```
0076FC 6606                bne     $7704           nein
0076FE 08C10003            bset    #3,D1           Bit für ALT-Taste setzen
007702 602C                bra     $7730
007704 0C0000B8            cmp.b   #$B8,D0         ALT-Taste losgelassen ?
007708 6606                bne     $7710           nein
00770A 08810003            bclr    #3,D1           Bit für ALT-Taste löschen
00770E 6020                bra     $7730
007710 0C00003A            cmp.b   #$3A,D0         CAPS LOCK-Taste gedrückt ?
007714 6620                bne     $7736           nein
007716 082D00000484        btst    #0,$484(A5)     Consol-Konfiguration holen
00771C 670E                beq     $772C           kein Tasten-Klick
00771E 2B7C00007D780A86    move.l  #$7D78,$A86(A5) Adresse der Tasten-Klick Soundtabelle
007726 1B7C00000A8A        move.b  #$0,$A8A(A5)    Sound starten
00772C 08410004            bchg    #4,D1           CAPS LOCK Status invertieren
007730 1B410A5D            move.b  D1,$A5D(A5)     neuen Shiftstatus merken
007734 4E75                rts

007736 08000007            btst    #7,D0           wurde Taste losgelassen ?
00773A 662A                bne     $7766           ja
00773C 4A2D0A7B            tst.b   $A7B(A5)        Repeat ?
007740 6616                bne     $7758           ja
007742 1B400A7B            move.b  D0,$A7B(A5)     Tastencode für Repeat merken
007746 1B7900000A7E0A7C    move.b  $A7E,$A7C(A5)   Verzögerung 1
00774E 1B7900000A7F0A7D    move.b  $A7F,$A7D(A5)   Verzögerung 2
007756 603A                bra     $7792

007758 1B7C00000A7C        move.b  #$0,$A7C(A5)    Zähler für Verzögerung löschen
00775E 1B7C00000A7D        move.b  #$0,$A7D(A5)    Zähler für Verzögerung löschen
007764 602C                bra     $7792
007766 4A2D0A7B            tst.b   $A7B(A5)
00776A 670E                beq     $777A
00776C 7200                moveq.l #0,D1
00776E 1B410A7B            move.b  D1,$A7B(A5)
007772 1B410A7C            move.b  D1,$A7C(A5)
007776 1B410A7D            move.b  D1,$A7D(A5)
00777A 0C0000C7            cmp.b   #$C7,D0         HOME-Taste losgelassen ?
00777E 6708                beq     $7788           ja
```

```
007780 0C0000D2          cmp.b   #$D2,D0          INSERT-Taste losgelassen ?
007784 66000238          bne     $79BE            nein
007788 082D00030A5D      btst    #3,$A5D(A5)      ALT-Taste noch gedrückt ?
00778E 6700022E          beq     $79BE            nein
007792 082D00000484      btst    #0,$484(A5)      Consol-Status
007798 670E              beq     $77A8            kein Tasten-Klick
00779A 2B7C00007D780A86  move.l  #$7D78,$A86(A5)  Adresse des Soundtabelle für Tasten-Klick
0077A2 1B7C00000A8A      move.b  #$0,$A8A(A5)     Sound starten
0077A8 2F08              move.l  A0,-(A7)         iorec für Keyboard merken
0077AA 7200              moveq.l #0,D1
0077AC 1200              move.b  D0,D1            Scan-Code nach D1
0077AE 206D0A5E          move.l  $A5E(A5),A0      Adresse der Standard Tastaturtabelle
0077B2 0240007F          and.w   #$7F,D0
0077B6 082D00040A5D      btst    #4,$A5D(A5)      CAPS LOCK aktiv ?
0077BC 6704              beq     $77C2            nein
0077BE 206D0A66          move.l  $A66(A5),A0      Adresse der CAPS LOCK Tastaturtabelle
0077C2 082D00000A5D      btst    #0,$A5D(A5)      rechte Shifttaste gedrückt ?
0077C8 6608              bne     $77D2            ja
0077CA 082D00010A5D      btst    #1,$A5D(A5)      linke Shifttaste gedrückt ?
0077D0 671A              beq     $77EC            nein
0077D2 0C00003B          cmp.b   #59,D0           Funktionstaste ?
0077D6 6510              bcs     $77E8            nein
0077D8 0C000044          cmp.b   #68,D0           Funktionstaste ?
0077DC 620A              bhi     $77E8            nein
0077DE 06410019          add.w   #25,D1           Offset nach GSX-Standard addieren
0077E2 7000              moveq.l #0,D0
0077E4 600001B2          bra     $7998
0077E8 206D0A62          move.l  $A62(A5),A0      Adresse der Shift-Tastaturtabelle
0077EC 10300000          move.b  0(A0,D0.w),D0    ASCII-Code aus Tabelle holen
0077F0 082D00020A5D      btst    #2,$A5D(A5)      CTRL-Taste gedrückt ?
0077F6 6760              beq     $7858            nein
0077F8 0C00000D          cmp.b   #13,D0           Carriage Return
0077FC 6604              bne     $7802            nein
0077FE 700A              moveq.l #10,D0           in Line Feed umwandlen
007800 672A              beq     $782C
007802 0C010047          cmp.b   #$47,D1          CTRL HOME ?
007806 6608              bne     $7810            nein
```

```
007808 06410030          add.w   #$30,D1          $30 addieren nach GSX-Standard
00780C 6000018A          bra     $7998
007810 0C01004B          cmp.b   #$4B,D1          CTRL Cursor links ?
007814 6608              bne     $781E
007816 7273              moveq.l #$73,D1          in GSX-Standard wandeln
007818 7000              moveq.l #0,D0
00781A 6000017C          bra     $7998
00781E 0C01004D          cmp.b   #$4D,D1          CTRL Cursor rechts ?
007822 6608              bne     $782C
007824 7274              moveq.l #$74,D1          in GSX-Standard wandeln
007826 7000              moveq.l #0,D0
007828 6000016E          bra     $7998
00782C 0C000032          cmp.b   #$32,D0
007830 6606              bne     $7838
007832 7000              moveq.l #0,D0
007834 60000162          bra     $7998
007838 0C000036          cmp.b   #$36,D0
00783C 6606              bne     $7844
00783E 701E              moveq.l #$1E,D0
007840 60000156          bra     $7998
007844 0C00002D          cmp.b   #$2D,D0
007848 6606              bne     $7850
00784A 701F              moveq.l #$1F,D0
00784C 6000014A          bra     $7998
007850 0240001F          and.w   #$1F,D0
007854 60000142          bra     $7998
007858 082D00030A5D      btst    #3,$A5D(A5)      ALT-Taste gedrückt ?
00785E 67000138          beq     $7998            nein
007862 0C01001A          cmp.b   #$1A,D1          'Ü' ?
007866 6618              bne     $7880            nein
007868 103C0040          move.b  #$40,D0          '@'
00786C 142D0A5D          move.b  $A5D(A5),D2      Shift-Status
007870 02020003          and.b   #3,D2            linke und/oder rechte Shiftaste gedrückt?
007874 67000122          beq     $7998            nein
007878 103C005C          move.b  #$5C,D0          '\'
00787C 6000011A          bra     $7998
007880 0C010027          cmp.b   #$27,D1          'Ö' ?
```

```
007884 6618              bne     $789E           nein
007886 103C005B          move.b  #$5B,D0         '['
00788A 142D0A5D          move.b  $A5D(A5),D2     Shift-Status
00788E 02020003          and.b   #3,D2           linke und/oder rechte Shiftaste gedrückt?
007892 67000104          beq     $7998           nein
007896 103C007B          move.b  #$7B,D0         '{'
00789A 600000FC          bra     $7998
00789E 0C010028          cmp.b   #$28,D1         'Ä' ?
0078A2 6618              bne     $78BC
0078A4 103C005D          move.b  #$5D,D0         ']'
0078A8 142D0A5D          move.b  $A5D(A5),D2     Shift-Status
0078AC 02020003          and.b   #3,D2           linke und/oder rechte Shiftaste gedrückt?
0078B0 670000E6          beq     $7998           nein
0078B4 103C007D          move.b  #$7D,D0         '}'
0078B8 600000DE          bra     $7998
0078BC 0C010062          cmp.b   #$62,D1         ATL HELP ?
0078C0 660A              bne     $78CC           nein
0078C2 526D04EE          addq.w  #1,$4EE(A5)     dumpflg für Hardcopy setzen
0078C6 205F              move.l  (A7)+,A0        Keyboard iorec zurückholen
0078C8 600000F4          bra     $79BE           fertig

0078CC 45F900007A2C      lea     $7A2C,A2        Zeiger auf Mosue Scancode Tabelle
0078D2 7403              moveq.l #3,D2           4 Werte testen
0078D4 B2322000          cmp.b   0(A2,D2.w),D1   Wert gefunden ?
0078D8 6700010E          beq     $79E8           ja
0078DC 51CAFFF6          dbra    D2,$78D4        nächsten Wert
0078E0 0C010048          cmp.b   #$48,D1         Cursor hoch ?
0078E4 661C              bne     $7902           nein
0078E6 123C0000          move.b  #0,D1           X-Offset für Cursor hoch
0078EA 143CFFF8          move.b  #-8,D2          Y-Offset für Cursor hoch
0078EE 102D0A5D          move.b  $A5D(A5),D0     Shift-Status holen
0078F2 02000003          and.b   #3,D0           linke und/oder rechte Shiftaste gedrückt?
0078F6 6700010E          beq     $7A06           nein
0078FA 143CFFFF          move.b  #-1,D2          nur ein Pixel nach oben mit Shift
0078FE 60000106          bra     $7A06
007902 0C01004B          cmp.b   #$4B,D1         Cursor links ?
007906 661C              bne     $7924           nein
```

```
007908 143C0000      move.b  #0,D2          Y-Offset für Cursor links
00790C 123CFFF8      move.b  #-8,D1         X-Offset für Cursor links
007910 102D0A5D      move.b  $A5D(A5),D0    Shift-Status holen
007914 02000003      and.b   #3,D0          linke und/oder rechte Shiftaste gedrückt?
007918 670000EC      beq     $7A06          nein
00791C 123CFFFF      move.b  #$-1,D1        nur ein Pixel nach links mit Shift
007920 600000E4      bra     $7A06
007924 0C01004D      cmp.b   #$4D,D1        Cursor rechts ?
007928 661C          bne     $7946          nein
00792A 123C0008      move.b  #8,D1          X-Offset für Cursor rechts
00792E 143C0000      move.b  #0,D2          Y-Offset für Cursor rechts
007932 102D0A5D      move.b  $A5D(A5),D0    Shift-Status
007936 02000003      and.b   #3,D0          linke und/oder rechte Shiftaste gedrückt?
00793A 670000CA      beq     $7A06          nein
00793E 123C0001      move.b  #1,D1          nur ein Pixel nach rechts mit Shift
007942 600000C2      bra     $7A06
007946 0C010050      cmp.b   #$50,D1        Cursor runter ?
00794A 661C          bne     $7968          nein
00794C 123C0000      move.b  #0,D1          X-Offset für Cursor down
007950 143C0008      move.b  #8,D2          Y-Offset für Cursor down
007954 102D0A5D      move.b  $A5D(A5),D0    Shift-Status
007958 02000003      and.b   #3,D0          linke und/oder rechte Shiftaste gedrückt?
00795C 670000A8      beq     $7A06          nein
007960 143C0001      move.b  #1,D2          nur ein Pixel nach unten mit Shift
007964 600000A0      bra     $7A06
007968 0C010002      cmp.b   #2,D1          '1'
00796C 650C          bcs     $797A          nicht größer oder gleich
00796E 0C01000D      cmp.b   #$D,D1         '='
007972 6206          bhi     $797A          nicht kleiner oder gleich
007974 06010076      add.b   #$76,D1
007978 600C          bra     $7986
00797A 0C000041      cmp.b   #$41,D0        'A'
00797E 650A          bcs     $798A
007980 0C00005A      cmp.b   #$5A,D0        'Z'
007984 6204          bhi     $798A
007986 7000          moveq.l #0,D0
007988 600E          bra     $7998
```

```
00798A 0C000061          cmp.b   #$61,D0          'a'
00798E 6508              bcs     $7998
007990 0C00007A          cmp.b   #$7A,D0          'z'
007994 6202              bhi     $7998
007996 60EE              bra     $7986
007998 E141              asl.w   #8,D1            Scancode nach Bit 8-15
00799A D041              add.w   D1,D0            ASCII-Code nach Bit 0-7
00799C 205F              move.l  (A7)+,A0         iorec-Zeiger holen
00799E 32280008          move.w  8(A0),D1         Tail Index
0079A2 5441              addq.w  #2,D1            plus 2
0079A4 B2680004          cmp.w   4(A0),D1         Pufferende erreicht ?
0079A8 6502              bcs     $79AC            nein
0079AA 7200              moveq.l #0,D1            Pufferzeiger wieder auf Pufferstart
0079AC B2680006          cmp.w   6(A0),D1         Head gleich Tail ?
0079B0 670C              beq     $79BE            ja, Puffer voll
0079B2 24680000          move.l  0(A0),A2         Pufferadresse
0079B6 35801000          move.w  D0,0(A2,D1.w)    Daten in Puffer schreiben
0079BA 31410008          move.w  D1,8(A0)         neuer Tail Index
0079BE 4E75              rts

************************************************ midibyte
0079C0 246D0A0E          move.l  $A0E(A5),A2      Zeiger auf MIDI-Interrupthandler
0079C4 4ED2              jmp     (A2)             Routine ausführen

************************************************ sysmidi
0079C6 32280008          move.w  8(A0),D1         Tail Index
0079CA 5241              addq.w  #1,D1            erhöhen
0079CC B2680004          cmp.w   4(A0),D1         Pufferende erreicht ?
0079D0 6502              bcs     $79D4            nein
0079D2 7200              moveq.l #0,D1            Pufferzeiger wieder auf Start
0079D4 B2680006          cmp.w   6(A0),D1         Head gleich Tail ?
0079D8 670C              beq     $79E6            ja, Puffer voll
0079DA 24680000          move.l  0(A0),A2         Pufferadresse
0079DE 15801000          move.b  D0,0(A2,D1.w)    empfangenes Byte in Puffer schreiben
0079E2 31410008          move.w  D1,8(A0)         neuer Tail Index
0079E6 4E75              rts
```

```
************************************************** keymaus1
0079E8 7605              moveq.l #5,D3             rechte Taste annehmen
0079EA 08010004          btst    #4,D1
0079EE 6702              beq     $79F2             ist rechte Taste ($47/$C7)
0079F0 7606              moveq.l #6,D3             linke Taste
0079F2 08010007          btst    #7,D1             gedrückt oder losgelassen ?
0079F6 6706              beq     $79FE             gedrückt
0079F8 07AD0A5D          bclr    D3,$A5D(A5)       Bit für Taste löschen
0079FC 6004              bra     $7A02
0079FE 07ED0A5D          bset    D3,$A5D(A5)       Bit für Taste setzen
007A02 7200              moveq.l #0,D1
007A04 7400              moveq.l #0,D2
************************************************** keymaus
007A06 41ED0A5A          lea     $A5A(A5),A0       Zeiger auf Maus-Emulator-Puffer
007A0A 246D0A1E          move.l  $A1E(A5),A2       Maus-Interrupt-Vektor
007A0E 4280              clr.l   D0
007A10 102D0A5D          move.b  $A5D(A5),D0       Status der 'Maus'-Tasten holen
007A14 EA08              lsr.b   #5,D0             Bit für rechte Taste nach Bit 0
007A16 060000F8          add.b   #$F8,D0           plus relativ Maus Header
007A1A 11400000          move.b  D0,0(A0)          Byte in Puffer
007A1E 11410001          move.b  D1,1(A0)          X-Wert speichern
007A22 11420002          move.b  D2,2(A0)          Y-Wert speichern
007A26 4E92              jsr     (A2)              Maus-Interrupt-Routine aufrufen
007A28 205F              move.l  (A7)+,A0          iorec für Keyboard zurückholen
007A2A 4E75              rts

************************************************** amuskey1
007A2C 47C7              dc.b    $47,$C7,$52,$D2   Scan-Code für Maus-Ersatz

************************************************** giaccess, Soundchip lesen/schreiben
007A30 302F0004          move.w  4(A7),D0          Daten
007A34 322F0006          move.w  6(A7),D1          Registernummer plus Read/Write
007A38 40E7              move.w  SR, -(A7)         Status merken
007A3A 007C0700          or.w    #$700,SR          IPL 7, Interrupts sperren
007A3E 48E76080          movem.l D1-D2/A0,-(A7)    Register retten
007A42 41F9FFFF8800      lea     $FFFF8800,A0      Adresse des Soundchips
007A48 1401              move.b  D1,D2             Registernummer holen
```

```
007A4A 0201000F          and.b   #15,D1          Register 0-15
007A4E 1081              move.b  D1,(A0)         Register selektieren
007A50 E302              asl.b   #1,D2           Read/Write-Bit testen
007A52 6404              bcc     $7A58           lesen
007A54 11400002          move.b  D0,2(A0)        Datenbyte in Soundchip-Register schreiben
007A58 7000              moveq.l #0,D0
007A5A 1010              move.b  (A0),D0         Byte vom Soundchip-Register lesen
007A5C 4CDF0106          movem.l (A7)+,D1-D2/A0  Register zurückholen
007A60 46DF              move.w  (A7)+,SR        Status zurückholen
007A62 4E75              rts

************************************************     rtsoff, RTS abschalten
007A64 7408              moveq.l #8,D2
007A66 6012              bra     $7A7A           Bit in Port A setzen

************************************************     rtson, RTS einschalten
007A68 74F7              moveq.l #$F7,D2
007A6A 6034              bra     $7AA0           Bit in Port A löschen

************************************************     dtroff, DTR abschalten
007A6C 7410              moveq.l #16,D2
007A6E 600A              bra     $7A7A           Bit in Port A setzen

************************************************     dtroff, DTR einschalten
007A70 74EF              moveq.l #-17,D2
007A72 602C              bra     $7AA0           Bit in Port A löschen

************************************************     ongibit, Bit in Soundchip Port A setzen
007A74 7400              moveq.l #0,D2
007A76 342F0004          move.w  4(A7),D2        Bitmuster holen
007A7A 48E7E000          movem.l D0-D2,-(A7)     Register retten
007A7E 40E7              move.w  SR, -(A7)       Status retten
007A80 007C0700          or.w    #$700,SR        IPL 7, Interrupts sperren
007A84 720E              moveq.l #14,D1          Port A
007A86 2F02              move.l  D2,-(A7)        Bitnummer retten
007A88 61AE              bsr     $7A38           Port A selektieren
007A8A 241F              move.l  (A7)+,D2        Bitnummer zurück
```

```
007A8C 8002              or.b    D2,D0           Bit(s) setzen
007A8E 728E              moveq.l #$8E,D1         Port A schreiben
007A90 61A6              bsr     $7A38           neuen Wert schreiben
007A92 46DF              move.w  (A7)+,SR        Status zurückholen
007A94 4CDF0007          movem.l (A7)+,D0-D2     Register zurückholen
007A98 4E75              rts

************************************************ offgibit, Bit im Soundchip Port A löschen
007A9A 7400              moveq.l #0,D2
007A9C 342F0004          move.w  4(A7),D2        Bitmuster holen
007AA0 48E7E000          movem.l D0-D2,-(A7)     Register retten
007AA4 40E7              move.w  SR, -(A7)       Status retten
007AA6 007C0700          or.w    #$700,SR        IPL 7, Interrupts sperren
007AAA 720E              moveq.l #14,D1          Port A
007AAC 2F02              move.l  D2,-(A7)        Bitmuster merken
007AAE 6188              bsr     $7A38           Port A selektieren
007AB0 241F              move.l  (A7)+,D2        Bitmuster zurückholen
007AB2 C002              and.b   D2,D0           Bit(s) löschen
007AB4 728E              moveq.l #$8E,D1         Port A schreiben
007AB6 6180              bsr     $7A38           neuen Wert schreiben
007AB8 46DF              move.w  (A7)+,SR        Status zurückholen
007ABA 4CDF0007          movem.l (A7)+,D0-D2     Register zurückholen
007ABE 4E75              rts

************************************************ initmouse
007AC0 4A6F0004          tst.w   4(A7)           Maus abschalten ?
007AC4 6726              beq     $7AEC           ja disable mouse
007AC6 2B6F000A0A1E      move.l  10(A7),$A1E(A5) Maus-Interrupt-Vektor
007ACC 266F0006          move.l  6(A7),A3        Adresse des Parameterblocks
007AD0 0C6F00010004      cmp.w   #1,4(A7)        relative mouse ?
007AD6 6724              beq     $7AFC           ja
007AD8 0C6F00020004      cmp.w   #2,4(A7)        absolute mouse ?
007ADE 6736              beq     $7B16           ja
007AE0 0C6F00040004      cmp.w   #4,4(A7)        Keycode mouse ?
007AE6 6770              beq     $7B58           ja
007AE8 7000              moveq.l #0,D0           Fehler, ungültig
007AEA 4E75              rts
```

```
************************************************      disable mouse
007AEC 7212                 moveq.l #$12,D1         disable mouse Befehl
007AEE 6100F1E0             bsr     $6CD0           an IKBD senden
007AF2 2B7C00007BC00A1E     move.l  #$7BC0,$A1E(A5) Maus-Interrupt-Vektor auf rts
007AFA 6070                 bra     $7B6C

************************************************      relative mouse
007AFC 45ED0A6A             lea     $A6A(A5),A2     Transfer Puffer Zeiger
007B00 14FC0008             move.b  #$8,(A2)+       relative mouse
007B04 14FC000B             move.b  #$B,(A2)+       relative mouse treshold x, y
007B08 6166                 bsr     $7B70           Mausparameter setzen
007B0A 7606                 moveq.l #6,D3           Länge des Strings - 1
007B0C 45ED0A6A             lea     $A6A(A5),A2     Transfer Puffer Zeiger
007B10 6100F1DE             bsr     $6CF0           String an IKBD senden
007B14 6056                 bra     $7B6C

************************************************      absolute mouse
007B16 45ED0A6A             lea     $A6A(A5),A2     Transfer Puffer Zeiger
007B1A 14FC0009             move.b  #9,(A2)+        absolute mouse
007B1E 14EB0004             move.b  4(A3),(A2)+     xmax msb
007B22 14EB0005             move.b  5(A3),(A2)+     xmax lsb
007B26 14EB0006             move.b  6(A3),(A2)+     ymax msb
007B2A 14EB0007             move.b  7(A3),(A2)+     ymax lsb
007B2E 14FC000C             move.b  #$C,(A2)+       absolute mouse scale
007B32 613C                 bsr     $7B70           Mausparameter setzen

007B34 14FC000E             move.b  #$E,(A2)+       initial absolute mouse position
007B38 14FC0000             move.b  #0,(A2)+        Füllbyte
007B3C 14EB0008             move.b  8(A3),(A2)+     Startposition x msb
007B40 14EB0009             move.b  9(A3),(A2)+     Startposition x lsb
007B44 14EB000A             move.b  10(A3),(A2)+    Startposition y msb
007B48 14EB000B             move.b  11(A3),(A2)+    Startposition y lsb
007B4C 7610                 moveq.l #16,D3          Stringlänge -1
007B4E 45ED0A6A             lea     $A6A(A5),A2     Transfer Puffer Zeiger
007B52 6100F19C             bsr     $6CF0           String an IKBD senden
007B56 6014                 bra     $7B6C
```

```
007B58 45ED0A6A          lea     $A6A(A5),A2     Transfer Puffer Zeiger
007B5C 14FC000A          move.b  #$A,(A2)+       mouse keycode mode
007B60 610E              bsr     $7B70           Mausparameter setzen
007B62 7605              moveq.l #5,D3           Länge des Strings -1
007B64 45ED0A6A          lea     $A6A(A5),A2     Transfer Puffer Zeiger
007B68 6100F186          bsr     $6CF0           String an IBKD senden
007B6C 70FF              moveq.l #-1,D0          Flag für ok
007B6E 4E75              rts

*************************************************  setmouse, Mausparameter setzen
007B70 14EB0002          move.b  2(A3),(A2)+     x treshold, scale, delta
007B74 14EB0003          move.b  3(A3),(A2)+     y treshold, scale, delta
007B78 7210              moveq.l #16,D1          top/bottom ?
007B7A 922B0000          sub.b   0(A3),D1
007B7E 14C1              move.b  D1,(A2)+
007B80 14FC0007          move.b  #7,(A2)+
007B84 14EB0001          move.b  1(A3),(A2)+
007B88 4E75              rts

*************************************************  xbtimer, Timer initialisieren
007B8A 7000              moveq.l #0,D0
007B8C 7200              moveq.l #0,D1
007B8E 7400              moveq.l #0,D2
007B90 302F0004          move.w  4(A7),D0        Timer Nummer (0-3 => A - D)
007B94 322F0006          move.w  6(A7),D1        Wert für Controll-Register
007B98 342F0008          move.w  8(A7),D2        Wert für Datenregister
007B9C 6100F4F2          bsr     $7090           Timerwerte setzen
007BA0 4AAF000A          tst.l   10(A7)          zugehöriger Interruptvektor
007BA4 6B1A              bmi     $7BC0           nicht benutzt ?
007BA6 246F000A          move.l  10(A7),A2       Interruptvektor
007BAA 7200              moveq.l #0,D1
007BAC 43F900007BC2      lea     $7BC2,A1        Tabelle zur Bestimmung der Interruptnummer
007BB2 0280000000FF      and.l   #$FF,D0
007BB8 10310000          move.b  0(A1,D0.w),D0   Interruptnummer holen
007BBC 6100F588          bsr     $7146           initint, MFP Interrupt setzen
007BC0 4E75              rts
```

```
************************************************
007BC2 0D080504            dc.b    13,8,5,4

************************************************
007BC6 4AAF0004            tst.l   4(A7)
007BCA 6B06                bmi     $7BD2
007BCC 2B6F00040A5E        move.l  4(A7),$A5E(A5)
007BD2 4AAF0008            tst.l   8(A7)
007BD6 6B06                bmi     $7BDE
007BD8 2B6F00080A62        move.l  8(A7),$A62(A5)
007BDE 4AAF000C            tst.l   12(A7)
007BE2 6B06                bmi     $7BEA
007BE4 2B6F000C0A66        move.l  12(A7),$A66(A5)
007BEA 203C00000A5E        move.l  #$A5E,D0
007BF0 4E75                rts

************************************************
007BF2 2B7C00006D620A5E    move.l  #$6D62,$A5E(A5)
007BFA 2B7C00006DE20A62    move.l  #$6DE2,$A62(A5)
007C02 2B7C00006E620A66    move.l  #$6E62,$A66(A5)
007C0A 4E75                rts

************************************************
007C0C 202D0A86            move.l  $A86(A5),D0
007C10 222F0004            move.l  4(A7),D1
007C14 6B08                bmi     $7C1E
007C16 2B410A86            move.l  D1,$A86(A5)
007C1A 422D0A8A            clr.b   $A8A(A5)
007C1E 4E75                rts

************************************************
007C20 302D0A8C            move.w  $A8C(A5),D0
007C24 4A6F0004            tst.w   4(A7)
007C28 6B06                bmi     $7C30
007C2A 3B6F00040A8C        move.w  4(A7),$A8C(A5)
007C30 4E75                rts
```

Interruptnummern der MFP-Timer

keytrans, Keyboard-Tabellen setzen
Standardtabelle ändern ?
nein
Adresse der Standardtabelle
Shifttabelle ändern ?
nein
Adresse der Shiftabelle
Caps Lock Tabelle ändern ?
nein
Adresse der Caps Lock Tabelle
Zeiger auf Adresse der Tabellen

bioskeys, Standard-Tastaturtabelle setzen
Standardtabelle
Shifttabelle
Caps Lock Tabelle

dosound, Sound starten
Soundstatus holen
Adresse der Soundtabelle
negativ, nicht setzen
neue Soundtabelle
Sound-Timer starten

setprt, Printerkonfiguration setzen/holen
Printerkonfiguration holen
neuer Wert
negativ, nicht setzen
Printerkonfiguration setzen

```
************************************************      kbrate, Tastatur-Repeat setzen/holen
007C32 302D0A7E            move.w  $A7E(A5),D0      Verzögerung vor Tasten-Repeat
007C36 4A6F0004            tst.w   4(A7)            neuen Wert testen
007C3A 6B16                bmi     $7C52            negativ, nicht setzen
007C3C 322F0004            move.w  4(A7),D1
007C40 1B410A7E            move.b  D1,$A7E(A5)      neuen Wert setzen
007C44 4A6F0006            tst.w   6(A7)            Wiederholrate
007C48 6B08                bmi     $7C52            negativ, nicht setzen
007C4A 322F0006            move.w  6(A7),D1
007C4E 1B410A7F            move.b  D1,$A7F(A5)      neuen Wert setzen
007C52 4E75                rts

************************************************      ikbdvecs, Zeiger auf IKBD + MIDI-Vektoren
007C54 203C00000A0E        move.l  #$A0E,D0         Adresse der Vektortabelle
007C5A 4E75                rts

************************************************      timercint, Timer C Interrupt
007C5C 52B9000004BA        addq.l  #1,$4BA          200 Hz Zähler erhöhen
007C62 E7F900000A84        rol.w   $A84             Divisorbit rotieren
007C68 6A4E                bpl     $7CB8            nicht vierter Interrupt, dann fertig
007C6A 48E7FFFE            movem.l D0-D7/A0-A6,-(A7) Register retten
007C6E 4BF900000000        lea     $0,A5            A5 löschen
007C74 614C                bsr     $7CC2            Sound verarbeiten
007C76 082D00010484        btst    #1,$484(A5)      Keyrepeat enabled ?
007C7C 672A                beq     $7CA8            nein
007C7E 4A2D0A7B            tst.b   $A7B(A5)         Taste gedrückt ?
007C82 6724                beq     $7CA8            nein
007C84 4A2D0A7C            tst.b   $A7C(A5)         Zähler für Startverzögerung
007C88 6706                beq     $7C90            nicht aktiv
007C8A 532D0A7C            subq.b  #1,$A7C(A5)      Zähler dekrementieren
007C8E 6618                bne     $7CA8            noch nicht abgelaufen ?
007C90 532D0A7D            subq.b  #1,$A7D(A5)      Zähler für Wiederholrate dekrementieren
007C94 6612                bne     $7CA8            noch nicht abgelaufen
007C96 1B6D0A7F0A7D        move.b  $A7F(A5),$A7D(A5) Zähler neu laden
007C9C 102D0A7B            move.b  $A7B(A5),D0      zu wiederholende Taste
007CA0 41ED09F2            lea     $9F2(A5),A0      Zeiger auf iorec Keyboard
007CA4 6100FAEC            bsr     $7792            Tastencode in Keyboardpuffer
```

```
007CA8 3F2D0442            move.w  $442(A5),-(A7)      20 Millisekunden pro Aufruf
007CAC 206D0400            move.l  $400(A5),A0         Event-Timer Vektor holen
007CB0 4E90                jsr     (A0)                Routine ausführen
007CB2 544F                addq.w  #2,A7               Stackpointer korrigieren
007CB4 4CDF7FFF            movem.l (A7)+,D0-D7/A0-A6   Register zurückholen
007CB8 08B90005FFFFFA11    bclr    #5,$FFFFFA11        Interrupt Service-Bit löschen
007CC0 4E73                rte

************************************************      sndirq, Sound-Interrupt-Routine
007CC2 48E7C080            movem.l D0-D1/A0,-(A7)      Register retten
007CC6 202D0A86            move.l  $A86(A5),D0         Zeiger auf Soundtabelle
007CCA 67000088            beq     $7D54               kein Sound aktiv ?
007CCE 2040                move.l  D0,A0               Zeiger anch A0
007CD0 102D0A8A            move.b  $A8A(A5),D0         Timerwert laden
007CD4 6708                beq     $7CDE               neuer Sound gestartet ?
007CD6 5300                subq.b  #1,D0               sonst Timer dekrementieren
007CD8 1B400A8A            move.b  D0,$A8A(A5)         und Wert merken
007CDC 6076                bra     $7D54               fertig

007CDE 1018                move.b  (A0)+,D0            Sound-Befehl holen
007CE0 6B2E                bmi     $7D10               Bit 7 gesetzt, dann spezieller Befehl
007CE2 13C0FFFF8800        move.b  D0,$FFFF8800        Register im Sound-Chip selektieren
007CE8 0C000007            cmp.b   #7,D0               Register 7?
007CEC 661A                bne     $7D08               nein
007CEE 1218                move.b  (A0)+,D1            Daten für Register 7
007CF0 0201003F            and.b   #$3F,D1             Bit 0 - 5 isolieren
007CF4 1039FFFF8800        move.b  $FFFF8800,D0        Mixer lesen
007CFA 020000C0            and.b   #$C0,D0             Bit 6 -7 isolieren
007CFE 8001                or.b    D1,D0               Daten dazu odern
007D00 13C0FFFF8802        move.b  D0,$FFFF8802        Byte in Soundchip-Register schreiben
007D06 60D6                bra     $7CDE               nächstes Sound Kommando
007D08 13D8FFFF8802        move.b  (A0)+,$FFFF8802     Byte direkt in Soundchip schreiben
007D0E 60CE                bra     $7CDE               nächstes Sound Kommando
```

```
007D10 5200                addq.b  #1,D0               war Befehl $FF ?
007D12 6A32                bpl     $7D46               ja
007D14 0C000081            cmp.b   #$81,D0             war Befehl $80 ?
007D18 6606                bne     $7D20               nein
007D1A 1B580A8B            move.b  (A0)+,$A8B(A5)      Byte für später merken
007D1E 60BE                bra     $7CDE               nächstes Sound Kommando

007D20 0C000082            cmp.b   #$82,D0             Kommando > $81 ?
007D24 6620                bne     $7D46               ja, Timer setzen
007D26 13D8FFFF8800        move.b  (A0)+,$FFFF8800     Register selektieren
007D2C 1018                move.b  (A0)+,D0            Increment-Wert
007D2E D12D0A8B            add.b   D0,$A8B(A5)         addieren
007D32 1018                move.b  (A0)+,D0            Endwert
007D34 13ED0A8BFFFF8802    move.b  $A8B(A5),$FFFF8802  Wert in Soundchip-Register schreiben
007D3C B02D0A8B            cmp.b   $A8B(A5),D0         Endwert erreicht ?
007D40 670E                beq     $7D50               ja
007D42 5948                subq.w  #4,A0               Soundpointer zurück auf gleiches Kommamdo
007D44 600A                bra     $7D50
007D46 1B580A8A            move.b  (A0)+,$A8A(A5)      nächsten Wert als Verzögerungstimer
007D4A 6604                bne     $7D50
007D4C 307C0000            move.w  #0,A0               Sound-Zeiger auf Null
007D50 2B480A86            move.l  A0,$A86(A5)         und merken
007D54 4CDF0103            movem.l (A7)+,D0-D1/A0      Register zurückholen
007D58 4E75                rts

*****************************************************    bellsnd, Ton nach CTRL G
007D5A 0034                dc.b    0,$34
007D5C 0100                dc.b    1,0
007D5E 0200                dc.b    2,0
007D60 0300                dc.b    3,0
007D62 0400                dc.b    4,0
007D64 0500                dc.b    5,0
007D66 0600                dc.b    6,0
007D68 07FE                dc.b    7,$FE               Envelope Kanal A
007D6A 0810                dc.b    8,$10
007D6C 0900                dc.b    9,0
007D6E 0A00                dc.b    10,0
```

```
007D70 0B00                dc.b    11,0
007D72 0C10                dc.b    12,$10
007D74 0D09                dc.b    13,9
007D76 FF00                dc.b    $FF,0

*************************************************      keyclk, Ton beim Tasten-Klick
007D78 003B                dc.b    0,$3B
007D7A 0100                dc.b    1,0
007D7C 0200                dc.b    2,0
007D7E 0300                dc.b    3,0
007D80 0400                dc.b    4,0
007D82 0500                dc.b    5,0
007D84 0600                dc.b    6,0
007D86 07FE                dc.b    7,$FE
007D88 0810                dc.b    8,$10          Envelope Kanal A
007D8A 0D03                dc.b    13,3           Envelope Single Attack
007D8C 0B80                dc.b    11,$80
007D8E 0C01                dc.b    12,1
007D90 FF00                dc.b    $FF,0

*************************************************      Hardcopy
007D92 4E560000            link    A6,#0
007D96 48E7070C            movem.l D5-D7/A4-A5,-(A7)  Register retten
007D9A 2A6E0008            move.l  8(A6),A5           Adresse des Parameterblocks
007D9E 287C00002600        move.l  #$2600,A4          Adresse des Arbeitsspeichers
007DA4 7E1E                moveq.l #30,D7             30 Bytes
007DA6 6004                bra     $7DAC
007DA8 18DD                move.b  (A5)+,(A4)+        Parameter in Arbeitsspeicher übertragen
007DAA 5347                subq.w  #1,D7
007DAC 4A47                tst.w   D7                 nächstes Byte
007DAE 6EF8                bgt     $7DA8
007DB0 4AB90000261A        tst.l   $261A              p masks, Halbtonmaske
007DB6 6708                beq     $7DC0              nicht benutzt ?
007DB8 20390000261A        move.l  $261A,D0           Maske laden
007DBE 6006                bra     $7DC6
007DC0 203C00016D4C        move.l  #$16D4C,D0         Default-Maske
007DC6 23C00000261A        move.l  D0,$261A           p masks
```

```
007DCC 0C79000100002618      cmp.w   #1,$2618        p port
007DD4 6306                  bls     $7DDC
007DD6 70FF                  moveq.l #-1,D0          Flag setzen
007DD8 60000BC6              bra     $89A0           Fehler, abbrechen

007DDC 4A7900002618          tst.w   $2618           p port
007DE2 6704                  beq     $7DE8           Centronics ?
007DE4 4240                  clr.w   D0              0= RS232
007DE6 6002                  bra     $7DEA
007DE8 7001                  moveq.l #1,D0           1= Centronics
007DEA 13C0000025FE          move.b  D0,$25FE        Druckerport
007DF0 4A7900002608          tst.w   $2608           height
007DF6 6642                  bne     $7E3A           ungleich Null ?
007DF8 6028                  bra     $7E22
007DFA 4A79000004EE          tst.w   $4EE            scrdump-Flag, nochmal ALT HELP gedrückt ?
007E00 6632                  bne     $7E34           ja, abbrechen
007E02 207900002600          move.l  $2600,A0        blkptr
007E08 1010                  move.b  (A0),D0         Byte aus Video-RAM holen
007E0A 4880                  ext.w   D0
007E0C 3E80                  move.w  D0,(A7)         auf den Stack
007E0E 61000B9A              bsr     $89AA           Byte ausgeben
007E12 52B900002600          addq.l  #1,$2600        blkptr erhöhen
007E18 4A40                  tst.w   D0              Ausgabe ok ?
007E1A 6706                  beq     $7E22           ja
007E1C 70FF                  moveq.l #-1,D0          Flag setzen
007E1E 60000B80              bra     $89A0           Fehler, abbrechen

007E22 4240                  clr.w   D0
007E24 303900002606          move.w  $2606,D0        width, Bildschirmbreite
007E2A 537900002606          subq.w  #1,$2606        width erniedrigen
007E30 4A40                  tst.w   D0              schon auf Null ?
007E32 66C6                  bne     $7DFA           nein, nächstes Byte ausgeben
007E34 4240                  clr.w   D0              Flag für ok
007E36 60000B68              bra     $89A0           fertig
```

```
007E3A 0C79000300002616     cmp.w   #3,$2616      p type, Epson S/W Matrix
007E42 6306                 bls     $7E4A
007E44 70FF                 moveq.l #-1,D0        Flag setzen
007E46 60000B58             bra     $89A0         Fehler, abbrechen

007E4A 0C79000100002610     cmp.w   #1,$2610      dstres, Druckerauflösung
007E52 6306                 bls     $7E5A
007E54 70FF                 moveq.l #-1,D0        Flag setzen
007E56 60000B48             bra     $89A0         Fehler, abbrechen

007E5A 0C7900020000260E     cmp.w   #2,$260E      srcres, Bildschirmauflösung
007E62 6306                 bls     $7E6A
007E64 70FF                 moveq.l #-1,D0        Flag setzen
007E66 60000B38             bra     $89A0         Fehler, abbrechen

007E6A 0C79000700002604     cmp.w   #7,$2604      offset testen
007E72 6306                 bls     $7E7A
007E74 70FF                 moveq.l #-1,D0        Flag setzen
007E76 60000B28             bra     $89A0         Fehler, abbrechen

007E7A 4A790000260E         tst.w   $260E         srcres, Bildschirmauflösung
007E80 6704                 beq     $7E86         low resolution ?
007E82 4240                 clr.w   D0
007E84 6002                 bra     $7E88
007E86 7001                 moveq.l #1,D0
007E88 13C000004F82         move.b  D0,$4F82      Flag für low resolution
007E8E 0C7900010000260E     cmp.w   #1,$260E      srcres
007E96 6704                 beq     $7E9C         medium resolution ?
007E98 4240                 clr.w   D0
007E9A 6002                 bra     $7E9E
007E9C 7001                 moveq.l #1,D0
007E9E 13C000004ED6         move.b  D0,$4ED6      Flag für medium resolution
007EA4 0C7900020000260E     cmp.w   #2,$260E      srcres, Bildschirmauflösung
007EAC 6704                 beq     $7EB2         High resolution ?
007EAE 4240                 clr.w   D0
007EB0 6002                 bra     $7EB4
007EB2 7001                 moveq.l #1,D0
```

```
007EB4 13C000004ED8          move.b  D0,$4ED8        Flag für high resolution
007EBA 4A7900002610          tst.w   $2610           dstres, Druckerauflösung
007EC0 6704                  beq     $7EC6           Test-Modus ?
007EC2 4240                  clr.w   D0              Qualitäts-Modus
007EC4 6002                  bra     $7EC8
007EC6 7001                  moveq.l #1,D0
007EC8 13C000004EE8          move.b  D0,$4EE8        Drucker-Modus
007ECE 0C79000100002616      cmp.w   #1,$2616        p type, ATARI Color Matrix
007ED6 6704                  beq     $7EDC
007ED8 4240                  clr.w   D0
007EDA 6002                  bra     $7EDE
007EDC 7001                  moveq.l #1,D0
007EDE 13C0000047CE          move.b  D0,$47CE        ATARI Farb-Matrix-Drucker
007EE4 0C79000200002616      cmp.w   #2,$2616        p type, ATARI Typenrad ?
007EEC 6704                  beq     $7EF2           ja
007EEE 4240                  clr.w   D0
007EF0 6002                  bra     $7EF4
007EF2 7001                  moveq.l #1,D0
007EF4 13C000004F84          move.b  D0,$4F84        Flag für ATARI Typenrad Drucker
007EFA 0C79000300002616      cmp.w   #3,$2616        p type, Epson S/W Matrix ?
007F02 6704                  beq     $7F08           ja
007F04 4240                  clr.w   D0
007F06 6002                  bra     $7F0A
007F08 7001                  moveq.l #1,D0
007F0A 13C0000047D0          move.b  D0,$47D0        Flag für Epson S/W-Matrix-Drucker
007F10 4A3900004F84          tst.b   $4F84           ATARI Typenrad-Drucker ?
007F16 6706                  beq     $7F1E           nein
007F18 70FF                  moveq.l #-1,D0
007F1A 60000A84              bra     $89A0           Fehler, abbrechen

007F1E 4A39000047D0          tst.b   $47D0           Epson S/W-Matrix-Drucker ?
007F24 670C                  beq     $7F32           nein
007F26 4A3900004EE8          tst.b   $4EE8           Drucker Testmodus ?
007F2C 6604                  bne     $7F32           nein
007F2E 7001                  moveq.l #1,D0
007F30 6008                  bra     $7F3A
007F32 103900004EE8          move.b  $4EE8,D0        Druckerauflösung
```

```
007F38 4880              ext.w   D0
007F3A 13C000004EE8      move.b  D0,$4EE8        Druckerauflösung
007F40 4A3900004F82      tst.b   $4F82           low resolution ?
007F46 6726              beq     $7F6E           nein
007F48 0C79014000002606  cmp.w   #320,$2606      width, 320 Punkte pro Zeile
007F50 631C              bls     $7F6E
007F52 4240              clr.w   D0
007F54 303900002606      move.w  $2606,D0        width
007F5A D07CFEC0          add.w   #$FEC0,D0       -320
007F5E D1790000260C      add.w   D0,$260C        plus right
007F64 33FC014000002606  move.w  #320,$2606      width, 320 Punkte pro Zeile
007F6C 6024              bra     $7F92
007F6E 0C79028000002606  cmp.w   #640,$2606      width, 640 Punkte pro Zeile
007F76 631A              bls     $7F92
007F78 4240              clr.w   D0
007F7A 303900002606      move.w  $2606,D0        width
007F80 D07CFD80          add.w   #$FD80,D0       -640
007F84 D1790000260C      add.w   D0,$260C        plus right
007F8A 33FC028000002606  move.w  #640,$2606      width auf 640
007F92 4A3900004ED8      tst.b   $4ED8           High resolution ?
007F98 660001E6          bne     $8180           ja
007F9C 4247              clr.w   D7              Zähler für Farben löschen
007F9E 600001D8          bra     $8178

007FA2 207900002612      move.l  $2612,A0        colpal
007FA8 4240              clr.w   D0
007FAA 3010              move.w  (A0),D0         Farbe holen
007FAC C07C0777          and.w   #$777,D0        nicht relevante Bits maskieren
007FB0 33C0000047BA      move.w  D0,$47BA        Farbe merken
007FB6 54B900002612      addq.l  #2,$2612        colpal-Zeiger auf nächste Farbe
007FBC 0C7907770000047BA cmp.w   #$777,$47BA     Farbe gleich weiß ?
007FC4 67000194          beq     $815A           ja
007FC8 3039000047BA      move.w  $47BA,D0        Farbe laden
007FCE C07C0007          and.w   #7,D0           Blau-Level isolieren
007FD2 33C0000035C2      move.w  D0,$35C2        und merken
007FD8 3039000047BA      move.w  $47BA,D0        Farbe laden
007FDE E840              asr.w   #4,D0
```

```
007FE0 C07C0007          and.w   #7,D0              Grün-Level isolieren
007FE4 33C000004EDA      move.w  D0,$4EDA          und merken
007FEA 3039000047BA      move.w  $47BA,D0          Farbe laden
007FF0 E040              asr.w   #8,D0
007FF2 C07C0007          and.w   #7,D0             Rot-Level isolieren
007FF6 33C000004694      move.w  D0,$4694          und merken
007FFC 4A39000047CE      tst.b   $47CE             ATARI Farb-Matrix-Drucker ?
008002 67000114          beq     $8118             nein
008006 3047              move.w  D7,A0
008008 D1C8              add.l   A0,A0
00800A D1FC00004EEC      add.l   #$4EEC,A0
008010 30B900004694      move.w  $4694,(A0)        Rot-Level
008016 3047              move.w  D7,A0
008018 D1C8              add.l   A0,A0
00801A 227C00004EEC      move.l  #$4EEC,A1
008020 30309800          move.w  0(A0,A1.l),D0
008024 B07900004EDA      cmp.w   $4EDA,D0          Grün-Level
00802A 6F08              ble     $8034
00802C 303900004EDA      move.w  $4EDA,D0          Grün-Level
008032 600E              bra     $8042
008034 3047              move.w  D7,A0
008036 D1C8              add.l   A0,A0
008038 227C00004EEC      move.l  #$4EEC,A1
00803E 30309800          move.w  0(A0,A1.l),D0
008042 3247              move.w  D7,A1
008044 D3C9              add.l   A1,A1
008046 D3FC00004EEC      add.l   #$4EEC,A1
00804C 3280              move.w  D0,(A1)
00804E 3047              move.w  D7,A0
008050 D1C8              add.l   A0,A0
008052 227C00004EEC      move.l  #$4EEC,A1
008058 30309800          move.w  0(A0,A1.l),D0
00805C B079000035C2      cmp.w   $35C2,D0          Blau-Level
008062 6F08              ble     $806C
008064 3039000035C2      move.w  $35C2,D0          Rot-Level
00806A 600E              bra     $807A
00806C 3047              move.w  D7,A0
```

```
00806E D1C8              add.l   A0,A0
008070 227C00004EEC      move.l  #$4EEC,A1
008076 30309800          move.w  0(A0,A1.l),D0
00807A 3247              move.w  D7,A1
00807C D3C9              add.l   A1,A1
00807E D3FC00004EEC      add.l   #$4EEC,A1
008084 3280              move.w  D0,(A1)
008086 303900004694      move.w  $4694,D0          Rot-Level
00808C 3247              move.w  D7,A1
00808E D3C9              add.l   A1,A1
008090 D3FC00004EEC      add.l   #$4EEC,A1
008096 3211              move.w  (A1),D1
008098 5241              addq.w  #1,D1
00809A 9041              sub.w   D1,D0
00809C 6E04              bgt     $80A2
00809E 4240              clr.w   D0
0080A0 6002              bra     $80A4
0080A2 7001              moveq.l #1,D0
0080A4 33C000004694      move.w  D0,$4694          Rot-Level
0080AA 303900004EDA      move.w  $4EDA,D0          Grün-Level
0080B0 3247              move.w  D7,A1
0080B2 D3C9              add.l   A1,A1
0080B4 D3FC00004EEC      add.l   #$4EEC,A1
0080BA 3211              move.w  (A1),D1
0080BC 5241              addq.w  #1,D1
0080BE 9041              sub.w   D1,D0
0080C0 6E04              bgt     $80C6
0080C2 4240              clr.w   D0
0080C4 6002              bra     $80C8
0080C6 7001              moveq.l #1,D0
0080C8 33C000004EDA      move.w  D0,$4EDA          Grün-Level
0080CE 3039000035C2      move.w  $35C2,D0          Rot-Level
0080D4 3247              move.w  D7,A1
0080D6 D3C9              add.l   A1,A1
0080D8 D3FC00004EEC      add.l   #$4EEC,A1
0080DE 3211              move.w  (A1),D1
0080E0 5241              addq.w  #1,D1
```

```
0080E2 9041              sub.w   D1,D0
0080E4 6E04              bgt     $80EA
0080E6 4240              clr.w   D0
0080E8 6002              bra     $80EC
0080EA 7001              moveq.l #1,D0
0080EC 33C0000035C2      move.w  D0,$35C2        Blau-Level
0080F2 303900004694      move.w  $4694,D0        Rot-Level
0080F8 E540              asl.w   #2,D0
0080FA 323900004EDA      move.w  $4EDA,D1        Grün-Level
008100 E341              asl.w   #1,D1
008102 D041              add.w   D1,D0
008104 D079000035C2      add.w   $35C2,D0        Blau-Level
00810A 3247              move.w  D7,A1
00810C D3C9              add.l   A1,A1
00810E D3FC00004698      add.l   #$4698,A1
008114 3280              move.w  D0,(A1)
008116 6040              bra     $8158
008118 303900004694      move.w  $4694,D0        Rot-Level
00811E C1FC001E          muls.w  #$1E,D0         mal 30, Gewichtung 30%
008122 323900004EDA      move.w  $4EDA,D1        Grün-Level
008128 C3FC003B          muls.w  #$3B,D1         mal 59, Gewichtung 59%
00812C D041              add.w   D1,D0
00812E 3239000035C2      move.w  $35C2,D1        Blau-Level
008134 C3FC000B          muls.w  #$B,D1          mal 11, Gewichtung 11%
008138 D041              add.w   D1,D0
00813A 48C0              ext.l   D0
00813C 81FC0064          divs.w  #$64,D0         durch 100, Skalierung
008140 3247              move.w  D7,A1
008142 D3C9              add.l   A1,A1
008144 D3FC00004EEC      add.l   #$4EEC,A1
00814A 3280              move.w  D0,(A1)
00814C 3047              move.w  D7,A0
00814E D1C8              add.l   A0,A0
008150 D1FC00004698      add.l   #$4698,A0
008156 4250              clr.w   (A0)
008158 601C              bra     $8176
```

```
00815A 3047              move.w  D7,A0
00815C D1C8              add.l   A0,A0
00815E D1FC00004698      add.l   #$4698,A0
008164 30BC0007          move.w  #7,(A0)
008168 3047              move.w  D7,A0
00816A D1C8              add.l   A0,A0
00816C D1FC00004EEC      add.l   #$4EEC,A0
008172 30BC0008          move.w  #8,(A0)
008176 5247              addq.w  #1,D7          Farbzähler erhöhen
008178 BE7C0010          cmp.w   #$10,D7        schon 16 Farben abgehandelt ?
00817C 6D00FE24          blt     $7FA2          nein, nächste Farbe
008180 4A3900004F82      tst.b   $4F82          low resolution ?
008186 6716              beq     $819E          nein
008188 7004              moveq.l #4,D0
00818A 33C000004F0C      move.w  D0,$4F0C
008190 33C000004EE2      move.w  D0,$4EE2
008196 33C000004768      move.w  D0,$4768
00819C 6038              bra     $81D6
00819E 4A3900004ED6      tst.b   $4ED6          medium resolution ?
0081A4 6718              beq     $81BE          nein
0081A6 7002              moveq.l #2,D0
0081A8 33C000004F0C      move.w  D0,$4F0C
0081AE 33C000004768      move.w  D0,$4768
0081B4 33FC000400004EE2  move.w  #4,$4EE2
0081BC 6018              bra     $81D6

0081BE 33FC000100004768  move.w  #1,$4768
0081C6 33FC000800004EE2  move.w  #8,$4EE2
0081CE 33FC000200004F0C  move.w  #2,$4F0C
0081D6 4A39000047D0      tst.b   $47D0          Epson S/W-Matrix-Drucker ?
0081DC 6706              beq     $81E4          nein
0081DE 3F3C0002          move.w  #2,-(A7)
0081E2 6004              bra     $81E8
0081E4 3F3C0001          move.w  #1,-(A7)
0081E8 303900004F0C      move.w  $4F0C,D0
0081EE 48C0              ext.l   D0
0081F0 81DF              divs.w  (A7)+,D0
```

```
0081F2 33C000004F0C     move.w  D0,$4F0C
0081F8 4240             clr.w   D0
0081FA 30390000260A     move.w  $260A,D0        left
008200 D07900002606     add.w   $2606,D0        heigth
008206 D0790000260C     add.w   $260C,D0        right
00820C C0F900004768     mulu.w  $4768,D0
008212 E848             lsr.w   #4,D0
008214 33C000004696     move.w  D0,$4696
00821A 303900004696     move.w  $4696,D0
008220 C1F900004EE2     muls.w  $4EE2,D0
008226 33C000003E80     move.w  D0,$3E80
00822C 203900002600     move.l  $2600,D0        blkptr
008232 C0BCFFFFFFFE     and.l   #$FFFFFFFE,D0   gerade Adresse erzeugen
008238 23C0000046B8     move.l  D0,$46B8        und merken
00823E 203900002600     move.l  $2600,D0        blkptr
008244 B0B900046B8      cmp.l   $46B8,D0
00824A 660A             bne     $8256
00824C 4240             clr.w   D0
00824E 303900002604     move.w  $2604,D0        offset
008254 600A             bra     $8260
008256 4240             clr.w   D0
008258 303900002604     move.w  $2604,D0        offset
00825E 5040             addq.w  #8,D0
008260 33C0000047BC     move.w  D0,$47BC
008266 4279000012EA     clr.w   $12EA
00826C 60000716         bra     $8984
008270 4A79000004EE     tst.w   $4EE            dumpflg, ALT HELP nochmal gedrückt ?
008276 6600071E         bne     $8996           ja, abbrechen
00827A 13FC000100003628 move.b  #1,$3628
008282 4240             clr.w   D0
008284 303900002606     move.w  $2606,D0        width
00828A C0F900004768     mulu.w  $4768,D0
008290 E848             lsr.w   #4,D0
008292 907900004768     sub.w   $4768,D0
008298 E348             lsl.w   #1,D0
00829A 4840             swap    D0
00829C 4240             clr.w   D0
```

```
00829E 4840                    swap    D0
0082A0 D0B9000046B8            add.l   $46B8,D0          Zeiger auf Video-RAM
0082A6 23C000004EDC            move.l  D0,$4EDC
0082AC 700F                    moveq.l #15,D0
0082AE 4241                    clr.w   D1
0082B0 323900002606            move.w  $2606,D1          width
0082B6 C27C000F                and.w   #$F,D1
0082BA 9041                    sub.w   D1,D0
0082BC 33C000004F12            move.w  D0,$4F12
0082C2 33F90000260600003E2C    move.w  $2606,$3E2C       width
0082CC 60000124                bra     $83F2
0082D0 4240                    clr.w   D0
0082D2 303900002608            move.w  $2608,D0          height
0082D8 9079000012EA            sub.w   $12EA,D0
0082DE 4840                    swap    D0
0082E0 4240                    clr.w   D0
0082E2 4840                    swap    D0
0082E4 80F900004EE2            divu.w  $4EE2,D0
0082EA 6708                    beq     $82F4
0082EC 303900004EE2            move.w  $4EE2,D0
0082F2 600E                    bra     $8302
0082F4 4240                    clr.w   D0
0082F6 303900002608            move.w  $2608,D0          height
0082FC 9079000012EA            sub.w   $12EA,D0
008302 33C000004ED2            move.w  D0,$4ED2
008308 23F900004EDC0000493C    move.l  $4EDC,$493C
008312 4247                    clr.w   D7
008314 6000009E                bra     $83B4
008318 427900004F1A            clr.w   $4F1A
00831E 33FC000100004F0E        move.w  #1,$4F0E
008326 23F90000493C000047BE    move.l  $493C,$47BE
008330 4246                    clr.w   D6
008332 6030                    bra     $8364
008334 2079000047BE            move.l  $47BE,A0
00833A 3010                    move.w  (A0),D0
00833C 720F                    moveq.l #15,D1
00833E 927900004F12            sub.w   $4F12,D1
```

```
008344 E260              asr.w   D1,D0
008346 C07C0001          and.w   #1,D0
00834A C1F900004F0E      muls.w  $4F0E,D0
008350 D17900004F1A      add.w   D0,$4F1A
008356 54B9000047BE      addq.l  #2,$47BE
00835C E1F900004F0E      asl.w   $4F0E
008362 5246              addq.w  #1,D6
008364 BC7900004768      cmp.w   $4768,D6
00836A 6DC8              blt     $8334
00836C 4A3900004ED8      tst.b   $4ED8            High resolution ?
008372 6712              beq     $8386            nein
008374 4A7900004F1A      tst.w   $4F1A
00837A 6708              beq     $8384
00837C 423900003628      clr.b   $3628
008382 603A              bra     $83BE
008384 601C              bra     $83A2
008386 307900004F1A      move.w  $4F1A,A0
00838C D1C8              add.l   A0,A0
00838E D1FC00004EEC      add.l   #$4EEC,A0
008394 0C500008          cmp.w   #8,(A0)
008398 6708              beq     $83A2
00839A 423900003628      clr.b   $3628
0083A0 601C              bra     $83BE
0083A2 303900004696      move.w  $4696,D0
0083A8 E340              asl.w   #1,D0
0083AA 48C0              ext.l   D0
0083AC D1B90000493C      add.l   D0,$493C
0083B2 5247              addq.w  #1,D7
0083B4 BE7900004ED2      cmp.w   $4ED2,D7
0083BA 6D00FF5C          blt     $8318
0083BE 4A3900003628      tst.b   $3628
0083C4 6736              beq     $83FC
0083C6 537900004F12      subq.w  #1,$4F12
0083CC 4A7900004F12      tst.w   $4F12
0083D2 6C18              bge     $83EC
0083D4 303900004768      move.w  $4768,D0
0083DA E340              asl.w   #1,D0
```

```
0083DC 48C0              ext.l   D0
0083DE 91B900004EDC      sub.l   D0,$4EDC
0083E4 33FC000F00004F12  move.w  #$F,$4F12
0083EC 537900003E2C      subq.w  #1,$3E2C
0083F2 4A7900003E2C      tst.w   $3E2C
0083F8 6E00FED6          bgt     $82D0
0083FC 3E3900003E2C      move.w  $3E2C,D7
008402 CFF900004F0C      muls.w  $4F0C,D7
008408 4A39000047D0      tst.b   $47D0            Epson S/W-Matrix-Drucker ?
00840E 670A              beq     $841A            nein
008410 3007              move.w  D7,D0
008412 48C0              ext.l   D0
008414 81FC0002          divs.w  #2,D0
008418 6002              bra     $841C
00841A 4240              clr.w   D0
00841C DE40              add.w   D0,D7
00841E 3007              move.w  D7,D0
008420 48C0              ext.l   D0
008422 81FC0100          divs.w  #$100,D0
008426 4840              swap    D0
008428 13C000003E86      move.b  D0,$3E86
00842E 3007              move.w  D7,D0
008430 48C0              ext.l   D0
008432 81FC0100          divs.w  #$100,D0
008436 13C000003E88      move.b  D0,$3E88
00843C 4279000047D2      clr.w   $47D2
008442 600004A6          bra     $88EA

008446 427900004F88      clr.w   $4F88
00844C 60000450          bra     $889E
008450 4A39000047CE      tst.b   $47CE            ATARI Farb-Matrix-Drucker ?
008456 675A              beq     $84B2            nein
008458 4A3900004ED8      tst.b   $4ED8            High resolution ?
00845E 6652              bne     $84B2            ja
008460 4A7900004F88      tst.w   $4F88
008466 6616              bne     $847E
008468 2EBC00016D5E      move.l  #$16D5E,(A7)     ESC 'X' 6
```

```
00846E 6100058E          bsr     $89FE           String an Drucker ausgeben
008472 4A40              tst.w   D0              Fehler ?
008474 6706              beq     $847C           nein
008476 70FF              moveq.l #-1,D0          Flag für Fehler
008478 60000526          bra     $89A0           Fehler, abbrechen

00847C 6034              bra     $84B2
00847E 0C79000100004F88  cmp.w   #1,$4F88
008486 6616              bne     $849E
008488 2EBC00016D63      move.l  #$16D63,(A7)    ESC 'X' 5
00848E 6100056E          bsr     $89FE           String an Drucker ausgeben
008492 4A40              tst.w   D0              Fehler ?
008494 6706              beq     $849C           nein
008496 70FF              moveq.l #-1,D0          Flag für Fehler
008498 60000506          bra     $89A0           Fehler, abbrechen

00849C 6014              bra     $84B2
00849E 2EBC00016D68      move.l  #$16D68,(A7)    ESC 'X' 3
0084A4 61000558          bsr     $89FE           String an Drucker ausgeben
0084A8 4A40              tst.w   D0              Fehler ?
0084AA 6706              beq     $84B2           nein
0084AC 70FF              moveq.l #-1,D0          Flag für Fehler
0084AE 600004F0          bra     $89A0           Fehler, abbrechen

0084B2 4A39000047D0      tst.b   $47D0          Epson S/W-Matrix-Drucker ?
0084B8 6708              beq     $84C2           nein
0084BA 2EBC00016D6D      move.l  #$16D6D,(A7)    ESC 'L', Bitimage 960 Punkte/Zeile
0084C0 6006              bra     $84C8
0084C2 2EBC00016D71      move.l  #$16D71,(A7)    ESC 'Y', Bitimage 960 Punkte/Zeile
0084C8 61000534          bsr     $89FE           String an Drucker ausgeben
0084CC 4A40              tst.w   D0              Fehler ?
0084CE 6706              beq     $84D6           nein
0084D0 70FF              moveq.l #-1,D0          Flag für Fehler
0084D2 600004CC          bra     $89A0           Fehler, abbrechen
```

```
0084D6 103900003E86          move.b  $3E86,D0        Byte holen
0084DC 4880                  ext.w   D0              Low-Byte der Anzahl
0084DE 3E80                  move.w  D0,(A7)         auf Stack
0084E0 610004C8              bsr     $89AA           Ausgabe
0084E4 4A40                  tst.w   D0              Fehler ?
0084E6 6706                  beq     $84EE           nein
0084E8 70FF                  moveq.l #-1,D0          Flag für Fehler
0084EA 600004B4              bra     $89A0           Fehler, abbrechen

0084EE 103900003E88          move.b  $3E88,D0        Byte holen
0084F4 4880                  ext.w   D0              Hi-Byte der Anzahl
0084F6 3E80                  move.w  D0,(A7)         auf Stack
0084F8 610004B0              bsr     $89AA           Ausgabe
0084FC 4A40                  tst.w   D0              Fehler ?
0084FE 6706                  beq     $8506           nein
008500 70FF                  moveq.l #-1,D0          Flag für Fehler
008502 6000049C              bra     $89A0           Fehler, abbrechen

008506 13FC000100004EEA      move.b  #1,$4EEA
00850E 23F9000046B800004EDC  move.l  $46B8,$4EDC
008518 33F9000047BC00004F12  move.w  $47BC,$4F12
008522 4279000012E8          clr.w   $12E8
008528 6000034C              bra     $8876

00852C 4247                  clr.w   D7
00852E 600C                  bra     $853C
008530 3047                  move.w  D7,A0
008532 D1FC000047D4          add.l   #$47D4,A0
008538 4210                  clr.b   (A0)
00853A 5247                  addq.w  #1,D7
00853C BE7C0008              cmp.w   #8,D7
008540 6DEE                  blt     $8530
008542 4247                  clr.w   D7
008544 6010                  bra     $8556
008546 3047                  move.w  D7,A0
008548 D1C8                  add.l   A0,A0
00854A D1FC00003E8A          add.l   #$3E8A,A0
```

```
008550 30BC0007              move.w  #7,(A0)
008554 5247                  addq.w  #1,D7
008556 BE7C0004              cmp.w   #4,D7
00855A 6DEA                  blt     $8546
00855C 4240                  clr.w   D0
00855E 303900002608          move.w  $2608,D0            height
008564 9079000012EA          sub.w   $12EA,D0
00856A 4840                  swap    D0
00856C 4240                  clr.w   D0
00856E 4840                  swap    D0
008570 80F900004EE2          divu.w  $4EE2,D0
008576 6708                  beq     $8580
008578 303900004EE2          move.w  $4EE2,D0
00857E 600E                  bra     $858E
008580 4240                  clr.w   D0
008582 303900002608          move.w  $2608,D0            height
008588 9079000012EA          sub.w   $12EA,D0
00858E 33C000004ED2          move.w  D0,$4ED2
008594 23F900004EDC0000493C  move.l  $4EDC,$493C
00859E 4247                  clr.w   D7
0085A0 600000F8              bra     $869A

0085A4 427900004F1A          clr.w   $4F1A
0085AA 33FC000100004F0E      move.w  #1,$4F0E
0085B2 23F90000493C000047BE  move.l  $493C,$47BE
0085BC 4246                  clr.w   D6
0085BE 6030                  bra     $85F0
0085C0 2079000047BE          move.l  $47BE,A0
0085C6 3010                  move.w  (A0),D0
0085C8 720F                  moveq.l #15,D1
0085CA 927900004F12          sub.w   $4F12,D1
0085D0 E260                  asr.w   D1,D0
0085D2 C07C0001              and.w   #1,D0
0085D6 C1F900004F0E          muls.w  $4F0E,D0
0085DC D17900004F1A          add.w   D0,$4F1A
0085E2 54B9000047BE          addq.l  #2,$47BE
0085E8 E1F900004F0E          asl.w   $4F0E
```

```
0085EE 5246              addq.w  #1,D6
0085F0 BC7900004768      cmp.w   $4768,D6
0085F6 6DC8              blt     $85C0
0085F8 4A3900004ED8      tst.b   $4ED8           High resolution ?
0085FE 6722              beq     $8622           nein
008600 4A7900004F1A      tst.w   $4F1A
008606 670C              beq     $8614
008608 20790000261A      move.l  $261A,A0        p masks
00860E 1010              move.b  (A0),D0
008610 4880              ext.w   D0
008612 6002              bra     $8616
008614 4240              clr.w   D0
008616 3247              move.w  D7,A1
008618 D3FC000047D4      add.l   #$47D4,A1
00861E 1280              move.b  D0,(A1)
008620 6066              bra     $8688

008622 3047              move.w  D7,A0
008624 D1C8              add.l   A0,A0
008626 D1FC00003E8A      add.l   #$3E8A,A0
00862C 327900004F1A      move.w  $4F1A,A1
008632 D3C9              add.l   A1,A1
008634 D3FC00004698      add.l   #$4698,A1
00863A 3091              move.w  (A1),(A0)
00863C 3047              move.w  D7,A0
00863E D0C8              add.w   A0,A0
008640 D1FC000047D4      add.l   #$47D4,A0
008646 327900004F1A      move.w  $4F1A,A1
00864C D3C9              add.l   A1,A1
00864E D3FC00004EEC      add.l   #$4EEC,A1
008654 3251              move.w  (A1),A1
008656 D2C9              add.w   A1,A1
008658 D3F90000261A      add.l   $261A,A1        p masks
00865E 1091              move.b  (A1),(A0)
008660 3047              move.w  D7,A0
008662 D0C8              add.w   A0,A0
008664 D1FC000047D4      add.l   #$47D4,A0
```

```
00866A 327900004F1A        move.w  $4F1A,A1
008670 D3C9                add.l   A1,A1
008672 D3FC00004EEC        add.l   #$4EEC,A1
008678 3251                move.w  (A1),A1
00867A D2C9                add.w   A1,A1
00867C D3F90000261A        add.l   $261A,A1            p masks
008682 116900010001        move.b  1(A1),1(A0)
008688 303900004696        move.w  $4696,D0
00868E E340                asl.w   #1,D0
008690 48C0                ext.l   D0
008692 D1B90000493C        add.l   D0,$493C
008698 5247                addq.w  #1,D7
00869A BE7900004ED2        cmp.w   $4ED2,D7
0086A0 6D00FF02            blt     $85A4
0086A4 4A39000047CE        tst.b   $47CE               ATARI Farb-Matrix-Drucker ?
0086AA 670000DE            beq     $878A               nein
0086AE 4A3900004ED8        tst.b   $4ED8               High resolution ?
0086B4 660000D4            bne     $878A               ja
0086B8 4247                clr.w   D7
0086BA 600000C4            bra     $8780
0086BE 423900004EE0        clr.b   $4EE0
0086C4 4A7900004F88        tst.w   $4F88
0086CA 6624                bne     $86F0
0086CC 3047                move.w  D7,A0
0086CE D1C8                add.l   A0,A0
0086D0 227C00003E8A        move.l  #$3E8A,A1
0086D6 30309800            move.w  0(A0,A1.l),D0
0086DA 48C0                ext.l   D0
0086DC 81FC0002            divs.w  #$2,D0
0086E0 4840                swap    D0
0086E2 4A40                tst.w   D0
0086E4 6708                beq     $86EE
0086E6 13FC000100004EE0    move.b  #$1,$4EE0
0086EE 606C                bra     $875C
0086F0 0C79000100004F88    cmp.w   #1,$4F88
0086F8 664A                bne     $8744
0086FA 3047                move.w  D7,A0
```

```
0086FC D1C8              add.l   A0,A0
0086FE D1FC00003E8A      add.l   #$3E8A,A0
008704 0C500002          cmp.w   #2,(A0)
008708 6730              beq     $873A
00870A 3047              move.w  D7,A0
00870C D1C8              add.l   A0,A0
00870E D1FC00003E8A      add.l   #$3E8A,A0
008714 0C500003          cmp.w   #3,(A0)
008718 6720              beq     $873A
00871A 3047              move.w  D7,A0
00871C D1C8              add.l   A0,A0
00871E D1FC00003E8A      add.l   #$3E8A,A0
008724 0C500006          cmp.w   #6,(A0)
008728 6710              beq     $873A
00872A 3047              move.w  D7,A0
00872C D1C8              add.l   A0,A0
00872E D1FC00003E8A      add.l   #$3E8A,A0
008734 0C500007          cmp.w   #7,(A0)
008738 6608              bne     $8742
00873A 13FC000100004EE0  move.b  #$1,$4EE0
008742 6018              bra     $875C
008744 3047              move.w  D7,A0
008746 D1C8              add.l   A0,A0
008748 D1FC00003E8A      add.l   #$3E8A,A0
00874E 0C500003          cmp.w   #3,(A0)
008752 6F08              ble     $875C
008754 13FC000100004EE0  move.b  #$1,$4EE0
00875C 4A3900004EE0      tst.b   $4EE0
008762 671A              beq     $877E
008764 3047              move.w  D7,A0
008766 D0C8              add.w   A0,A0
008768 D1FC000047D4      add.l   #$47D4,A0
00876E 4210              clr.b   (A0)
008770 3047              move.w  D7,A0
008772 D0C8              add.w   A0,A0
008774 D1FC000047D4      add.l   #$47D4,A0
00877A 42280001          clr.b   1(A0)
```

```
00877E 5247              addq.w  #1,D7
008780 BE7900004ED2      cmp.w   $4ED2,D7
008786 6D00FF36          blt     $86BE
00878A 7E04              moveq.l #4,D7
00878C 60000086          bra     $8814
008790 4239000035BE      clr.b   $35BE
008796 33FC008000004F10  move.w  #$80,$4F10
00879E 4246              clr.w   D6
0087A0 603E              bra     $87E0
0087A2 207C000047D4      move.l  #$47D4,A0
0087A8 10306000          move.b  0(A0,D6.w),D0
0087AC 4880              ext.w   D0
0087AE 7207              moveq.l #7,D1
0087B0 9247              sub.w   D7,D1
0087B2 E260              asr.w   D1,D0
0087B4 C07C0001          and.w   #$1,D0
0087B8 C1F900004F10      muls.w  $4F10,D0
0087BE 1239000035BE      move.b  $35BE,D1
0087C4 D200              add.b   D0,D1
0087C6 13C1000035BE      move.b  D1,$35BE
0087CC 303900004F10      move.w  $4F10,D0
0087D2 48C0              ext.l   D0
0087D4 81FC0002          divs.w  #$2,D0
0087D8 33C000004F10      move.w  D0,$4F10
0087DE 5246              addq.w  #1,D6
0087E0 BC7C0008          cmp.w   #$8,D6
0087E4 6DBC              blt     $87A2
0087E6 1039000035BE      move.b  $35BE,D0
0087EC 4880              ext.w   D0
0087EE 3E80              move.w  D0,(A7)
0087F0 610001B8          bsr     $89AA            Ausgabe
0087F4 4A40              tst.w   D0
0087F6 6706              beq     $87FE
0087F8 70FF              moveq.l #-1,D0
0087FA 600001A4          bra     $89A0            Fehler, abbrechen
```

```
0087FE 4A3900004EEA        tst.b   $4EEA
008804 6704                beq     $880A
008806 4240                clr.w   D0
008808 6002                bra     $880C
00880A 7001                moveq.l #1,D0
00880C 13C000004EEA        move.b  D0,$4EEA
008812 5247                addq.w  #1,D7
008814 303900004F0C        move.w  $4F0C,D0
00881A 5840                addq.w  #4,D0
00881C BE40                cmp.w   D0,D7
00881E 6D00FF70            blt     $8790
008822 4A39000047D0        tst.b   $47D0          Epson S/W Matrix-Drucker ?
008828 6720                beq     $884A          nein
00882A 4A3900004EEA        tst.b   $4EEA
008830 6718                beq     $884A
008832 1039000035BE        move.b  $35BE,D0
008838 4880                ext.w   D0
00883A 3E80                move.w  D0,(A7)
00883C 6100016C            bsr     $89AA          Ausgabe
008840 4A40                tst.w   D0             ok ?
008842 6706                beq     $884A          ja
008844 70FF                moveq.l #-1,D0         Flag setzen
008846 60000158            bra     $89A0          Fehler, abbrechen

00884A 527900004F12        addq.w  #1,$4F12
008850 0C79000F00004F12    cmp.w   #15,$4F12
008858 6F16                ble     $8870
00885A 303900004768        move.w  $4768,D0
008860 E340                asl.w   #1,D0
008862 48C0                ext.l   D0
008864 D1B900004EDC        add.l   D0,$4EDC
00886A 427900004F12        clr.w   $4F12
008870 5279000012E8        addq.w  #1,$12E8
008876 3039000012E8        move.w  $12E8,D0
00887C B07900003E2C        cmp.w   $3E2C,D0
008882 6D00FCA8            blt     $852C
008886 3EBC000D            move.w  #$D,(A7)       CR
```

```
00888A 6100011E          bsr     $89AA          Ausgabe
00888E 4A40              tst.w   D0             ok ?
008890 6706              beq     $8898          ja
008892 70FF              moveq.l #-1,D0         Flag setzen
008894 6000010A          bra     $89A0          Fehler, abbrechen

008898 527900004F88      addq.w  #1,$4F88
00889E 4A39000047CE      tst.b   $47CE          ATARI Farb-Matrix-Drucker ?
0088A4 670C              beq     $88B2          nein
0088A6 4A3900004ED8      tst.b   $4ED8          High resolution ?
0088AC 6604              bne     $88B2          ja
0088AE 7003              moveq.l #3,D0
0088B0 6002              bra     $88B4
0088B2 7001              moveq.l #1,D0
0088B4 B07900004F88      cmp.w   $4F88,D0
0088BA 6E00FB94          bgt     $8450
0088BE 2EBC00016D75      move.l  #$16D75,(A7)   ESC '3' 1, 1/216" Zeilenabstand
0088C4 61000138          bsr     $89FE          String an Drucker ausgeben
0088C8 4A40              tst.w   D0
0088CA 6706              beq     $88D2
0088CC 70FF              moveq.l #-1,D0         Flag setzen
0088CE 600000D0          bra     $89A0          Fehler, abbrechen

0088D2 3EBC000A          move.w  #$A,(A7)       LF
0088D6 610000D2          bsr     $89AA          Ausgabe
0088DA 4A40              tst.w   D0             ok ?
0088DC 6706              beq     $88E4          ja
0088DE 70FF              moveq.l #-1,D0         Flag setzen
0088E0 600000BE          bra     $89A0          Fehler, abbrechen

0088E4 5279000047D2      addq.w  #1,$47D2
0088EA 4A3900004EE8      tst.b   $4EE8          Druckerauflösung
0088F0 6704              beq     $88F6
0088F2 7001              moveq.l #1,D0
0088F4 6002              bra     $88F8
0088F6 7002              moveq.l #2,D0
0088F8 B079000047D2      cmp.w   $47D2,D0
```

```
0088FE 6E00FB46          bgt     $8446
008902 4A3900004EE8      tst.b   $4EE8          Druckerauflösung
008908 673E              beq     $8948
00890A 4247              clr.w   D7
00890C 6028              bra     $8936
00890E 2EBC00016D7A      move.l  #$16D7A,(A7)   ESC '3' 1, 1/216" Zeilenabstand
008914 610000E8          bsr     $89FE          String an Drucker ausgeben
008918 4A40              tst.w   D0             Fehler bei Ausgabe ?
00891A 6706              beq     $8922          nein
00891C 70FF              moveq.l #-1,D0         Flag setzen
00891E 60000080          bra     $89A0          Fehler, abbrechen

008922 3EBC000A          move.W  #$A,(A7)       LF
008926 61000082          bsr     $89AA          Ausgabe
00892A 4A40              tst.w   D0             Fehler bei Ausgabe ?
00892C 6706              beq     $8934          nein
00892E 70FF              moveq.l #-1,D0         Flag setzen
008930 6000006E          bra     $89A0          Fehler, abbrechen

008934 5247              addq.W  #1,D7
008936 4A39000047D0      tst.b   $47D0          Epson S/W-Matrix-Drucker ?
00893C 6704              beq     $8942          nein
00893E 7002              moveq.l #2,D0
008940 6002              bra     $8944
008942 7001              moveq.l #1,D0
008944 BE40              cmp.W   D0,D7
008946 6DC6              blt     $890E
008948 2EBC00016D7F      move.l  #$16D7F,(A7)   ESC '1', 7/72" Zeilenabstand
00894E 610000AE          bsr     $89FE          String an Drucker ausgeben
008952 4A40              tst.w   D0             Fehler bei Ausgabe ?
008954 6704              beq     $895A          nein
008956 70FF              moveq.l #-1,D0         Flag setzen
008958 6046              bra     $89A0          Fehler, abbrechen
```

```
00895A 3EBC000A            move.w  #$A,(A7)            LF
00895E 614A                bsr     $89AA               Ausgabe
008960 4A40                tst.w   D0                  Fehler bei Ausgabe ?
008962 6704                beq     $8968               nein
008964 70FF                moveq.l #-1,D0
008966 6038                bra     $89A0               Fehler, abbrechen

008968 303900003E80        move.w  $3E80,D0
00896E E340                asl.w   #1,D0
008970 48C0                ext.l   D0
008972 D1B900046B8         add.l   D0,$46B8
008978 303900004EE2        move.w  $4EE2,D0
00897E D179000012EA        add.w   D0,$12EA
008984 4240                clr.w   D0
008986 303900002608        move.w  $2608,D0            height
00898C B079000012EA        cmp.w   $12EA,D0
008992 6200F8DC            bhi     $8270
008996 2EBC00016D83        move.l  #$16D83,(A7)        ESC '2', 1/6" Zoll Zeilenabstand
00899C 6160                bsr     $89FE               String an Drucker ausgeben
00899E 4240                clr.w   D0                  Flag für ok
0089A0 4A9F                tst.l   (A7)+
0089A2 4CDF30C0            movem.l (A7)+,D6-D7/A4-A5   Register zurückholen
0089A6 4E5E                unlk    A6
0089A8 4E75                rts

**********************************************         ein Zeichen ausgeben
0089AA 4E56FFFC            link    A6,#-4
0089AE 4A39000025FE        tst.b   $25FE               Druckerport
0089B4 6722                beq     $89D8               RS232 ?
0089B6 102E0009            move.b  9(A6),D0            auszugebendes Zeichen
0089BA 4880                ext.w   D0                  auf Wort erweitern
0089BC 3E80                move.w  D0,(A7)             und auf Stack
0089BE 102E0009            move.b  9(A6),D0            auszugebendes Zeichen
0089C2 4880                ext.w   D0                  auf Wort erweitern
0089C4 3F00                move.w  D0,-(A7)            und nochmal auf Stack (?)
0089C6 4EB900008A2C        jsr     $8A2C               über Centronics ausgeben
0089CC 548F                addq.l  #2,A7               Stackpointer korrigieren
```

```
0089CE 4A40              tst.w   D0             ok ?
0089D0 6604              bne     $89D6           ja
0089D2 70FF              moveq.l #-1,D0          Flag für Fehler
0089D4 6024              bra     $89FA           fertig
0089D6 6020              bra     $89F8           fehlerfreier Abschluß

0089D8 102E0009          move.b  9(A6),D0        auszugebendes Zeichen
0089DC 4880              ext.w   D0              auf Wort erweitern
0089DE 3E80              move.w  D0,(A7)         und auf Stack
0089E0 102E0009          move.b  9(A6),D0        auszugebendes Zeichen
0089E4 4880              ext.w   D0              auf Wort erweiteren
0089E6 3F00              move.w  D0,-(A7)        und nochmal auf Stack (?)
0089E8 4EB900008A46      jsr     $8A46           über RS232 ausgeben
0089EE 548F              addq.l  #2,A7           Stackpointer korrigieren
0089F0 4A40              tst.w   D0              ok ?
0089F2 6604              bne     $89F8           ja
0089F4 70FF              moveq.l #-1,D0          Fehlerflag setzen
0089F6 6002              bra     $89FA
0089F8 4240              clr.w   D0
0089FA 4E5E              unlk    A6
0089FC 4E75              rts

*********************************************    String an Drucker ausgeben
0089FE 4E56FFFC          link    A6,#-4
008A02 6018              bra     $8A1C
008A04 206E0008          move.l  8(A6),A0        Stringadresse holen
008A08 1010              move.b  (A0),D0         Zeichen des Strings
008A0A 4880              ext.w   D0              auf Wort erweitern
008A0C 3E80              move.w  D0,(A7)
008A0E 619A              bsr     $89AA           Zeichen ausgeben
008A10 52AE0008          addq.l  #1,8(A6)        Zeiger auf nächstes Zeichen
008A14 4A40              tst.w   D0              fehlerfreie Ausgabe ?
008A16 6704              beq     $8A1C           ja
008A18 70FF              moveq.l #-1,D0          Fehler
008A1A 600C              bra     $8A28           Ausgabe abbrechen
008A1C 206E0008          move.l  8(A6),A0        Zeiger auf String
008A20 0C1000FF          cmp.b   #$FF,(A0)       $FF als Ende-Kennzeichen
```

```
008A24 66DE              bne     $8A04           nächstes Zeichen
008A26 4240              clr.w   D0              ok
008A28 4E5E              unlk    A6
008A2A 4E75              rts

*****************************************************  Centronics-Ausgabe
008A2C 302F0006          move.w  6(A7),D0        auszugebendes Zeichen
008A30 48E71F1E          movem.l D3-D7/A3-A6,-(A7) Register retten
008A34 3F00              move.w  D0,-(A7)        auszugendendes Zeichen
008A36 3F00              move.w  D0,-(A7)
008A38 9BCD              sub.l   A5,A5           A5 löschen
008A3A 6100E198          bsr     $6BD4           Zeichen auf Centronicsport ausgeben
008A3E 584F              addq.w  #4,A7           Stackpointer korrigieren
008A40 4CDF78F8          movem.l (A7)+,D3-D7/A3-A6 Register zurückholen
008A44 4E75              rts

*****************************************************  RS232 Ausgabe
008A46 302F0006          move.w  6(A7),D0        auszugebendes Zeichen
008A4A 48E71F1E          movem.l D3-D7/A3-A6,-(A7) Register retten
008A4E 3F00              move.w  D0,-(A7)        auszugebendes Zeichen
008A50 3F00              move.w  D0,-(A7)
008A52 9BCD              sub.l   A5,A5           A5 löschen
008A54 6100E258          bsr     $6CAE           Zeichen über RS232 ausgeben
008A58 584F              addq.w  #4,A7           Stackpointer korrigieren
008A5A 4CDF78F8          movem.l (A7)+,D3-D7/A3-A6 Register zurückholen
008A5E 4E75              rts

*****************************************************  VDI ESCAPE-Funktionen
008A60 207900002580      move.l  $2580,A0        Zeiger auf CONTRL Array
008A66 3028000A          move.w  10(A0),D0       Funktionsnummer
008A6A B07C0013          cmp.w   #$13,D0         größer 19 ?
008A6E 6236              bhi     $8AA6
008A72 307B000A          move.w  $8A7E(PC,D0.W),A0  relative Adresse aus Tabelle holen
008A76 D1FC00008C7A      add.l   #$8C7A,A0       Basisadresse addieren
008A7C 4ED0              jmp     (A0)            Routine ausführen
```

```
008A7E 0000             dc.w    $8C7A-$8C7A     0, rts
008A80 FFD8             dc.w    $8C52-$8C7A     1, Inquire addressable alpha character cells
008A82 0012             dc.w    $8C8C-$8C7A     2, Exit alpha mode
008A84 000C             dc.w    $8C86-$8C7A     3, Enter alpha mode
008A86 001A             dc.w    $8C94-$8C7A     4, Alpha cursor up
008A88 002E             dc.w    $8CA8-$8C7A     5, Alpha cursor down
008A8A 0048             dc.w    $8CC2-$8C7A     6, Alpha cursor right
008A8C 0062             dc.w    $8CDC-$8C7A     7, Alpha cursor left
008A8E 0076             dc.w    $8CF0-$8C7A     8, Home alpha cursor
008A90 007E             dc.w    $8CF8-$8C7A     9, Erase to end of alpha screen
008A92 00AA             dc.w    $8D24-$8C7A     10, Erase to end of alpha text line
008A94 0114             dc.w    $8D8E-$8C7A     11, Direct alpha cursor address
008A96 0128             dc.w    $8DA2-$8C7A     12, Output cursor adressable alpha text
008A98 014E             dc.w    $8DC8-$8C7A     13, Reverse video on
008A9A 0158             dc.w    $8DD2-$8C7A     14, Reverse video off
008A9C 0162             dc.w    $8DDC-$8C7A     15, Inquire current alpha cursor address
008A9E 018C             dc.w    $8E06-$8C7A     16, Inquire tablet status
008AA0 0002             dc.w    $8C7C-$8C7A     17, Hardcopy
008AA2 01A4             dc.w    $8E1E-$8C7A     18, Place graphic cursor at location
008AA4 01B4             dc.w    $8E2E-$8C7A     19, Remove last graphic cursor

*************************************************
008AA6 B07C0065         cmp.w   #101,D0         ESC VDI 101 ?
008AAA 670A             beq     $8AB6
008AAC B07C0066         cmp.w   #102,D0         ESC VDI 102 ?
008AB0 6700094E         beq     $9400           ja, Fontdaten initialisieren
008AB4 4E75             rts

*************************************************   VDI ESC 101
008AB6 61000448         bsr     $8F00           Cursor aus
008ABA 207900002584     move.l  $2584,A0        Zeiger auf INTIN Array
008AC0 3010             move.w  (A0),D0         Parameter holen
008AC2 C0F90000257E     mulu.w  $257E,D0        mal Anzahl Bytes pro Bildschirmzeile
008AC8 33C00000255E     move.w  D0,$255E
008ACE 6000041E         bra     $8EEE
```

```
*****************************************************
008AD2 322F0006              move.w  6(A7),D1        ascout
008AD6 024100FF              and.w   #$FF,D1         Zeichen von Stack holen
008ADA 600005D2              bra     $90AE           nur Lo-Byte verarbeiten
                                                     Zeichen ausgeben
*****************************************************
008ADE 322F0006              move.w  6(A7),D1        conout
008AE2 024100FF              and.w   #$FF,D1         Zeichen vom Stack holen
008AE6 2079000004A8          move.l  $4A8,A0         nur Lo-Byte verarbeiten
008AEC 4ED0                  jmp     (A0)            conout-Vektor holen
                                                     und Routine ausführen
*****************************************************
008AEE B27C0020              cmp.w   #$20,D1         Standard conout
008AF2 6C0005BA              bge     $90AE           Controlcode ?
008AF6 B23C001B              cmp.b   #$1B,D1         nein, Zeichen ausgeben
008AFA 660C                  bne     $8B08           ESC ?
008AFC 23FC00008B4A000004A8 move.l  #$8B4A,$4A8     nein, andere CTRL-Codes verarbeiten
008B06 4E75                  rts                     conout-Vektor auf ESC-Verarbeitung
*****************************************************
008B08 5F41                  subq.w  #7,D1           CTRL-Codes verarbeiten
008B0A 6B22                  bmi     $8B2E           kleiner 7 ?
008B0C B27C0006              cmp.w   #6,D1           ignorieren, RTS
008B10 6E1C                  bgt     $8B2E           größer 13 ?
008B12 E349                  lsl.w   #1,D1           ignorieren, RTS
008B14 307B100A              move.w  $8B20(PC,D1.w),A0 auf Wortverarbeitung bringen
008B18 D1FC00008B30          add.l   #$8B30,A0       relative Adresse aus Tabelle holen
008B1E 4ED0                  jmp     (A0)            Basisadresse dazu addieren
                                                     und entsprechende Routine ausführen
*****************************************************
008B20 0000                  dc.w    $8B30-$8B30     Sprungtabelle für CTRL-Codes
008B22 01AC                  dc.w    $8CDC-$8B30     7, BEL
008B24 0004                  dc.w    $8B34-$8B30     8, BS
008B26 04AA                  dc.w    $8FDA-$8B30     9, TAB
008B28 04AA                  dc.w    $8FDA-$8B30     10, LF
008B2A 04AA                  dc.w    $8FDA-$8B30     11, VT
008B2C 049E                  dc.w    $8FCE-$8B30     12, FF
                                                     13, CR
```

```
*****************************************************
008B2E 4E75                 rts

*****************************************************   BEL
008B30 6000E218             bra     $6D4A               Ton ausgeben

*****************************************************   TAB
008B34 303900002560         move.w  $2560,D0            Cursorspalte
008B3A 0240FFF8             and.w   #$FFF8,D0           in durch 8 teilbare Zahl wandeln
008B3E 5040                 addq.w  #8,D0               und 8 addieren
008B40 323900002562         move.w  $2562,D1            Cursorzeile
008B46 60000764             bra     $92AC               Cursor neu setzen

*****************************************************   Zeichen nach ESC verarbeiten
008B4A 23FC00008AEE000004A8 move.l  #$8AEE,$4A8         conout-Vektor wieder auf Standard
008B54 927C0041             sub.w   #$41,D1             minus 'A'
008B58 6BD4                 bmi     $8B2E               kleiner, dann ignorieren
008B5A B27C000C             cmp.w   #$C,D1              'M'
008B5E 6F50                 ble     $8BB0               zur ESC-Tabelle für Großbuchstaben
008B60 B27C0018             cmp.w   #$18,D1             'Y' für Cursor setzen ?
008B64 663C                 bne     $8BA2               nein, auf Kleinbuchstaben testen
008B66 23FC00008B72000004A8 move.l  #$8B72,$4A8         conout-Vektor für ESC Y
008B70 4E75                 rts

*****************************************************   Zeile nach ESC Y verarbeiten
008B72 927C0020             sub.w   #$20,D1             Offset abziehen
008B76 33C1000004AC         move.w  D1,$4AC             und Zeilenwert merken
008B7C 23FC00008B88000004A8 move.l  #$8B88,$4A8         conout-Vektor auf Spalte verarbeiten
008B86 4E75                 rts

*****************************************************   Spalte nach ESC Y verarbeiten
008B88 927C0020             sub.w   #$20,D1             Offset abziehen
008B8C 3001                 move.w  D1,D0               Spaltenwert
008B8E 3239000004AC         move.w  $4AC,D1             und Zeile
008B94 23FC00008AEE000004A8 move.l  #$8AEE,$4A8         conout-Vektor wieder auf Standard
008B9E 6000070C             bra     $92AC               und Cursor setzen
```

```
*****************************************************   auf ESC Kleinbuchstabe testen
008BA2 927C0021             sub.w   #$21,D1            Offset abziehen
008BA6 6B86                 bmi     $8B2E              kleiner 'b' ignorieren
008BA8 B27C0015             cmp.w   #$15,D1            'w'
008BAC 6F10                 ble     $8BBE               kleiner oder gleich, Sequenz verarbeiten
008BAE 4E75                 rts

*****************************************************   ESC Großbuchstabe
008BB0 E349                 lsl.w   #1,D1              Code mal 2 für Wortzugriff
008BB2 307B1058             move.w  $8C0C(PC,D1.w),A0  relative Adresse aus Tabelle holen
008BB6 D1FC00008B2E         add.l   #$8B2E,A0          Basisadresse addieren
008BBC 4ED0                 jmp     (A0)               und Routine ausführen

*****************************************************   ESC Kleinbuchstabe
008BBE E349                 lsl.w   #1,D1              Code mal 2 für Wortzugriff
008BC0 307B1064             move.w  $8C26(PC,D1.w),A0  relative Adresse aus Tabelle holen
008BC4 D1FC00008B2E         add.l   #$8B2E,A0          Basisadresse addieren
008BCA 4ED0                 jmp     (A0)               und Routine ausführen

*****************************************************   ESC b  Schriftfarbe setzen
008BCC 23FC00008BD8000004A8 move.l  #$8BD8,$4A8        conout-Vektor setzen
008BD6 4E75                 rts

*****************************************************
008BD8 23FC00008AEE000004A8 move.l  #$8AEE,$4A8        Standard conout-Vektor setzen
008BE2 927C0020             sub.w   #$20,D1            Offset abziehen
008BE6 3001                 move.w  D1,D0
008BE8 60000292             bra     $8E7C              Schriftfarbe setzen

*****************************************************   ESC c  Hintergrundfarbe setzen
008BEC 23FC00008BF8000004A8 move.l  #$8BF8,$4A8        conout-Vektor setzen
008BF6 4E75                 rts
```

```
**************************************************
008BF8 23FC00008AEE000004A8 move.l  #$8AEE,$4A8     Standard conout-Vektor
008C02 927C0020             sub.w   #$20,D1         Offset abziehen
008C06 3001                 move.w  D1,D0
008C08 6000027E             bra     $8E88           Hintergrundfarbe setzen

**************************************************  Adreßtabelle für ESC Großbuchstabe
008C0C 0166                 dc.w    $8B2E-$8B2E     ESC A
008C0E 017A                 dc.w    $8B2E-$8B2E     ESC B
008C10 0194                 dc.w    $8B2E-$8B2E     ESC C
008C12 01AE                 dc.w    $8B2E-$8B2E     ESC D
008C14 0162                 dc.w    $8B2E-$8B2E     ESC E
008C16 0000                 dc.w    $8B2E-$8B2E     ESC F, rts
008C18 0000                 dc.w    $8B2E-$8B2E     ESC G, rts
008C1A 01C2                 dc.w    $8B2E-$8B2E     ESC H
008C1C 0306                 dc.w    $8B2E-$8B2E     ESC I
008C1E 01CA                 dc.w    $8B2E-$8B2E     ESC J
008C20 01F6                 dc.w    $8B2E-$8B2E     ESC K
008C22 0320                 dc.w    $8B2E-$8B2E     ESC L
008C24 033E                 dc.w    $8B2E-$8B2E     ESC M

**************************************************  Adreßtabelle für ESC Kleinbuchstabe
008C26 009E                 dc.w    $8B2E-$8B2E     ESC b
008C28 00BE                 dc.w    $8B2E-$8B2E     ESC c
008C2A 0366                 dc.w    $8B2E-$8B2E     ESC d
008C2C 0382                 dc.w    $8B2E-$8B2E     ESC e
008C2E 03D2                 dc.w    $8B2E-$8B2E     ESC f
008C30 0000                 dc.w    $8B2E-$8B2E     ESC g, rts
008C32 0000                 dc.w    $8B2E-$8B2E     ESC h, rts
008C34 0000                 dc.w    $8B2E-$8B2E     ESC i, rts
008C36 03F2                 dc.w    $8B2E-$8B2E     ESC j
008C38 040E                 dc.w    $8B2E-$8B2E     ESC k
008C3A 0428                 dc.w    $8B2E-$8B2E     ESC l
008C3C 0000                 dc.w    $8B2E-$8B2E     ESC m, rts
008C3E 0000                 dc.w    $8B2E-$8B2E     ESC n, rts
008C40 0446                 dc.w    $8B2E-$8B2E     ESC o
008C42 029A                 dc.w    $8B2E-$8B2E     ESC p
```

```
008C44 02A4              dc.w    $8B2E-$8B2E        ESC q
008C46 0000              dc.w    $8B2E-$8B2E        ESC r, rts
008C48 0000              dc.w    $8B2E-$8B2E        ESC s, rts
008C4A 0000              dc.w    $8B2E-$8B2E        ESC t, rts
008C4C 0000              dc.w    $8B2E-$8B2E        ESC u, rts
008C4E 048C              dc.w    $8B2E-$8B2E        ESC v
008C50 0496              dc.w    $8B2E-$8B2E        ESC w

************************************************    VDI ESC 1, Bildschirmgröße holen
008C52 207900002580      move.l  $2580,A0           Zeiger auf CONTRL Array
008C58 317C00020008      move.w  #2,8(A0)           2 Ergebniswerte
008C5E 20790000258C      move.l  $258C,A0           Zeiger auf INTOUT Array
008C64 303900002550      move.w  $2550,D0           maximale Cursorspalte (79)
008C6A 5240              addq.w  #1,D0              plus 1 gleich Anzahl Spalten
008C6C 31400002          move.w  D0,2(A0)           als zweites Resultat
008C70 303900002552      move.w  $2552,D0           maximale Cursorzeile (24)
008C76 5240              addq.w  #1,D0              plus 1 gleich Anzahl Zeilen
008C78 3080              move.w  D0,(A0)            als erstes Ergebnis
008C7A 4E75              rts

************************************************    VDI ESC 17
008C7C 3F3C0014          move.w  #$14,-(A7)         Hardcopy
008C80 4E4E              trap    #14
008C82 548F              addq.l  #2,A7              Stack wieder in Ordnung bringen
008C84 4E75              rts

************************************************    VDI ESC 3, Enter alpha mode
008C86 6108              bsr     $8C90              ESC E, Clear Home, Bildschirm löschen
008C88 60000226          bra     $8EB0              ESC e, Cursor ein

************************************************    VDI ESC 2, Exit alpha mode
008C8C 61000272          bsr     $8F00              ESC f, Cursor aus
```

```
************************************************      ESC E, Clear Home
008C90 615E                bsr     $8CF0              ESC H, Cursor Home
008C92 6064                bra     $8CF8              ESC J, Rest des Bildschirms löschen

************************************************      ESC A, Cursor hoch, VDI ESC 4
008C94 323900002562        move.w  $2562,D1           Cursorzeile
008C9A 67DE                beq     $8C7A              null, dann schon fertig
008C9C 5341                subq.w  #1,D1              eins abziehen
008C9E 303900002560        move.w  $2560,D0           Cursorspalte holen
008CA4 60000606            bra     $92AC              und Cursor setzen

************************************************      ESC B, Cursor down, VDI ESC 5
008CA8 323900002562        move.w  $2562,D1           Cursorzeile
008CAE B27900002552        cmp.w   $2552,D1           mit maximaler Cursorzeile vergleichen
008CB4 67C4                beq     $8C7A              schon in unterster Zeile, fertig
008CB6 5241                addq.w  #1,D1              um eins erhöhen
008CB8 303900002560        move.w  $2560,D0           Cursorspalte
008CBE 600005EC            bra     $92AC              Cursor setzen

************************************************      ESC C, Cursor rechts, VDI ESC 6
008CC2 303900002560        move.w  $2560,D0           Cursorspalte
008CC8 B07900002550        cmp.w   $2550,D0           mit Maximalwert (79) vergleichen
008CCE 67AA                beq     $8C7A
008CD0 5240                addq.w  #1,D0              um eins erhöhen
008CD2 323900002562        move.w  $2562,D1           Cursorzeile holen
008CD8 600005D2            bra     $92AC              und Cursor setzen

************************************************      ESC D, DEL, Cursor links, VDI ESC 7
008CDC 303900002560        move.w  $2560,D0           Cursorspalte
008CE2 6796                beq     $8C7A              schon Null ?
008CE4 5340                subq.w  #1,D0              um eins vermindern
008CE6 323900002562        move.w  $2562,D1           Cursorzeile
008CEC 600005BE            bra     $92AC              Cursor setzen
```

```
************************************************      ESC H, Cursor Home, VDI ESC 8
008CF0 7000                 moveq.l #0,D0             Spalte
008CF2 3200                 move.w  D0,D1             und Zeile auf Null
008CF4 600005B6             bra     $92AC             Cursor setzen

************************************************      ESC J, Rest des Bildschirms löschen, VDI 9
008CF8 612A                 bsr     $8D24             ESC K, Rest der Zeile löschen
008CFA 323900002562         move.w  $2562,D1          Cursorzeile
008D00 B27900002552         cmp.w   $2552,D1          mit maximaler Cursorzeile vergleichen
008D06 6700FF72             beq     $8C7A
008D0A 5241                 addq.w  #1,D1
008D0C 4841                 swap    D1
008D0E 323C0000             move.w  #0,D1
008D12 343900002552         move.w  $2552,D2          maximale Cursorzeile
008D18 4842                 swap    D2
008D1A 343900002550         move.w  $2550,D2          maximale Cursorspalte (79)
008D20 60000436             bra     $9158

************************************************      ESC K, Rest der Zeile löschen, VDI ESC 10
008D24 08B9000300002576     bclr    #3,$2576          Flag für Zeilenüberlauf löschen
008D2C 40E7                 move.w  SR, -(A7)         alten Wert merken
008D2E 610001D0             bsr     $8F00             ESC f, Cursor aus
008D32 610001EC             bsr     $8F20             ESC j, Cursorposition speichern
008D36 323900002560         move.w  $2560,D1          Cursorspalte
008D3C 08010000             btst    #0,D1
008D40 6716                 beq     $8D58
008D42 B27900002550         cmp.w   $2550,D1          mit Maximalwert (79) vergleichen
008D48 673A                 beq     $8D84             in letzter Spalte, dann Blank ausgeben
008D4A 323C0020             move.w  #$20,D1           Blank
008D4E 6100035E             bsr     $90AE             ausgeben
008D52 323900002560         move.w  $2560,D1          Cursorspalte
008D58 4841                 swap    D1
008D5A 323900002562         move.w  $2562,D1          Cursorzeile
008D60 3401                 move.w  D1,D2
008D62 4841                 swap    D1
008D64 4842                 swap    D2
008D66 343900002550         move.w  $2550,D2          maximale Cursorspalte
```

```
008D6C 610003EA          bsr     $9158
008D70 44DF              move.w  (A7)+,CCR       Flag zurückholen
008D72 6708              beq     $8D7C           nicht gesetzt gewesen ?
008D74 08F9000300002576  bset    #3,$2576        neu setzen
008D7C 610001BE          bsr     $8F3C           ESC k, Cursorposition wiederholen
008D80 6000016C          bra     $8EEE           Cursor wieder einschalten

008D84 323C0020          move.w  #$20,D1         Blank
008D88 61000324          bsr     $90AE           ausgeben
008D8C 60E2              bra     $8D70

************************************************ VDI ESC 11, Cursor setzen
008D8E 207900002584      move.l  $2584,A0        Zeiger auf INTIN Array
008D94 3210              move.w  (A0),D1         Zeile holen
008D96 5341              subq.w  #1,D1           minus Offset
008D98 30280002          move.w  2(A0),D0        Spalte holen
008D9C 5340              subq.w  #1,D0           minus Offset
008D9E 6000050C          bra     $92AC           Cursor setzen

************************************************ VDI ESC 12, Textausgabe
008DA2 207900002580      move.l  $2580,A0        Zeiger auf CONTRL Array
008DA8 30280006          move.w  6(A0),D0        Anzahl der Zeichen
008DAC 207900002584      move.l  $2584,A0        Zeiger auf INTIN Array
008DB2 600E              bra     $8DC2

008DB4 3218              move.w  (A0)+,D1        Zeichen nach D1 holen
008DB6 48E78080          movem.l D0/A0,-(A7)     Register retten
008DBA 6100FD26          bsr     $8AE2           Zeichen in D1 ausgeben
008DBE 4CDF0101          movem.l (A7)+,D0/A0     Register zurück
008DC2 51C8FFF0          dbra    D0,$8DB4        nächstes Zeichen ausgeben
008DC6 4E75              rts

********************************************** ESC p, reverse ein, VDI ESC 13
008DC8 08F9000400002576  bset    #4,$2576        Flag für reverse setzen
008DD0 4E75              rts
```

```
************************************************
008DD2 08B9000400002576     bclr    #4,$2576
008DDA 4E75                 rts

************************************************
008DDC 207900002580         move.l  $2580,A0
008DE2 317C00020008         move.w  #2,8(A0)
008DE8 20790000258C         move.l  $258C,A0
008DEE 303900002562         move.w  $2562,D0
008DF4 5240                 addq.w  #1,D0
008DF6 3080                 move.w  D0,(A0)
008DF8 303900002560         move.w  $2560,D0
008DFE 5240                 addq.w  #1,D0
008E00 31400002             move.w  D0,2(A0)
008E04 4E75                 rts

************************************************
008E06 207900002580         move.l  $2580,A0
008E0C 317C00010008         move.w  #1,8(A0)
008E12 20790000258C         move.l  $258C,A0
008E18 30BC0001             move.w  #1,(A0)
008E1C 4E75                 rts

************************************************
008E1E 207900002584         move.l  $2584,A0
008E24 30BC0000             move.w  #0,(A0)
008E28 4EF90000FF50         jmp     $FF50

************************************************
008E2E 4EF90000FF78         jmp     $FF78

************************************************
008E34 323900002562         move.w  $2562,D1
008E3A 6600FE60             bne     $8C9C
008E3E 3F3900002560         move.w  $2560,-(A7)
008E44 6108                 bsr     $8E4E
008E46 301F                 move.w  (A7)+,D0
```

```
ESC q, reverse aus, VDI ESC 14
Flag für reverse löschen

VDI ESC 15, Cursorposition holen
Zeiger auf CONTRL Array
2 Ergebniswerte
Zeiger auf INTOUT Array
Cursorzeile
plus Offset
als erstes Ergebnis
Cursorspalte
plus Offset
als zweites Ergebnis

VDI ESC 16, Inquire tablet status
Zeiger auf CONTRL Array
ein Ergebniswert
Zeiger auf INTOUT Array
Tablet verfügbar

VDI ESC 18, Graphic Cursor setzen
Zeiger auf INTIN Array
keine Ergebniswerte
Graphic Cursor setzen

VDI ESC 19, Graphic Cursor löschen

ESC I, Cursor hoch, bei Bedarf scrollen
Cursorzeile
nicht in Zeile 0, dann Cursor hoch
Cursorspalte retten
ESC L, Zeile einfügen
Cursorspalte wieder zurück
```

```
008E48 7200              moveq.l #0,D1          Cursorzeile auf Null
008E4A 60000460          bra     $92AC          Cursor setzen

************************************************ ESC L, Zeile einfügen
008E4E 610000B0          bsr     $8F00          ESC f, Cursor aus
008E52 323900002562      move.w  $2562,D1       Cursorzeile
008E58 6100056E          bsr     $93C8          Rest des Bildschirms nach unten schieben
008E5C 4240              clr.w   D0             Cursor in Spalte 0
008E5E 323900002562      move.w  $2562,D1       Cursorzeile
008E64 61000446          bsr     $92AC          Cursor setzen
008E68 60000084          bra     $8EEE          Cursor wieder einschalten

************************************************ ESC M, Zeile löschen
008E6C 61000092          bsr     $8F00          ESC f, Cursor aus
008E70 323900002562      move.w  $2562,D1       Cursorzeile
008E76 61000508          bsr     $9380          Rest des Bildschirms nach oben schieben
008E7A 60E0              bra     $8E5C          s.o.

************************************************ Schriftfarbe setzen
008E7C C07C000F          and.w   #$F,D0         mod 16, 0..15
008E80 33C000002558      move.w  D0,$2558       Schriftfarbe merken
008E86 4E75              rts

************************************************ Hintergrundfarbe setzen
008E88 C07C000F          and.w   #$F,D0         mod 16, 0..15
008E8C 33C000002556      move.w  D0,$2556       Hintergrundfarbe merken
008E92 4E75              rts

************************************************ ESC d, Bildschirm bis Cursor löschen
008E94 610000DE          bsr     $8F74          ESC o, Zeile bis Cursor löschen
008E98 343900002562      move.w  $2562,D2       Cursorzeile
008E9E 67F2              beq     $8E92          schon Null, fertig
008EA0 5342              subq.w  #1,D2
008EA2 4842              swap    D2
008EA4 343900002550      move.w  $2550,D2       maximale Cursorspalte
008EAA 7200              moveq.l #0,D1
008EAC 600002AA          bra     $9158
```

```
************************************************      ESC e, Cursor ein
008EB0 4A7900002422          tst.w   $2422           Cursor schon ein ?
008EB6 67DA                  beq     $8E92           ja, fertig
008EB8 427900002422          clr.w   $2422           Flag für Cursor ein
008EBE 41F900002576          lea     $2576,A0
008EC4 08100000              btst    #0,(A0)
008EC8 6618                  bne     $8EE2
008ECA 08D00002              bset    #2,(A0)
008ECE 303900002560          move.w  $2560,D0        Cursorspalte
008ED4 323900002562          move.w  $2562,D1        Cursorzeile
008EDA 6100031C              bsr     $91F8           Cursorposition berechnen
008EDE 6000043E              bra     $931E           Zeichen an Cursorposition invertieren

008EE2 61EA                  bsr     $8ECE           Zeichen an Cursorposition invertieren
008EE4 08D00001              bset    #1,(A0)
008EE8 08D00002              bset    #2,(A0)
008EEC 4E75                  rts

************************************************
008EEE 4A7900002422          tst.w   $2422           Cursor eingeschaltet ?
008EF4 679C                  beq     $8E92           ja, rts
008EF6 537900002422          subq.w  #1,$2422
008EFC 67C0                  beq     $8EBE           s.o.
008EFE 4E75                  rts

************************************************      ESC f, Cursor aus
008F00 527900002422          addq.w  #1,$2422        Flag für Cursor aus
008F06 41F900002576          lea     $2576,A0
008F0C 08900002              bclr    #2,(A0)         Flag für Cursor löschen
008F10 6780                  beq     $8E92           Cursor war schon aus, rts
008F12 08100000              btst    #0,(A0)
008F16 67B6                  beq     $8ECE
008F18 08900001              bclr    #1,(A0)
008F1C 66B0                  bne     $8ECE
008F1E 4E75                  rts
```

```
************************************************      ESC j, Cursorposition speichern
008F20 08F9000500002576      bset    #5,$2576         Flag für Cursor gespeichert
008F28 41F90000242E          lea     $242E,A0         Adresse des Zwischenspeichers
008F2E 30F900002560          move.w  $2560,(A0)+      Cursorspalte
008F34 30B900002562          move.w  $2562,(A0)       Cursorzeile
008F3A 4E75                  rts

************************************************      ESC k, Cursor auf gespeicherte Position
008F3C 08B9000500002576      bclr    #5,$2576         war Cursorposition gespeichert ?
008F44 6700FDAA              beq     $8CF0            nein
008F48 41F90000242E          lea     $242E,A0         Adresse des Zwischenspeichers
008F4E 3018                  move.w  (A0)+,D0         Cursorspalte
008F50 3210                  move.w  (A0),D1          Cursorzeile
008F52 60000358              bra     $92AC            Cursor setzen

************************************************      ESC l, Zeile löschen
008F56 61A8                  bsr     $8F00            ESC f, Cursor ausschalten
008F58 323900002562          move.w  $2562,D1         Cursorzeile
008F5E 3401                  move.w  D1,D2
008F60 4841                  swap    D1
008F62 4241                  clr.w   D1
008F64 4842                  swap    D2
008F66 343900002550          move.w  $2550,D2         maximale Cursorspalte
008F6C 610001EA              bsr     $9158
008F70 6000FEEA              bra     $8E5C            Cursor in Spalte Null

************************************************      ESC o, Zeile bis Cursor löschen
008F74 618A                  bsr     $8F00            ESC f, Cursor ausschalten
008F76 61A8                  bsr     $8F20            ESC j, Cursorposition speichern
008F78 343900002560          move.w  $2560,D2         Cursorspalte
008F7E 6730                  beq     $8FB0            Null, schon fertig
008F80 08020000              btst    #0,D2
008F84 6610                  bne     $8F96
008F86 323C0020              move.w  #$20,D1          Blank
008F8A 61000122              bsr     $90AE            ausgeben
008F8E 343900002560          move.w  $2560,D2         Cursorspalte
008F94 5542                  subq.w  #2,D2
```

```
008F96 4842              swap    D2
008F98 343900002562      move.w  $2562,D2        Cursorzeile
008F9E 3202              move.w  D2,D1
008FA0 4842              swap    D2
008FA2 4841              swap    D1
008FA4 4241              clr.w   D1
008FA6 610001B0          bsr     $9158
008FAA 6190              bsr     $8F3C           ESC k, Cursorposition zurückholen
008FAC 6000FF40          bra     $8EEE           und Cursor wieder einschalten

008FB0 323C0020          move.w  #$20,D1         Blank
008FB4 610000F8          bsr     $90AE           ausgeben
008FB8 60F0              bra     $8FAA

************************************************* ESC v, neue Zeile am Zeilenende
008FBA 08F9000300002576  bset    #3,$2576        Flag setzen
008FC2 4E75              rts

************************************************* ESC w, keine neuen Zeile am Zeilenende
008FC4 08B9000300002576  bclr    #3,$2576        Flag löschen
008FCC 4E75              rts

************************************************* CR, Cursor in Spalte Null
008FCE 323900002562      move.w  $2562,D1        Cursorzeile
008FD4 4240              clr.w   D0              Spalte auf Null
008FD6 600002D4          bra     $92AC           Cursor setzen

************************************************* LF, (VT, FF), Cursor nach unten
008FDA 303900002562      move.w  $2562,D0        Cursorzeile
008FE0 B07900002552      cmp.w   $2552,D0        mit maximaler Cursorzeile vergleichen
008FE6 6600FCC0          bne     $8CA8           nicht in unterster Zeile, nur Cursor down
008FEA 6100FF14          bsr     $8F00           ESC f, Cursor ausschalten
008FEE 4241              clr.w   D1
008FF0 6100038E          bsr     $9380           Bildschirm nach unten scrollen
008FF4 6000FEF8          bra     $8EEE           und Cursor wieder einschalten
```

```
************************************************      Cursor blinken
008FF8 41F900002576         lea     $2576,A0        Adresse des Flagworts
008FFE 08100002             btst    #2,(A0)         Cursor eingeschaltet ?
009002 671E                 beq     $9022           nein
009004 08100000             btst    #0,(A0)         Cursor blinkend ?
009008 6718                 beq     $9022           nein
00900A 43F900002565         lea     $2565,A1        Adresse des Blinkzählers
009010 5311                 subq.b  #1,(A1)         Zähler vermindern
009012 660E                 bne     $9022           noch nicht abgelaufen ?
009014 12B900002564         move.b  $2564,(A1)      Blinkzähler wieder neu laden
00901A 08500001             bchg    #1,(A0)         Cursorphase invertieren
00901E 6000FEAE             bra     $8ECE           Cursor einschalten

009022 4E75                 rts

************************************************      Cursor Konfiguration, XBIOS Nr. 21
009024 302F0004             move.w  4(A7),D0        Nummer von Stack holen
009028 6BF8                 bmi     $9022           negativ, ignorieren
00902A B07C0005             cmp.w   #5,D0           größer 5 ?
00902E 6EF2                 bgt     $9022           ja, ignorieren
009030 E340                 asl.w   #1,D0

009032 41F90000904A         lea     $904A,A0        Basisadresse der Tabelle
009038 D0FB0004             add.w   $903E(PC,D0.w),A0 plus relative Adresse
00903C 4ED0                 jmp     (A0)            Funktion ausführen

************************************************      Adresse der Routinen
00903E 0000                 dc.w    $904A-$904A
009040 0004                 dc.w    $904E-$904A
009042 0008                 dc.w    $9052-$904A
009044 0016                 dc.w    $9060-$904A
009046 0024                 dc.w    $906E-$904A
009048 002C                 dc.w    $9076-$904A
```

```
*************************************************
00904A 6000FEB4          bra     $8F00

*************************************************
00904E 6000FE60          bra     $8EB0

*************************************************
009052 6100FEAC          bsr     $8F00
009056 08ED00002576      bset    #0,$2576(A5)
00905C 6000FE90          bra     $8EEE

*************************************************
009060 6100FE9E          bsr     $8F00
009064 08AD00002576      bclr    #0,$2576(A5)
00906A 6000FE82          bra     $8EEE

*************************************************
00906E 1B6F00072564      move.b  7(A7),$2564(A5)
009074 4E75              rts

*************************************************
009076 7000              moveq.l #0,D0
009078 102D2564          move.b  $2564(A5),D0
00907C 4E75              rts

*************************************************
00907E 36390000256C      move.w  $256C,D3
009084 B243              cmp.w   D3,D1
009086 6522              bcs     $90AA
009088 B2790000256A      cmp.w   $256A,D1
00908E 621A              bhi     $90AA
009090 207900002572      move.l  $2572,A0
009096 D241              add.w   D1,D1
009098 32301000          move.w  (A0,D1.w),D1
00909C E649              lsr.w   #3,D1
00909E 207900002566      move.l  $2566,A0
0090A4 D0C1              add.w   D1,A0
```

```
0
ESC f, Cursor ausschalten

1
ESC e, Cursor einschalten

2
ESC f, Cursor ausschalten

Cursor wieder einschalten

3
ESC f, Cursor ausschalten

Cursor wieder einschalten

4
Cursor Blinkrate setzen

5

Cursor Blinkrate laden

Fontdaten für Zeichen in D1 berechnen
kleinster ASCII-Code im Font
mit auszugebendem Zeichen vergleichen
Zeichen nicht im Font
größter ASCII-Code im Font
Zeichen nicht im Font
Font Offsetpointer laden
Code mal zwei
ergibt Bitnummer im Font
durch 8 ergibt Bytenummer
Pointer auf Fontdaten
ergibt Zeiger auf Daten dieses Zeichens
```

```
0090A6 4243                clr.w   D3              Flag für Zeichen im Font
0090A8 4E75                rts
0090AA 7601                moveq.l #1,D3           Zeichen nicht im Font vorhanden
0090AC 4E75                rts

************************************************   ascout, Steuercodes ignorieren
0090AE 61CE                bsr     $907E           Zeichen im Font vorhanden ?
0090B0 6702                beq     $90B4           ja
0090B2 4E75                rts

0090B4 22790000255A        move.l  $255A,A1        Bildschirmadresse des Zeichens
0090BA 3E3900002556        move.w  $2556,D7        Hintergrundfarbe
0090C0 4847                swap    D7              in oberes Wort
0090C2 3E3900002558        move.w  $2558,D7        Schriftfarbe in unteres Wort
0090C8 0839000400002576    btst    #4,$2576        reverse eingeschaltet ?
0090D0 6702                beq     $90D4           nein
0090D2 4847                swap    D7              Schrift- und Hintergrundfarbe vertauschen
0090D4 08B9000200002576    bclr    #2,$2576
0090DC 40E7                move.w  SR, -(A7)       Status merken
0090DE 61000160            bsr     $9240           neue Position berechnen
0090E2 22790000255A        move.l  $255A,A1        Bildschirmadresse
0090E8 303900002560        move.w  $2560,D0        Cursorspalte
0090EE 323900002562        move.w  $2562,D1        Cursorzeile
0090F4 61000252            bsr     $9348           relative Bildschirmadresse berechnen
0090F8 6732                beq     $912C
0090FA 303900002554        move.w  $2554,D0        Anzahl Bytes pro Characterzeile
009100 C0C1                mulu.w  D1,D0
009102 22790000044E        move.l  $44E,A1         Adresse des Bildschirm-RAMs
009108 D3C0                add.l   D0,A1
00910A 4240                clr.w   D0
00910C B27900002552        cmp.w   $2552,D1        mit maximaler Cursorzeile vergleichen
009112 640A                bcc     $911E
009114 D2F900002554        add.w   $2554,A1        plus Anzahl Bytes pro Characterzeile
00911A 5241                addq.w  #1,D1
00911C 600E                bra     $912C
```

```
00911E 48E7C040          movem.l D0-D1/A1,-(A7)      Register retten
009122 7200              moveq.l #0,D1
009124 6100025A          bsr     $9380
009128 4CDF0203          movem.l (A7)+,D0-D1/A1      Register zurückholen
00912C 23C90000255A      move.l  A1,$255A            Bildschirmadresse
009132 33C000002560      move.w  D0,$2560            Cursorspalte
009138 33C100002562      move.w  D1,$2562            Cursorzeile
00913E 44DF              move.w  (A7)+,CCR           Status zurückholen
009140 6714              beq     $9156
009142 610001DA          bsr     $931E
009146 08F9000100002576  bset    #1,$2576
00914E 08F9000200002576  bset    #2,$2576
009156 4E75              rts

009158 9481              sub.l   D1,D2
00915A 3001              move.w  D1,D0
00915C 4841              swap    D1
00915E 61000098          bsr     $91F8               Cursorposition berechnen
009162 E242              asr.w   #1,D2
009164 36390000257C      move.w  $257C,D3            Anzahl der Bildebenen (1,2 oder 4)
00916A 0C430004          cmp.w   #4,D3
00916E 6602              bne     $9172
009170 5343              subq.w  #1,D3
009172 3202              move.w  D2,D1
009174 5241              addq.w  #1,D1
009176 E761              asl.w   D3,D1
009178 34790000257E      move.w  $257E,A2            Bytes pro Bildschirmzeile
00917E 94C1              sub.w   D1,A2
009180 3202              move.w  D2,D1
009182 4842              swap    D2
009184 5242              addq.w  #1,D2
009186 C4F90000254E      mulu.w  $254E,D2            mal Höhe eines Zeichens
00918C 5342              subq.w  #1,D2
00918E 4280              clr.l   D0
009190 3A3900002556      move.w  $2556,D5            Hintergrundfarbe
009196 0C7900020000257C  cmp.w   #2,$257C            Anzahl der Bildebenen
00919E 6B44              bmi     $91E4
```

```
0091A0 6728             beq     $91CA
0091A2 E245             asr.w   #1,D5
0091A4 4040             negx.w  D0
0091A6 4840             swap    D0
0091A8 E245             asr.w   #1,D5
0091AA 4040             negx.w  D0
0091AC 4283             clr.l   D3
0091AE E245             asr.w   #1,D5
0091B0 4043             negx.w  D3
0091B2 4843             swap    D3
0091B4 E245             asr.w   #1,D5
0091B6 4043             negx.w  D3
0091B8 3A01             move.w  D1,D5
0091BA 22C0             move.l  D0,(A1)+
0091BC 22C3             move.l  D3,(A1)+
0091BE 51CDFFFA         dbra    D5,$91BA
0091C2 D3CA             add.l   A2,A1
0091C4 51CAFFF2         dbra    D2,$91B8
0091C8 4E75             rts

0091CA E245             asr.w   #1,D5
0091CC 4040             negx.w  D0
0091CE 4840             swap    D0
0091D0 E245             asr.w   #1,D5
0091D2 4040             negx.w  D0
0091D4 3A01             move.w  D1,D5
0091D6 22C0             move.l  D0,(A1)+
0091D8 51CDFFFC         dbra    D5,$91D6
0091DC D3CA             add.l   A2,A1
0091DE 51CAFFF4         dbra    D2,$91D4
0091E2 4E75             rts

0091E4 E245             asr.w   #1,D5
0091E6 4040             negx.w  D0
0091E8 3A01             move.w  D1,D5
0091EA 32C0             move.w  D0,(A1)+
0091EC 51CDFFFC         dbra    D5,$91EA
```

```
0091F0 D3CA              add.l   A2,A1
0091F2 51CAFFF4          dbra    D2,$91E8
0091F6 4E75              rts

************************************************  Cursorposition berechnen (D0/D1)
0091F8 363900002550      move.w  $2550,D3        maximale Cursorspalte
0091FE B640              cmp.w   D0,D3
009200 6A02              bpl     $9204           Spaltenwert kleiner ?
009202 3003              move.w  D3,D0           sonst Maximalwert benutzen
009204 363900002552      move.w  $2552,D3        maximale Cursorzeile
00920A B641              cmp.w   D1,D3
00920C 6A02              bpl     $9210           Zeilenwert kleiner ?
00920E 3203              move.w  D3,D1           sonst Maximalwert benutzen
009210 36390000257C      move.w  $257C,D3        Anzahl der Bildebenen
009216 3A00              move.w  D0,D5           Spalte nach D5
009218 08850000          bclr    #0,D5
00921C C6C5              mulu.w  D5,D3           mal Maximalwert
00921E 08000000          btst    #0,D0
009222 6702              beq     $9226
009224 5283              addq.l  #1,D3
009226 3A3900002554      move.w  $2554,D5        Anzahl Bytes pro Characterzeile
00922C CAC1              mulu.w  D1,D5           mal Zeilenwert
00922E 22790000044E      move.l  $44E,A1         Basisadresse des Bildschirm-RAMs
009234 D3C5              add.l   D5,A1           plus Offset für Zeile
009236 D3C3              add.l   D3,A1           plus Offset für Spalte
009238 D2F90000255E      add.w   $255E,A1        plus Anzahl Bytes pro Rasterzeile
00923E 4E75              rts

009240 34790000256E      move.w  $256E,A2        formwidth
009246 36790000257E      move.w  $257E,A3        Bytes pro Bildschirmzeile
00924C 38390000254E      move.w  $254E,D4        Höhe eines Zeichens
009252 5344              subq.w  #1,D4
009254 3C390000257C      move.w  $257C,D6        Anzahl der Bildebenen
00925A 5346              subq.w  #1,D6
00925C 3A04              move.w  D4,D5
00925E 2848              move.l  A0,A4
009260 2A49              move.l  A1,A5
```

```
009262 E287             asr.l   #1,D7
009264 0807000F         btst    #15,D7
009268 6706             beq     $9270
00926A 642A             bcc     $9296
00926C 76FF             moveq.l #-1,D3
00926E 6004             bra     $9274
009270 6512             bcs     $9284
009272 7600             moveq.l #0,D3
009274 1A83             move.b  D3,(A5)
009276 DACB             add.w   A3,A5
009278 51CDFFFA         dbra    D5,$9274
00927C 5449             addq.w  #2,A1
00927E 51CEFFDC         dbra    D6,$925C
009282 4E75             rts

009284 1A94             move.b  (A4),(A5)
009286 DACB             add.w   A3,A5
009288 D8CA             add.w   A2,A4
00928A 51CDFFF8         dbra    D5,$9284
00928E 5449             addq.w  #2,A1
009290 51CEFFCA         dbra    D6,$925C
009294 4E75             rts

009296 1614             move.b  (A4),D3
009298 4603             not.b   D3
00929A 1A83             move.b  D3,(A5)
00929C DACB             add.w   A3,A5
00929E D8CA             add.w   A2,A4
0092A0 51CDFFF4         dbra    D5,$9296
0092A4 5449             addq.w  #2,A1
0092A6 51CEFFB4         dbra    D6,$925C
0092AA 4E75             rts
```

```
************************************************
0092AC B07900002550          cmp.w   $2550,D0        Cursor setzen
0092B2 6306                  bls     $92BA           Spalte mit Maximalwert vergleichen
0092B4 303900002550          move.w  $2550,D0        kleiner ?
0092BA B27900002552          cmp.w   $2552,D1        sonst Maximalwert verwenden
0092C0 6306                  bls     $92C8           Zeile mit Maximalwert vergleichen
0092C2 323900002552          move.w  $2552,D1        kleiner ?
0092C8 33C000002560          move.w  D0,$2560        sonst Maximalwert verwenden
0092CE 33C100002562          move.w  D1,$2562        Cursorspalte abspeichern
0092D4 41F900002576          lea     $2576,A0        Cursorzeile abspeichern
0092DA 08100002              btst    #2,(A0)         Adresse des Cursorflags
0092DE 6732                  beq     $9312
0092E0 08100000              btst    #0,(A0)         Zeichen an alter Position invertiert ?
0092E4 670A                  beq     $92F0           ja
0092E6 08900002              bclr    #2,(A0)
0092EA 08100001              btst    #1,(A0)
0092EE 671E                  beq     $930E
0092F0 22790000255A          move.l  $255A,A1        Bildschirmadresse des alten Cursors
0092F6 6126                  bsr     $931E           Zeichen an Cursorposition invertieren
0092F8 6100FEFE              bsr     $91F8           neue Cursoradresse berechnen
0092FC 23C90000255A          move.l  A1,$255A        Bildschirmadresse des Cursors
009302 611A                  bsr     $931E           Zeichen an Cursorposition invertieren
009304 08F9000200002576      bset    #2,$2576        Flag für Zeichen ist invertiert
00930C 4E75                  rts

00930E 08D00002              bset    #2,(A0)
009312 6100FEE4              bsr     $91F8           neue Cursorposition berechnen
009316 23C90000255A          move.l  A1,$255A        Bildschirmadresse der Cursorposition
00931C 4E75                  rts

************************************************   Zeichen an Cursorposition invertieren
00931E 34790000257E          move.w  $257E,A2        Bytes pro Bildschirmzeile
009324 38390000254E          move.w  $254E,D4        Höhe eines Zeichen
00932A 5344                  subq.w  #1,D4           als dbra Zähler
00932C 3C390000257C          move.w  $257C,D6        Anzahl der Bildebenen
009332 5346                  subq.w  #1,D6           als dbra Zähler
009334 3A04                  move.w  D4,D5           Zähler für Rasterzeilen
```

```
009336 2849                move.l  A1,A4           Bildschirmadresse des Zeichens
009338 4614                not.b   (A4)            eine Rasterzeile des Zeichens invertieren
00933A D8CA                add.w   A2,A4           Zeiger auf nächste Rasterzeile
00933C 51CDFFFA            dbra    D5,$9338        nächste Rasterzeile
009340 5449                addq.w  #2,A1
009342 51CEFFF0            dbra    D6,$9334        nächste Bildebene
009346 4E75                rts

************************************************
009348 B07900002550        cmp.w   $2550,D0        maximale Cursorspalte
00934E 6612                bne     $9362
009350 0839000300002576    btst    #3,$2576        Flag für Zeilenüberlauf gesetzt ?
009358 6604                bne     $935E           ja
00935A 4243                clr.w   D3
00935C 4E75                rts

00935E 7601                moveq.l #1,D3
009360 4E75                rts

009362 5240                addq.w  #1,D0
009364 08000000            btst    #0,D0
009368 6706                beq     $9370
00936A 5249                addq.w  #1,A1
00936C 4243                clr.w   D3
00936E 4E75                rts

009370 363900002570        move.w  $257C,D3        Anzahl der Bildebenen
009376 E343                asl.w   #1,D3
009378 5343                subq.w  #1,D3
00937A D2C3                add.w   D3,A1
00937C 4243                clr.w   D3
00937E 4E75                rts
```

```
************************************************
009380 26790000044E         move.l  $44E,A3          Bildschirm ab Zeile D1 nach unten scrollen
009386 363900002554         move.w  $2554,D3         Adresse des Bildschirm-RAMs
00938C C6C1                 mulu.w  D1,D3            Anzahl Bytes pro Characterzeile
00938E 47F33000             lea     0(A3,D3.w),A3    mit Anzahl Zeilen multiplizieren
009392 4441                 neg.w   D1               ergibt Adresse der Zeile
009394 D27900002552         add.w   $2552,D1         aktuelle Zeile
00939A 363900002554         move.w  $2554,D3         maximale Cursorzeile
0093A0 45F33000             lea     0(A3,D3.w),A2    Anzahl Bytes pro Characterzeile
0093A4 C6C1                 mulu.w  D1,D3            ergibt Adresse der Zeile 1
0093A6 E443                 asr.w   #2,D3            Anzahl Bytes zu verschieben
0093A8 6002                 bra     $93AC            Anzahl Langworte
0093AA 26DA                 move.l  (A2)+,(A3)+      
0093AC 51CBFFFC             dbra    D3,$93AA         Bildschirmzeilen kopieren
0093B0 323900002552         move.w  $2552,D1         
0093B6 3401                 move.w  D1,D2            maximale Cursorzeile
0093B8 4841                 swap    D1               
0093BA 4842                 swap    D2               
0093BC 4241                 clr.w   D1               
0093BE 343900002550         move.w  $2550,D2         
0093C4 6000FD92             bra     $9158            maximale Cursorspalte

************************************************
0093C8 26790000044E         move.l  $44E,A3          Bildschirm ab Zeile D1 nach oben scrollen
0093CE 363900002552         move.w  $2552,D3         Adresse des Bildschirm-RAMs
0093D4 C6F900002554         mulu.w  $2554,D3         maximale Cursorzeile
0093DA 47F33000             lea     (A3,D3.w),A3     mit Anzahl Bytes pro Characterzeile mult.
0093DE 363900002554         move.w  $2554,D3         ergibt Adresse der letzten Characterzeile
0093E4 45F33000             lea     (A3,D3.w),A2     Anzahl Bytes pro Zeile
0093E8 3001                 move.w  D1,D0            ergibt Adresse der Zeile 1
0093EA 4440                 neg.w   D0               aktuelle Zeile
0093EC D07900002552         add.w   $2552,D0         
0093F2 C6C0                 mulu.w  D0,D3            maximale Cursorzeile addieren
0093F4 E443                 asr.w   #2,D3            
0093F6 6002                 bra     $93FA            durch 4 für Langwort-Transfer
0093F8 2523                 move.l  -(A3),-(A2)      
0093FA 51CBFFFC             dbra    D3,$93F8         Bildschirmzeilen kopieren
```

```
0093FE 60B6                bra     $93B6

************************************************      VDI ESC 102, Fontparameter initialisieren
009400 207900002584        move.l  $2584,A0          Zeiger auf INTIN Array
009406 2050                move.l  (A0),A0           Adresse des Font-Headers
009408 30280052            move.w  82(A0),D0         formhight, Höhe eines Zeichens
00940C 33C00000254E        move.w  D0,$254E          merken
009412 32390000257E        move.w  $257E,D1          Bytes pro Bildschirmzeile
009418 C2C0                mulu.w  D0,D1             mal Höhe eines Zeichen
00941A 33C100002554        move.w  D1,$2554          ergibt Bytes pro Characterzeile
009420 7200                moveq.l #0,D1
009422 323900002578        move.w  $2578,D1          Anzahl Rasterzeilen auf dem Bildschirm
009428 82C0                divu.w  D0,D1             durch Fonthöhe dividieren
00942A 5341                subq.w  #1,D1             minus
00942C 33C100002552        move.w  D1,$2552          ergibt maximale Cursorzeile
009432 7200                moveq.l #0,D1
009434 323900002570        move.w  $2570,D1          Bildschirmbreite in Bit
00943A 82E80034            divu.w  52(A0),D1         durch maximale Zeichenbreite dividieren
00943E 5341                subq.w  #1,D1             minus 1
009440 33C100002550        move.w  D1,$2550          ergibt maximale Cursorspalte
009446 33E800500000256E    move.w  80(A0),$256E      formwidht, Breite des Fonts
00944E 33E800240000256C    move.w  36(A0),$256C      kleinster ASCII-Code im Font
009456 33E800260000256A    move.w  38(A0),$256A      größter ASCII-Code im Font
00945E 23E8004C00002566    move.l  76(A0),$2566      Zeiger auf Fontdaten
009466 23E8004800002572    move.l  72(A0),$2572      Zeiger auf Offsettabelle
00946E 4E75                rts
```

```
************************************************
00F6C4 10390000044C          move.b  $44C,D0      Bildschirmausgabe initialisieren
00F6CA C07C0003              and.w   #3,D0        sshiftmd, Bildschirmauflösung
00F6CE B07C0003              cmp.w   #3,D0        Bit 0 und 1 isolieren
00F6D2 6604                  bne     $F6D8        3 ?
00F6D4 303C0002              move.w  #2,D0        nein
00F6D8 3F00                  move.w  D0,-(A7)     durch 2 (high resolution) ersetzen
00F6DA 6100007E              bsr     $F75A        Auflösung merken
00F6DE 301F                  move.w  (A7)+,D0     Parameter für Bildschirmauflösung setzen
00F6E0 41F900017EAA          lea     $17EAA,A0    Auflösung zurückholen
00F6E6 B07C0002              cmp.w   #2,D0        Adresse des 8*8 System Font
00F6EA 6606                  bne     $F6F2        high resolution ?
00F6EC 41F900018906          lea     $18906,A0    nein
00F6F2 61009D14              bsr     $9408        Adresse des 8*16 System Font
00F6F6 33FCFFFF00002558      move.w  #$FFFF,$2558 Fontdaten initialisieren
00F6FE 7000                  moveq.l #0,D0        Schriftfarbe auf schwarz

00F700 33C000002556          move.w  D0,$2556     Hintergrundfarbe weiß
00F706 33C000002560          move.w  D0,$2560     Cursorspalte Null
00F70C 33C000002562          move.w  D0,$2562     Cursorzeile Null
00F712 33C00000255E          move.w  D0,$255E

00F718 20790000044E          move.l  $44E,A0      Adresse des Video-RAMs
00F71E 23C80000255A          move.l  A0,$255A     als Bildschirmadresse des Cursors
00F724 13FC000100002576      move.b  #1,$2576     Cursorflag setzen
00F72C 13FC001E00002565      move.b  #$1E,$2565   Cursor Blinkzähler auf 30
00F734 13FC001E00002564      move.b  #$1E,$2564   Cursor Blinkrate auf 30
00F73C 33FC000100002422      move.w  #1,$2422     Flag für Cursor ein setzen
00F744 323C1F3F              move.w  #$1F3F,D1    8000 Langworte
00F748 20C0                  move.l  D0,(A0)+     Bildschirm löschen
00F74A 51C9FFFC              dbra    D1,$F748
00F74E 23FC00008AEE000004A8 move.l  #$8AEE,$4A8  conout-Vektor auf Standard
00F758 4E75                  rts

************************************************
00F75A 7200                  moveq.l #0,D1        Bildschirmausgabe initialisieren
00F75C 123B0030              move.b  $F78E(PC,D0.w),D1 D0 enthält Bildschirmauflösung
00F760 33C10000257C          move.w  D1,$257C     Anzahl der Bildebenen holen
00F766 123B0029              move.b  $F791(PC,D0.w),D1 und merken
                                                  Bytes pro Bildschirmzeile holen
```

```
00F76A 33C10000257E          move.w  D1,$257E              und merken
00F770 33C10000257A          move.w  D1,$257A
00F776 E340                  asl.w   #1,D0
00F778 323B001A              move.w  $F794(PC,D0.w),D1     Bildschirmhöhe holen
00F77C 33C100002578          move.w  D1,$2578              und merken
00F782 323B0016              move.w  $F79A(PC,D0.w),D1     Bildschirmbreite holen
00F786 33C100002570          move.w  D1,$2570              und merken
00F78C 4E75                  rts

************************************************           Bildschirmparameter
00F78E 040201                dc.b    4,2,1                 Anzahl der Bildebenen
00F791 A0A050                dc.b    160,160,80            Bytes pro Bildschirmzeile
00F794 00C800C80190          dc.w    200,200,400           Bildschirmhöhen
00F79A 014002800280          dc.w    320,640,640           Bildschirmbreiten
```

Die Systemfonts des ATARI ST

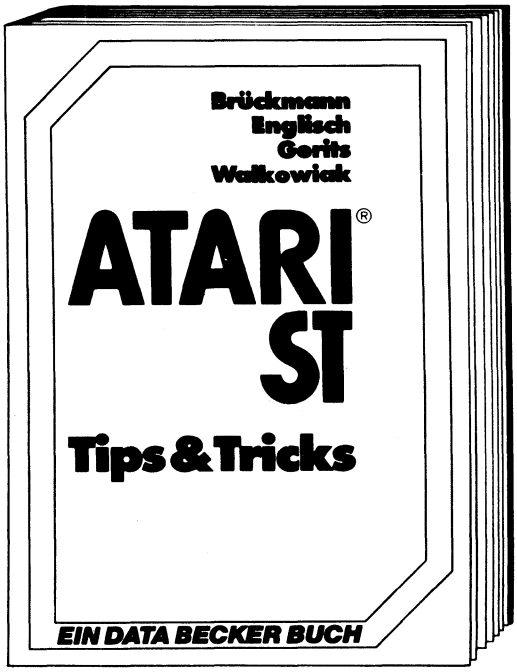
Das Betriebssystem enthält bereits drei verschiedene Fonts zur Zeichenausgabe.

Der 6*6 Font dient zur Beschriftung von Icons, der 8*8-Font wird als Standardfont bei angeschlossenem Farbmonitor benutzt, während der 8*16-Font bei der Monochromausgabe benutzt wird. Die Abbildungen auf der nächsten Seite enthalten die Zeichen mit den ASCII-Kodes 1 bis 255.

6*6 Systemfont

8*8 Systemfont

8*16 Systemfont



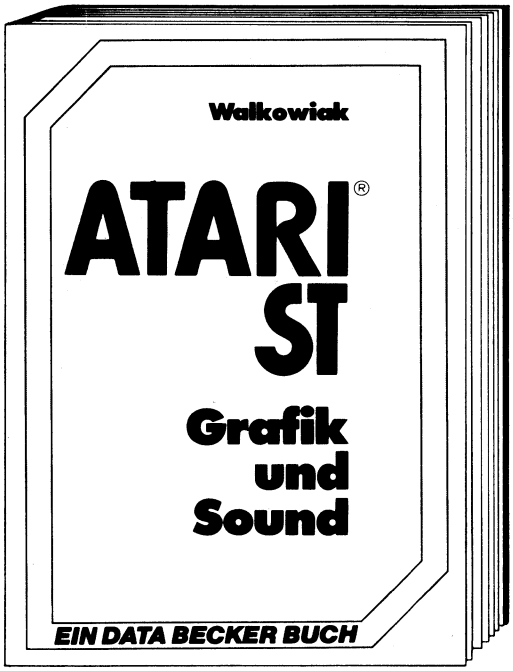
Eine riesige Fundgrube faszinierender Tips & Tricks um Ihren ATARI ST voll auszunutzen. Von phantastischen Grafiken über raffinierte Programme in BASIC, Assembler und C bis hin zu fortgeschrittenen Anwendungsmöglichkeiten.

ATARI ST Tips & Tricks, über 250 Seiten, DM 49,–

Erscheint ca. November 85

LOGO ist keineswegs nur eine Sprache für Kinder, sondern eröffnet viele interessante Bereiche wie z. B.: Rechnen mit Logo, Grafikprogrammierung, Wörter- und Listenverarbeitung, Prozeduren, Rekursionen, Sortierroutinen, Maskengenerator, Datenstrukturen und Künstliche Intelligenz. Mit LOGO können Sie schwierige und komplexe Probleme oft leichter lösen, als mit anderen Sprachen!
Das LOGO-Trainingsbuch zum ATARI ST, ca. 250 Seiten, DM 49,–

Erscheint ca. November 85



Grafik und Sound auf dem ATARI ST. Ein Traum wird wahr! Grafikgrundlagen, Animationsgrafik, Funktionsdiagramme, 2D/3D-Grafik, CAD, Soundgrundlagen und das MIDI-Interface sind nur einige Schwerpunkte des Buches. Werden Sie zum Bildschirmkünstler und Coumputerdirigenten.
ATARI ST Grafik & Sound, ca. 300 Seiten, DM 49,–

Erscheint ca. November 85

Die Vorteile des 68000-Prozessors zeigt dieses ausführliche Nachschlagewerk. Detailliertes Sachwissen: Entwicklung des 68000, Aufbau, Signal- und Busbeschreibung, Peripheriebausteine, Befehlssatz, Programmierbeispiele, Vergleich mit anderen 16-Bit-Prozessoren, weitere Prozessoren der Familie und vieles mehr.
Das Prozessorbuch zum 68000,
ca. 510 Seiten, DM 59,–

Nutzen Sie Ihren ATARI ST voll: Zahlensysteme, Bitmanipulation, der 68000 im ATARI ST, Registerverwendung, Struktur des Befehlssatzes, Programmstrukturen: Rekursion, Stacks, Prozeduren und Funktionen, Grundlagen der Assemblerprogrammierung, Systemroutinen. Ein Spitzenbuch.
ATARI ST Maschinensprache, über 200 Seiten, DM 39,–

Ein Buch für den Programmierer, der GEM effektiv nutzen will: Arbeiten mit der Maus, Icons, Virtual Device Interface, Application Environment Service, Graphics Device Operating System. Schwerpunkt: Einbinden von GEM-Routinen in BASIC, C und 68000-Assembler. Ein Standardwerk zum Betriebssystem der Zukunft.
Das große GEM-Buch zum ATARI ST,
ca. 350 Seiten, DM 49,–

DAS STEHT DRIN:

Dieser INTERN- Band wird sicher schnell zu einem Standardwerk werden. Sie finden alles Wichtige zur Hardware und Programmierung Ihres ATARI ST. Für den professionellen Einsatz unentbehrlich.

Aus dem Inhalt:

- 68000 - Prozessor
- Customer - Chips
- Disk - Controller WD 1772
- MFP 68901
- ACIA' s 6850
- Soundgenerator YM - 2149
- Centronics - Anschluß
- RS - 232
- MIDI - Anschlüsse
- DMA - Schnittstelle
- GEMDOS
- BIOS & XBIOS
- Interruptstruktur
- Befehlssatz
- BIOS - Listing

UND GESCHRIEBEN HAT DIESES BUCH:

Das bewährte Team der DATA BECKER Entwicklungsabteilung mit Klaus Gerits, Lothar Englisch und Rolf Brückmann, die als Fachbuchautoren vieler Bestseller sich in dieser nicht leichten Materie bestens auskennen.

ISBN 3-89011-119-X